1 Code source Desktop

Listing 1 – Code source de WindowChart.cs

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste représentant une ville.
 */
using LiveCharts;
using LiveCharts.Defaults;
using LiveCharts.Wpf;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Reflection;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media;

namespace CovidPropagation
{
    public delegate void SaveEventHandler(object source, ChartData e);

    /// <summary>
    /// Logique d'interaction pour WindowGraph.xaml
    /// </summary>
    public partial class WindowChart : Window
    {
        public event SaveEventHandler OnSave;
        private const int MAX_NUMBER_OF_CURVES = 5;
        int cellX;
        int cellY;
        int sizeX;
        int sizeY;
        ComboBox[] cbxDatas;
        ChartData graphicDatas;
        int currentCurvesIndex;
        object chart;

        public WindowChart(int cellX, int cellY, int sizeX, int sizeY,
            ChartData graphicDatas)
        {
            InitializeComponent();
            this.cellX = cellX;
            this.cellY = cellY;
            this.sizeX = sizeX;
            this.sizeY = sizeY;
            this.graphicDatas = graphicDatas;
            cbxDatas = new ComboBox[MAX_NUMBER_OF_CURVES];
```

```
// Créé et positionne les combobox de valeurs pour leur
   futur utilisation.
int curvesRow = Grid.GetRow(cbxQuantityOfCurves);
int curvesColumn = Grid.GetColumn(cbxQuantityOfCurves);
for (int i = 1; i <= MAX_NUMBER_OF_CURVES; i++)
{
    cbxQuantityOfCurves.Items.Add(i);
    cbxDatas[i - 1] = new ComboBox();

    curvesRow++;
    Grid.SetRow(cbxDatas[i - 1], curvesRow);
    Grid.SetColumn(cbxDatas[i - 1], curvesColumn);

    cbxDatas[i - 1].ItemsSource = from ChartsDisplayData n
                                  in
                                      Enum.GetValues(typeof(ChartsDispla
                                  select
                                      GetEnumDescription(n);

    cbxDatas[i - 1].SelectedIndex = 0;
    cbxDatas[i - 1].Margin = new Thickness(0, 2, 0, 2);
    grdContent.Children.Add(cbxDatas[i - 1]);
}

// Sélectionne les description des enums et les insères
   dans les combobox.
cbxValueX.ItemsSource = from ChartsAxisData n
                        in
                            Enum.GetValues(typeof(ChartsAxisData))
                        select GetEnumDescription(n);

cbxValueY.ItemsSource = from ChartsAxisData n
                        in
                            Enum.GetValues(typeof(ChartsAxisData))
                        select GetEnumDescription(n);

cbxGraphType.ItemsSource = from UIType n
                           in Enum.GetValues(typeof(UIType))
                           where n != UIType.GUI
                           select GetEnumDescription(n);

cbxValueX.SelectedIndex = this.graphicDatas.AxisX;
cbxValueY.SelectedIndex = this.graphicDatas.AxisY;
cbxGraphType.SelectedIndex = this.graphicDatas.UIType;


currentCurvesIndex = this.graphicDatas.Datas.Length - 1;
cbxQuantityOfCurves.SelectedIndex = currentCurvesIndex;

for (int i = 0; i < this.graphicDatas.Datas.Length; i++)
{
    cbxDatas[i].Visibility = Visibility.Visible;
    cbxDatas[i].SelectedIndex = this.graphicDatas.Datas[i];
}
```

```
        }

        /// <summary>
        /// Récupère la description des valeurs d'un enum pour un
            affichage plus logique.
        /// </summary>
        /// <param name="value">Valeur du enum à convertir en
            description.</param>
        /// <returns>Description du enum</returns>
        public static string GetEnumDescription(Enum value)
        {
            string result;
            FieldInfo fi = value.GetType().GetField(value.ToString());
            DescriptionAttribute[] attributes =
                fi.GetCustomAttributes(typeof(DescriptionAttribute),
                false) as DescriptionAttribute[];

            if (attributes != null && attributes.Any())
                result = attributes.First().Description;
            else
                result = value.ToString();

            return result;
        }

        /// <summary>
        /// Lorsque l'utilisateur sauvegarde les valeurs modifiées.
        /// </summary>
        private void Save_Click(object sender, RoutedEventArgs e)
        {
            if (OnSave != null)
            {
                List<int> datas = new List<int>();
                for (int i = 0; i <= cbxQuantityOfCurves.SelectedIndex;
                    i++)
                {
                    datas.Add(cbxDatas[i].SelectedIndex);
                }
                ChartData graphicDatas = new ChartData(cellX, cellY,
                    sizeX, sizeY, datas.ToArray(),
                    cbxGraphType.SelectedIndex, cbxValueX.SelectedIndex,
                    cbxValueY.SelectedIndex);
                OnSave(10, graphicDatas);
                this.Close();
            }
        }

        /// <summary>
        /// Lorsque l'utilisateur décide d'annuler ses modification du
            graphique.
        /// </summary>
        private void Cancel_Click(object sender, RoutedEventArgs e)
        {
            this.Close();
        }
```

```
        /// <summary>
        /// Lorsqu'un type de graphique est selectionné, affiche le bon
            graphique ainsi que les paramètres disponibles.
        /// </summary>
        private void GraphType_SelectionChanged(object sender,
            SelectionChangedEventArgs e)
        {
            if (cbxValueX != null)
            {
                cbxValueX.IsEnabled = true;
                cbxValueY.IsEnabled = true;
                cbxQuantityOfCurves.IsEnabled = true;
            }
            switch ((UIType)cbxGraphType.SelectedIndex)
            {
                case UIType.Linear:
                    chart = CreateCartesianGraph();
                    SetData(DisplayCurvesOnGraph);
                    break;
                case UIType.Vertical:
                    chart = CreateCartesianGraph();
                    SetData(DisplayColumnsOnGraph);
                    break;
                case UIType.Horizontal:
                    chart = CreateCartesianGraph();
                    SetData(DisplayRowsOnGraph);
                    break;
                case UIType.PieChart:
                    chart = CreatePieGraph();
                    SetData(DisplayPieSectionOnGraph);
                    if (cbxValueX != null)
                    {
                        cbxValueX.IsEnabled = false;
                        cbxValueY.IsEnabled = false;
                        cbxQuantityOfCurves.SelectedIndex = 1;
                    }
                    break;
                case UIType.HeatMap:
                    chart = CreateCartesianGraph();
                    SetData(DisplayHeatMapOnGraph);
                    if (cbxValueX != null)
                    {
                        cbxValueX.SelectedIndex = 1;
                        cbxValueY.SelectedIndex = 0;
                        cbxValueX.IsEnabled = false;
                        cbxValueY.IsEnabled = false;
                        cbxQuantityOfCurves.IsEnabled = false;
                        cbxQuantityOfCurves.SelectedIndex = 0;
                    }
                    break;
                default:
                    break;
            }
        }
```

```
        /// <summary>
        /// Lorsque le nombre de données à afficher est modifié par
           l'utilisateur.
        /// Modifie les données qui sont affichées dans le graphique
           pour en ajouter ou en retirer.
        /// </summary>
        private void DataQuantity_SelectionChanged(object sender,
           SelectionChangedEventArgs e)
        {
            currentCurvesIndex =
               cbxQuantityOfCurves.Items.IndexOf(cbxQuantityOfCurves.SelectedItem);

            switch ((UIType)cbxGraphType.SelectedIndex)
            {
                case UIType.Linear:
                    SetData(DisplayCurvesOnGraph);
                    break;
                case UIType.Vertical:
                    SetData(DisplayColumnsOnGraph);
                    break;
                case UIType.Horizontal:
                    SetData(DisplayRowsOnGraph);
                    break;
                case UIType.PieChart:
                    SetData(DisplayPieSectionOnGraph);
                    break;
                case UIType.HeatMap:
                    SetData(DisplayHeatMapOnGraph);
                    break;
                default:
                    break;
            }
        }

        /// <summary>
        /// Affiche le nombre de combobox nécessaire en fonction du
           nombre de données à afficher choisis par l'utilisateur
        /// </summary>
        /// <param name="callback">Méthode qui sera appelée pour
           l'affichage des données du graphique (Courbe, colonne,
           etc.)</param>
        private void SetData(Func<int, bool, bool> callback)
        {
            if (cbxDatas != null)
            {
                for (int i = 0; i < MAX_NUMBER_OF_CURVES; i++)
                {
                    if (i <= currentCurvesIndex)
                    {
                        cbxDatas[i].Visibility = Visibility.Visible;
                        callback(i, true);
                        cbxDatas[i].Tag = i;
                        cbxDatas[i].SelectionChanged +=
                           CbxDatas_SelectionChanged;
```

```
                }
                else
                {
                    cbxDatas[i].Visibility = Visibility.Hidden;
                    callback(i, false);
                }
            }
        }
    }

    /// <summary>
    /// SetData utilisé pour les graphiques heatmap qui ne contient
       qu'une "courbe" de valeur et qui n'a donc besoin que d'un
       unique appelle.
    /// </summary>
    /// <param name="callback">Méthode qui sera appelée pour
       l'affichage des données du graphique (Courbe, colonne,
       etc.)</param>
    private void SetData(Func<bool> callback)
    {
        if (cbxDatas != null)
        {
            callback();
        }
    }

    /// <summary>
    /// Affiche les courbes sur un graphique cartesien.
    /// Définit si une courbe doit rester affichée, être ajoutée,
       ou supprimée.
    /// </summary>
    /// <param name="index">Index de la courbe à modifier.</param>
    /// <param name="isDisplayed">Si la courbe doit être affichée
       ou non.</param>
    /// <returns>Valeur obligatoir à l'utilisation de
       callback</returns>
    private bool DisplayCurvesOnGraph(int index, bool isDisplayed)
    {
        chart ??= CreateCartesianGraph();

        CartesianChart cartesianChart = (CartesianChart)chart;

        // Si l'index doit être affiché et n'existe pas déjà
        if (isDisplayed && index >
            cartesianChart.Series.GetLastIndex())
        {
            Random rdm = GlobalVariables.rdm;
            cartesianChart.Series.Add(new LineSeries
            {
                Title = cbxDatas[index].SelectedItem.ToString(),
                Values = new ChartValues<double> {
                    rdm.Next(1, 10),
                    rdm.Next(1, 10),
                    rdm.Next(1, 10),
                    rdm.Next(1, 10),
```

```
                    rdm.Next(1, 10)
                },
                PointGeometry = null,
                DataLabels = false
            });
        }
        else if(!isDisplayed) // Sinon retire le dernier index
        {
            int lastIndex = cartesianChart.Series.GetLastIndex();
            if (lastIndex > 0 && lastIndex > currentCurvesIndex)
                cartesianChart.Series.RemoveAt(lastIndex);
        }
        return true;
    }

    /// <summary>
    /// Affiche les colonnes sur un graphique cartesien.
    /// Définit si une colonne doit rester affichée, être ajoutée,
       ou supprimée.
    /// </summary>
    /// <param name="index">Index de la colonne à modifier.</param>
    /// <param name="isDisplayed">Si la colonne doit être affichée
       ou non.</param>
    /// <returns>Valeur obligatoir à l'utilisation de
       callback</returns>
    private bool DisplayColumnsOnGraph(int index, bool isDisplayed)
    {
        if (chart == null)
            chart = CreateCartesianGraph();

        CartesianChart cartesianChart = (CartesianChart)chart;

        if (isDisplayed && index >
           cartesianChart.Series.GetLastIndex())
        {
            Random rdm = GlobalVariables.rdm;
            cartesianChart.Series.Add(new ColumnSeries
            {
                Title = cbxDatas[index].SelectedItem.ToString(),
                Values = new ChartValues<double> {
                    rdm.Next(1, 10),
                    rdm.Next(1, 10),
                    rdm.Next(1, 10),
                    rdm.Next(1, 10),
                    rdm.Next(1, 10)
                },
                PointGeometry = null,
                DataLabels = false
            });
        }
        else
        {
            int lastIndex = cartesianChart.Series.GetLastIndex();
            if (lastIndex > 0 && lastIndex > currentCurvesIndex)
                cartesianChart.Series.RemoveAt(lastIndex);
```

```
            }
            return true;
        }

        /// <summary>
        /// Affiche les lignes sur un graphique cartesien.
        /// Définit si une ligne doit rester affichée, être ajoutée, ou
            supprimée.
        /// </summary>
        /// <param name="index">Index de la ligne à modifier.</param>
        /// <param name="isDisplayed">Si la ligne doit être affichée ou
            non.</param>
        /// <returns>Valeur obligatoir à l'utilisation de
            callback</returns>
        private bool DisplayRowsOnGraph(int index, bool isDisplayed)
        {
            if (chart == null)
                chart = CreateCartesianGraph();

            CartesianChart cartesianChart = (CartesianChart)chart;

            if (isDisplayed && index >
                cartesianChart.Series.GetLastIndex())
            {
                Random rdm = GlobalVariables.rdm;
                cartesianChart.Series.Add(new RowSeries
                {
                    Title = cbxDatas[index].SelectedItem.ToString(),
                    Values = new ChartValues<double> {
                        rdm.Next(1, 10),
                        rdm.Next(1, 10),
                        rdm.Next(1, 10),
                        rdm.Next(1, 10),
                        rdm.Next(1, 10)
                    },
                    PointGeometry = null,
                    DataLabels = false
                });
            }
            else
            {
                int lastIndex = cartesianChart.Series.GetLastIndex();
                if (lastIndex > 0 && lastIndex > currentCurvesIndex)
                    cartesianChart.Series.RemoveAt(lastIndex);
            }
            return true;
        }

        /// <summary>
        /// Affiche les sections sur un graphique cylindrique.
        /// Définit si une section doit rester affichée, être ajoutée,
            ou supprimée.
        /// </summary>
        /// <param name="index">Index de la section à modifier.</param>
```

```
        /// <param name="isDisplayed">Si la section doit être affichée
            ou non.</param>
        /// <returns>Valeur obligatoir à l'utilisation de
            callback</returns>
        private bool DisplayPieSectionOnGraph(int index, bool
            isDisplayed)
        {
            if (chart == null)
                chart = CreatePieGraph();

            PieChart pieChart = (PieChart)chart;

            if (isDisplayed && index > pieChart.Series.GetLastIndex())
            {
                Random rdm = GlobalVariables.rdm;
                pieChart.Series.Add(new PieSeries
                {
                    Title = cbxDatas[index].SelectedItem.ToString(),
                    Values = new ChartValues<double> {
                        rdm.Next(1, 10)
                    },
                    PointGeometry = null,
                    DataLabels = false
                });
            }
            else
            {
                int lastIndex = pieChart.Series.GetLastIndex();
                if (lastIndex > 0 && lastIndex > currentCurvesIndex)
                    pieChart.Series.RemoveAt(lastIndex);
            }
            return true;
        }

        /// <summary>
        /// Remplit le graphique heatmap de valeur s'il n'y en a pas
            déjà.
        /// </summary>
        /// <param name="index"></param>
        /// <param name="isDisplayed"></param>
        /// <returns>Valeur obligatoir à l'utilisation de
            callback</returns>
        private bool DisplayHeatMapOnGraph()
        {
            // Index & isDisplayed useless
            if (chart == null)
                chart = CreateCartesianGraph();

            CartesianChart cartesianChart = (CartesianChart)chart;
            Random rdm = GlobalVariables.rdm;

            if (cartesianChart.Series.Count == 0)
            {
                ChartValues<HeatPoint> values = new
                    ChartValues<HeatPoint>
```

```
{
    new HeatPoint(0, 0, rdm.Next(0, 10)),
    new HeatPoint(0, 1, rdm.Next(0, 10)),
    new HeatPoint(0, 2, rdm.Next(0, 10)),
    new HeatPoint(0, 3, rdm.Next(0, 10)),
    new HeatPoint(0, 4, rdm.Next(0, 10)),
    new HeatPoint(0, 5, rdm.Next(0, 10)),
    new HeatPoint(0, 6, rdm.Next(0, 10)),

    new HeatPoint(1, 0, rdm.Next(0, 10)),
    new HeatPoint(1, 1, rdm.Next(0, 10)),
    new HeatPoint(1, 2, rdm.Next(0, 10)),
    new HeatPoint(1, 3, rdm.Next(0, 10)),
    new HeatPoint(1, 4, rdm.Next(0, 10)),
    new HeatPoint(1, 5, rdm.Next(0, 10)),
    new HeatPoint(1, 6, rdm.Next(0, 10)),

    new HeatPoint(2, 0, rdm.Next(0, 10)),
    new HeatPoint(2, 1, rdm.Next(0, 10)),
    new HeatPoint(2, 2, rdm.Next(0, 10)),
    new HeatPoint(2, 3, rdm.Next(0, 10)),
    new HeatPoint(2, 4, rdm.Next(0, 10)),
    new HeatPoint(2, 5, rdm.Next(0, 10)),
    new HeatPoint(2, 6, rdm.Next(0, 10)),

    new HeatPoint(3, 0, rdm.Next(0, 10)),
    new HeatPoint(3, 1, rdm.Next(0, 10)),
    new HeatPoint(3, 2, rdm.Next(0, 10)),
    new HeatPoint(3, 3, rdm.Next(0, 10)),
    new HeatPoint(3, 4, rdm.Next(0, 10)),
    new HeatPoint(3, 5, rdm.Next(0, 10)),
    new HeatPoint(3, 6, rdm.Next(0, 10)),

    new HeatPoint(4, 0, rdm.Next(0, 10)),
    new HeatPoint(4, 1, rdm.Next(0, 10)),
    new HeatPoint(4, 2, rdm.Next(0, 10)),
    new HeatPoint(4, 3, rdm.Next(0, 10)),
    new HeatPoint(4, 4, rdm.Next(0, 10)),
    new HeatPoint(4, 5, rdm.Next(0, 10)),
    new HeatPoint(4, 6, rdm.Next(0, 10))
};

HeatSeries heatSeries = new HeatSeries();
heatSeries.Values = values;
heatSeries.Title = cbxDatas[0].SelectedItem.ToString();
heatSeries.GradientStopCollection = new
   GradientStopCollection()
{
    new GradientStop(Colors.Green, 0),
    new GradientStop(Colors.GreenYellow, 0.25),
    new GradientStop(Colors.Yellow, 0.5),
    new GradientStop(Colors.Orange, 0.75),
    new GradientStop(Colors.Red, 1)
};
heatSeries.DataLabels = true;
```

```
            cartesianChart.Series.Add(heatSeries);
        }
        DataContext = this;
        return true;
    }

    /// <summary>
    /// Lorsque la valeur choisie par l'utilisateur pour une courbe
       est modifié.
    /// Change la valeur le nom de la courbe ainsi que ses valeurs.
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void CbxDatas_SelectionChanged(object sender,
       SelectionChangedEventArgs e)
    {
        ComboBox cbxData = (ComboBox)sender;
        switch ((UIType)cbxGraphType.SelectedIndex)
        {
            case UIType.Linear:
                CartesianChart linearChart = (CartesianChart)chart;
                linearChart.Series[Convert.ToInt32(cbxData.Tag)] =
                    new LineSeries
                {
                    Title = cbxData.SelectedItem.ToString(),
                    Values =
                        linearChart.Series[Convert.ToInt32(cbxData.Tag)].Values
                };
                break;
            case UIType.Vertical:
                CartesianChart verticalChart =
                    (CartesianChart)chart;
                verticalChart.Series[Convert.ToInt32(cbxData.Tag)]
                    = new ColumnSeries
                {
                    Title = cbxData.SelectedItem.ToString(),
                    Values =
                        verticalChart.Series[Convert.ToInt32(cbxData.Tag)].Values
                };
                break;
            case UIType.Horizontal:
                CartesianChart horizontalChart =
                    (CartesianChart)chart;
                horizontalChart.Series[Convert.ToInt32(cbxData.Tag)]
                    = new RowSeries
                {
                    Title = cbxData.SelectedItem.ToString(),
                    Values =
                        horizontalChart.Series[Convert.ToInt32(cbxData.Tag)].Valu
                };
                break;
            case UIType.PieChart:
                PieChart pieChart = (PieChart)chart;
                pieChart.Series[Convert.ToInt32(cbxData.Tag)] = new
                    PieSeries
```

```
                {
                    Title = cbxData.SelectedItem.ToString(),
                    Values =
                        pieChart.Series[Convert.ToInt32(cbxData.Tag)].Values
                };
                break;
            case UIType.HeatMap:
                chart = CreateCartesianGraph();
                SetData(DisplayHeatMapOnGraph);
                break;
            default:
                break;
        }
    }

    /// <summary>
    /// Créé un graphique cartesien avec des valeurs par défaut.
    /// </summary>
    /// <returns>Graphique créé.</returns>
    private CartesianChart CreateCartesianGraph()
    {
        CartesianChart cartesianChart = new CartesianChart();
        cartesianChart.Series = new SeriesCollection();
        cartesianChart.LegendLocation = LegendLocation.Right;
        cartesianChart.Foreground = Brushes.Gray;
        cartesianChart.DisableAnimations = true;
        cartesianChart.Hoverable = false;

        Axis axisX = new Axis();
        axisX.Foreground = Brushes.Gray;
        axisX.MaxValue = double.NaN;
        axisX.Title =
            GetEnumDescription((ChartsAxisData)graphicDatas.AxisX);
        cartesianChart.AxisX.Add(axisX);

        Axis axisY = new Axis();
        axisY.Foreground = Brushes.Gray;
        axisY.MaxValue = double.NaN;
        axisY.Title =
            GetEnumDescription((ChartsAxisData)graphicDatas.AxisY);
        cartesianChart.AxisY.Add(axisY);

        DataContext = this;
        cartesianChart.VerticalAlignment =
            VerticalAlignment.Stretch;
        cartesianChart.HorizontalAlignment =
            HorizontalAlignment.Stretch;
        ugrGraph.Children.Clear();
        ugrGraph.Children.Add(cartesianChart);

        return cartesianChart;
    }

    /// <summary>
    /// Créé un graphique cylindrique.
```

```
        /// </summary>
        /// <returns>Le graphique cylindrique créé.</returns>
        private PieChart CreatePieGraph()
        {
            PieChart pieChart = new PieChart();
            pieChart.Series = new SeriesCollection();
            pieChart.LegendLocation = LegendLocation.Right;
            pieChart.DisableAnimations = true;
            pieChart.Hoverable = false;

            DataContext = this;
            pieChart.VerticalAlignment = VerticalAlignment.Stretch;
            pieChart.HorizontalAlignment = HorizontalAlignment.Stretch;

            ugrGraph.Children.Clear();
            ugrGraph.Children.Add(pieChart);

            return pieChart;
        }

        /// <summary>
        /// Lorsque la valeur d0un des axes est modifée.
        /// Change le nom de l'axe ainsi que ses valeurs.
        /// </summary>
        private void AxisValue_SelectionChanged(object sender,
            SelectionChangedEventArgs e)
        {
            ComboBox cbxAxis = (ComboBox)sender;
            ChartsAxisData axisData =
                (ChartsAxisData)cbxAxis.SelectedIndex;
            CartesianChart cartesianChart = (CartesianChart)chart;
            Axis axis;

            if ((string)cbxAxis.Tag == "X")
                axis = cartesianChart.AxisX[0];
            else
                axis = cartesianChart.AxisY[0];

            if (axisData >= 0)
            {
                if ((string)cbxAxis.Tag == "X")
                    axis.Title = GetEnumDescription(axisData);
                else
                    axis.Title = GetEnumDescription(axisData);
            }
        }
    }
}
```

Listing 2 – Code source de PageSimulationSettings.cs

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
```

```
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste représentant une ville.
 */
using System;
using System.Text.RegularExpressions;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media;

namespace CovidPropagation
{
    /// <summary>
    /// Logique d'interaction pour PageSimulationSettings.xaml
    /// </summary>
    public partial class PageSimulationSettings : Page
    {
        public PageSimulationSettings()
        {
            InitializeComponent();
            SetUIParameters();
            SetParametersData();
        }

        /// <summary>
        /// Prend les valeurs des paramètres et active/désactive les
           champs nécessaires.
        /// </summary>
        private void SetUIParameters()
        {
            // Ajoute un évènement qui est trigger lorsque ctrl + v est
               pressé dans le textbox
            DataObject.AddPastingHandler(tbxCoughMaxQuanta,
               this.OnCancelCommand);
            DataObject.AddPastingHandler(tbxCoughMinQuanta,
               this.OnCancelCommand);
            DataObject.AddPastingHandler(tbxNbPeople,
               this.OnCancelCommand);
            DataObject.AddPastingHandler(tbxProbabilityOfInfected,
               this.OnCancelCommand);
            DataObject.AddPastingHandler(tbxVaccinationDuration,
               this.OnCancelCommand);
            DataObject.AddPastingHandler(tbxVaccinationEfficiency,
               this.OnCancelCommand);
            DataObject.AddPastingHandler(tbxVirusImmunityEfficiency,
               this.OnCancelCommand);
            DataObject.AddPastingHandler(tbxVirusMaxDuration,
               this.OnCancelCommand);
            DataObject.AddPastingHandler(tbxVirusMaxImmunityDuration,
               this.OnCancelCommand);
            DataObject.AddPastingHandler(tbxVirusMaxIncubationDuration,
               this.OnCancelCommand);
            DataObject.AddPastingHandler(tbxVirusMinDuration,
               this.OnCancelCommand);
            DataObject.AddPastingHandler(tbxVirusMinImmunityDuration,
```

```
            this.OnCancelCommand);
        DataObject.AddPastingHandler(tbxVirusMinIncubationDuration,
            this.OnCancelCommand);

        // Distanciation
        chxDistanciation.IsChecked =
            SimulationGeneralParameters.IsDistanciationMeasuresEnabled;
        tbxDistanciationNbPeopleToStart.IsEnabled =
            SimulationGeneralParameters.IsDistanciationMeasuresEnabled;
        tbxDistanciationNbPeopleToStop.IsEnabled =
            SimulationGeneralParameters.IsDistanciationMeasuresEnabled;

        tbxProbabilityForHealthyToBeQuarantined.IsEnabled =
            QuarantineParameters.IshealthyQuarantined;
        tbxQuarantinedDurationForHealthy.IsEnabled =
            QuarantineParameters.IshealthyQuarantined;

        tbxProbabilityForInfectedToBeQuarantined.IsEnabled =
            QuarantineParameters.IsInfectedQuarantined;
        tbxQuarantinedDurationForInfected.IsEnabled =
            QuarantineParameters.IsInfectedQuarantined;

        tbxProbabilityForInfectiousToBeQuarantined.IsEnabled =
            QuarantineParameters.IsInfectiousQuarantined;
        tbxQuarantinedDurationForInfectious.IsEnabled =
            QuarantineParameters.IsInfectiousQuarantined;

        tbxProbabilityForImmuneToBeQuarantined.IsEnabled =
            QuarantineParameters.IsImmuneQuarantined;
        tbxQuarantinedDurationForImmune.IsEnabled =
            QuarantineParameters.IsImmuneQuarantined;

        // Masques
        chxMask.IsChecked =
            SimulationGeneralParameters.IsMaskMeasuresEnabled;
        tbxMaskNbPeopleToStart.IsEnabled =
            SimulationGeneralParameters.IsMaskMeasuresEnabled;
        tbxMaskNbPeopleToStop.IsEnabled =
            SimulationGeneralParameters.IsMaskMeasuresEnabled;
        if (!SimulationGeneralParameters.IsMaskMeasuresEnabled)
        {
             splClientMask.IsEnabled = false;
             splPersonnelMask.IsEnabled = false;

             rdbClientMaskIsOn.IsChecked =
                 MaskParameters.IsClientMaskOn;
             rdbClientMaskIsOff.IsChecked =
                 !MaskParameters.IsClientMaskOn;
             rdbPersonnelMaskIsOn.IsChecked =
                 MaskParameters.IsPersonnelMaskOn;
             rdbPersonnelMaskIsOff.IsChecked =
                 !MaskParameters.IsPersonnelMaskOn;
        }

        // Quarantaine
```

```
            chxQuarantaine.IsChecked =
                SimulationGeneralParameters.IsQuarantineMeasuresEnabled;
            tbxQuarantineNbPeopleToStart.IsEnabled =
                SimulationGeneralParameters.IsQuarantineMeasuresEnabled;
            tbxQuarantineNbPeopleToStop.IsEnabled =
                SimulationGeneralParameters.IsQuarantineMeasuresEnabled;
            pnlCbxQuarantine.IsEnabled =
                SimulationGeneralParameters.IsQuarantineMeasuresEnabled;
            if
                (!SimulationGeneralParameters.IsQuarantineMeasuresEnabled)
            {
                chxQuarantineHealthy.IsChecked = false;
                chxQuarantineImmune.IsChecked = false;
                chxQuarantineInfected.IsChecked = false;
                chxQuarantineInfectious.IsChecked = false;
            }

            // Vaccination
            chxVaccination.IsChecked =
                SimulationGeneralParameters.IsVaccinationMeasuresEnabled;
            tbxVaccinationDuration.IsEnabled =
                SimulationGeneralParameters.IsVaccinationMeasuresEnabled;
            tbxVaccinationEfficiency.IsEnabled =
                SimulationGeneralParameters.IsVaccinationMeasuresEnabled;
            tbxVaccinationNbPeopleToStart.IsEnabled =
                SimulationGeneralParameters.IsVaccinationMeasuresEnabled;
            tbxVaccinationNbPeopleToStop.IsEnabled =
                SimulationGeneralParameters.IsVaccinationMeasuresEnabled;
        }

        /// <summary>
        /// Prend les valeurs des paramètres et les affiches.
        /// </summary>
        private void SetParametersData()
        {
            // Général
            tbxNbPeople.Text =
                SimulationGeneralParameters.NbPeople.ToString();
            tbxProbabilityOfInfected.Text =
                (SimulationGeneralParameters.ProbabilityOfInfected *
                100).ToString();

            // Triggers mesures
            tbxQuarantineNbPeopleToStart.Text =
                SimulationGeneralParameters.NbInfecetdForQuarantineActivation.ToStrin
            tbxQuarantineNbPeopleToStop.Text =
                SimulationGeneralParameters.NbInfecetdForQuarantineDeactivation.ToSt

            tbxVaccinationNbPeopleToStart.Text =
                SimulationGeneralParameters.NbInfecetdForVaccinationActivation.ToStr
            tbxVaccinationNbPeopleToStop.Text =
                SimulationGeneralParameters.NbInfecetdForVaccinationDeactivation.ToS

            tbxMaskNbPeopleToStart.Text =
                SimulationGeneralParameters.NbInfecetdForMaskActivation.ToString();
```

```
            tbxMaskNbPeopleToStop.Text =
                SimulationGeneralParameters.NbInfecetdForMaskDeactivation.ToString()

            tbxDistanciationNbPeopleToStart.Text =
                SimulationGeneralParameters.NbInfecetdForDistanciationActivation.ToSt
            tbxDistanciationNbPeopleToStop.Text =
                SimulationGeneralParameters.NbInfecetdForDistanciationDeactivation.To

            // Quarantaine
            tbxProbabilityForHealthyToBeQuarantined.Text =
                (QuarantineParameters.ProbabilityOfHealthyQuarantined *
                100).ToString();
            tbxQuarantinedDurationForHealthy.Text =
                QuarantineParameters.DurationHealthyQuarantined.ToString();

            tbxProbabilityForInfectedToBeQuarantined.Text =
                (QuarantineParameters.ProbabilityOfInfectedQuarantined *
                100).ToString();
            tbxQuarantinedDurationForInfected.Text =
                QuarantineParameters.DurationInfectedQuarantined.ToString();

            tbxProbabilityForInfectiousToBeQuarantined.Text =
                (QuarantineParameters.ProbabilityOfHealthyQuarantined *
                100).ToString();
            tbxQuarantinedDurationForInfectious.Text =
                QuarantineParameters.DurationInfectiousQuarantined.ToString();

            tbxProbabilityForImmuneToBeQuarantined.Text =
                (QuarantineParameters.ProbabilityOfImmuneQuarantined *
                100).ToString();
            tbxQuarantinedDurationForImmune.Text =
                QuarantineParameters.DurationImmuneQuarantined.ToString();

            // Vaccin
            tbxVaccinationDuration.Text =
                VaccinationParameters.Duration.ToString();
            tbxVaccinationEfficiency.Text =
                (VaccinationParameters.Efficiency * 100).ToString();

            // Virus
            tbxVirusMinDuration.Text =
                VirusParameters.DurationMin.ToString();
            tbxVirusMaxDuration.Text =
                VirusParameters.DurationMax.ToString();

            tbxVirusMinIncubationDuration.Text =
                VirusParameters.IncubationDurationMin.ToString();
            tbxVirusMaxIncubationDuration.Text =
                VirusParameters.IncubationDurationMax.ToString();

            tbxVirusMinImmunityDuration.Text =
                VirusParameters.ImmunityDurationMin.ToString();
            tbxVirusMaxImmunityDuration.Text =
                VirusParameters.ImmunityDurationMax.ToString();
```

```
            tbxVirusImmunityEfficiency.Text =
                (VirusParameters.ImmunityEfficiency * 100).ToString();
                // Transformation de probabilités en pourcentage

            // Quanta
            tbxCoughMinQuanta.Text =
                VirusParameters.CoughMinQuanta.ToString();
            tbxCoughMaxQuanta.Text =
                VirusParameters.CoughMaxQuanta.ToString();
        }

        /// <summary>
        /// Regex qui filtres les caractères non numériques allant de
            1'000 à 99'999'999.
        /// </summary>
        private void NbPersons_LeaveFocus(object sender,
            RoutedEventArgs e)
        {
            TextBox tbx = sender as TextBox;
            Regex regex = new Regex("^([1-9][0-9]{3,7})$");
            string defaultValue = "10000";
            CheckTextBoxFormat(tbx, regex, defaultValue);
        }

        /// <summary>
        /// Regex qui filtres les caractères non numériques allant de 0
            à 99'999'999.
        /// </summary>
        private void NbPersonsMeasures_LeaveFocus(object sender,
            RoutedEventArgs e)
        {
            TextBox tbx = sender as TextBox;
            Regex regex = new Regex("^([0-9]{0,8})$");
            string defaultValue = "1000";
            CheckTextBoxFormat(tbx, regex, defaultValue);
        }

        /// <summary>
        /// Regex qui filtres les caractères non numériques allant de 0
            à 1'000'000.
        /// </summary>
        private void DayDuration_LeaveFocus(object sender,
            RoutedEventArgs e)
        {
            TextBox tbx = sender as TextBox;
            Regex regex = new Regex("^([0-9][0-9]{0,5})$");
            string defaultValue = "7";
            CheckTextBoxFormat(tbx, regex, defaultValue);
        }

        /// <summary>
        /// Regex qui filtres les caractères non numériques allant de 0
            à 1'000'000.
        /// </summary>
        private void MonthDuration_LeaveFocus(object sender,
```

```
        RoutedEventArgs e)
    {
        TextBox tbx = sender as TextBox;
        Regex regex = new Regex("^([0-9][0-9]{0,5})$");
        string defaultValue = "6";
        CheckTextBoxFormat(tbx, regex, defaultValue);
    }

    /// <summary>
    /// Regex qui filtres les caractères non numériques allant de 0
        à 800.
    /// </summary>
    private void Quantas_LeaveFocus(object sender, RoutedEventArgs
        e)
    {
        TextBox tbx = sender as TextBox;
        Regex regex = new
            Regex("^([1-7][0-9]{0,2}|800|[0-9]{1,2})$");
        string defaultValue = "200";
        CheckTextBoxFormat(tbx, regex, defaultValue);
    }

    /// <summary>
    /// Regex qui filtres les caractères non numériques allant de
        0% à 100% en prenant en compte les décimals.
    /// </summary>
    private void Probability_LeaveFocus(object sender,
        RoutedEventArgs e)
    {
        TextBox tbx = sender as TextBox;
        Regex regex = new
            Regex(@"\b(?<!\.)(?!0+(?:\.0+)?%)(?:\d|[1-9]\d|100)(?:(?<!100)\.\d+)
        string defaultValue = "0.00";
        CheckTextBoxFormat(tbx, regex, defaultValue);
    }

    /// <summary>
    /// Vérifie que le format du textBox entre en accord avec le
        regex attribué.
    /// </summary>
    /// <param name="tbx">Textbox modifié.</param>
    /// <param name="regex">Règle des valeurs du textbox.</param>
    /// <param name="defaultValue">Valeur placée dans le textbox si
        l'utilisateur rentre un format incorrect.</param>
    private void CheckTextBoxFormat(TextBox tbx, Regex regex,
        string defaultValue)
    {
        if (!regex.IsMatch(tbx.Text))
        {
            tbx.Background =
                this.FindResource("highlightWrongFormat") as Brush;
            tbx.Text = defaultValue;
        }
        else
        {
```

```
            tbx.Background =
                this.FindResource("highlightCorrectFormat") as Brush;
        }
    }

    /// <summary>
    /// S'active lors d'un ctrl+v. Permet d'éviter de coller des
        caratères qui ne sont pas du texte.
    /// </summary>
    private void OnCancelCommand(object sender, DataObjectEventArgs
        e)
    {
        e.CancelCommand();
    }

    /// <summary>
    /// Active ou désactive les champs lié à un checkbox.
    /// </summary>
    /// <param name="sender">Checkbox qui a trigger
        l'évènement.</param>
    private void Measure_Checked(object sender, RoutedEventArgs e)
    {
        CheckBox chx = sender as CheckBox;

        switch (chx.Tag)
        {
            default:
            case "Mask":
                tbxMaskNbPeopleToStart.IsEnabled = (chx.IsChecked
                    == true);
                tbxMaskNbPeopleToStop.IsEnabled = (chx.IsChecked ==
                    true);
                splClientMask.IsEnabled = (chx.IsChecked == true);
                splPersonnelMask.IsEnabled = (chx.IsChecked ==
                    true);
                break;
            case "Distancing":
                tbxDistanciationNbPeopleToStart.IsEnabled =
                    (chx.IsChecked == true);
                tbxDistanciationNbPeopleToStop.IsEnabled =
                    (chx.IsChecked == true);
                break;
            case "Quarantine":
                pnlCbxQuarantine.IsEnabled = (chx.IsChecked ==
                    true);
                tbxQuarantineNbPeopleToStart.IsEnabled =
                    (chx.IsChecked == true);
                tbxQuarantineNbPeopleToStop.IsEnabled =
                    (chx.IsChecked == true);
                if (chx.IsChecked == false)
                {
                    chxQuarantineHealthy.IsChecked = false;
                    chxQuarantineImmune.IsChecked = false;
                    chxQuarantineInfected.IsChecked = false;
                    chxQuarantineInfectious.IsChecked = false;
```

```
                }
                break;
            case "Vaccination":
                tbxVaccinationDuration.IsEnabled = (chx.IsChecked
                    == true);
                tbxVaccinationEfficiency.IsEnabled = (chx.IsChecked
                    == true);
                tbxVaccinationNbPeopleToStart.IsEnabled =
                    (chx.IsChecked == true);
                tbxVaccinationNbPeopleToStop.IsEnabled =
                    (chx.IsChecked == true);
                break;
        }
    }

    /// <summary>
    /// Active ou désactive les champs lié au checkbox de la
        quarantaine.
    /// </summary>
    /// <param name="sender">CheckBox qui a été coché.</param>
    private void QuarantineTypeChecked_Checked(object sender,
        RoutedEventArgs e)
    {
        CheckBox chx = sender as CheckBox;
        switch (chx.Tag.ToString())
        {
            default:
            case "Healthy":
                tbxProbabilityForHealthyToBeQuarantined.IsEnabled =
                    chxQuarantineHealthy.IsChecked == true;
                tbxQuarantinedDurationForHealthy.IsEnabled =
                    chxQuarantineHealthy.IsChecked == true;
                break;
            case "Infected":
                tbxProbabilityForInfectedToBeQuarantined.IsEnabled
                    = chxQuarantineInfected.IsChecked == true;
                tbxQuarantinedDurationForInfected.IsEnabled =
                    chxQuarantineInfected.IsChecked == true;
                break;
            case "Infectious":
                tbxProbabilityForInfectiousToBeQuarantined.IsEnabled
                    = chxQuarantineInfectious.IsChecked == true;
                tbxQuarantinedDurationForInfectious.IsEnabled =
                    chxQuarantineInfectious.IsChecked == true;
                break;
            case "Immune":
                tbxProbabilityForImmuneToBeQuarantined.IsEnabled =
                    chxQuarantineImmune.IsChecked == true;
                tbxQuarantinedDurationForImmune.IsEnabled =
                    chxQuarantineImmune.IsChecked == true;
                break;
        }
    }

    /// <summary>
```

```
        /// Sauvegarde les paramètres que l'utilisateur a entré.
        /// </summary>
        private void Save_Click(object sender, RoutedEventArgs e)
        {
            SetVirusParameters();
            SetGeneralParameters();
            SetMaskMeasureParameters();
            SetQuarantineMeasureParameters();
            SetVaccinationMeasureParameters();
        }

        /// <summary>
        /// Récupère les paramètres du virus et les affiches.
        /// </summary>
        private void SetVirusParameters()
        {
            VirusParameters.Init();

            if (tbxVirusMinIncubationDuration.Text.Length > 0)
                VirusParameters.IncubationDurationMin =
                    Convert.ToInt32(tbxVirusMinIncubationDuration.Text);

            if (tbxVirusMaxIncubationDuration.Text.Length > 0)
                VirusParameters.IncubationDurationMax =
                    Convert.ToInt32(tbxVirusMaxIncubationDuration.Text);

            if (tbxVirusMaxDuration.Text.Length > 0)
                VirusParameters.DurationMax =
                    Convert.ToInt32(tbxVirusMaxDuration.Text);

            if (tbxVirusMinDuration.Text.Length > 0)
                VirusParameters.DurationMin =
                    Convert.ToInt32(tbxVirusMinDuration.Text);

            if (tbxVirusMinImmunityDuration.Text.Length > 0)
                VirusParameters.ImmunityDurationMin =
                    Convert.ToInt32(tbxVirusMinImmunityDuration.Text);

            if (tbxVirusMaxImmunityDuration.Text.Length > 0)
                VirusParameters.ImmunityDurationMax =
                    Convert.ToInt32(tbxVirusMaxImmunityDuration.Text);

            if (tbxVirusImmunityEfficiency.Text.Length > 0)
                VirusParameters.ImmunityEfficiency =
                    Convert.ToDouble(tbxVirusImmunityEfficiency.Text) /
                    100; // Transformation de pourcentage en probabilités

            // Symptoms
            if (tbxCoughMinQuanta.Text.Length > 0)
                VirusParameters.CoughMinQuanta =
                    Convert.ToInt32(tbxCoughMinQuanta.Text);

            if (tbxCoughMaxQuanta.Text.Length > 0)
                VirusParameters.CoughMaxQuanta =
                    Convert.ToInt32(tbxCoughMaxQuanta.Text);
```

```
            if (rdbCoughSymptomEnabled.IsChecked == true)
                VirusParameters.IsCoughSymptomActive = true;
            else if (rdbCoughSymptomDisabled.IsChecked == true)
                VirusParameters.IsCoughSymptomActive = false;

            // Transmission
            if (rdbAerosolTransmissionIsEnabled.IsChecked == true)
                VirusParameters.IsAerosolTransmissionActive = true;
            else if (rdbAerosolTransmissionIsDisabled.IsChecked == true)
                VirusParameters.IsCoughSymptomActive = false;
        }

        /// <summary>
        /// Récupère les paramètres généraux et les affiches.
        /// </summary>
        private void SetGeneralParameters()
        {
            // Nombre d'individu, et pourcentage d'infectés dès le
               départ.
            if (tbxNbPeople.Text.Length > 0)
                SimulationGeneralParameters.NbPeople =
                    Convert.ToInt32(tbxNbPeople.Text);

            if (tbxProbabilityOfInfected.Text.Length > 0)
                SimulationGeneralParameters.ProbabilityOfInfected =
                    (float)Convert.ToDouble(tbxProbabilityOfInfected.Text)
                    / 100;

            // Si les mesures sont activées où non.
            if (chxMask.IsChecked == true)
                SimulationGeneralParameters.IsMaskMeasuresEnabled =
                    true;

            if (chxDistanciation.IsChecked == true)
                SimulationGeneralParameters.IsDistanciationMeasuresEnabled
                    = true;

            if (chxQuarantaine.IsChecked == true)
                SimulationGeneralParameters.IsQuarantineMeasuresEnabled
                    = true;

            if (chxVaccination.IsChecked == true)
                SimulationGeneralParameters.IsVaccinationMeasuresEnabled
                    = true;

            // Trigger des mesures
            // Masques
            if (tbxMaskNbPeopleToStart.Text.Length > 0)
                SimulationGeneralParameters.NbInfecetdForMaskActivation
                    = Convert.ToInt32(tbxMaskNbPeopleToStart.Text);

            if (tbxMaskNbPeopleToStop.Text.Length > 0)
                SimulationGeneralParameters.NbInfecetdForMaskDeactivation
                    = Convert.ToInt32(tbxMaskNbPeopleToStop.Text);
```

```
            // Distanciation
            if (tbxDistanciationNbPeopleToStart.Text.Length > 0)
                SimulationGeneralParameters.NbInfecetdForDistanciationActivation
                    =
                    Convert.ToInt32(tbxDistanciationNbPeopleToStart.Text);

            if (tbxDistanciationNbPeopleToStop.Text.Length > 0)
                SimulationGeneralParameters.NbInfecetdForDistanciationDeactivation
                    =
                    Convert.ToInt32(tbxDistanciationNbPeopleToStop.Text);

            // Quarantaine
            if (tbxQuarantineNbPeopleToStart.Text.Length > 0)
                SimulationGeneralParameters.NbInfecetdForQuarantineActivation
                    = Convert.ToInt32(tbxQuarantineNbPeopleToStart.Text);

            if (tbxQuarantineNbPeopleToStop.Text.Length > 0)
                SimulationGeneralParameters.NbInfecetdForQuarantineDeactivation
                    = Convert.ToInt32(tbxQuarantineNbPeopleToStop.Text);

            // Vaccination
            if (tbxVaccinationNbPeopleToStart.Text.Length > 0)
                SimulationGeneralParameters.NbInfecetdForVaccinationActivation
                    =
                    Convert.ToInt32(tbxVaccinationNbPeopleToStart.Text);

            if (tbxVaccinationNbPeopleToStop.Text.Length > 0)
                SimulationGeneralParameters.NbInfecetdForVaccinationDeactivation
                    = Convert.ToInt32(tbxVaccinationNbPeopleToStop.Text);
        }

        /// <summary>
        /// Sauvegarde les paramètres des masques s'il sont actifs.
        /// </summary>
        private void SetMaskMeasureParameters()
        {
            MaskParameters.Init();

            // Nombre d'individu, et pourcentage d'infectés dès le
               départ.
            if (rdbClientMaskIsOn.IsChecked == true)
                MaskParameters.IsClientMaskOn =
                    (rdbClientMaskIsOn.IsChecked == true);

            if (rdbPersonnelMaskIsOn.IsChecked == true)
                MaskParameters.IsPersonnelMaskOn =
                    (rdbPersonnelMaskIsOn.IsChecked == true);
        }

        /// <summary>
        /// Sauvegarde les paramètres des mesures de quarantaines.
        /// </summary>
        private void SetQuarantineMeasureParameters()
        {
```

```
            QuarantineParameters.Init();

            // Saint
            if (chxQuarantineHealthy.IsChecked == true)
            {
                QuarantineParameters.IshealthyQuarantined = true;
                QuarantineParameters.ProbabilityOfHealthyQuarantined =
                    Convert.ToDouble(tbxProbabilityForHealthyToBeQuarantined.Text)
                    / 100;
                QuarantineParameters.DurationHealthyQuarantined =
                    Convert.ToInt32(tbxQuarantinedDurationForHealthy.Text);
            }
            else
            {
                QuarantineParameters.IshealthyQuarantined = false;
            }

            // Infectés
            if (chxQuarantineInfected.IsChecked == true)
            {
                QuarantineParameters.IsInfectedQuarantined = true;
                QuarantineParameters.ProbabilityOfInfectedQuarantined =
                    Convert.ToDouble(tbxProbabilityForInfectedToBeQuarantined.Text)
                    / 100;
                QuarantineParameters.DurationInfectedQuarantined =
                    Convert.ToInt32(tbxQuarantinedDurationForInfected.Text);
            }
            else
            {
                QuarantineParameters.IsInfectedQuarantined = false;
            }

            // Contagieux
            if (chxQuarantineInfectious.IsChecked == true)
            {
                QuarantineParameters.IsInfectiousQuarantined = true;
                QuarantineParameters.ProbabilityOfHealthyQuarantined =
                    Convert.ToDouble(tbxProbabilityForInfectiousToBeQuarantined.Text)
                    / 100;
                QuarantineParameters.DurationInfectiousQuarantined =
                    Convert.ToInt32(tbxQuarantinedDurationForInfectious.Text);
            }
            else
            {
                QuarantineParameters.IsInfectiousQuarantined = false;
            }

            // Immunisés
            if (chxQuarantineImmune.IsChecked == true)
            {
                QuarantineParameters.IsImmuneQuarantined = true;
                QuarantineParameters.ProbabilityOfImmuneQuarantined =
                    Convert.ToDouble(tbxProbabilityForImmuneToBeQuarantined.Text)
                    / 100;
                QuarantineParameters.DurationImmuneQuarantined =
```

```
                    Convert.ToInt32(tbxQuarantinedDurationForImmune.Text);
            }
            else
            {
                QuarantineParameters.IsImmuneQuarantined = false;
            }
        }

        /// <summary>
        /// Sauvegarde les paramètres de vaccination.
        /// </summary>
        private void SetVaccinationMeasureParameters()
        {
            VaccinationParameters.Init();

            // Durée du vaccin
            if (tbxVaccinationDuration.Text.Length > 0)
                VaccinationParameters.Duration =
                    Convert.ToInt32(tbxVaccinationDuration.Text);

            // Efficacité du vaccin
            if (tbxVaccinationEfficiency.Text.Length > 0)
                VaccinationParameters.Efficiency =
                    Convert.ToInt32(tbxVaccinationEfficiency.Text);
        }

        /// <summary>
        /// Annule les modifications et remet les paramètres précédents.
        /// </summary>
        private void Cancel_Click(object sender, RoutedEventArgs e)
        {
            SetUIParameters();
            SetParametersData();
        }

        /// <summary>
        /// Remet les paramètres par défaut.
        /// </summary>
        private void Default_Click(object sender, RoutedEventArgs e)
        {
            VirusParameters.Init();
            SimulationGeneralParameters.Init();
            MaskParameters.Init();
            VaccinationParameters.Init();
            QuarantineParameters.Init();

            SetUIParameters();
            SetParametersData();
        }
    }
}
```

Listing 3 – Code source de GraphicsEnums.cs

```
/*
```

```
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste représentant une ville.
 */
using System.ComponentModel;

namespace CovidPropagation
{
    public enum UIType
    {
        [Description("Linéaire")]
        Linear,
        [Description("Vertical")]
        Vertical,
        [Description("Horizontal")]
        Horizontal,
        [Description("Secteur")]
        PieChart,
        [Description("Carte thermique")]
        HeatMap,
        [Description("GUI")]
        GUI
    }

    public enum ChartsAxisData
    {
        [Description("Périodes")]
        TimeFrame,
        [Description("Jour")]
        Day,
        [Description("Semaine")]
        Week,
        [Description("Nombre de personnes")]
        QuantityOfPeople,
        [Description("Nombre de cas")]
        QuantityOfCase,
        [Description("Nombre de décès")]
        QuantityOfDeath,
        [Description("Nombre d'hospitalisation")]
        QuantityOfHospitalisation,
        [Description("Nombre d'immunisés")]
        QuantityOfImmune,
        [Description("Nombre de contaminés")]
        QuantityOfContamination,
        [Description("Nombre de reproduction")]
        Re,
    }

    public enum ChartsDisplayData
    {
        [Description("Personnes")]
        Persons,
```

```
        [Description("Cas")]
        Cases,
        [Description("Infectieux")]
        Infectious,
        [Description("Incubation")]
        Incubations,
        [Description("Sains")]
        Healthy,
        [Description("Décès")]
        Death,
        [Description("Immunisés")]
        Immune,
        [Description("Hospitalisés")]
        Hospitalisation,
        [Description("Re")]
        Re,
        [Description("Contamination")]
        Contamination
    }

    public enum ChartsDisplayInterval
    {
        [Description("Jour")]
        Day,
        [Description("Semaine")]
        Week,
        [Description("Mois")]
        Month,
        [Description("Total")]
        Total
    }
}
```

Listing 4 – Code source de WindowRawDatas.cs

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste représentant une ville.
 */
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media;

namespace CovidPropagation
{
    /// <summary>
    /// Classe permettant l'affichage des données brutes de la
```

```
        simulation dans une page indépendante.
    /// </summary>
    public partial class WindowRawDatas : Window
    {
        List<Label> labelsName;
        List<Label> labelsValue;
        public WindowRawDatas()
        {
            InitializeComponent();
            labelsName = new List<Label>();
            labelsValue = new List<Label>();
        }

        /// <summary>
        /// Créé les différents groupes de labels et les positionnes.
        /// </summary>
        /// <param name="datas">Données de la simulation.</param>
        public void CreateLabels(SimulationDatas datas)
        {
            CreateRows((datas.CentralizedDatas.Count + 1) * 2);

            // Last
            int columnLast = 0;
            int rowLast = 1;
            CreateTitleLabel("Denières valeurs enregistrée:",
                columnLast, rowLast - 1);
            rowLast = CreateLabelsGroup(columnLast, rowLast, datas);

            // Average
            int columnAverage = 2;
            int rowAverage = 1;
            CreateTitleLabel("Moyenne des valeurs enregistrées:",
                columnAverage, rowAverage - 1);
            CreateLabelsGroup(columnAverage, rowAverage, datas);

            // Max
            int columnMax = 0;
            int rowMax = rowLast + 1;
            CreateTitleLabel("Valeurs maximums enregistrées:",
                columnMax, rowMax - 1);
            CreateLabelsGroup(columnMax, rowMax, datas);

            // Min
            int columnMin = 2;
            int rowMin = rowLast + 1;
            CreateTitleLabel("Valeurs minimums enregistrées:",
                columnMin, rowMin - 1);
            CreateLabelsGroup(columnMin, rowMin, datas);
        }

        /// <summary>
        /// Créé plusieurs labels formant un groupe.
        /// </summary>
        /// <param name="column">Colonne où le groupe doit être
            affiché.</param>
```

```
        /// <param name="row">La ligne de départ de ce groupe de
            labels.</param>
        /// <param name="datas">Les données que les labels
            afficheront.</param>
        /// <returns></returns>
        private int CreateLabelsGroup(int column, int row,
            SimulationDatas datas)
        {
            foreach (KeyValuePair<string, List<double>> item in
                datas.CentralizedDatas)
            {
                CreateLabel(item.Key, item.Value.Last(), column, row);
                row++;
            }
            return row;
        }

        /// <summary>
        /// Créé les labels à la position demandée avec le contenu
            demandé.
        /// La clé étant le nom des données et la valeurs, la valeur
            des données.
        /// </summary>
        /// <param name="key">Nom des données.</param>
        /// <param name="value">Valeur des données.</param>
        /// <param name="x">Position X.</param>
        /// <param name="y">Position Y.</param>
        private void CreateLabel(string key, double value, int x, int y)
        {
            Label labelName = new Label();
            Label labelValue = new Label();

            labelName.Content = $"{key}: ";
            labelName.Foreground = Brushes.LightGray;
            labelName.FontSize = 15;

            labelValue.Content = $"{value}";
            labelValue.Foreground = Brushes.LightGray;
            labelValue.FontSize = 15;
            labelValue.BorderThickness = new Thickness(0.5d,0,3,0);
            labelValue.BorderBrush = Brushes.LightGray;

            Grid.SetColumn(labelName, x);
            Grid.SetRow(labelName, y);

            Grid.SetColumn(labelValue, x + 1);
            Grid.SetRow(labelValue, y);

            grdContent.Children.Add(labelName);
            grdContent.Children.Add(labelValue);

            labelsName.Add(labelName);
            labelsValue.Add(labelValue);
        }
```

```
        /// <summary>
        /// Créé les labels de titres à la position demandée avec le
            contenu demandé.
        /// Change la police ainsi que le background.
        /// </summary>
        /// <param name="content">Contenu du label.</param>
        /// <param name="x">Position X.</param>
        /// <param name="y">Position Y.</param>
        private void CreateTitleLabel(string content, int x, int y)
        {
            Label labelTitle = new Label();

            labelTitle.Content = $"{content}";
            labelTitle.Foreground = this.FindResource("normalDark") as
                Brush;
            labelTitle.FontSize = 18;
            labelTitle.Background = this.FindResource("normalGreen") as
                Brush;

            Grid.SetColumn(labelTitle, x);
            Grid.SetRow(labelTitle, y);
            Grid.SetColumnSpan(labelTitle, 2);

            grdContent.Children.Add(labelTitle);
        }

        /// <summary>
        /// Créé une ou plusieurs nouvelles lignes dans la grid.
        /// </summary>
        /// <param name="nbRows">Le nombre de ligne à ajouter.</param>
        private void CreateRows(int nbRows)
        {
            for (int i = 0; i < nbRows; i++)
            {
                RowDefinition newRow = new RowDefinition();
                newRow.MaxHeight = 30;
                newRow.MinHeight = 30;
                grdContent.RowDefinitions.Add(newRow);
            }
        }

        /// <summary>
        /// Met à jour les labels avec les derières données.
        /// </summary>
        /// <param name="datas">Données à afficher</param>
        public void UpdateLabels(SimulationDatas datas)
        {
            try
            {
                Dispatcher.Invoke(() =>
                {
                    // Affiche les dernières données de la simulation.
                    int labelsIndex = 0;
                    int index = 0;
                    foreach (KeyValuePair<string, List<double>> item in
```

```
                    datas.CentralizedDatas)
                {
                    labelsName[labelsIndex].Content = $"{item.Key}:
                        ";
                    labelsValue[labelsIndex].Content =
                        $"{item.Value.Last().ToString("F2")}";
                    labelsIndex++;
                }

                // Affiche la moyenne des données de la simulation.
                index = 0;
                foreach (KeyValuePair<string, List<double>> item in
                    datas.CentralizedDatas)
                {
                    labelsName[labelsIndex].Content = $"{item.Key}:
                        ";
                    labelsValue[labelsIndex].Content =
                        $"{item.Value.Average().ToString("F2")}";
                    labelsIndex++;
                    index++;
                }

                // Affiche la valeur maximum enregistrée des
                    données de la simulation
                index = 0;
                foreach (KeyValuePair<string, List<double>> item in
                    datas.CentralizedDatas)
                {
                    labelsName[labelsIndex].Content = $"{item.Key}:
                        ";
                    labelsValue[labelsIndex].Content =
                        $"{item.Value.Max().ToString("F2")}";
                    labelsIndex++;
                    index++;
                }

                // Affiche la valeur minimum enregistrée des
                    données de la simulation
                index = 0;
                foreach (KeyValuePair<string, List<double>> item in
                    datas.CentralizedDatas)
                {
                    labelsName[labelsIndex].Content = $"{item.Key}:
                        ";
                    labelsValue[labelsIndex].Content =
                        $"{item.Value.Min().ToString("F2")}";
                    labelsIndex++;
                    index++;
                }
            });
        }
        catch (Exception ex)
        {
            Debug.WriteLine("Update labels error : " + ex);
        }
```

```
        }
    }
}
```

Listing 5 – Code source de PageSimulation.cs

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste représentant une ville.
 */
using System;
using System.Collections.Generic;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media;
using System.Diagnostics;
using LiveCharts.Wpf;
using LiveCharts;
using System.ComponentModel;
using System.Reflection;
using LiveCharts.Defaults;
using System.Linq;
using LiveCharts.Geared;
using System.Threading;
using System.Threading.Tasks;
using System.Runtime.InteropServices;
using System.IO.Pipes;
using System.IO;
using System.Text.Json;

namespace CovidPropagation
{
    /// <summary>
    /// Logique d'interaction pour PageSimulation.xaml
    /// </summary>
    public partial class PageSimulation : Page
    {
        // Unity Process
        [DllImport("user32.dll")]
        static extern bool MoveWindow(IntPtr handle, int x, int y, int
            width, int height, bool redraw);

        internal delegate int WindowEnumProc(IntPtr hwnd, IntPtr
            lparam);
        [DllImport("user32.dll")]
        internal static extern bool EnumChildWindows(IntPtr hwnd,
            WindowEnumProc func, IntPtr lParam);

        [DllImport("user32.dll")]
        static extern int SendMessage(IntPtr hWnd, int msg, IntPtr
            wParam, IntPtr lParam);
```

```
        private const double FONTSIZE = 15;
        private const double FIRST_CASE_COLUMN_SIZE = 20;
        private const double SECOND_CASE_COLUMN_SIZE = 20;
        private const double THIRD_CASE_COLUMN_SIZE = 70;
        private const double FOURTH_CASE_COLUMN_SIZE = 50;

        private const double FIRSTROW_CASE_MIN_HEIGHT = 20;
        private const double FIRSTROW_CASE_MAX_HEIGHT = 20;

        private const string BTNLEFT_CASE_DISPLAY_CHARACTER = "<";
        private const string BTNRIGHT_CASE_DISPLAY_CHARACTER = ">";
        private const string BTNAUTO_CASE_DISPLAY_CHARACTER = "auto.";

        private const int BTNLEFT_CASE_COLUMN_POSITION = 0;
        private const int BTNLEFT_CASE_ROW_POSITION = 0;

        private const int BTNRIGHT_CASE_COLUMN_POSITION = 1;
        private const int BTNRIGHT_CASE_ROW_POSITION = 0;

        private const int CBXTIMEINCREMENT_CASE_COLUMN_POSITION = 2;
        private const int CBXTIMEINCREMENT_CASE_ROW_POSITION = 0;

        private const int BTNAUTO_CASE_COLUMN_POSITION = 3;
        private const int BTNAUTO_CASE_ROW_POSITION = 0;

        private const int CHART_CASE_COLUMN_POSITION = 1;
        private const int CHART_CASE_ROW_POSITION = 1;
        private const int CHART_CASE_COLUMN_SPAN = 5;
        private const int CHART_CASE_ROW_SPAN = 1;

        private const double DEFAULT_AXIS_MAXVALUE = 10;

        private const double HEATMAP_GREEN_DEFAULT_VALUE = 0;
        private const double HEATMAP_GREEN_YELLOW_DEFAULT_VALUE = 0.25;
        private const double HEATMAP_YELLOW_DEFAULT_VALUE = 0.5;
        private const double HEATMAP_ORANGE_DEFAULT_VALUE = 0.75;
        private const double HEATMAP_RED_DEFAULT_VALUE = 1;

        private const string UNITY_INITIALIZE_STRING = "Initialize ";
        private const string UNITY_ITERATE_STRING = "Iterate ";

        private const string PIPELINE_NAME = "SimulationToUnity";

        WindowRawDatas rawDatasWindow;
        MainWindow mw;
        public ChartData[,] chartDatas; // Contient les données des
            graphiques avec une position x et y
        Simulation sim;
        Dictionary<UIType, object> charts;
        Grid grdStruct;

        System.Windows.Forms.Integration.WindowsFormsHost wfhUnityHost;
            // Contient le panel Unity
        System.Windows.Forms.Panel panelUnity; // Affiche la fenêtre
```

```
        unity

    private Process process; // Process contenant le programme Unity
    private IntPtr unityHWND = IntPtr.Zero; // Handler d'unity
    NamedPipeServerStream pipeServer; // Permet la création du
        pipeline connectant WPF à Unity
    StreamString ss; // Stream permettant l'envoie de données à
        Unity
    Thread server;

    public PageSimulation()
    {
        InitializeComponent();
        rawDatasWindow = new WindowRawDatas();
        mw = (MainWindow)Application.Current.MainWindow;
        charts = new Dictionary<UIType, object>();
        sim = new Simulation();
    }

    private void OpenRawDatasWindow_Click(object sender,
        RoutedEventArgs e)
    {
        rawDatasWindow.Show();
        rawDatasWindow.Focus();
    }

    /// <summary>
    /// Modifie l'interval et donc la vitesse de la simulation en
        fonction de la valeur du slider.
    /// </summary>
    private void IntervalSlider_DragCompleted(object sender,
        System.Windows.Controls.Primitives.DragCompletedEventArgs e)
    {
        sim.Interval = Convert.ToInt32(intervalSlider.Maximum -
            intervalSlider.Value);
    }

    /// <summary>
    /// Créé la simulation si celle-ci n'est pas initialisée ou
        relance le timer.
    /// </summary>
    private void Start_Click(object sender, RoutedEventArgs e)
    {
        if (!sim.IsInitialized)
        {
            sim.Initialize();
            sim.Interval = GlobalVariables.DEFAULT_INTERVAL;

            if(panelUnity != null)
            {
                sim.OnGUIUpdate += new
                    GUIDataEventHandler(OnGUIUpdate);
                sim.OnGUIInitialize += new
                    InitializeGUIEventHandler(OnGUIInitialize);
            }
```

```
            rawDatasWindow.CreateLabels(sim.GetAllDatas(0,0,0,0,0));
            sim.OnDataUpdate += new
                DataUpdateEventHandler(rawDatasWindow.UpdateLabels);

            if (panelUnity == null)
            {
                StartSimulationIteration();
            }
            else
            {
                LoadUnityExe();
                ConnectToUnity(); // Connexion à unity puis
                    démarrage de l'itération.
            }

            intervalSlider.Value =
                Convert.ToInt32(intervalSlider.Maximum -
                sim.Interval);
            mw.btnGraphicSettings.IsEnabled = false;
            mw.btnSettings.IsEnabled = false;
            btnOpenRawDatas.IsEnabled = true;
        }
        sim.Start();
    }

    private void StartSimulationIteration()
    {
        Task.Factory.StartNew(() => sim.Iterate());
    }

    /// <summary>
    /// Met en pause la simulation
    /// </summary>
    private void Break_Click(object sender, RoutedEventArgs e)
    {
        sim.Stop();
    }

    private void Reset_Click(object sender, RoutedEventArgs e)
    {
        sim.Stop();
        sim = new Simulation();
        if (chartDatas == null)
            chartDatas = new ChartData[0,0];

        for (int x = 0; x < chartDatas.GetLength(0); x++)
        {
            for (int y = 0; y < chartDatas.GetLength(1); y++)
            {
                chartDatas[x, y].DisplayWindow = 0;
            }
        }

        grdContent.Children.Clear();
```

```
            if (panelUnity != null)
            {
                CloseUnity();
            }

            SetGrid(grdStruct, chartDatas);
            mw.btnGraphicSettings.IsEnabled = true;
            mw.btnSettings.IsEnabled = true;
            btnOpenRawDatas.IsEnabled = false;
        }

        #region View

        /// <summary>
        /// Modifie la grille de cette page pour qu'elle correspondent à
            celle modifiée dans les paramètres graphiques.
        /// </summary>
        /// <param name="grd">Grille contenant les colonnes et lignes à
           afficher.</param>
        /// <param name="chartDatas">Données des graphiques à afficher
           dans la grille.</param>
        public void SetGrid(Grid grd, ChartData[,] chartDatas)
        {
            this.chartDatas = chartDatas;
            grdStruct = grd;
            grdContent = grd;
            slvScroller.Content = grdContent;
            charts.Clear();
            DisplayUI();
        }

        /// <summary>
        /// Permet de déplacer l'affichage de données dans le temps en
           fonction de l'interval actuel du graphique.
        /// (Passer au jours suivant par exemple.)
        /// </summary>
        private void MoveChartDataForward_Click(object sender,
           RoutedEventArgs e)
        {
            Button btn = (Button)sender;
            Grid chartGrid = VisualTreeHelper.GetParent(btn) as Grid;
            CartesianChart chart =
                (CartesianChart)chartGrid.Children[4];
            ChartData chartData = (ChartData)chart.Tag;

            chartData.DisplayWindow++;
            chartData.AutoDisplay = false;

            chart.Tag = chartData;
            sim.TriggerDisplayChanges();
        }

        /// <summary>
        /// Permet de déplacer l'affichage de données dans le temps en
```

```
        fonction de l'interval actuel du graphique.
    /// (Passer au jours précédent par exemple.)
    /// </summary>
    private void MoveChartDataBackward_Click(object sender,
        RoutedEventArgs e)
    {
        Button btn = (Button)sender;
        Grid chartGrid = VisualTreeHelper.GetParent(btn) as Grid;
        CartesianChart chart =
            (CartesianChart)chartGrid.Children[4];
        ChartData chartData = (ChartData)chart.Tag;

        if (chartData.DisplayWindow > 0)
            chartData.DisplayWindow--;

        chartData.AutoDisplay = false;

        chart.Tag = chartData;
        sim.TriggerDisplayChanges();
    }

    /// <summary>
    /// Lorsque la sélection de l'interval de temps des graphiques
        change.
    /// Réaffiche les graphiques pour appliquer la modification.
    /// </summary>
    private void TimeInterval_SelectionChanged(object sender,
        SelectionChangedEventArgs e)
    {
        ComboBox cbx = (ComboBox)sender;
        Grid chartGrid = VisualTreeHelper.GetParent(cbx) as Grid;
        CartesianChart chart =
            (CartesianChart)chartGrid.Children[4];
        ChartData chartData = (ChartData)chart.Tag;
        chartData.DisplayInterval = cbx.SelectedIndex;
        chart.Tag = chartData;

        sim.TriggerDisplayChanges();
    }

    /// <summary>
    /// Active le mode autoDisplay sur le graphique.
    /// Une fois activé, l'affichage suis les dernières données du
        graphique automatiquement.
    /// </summary>
    private void MoveChartAuto_Click(object sender, RoutedEventArgs
        e)
    {
        Button btn = (Button)sender;
        Grid chartGrid = VisualTreeHelper.GetParent(btn) as Grid;
        CartesianChart chart =
            (CartesianChart)chartGrid.Children[4];
        ChartData chartData = (ChartData)chart.Tag;
        chartData.AutoDisplay = true;
        chart.Tag = chartData;
```

```
            sim.TriggerDisplayChanges();

        }

        /// <summary>
        /// Créé un bouton qui permet de modifier le contenu des
           graphiques en se déplacant dans le jours/semaine/mois
           suivant ou précédent ou en activant l'affichage automatique.
        /// </summary>
        /// <param name="content">Contenu textuel du bouton.</param>
        /// <returns>Bonton lié au graphique.</returns>
        private Button CreateChartButton(string content)
        {
            Button btn = new Button();
            btn.Style = this.FindResource("GraphButtonStyle") as Style;
            btn.Content = content;
            btn.Foreground = Brushes.White;
            btn.FontSize = FONTSIZE;
            btn.VerticalAlignment = VerticalAlignment.Stretch;
            btn.HorizontalAlignment = HorizontalAlignment.Stretch;
            return btn;
        }

        /// <summary>
        /// Créé un combobo permettant la sélection de l'interval à
           afficher (Jours, semaine, mois et total)
        /// </summary>
        /// <returns>Combobox contenant les valeurs temporels.</returns>
        private ComboBox CreateChartCombobox()
        {
            ComboBox cbx = new ComboBox();
            cbx.VerticalAlignment = VerticalAlignment.Stretch;
            cbx.HorizontalAlignment = HorizontalAlignment.Stretch;
            cbx.ItemsSource = from ChartsDisplayInterval n
                              in
                                  Enum.GetValues(typeof(ChartsDisplayInterval))
                              select GetEnumDescription(n);

            cbx.SelectedIndex = 0;
            return cbx;
        }

        /// <summary>
        /// Affiche le bon type de graphique au bon endroit dans la
           grille.
        /// Change le contenu du graphique en fonction de son type.
        /// </summary>
        private void DisplayUI()
        {
            foreach (ChartData chartData in chartDatas)
            {
                if (chartData.SpanX > 0)
                {
                    object uiElement;
```

```
                // Créé les graphiques, leur ajoute leur contenu et
                   s'abonne à un évènement qui mettra à jour les
                   données.
                switch (chartData.UIType)
                {
                    default:
                    case (int)UIType.Linear:
                        uiElement =
                            CreateCartesianChart((ChartsAxisData)chartData.AxisX
                            (ChartsAxisData)chartData.AxisY);
                        ((CartesianChart)uiElement).Tag = chartData;
                        AddCurvesToCartesianChart((CartesianChart)uiElement,
                            Array.ConvertAll(chartData.Datas, d =>
                            (ChartsDisplayData)d));
                        SubscribeToChartEvents((CartesianChart)uiElement,
                            chartData);
                        break;
                    case (int)UIType.Vertical:
                        uiElement =
                            CreateCartesianChart((ChartsAxisData)chartData.AxisX
                            (ChartsAxisData)chartData.AxisY);
                        ((CartesianChart)uiElement).Tag = chartData;
                        AddColumnsToCartesianChart((CartesianChart)uiElement,
                            Array.ConvertAll(chartData.Datas, d =>
                            (ChartsDisplayData)d));
                        SubscribeToChartEvents((CartesianChart)uiElement,
                            chartData);
                        break;
                    case (int)UIType.Horizontal:
                        uiElement =
                            CreateCartesianChart((ChartsAxisData)chartData.AxisX
                            (ChartsAxisData)chartData.AxisY);
                        ((CartesianChart)uiElement).Tag = chartData;
                        AddRowsToCartesianChart((CartesianChart)uiElement,
                            Array.ConvertAll(chartData.Datas, d =>
                            (ChartsDisplayData)d));
                        SubscribeToChartEvents((CartesianChart)uiElement,
                            chartData);
                        break;
                    case (int)UIType.PieChart:
                        uiElement = CreatePieChart();
                        AddSectionToPieChart((PieChart)uiElement,
                            Array.ConvertAll(chartData.Datas, d =>
                            (ChartsDisplayData)d));
                        sim.OnDataUpdate += new
                            DataUpdateEventHandler(((PieChart)uiElement).OnDataU
                        break;
                    case (int)UIType.HeatMap:
                        uiElement =
                            CreateCartesianChart((ChartsAxisData)chartData.AxisX
                            (ChartsAxisData)chartData.AxisY);
                        ((CartesianChart)uiElement).Tag = chartData;
                        AddHeatMapToCartesianChart((CartesianChart)uiElement,
                            Array.ConvertAll(chartData.Datas, d =>
                            (ChartsDisplayData)d));
```

```
                    SubscribeToChartEvents((CartesianChart)uiElement,
                        chartData);
                    break;
                case (int)UIType.GUI:
                    uiElement = null;
                    wfhUnityHost = new
                        System.Windows.Forms.Integration.WindowsFormsHost();
                    panelUnity = new
                        System.Windows.Forms.Panel();
                    panelUnity.Resize += panel1_Resize;

                    wfhUnityHost.VerticalAlignment =
                        VerticalAlignment.Stretch;
                    wfhUnityHost.HorizontalAlignment =
                        HorizontalAlignment.Stretch;

                    Grid.SetColumn(wfhUnityHost, chartData.X);
                    Grid.SetRow(wfhUnityHost, chartData.Y);
                    Grid.SetColumnSpan(wfhUnityHost,
                        chartData.SpanX);
                    Grid.SetRowSpan(wfhUnityHost,
                        chartData.SpanY);

                    wfhUnityHost.Child = panelUnity;
                    grdContent.Children.Add((UIElement)wfhUnityHost);
                    break;
            }

            if (chartData.UIType != (int)UIType.PieChart &&
                chartData.UIType != (int)UIType.HeatMap &&
                chartData.UIType != (int)UIType.GUI)
            {
                Grid chartGrid = new Grid();
                ColumnDefinition firstColumn = new
                    ColumnDefinition();
                ColumnDefinition secondColumn = new
                    ColumnDefinition();
                ColumnDefinition thirdColumn = new
                    ColumnDefinition();
                ColumnDefinition thourthColumn = new
                    ColumnDefinition();
                ColumnDefinition fifthColumn = new
                    ColumnDefinition();

                firstColumn.MinWidth = FIRST_CASE_COLUMN_SIZE;
                firstColumn.MaxWidth = FIRST_CASE_COLUMN_SIZE;
                secondColumn.MinWidth = SECOND_CASE_COLUMN_SIZE;
                secondColumn.MaxWidth = SECOND_CASE_COLUMN_SIZE;
                thirdColumn.MinWidth = THIRD_CASE_COLUMN_SIZE;
                thirdColumn.MaxWidth = THIRD_CASE_COLUMN_SIZE;
                thourthColumn.MinWidth =
                    FOURTH_CASE_COLUMN_SIZE;
                thourthColumn.MaxWidth =
                    FOURTH_CASE_COLUMN_SIZE;
```

```
chartGrid.ColumnDefinitions.Add(firstColumn);
chartGrid.ColumnDefinitions.Add(secondColumn);
chartGrid.ColumnDefinitions.Add(thirdColumn);
chartGrid.ColumnDefinitions.Add(thourthColumn);
chartGrid.ColumnDefinitions.Add(fifthColumn);

RowDefinition firstRow = new RowDefinition();
RowDefinition chartRow = new RowDefinition();

firstRow.MinHeight = FIRSTROW_CASE_MIN_HEIGHT;
firstRow.MaxHeight = FIRSTROW_CASE_MAX_HEIGHT;

chartGrid.RowDefinitions.Add(firstRow);
chartGrid.RowDefinitions.Add(chartRow);

Button btnLeft =
    CreateChartButton(BTNLEFT_CASE_DISPLAY_CHARACTER);
Button btnRight =
    CreateChartButton(BTNRIGHT_CASE_DISPLAY_CHARACTER);
ComboBox cbxTimeIncrement =
    CreateChartCombobox();
Button btnAuto =
    CreateChartButton(BTNAUTO_CASE_DISPLAY_CHARACTER);

btnLeft.Click += MoveChartDataBackward_Click;
btnRight.Click += MoveChartDataForward_Click;
cbxTimeIncrement.SelectionChanged +=
    TimeInterval_SelectionChanged;
btnAuto.Click += MoveChartAuto_Click;

Grid.SetColumn(btnLeft,
    BTNLEFT_CASE_COLUMN_POSITION);
Grid.SetRow(btnLeft, BTNLEFT_CASE_ROW_POSITION);

Grid.SetColumn(btnRight,
    BTNRIGHT_CASE_COLUMN_POSITION);
Grid.SetRow(btnRight,
    BTNRIGHT_CASE_ROW_POSITION);

Grid.SetColumn(cbxTimeIncrement,
    CBXTIMEINCREMENT_CASE_COLUMN_POSITION);
Grid.SetRow(cbxTimeIncrement,
    CBXTIMEINCREMENT_CASE_ROW_POSITION);

Grid.SetColumn(btnAuto,
    BTNAUTO_CASE_COLUMN_POSITION);
Grid.SetRow(btnAuto, BTNAUTO_CASE_ROW_POSITION);

Grid.SetColumn((UIElement)uiElement,
    CHART_CASE_COLUMN_POSITION);
Grid.SetRow((UIElement)uiElement,
    CHART_CASE_ROW_POSITION);
Grid.SetColumnSpan((UIElement)uiElement,
    CHART_CASE_COLUMN_SPAN);
Grid.SetRowSpan((UIElement)uiElement,
```

```
                        CHART_CASE_ROW_SPAN);

                    chartGrid.Children.Add(btnLeft);
                    chartGrid.Children.Add(btnRight);
                    chartGrid.Children.Add(cbxTimeIncrement);
                    chartGrid.Children.Add(btnAuto);
                    chartGrid.Children.Add((UIElement)uiElement);

                    Grid.SetColumn(chartGrid, chartData.X);
                    Grid.SetRow(chartGrid, chartData.Y);
                    Grid.SetColumnSpan(chartGrid, chartData.SpanX);
                    Grid.SetRowSpan(chartGrid, chartData.SpanY);

                    grdContent.Children.Add(chartGrid);
                }
                else if(chartData.UIType == (int)UIType.PieChart ||
                    (UIType)chartData.UIType == UIType.HeatMap)
                {
                    Grid.SetColumn((UIElement)uiElement,
                        chartData.X);
                    Grid.SetRow((UIElement)uiElement, chartData.Y);
                    Grid.SetColumnSpan((UIElement)uiElement,
                        chartData.SpanX);
                    Grid.SetRowSpan((UIElement)uiElement,
                        chartData.SpanY);

                    grdContent.Children.Add((UIElement)uiElement);
                }
            }
        }
    }

    /// <summary>
    /// S'abonne aux évènements permettant l'ajout de donéées ainsi
        que la modification de l'affichage.
    /// </summary>
    /// <param name="chart"></param>
    /// <param name="chartData"></param>
    private void SubscribeToChartEvents(CartesianChart chart,
        ChartData chartData)
    {
        sim.OnDisplay += new
            DispalyChangeEventHandler(chart.Display);
    }

    /// <summary>
    /// Créé un graphique cartésien permettant l'ajout de courbe,
        colonne, ligne, heatmap.
    /// </summary>
    /// <param name="axeXDatas">Donnée sur l'axe X</param>
    /// <param name="axeYDatas">Donnée sur l'axe Y</param>
    /// <returns>Le graphique créé.</returns>
    private CartesianChart CreateCartesianChart(ChartsAxisData
        axeXDatas, ChartsAxisData axeYDatas)
    {
```

```
            CartesianChart cartesianChart = new CartesianChart();
            cartesianChart.Series = new SeriesCollection();
            cartesianChart.LegendLocation = LegendLocation.Top;
            cartesianChart.Foreground = Brushes.Gray;
            cartesianChart.DisableAnimations = true;
            cartesianChart.Hoverable = false;

            Axis axisX = CreateAxis(axeXDatas);
            axisX.MinValue = 0;
            axisX.MaxValue = 1; // Nombre de période sur une semaine.
            cartesianChart.AxisX.Add(axisX);

            Axis axisY = CreateAxis(axeYDatas);
            axisY.MinValue = 0;
            cartesianChart.AxisY.Add(axisY);

            DataContext = this;
            cartesianChart.VerticalAlignment =
                VerticalAlignment.Stretch;
            cartesianChart.HorizontalAlignment =
                HorizontalAlignment.Stretch;

            return cartesianChart;
        }

        /// <summary>
        /// Créé un graphique cylindrique.
        /// </summary>
        /// <returns>Le graphique cylindrique créé.</returns>
        private PieChart CreatePieChart()
        {
            PieChart pieChart = new PieChart();
            pieChart.Series = new SeriesCollection();
            pieChart.LegendLocation = LegendLocation.Right;
            pieChart.Foreground = Brushes.Gray;
            pieChart.DisableAnimations = true;
            pieChart.Hoverable = false;

            DataContext = this;
            pieChart.VerticalAlignment = VerticalAlignment.Stretch;
            pieChart.HorizontalAlignment = HorizontalAlignment.Stretch;

            return pieChart;
        }

        /// <summary>
        /// Créé un axe pour un graphique cartésien.
        /// </summary>
        /// <param name="axeDatas">Nom de l'axe.</param>
        /// <returns>L'axe créé.</returns>
        private Axis CreateAxis(ChartsAxisData axeDatas)
        {
            Axis axis = new Axis();
            axis.Title = GetEnumDescription(axeDatas);
            axis.MaxValue = DEFAULT_AXIS_MAXVALUE;
```

```
        return axis;
    }

    /// <summary>
    /// Ajoute une ou plusieurs courbes à un graphique cartésien.
    /// </summary>
    /// <param name="chart">graphique où la courbe sera
        ajoutée.</param>
    /// <param name="curvesData">Type de données à afficher par
        courbe.</param>
    private void AddCurvesToCartesianChart(CartesianChart chart,
        ChartsDisplayData[] curvesData)
    {
        for (int i = 0; i < curvesData.Length; i++)
        {
            ChartValues<double> values = new ChartValues<double>();
            chart.Series.Add(new LineSeries
            {
                Title =
                    GetEnumDescription(curvesData[i]).ToString(),
                Foreground = Brushes.Gray,
                Tag = curvesData[i],
                PointGeometry = null,
                Values = values,
                DataLabels = false,
            });;
            chart.Series[0].Values =
                values.AsGearedValues().WithQuality(Quality.Low);
        }
    }

    /// <summary>
    /// Ajoute une ou plusieurs colonnes à un graphique cartésien.
    /// </summary>
    /// <param name="chart">graphique où la colonne sera
        ajoutée.</param>
    /// <param name="curvesData">Type de données à afficher par
        colonne.</param>
    private void AddColumnsToCartesianChart(CartesianChart chart,
        ChartsDisplayData[] curvesData)
    {
        for (int i = 0; i < curvesData.Length; i++)
        {
            ChartValues<double> values = new ChartValues<double>();
            chart.Series.Add(new ColumnSeries
            {
                Title =
                    GetEnumDescription(curvesData[i]).ToString(),
                Foreground = Brushes.Gray,
                Tag = curvesData[i],
                Values = values,
                DataLabels = false
            });
        }
```

```
        }

        /// <summary>
        /// Ajoute une ou plusieurs lignes à un graphique cartésien.
        /// </summary>
        /// <param name="chart">graphique où la ligne sera
            ajoutée.</param>
        /// <param name="curvesData">Type de données à afficher par
            ligne.</param>
        private void AddRowsToCartesianChart(CartesianChart chart,
            ChartsDisplayData[] curvesData)
        {
            for (int i = 0; i < curvesData.Length; i++)
            {
                chart.Series.Add(new RowSeries
                {
                    Title =
                        GetEnumDescription(curvesData[i]).ToString(),
                    Foreground = Brushes.Gray,
                    Tag = curvesData[i],
                    Values = new ChartValues<double> ()
                });
            }
        }

        /// <summary>
        /// Ajoute une ou plusieurs section à un graphique cylindrique.
        /// </summary>
        /// <param name="chart">Graphique où la section sera
            ajoutée.</param>
        /// <param name="curvesData">Type de données à afficher par
            section.</param>
        private void AddSectionToPieChart(PieChart chart,
            ChartsDisplayData[] curvesData)
        {
            for (int i = 0; i < curvesData.Length; i++)
            {
                chart.Series.Add(new PieSeries
                {
                    Title =
                        GetEnumDescription(curvesData[i]).ToString(),
                    Foreground = Brushes.Gray,
                    Tag = curvesData[i],
                    Values = new ChartValues<double>()
                });
            }
        }

        /// <summary>
        /// Ajoute une heatMap à un graphique cartésien.
        /// </summary>
        /// <param name="chart">graphique où la heatMap sera
            ajoutée.</param>
        /// <param name="heatMapData">Type de données à afficher dans
            la heatMap.</param>
```

```
        private void AddHeatMapToCartesianChart(CartesianChart chart,
            ChartsDisplayData[] heatMapData)
        {
            ChartValues<HeatPoint> values = new
                ChartValues<HeatPoint>();

            HeatSeries heatSeries = new HeatSeries();
            heatSeries.Values = values;
            heatSeries.Title =
                GetEnumDescription(heatMapData[0]).ToString();
            heatSeries.Tag = heatMapData[0];
            heatSeries.DataLabels = false;
            heatSeries.GradientStopCollection = new
                GradientStopCollection()
            {
                new GradientStop(Colors.Green,
                    HEATMAP_GREEN_DEFAULT_VALUE),
                new GradientStop(Colors.GreenYellow,
                    HEATMAP_GREEN_YELLOW_DEFAULT_VALUE),
                new GradientStop(Colors.Yellow,
                    HEATMAP_YELLOW_DEFAULT_VALUE),
                new GradientStop(Colors.Orange,
                    HEATMAP_ORANGE_DEFAULT_VALUE),
                new GradientStop(Colors.Red, HEATMAP_RED_DEFAULT_VALUE)
            };
            chart.Series.Add(heatSeries);
        }

        /// <summary>
        /// Récupère la description des valeurs d'un enum pour un
            affichage plus logique.
        /// </summary>
        /// <param name="value">Valeur du enum à convertir en
            description.</param>
        /// <returns>Description du enum</returns>
        public static string GetEnumDescription(Enum value)
        {
            string result;
            FieldInfo fi = value.GetType().GetField(value.ToString());
            DescriptionAttribute[] attributes =
                fi.GetCustomAttributes(typeof(DescriptionAttribute),
                false) as DescriptionAttribute[];

            if (attributes != null && attributes.Any())
                result = attributes.First().Description;
            else
                result = value.ToString();

            return result;
        }

        #endregion

        #region Unity
```

```
        /// <summary>
        /// Lance le programme unity situé dans les fichier de ce
            programme.
        /// Lui donne un handle qui s'occupera de le positionner et le
            controler.
        /// </summary>
        public void LoadUnityExe()
        {
            IntPtr unityHandle = panelUnity.Handle;

            //Start embedded Unity Application
            Task.Run(() =>
            {
                process = new Process();
                //process.StartInfo.FileName =
                    @".\GUIBuild\CovidPropagationGUI.exe";
                process.StartInfo.FileName =
                    @".\GUIBuild2D\CovidPropagationGUI2D.exe";
                process.StartInfo.Arguments = "-parentHWND " +
                    unityHandle.ToInt32() + " " +
                    Environment.CommandLine;
                process.StartInfo.UseShellExecute = true;
                process.StartInfo.CreateNoWindow = true;
                process.Start();

                if (process.WaitForInputIdle())
                {
                    EnumChildWindows(unityHandle, WindowEnum,
                        IntPtr.Zero);
                }
            });
        }

        private int WindowEnum(IntPtr hwnd, IntPtr lparam)
        {
            unityHWND = hwnd;
            return 0;
        }


        /// <summary>
        /// Se connecte à unity via un pipeline.
        /// </summary>
        public void ConnectToUnity()
        {
            server = new Thread(ServerThread);
            server.Start();
        }

        /// <summary>
        /// Initialise le pipeline et attend que la connexion soit é
            tablis pour lancer la simulation.
        /// </summary>
        private void ServerThread(object data)
        {
```

```
            int numThreads = 1;
            pipeServer = new NamedPipeServerStream(PIPELINE_NAME,
                PipeDirection.Out, numThreads);

            pipeServer.WaitForConnection();

            ss = new StreamString(pipeServer);
            StartSimulationIteration();
        }

        /// <summary>
        /// Stop la connexion entre WPF et unity et éteint les
            processus d'unity.
        /// </summary>
        private void CloseUnity()
        {
            if (ss != null)
            {
                ss.CloseLink();
            }

            process.CloseMainWindow();
            process.Kill();
            process.Close();
        }

        /// <summary>
        /// Lorsque la page est fermée, ferme le programme unity ainsi
            que le pipeline.
        /// </summary>
        private void Window_Closed(object sender, EventArgs e)
        {
            CloseUnity();
        }

        /// <summary>
        /// Resize la fenêtre unity lorsque le panel le contenant
            change de taille.
        /// </summary>
        private void panel1_Resize(object sender, EventArgs e)
        {
            MoveWindow(unityHWND, 0, 0, panelUnity.Width,
                panelUnity.Height, true);
        }

        /// <summary>
        /// Convertit les objets en json et les envoies au GUI pour
            créer l'interface graphique en fonction des données de la
            simulation.
        /// </summary>
        /// <param name="populationDatas">Données de la population à
            envoyer.</param>
        /// <param name="siteDatas">Données des sites à envoyer.</param>
        private void OnGUIInitialize(DataPopulation populationDatas,
            DataSites siteDatas)
```

```
        {
            if (ss != null)
            {
                string objectToSend = UNITY_INITIALIZE_STRING;
                objectToSend +=
                    JsonSerializer.Serialize(populationDatas);
                objectToSend += " " +
                    JsonSerializer.Serialize(siteDatas);
                ss.WriteString(objectToSend);
            }
        }

        /// <summary>
        /// Lorsque la simulation fait une itération, récupère les
            données de celle-ci et les envoies au GUI au format JSON.
        /// </summary>
        private void OnGUIUpdate(int[] personsNewSite, int[]
            personsNewState)
        {
            if (ss != null)
            {
                DataIteration jsonIteration = new
                    DataIteration(personsNewSite, personsNewState);
                string objectToSend = UNITY_ITERATE_STRING;
                objectToSend += JsonSerializer.Serialize(jsonIteration);
                ss.WriteString(objectToSend);
            }
        }
        #endregion
    }
}
```

Listing 6 – Code source de App.cs

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Input;
using System.Diagnostics;

namespace CovidPropagation
{
    /// <summary>
    /// Interaction logic for App.xaml
    /// </summary>
    public partial class App : Application
    {
        private void Close_Click(object sender, RoutedEventArgs e)
        {
            Window currentWindow = GetCurrentWindow();
            if (currentWindow.GetType() == typeof(MainWindow))
```

```
                Current.Shutdown();
            else
                GetCurrentWindow().Hide();
        }

        private void Maximize_Click(object sender, RoutedEventArgs e)
        {
            Window currentWindow = GetCurrentWindow();
            if (currentWindow.WindowState == WindowState.Normal)
                currentWindow.WindowState = WindowState.Maximized;
            else
                currentWindow.WindowState = WindowState.Normal;
        }

        private void Minimize_Click(object sender, RoutedEventArgs e)
        {
            GetCurrentWindow().WindowState = WindowState.Minimized;
        }

        private void TextBlock_MouseDown(object sender,
           System.Windows.Input.MouseButtonEventArgs e)
        {
            if (e.ChangedButton == MouseButton.Left)
                GetCurrentWindow().DragMove();
        }

        private Window GetCurrentWindow()
        {
            return Current.Windows.OfType<Window>().SingleOrDefault(x
               => x.IsActive);
        }
    }
}
```

Listing 7 – Code source de PageGraphicSettings.cs

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste représentant une ville.
 */
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;

namespace CovidPropagation
{
    /// <summary>
    /// Logique d'interaction pour PageGraphicSettings.xaml
```

```
    /// </summary>
    public partial class PageGraphicSettings : Page
    {
        private const int ROW_MIN_HEIGHT = 150;
        private const int COLUMN_MIN_WIDTH = 150;
        private const int MAX_GRID_SIZE = 10;

        private const int DEFAULT_AXIS_X_VALUE = 0;
        private const int DEFAULT_AXIS_Y_VALUE = 3;
        private const int DEFAULT_CHART_TYPE = 0;

        public static ChartData[,] chartsDatas = new
            ChartData[MAX_GRID_SIZE, MAX_GRID_SIZE];
        private bool[,] gridHasContent = new bool[MAX_GRID_SIZE,
            MAX_GRID_SIZE];
        public int oldX = MAX_GRID_SIZE + 1;
        int oldY = MAX_GRID_SIZE + 1;
        int maxCellColumnSpan;
        int maxCellRowSpan;

        Grid dynamicGrid;

        public PageGraphicSettings()
        {
            InitializeComponent();

            dynamicGrid = grdContent;
            scrollerViewer.Content = dynamicGrid;
            maxCellColumnSpan = dynamicGrid.ColumnDefinitions.Count;
            maxCellRowSpan = dynamicGrid.RowDefinitions.Count;
        }

        private void AddRow_Click(object sender, RoutedEventArgs e)
        {
            if (dynamicGrid.RowDefinitions.Count < MAX_GRID_SIZE)
            {
                RowDefinition newGridRow = new RowDefinition();
                newGridRow.MinHeight = ROW_MIN_HEIGHT;
                dynamicGrid.RowDefinitions.Add(newGridRow);
                maxCellRowSpan = dynamicGrid.RowDefinitions.Count;
            }
        }

        private void RemoveRow_Click(object sender, RoutedEventArgs e)
        {
            if (dynamicGrid.RowDefinitions.Count > 1)
            {
                dynamicGrid.RowDefinitions.RemoveAt(dynamicGrid.RowDefinitions.GetLa
                maxCellRowSpan = dynamicGrid.RowDefinitions.Count;
            }
        }

        private void AddColumn_Click(object sender, RoutedEventArgs e)
        {
            if (dynamicGrid.ColumnDefinitions.Count < MAX_GRID_SIZE)
```

```
            {
                ColumnDefinition newGridColumn = new ColumnDefinition();
                newGridColumn.MinWidth = COLUMN_MIN_WIDTH;
                dynamicGrid.ColumnDefinitions.Add(newGridColumn);
                maxCellColumnSpan = dynamicGrid.ColumnDefinitions.Count;
            }
        }

        private void RemoveColumn_Click(object sender, RoutedEventArgs
            e)
        {
            if(dynamicGrid.ColumnDefinitions.Count > 1)
            {
                dynamicGrid.ColumnDefinitions.RemoveAt(dynamicGrid.ColumnDefinitions
                maxCellColumnSpan = dynamicGrid.ColumnDefinitions.Count;
            }
        }

        private void AddChart_Click(object sender, RoutedEventArgs e)
        {
            CreateCase(true);
        }

        private void AddGUI_Click(object sender, RoutedEventArgs e)
        {
            CreateCase(false);
            btnAddGUI.IsEnabled = false;
        }

        private void CreateCase(bool isChart)
        {
            int[] emptyIndex = GetFirstEmptyCell();
            int x = emptyIndex[0];
            int y = emptyIndex[1];

            if (x < MAX_GRID_SIZE)
            {
                Grid cell = new Grid();
                Button btnRemove = CreateChartButton("GraphCloseStyle",
                    "./Images/close.png");
                Button btnMove = CreateChartButton("GraphButtonStyle",
                    "./Images/cursor-move.png");

                Button btnWidthPlus =
                    CreateChartButton("GraphButtonStyle",
                    "./Images/arrow-right.png");
                Button btnWidthMinus =
                    CreateChartButton("GraphButtonStyle",
                    "./Images/arrow-left.png");

                Button btnHeightPlus =
                    CreateChartButton("GraphButtonStyle",
                    "./Images/arrow-down.png");
                Button btnHeightMinus =
                    CreateChartButton("GraphButtonStyle",
```

```
            "./Images/arrow-up.png");

        Button btnChartSettings = null;
        Image imgUnityGUI = null;

        if (isChart)
        {
            btnChartSettings =
                CreateChartButton("GraphButtonStyle",
                "./Images/cog.png");
            cell.Tag = "Chart";
        }
        else
        {
            imgUnityGUI = new Image();
            imgUnityGUI.Source = new BitmapImage(new
                Uri("./Images/UnityLogo.png", UriKind.Relative));
            imgUnityGUI.Height = 75;
            cell.Tag = "GUI";
        }

        RowDefinition firstRow = new RowDefinition();
        firstRow.MinHeight = 30;
        firstRow.MaxHeight = 30;
        cell.RowDefinitions.Add(firstRow);
        CreateRows(cell, 4);
        CreateColumns(cell, 6);

        cell.VerticalAlignment = VerticalAlignment.Stretch;
        cell.HorizontalAlignment = HorizontalAlignment.Stretch;

        btnMove.PreviewMouseDown += ChartDragOn_MouseDown;
        btnMove.PreviewMouseUp += ChartDragOff_MouseUp;
        btnRemove.Click += RemoveChart_Click;

        btnWidthPlus.Click += ChartWidthUp_Click;
        btnWidthMinus.Click += ChartWidthDown_Click;
        btnHeightPlus.Click += ChartHeightUp_Click;
        btnHeightMinus.Click += ChartHeightDown_Click;

        if (isChart && btnChartSettings != null)
            btnChartSettings.Click += OpenChartSettings_Click;

        Grid.SetColumn(btnWidthPlus, 0);
        Grid.SetRow(btnWidthPlus, 0);

        Grid.SetColumn(btnWidthMinus, 1);
        Grid.SetRow(btnWidthMinus, 0);

        Grid.SetColumn(btnHeightPlus, 2);
        Grid.SetRow(btnHeightPlus, 0);

        Grid.SetColumn(btnHeightMinus, 3);
        Grid.SetRow(btnHeightMinus, 0);
```

```
                Grid.SetColumn(btnMove, 4);
                Grid.SetRow(btnMove, 0);

                Grid.SetColumn(btnRemove, 5);
                Grid.SetRow(btnRemove, 0);

                if (isChart)
                {
                    Grid.SetColumn(btnChartSettings, 0);
                    Grid.SetColumnSpan(btnChartSettings, 6);
                    Grid.SetRow(btnChartSettings, 1);
                    Grid.SetRowSpan(btnChartSettings, 4);
                }
                else
                {
                    Grid.SetColumn(imgUnityGUI, 0);
                    Grid.SetColumnSpan(imgUnityGUI, 6);
                    Grid.SetRow(imgUnityGUI, 1);
                    Grid.SetRowSpan(imgUnityGUI, 4);
                }

                cell.Children.Add(btnMove);
                cell.Children.Add(btnRemove);
                cell.Children.Add(btnWidthPlus);
                cell.Children.Add(btnWidthMinus);
                cell.Children.Add(btnHeightPlus);
                cell.Children.Add(btnHeightMinus);

                if (isChart)
                    cell.Children.Add(btnChartSettings);
                else
                    cell.Children.Add(imgUnityGUI);

                cell.Background = Brushes.White;

                Grid.SetColumn(cell, x);
                Grid.SetRow(cell, y);

                gridHasContent[x, y] = true;
                ChartData grpData;
                if (isChart)
                    grpData = new ChartData(x, y, 1, 1, new int[] { 0
                        }, DEFAULT_CHART_TYPE, DEFAULT_AXIS_X_VALUE,
                        DEFAULT_AXIS_Y_VALUE);
                else
                    grpData = new ChartData(x, y, 1, 1, new int[] { 0
                        }, (int)UIType.GUI);

                chartsDatas[x, y] = grpData;
                dynamicGrid.Children.Add(cell);
            }
        }

        private void CreateColumns(Grid cell, int quantity)
        {
```

```
            for (int i = 0; i < quantity; i++)
            {
                cell.ColumnDefinitions.Add(new ColumnDefinition());
            }
        }

        private void CreateRows(Grid cell, int quantity)
        {
            for (int i = 0; i < quantity; i++)
            {
                cell.RowDefinitions.Add(new RowDefinition());
            }
        }

        private Button CreateChartButton(string style, string
            imageSource)
        {
            Button btn = new Button();
            btn.Style = this.FindResource(style) as Style;
            Image img = new Image();
            img.Source = new BitmapImage(new Uri(imageSource,
                UriKind.Relative));
            img.Height = 30;
            btn.Content = img;
            btn.VerticalAlignment = VerticalAlignment.Stretch;
            btn.HorizontalAlignment = HorizontalAlignment.Stretch;
            return btn;
        }

        private void RemoveChart_Click(object sender, RoutedEventArgs e)
        {
            Button btn = (Button)sender;
            Grid cell = VisualTreeHelper.GetParent(btn) as Grid;

            int x = Grid.GetColumn(cell), y = Grid.GetRow(cell);
            int columnSpan = Grid.GetColumnSpan(cell), rowSpan =
                Grid.GetRowSpan(cell);
            SetCellsContent(x, y, x + columnSpan, y + rowSpan, false);
            chartsDatas[x, y].SetAsNull();
            dynamicGrid.Children.Remove(cell);

            if ((string)cell.Tag == "GUI")
                btnAddGUI.IsEnabled = true;
        }

        private void ChartDragOn_MouseDown(object sender,
            MouseButtonEventArgs e)
        {
            Button btn = (Button)sender;
            Grid cell = VisualTreeHelper.GetParent(btn) as Grid;
            oldX = Grid.GetColumn(cell);
            oldY = Grid.GetRow(cell);
            int columnSpan = Grid.GetColumnSpan(cell), rowSpan =
                Grid.GetRowSpan(cell);
            SetCellsContent(oldX, oldY, oldX + columnSpan, oldY +
```

```
            rowSpan, false); // Libère l'espace anciennement occupé
        }

        private void ChartDragOff_MouseUp(object sender,
            MouseButtonEventArgs e)
        {
            Button btn = (Button)sender;
            Grid cell = VisualTreeHelper.GetParent(btn) as Grid;
            int[] newCoordinates =
                GetCoordinateWithSize(Mouse.GetPosition(dynamicGrid).X,
                Mouse.GetPosition(dynamicGrid).Y);

            //if (!gridHasContent[newCoordinates[0],
                newCoordinates[1]]) // ChechIfCellsEmpty
            int x = newCoordinates[0], y = newCoordinates[1];
            int columnSpan = Grid.GetColumnSpan(cell), rowSpan =
                Grid.GetRowSpan(cell);
            if (ChechIfCellsEmpty(x, y, x + columnSpan, y + rowSpan))
            {
                // repositionne l'élément
                Grid.SetColumn(cell, newCoordinates[0]);
                Grid.SetRow(cell, newCoordinates[1]);
                SetCellsContent(x, y, x + columnSpan, y + rowSpan,
                    true); // Bloque le nouvel espace occupé
                chartsDatas[x, y] = chartsDatas[oldX,
                    oldY].CloneInNewLocation(x,y);
                chartsDatas[oldX, oldY].SetAsNull();
            }
            else
            {
                SetCellsContent(oldX, oldY, oldX + columnSpan, oldY +
                    rowSpan, true); // Aucun nouvel emplacement trouvé,
                    on rebloque l'ancienne espace liberé
            }
            oldX = MAX_GRID_SIZE + 1;
            oldY = MAX_GRID_SIZE + 1;
        }

        private void ChartWidthUp_Click(object sender, RoutedEventArgs
            e)
        {
            Button btn = (Button)sender;
            Grid cell = VisualTreeHelper.GetParent(btn) as Grid;

            int x = Grid.GetColumn(cell);
            int y = Grid.GetRow(cell);
            int columnSpan = Grid.GetColumnSpan(cell);
            int rowSpan = Grid.GetRowSpan(cell);

            if (ChechIfCellsEmpty(x + columnSpan, y, x + columnSpan +
                1, y + rowSpan) && columnSpan < maxCellColumnSpan)
            {
                Grid.SetColumnSpan(cell, columnSpan + 1);
                SetCellsContent(x, y, x + columnSpan + 1, y + rowSpan,
                    true);
```

```
            chartsDatas[x, y].SpanX++;
        }
    }

    private void ChartWidthDown_Click(object sender,
        RoutedEventArgs e)
    {

        Button btn = (Button)sender;
        Grid cell = VisualTreeHelper.GetParent(btn) as Grid;

        int x = Grid.GetColumn(cell);
        int y = Grid.GetRow(cell);
        int columnSpan = Grid.GetColumnSpan(cell);
        int rowSpan = Grid.GetRowSpan(cell);
        if (Grid.GetColumnSpan(cell) > 1)
        {
            Grid.SetColumnSpan(cell, columnSpan - 1);
            SetCellsContent(x + 1, y, x + columnSpan + 1, y +
                rowSpan, false);
            chartsDatas[x, y].SpanX--;
        }
    }

    private void ChartHeightUp_Click(object sender, RoutedEventArgs
        e)
    {
        Button btn = (Button)sender;
        Grid cell = VisualTreeHelper.GetParent(btn) as Grid;

        int x = Grid.GetColumn(cell);
        int y = Grid.GetRow(cell);
        int columnSpan = Grid.GetColumnSpan(cell);
        int rowSpan = Grid.GetRowSpan(cell);

        if (ChechIfCellsEmpty(x, y + rowSpan, x + columnSpan, y +
            rowSpan + 1) && rowSpan < maxCellRowSpan)
        {
            Grid.SetRowSpan(cell, rowSpan + 1);
            SetCellsContent(x, y, x + columnSpan, y + rowSpan + 1,
                true);

            chartsDatas[x, y].SpanY++;
        }
    }

    private void ChartHeightDown_Click(object sender,
        RoutedEventArgs e)
    {

        Button btn = (Button)sender;
        Grid cell = VisualTreeHelper.GetParent(btn) as Grid;

        int x = Grid.GetColumn(cell);
        int y = Grid.GetRow(cell);
```

```
            int columnSpan = Grid.GetColumnSpan(cell);
            int rowSpan = Grid.GetRowSpan(cell);

            if (Grid.GetRowSpan(cell) > 1)
            {
                Grid.SetRowSpan(cell, rowSpan - 1);
                SetCellsContent(x, y + 1, x + columnSpan, y + rowSpan +
                    1, false);
                chartsDatas[x, y].SpanY--;
            }
        }

        private void OpenChartSettings_Click(object sender,
            RoutedEventArgs e)
        {
            int chartCellX;
            int chartCellY;
            int sizeX;
            int sizeY;
            Button btn = (Button)sender;
            Grid cell = VisualTreeHelper.GetParent(btn) as Grid;
            chartCellX = Grid.GetColumn(cell);
            chartCellY = Grid.GetRow(cell);
            sizeX = Grid.GetColumnSpan(cell);
            sizeY = Grid.GetRowSpan(cell);
            WindowChart chartWindow = new WindowChart(chartCellX,
                chartCellY, sizeX, sizeY, chartsDatas[chartCellX,
                chartCellY]);
            chartWindow.OnSave += new SaveEventHandler(OnSave);
            chartWindow.Show();
        }

        static void OnSave(object source, ChartData e)
        {
            chartsDatas[e.X, e.Y] = e;
        }

        public ChartData[,] GetChartsData()
        {
            return chartsDatas;
        }

        public Grid GetGrid()
        {
            Grid result = new Grid();
            result.VerticalAlignment = VerticalAlignment.Stretch;
            result.HorizontalAlignment = HorizontalAlignment.Stretch;

            for (int i = 0; i < grdContent.ColumnDefinitions.Count; i++)
            {
                ColumnDefinition newGridColumn = new ColumnDefinition();
                newGridColumn.MinWidth = COLUMN_MIN_WIDTH;
                result.ColumnDefinitions.Add(newGridColumn);
            }
```

```
            for (int i = 0; i < grdContent.RowDefinitions.Count; i++)
            {
                RowDefinition newGridRow = new RowDefinition();
                newGridRow.MinHeight = ROW_MIN_HEIGHT;
                result.RowDefinitions.Add(newGridRow);
            }
            return result;
        }

        private void SetCellsContent(int xStart, int yStart, int xStop,
            int yStop, bool hasContent)
        {
            for (int y = yStart; y < yStop; y++)
            {
                for (int x = xStart; x < xStop; x++)
                {
                    if (x < gridHasContent.GetLength(0) && y <
                        gridHasContent.GetLength(1))
                        gridHasContent[x, y] = hasContent;
                }
            }
        }

        private bool ChechIfCellsEmpty(int xStart, int yStart, int
            xStop, int yStop)
        {
            // check si entre x, y et x+width, y+height est vide
            bool result = true;
            for (int y = yStart; y < yStop; y++)
            {
                for (int x = xStart; x < xStop; x++)
                {
                    if (x < gridHasContent.GetLength(0) && y <
                        gridHasContent.GetLength(1) && gridHasContent[x,
                        y])
                    {
                        result = false;
                        y = yStop;
                        break;
                    }
                }
            }
            return result;
        }

        private int[] GetCoordinateWithSize(double x, double y)
        {
            int cellCountX = 1;
            int cellCountY = 1;

            double cellWidth = dynamicGrid.ActualWidth /
                dynamicGrid.ColumnDefinitions.Count;
            double cellHeight = dynamicGrid.ActualHeight /
                dynamicGrid.RowDefinitions.Count;
```

```
            while (x > cellWidth * cellCountX)
                cellCountX++;

            while (y > cellHeight * cellCountY)
                cellCountY++;

            return new int[] { cellCountX-1, cellCountY-1 };
        }

        private int[] GetFirstEmptyCell()
        {
            int[] indexes = new int[] { MAX_GRID_SIZE + 1,
               MAX_GRID_SIZE + 1 };

            for (int y = 0; y < dynamicGrid.RowDefinitions.Count; y++)
            {
                for (int x = 0; x <
                   dynamicGrid.ColumnDefinitions.Count; x++)
                {
                    if (!gridHasContent[x, y])
                    {
                        indexes[0] = x;
                        indexes[1] = y;
                        y = dynamicGrid.RowDefinitions.Count;
                        break;
                    }
                }
            }
            return indexes;
        }

    }
}
```

Listing 8 – Code source de ChartData.cs

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste représentant une ville.
 */
using System;

namespace CovidPropagation
{
    public struct ChartData
    {
        public ChartData(int x, int y, int sizeX, int sizeY, int[]
           datas, int graphicType = 0, int valueX = 0, int valueY = 0,
           int displayInterval = 0, bool autoDisplay = true, int
           displayWindow = 0)
        {
```

```
            X = x;
            Y = y;
            SpanX = sizeX;
            SpanY = sizeY;
            Datas = datas;
            UIType = graphicType;
            AxisX = valueX;
            AxisY = valueY;
            DisplayInterval = displayInterval;
            AutoDisplay = autoDisplay;
            DisplayWindow = displayWindow;
        }

        public int X { get; set; }
        public int Y { get; set; }
        public int SpanX { get; set; }
        public int SpanY { get; set; }
        public int UIType { get; set; }
        public int AxisX { get; set; }
        public int AxisY { get; set; }
        public int DisplayInterval { get; set; }
        public int[] Datas { get; set; }
        public bool AutoDisplay { get; set; }
        public int DisplayWindow { get; set; }

        public override string ToString()
        {
            string result = $"X: {X}{Environment.NewLine}" +
                    $"Y: {Y}{Environment.NewLine}" +
                    $"SizeX: {SpanX}{Environment.NewLine}" +
                    $"SizeY: {SpanY}{Environment.NewLine}" +
                    $"GraphicType: {UIType}{Environment.NewLine}" +
                    $"ValueX: {AxisX}{Environment.NewLine}" +
                    $"ValueY: {AxisY}{Environment.NewLine}" +
                    $"UpdateInterval:
                       {DisplayInterval}{Environment.NewLine}" +
                    $"Datas: {Environment.NewLine}";
            foreach (int item in Datas)
            {
                result += $"{item} {Environment.NewLine}";
            }
            return result;
        }

        public void SetAsNull()
        {
            X = 0;
            Y = 0;
            SpanX = 0;
            SpanY = 0;
            UIType = 0;
            AxisX = 0;
            AxisY = 0;
            Datas = new int[0];
            AutoDisplay = true;
```

```
            DisplayWindow = 0;
        }

        public ChartData CloneInNewLocation(int newX, int newY)
        {
            return new ChartData(newX, newY, SpanX, SpanY, Datas,
                UIType, AxisX, AxisY, DisplayInterval, AutoDisplay,
                DisplayWindow);
        }
    }
}
```

Listing 9 – Code source de MainWindow.cs

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste représentant une ville.
 */
using System.Windows;
using System.Windows.Navigation;

namespace CovidPropagation
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        PageSimulation pageSimulation;
        PageSimulationSettings pageSimulationSettings;
        PageGraphicSettings pageGraphicSettings;

        public MainWindow()
        {
            InitializeComponent();

            SimulationGeneralParameters.Init();
            VirusParameters.Init();
            MaskParameters.Init();
            VaccinationParameters.Init();
            QuarantineParameters.Init();

            pageSimulation = new PageSimulation();
            pageSimulationSettings = new PageSimulationSettings();
            pageGraphicSettings = new PageGraphicSettings();
            MainContent.NavigationUIVisibility =
                NavigationUIVisibility.Hidden; // Cache la barre de
                navigation du contenu
        }

        private void SimulationPage_Click(object sender,
```

```
            RoutedEventArgs e)
        {
            pageSimulation.SetGrid(pageGraphicSettings.GetGrid(),
                pageGraphicSettings.GetChartsData());
            MainContent.Navigate(pageSimulation);
        }

        private void GraphicSettingsPage_Click(object sender,
            RoutedEventArgs e)
        {
            MainContent.Navigate(pageGraphicSettings);
        }

        private void SettingsPage_Click(object sender, RoutedEventArgs
            e)
        {
            MainContent.Navigate(pageSimulationSettings);
        }
    }
}
```

Listing 10 – Code source de F*Simulation/StreamString.cs*

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste représentant une ville.
 */
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace CovidPropagation
{
    public class StreamString
    {
        private BinaryWriter stream;
        private UnicodeEncoding streamEncoding;

        public StreamString(Stream stream)
        {
            this.stream = new BinaryWriter(stream);
            streamEncoding = new UnicodeEncoding();
        }

        /// <summary>
        /// ID Documentation : Write_String
        /// Écrit en byte le message fournit en string, insère sa
```

```
            taille dans les 4 premiers bytes et envoies les données.
        /// </summary>
        /// <param name="outString"></param>
        public async void WriteString(string outString)
        {
            await Task.Run(() => {
                byte[] outBuffer = streamEncoding.GetBytes(outString);
                int len = outBuffer.Length;

                List<byte> dataToSend = new List<byte>();
                dataToSend.Add((byte)(len >> 24));
                dataToSend.Add((byte)(len >> 16));
                dataToSend.Add((byte)(len >> 8));
                dataToSend.Add((byte)(len >> 0));
                dataToSend.AddRange(outBuffer.ToList());
                try
                {
                    stream.Write(dataToSend.ToArray(), 0,
                        dataToSend.Count);
                    stream.Flush();
                }
                catch (Exception ex)
                {
                    Debug.WriteLine("Pipeline is borken " + ex);
                }
            });
        }

        public void CloseLink()
        {
            if (stream != null)
            {
                stream.Close();
            }

        }
    }
}
```

Listing 11 – Code source de F*Simulation/ExtensionMethods.cs*

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste représentant une ville.
 */
using LiveCharts;
using LiveCharts.Defaults;
using LiveCharts.Geared;
using LiveCharts.Helpers;
using LiveCharts.Wpf;
using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Windows;

namespace CovidPropagation
{
    static class ExtensionMethods
    {
        /// <summary>
        /// Convertis un booléen en int.
        /// </summary>
        /// <returns>1 ou 0 représentat la valeur du bool.</returns>
        public static int ConvertToInt(this bool value)
        {
            return value ? 1 : 0;
        }

        /// <summary>
        /// Choisis aléatoirement (50/50) si la valeur retournée est
           true ou false.
        /// </summary>
        /// <returns>Résultat du flipCoing</returns>
        public static bool NextBoolean(this Random value)
        {
            return Convert.ToBoolean(value.Next(0, 2));
        }

        /// <summary>
        /// Choisis aléatoirement une valeur qui résultera en
           true/false. Le true à un poid définit.
        /// </summary>
        /// <param name="trueWeight">Poid du true (format double 0 à
           1)</param>
        /// <returns></returns>
        public static bool NextBoolean(this Random value, double
           trueWeight)
        {
            bool result = false;
            if (value.NextDouble() < trueWeight)
                result = true;

            return result;
        }

        /// <summary>
        /// Choisi en fonction des probabilité données un objet de du
           tableau et retourne celui sélectionné. Chaque objet à un
           poid qui défini les chances qu'il a d'être sélectionné.
        /// La somme des poids ne peut être plus grande ou plus petite
           que 1
        /// </summary>
        /// <param name="weight">Poid de l'objet</param>
        /// <returns>L'objet qui a été choisis</returns>
        public static object NextProbability(this Random value,
           KeyValuePair<object, double>[] weight)
```

```
        {
            double weightSum = weight.Sum(w => w.Value);
            if (weightSum > 1 && weightSum < 1)
                throw new ArgumentException("Weight array sum must be
                    equal to 1.");

            object result = 0;
            double rdm = value.NextDouble();
            double sumPreviousWeight = 0;
            weight = weight.OrderBy(w => w.Value).ToArray();
            for (int i = 0; i < weight.Length; i++)
            {
                sumPreviousWeight += weight[i].Value;
                if (rdm < sumPreviousWeight)
                {
                    result = weight[i].Key;
                    break;
                }
            }
            return result;
        }

        /// <summary>
        /// Si le résultat obtenu est trop proche du maximum, la valeur
            resultante est modifiée pour obtenir un résultat suffisant.
        /// Inclusif
        /// </summary>
        /// <param name="minValue">Valeurs minimum requise entre le
            résultat du random et le maximum possible</param>
        /// <returns></returns>
        public static int NextWithMinimum(this Random value, int
            minRdm, int maxRdm, int minValue)
        {
            int result = value.Next(minRdm, maxRdm + 1);

            if (result > (maxRdm - minValue) && result != maxRdm)
                result = maxRdm - minValue;

            return result;
        }

        /// <summary>
        /// Identique à Random.Next() mais la valeur la plus grande est
            incluse.
        /// </summary>
        /// <param name="minRdm">Valeur minimum à tirer au sort.</param>
        /// <param name="maxRdm">Valeur maximum à tirer au sort.</param>
        /// <returns></returns>
        public static int NextInclusive(this Random value, int minRdm,
            int maxRdm)
        {
            return value.Next(minRdm, maxRdm + 1);
        }

        /// <summary>
```

```
        /// Identique à Random.NextDouble() mais avec un minimum et un
            maximum.
        /// </summary>
        /// <param name="minRdm">Valeur minimum à tirer au sort.</param>
        /// <param name="maxRdm">Valeur maximum à tirer au sort.</param>
        /// <returns></returns>
        public static double NextDoubleInclusive(this Random value,
            double minRdm, double maxRdm)
        {
            return value.NextDouble() * (maxRdm - minRdm) + minRdm;
        }


        /// <summary>
        /// Permet de mélanger une liste de tout type.
        /// </summary>
        public static void Shuffle<T>(this IList<T> ts)
        {
            var count = ts.Count;
            var last = count - 1;
            for (var i = 0; i < last; ++i)
            {
                var r = GlobalVariables.rdm.Next(i, count);
                var tmp = ts[i];
                ts[i] = ts[r];
                ts[r] = tmp;
            }
        }

        /// <summary>
        /// Convertit un caractère string en char. Si la string est
            plus grande qu'un caractère, seul le premier est récupéré.
        /// </summary>
        /// <returns>Premier caractère de la string en char</returns>
        public static char ToChar(this string value)
        {
            return value.ToCharArray()[0];
        }

        /// <summary>
        /// Récupère le dernier index d'une liste.
        /// Permet d'éviter d'écrire List.Count - 1 lors d'un for ou de
            récupérer le dernier élément d'une liste.
        /// </summary>
        /// <returns>Dernier index de la liste.</returns>
        public static int GetLastIndex<T>(this IEnumerable<T> value)
        {
            int result = 0;
            if (value != null)
                result = value.Count() - 1;

            return result;
        }

        /// <summary>
```

```
        /// Si la valeur d'un double vaut NaN, alors le transform en 0.
        /// </summary>
        /// <returns>0 ou la valeur du double si autre que NaN</returns>
        public static double SetValueIfNaN(this double value)
        {
            return double.IsNaN(value) ? 0 : value;
        }

        #region Charts

        /// <summary>
        /// Lorsque les données d'un graphique cylindrique sont mise à
            jours.
        /// Récupère la dernière données de chaque section de la
            simulation et l'affiche.
        /// </summary>
        public static void OnDataUpdatePieChart(this PieChart chart,
            SimulationDatas e)
        {
            Application.Current.Dispatcher.Invoke((Action)(() =>
            {
                ChartValues<double> cv;
                foreach (PieSeries serie in chart.Series)
                {
                    cv = new ChartValues<double>();
                    cv.Add(e.GetDataFromEnum((ChartsDisplayData)serie.Tag).Last());
                    serie.Values = cv;
                }
            }));
        }

        private static void SetMaxAxisValue(Axis axis, double
            axisMaxValue)
        {
            if (axisMaxValue <= 5)
                axis.MaxValue = 5;
            else
                axis.MaxValue = double.NaN;
        }

        /// <summary>
        /// Affiche les données des graphiques dans le format des
            différents graphiques.
        /// Affiche les données suivant l'interval de temps donné.
        /// Pour les graphiques à colonne et en ligne, les données sont
            réduites.
        /// Par exemple, pour afficher une semaine, à la palce
            d'afficher 48*7 colonnes, 7 colonnes sont affichées
            correspondant aux jours.
        /// </summary>
        /// <param name="chart">Graphique à modifier.</param>
        /// <param name="e">Valeurs de la simulation.</param>
        /// <param name="isDisplayChange">Si l'affichage change et
            qu'il est nécessaire de recharger les données du
            graphiques.</param>
```

```
public static void Display(this CartesianChart chart,
    SimulationDatas e, bool isDisplayChange)
{
     Application.Current.Dispatcher.Invoke((Action)(() =>
     {
          int interval = 48;
          Axis axisX = chart.AxisX[0];
          Axis axisY = chart.AxisY[0];
          Axis currentAxis = axisX;
          double maxValue = chart.Series[0].Values.Count;
          ChartData datas = (ChartData)chart.Tag;
          int displayWindow = datas.DisplayWindow;

          switch ((UIType)datas.UIType)
          {
               case UIType.Linear:
                    axisX.MaxValue = maxValue;

                    // Récupère l'interval
                    switch
                       ((ChartsDisplayInterval)datas.DisplayInterval)
                    {
                         default:
                         case ChartsDisplayInterval.Day:
                              interval = 48;
                              break;
                         case ChartsDisplayInterval.Week:
                              interval = 336;
                              break;
                         case ChartsDisplayInterval.Month:
                              interval = 1440;
                              break;
                         case ChartsDisplayInterval.Total:
                              interval = 0;
                              break;
                    }

                    double axisYMaxValue = 0;
                    // ID Documentation : Curve_Chart
                    foreach (LineSeries serie in chart.Series)
                    {
                         List<double> lineSerieDatas = new
                            List<double>(e.GetDataFromEnum((ChartsDisplayData)ser

                         // Si l'affichage n'est pas "total" et que
                            lineSerieDatas a suffisament de données
                         if (interval != 0 &&
                            lineSerieDatas.GetLastIndex() > interval)
                         {
                              // Permet de garder le focus sur la
                                 dernière fenêtre d'informations
                              if (datas.AutoDisplay)
                                   displayWindow =
                                       lineSerieDatas.GetLastIndex() /
                                       interval;
```

```
                // Permet d'éviter d'incrémenter
                   displayWindow au delà de la quantité
                   d'interval actuellement disponible.
                if (displayWindow >
                   lineSerieDatas.GetLastIndex() /
                   interval)
                    displayWindow =
                       lineSerieDatas.GetLastIndex() /
                       interval;

                // Différencie le premier jour du reste.
                if ((interval * displayWindow +
                   interval) <
                   lineSerieDatas.GetLastIndex())
                {
                    int lastItemIndex = interval *
                       displayWindow + interval;
                    lineSerieDatas.RemoveRange(lastItemIndex,
                       lineSerieDatas.GetLastIndex() -
                       lastItemIndex + 1);
                    lineSerieDatas.RemoveRange(0,
                       interval * displayWindow);
                }
                else
                {
                    lineSerieDatas.RemoveRange(0,
                       lineSerieDatas.GetLastIndex() -
                       interval);
                }
            }
            double serieMaxValue = lineSerieDatas.Max();
            if (serieMaxValue > axisYMaxValue)
            {
                axisYMaxValue = serieMaxValue;
            }
            serie.Values =
               lineSerieDatas.AsGearedValues().WithQuality(Quality.
        }
        SetMaxAxisValue(axisY, axisYMaxValue);

        // ID Documentation : Curve_Chart_AxeX
        if (maxValue - interval > 0 && interval != 0)
        {
            axisX.MinValue = maxValue - interval;
            axisX.MaxValue = maxValue;
        }
        else if (interval == 0)
        {
            axisX.MinValue = 0;
            axisX.MaxValue = maxValue;
        }
        else
        {
            axisX.MinValue = 0;
```

```
                    axisX.MaxValue = interval;
                }

                datas.DisplayWindow = displayWindow;
                chart.Tag = datas;
                break;
            case UIType.Vertical:
                //SetMaxAxisValue(axisY);
                DisplayColumnRowChart(chart, e, axisX, axisY,
                    isDisplayChange);
                break;
            case UIType.Horizontal:
                //SetMaxAxisValue(axisX);
                DisplayColumnRowChart(chart, e, axisY, axisX,
                    isDisplayChange);
                break;
            case UIType.HeatMap:
                HeatSeries serieHeatMap =
                    (HeatSeries)chart.Series[0];
                List<double> simulationData = new
                    List<double>(e.GetDataFromEnum((ChartsDisplayData)serieHe

                if (simulationData != null)
                {
                    // Trouver premier jour semaine
                    int x = 0;
                    int y = 0;
                    int week = 0;
                    double value = 0;
                    if (simulationData.Count > 0)
                    {
                        // Parcours les données reçues
                        for (int i = 1; i <=
                            simulationData.Count; i++)
                        {
                            // Ajotue les données à value
                            value += simulationData[i - 1];
                            // Si il y a de nouvelles valeurs
                            if (serieHeatMap.Values.Count <
                                (i/4))
                            {
                                // Si value a pu récupérer 4
                                    valeurs
                                if (i % 4 == 0)
                                {
                                    // Si c'est la première
                                        semaine, créé des
                                        heatpoint
                                    if (week == 0)
                                    {
                                        double avgValue = value
                                            / 4;
                                        serieHeatMap.Values.Add(new
                                            HeatPoint(x, y,
                                            avgValue));
```

```
                                        }
                                        else // Sinon, les ajoutent
                                            au heatpoint déjà
                                            existant et en fait la
                                            moyenne
                                        {
                                            HeatPoint hm =
                                                ((HeatPoint)serieHeatMap.Values[(
                                                * 12) + y]);
                                            hm.Weight = ((week * 4)
                                                * hm.Weight + value)
                                                / ((week * 4) + 4);
                                        }
                                    }
                                }
                                // Lors d'un changement de semaine.
                                if (i % 336 == 0)
                                {
                                    week++;
                                    x = 0;
                                    y = 0;
                                    value = 0;
                                }
                                else if (i % 48 == 0) // Lors d'un
                                    changement de jour.
                                {
                                    x++;
                                    y = 0;
                                    value = 0;
                                }
                                else if (i % 4 == 0) // Lors d'un
                                    changement de case.
                                {
                                    y++;
                                    value = 0;
                                }
                            }
                        }
                    }
                    axisX.MaxValue = 7;
                    axisY.MaxValue = 12;
                    break;
                }
            }));
        }

        private static void DisplayColumnRowChart(CartesianChart chart,
            SimulationDatas e, Axis axis, Axis axisValueSize, bool
            isDisplayChange)
        {
            int interval;
            ChartValues<double> cv;
            ChartData datas = (ChartData)chart.Tag;
            double maxValue = chart.Series[0].Values.Count;
            switch ((ChartsDisplayInterval)datas.DisplayInterval)
```

```
            {
                default:
                case ChartsDisplayInterval.Day:
                    interval = 12;
                    DisplayColumnRow(chart, e, 4, isDisplayChange, 48);
                    maxValue = 12;
                    break;
                case ChartsDisplayInterval.Week:
                    interval = 7;
                    DisplayColumnRow(chart, e, 48, isDisplayChange,
                        336);
                    maxValue = interval;
                    break;
                case ChartsDisplayInterval.Month:
                    interval = 4;
                    DisplayColumnRow(chart, e, 336, isDisplayChange,
                        1440);
                    maxValue = interval;
                    break;
                case ChartsDisplayInterval.Total:
                    foreach (Series serie in chart.Series)
                    {
                        cv = new ChartValues<double>();
                        List<double> day = new List<double>();
                        List<double> avg = new List<double>();
                        List<double> columnDatas =
                            e.GetDataFromEnum((ChartsDisplayData)serie.Tag);
                        avg.Add(columnDatas.Average());
                        cv.AddRange(avg);
                        serie.Values = cv;
                    }
                    interval = 1;
                    break;
            }

            double axisMaxValue = 0;
            foreach (Series serie in chart.Series)
            {
                double serieMaxValue =
                    e.GetDataFromEnum((ChartsDisplayData)serie.Tag).Max();
                if (serieMaxValue > axisMaxValue)
                {
                    axisMaxValue = serieMaxValue;
                }
            }

            SetMaxAxisValue(axisValueSize, axisMaxValue);

            if (maxValue - interval > 0 && interval != 0)
            {
                axis.MinValue = maxValue - interval;
                axis.MaxValue = maxValue;
            }
            else
            {
```

```
            axis.MinValue = 0;
            axis.MaxValue = interval;
        }
    }

    /// <summary>
    /// ID Documentation : ColumnRow_Chart
    /// </summary>
    /// <param name="chart">Graphique à modifier.</param>
    /// <param name="e">Valeurs de la simulation.</param>
    /// <param name="modulo">Nombre par lequel les données seront
        regroupée. (Ex: Modulo = 4 pour 12 données alors il y aura 3
        colonne/ligne d'ajoutées.)</param>
    /// <param name="isDisplayChange">Si l'affichage change et
        qu'il est nécessaire de recharger les données du
        graphiques.</param>
    /// <param name="interval">Interval à afficher.</param>
    private static void DisplayColumnRow(CartesianChart chart,
        SimulationDatas e, int modulo, bool isDisplayChange, int
        interval)
    {
        ChartValues<double> cv;
        ChartData datas = (ChartData)chart.Tag;
        foreach (Series serie in chart.Series)
        {
            cv = new ChartValues<double>();
            List<double> columnDatas = new List<double>();
            List<double> columnAvg = new List<double>();
            List<double> colRowSerieDatas = new
                List<double>(e.GetDataFromEnum((ChartsDisplayData)serie.Tag));

            if (colRowSerieDatas.GetLastIndex() > interval &&
                datas.AutoDisplay)
            {
                colRowSerieDatas.RemoveRange(0,
                    colRowSerieDatas.GetLastIndex() - interval);
            }
            else if ((interval * datas.DisplayWindow + interval) <
                colRowSerieDatas.GetLastIndex())
            {
                int lastItemIndex = interval * datas.DisplayWindow
                    + interval;
                colRowSerieDatas.RemoveRange(lastItemIndex,
                    colRowSerieDatas.GetLastIndex() - lastItemIndex
                    + 1);
                colRowSerieDatas.RemoveRange(0, interval *
                    datas.DisplayWindow);
            }

            // parcours les données et créé des moyennes pour
                correspondre au nombre de ligne/colonne de
                l'interval.
            //      Par exemple: 48 données pour 1 jour = 12
                colonnes/lignes, 48 données pour 1 semaine = 1
                colonne/ligne.
```

```
                for (int i = 0; i < colRowSerieDatas.Count; i++)
                {
                    columnDatas.Add(colRowSerieDatas[i]);
                    if (i % modulo == 0)
                    {
                        columnAvg.Add(columnDatas.Average());
                        columnDatas.Clear();
                    }
                }
                serie.Values = columnAvg.AsChartValues();
            }
        }

        #endregion
    }
}
```

Listing 12 – Code source de F*Simulation/FParameters/SimulationGeneralParameters.cs*

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste représentant une ville.
 */

namespace CovidPropagation
{
    public static class SimulationGeneralParameters
    {
        private const int DEFAULT_NUMBER_OF_PEOPLE = 100000;
        private const float DEFAULT_PROBABILITY_OF_BEING_INFECTED =
            0.01f;

        private const int DEFAULT_MIN_TRIGGER = 1000;
        public static void Init()
        {
            NbPeople = DEFAULT_NUMBER_OF_PEOPLE;
            ProbabilityOfInfected =
                DEFAULT_PROBABILITY_OF_BEING_INFECTED;

            IsMaskMeasuresEnabled = false;
            IsDistanciationMeasuresEnabled = false;
            IsQuarantineMeasuresEnabled = false;
            IsVaccinationMeasuresEnabled = false;

            NbInfecetdForMaskActivation = DEFAULT_MIN_TRIGGER;
            NbInfecetdForMaskDeactivation = DEFAULT_MIN_TRIGGER;
            NbInfecetdForDistanciationActivation = DEFAULT_MIN_TRIGGER;
            NbInfecetdForDistanciationDeactivation =
                DEFAULT_MIN_TRIGGER;
            NbInfecetdForQuarantineActivation = DEFAULT_MIN_TRIGGER;
            NbInfecetdForQuarantineDeactivation = DEFAULT_MIN_TRIGGER;
```

```
            NbInfecetdForVaccinationActivation = DEFAULT_MIN_TRIGGER;
            NbInfecetdForVaccinationDeactivation = DEFAULT_MIN_TRIGGER;
        }
        // Paramètres généraux
        public static int NbPeople { get; set; }
        public static float ProbabilityOfInfected { get; set; }

        // Si les mesures son activées
        public static bool IsMaskMeasuresEnabled { get; set; }
        public static bool IsDistanciationMeasuresEnabled { get; set; }
        public static bool IsQuarantineMeasuresEnabled { get; set; }
        public static bool IsVaccinationMeasuresEnabled { get; set; }

        // Trigger des mesures
        public static int NbInfecetdForMaskActivation { get; set; }
        public static int NbInfecetdForMaskDeactivation { get; set; }
        public static int NbInfecetdForDistanciationActivation { get;
            set; }
        public static int NbInfecetdForDistanciationDeactivation { get;
            set; }
        public static int NbInfecetdForQuarantineActivation { get; set;
            }
        public static int NbInfecetdForQuarantineDeactivation { get;
            set; }
        public static int NbInfecetdForVaccinationActivation { get;
            set; }
        public static int NbInfecetdForVaccinationDeactivation { get;
            set; }
    }
}
```

Listing 13 – Code source de $\mathsf{F}_{Simulation/FParameters/MaskParameters.cs}$

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste représentant une ville.
 */

namespace CovidPropagation
{
    /// <summary>
    /// Contient les valeurs des paramètres du port du masque.
    /// </summary>
    class MaskParameters
    {
        public static void Init()
        {
            IsClientMaskOn = false;
            IsPersonnelMaskOn = false;
        }
        // Paramètres généraux
```

```
        public static bool IsClientMaskOn { get; set; }
        public static bool IsPersonnelMaskOn { get; set; }
    }
}
```

Listing 14 – Code source de F$_{S}imulation/F_{P}arameters/VirusParameters.cs$

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste représentant une ville.
 */

using System;
using System.Xml;

namespace CovidPropagation
{
    /// <summary>
    /// Classe contenant les paramètres du virus.
    /// </summary>
    public static class VirusParameters
    {
        private const int DEFAULT_MIN_QUANTA = 100;
        private const int DEFAULT_MAX_QUANTA = 200;
        private const string INCUBATION_DURATION_MIN_NAME =
            "IncubationDurationMin";
        private const string INCUBATION_DURATION_MAX_NAME =
            "IncubationDurationMax";

        private const string DURATION_MIN_NAME = "DurationMin";
        private const string DURATION_MAX_NAME = "DurationMax";

        private const string DECAY_RATE_OF_VIRUS_NAME =
            "DecayRateOfVirus";
        private const string DEPOSITION_ON_SURFACE_RATE_NAME =
            "DepositionOnSurfaceRate";


        private const string IMMUNITY_DURATION_MIN_NAME =
            "ImmunityDurationMin";
        private const string IMMUNITY_DURATION_MAX_NAME =
            "ImmunityDurationMax";
        private const string IMMUNITY_EFFICIENCY_NAME =
            "ImmunityEfficiency";

        /// <summary>
        /// Initialise les variables de la class en allant chercher les
            valeurs dans un fichier XML.
        /// </summary>
        public static void Init()
        {
```

```
            XmlDocument xmlDatas = new XmlDocument();
            xmlDatas.Load("./CovidData.xml");

            IncubationDurationMin =
                Convert.ToInt32(xmlDatas.GetElementsByTagName(INCUBATION_DURATION_MIN
            IncubationDurationMax =
                Convert.ToInt32(xmlDatas.GetElementsByTagName(INCUBATION_DURATION_MAX
            DurationMin =
                Convert.ToInt32(xmlDatas.GetElementsByTagName(DURATION_MIN_NAME)[0].I
            DurationMax =
                Convert.ToInt32(xmlDatas.GetElementsByTagName(DURATION_MAX_NAME)[0].I
            DecayRateOfVirus =
                Convert.ToDouble(xmlDatas.GetElementsByTagName(DECAY_RATE_OF_VIRUS_NA
            DepositionOnSurfaceRate =
                Convert.ToDouble(xmlDatas.GetElementsByTagName(DEPOSITION_ON_SURFACE_
            ImmunityDurationMin =
                Convert.ToInt32(xmlDatas.GetElementsByTagName(IMMUNITY_DURATION_MIN_N
            ImmunityDurationMax =
                Convert.ToInt32(xmlDatas.GetElementsByTagName(IMMUNITY_DURATION_MAX_N
            ImmunityEfficiency =
                Convert.ToDouble(xmlDatas.GetElementsByTagName(IMMUNITY_EFFICIENCY_NA
            CoughMinQuanta = DEFAULT_MIN_QUANTA;
            CoughMaxQuanta = DEFAULT_MAX_QUANTA;
            IsCoughSymptomActive = true;
            IsAerosolTransmissionActive = true;
        }
        // Virus
        public static int IncubationDurationMin { get; set; }
        public static int IncubationDurationMax { get; set; }
        public static int DurationMin { get; set; }
        public static int DurationMax { get; set; }
        public static double DecayRateOfVirus { get; set; }
        public static double DepositionOnSurfaceRate { get; set; }
        public static int ImmunityDurationMin { get; set; }
        public static int ImmunityDurationMax { get; set; }
        public static double ImmunityEfficiency { get; set; }

        // Symptoms
        public static int CoughMaxQuanta { get; set; }
        public static int CoughMinQuanta { get; set; }
        public static bool IsCoughSymptomActive { get; set; }

        // Transmission
        public static bool IsAerosolTransmissionActive { get; set; }
    }
}
```

Listing 15 – Code source de F*Simulation/FParameters/QuarantineParameters.cs*

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
```

```
    vaste représentant une ville.
 */

namespace CovidPropagation
{
    static class QuarantineParameters
    {
        private const double DEFAULT_PROBABILITY_OF_BEING_QUARANTINED =
            0.95;
        private const int DEFAULT_DURATION_OF_BEING_QUARANTINED = 14;
        public static void Init()
        {
            IshealthyQuarantined = false;
            IsInfectedQuarantined = false;
            IsInfectiousQuarantined = false;
            IsImmuneQuarantined = false;

            ProbabilityOfHealthyQuarantined =
                DEFAULT_PROBABILITY_OF_BEING_QUARANTINED;
            ProbabilityOfInfectedQuarantined =
                DEFAULT_PROBABILITY_OF_BEING_QUARANTINED;
            ProbabilityOfInfectiousQuarantined =
                DEFAULT_PROBABILITY_OF_BEING_QUARANTINED;
            ProbabilityOfImmuneQuarantined =
                DEFAULT_PROBABILITY_OF_BEING_QUARANTINED;

            DurationHealthyQuarantined =
                DEFAULT_DURATION_OF_BEING_QUARANTINED;
            DurationInfectedQuarantined =
                DEFAULT_DURATION_OF_BEING_QUARANTINED;
            DurationInfectiousQuarantined =
                DEFAULT_DURATION_OF_BEING_QUARANTINED;
            DurationImmuneQuarantined =
                DEFAULT_DURATION_OF_BEING_QUARANTINED;
        }
        // Paramètres généraux
        public static bool IshealthyQuarantined { get; set; }
        public static bool IsInfectedQuarantined { get; set; }
        public static bool IsInfectiousQuarantined { get; set; }
        public static bool IsImmuneQuarantined { get; set; }

        public static double ProbabilityOfHealthyQuarantined { get;
            set; }
        public static double ProbabilityOfInfectedQuarantined { get;
            set; }
        public static double ProbabilityOfInfectiousQuarantined { get;
            set; }
        public static double ProbabilityOfImmuneQuarantined { get; set;
            }

        public static int DurationHealthyQuarantined { get; set; }
        public static int DurationInfectedQuarantined { get; set; }
        public static int DurationInfectiousQuarantined { get; set; }
        public static int DurationImmuneQuarantined { get; set; }
    }
```

```
}
```

Listing 16 – Code source de $\mathsf{F}_{Simulation/FParameters/VaccinationParameters.cs}$

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste représentant une ville.
 */

namespace CovidPropagation
{
    /// <summary>
    /// Classe contenant les paramètres de la vaccination.
    /// </summary>
    class VaccinationParameters
    {
        private const int VACCIN_DURATION = 30 * 6;
        private const double VACCIN_EFFICIENCY = 0.95;
        public static void Init()
        {
            Duration = VACCIN_DURATION;
            Efficiency = VACCIN_EFFICIENCY;
        }

        public static int Duration { get; set; }
        public static double Efficiency { get; set; }
    }
}
```

Listing 17 – Code source de $\mathsf{F}_{Simulation/FPersons/TimeFrame.cs}$

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste représentant une ville.
 */

namespace CovidPropagation
{
    /// <summary>
    /// ID Documentation : Acitvity_Class
    /// Période composé d'une activité.
    /// </summary>
    public class TimeFrame
    {
        private Site _activity;
        private SitePersonStatus _personStatus;
```

```
        public Site Activity { get => _activity; }
        public SitePersonStatus PersonStatus { get => _personStatus; }

        public TimeFrame(Site activity, SitePersonStatus reason)
        {
            _activity = activity;
            _personStatus = reason;
        }
    }
}
```

Listing 18 – Code source de F$_S$*imulation/F$_P$ersons/PersonState.cs*

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste représentant une ville.
 */

namespace CovidPropagation
{
    /// <summary>
    /// État des individus.
    /// </summary>
    public enum PersonState
    {
        Dead = -1,
        Healthy = 0,
        Immune = 1,
        Infected = 2,
        Infectious = 3,
        Asymptomatic = 4
    }
}
```

Listing 19 – Code source de F$_S$*imulation/F$_P$ersons/Person.cs*

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste représentant une ville.
 */
using System;
using System.Linq;
using System.Collections.Generic;

namespace CovidPropagation
{
    /// <summary>
```

```
    /// ID Documentation : Person_Class
    /// Individus se déplacant dans différents lieu en fonction de son
       planning.
    /// Peut être infecté et modifier les chances d'être infecté dans
       un lieu.
    /// </summary>
    public class Person
    {
        private const double PROBABILITY_OF_WEARING_CLOTH_MASK = 0.18;
        private const double PROBABILITY_OF_WEARING_FACESHIELD = 0.02;
        private const double PROBABILITY_OF_WEARING_N95_MASK = 0.01;
        private const double PROBABILITY_OF_WEARING_SURGICA_MASKL = 0.7;

        private const int MIN_RESISTANCE_BEFORE_HOSPITALISATION = 30;
        private const int MIN_RESISTANCE_BEFORE_DEATH = 10;

        private const int MIN_TIME_TO_DIE = 5;  // En jours
        private const int MAX_TIME_TO_DIE = 13; // En jours

        private const int DEFAULT_DEATH_DECREMENT = 2;

        private static int ids = 0;
        private int id;

        private Planning _planning;
        private Site _currentSite;
        private PersonState _state;
        private List<Ilness> _ilnesses;
        private List<Symptom> _symptoms;
        private double _virusResistance;
        private double _baseVirusResistance;
        private int _age;
        private int _virusDuration;
        private int _virusIncubationDuration;
        private int _immunityDuration;
        private double _immunityProtection;
        private Random _rdm;
        private double _quantaExhalationRate;
        private bool _hasMask;
        private Mask _mask;
        private bool _isQuarantined;
        private bool _healthyQuarantined;
        private bool _infectedQuarantined;
        private bool _infectiousQuarantined;
        private bool _immuneQuarantined;
        private int _quarantineDuration;
        private Home _quarantineLocation;
        private Hospital _hospitalCovid;
        private bool _mustLeaveHospital;
        private int _timeBeforeDeath;


        public PersonState CurrentState { get => _state; set => _state
           = value; }
        public double QuantaExhalationRate { get =>
```

```
        _quantaExhalationRate; }
    public bool HasMask { get => _hasMask; }
    public double ExhalationMaskEfficiency { get =>
        _mask.ExhalationMaskEfficiency; }
    public double InhalationMaskEfficiency { get =>
        _mask.InhalationMaskEfficiency; }
    public int Age { get => _age; set => _age = value; }
    internal List<Ilness> Ilnesses { get => _ilnesses; set =>
        _ilnesses = value; }
    public bool MustLeaveHospital { set => _mustLeaveHospital =
        value; }
    public int Id { get => id; set => id = value; }

    public Person(Planning planning, Hospital hospital, int age =
        GlobalVariables.DEFAULT_PERSON_AGE, PersonState state =
        PersonState.Healthy)
    {
         _planning = planning;
         _state = state;
         _rdm = GlobalVariables.rdm;
         Age = age;
         _hospitalCovid = hospital;
         MustLeaveHospital = false;
         Ilnesses = new List<Ilness>();
         _symptoms = new List<Symptom>();
         _isQuarantined = false;
         _healthyQuarantined = false;
         _infectedQuarantined = false;
         _infectiousQuarantined = false;
         _immuneQuarantined = false;

         KeyValuePair<Object, double>[] probabilityOfMaskType = {
             new KeyValuePair<Object, double>(TypeOfMask.Cloth,
                 PROBABILITY_OF_WEARING_CLOTH_MASK),
             new KeyValuePair<Object, double>(TypeOfMask.FaceShield,
                 PROBABILITY_OF_WEARING_FACESHIELD),
             new KeyValuePair<Object, double>(TypeOfMask.N95,
                 PROBABILITY_OF_WEARING_N95_MASK),
             new KeyValuePair<Object, double>(TypeOfMask.Surgical,
                 PROBABILITY_OF_WEARING_SURGICA_MASKL),
         };

         _mask = new
             Mask((TypeOfMask)_rdm.NextProbability(probabilityOfMaskType));
         _hasMask = false;

         // Initialise la résistance de base de la personne
         if (_rdm.Next(0, 100) <
             GlobalVariables.PERCENTAGE_OF_ASYMPTOMATIC)
              _baseVirusResistance =
                  _rdm.Next(GlobalVariables.ASYMPTOMATIC_MIN_RESISTANCE,
                  GlobalVariables.ASYMPTOMATIC_MAX_RESISTANCE);
         else
              _baseVirusResistance =
                  _rdm.Next(GlobalVariables.SYMPTOMATIC_MIN_RESISTANCE,
```

```
                GlobalVariables.SYMPTOMATIC_MAX_RESISTANCE);

        _virusResistance = _baseVirusResistance;
        _currentSite = _planning.GetActivity();
        _quarantineLocation = (Home)_currentSite; // Possible
            uniquement car tous les individus se trouve chez eux au
            démarrage.
        _quantaExhalationRate =
            _currentSite.AverageQuantaExhalationRate;
        if ((int)_state >= (int)PersonState.Infected)
        {
            SetInfectionDurations(_state);
            if ((int)_state > (int)PersonState.Infected)
                VirusIncubationOver();
        }

        ids++;
        Id = ids;
    }

    /// <summary>
    /// ID Documentation : Change_Activity
    /// Change d'activité et recalcul les chances d'attraper le
        virus.
    /// Change d'état si besoin.
    /// Décrémente la durée d'incubation du virus si infecté.
    /// Décrémente la durée de vie du virus si infecté et que la
        durée d'incubation est terminée.
    /// Décrémente la durée d'immunité si la personne est immunisé.
    /// Calcul si l'individus attrape une maladie et decrémente la
        durée de celles déjà existantes.
    /// Retire les maladies dont la durée est terminée.
    /// </summary>
    public Site ChangeActivity()
    {
        // Quitte l'hôpital
        if (_mustLeaveHospital)
        {
            _hospitalCovid.LeaveForCovid(this);
            _mustLeaveHospital = false;
        }

        // Si la résistance de l'individus est suffisament faible
            pour décèder, qu'il ne se situe pas à l'hopital et que
            le virus s'est développé.
        if (_virusResistance <= MIN_RESISTANCE_BEFORE_DEATH &&
            _currentSite != _hospitalCovid && (int)_state >
            (int)PersonState.Infected)
        {
            DecrementUntilDeath(DEFAULT_DEATH_DECREMENT);
        }

        if (!_isQuarantined)
        {
            // Vérifie s'il doit aller en quarantaine
```

```
            switch (_state)
            {
                default:
                case PersonState.Dead:
                    _isQuarantined = false;
                    break;
                case PersonState.Asymptomatic:
                case PersonState.Healthy:
                    if (_healthyQuarantined &&
                        QuarantineParameters.ProbabilityOfHealthyQuarantined
                        > _rdm.NextDouble())
                    {
                        _quarantineDuration =
                            QuarantineParameters.DurationHealthyQuarantined
                            * GlobalVariables.NUMBER_OF_TIMEFRAME;
                        _isQuarantined = true;
                    }
                    break;
                case PersonState.Immune:
                    if (_immuneQuarantined &&
                        QuarantineParameters.ProbabilityOfImmuneQuarantined
                        > _rdm.NextDouble())
                    {
                        _quarantineDuration =
                            QuarantineParameters.DurationImmuneQuarantined
                            * GlobalVariables.NUMBER_OF_TIMEFRAME;
                        _isQuarantined = true;
                    }
                    break;
                case PersonState.Infected:
                    if (_infectedQuarantined &&
                        QuarantineParameters.ProbabilityOfInfectedQuarantined
                        > _rdm.NextDouble())
                    {
                        _quarantineDuration =
                            QuarantineParameters.DurationInfectedQuarantined
                            * GlobalVariables.NUMBER_OF_TIMEFRAME;
                        _isQuarantined = true;
                    }
                    break;
                case PersonState.Infectious:
                    if (_infectiousQuarantined &&
                        QuarantineParameters.ProbabilityOfInfectiousQuarantined
                        > _rdm.NextDouble())
                    {
                        _quarantineDuration =
                            QuarantineParameters.DurationInfectiousQuarantined
                            * GlobalVariables.NUMBER_OF_TIMEFRAME;
                        _isQuarantined = true;
                    }
                    break;
            }
        }

        // ID Documentation : Person_Hospital
```

```
            // Si la résistance de l'individus est suffisament faible
               pour être hôspitaliser, et que le virus s'est développé.
            if (_virusResistance <
               MIN_RESISTANCE_BEFORE_HOSPITALISATION && (int)_state >
               (int)PersonState.Infected)
            {
                // S'il ne se situe pas déjà à l'hôpital
                if (_currentSite != _hospitalCovid)
                {
                    // Entre dans l'hôpital s'il y a de la place.
                    // Sinon, se confine.
                    if (_hospitalCovid.EnterForCovid(this))
                    {
                        _currentSite.Leave(this);
                        _currentSite = _hospitalCovid;
                        _timeBeforeDeath =
                           Convert.ToInt32((_rdm.Next(MIN_TIME_TO_DIE,
                           MAX_TIME_TO_DIE) + _rdm.NextDouble()) *
                           GlobalVariables.NUMBER_OF_TIMEFRAME);
                    }
                    else
                    {
                        _currentSite.Leave(this);
                        _currentSite = _quarantineLocation;
                        _quarantineLocation.Enter(this,
                           SitePersonStatus.Other);
                    }
                }
            }else if (_isQuarantined)
            {
                // Ajouter trajet
                _currentSite.Leave(this);
                _currentSite = _quarantineLocation;
                _quarantineLocation.Enter(this, SitePersonStatus.Other);

                if (_quarantineDuration <= 0)
                    _isQuarantined = false;

                _quarantineDuration--;
            }else
            {
                // Quitte le lieu précédent s'il est différent,
                   récupère le nouveau et entre dedans.
                Site newSite = _planning.GetActivity();
                if (_currentSite != newSite)
                {
                    _currentSite.Leave(this);
                    _currentSite = newSite;
                    _currentSite.Enter(this,
                        _planning.PersonTypeInActivity());

                    // Calul les quantas exhalé en fonction des
                       symptômes ainsi que du lieux.
                    _quantaExhalationRate =
                        _currentSite.AverageQuantaExhalationRate;
```

```
                if (_symptoms.OfType<CoughSymptom>().Any())
                {
                    _quantaExhalationRate +=
                        _symptoms.OfType<CoughSymptom>().First().QuantaAddedByCou
                }
            }
        }

        return _currentSite;
    }

    /// <summary>
    /// Décrémente le temps restant de l'individu de vivre
       lorsqu'il est hospitalisé.
    /// </summary>
    public void GetHospitalTreatment()
    {
        if (_virusResistance <= MIN_RESISTANCE_BEFORE_DEATH)
        {
            DecrementUntilDeath(1);
        }
    }

    /// <summary>
    /// Décrémente la durée avant le décès de l'individu et change
       son état lorsque la durée atteint 0.
    /// </summary>
    /// <param name="decrement">Valeur décrémentant la durée avant
       le décès.</param>
    private void DecrementUntilDeath(int decrement)
    {
        if (_timeBeforeDeath <= 0)
        {
            _state = PersonState.Dead;
        }
        _timeBeforeDeath -= decrement;
    }

    /// <summary>
    /// ID Documentation : Check_State
    /// Vérifie l'état de contamination de la personne et le change
       si elle est infectée.
    /// Décémente la durée du virus ainsi que des maladies.
    /// Attrape possiblement des maladies.
    /// </summary>
    public PersonState ChechState()
    {
        double contaminationProbability =
            _currentSite.GetProbabilityOfInfection();

        if (_state == PersonState.Healthy &&
            contaminationProbability >= _rdm.NextDouble())
        {
            _currentSite.HasEnvironnementChanged = true;
            SetInfectionDurations(PersonState.Infected);
```

```
            }
            else if(_state == PersonState.Immune &&
                contaminationProbability - _immunityProtection *
                (contaminationProbability / 100) >= _rdm.NextDouble())
                // Réduit les probabilité d'infection en fonction de la
                protection immunitaire.
            {
                _currentSite.HasEnvironnementChanged = true;
                SetInfectionDurations(PersonState.Infected);
            }

            ContractIlness();
            DecreaseImmunityDuration();
            DecreaseVirusDuration();
            return _state;
        }

        /// <summary>
        /// ID Documentation : Virus_Incubation
        /// Initialize la durée de vie du virus ainsi que sa durée
            d'incubation.
        /// </summary>
        /// <param name="state">Nouvel état de l'individu.</param>
        private void SetInfectionDurations(PersonState state)
        {
            _state = state;
            _virusDuration = Virus.Duration *
                GlobalVariables.NUMBER_OF_TIMEFRAME;
            _virusIncubationDuration =
                Convert.ToInt32(Virus.IncubationDuration *
                GlobalVariables.NUMBER_OF_TIMEFRAME);
        }

        /// <summary>
        /// Met le masque de la personne.
        /// </summary>
        public void PutMaskOn()
        {
            _hasMask = true;
        }

        /// <summary>
        /// Retire le masque de la personne.
        /// </summary>
        public void RemoveMask()
        {
            _hasMask = false;
        }

        /// <summary>
        /// Initialise la quarantaine ainsi que sa durée.
        /// </summary>
        public void SetQuarantine(bool healthyQuarantined, bool
            infectedQuarantined, bool infectiousQuarantined, bool
            immuneQuarantined)
```

```
        {
            _healthyQuarantined = healthyQuarantined;
            _infectedQuarantined = infectedQuarantined;
            _infectiousQuarantined = infectiousQuarantined;
            _immuneQuarantined = immuneQuarantined;
        }

        /// <summary>
        /// Vaccine l'individu qui devient immunisé pendant la durée du
            vaccin.
        /// </summary>
        public void GetVaccinated()
        {
            _state = PersonState.Immune;
            _immunityDuration = VaccinationParameters.Duration *
                GlobalVariables.NUMBER_OF_TIMEFRAME;
            _immunityProtection = VaccinationParameters.Efficiency;
        }

        /// <summary>
        /// Décrémente la durée d'immunité si la personne est immunisé
            et change son état.
        /// </summary>
        private void DecreaseImmunityDuration()
        {
            if (_state == PersonState.Immune)
            {
                if (_immunityDuration > 0)
                    _immunityDuration--;
                else
                    _state = PersonState.Healthy;
            }
        }

        /// <summary>
        /// ID Documentation : Incubation_Acitvity_Decrement
        /// Décrémente la durée d'incubation du virus si infecté.
        /// Décrémente la durée de vie du virus si infecté et que la
            durée d'incubation est terminée.
        /// Et change l'état en fonction du résultat.
        /// </summary>
        private void DecreaseVirusDuration()
        {
            // Si l'individu est infecté.
            if (_state == PersonState.Infected)
            {
                // Diminu la durée d'incubation du virus ou change
                    l'état de l'individu et ajoute les symptômes.
                if (_virusIncubationDuration > 0)
                    _virusIncubationDuration--;
                else
                {
                    VirusIncubationOver();
                }
            }else if ((int)_state > (int)PersonState.Infected) // S'il
```

```
            est contagieux.
        {
            // Diminu la durée du virus ou change l'état de
               l'individu pour qu'il soit immunisé
            if (_virusDuration > 0)
                _virusDuration--;
            else
            {
                _state = PersonState.Immune;
                _immunityDuration =
                   _rdm.Next(Virus.ImmunityDurationMin,
                   Virus.ImmunityDurationMax) *
                   GlobalVariables.NUMBER_OF_TIMEFRAME;
                _immunityProtection = Virus.ImmunityEfficiency;
            }
        }
    }

    /// <summary>
    /// ID Documentation : Symptoms_After_Incubation
    /// Lorsque l'incubation du virus est terminée, change l'état
       en asymptomatique ou en infectieux.
    /// </summary>
    private void VirusIncubationOver()
    {
        if (_virusResistance >
           GlobalVariables.ASYMPTOMATIC_MIN_RESISTANCE)
        {
            _state = PersonState.Asymptomatic;
        }
        else
        {
            _state = PersonState.Infectious;
            _symptoms.AddRange(Virus.GetCommonSymptoms());
        }
    }

    /// <summary>
    /// ID Documentation : Contract_Ilness
    /// Calcul si l'individus attrape une maladie et decrémente la
       durée de celles déjà existantes.
    /// Retire les maladies dont la durée est terminée.
    /// </summary>
    private void ContractIlness()
    {
        // Si attrape alors Modifier pour prendre en compte l'âge
        if (GlobalVariables.ILNESS_INFECTION_PROBABILITY >
           _rdm.NextDouble())
        {
            Ilness newIlness = new Ilness(Age, _rdm);
            Ilnesses.Add(newIlness);
        }
        RecalculateVirusResistance(1);
    }
```

```
        /// <summary>
        /// Recalcule la résistance au virus de l'individu.
        /// Vérifie si l'individu est toujours infecté par ses maladies.
        /// </summary>
        /// <param name="decrement">Valeur décrémentant la durée
           restante</param>
        public void RecalculateVirusResistance(int decrement)
        {
            Ilnesses.ForEach(i =>
               i.DecrementTimeBeforeDesapearance(decrement));
            Ilnesses.RemoveAll(i => i.Desapear());
            _virusResistance = _baseVirusResistance - Ilnesses.Sum(i =>
               i.Attack);
        }

        /// <summary>
        /// Récupère le prochain site de l'individu.
        /// </summary>
        /// <returns>Site contenu dans la prochaine activité</returns>
        public Site GetNextActivitySite()
        {
            return _planning.GetNextActivity();
        }
    }
}
```

Listing 20 – Code source de F$_{Simulation/FPersons/Day.cs}$

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste représentant une ville.
 */
using System;
using System.Collections.Generic;
using System.Linq;

namespace CovidPropagation
{
    /// <summary>
    /// ID Documentation : Day_Class
    /// Jour composé de plusieurs périodes.
    /// </summary>
    public class Day
    {
        private const int FREEDAY_MORNING_TIME_FRAME_MAX = 22;
        private const int FREEDAY_MORNING_MIN = 12;

        private const int FREEDAY_NOON_TIME_FRAME_MAX = 4;
        private const int FREEDAY_NOON_MIN = 3;
```

```
        private const int FREEDAY_AFTER_NOON_TIME_FRAME_MAX = 10;
        private const int FREEDAY_AFTER_NOON_MIN = 5;

        private const int FREEDAY_EVENING_TIME_FRAME_MAX = 8;
        private const int FREEDAY_EVENING_MIN = 3;

        private const int FREEDAY_MIN_TIME_FRAME_VALUE = 3;

        private const int WORKDAY_MORNING_TIME_FRAME_MAX = 16;
        private const int WORKDAY_MORNING_MIN = 4;
        private const int WORKDAY_MORNING_TIME_FRAME_TOTAL = 22;

        private const int WORKDAY_NOON_TIME_FRAME_MAX = 4;
        private const int WORKDAY_NOON_MIN = 3;

        private const int WORKDAY_AFTER_NOON_TIME_FRAME_MAX = 10;
        private const int WORKDAY_AFTER_NOON_MIN = 4;
        private const int WORKDAY_AFTER_NOON_TIME_FRAME_TOTAL = 10;

        private const int WORKDAY_EVENING_TIME_FRAME_MAX = 8;
        private const int WORKDAY_EVENING_MIN = 3;

        private const int WORKDAY_MIN_TIME_FRAME_VALUE = 3;


        private TimeFrame[] _timeFrames;
        public TimeFrame[] TimeFrames { get => _timeFrames; }

        /// <summary>
        /// ID Documentation : Day_Creation
        /// </summary>
        public Day(Dictionary<SiteType, List<Site>> personSites, bool
            isWorkDay)
        {
            if (isWorkDay)
                _timeFrames = CreateWorkDay(personSites);
            else
                _timeFrames = CreateFreeDay(personSites);
        }

        /// <summary>
        /// Récupère l'activité à l'index spécifié.
        /// </summary>
        /// <param name="timeFrame">Index de la période à
            récupérer</param>
        /// <returns>Activité dans la période.</returns>
        public Site GetActivity(int timeFrame)
        {
            return _timeFrames[timeFrame].Activity;
        }

        /// <summary>
        /// Récupère l'activité actuelle
        /// </summary>
        /// <returns>L'activité en cours.</returns>
```

```
        public Site GetCurrentActivity()
        {
            return _timeFrames[TimeManager.CurrentTimeFrame].Activity;
        }

        /// <summary>
        /// Récupère la raison pour laquel l'individu est dans le lieu
           actuel.
        /// </summary>
        /// <returns>Raison de se situer sur le lieu actuel.</returns>
        public SitePersonStatus GetCurrentPersonTypeInActivity()
        {
            return
                _timeFrames[TimeManager.CurrentTimeFrame].PersonStatus;
        }

        /// <summary>
        /// ID Documentation : WorkDay_Creation
        /// Créé un jour de travail pour une personne avec les lieux
           qui lui sont fournies.
        /// </summary>
        /// <param name="personSites">Lieux avec lequel le planning
           sera créé</param>
        /// <returns></returns>
        private TimeFrame[] CreateWorkDay(Dictionary<SiteType,
           List<Site>> personSites)
        {
            int totalTimeFrame = GlobalVariables.NUMBER_OF_TIMEFRAME;
            Random rdm = GlobalVariables.rdm;
            List<TimeFrame> timeFrames;

            int morningTimeFrameMax = WORKDAY_MORNING_TIME_FRAME_MAX,
                morningVariation = WORKDAY_MORNING_MIN;
            int morningTimeFrameTotal =
                WORKDAY_MORNING_TIME_FRAME_TOTAL;
            int noonTimeFrameMax = WORKDAY_NOON_TIME_FRAME_MAX, noonMin
                = WORKDAY_NOON_MIN;
            int afterNoonTimeFrameMax =
                WORKDAY_AFTER_NOON_TIME_FRAME_MAX, afterNoonVariation =
                WORKDAY_AFTER_NOON_MIN;
            int afterNoonTimeFrameTotal =
                WORKDAY_AFTER_NOON_TIME_FRAME_TOTAL;
            int eveningTimeFrameMax = WORKDAY_EVENING_TIME_FRAME_MAX,
                eveningMin = WORKDAY_EVENING_MIN;
            int nightTimeFrame; // remplit ce qu'il manque

            int morningWorkTimeFrame = 0;
            int afterNoonFreeTimeTimeFrame = 0;
            int eveningActivityTimeFrame = 0;

            int morningTimeFrame = morningTimeFrameMax -
                rdm.NextWithMinimum(0, morningVariation, 0);
            int noonTimeFrame = rdm.Next(noonMin, noonTimeFrameMax + 1);
            int afterNoonWorkTimeFrame = afterNoonTimeFrameMax -
                rdm.NextWithMinimum(0, afterNoonVariation, 0);
```

```
            int eveningTimeFrame = rdm.NextWithMinimum(eveningMin,
                eveningTimeFrameMax, WORKDAY_MIN_TIME_FRAME_VALUE);

            #region Activities

            if (morningTimeFrame < morningTimeFrameTotal)
                morningWorkTimeFrame = morningTimeFrameTotal -
                    morningTimeFrame;

            if (noonTimeFrame < noonTimeFrameMax)
                afterNoonTimeFrameMax += 1;

            if (afterNoonWorkTimeFrame < afterNoonTimeFrameTotal)
                afterNoonFreeTimeTimeFrame = afterNoonTimeFrameTotal -
                    afterNoonWorkTimeFrame;

            if (eveningTimeFrame < eveningTimeFrameMax)
                eveningActivityTimeFrame = eveningTimeFrameMax -
                    eveningTimeFrame;
            #endregion

            nightTimeFrame = (morningTimeFrame + morningWorkTimeFrame) +
                             (noonTimeFrame) +
                             (afterNoonWorkTimeFrame +
                                 afterNoonFreeTimeTimeFrame) +
                             (eveningTimeFrame +
                                 eveningActivityTimeFrame);

            nightTimeFrame = totalTimeFrame - nightTimeFrame;

            // Choisis les lieux où l'individus va passer son temps en
                fonction des données reçues.
            KeyValuePair<Site, SitePersonStatus> hometSite = new
                KeyValuePair<Site, SitePersonStatus>(
                personSites[SiteType.Home][rdm.Next(0,
                    personSites[SiteType.Home].Count)],
                SitePersonStatus.Other
                );

            // Récupère le lieu de travail et la raison d'y aller
            KeyValuePair<Site, SitePersonStatus> workSite = new
                KeyValuePair<Site, SitePersonStatus>(
                personSites[SiteType.WorkPlace][rdm.Next(0,
                    personSites[SiteType.WorkPlace].Count)],
                SitePersonStatus.Worker
                );

            // Manger sur le lieu de travail ou ailleurs
            KeyValuePair<Site, SitePersonStatus> noonSite;
            if (rdm.NextBoolean())
                noonSite = new KeyValuePair<Site, SitePersonStatus>(
                personSites[SiteType.Eat][rdm.Next(0,
                    personSites[SiteType.Eat].Count)],
                SitePersonStatus.Client
                );
```

```
            else
                noonSite = workSite;

            // Récupère le lieux de l'après midi si l'individu quitte
               le travail plus tôt
            SiteType afterNoonSiteType = (SiteType)rdm.Next(0,
               (int)SiteType.Eat);
            KeyValuePair<Site, SitePersonStatus> afterNoonFreeTimeSite
               = new KeyValuePair<Site, SitePersonStatus>(
                personSites[afterNoonSiteType][rdm.Next(0,
                   personSites[afterNoonSiteType].Count)],
                SitePersonStatus.Worker
                );

            // Récupère le lieux du soir si l'individu quitte le
               travail plus tôt
            KeyValuePair<Site, SitePersonStatus> eveningActivitySite =
               new KeyValuePair<Site, SitePersonStatus>(
                personSites[SiteType.Eat][rdm.Next(0,
                   personSites[SiteType.Eat].Count)],
                SitePersonStatus.Client
                );


            // Récupère le moyen de transport et la raison de se situer
               dedans
            KeyValuePair<Site, SitePersonStatus> transportSite = new
               KeyValuePair<Site, SitePersonStatus>(
                personSites[SiteType.Transport][rdm.Next(0,
                   personSites[SiteType.Transport].Count)],
                SitePersonStatus.Other
                );

            timeFrames = new List<TimeFrame>(totalTimeFrame);

            // Une fois que tous les lieux sont choisis, les périodes
               de la journée sont créés.
            CreateMorning(timeFrames, hometSite, morningTimeFrame,
               workSite, morningWorkTimeFrame, transportSite);
            CreateNoon(timeFrames, noonSite, noonTimeFrame,
               transportSite);
            CreateAfterNoon(timeFrames, workSite,
               afterNoonWorkTimeFrame, afterNoonFreeTimeSite,
               afterNoonFreeTimeTimeFrame, transportSite);
            CreateEvening(timeFrames, hometSite, eveningTimeFrame,
               eveningActivitySite, eveningActivityTimeFrame,
               transportSite);
            CreateNight(timeFrames, hometSite, nightTimeFrame,
               transportSite);

            return timeFrames.ToArray();
        }

        /// <summary>
        /// Créé un jour de congé pour une personne avec les lieux qui
```

```
        lui sont attribués
    /// </summary>
    /// <param name="personSites">Lieux avec lequel le planning
        sera créé</param>
    /// <returns>Tableaux contenant les timesFrames créés.</returns>
    private TimeFrame[] CreateFreeDay(Dictionary<SiteType,
        List<Site>> personSites)
    {
        int totalTimeFrame = GlobalVariables.NUMBER_OF_TIMEFRAME;
        Random rdm = GlobalVariables.rdm;
        List<TimeFrame> timeFrames;

        int morningTimeFrameMax = FREEDAY_MORNING_TIME_FRAME_MAX,
            morningMin = FREEDAY_MORNING_MIN;
        int noonTimeFrameMax = FREEDAY_NOON_TIME_FRAME_MAX, noonMin
            = FREEDAY_NOON_MIN;
        int afterNoonTimeFrameMax =
            FREEDAY_AFTER_NOON_TIME_FRAME_MAX, afterNoonMin =
            FREEDAY_AFTER_NOON_MIN;
        int eveningTimeFrameMax = FREEDAY_EVENING_TIME_FRAME_MAX,
            eveningMin = FREEDAY_EVENING_MIN;
        int nightTimeFrame; // remplit ce qu'il manque

        int morningActivityTimeFrame = 0;
        int afterNoonActivityTimeFrame = 0;
        int eveningActivityTimeFrame = 0;

        int morningTimeFrame = rdm.NextWithMinimum(morningMin,
            morningTimeFrameMax, FREEDAY_MIN_TIME_FRAME_VALUE);
        int noonTimeFrame = rdm.Next(noonMin, noonTimeFrameMax + 1);
        int afterNoonTimeFrame = rdm.NextWithMinimum(afterNoonMin,
            afterNoonTimeFrameMax, FREEDAY_MIN_TIME_FRAME_VALUE);
        int eveningTimeFrame = rdm.NextWithMinimum(eveningMin,
            eveningTimeFrameMax, FREEDAY_MIN_TIME_FRAME_VALUE);

        // Activities
        #region Activities

        if (morningTimeFrame < morningTimeFrameMax)
            morningActivityTimeFrame = morningTimeFrameMax -
                morningTimeFrame;

        if (noonTimeFrame < noonTimeFrameMax)
            afterNoonTimeFrameMax += 1;

        if (afterNoonTimeFrame < afterNoonTimeFrameMax)
            afterNoonActivityTimeFrame = afterNoonTimeFrameMax -
                afterNoonTimeFrame;

        if (eveningTimeFrame < eveningTimeFrameMax)
            eveningActivityTimeFrame = eveningTimeFrameMax -
                eveningTimeFrame;
        #endregion

        nightTimeFrame = (morningTimeFrame +
```

```
            morningActivityTimeFrame) +
                        (noonTimeFrame) +
                        (afterNoonTimeFrame +
                            afterNoonActivityTimeFrame) +
                        (eveningTimeFrame +
                            eveningActivityTimeFrame);

        nightTimeFrame = totalTimeFrame - nightTimeFrame;

        KeyValuePair<Site, SitePersonStatus> homeSite = new
            KeyValuePair<Site, SitePersonStatus>(
             personSites[SiteType.Home][rdm.Next(0,
                personSites[SiteType.Home].Count)],
             SitePersonStatus.Other
             );

        SiteType morningActivitySiteType = (SiteType)rdm.Next(0,
            (int)SiteType.Eat);
        KeyValuePair<Site, SitePersonStatus> morningActivitySite =
            new KeyValuePair<Site, SitePersonStatus>(
             personSites[morningActivitySiteType][rdm.Next(0,
                personSites[morningActivitySiteType].Count)],
             SitePersonStatus.Client
             );

        KeyValuePair<Site, SitePersonStatus> noonSite = new
            KeyValuePair<Site, SitePersonStatus>(
             personSites[SiteType.Eat][rdm.Next(0,
                personSites[SiteType.Eat].Count)],
             SitePersonStatus.Client
             );

        SiteType afterNoonActivitySiteType = (SiteType)rdm.Next(0,
            (int)SiteType.Eat);
        KeyValuePair<Site, SitePersonStatus> afterNoonActivitySite
            = new KeyValuePair<Site, SitePersonStatus>(
             personSites[afterNoonActivitySiteType][rdm.Next(0,
                personSites[afterNoonActivitySiteType].Count)],
             SitePersonStatus.Other
             );

        SiteType eveningActivitySiteType = (SiteType)rdm.Next(0,
            (int)SiteType.Eat);
        KeyValuePair<Site, SitePersonStatus> eveningActivitySite =
            new KeyValuePair<Site, SitePersonStatus>(
             personSites[eveningActivitySiteType][rdm.Next(0,
                personSites[eveningActivitySiteType].Count)],
             SitePersonStatus.Other
             );


        KeyValuePair<Site, SitePersonStatus> transportSite = new
            KeyValuePair<Site, SitePersonStatus>(
             personSites[SiteType.Transport][rdm.Next(0,
                personSites[SiteType.Transport].Count)],
```

```
                SitePersonStatus.Other
                );

            timeFrames = new List<TimeFrame>(totalTimeFrame);
            CreateMorning(timeFrames, homeSite, morningTimeFrame,
                morningActivitySite, morningActivityTimeFrame,
                transportSite);
            CreateNoon(timeFrames, noonSite, noonTimeFrame,
                transportSite);
            CreateAfterNoon(timeFrames, homeSite, afterNoonTimeFrame,
                afterNoonActivitySite, afterNoonActivityTimeFrame,
                transportSite);
            CreateEvening(timeFrames, homeSite, eveningTimeFrame,
                eveningActivitySite, eveningActivityTimeFrame,
                transportSite);
            CreateNight(timeFrames, homeSite, nightTimeFrame,
                transportSite);

            return timeFrames.ToArray();
        }

        /// <summary>
        /// Créé la matinée d'une personne dans son planning.
        /// Comprend un certain temps dans son appartement, puis une
            possible activité.
        /// Peut comprendre un lieu de travail à la place d'une
            activité.
        /// Si les lieux sont différents les uns des autres, des
            trajets sont ajouté entre eux modifiant les timeFrames de
            ceux-ci.
        /// </summary>
        /// <param name="morningTimeFrames">Durée de la matinée en
            timeFrames</param>
        /// <param name="home">Appartement de la personne.</param>
        /// <param name="homeTimeFrame">Durée dans l'appartement en
            timeFrames</param>
        /// <param name="activity">Lieu de l'activité à effectuer /
            travail.</param>
        /// <param name="activityTimeFrame">Durée de l'activité /
            travail.</param>
        /// <param name="transport">Type de transport utilisé pour se
            déplacer entre deux lieux.</param>
        private void CreateMorning(List<TimeFrame> morningTimeFrames,
            KeyValuePair<Site, SitePersonStatus> home, int
            homeTimeFrame, KeyValuePair<Site, SitePersonStatus>
            activity, int activityTimeFrame, KeyValuePair<Site,
            SitePersonStatus> transport)
        {
            morningTimeFrames.AddRange(Enumerable.Repeat(new
                TimeFrame(home.Key, home.Value), homeTimeFrame));
            if (activity.Key != home.Key && activityTimeFrame > 0)
            {
                morningTimeFrames.RemoveAt(morningTimeFrames.GetLastIndex());
                morningTimeFrames.Add(new TimeFrame(transport.Key,
                    transport.Value));
```

```
            }
            morningTimeFrames.AddRange(Enumerable.Repeat(new
                TimeFrame(activity.Key, activity.Value),
                activityTimeFrame));
        }

        /// <summary>
        /// Créé les heures du midi d'une personne dans son plannning.
        /// Peut se situer dans divers lieu d'on le lieu où elle se
            situe déjà.
        /// Si le lieu est différent, des trajets sont ajouté.
        /// </summary>
        /// <param name="timeFrames">Liste des précédents lieux.</param>
        /// <param name="site">Lieu dans lequel la section de midi se
            situe.</param>
        /// <param name="siteTimeFrames">Durée de la section de midi en
            timeFrames.</param>
        /// <param name="transportSite">Type de transport utilisé pour
            se déplacer entre deux lieux.</param>
        private void CreateNoon(List<TimeFrame> timeFrames,
            KeyValuePair<Site, SitePersonStatus> site, int
            siteTimeFrames, KeyValuePair<Site, SitePersonStatus>
            transportSite)
        {
            if (timeFrames[timeFrames.GetLastIndex()].Activity !=
                site.Key)
            {
                timeFrames.RemoveAt(timeFrames.GetLastIndex());
                timeFrames.Add(new TimeFrame(transportSite.Key,
                    transportSite.Value));
            }
            timeFrames.AddRange(Enumerable.Repeat(new
                TimeFrame(site.Key, site.Value), siteTimeFrames));
        }

        /// <summary>
        /// Créé l'après-midi d'une personne dans son planning.
        /// Comprend un certain temps dans son appartement ou au
            travail, puis une possible activité.
        /// Si les lieux sont différents les uns des autres, des
            trajets sont ajouté entre eux modifiant les timeFrames de
            ceux-ci.
        /// </summary>
        /// <param name="timeFrames">Liste des précédents lieux.</param>
        /// <param name="site">Lieu dans lequel l'après-midi se
            situe.</param>
        /// <param name="siteTimeFrames">Durée de la section de
            l'après-midi en timeFrames.</param>
        /// <param name="activitySite">Lieu d'une possible
            activité.</param>
        /// <param name="activityTimeFrames">Durée de l'activité si
            celle-ci à lieu en timeFrames.</param>
        /// <param name="transportSite">Type de transport utilisé pour
            se déplacer entre deux lieux.</param>
        private void CreateAfterNoon(List<TimeFrame> timeFrames,
```

```
        KeyValuePair<Site, SitePersonStatus> site, int
        siteTimeFrames, KeyValuePair<Site, SitePersonStatus>
        activitySite, int activityTimeFrames, KeyValuePair<Site,
        SitePersonStatus> transportSite)
    {
        if (timeFrames[timeFrames.GetLastIndex()].Activity !=
            site.Key)
        {
            timeFrames.Add(new TimeFrame(transportSite.Key,
                transportSite.Value));
            siteTimeFrames--;
        }
        timeFrames.AddRange(Enumerable.Repeat(new
            TimeFrame(site.Key, site.Value), siteTimeFrames));
        if (activitySite.Key != site.Key && activityTimeFrames > 0)
        {
            timeFrames.Add(new TimeFrame(transportSite.Key,
                transportSite.Value));
            activityTimeFrames--;
        }
        timeFrames.AddRange(Enumerable.Repeat(new
            TimeFrame(activitySite.Key, activitySite.Value),
            activityTimeFrames));
    }

    /// <summary>
    /// Créé la soirée d'une personne dans son planning.
    /// Comprend un certain temps dans son appartement, puis une
        possible activité.
    /// Si les lieux sont différents les uns des autres, des
        trajets sont ajouté entre eux modifiant les timeFrames de
        ceux-ci.
    /// </summary>
    /// <param name="timeFrames">Liste des précédents lieux.</param>
    /// <param name="site">Lieu dans lequel la soirée se
        situe.</param>
    /// <param name="siteTimeFrames">Durée de la section de la
        soirée en timeFrames.</param>
    /// <param name="activitySite">Lieu d'une possible
        activité.</param>
    /// <param name="activityTimeFrames">Durée de l'activité si
        celle-ci à lieu en timeFrames.</param>
    /// <param name="transportSite">Type de transport utilisé pour
        se déplacer entre deux lieux.</param>
    private void CreateEvening(List<TimeFrame> timeFrames,
        KeyValuePair<Site, SitePersonStatus> site, int
        siteTimeFrames, KeyValuePair<Site, SitePersonStatus>
        activitySite, int activityTimeFrames, KeyValuePair<Site,
        SitePersonStatus> transportSite)
    {
        if (timeFrames[timeFrames.GetLastIndex()].Activity !=
            site.Key)
        {
            timeFrames.RemoveAt(timeFrames.GetLastIndex());
            timeFrames.Add(new TimeFrame(transportSite.Key,
```

```
                transportSite.Value));
            }
            timeFrames.AddRange(Enumerable.Repeat(new
                TimeFrame(site.Key, site.Value), siteTimeFrames));
            if (activitySite.Key != site.Key && activityTimeFrames > 0)
            {
                timeFrames.Add(new TimeFrame(transportSite.Key,
                    transportSite.Value));
                activityTimeFrames--;
            }
            timeFrames.AddRange(Enumerable.Repeat(new
                TimeFrame(activitySite.Key, activitySite.Value),
                activityTimeFrames));
        }

        /// <summary>
        /// Créé la fin de la journée d'une personne dans son planning.
        /// Si la personne est déjà chez-elle, remplit simplement le
            nombre de timeFrames restant en choisissant la maison comme
            location.
        /// Si la personne n'est pas encore chez-elle, alors elle est
            déplacée.
        /// </summary>
        /// <param name="timeFrames">Liste des précédents lieux.</param>
        /// <param name="site">Lieu dans lequel la nuit se
            situe.</param>
        /// <param name="siteTimeFrames">Durée de la section de la
            soirée en timeFrames.</param>
        /// <param name="transportSite"></param>
        private void CreateNight(List<TimeFrame> timeFrames,
            KeyValuePair<Site, SitePersonStatus> site, int
            siteTimeFrames, KeyValuePair<Site, SitePersonStatus>
            transportSite)
        {
            if (timeFrames[timeFrames.GetLastIndex()].Activity !=
                site.Key)
            {
                timeFrames.RemoveAt(timeFrames.GetLastIndex());
                timeFrames.Add(new TimeFrame(transportSite.Key,
                    transportSite.Value));
            }
            timeFrames.AddRange(Enumerable.Repeat(new
                TimeFrame(site.Key, site.Value), siteTimeFrames));
        }
    }
}
```

Listing 21 – Code source de $\mathsf{F}_{Simulation/Fpersons/Ilness.cs}$

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
 * Version       : 1.0
```

```
 * Description   : Simule la propagation du covid dans un environnement
    vaste représentant une ville.
 */
using System;

namespace CovidPropagation
{
    /// <summary>
    /// Maladie impactant la résistance au virus d'une personne.
    /// </summary>
    class Ilness
    {
        private const int MIN_TIME_BEFORE_DESAPEARING = 1;
        private const int MAX_TIME_BEFORE_DESAPEARING = 7;

        private const int FIRST_STAGE_AGE = 20;
        private const int FIRST_STAGE_MIN_ATTACK = 3;
        private const int FIRST_STAGE_MAX_ATTACK = 6;

        private const int SECOND_STAGE_AGE = 30;
        private const int SECOND_STAGE_MIN_ATTACK = 5;
        private const int SECOND_STAGE_MAX_ATTACK = 9;

        private const int THIRD_STAGE_AGE = 40;
        private const int THIRD_STAGE_MIN_ATTACK = 5;
        private const int THIRD_STAGE_MAX_ATTACK = 13;

        private const int FOURTH_STAGE_AGE = 60;
        private const int FOURTH_STAGE_MIN_ATTACK = 5;
        private const int FOURTH_STAGE_MAX_ATTACK = 16;

        private const int FIFTH_STAGE_AGE = 70;
        private const int FIFTH_STAGE_MIN_ATTACK = 5;
        private const int FIFTH_STAGE_MAX_ATTACK = 19;

        private const int SIXTH_STAGE_MIN_ATTACK = 5;
        private const int SIXTH_STAGE_MAX_ATTACK = 21;

        int timeBeforeDesapearing; // En période
        int attack;
        public int Attack { get => attack; set => attack = value; }
        public Ilness(int age, Random rdm)
        {
            timeBeforeDesapearing =
               rdm.Next(MIN_TIME_BEFORE_DESAPEARING,
               MAX_TIME_BEFORE_DESAPEARING) *
               GlobalVariables.NUMBER_OF_TIMEFRAME; // entre 7 jours et
               30 (Temporaire)
            Attack = CalculateAttack(age, rdm); // Entre 5 et 20, en
               fonction de l'âge
        }

        /// <summary>
        /// ID Documentation : Ilness_Impact
        /// Calcul à quel point la maladie baisse les défenses d'une
```

```
        personne en fonction de son âge.
    /// </summary>
    /// <param name="age">Age de l'individu.</param>
    /// <param name="rdm">Chances que la maladie soit grave.</param>
    /// <returns>"Puissance" de la maladie.</returns>
    private int CalculateAttack(int age, Random rdm)
    {
        // Entre 5 et 20
        // Plus age est élevé plus c'est haut
        int result;
        if (age < FIRST_STAGE_AGE)
        {
            result = rdm.Next(FIRST_STAGE_MIN_ATTACK,
                FIRST_STAGE_MAX_ATTACK);
        }
        else if (age < SECOND_STAGE_AGE)
        {
            result = rdm.Next(SECOND_STAGE_MIN_ATTACK,
                SECOND_STAGE_MAX_ATTACK);
        }
        else if (age < THIRD_STAGE_AGE)
        {
            result = rdm.Next(THIRD_STAGE_MIN_ATTACK,
                THIRD_STAGE_MAX_ATTACK);
        }
        else if (age < FOURTH_STAGE_AGE)
        {
            result = rdm.Next(FOURTH_STAGE_MIN_ATTACK,
                FOURTH_STAGE_MAX_ATTACK);
        }
        else if(age < FIFTH_STAGE_AGE)
        {
            result = rdm.Next(FIFTH_STAGE_MIN_ATTACK,
                FIFTH_STAGE_MAX_ATTACK);
        }
        else
        {
            result = rdm.Next(SIXTH_STAGE_MIN_ATTACK,
                SIXTH_STAGE_MAX_ATTACK);
        }
        return result;
    }

    /// <summary>
    /// Décrémente le temps avant que la maladie ne disparaisse.
    /// </summary>
    public void DecrementTimeBeforeDesapearance(int decrement)
    {
        timeBeforeDesapearing -= decrement;
    }

    /// <summary>
    /// Informe si la maladie doit disparaitre ou non.
    /// </summary>
    public bool Desapear()
```

```
        {
            return timeBeforeDesapearing <= 0;
        }
    }
}
```

Listing 22 – Code source de $\mathsf{F}_{Simulation/Fpersons/Mask.cs}$

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste représentant une ville.
 */

using System;

namespace CovidPropagation
{
    /// <summary>
    /// Différents type de masques ayant des propriétés différentes.
    /// </summary>
    public enum TypeOfMask
    {
        N95 = 0,
        Surgical = 1,
        FaceShield = 2,
        Cloth = 3
    }

    class Mask
    {
        private const double CLOTH_MASK_EXHALATION_EFFICIENCY = 0.5d;
        private const double CLOTH_MASK_INHALATION_EFFICIENCY_MIN =
            0.2d;
        private const double CLOTH_MASK_INHALATION_EFFICIENCY_MAX =
            0.5d;

        private const double SURGICAL_MASK_EXHALATION_EFFICIENCY =
            0.65d;
        private const double SURGICAL_MASK_INHALATION_EFFICIENCY_MIN =
            0.3d;
        private const double SURGICAL_MASK_INHALATION_EFFICIENCY_MAX =
            0.5d;


        private const double FACE_SHIELD_MASK_EXHALATION_EFFICIENCY =
            0.23d;
        private const double FACE_SHIELD_MASK_INHALATION_EFFICIENCY =
            0.23d;


        private const double N95_MASK_EXHALATION_EFFICIENCY = 0.9d;
```

```
        private const double N95_MASK_INHALATION_EFFICIENCY = 0.9d;


        private double _exhalationMaskEfficiency;
        private double _inhalationMaskEfficiency;
        private Random rdm;
        public double ExhalationMaskEfficiency { get =>
            _exhalationMaskEfficiency; }
        public double InhalationMaskEfficiency { get =>
            _inhalationMaskEfficiency; }

        public Mask(TypeOfMask typeOfMask)
        {
            rdm = GlobalVariables.rdm;
            switch (typeOfMask)
            {
                default:
                case TypeOfMask.Cloth:
                    _exhalationMaskEfficiency =
                        CLOTH_MASK_EXHALATION_EFFICIENCY;
                    _inhalationMaskEfficiency =
                        rdm.NextDoubleInclusive(CLOTH_MASK_INHALATION_EFFICIENCY_MIN
                        CLOTH_MASK_INHALATION_EFFICIENCY_MAX);
                    break;
                case TypeOfMask.Surgical:
                    _exhalationMaskEfficiency =
                        SURGICAL_MASK_EXHALATION_EFFICIENCY;
                    _inhalationMaskEfficiency =
                        rdm.NextDoubleInclusive(SURGICAL_MASK_INHALATION_EFFICIENCY_M
                        SURGICAL_MASK_INHALATION_EFFICIENCY_MAX);
                    break;
                case TypeOfMask.FaceShield:
                    _exhalationMaskEfficiency =
                        FACE_SHIELD_MASK_EXHALATION_EFFICIENCY;
                    _inhalationMaskEfficiency =
                        FACE_SHIELD_MASK_INHALATION_EFFICIENCY;
                    break;
                case TypeOfMask.N95:
                    _exhalationMaskEfficiency =
                        N95_MASK_EXHALATION_EFFICIENCY;
                    _inhalationMaskEfficiency =
                        N95_MASK_INHALATION_EFFICIENCY;
                    break;
            }
        }
    }
}
```

Listing 23 – Code source de F$_{Simulation/Fpersons/Planning.cs}$

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
 * Version       : 1.0
```

```
 * Description    : Simule la propagation du covid dans un environnement
    vaste représentant une ville.
 */
using System;
using System.Collections.Generic;

namespace CovidPropagation
{
    /// <summary>
    /// ID Documentation : Planning_Class
    /// Gère le planning des individus. Composé de Jours, qui sont
       composés de périodes, qui sont composé d'activités.
    /// </summary>
    public class Planning
    {
        private const int SUNDAY_INDEX = 6;
        private const int SATURDAY_INDEX = 5;
        private const int FRIDAY_INDEX = 4;

        private const double WORK_ON_SUNDAY_PROBABILITY = 0.1;
        private const double WORK_ON_SATURDAY_PROBABILITY = 0.3;
        private Day[] _days = new Day[GlobalVariables.NUMBER_OF_DAY];
        public Day[] Days { get => _days; }

        /// <summary>
        /// ID Documentation : Planing_Creation
        /// </summary>
        public Planning(Dictionary<SiteType, List<Site>> personSites,
           int nbWorkDays)
        {
            bool[] workDays = new bool[GlobalVariables.NUMBER_OF_DAY];
            Random rdm = GlobalVariables.rdm;
            // Calcul les jours de la semaine ou l'individu travail
            if (nbWorkDays > 0 &&
               rdm.NextBoolean(WORK_ON_SUNDAY_PROBABILITY))
            {
                workDays[SUNDAY_INDEX] = true;
                nbWorkDays--;
            }
            if (nbWorkDays > 0 &&
               rdm.NextBoolean(WORK_ON_SATURDAY_PROBABILITY))
            {
                workDays[SATURDAY_INDEX] = true;
                nbWorkDays--;
            }

            while (nbWorkDays > 0)
            {
                workDays[rdm.NextInclusive(0, FRIDAY_INDEX)] = true;
                nbWorkDays--;
            }

            for (int i = 0; i < _days.Length; i++)
            {
                if (workDays[i])
```

```
                    _days[i] = new Day(personSites, true);
                else
                    _days[i] = new Day(personSites, false);
            }
        }

        /// <summary>
        /// Récupère une activité au jour et à la période demandée.
        /// </summary>
        /// <param name="dayOfWeek">int du jour de la semaine</param>
        /// <param name="timeFrameOfDay">int de la période de la
            journée</param>
        /// <returns>L'activité au jour et à la période
            demandé</returns>
        public Site GetActivity(int dayOfWeek, int timeFrameOfDay)
        {
            return _days[dayOfWeek].GetActivity(timeFrameOfDay);
        }

        /// <summary>
        /// Récupère une activité au jour et à la période actuelle.
        /// </summary>
        /// <returns>L'activité actuelle</returns>
        public Site GetActivity()
        {
            return _days[TimeManager.CurrentDay].GetCurrentActivity();
        }

        /// <summary>
        /// Récupère l'occupation actuel de l'individu durant cette
            activité.
        /// </summary>
        /// <returns>Occupation de l'individu durant l'activité
            actuelle.</returns>
        public SitePersonStatus PersonTypeInActivity()
        {
            return
                _days[TimeManager.CurrentDay].GetCurrentPersonTypeInActivity();
        }

        /// <summary>
        /// Récupère la prochaine activité.
        /// </summary>
        /// <returns>L'activité qui sera réalisé après un tick</returns>
        public Site GetNextActivity()
        {
            int[] nextTimeFrame = TimeManager.GetNextTimeFrame();
            return
                _days[nextTimeFrame[0]].GetActivity(nextTimeFrame[1]);
        }
    }
}
```

Listing 24 – Code source de F*Simulation/TimeManager.cs*

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste représentant une ville.
 */
using System;

namespace CovidPropagation
{
    static class TimeManager
    {
        private static int currentDay;
        private static int currentTimeFrame;
        private static string[] daysOfWeek = new string[] { "Lundi",
            "Mardi", "Mercredi", "Jeudi", "Vendredi", "Samedi",
            "Dimanche"};

        public static int CurrentDay { get => currentDay; }
        public static string CurrentDayString { get =>
            daysOfWeek[currentDay]; }
        public static int CurrentTimeFrame { get => currentTimeFrame; }
        public static string CurrentHour { get => GetTime(); }

        public static void Init()
        {
            currentDay = 0;
            currentTimeFrame = 0;
        }

        /// <summary>
        /// Change la période actuelle pour passer à la suivante.
        /// </summary>
        public static void NextTimeFrame()
        {
            // Incrémente les périodes temps qu'elles ne dépassent pas
               la limite.
            if (currentTimeFrame <
               GlobalVariables.NUMBER_OF_TIMEFRAME-1)
            {
                currentTimeFrame++;
            }
            else
            {
                // Si elles dépassent la limite, on change de jour et
                   on vérifie la même chose pour eux.
                currentTimeFrame = 0; // Réinitialisation de la période
                   car changement de jour
                if (currentDay < GlobalVariables.NUMBER_OF_DAY-1)
                {
                    currentDay++;
                }
                else
```

```
                {
                    currentDay = 0;
                }
            }
        }

        /// <summary>
        /// Récupère la prochaine période sans altérer le temps.
        /// </summary>
        /// <returns>Tableau int contenant le jour et la période du
           prochain Tick.</returns>
        public static int[] GetNextTimeFrame()
        {
            // Identique à NextTimeFrame sauf qu'on utilise des
               variables encapsulées.
            int tempCurrentTimeFrame = currentTimeFrame;
            int tempCurrentDay = currentDay;
            if (tempCurrentTimeFrame <
               GlobalVariables.NUMBER_OF_TIMEFRAME - 1)
            {
                tempCurrentTimeFrame++;
            }
            else
            {
                tempCurrentTimeFrame = 0;
                if (tempCurrentDay < GlobalVariables.NUMBER_OF_DAY - 1)
                {
                    tempCurrentDay++;
                }
                else
                {
                    tempCurrentDay = 0;
                }
            }

            return new int[] { tempCurrentDay, tempCurrentTimeFrame };
        }

        /// <summary>
        /// Convertit les périodes actuelles en heures. Retourne le
           résultat en string.
        /// </summary>
        /// <returns>Résultat en string.</returns>
        private static string GetTime()
        {
            float time = currentTimeFrame *
               GlobalVariables.DURATION_OF_TIMEFRAME / 60f;
            int hours = (int)Math.Truncate(time);
            int minutes = (int)((time - hours) * 60);
            return $"{hours.ToString("00")}:{minutes.ToString("00")}";
        }
    }
}
```

Listing 25 – Code source de F$_{S}imulation/F_{S}ites/Outside.cs$

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste représentant une ville.
 */

namespace CovidPropagation
{
    /// <summary>
    /// Lieu spécial représentant le monde à l'extérieur des bâtiments.
    /// </summary>
    class Outside : Site
    {
        private const int LENGTH = 1000000;
        private const int WIDTH = 1000000;
        private const int HEIGHT = 1000000;
        private const double VENTILATION_WITH_OUTSIDE = 20;

        private static SiteType[] outsideTypes = new SiteType[] {
            SiteType.Transport, SiteType.Eat };
        public Outside(double length = LENGTH,
                       double width = WIDTH,
                       double height = HEIGHT,
                       double ventilationWithOutside =
                           VENTILATION_WITH_OUTSIDE,
                       double additionalControlMeasures =
                           GlobalVariables.BUILDING_ADDITIONAL_CONTROL_MEASURES)
                           :
                base(outsideTypes, length, width, height,
                    ventilationWithOutside, additionalControlMeasures)
        {

        }
    }
}
```

Listing 26 – Code source de F$_{S}imulation/F_{S}ites/Company.cs$

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste représentant une ville.
 */

namespace CovidPropagation
{
    public class Company : WorkSite
    {
```

```
        private const int LENGTH = 150;
        private const int WIDTH = 150;
        private const int HEIGHT = 4;
        private const int NUMBER_OF_WORK_PLACE = 150;
        private static SiteType[] companyTypes = new SiteType[] {
            SiteType.Eat, SiteType.WorkPlace };
        public Company(double length = LENGTH,
                       double width = WIDTH,
                       double height = HEIGHT,
                       double ventilationWithOutside =
                           GlobalVariables.BUILDING_VENTILATION_WITH_OUTSIDE,
                       double additionalControlMeasures =
                           GlobalVariables.BUILDING_ADDITIONAL_CONTROL_MEASURES)
                           :
                base(companyTypes, length, width, height,
                    ventilationWithOutside, additionalControlMeasures,
                    NUMBER_OF_WORK_PLACE)
        {

        }

        public Company() : this(GlobalVariables.BUILDING_LENGTH,
                  GlobalVariables.BUILDING_WIDTH,
                  GlobalVariables.BUILDING_HEIGHT,
                  GlobalVariables.BUILDING_VENTILATION_WITH_OUTSIDE,
                  GlobalVariables.BUILDING_ADDITIONAL_CONTROL_MEASURES)
        {

        }
    }
}
```

Listing 27 – Code source de $\mathsf{F}_{S}simulation/F_{S}ites/Home.cs$

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste représentant une ville.
 */

namespace CovidPropagation
{
    public class Home : Site
    {
        private const int LENGTH = 10;
        private const int WIDTH = 10;
        private const int HEIGHT = 2;
        private static SiteType[] homeTypes = new SiteType[] {
            SiteType.Home, SiteType.Eat };
        public Home(double length = LENGTH,
                       double width = WIDTH,
                       double height = HEIGHT,
```

```
                            double ventilationWithOutside =
                                GlobalVariables.BUILDING_VENTILATION_WITH_OUTSIDE,
                            double additionalControlMeasures =
                                GlobalVariables.BUILDING_ADDITIONAL_CONTROL_MEASURES)
                                :
                base(homeTypes, length, width, height,
                    ventilationWithOutside, additionalControlMeasures)
        {

        }
    }
}
```

Listing 28 – Code source de $\mathsf{F}_{S}imulation/F_{S}ites/Supermarket.cs$

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 29.04.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste tel qu'une ville.
 */

namespace CovidPropagation
{
    public class Supermarket : WorkSite
    {
        private const int LENGTH = 50;
        private const int WIDTH = 40;
        private const int HEIGHT = 5;
        private const int NUMBER_OF_WORK_PLACE = 15;

        private static SiteType[] supermarketTypes = new SiteType[] {
            SiteType.Store, SiteType.WorkPlace };
        public Supermarket(double length = LENGTH,
                           double width = WIDTH,
                           double height = HEIGHT,
                           double ventilationWithOutside =
                               GlobalVariables.BUILDING_VENTILATION_WITH_OUTSIDE,
                           double additionalControlMeasures =
                               GlobalVariables.BUILDING_ADDITIONAL_CONTROL_MEASURES)
                               :
                base(supermarketTypes, length, width, height,
                    ventilationWithOutside, additionalControlMeasures,
                    NUMBER_OF_WORK_PLACE)
        {

        }

        public Supermarket() : this(GlobalVariables.BUILDING_LENGTH,
                  GlobalVariables.BUILDING_WIDTH,
                  GlobalVariables.BUILDING_HEIGHT,
                  GlobalVariables.BUILDING_VENTILATION_WITH_OUTSIDE,
                  GlobalVariables.BUILDING_ADDITIONAL_CONTROL_MEASURES)
```

```
        {

        }
    }
}
```

Listing 29 – Code source de $\mathsf{F}_{Simulation/F_{Sites}/Site.cs}$

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste représentant une ville.
 */
using System;
using System.Collections.Generic;
using System.Linq;

namespace CovidPropagation
{
    /// <summary>
    /// ID Documentation : Site_Class
    /// Lieu dans lequel des personnes entrent et sortent et propageant
       le virus.
    /// </summary>
    public class Site
    {
        private const double DURATION_OF_HOUR = 60;

        List<Person> persons;
        bool hasEnvironnementChanged;
        double averageQuantaExhalationRate;
        SiteType[] types;
        bool clientsMustWearMasks;
        bool workersMustWearMasks;
        AerosolTransmissionData aerosolDatas;

        #region probability
        // Size
        protected double length;
        protected double width;
        protected double height;
        protected double air;
        protected double volume;

        // Air
        protected double temperature; // En Celsius
        protected double humidity; // %

        // Ventilation & deposition
        protected double ventilationWithOutside;
        protected double decayRateOfVirus;
        protected double depositionOnSurfaceRate;
```

```
        protected double additionalControlMeasures;
        private double sumFirstOrderLossRate;

        protected double ventilationPerPersonRate;

        private double probabilityOfInfection;
        double fractionPersonsWithMask;
        double exhalationMaskEfficiency;
        double netEmissionRate;
        double avgQuantaConcentration;
        double quantaInhaledPerPerson;

        protected int nbPersons;
        protected int nbInfectiousPersons;
        protected double fractionOfImmune;
        protected int nbPersonsWithMask;
        protected double inhalationMaskEfficiency;
        protected double probabilityOfBeingInfectious;
        protected double quantaExhalationRateOfInfected;

        protected double probabilityOfOneInfection;
        protected double nbOfInfectiousPersons;
        protected double virusAraisingCases;

        public SiteType[] Type { get => types; }
        public int NbPersons { get => nbPersons; set => nbPersons =
            value; }
        public int NbInfectiousPersons { get => nbInfectiousPersons;
            set => nbInfectiousPersons = value; }
        public double FractionOfImmune { get => fractionOfImmune; set
            => fractionOfImmune = value; }
        public int NbPersonsWithMask { get => nbPersonsWithMask; set =>
            nbPersonsWithMask = value; }
        public double InhalationMaskEfficiency { get =>
            inhalationMaskEfficiency; set => inhalationMaskEfficiency =
            value; }
        public double ProbabilityOfBeingInfectious { get =>
            probabilityOfBeingInfectious; set =>
            probabilityOfBeingInfectious = value; }
        public double QuantaExhalationRateOfInfected { get =>
            quantaExhalationRateOfInfected; set =>
            quantaExhalationRateOfInfected = value; }
        public double ProbabilityOfOneInfection { get =>
            probabilityOfOneInfection; set => probabilityOfOneInfection
            = value; }
        public double NbOfInfectiousPersons { get =>
            nbOfInfectiousPersons; set => nbOfInfectiousPersons = value;
            }
        public double VirusAraisingCases { get => virusAraisingCases;
            set => virusAraisingCases = value; }
        public double FractionPersonsWithMask { get =>
            fractionPersonsWithMask; set => fractionPersonsWithMask =
            value; }
        public double ExhalationMaskEfficiency { get =>
            exhalationMaskEfficiency; set => exhalationMaskEfficiency =
```

```
        value; }
    public double NetEmissionRate { get => netEmissionRate; set =>
        netEmissionRate = value; }
    public double AvgQuantaConcentration { get =>
        avgQuantaConcentration; set => avgQuantaConcentration =
        value; }
    public double QuantaInhaledPerPerson { get =>
        quantaInhaledPerPerson; set => quantaInhaledPerPerson =
        value; }
    public double SumFirstOrderLossRate { get =>
        sumFirstOrderLossRate; set => sumFirstOrderLossRate = value;
        }
    public double ProbabilityOfInfection { get =>
        probabilityOfInfection; set => probabilityOfInfection =
        value; }
    public double AverageQuantaExhalationRate { get =>
        averageQuantaExhalationRate; set =>
        averageQuantaExhalationRate = value; }
    public bool HasEnvironnementChanged { get =>
        hasEnvironnementChanged; set => hasEnvironnementChanged =
        value; }

    #endregion

    public Site(SiteType[] types, double length, double width,
        double height,
                double ventilationWithOutside = 0.7d, double
                    additionalControlMeasures = 0)
    {
         this.types = types;
         this.length = length;
         this.width = width;
         this.height = height;

         this.ventilationWithOutside = ventilationWithOutside;
         this.decayRateOfVirus = Virus.DecayRateOfVirus;// récupérer
            du virus
         this.depositionOnSurfaceRate =
            Virus.DepositionOnSurfaceRate; // récupérer du virus
         this.additionalControlMeasures = additionalControlMeasures;

         this.SumFirstOrderLossRate = this.ventilationWithOutside +
            decayRateOfVirus + depositionOnSurfaceRate +
            this.additionalControlMeasures;
         air = this.length * this.width;
         volume = air * this.height;

         persons = new List<Person>();
         HasEnvironnementChanged = true;
         AverageQuantaExhalationRate =
            GlobalVariables.AVERAGE_QUANTA_EXHALATION;
    }

    /// <summary>
    /// Récupère les taux de probabilité d'infection d'une personne
```

```
        dans ce lieu.
    /// </summary>
    /// <returns>Taux de probabilité d'infection actuellement dans
        ce lieu</returns>
    public double GetProbabilityOfInfection()
    {
        return ProbabilityOfInfection;
    }

    /// <summary>
    /// ID Documentation : Calculate_Probability
    /// Calcul le taux de probabilité d'infection d'une personne
        dans ce lieu.
    /// </summary>
    public void CalculateprobabilityOfInfection()
    {
        if (HasEnvironnementChanged && persons.Count > 0)
        {
            if
                (Virus.IsTransmissibleBy(typeof(AerosolTransmission)))
            {
                AerosolTransmission aerosolTransmission =
                    Virus.GetTransmission(typeof(AerosolTransmission))
                    as AerosolTransmission;
                NbPersons = persons.Count;
                NbInfectiousPersons =
                    CountNumberInfectiousPersons();
                FractionOfImmune = GetFractionOfImmune();

                NbPersonsWithMask = CountPersonsWithMask();
                FractionPersonsWithMask =
                    GetFractionPersonsWithMask();

                InhalationMaskEfficiency =
                    GetInhalationMaskEfficiency();
                ExhalationMaskEfficiency =
                    GetExhalationMaskEfficiency();

                ProbabilityOfBeingInfectious =
                    GetprobabilityOfBeingInfectious();

                QuantaExhalationRateOfInfected =
                    GetQuantaExhalationRateofInfected();
                NetEmissionRate =
                    GetNetEmissionRate(QuantaExhalationRateOfInfected,
                    ExhalationMaskEfficiency,
                                                        FractionPersonsWithM
                                                            NbInfectiousPerso

                double eventDuration =
                    GlobalVariables.DURATION_OF_TIMEFRAME /
                    DURATION_OF_HOUR;
                AvgQuantaConcentration =
                    GetAverageQuantaConcentration(NetEmissionRate,
                    eventDuration);
```

```
                QuantaInhaledPerPerson =
                    GetQuantaInhaledPerPerson(AvgQuantaConcentration,
                    eventDuration, InhalationMaskEfficiency,
                    FractionPersonsWithMask);

                aerosolDatas =
                    aerosolTransmission.CalculateRisk(NbPersons,
                    NbInfectiousPersons, FractionOfImmune,









                ProbabilityOfInfection =
                    aerosolDatas.ProbabilityOfInfection.SetValueIfNaN();
                ProbabilityOfOneInfection =
                    aerosolDatas.ProbabilityOfOneInfection.SetValueIfNaN();
                NbOfInfectiousPersons =
                    aerosolDatas.NOfInfectiousPersons.SetValueIfNaN();
                VirusAraisingCases =
                    aerosolDatas.VirusAraisingCases.SetValueIfNaN();
            }
        }
        HasEnvironnementChanged = false;
    }

    /// <summary>
    /// Fait entrer une personne sur le site.
    /// L'ajoutant à la liste de personne et informant le site
       qu'il doit recalculer le taux de propagation dans le lieu.
    /// Lui demande de porter le masque si nécessaire.
    /// </summary>
    /// <param name="personEntering">La personne qui entre</param>
    public void Enter(Person personEntering, SitePersonStatus
       sitePersonStatus)
    {
        persons.Add(personEntering);
        HasEnvironnementChanged = true;

        if (sitePersonStatus == SitePersonStatus.Client &&
            clientsMustWearMasks)
            personEntering.PutMaskOn();
        else if(sitePersonStatus == SitePersonStatus.Worker &&
            workersMustWearMasks)
            personEntering.PutMaskOn();
    }

    /// <summary>
    /// Fait sortir une personne du site.
    /// La retirant de la liste de personne et informant le site
       qu'il doit recalculer le taux de propagation dans le lieu.
```

```
        /// </summary>
        /// <param name="personLeaving">La personne quittant le
            site</param>
        public void Leave(Person personLeaving)
        {
            persons.Remove(personLeaving);
            HasEnvironnementChanged = true;
            personLeaving.RemoveMask();
        }

        /// <summary>
        /// Applique oui ou non la mesure du port des masques aux
            client et/ou aux employés.
        /// </summary>
        /// <param name="clientsMustWearMasks">True si les clients
            doivent porter le masque.</param>
        /// <param name="workersMustWearMasks">True si les employés
            doivent porter le masque</param>
        public void SetMaskMeasure(bool clientsMustWearMasks, bool
            workersMustWearMasks)
        {
            this.clientsMustWearMasks = clientsMustWearMasks;
            this.workersMustWearMasks = workersMustWearMasks;
        }

        /// <summary>
        /// Définit si les distanciation social sont en place ou non.
        /// Change les mesures de controles additionnels en fonction du
            paramètre.
        /// </summary>
        public void SetDistanciations(bool isDistanciationSet)
        {
            additionalControlMeasures =
                isDistanciationSet.ConvertToInt();
            SumFirstOrderLossRate = ventilationWithOutside +
                decayRateOfVirus + depositionOnSurfaceRate +
                additionalControlMeasures;
        }

        #region Calculs

        /// <summary>
        /// Compte le nombre d'infecté dans le bâtiments.
        /// </summary>
        /// <returns>Nombre d'infectés.</returns>
        private int CountNumberInfectiousPersons()
        {
            return persons.Where(p => (int)p.CurrentState >=
                (int)PersonState.Infectious).Count();
        }

        /// <summary>
        /// Compte le pourcentage d'individus infectés dans le bâtiment.
        /// </summary>
        /// <returns>Pourcentage d'infecté dans le bâtiment.</returns>
```

```
        private double GetFractionOfImmune()
        {
            return (double)persons.Where(p => p.CurrentState ==
                PersonState.Immune).Count() / NbPersons * 100d;
        }

        /// <summary>
        /// Compte le nombre de personnes qui portent le masque dans le
            bâtiment.
        /// </summary>
        /// <returns>Nombre de personnes qui portent le
            masque.</returns>
        private int CountPersonsWithMask()
        {
            return persons.Where(p => p.HasMask).Count();
        }

        /// <summary>
        /// Récupère le pourcentage de personnes qui porte le masque
            dans le bâtiment.
        /// </summary>
        /// <returns>Pourcentage de personnes portant le
            masque.</returns>
        private double GetFractionPersonsWithMask()
        {
            return NbPersonsWithMask / (NbPersons / 100d) / 100d;
                //fractionPersonsWithMask =
                fractionPersonsWithMask.SetValueIfNaN();
        }

        /// <summary>
        /// Récupère la moyenne d'éfficacité de filtrage, lors d'une
            inhalation, des masque présents dans le bâtiment.
        /// </summary>
        /// <returns>Moyenne d'efficacité des masques</returns>
        private double GetInhalationMaskEfficiency()
        {
            double inhalationMaskEfficiency = (double)persons.Where(p
                => p.HasMask)
                                                              .Sum(p =>
                                                                 p.InhalationMaskEffi
                                                              /
                                                                 NbPersonsWithMask;
            return inhalationMaskEfficiency.SetValueIfNaN();
        }

        /// <summary>
        /// Récupère la moyenne d'éfficacité de filtrage, lors d'une
            exhalation, des masque présents dans le bâtiment.
        /// </summary>
        /// <returns>Moyenne d'efficacité des masques</returns>
        private double GetExhalationMaskEfficiency()
        {
            double exhalationMaskEfficiency = (double)persons.Where(p
                => p.HasMask)
```

```
                                                        .Sum(p =>
                                                            p.ExhalationMaskEffi
                                                        /
                                                            NbPersonsWithMask;
        return exhalationMaskEfficiency.SetValueIfNaN();
    }

    /// <summary>
    /// Récupère la probabilité qu'une personne soit infectieuse.
    /// </summary>
    /// <returns>Probabilité qu'une personne soit
        infectieuse.</returns>
    private double GetprobabilityOfBeingInfectious()
    {
        double probabilityOfBeingInfectious =
            (double)persons.Where(p => (int)p.CurrentState >=
            (int)PersonState.Infectious).Count() /
            (double)nbPersons; // A modifier pour entrer en accord
            avec la simulation
        return probabilityOfBeingInfectious.SetValueIfNaN();
    }

    /// <summary>
    /// Récupère la moyenne de quanta exhalé par les personnes
        infectées et contagieuses.
    /// </summary>
    /// <returns>Moyenne des quantas exhalés</returns>
    private double GetQuantaExhalationRateofInfected()
    {
        double quantaExhalationRateOfInfected = persons.Where(p =>
            (int)p.CurrentState >= (int)PersonState.Infectious)
                                                        .Sum(p =>
                                                            p.QuantaExhalationRat
                                                            * (1 -
                                                            p.ExhalationMaskEffic
                                                            *
                                                            p.HasMask.ConvertToIn
                                                            /
                                                            NbInfectiousPersons;

        return quantaExhalationRateOfInfected.SetValueIfNaN();
    }

    /// <summary>
    /// Récupère les emissions de quantas en prenom en compte le
        nombre d'infecté ainsi que le nombre de personne portant le
        masque et leur efficacités.
    /// </summary>
    /// <returns>Emissions de quantas</returns>
    private double GetNetEmissionRate(double
        quantaExhalationRateOfInfected, double
        exhalationMaskEfficiency, double fractionPersonsWithMask,
        double nbInfectiousPersons)
    {
        return quantaExhalationRateOfInfected * (1 -
```

```
                exhalationMaskEfficiency * fractionPersonsWithMask) *
                nbInfectiousPersons;
        }

        /// <summary>
        /// Récupère la moyenne de concentration de quanta durant la
            durée d'un évènement.
        /// </summary>
        /// <returns>Moyenne de concentration de quanta durant une
            période T.</returns>
        private double GetAverageQuantaConcentration(double
            netEmissionRate, double eventDuration)
        {
            double avgQuantaConcentration = Math.Abs(netEmissionRate) /
                                            Math.Abs(SumFirstOrderLossRate)
                                                /
                                            volume *
                                            ( 1 - (1 /
                                                Math.Abs(SumFirstOrderLossRate)
                                                / eventDuration) * (1 -
                                                Math.Exp(-SumFirstOrderLossRate
                                                * eventDuration)));

            return avgQuantaConcentration.SetValueIfNaN();
        }

        /// <summary>
        /// Calcul la moyenne de quantas inhalé par personne durant une
            période en prenant en compte les masques et leur efficacité.
        /// </summary>
        /// <returns>Moyenne de quantas inhalé par personnes.</returns>
        private double GetQuantaInhaledPerPerson(double
            avgQuantaConcentration, double eventDuration, double
            inhalationMaskEfficiency, double fractionPersonsWithMask)
        {
            double quantaInhaledPerPerson = avgQuantaConcentration *
                Math.Abs(SumFirstOrderLossRate) * eventDuration * (1 -
                inhalationMaskEfficiency * fractionPersonsWithMask);
            return quantaInhaledPerPerson.SetValueIfNaN();
        }

        #endregion
    }
}
```

Listing 30 – Code source de $\mathsf{F}_{Simulation/Fsites/Bus.cs}$

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 06.05.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste tel qu'une ville.
 */
```

```
namespace CovidPropagation
{
    /// <summary>
    /// Lieu similaire au voiture mais comprenant un chauffeur et
       plusieurs passagers.
    /// </summary>
    class Bus : WorkSite
    {
        private const int NUMBER_OF_WORK_PLACE = 1;
        private static SiteType[] busTypes = new SiteType[] {
            SiteType.Transport, SiteType.WorkPlace };

        public Bus(double length = GlobalVariables.BUS_LENGTH,
                   double width = GlobalVariables.BUS_WIDTH,
                   double height = GlobalVariables.BUS_HEIGHT,
                   double ventilationWithOutside =
                      GlobalVariables.BUS_VENTILATION_WITH_OUTSIDE,
                   double additionalControlMeasures =
                      GlobalVariables.BUS_ADDITIONAL_CONTROL_MEASURES) :
                  base(busTypes, length, width, height,
                     ventilationWithOutside, additionalControlMeasures,
                     NUMBER_OF_WORK_PLACE)
        {

        }
    }
}
```

Listing 31 – Code source de $\mathsf{F}_{S}imulation/F_{S}ites/Bike.cs$

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste représentant une ville.
 */

namespace CovidPropagation
{
    class Bike : Site
    {
        private static SiteType[] bikeTypes = new SiteType[] {
            SiteType.Transport };

        public Bike(double length = GlobalVariables.CAR_LENGTH,
                    double width = GlobalVariables.CAR_WIDTH,
                    double height = GlobalVariables.CAR_HEIGHT,
                    double ventilationWithOutside =
                       GlobalVariables.CAR_VENTILATION_WITH_OUTSIDE,
                    double additionalControlMeasures =
                       GlobalVariables.CAR_ADDITIONAL_CONTROL_MEASURES) :
                   base(bikeTypes, length, width, height,
```

```
                    ventilationWithOutside, additionalControlMeasures)
        {

        }
    }
}
```

Listing 32 – Code source de $\mathsf{F}_S simulation/F_S ites/Car.cs$

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste représentant une ville.
 */

namespace CovidPropagation
{
    class Car : Site
    {
        private static SiteType[] carTypes = new SiteType[] {
            SiteType.Transport };

        public Car(double length = GlobalVariables.CAR_LENGTH,
                   double width = GlobalVariables.CAR_WIDTH,
                   double height = GlobalVariables.CAR_HEIGHT,
                   double ventilationWithOutside =
                       GlobalVariables.CAR_VENTILATION_WITH_OUTSIDE,
                   double additionalControlMeasures =
                       GlobalVariables.CAR_ADDITIONAL_CONTROL_MEASURES) :
                  base(carTypes, length, width, height,
                       ventilationWithOutside, additionalControlMeasures)
        {

        }
    }
}
```

Listing 33 – Code source de $\mathsf{F}_S simulation/F_S ites/WorkSite.cs$

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste représentant une ville.
 */

namespace CovidPropagation
{
    /// <summary>
    /// Lieu dans lequel les individus travaillent.
```

```
    /// </summary>
    public class WorkSite : Site
    {
        private int _nbPlaces;
        private int _nbWorker;
        public WorkSite(SiteType[] workPlaceTypes,
                        double length,
                        double width,
                        double height,
                        double ventilationWithOutside,
                        double additionalControlMeasures,
                        int nbPlaces) :
                    base(workPlaceTypes, length, width, height,
                       ventilationWithOutside, additionalControlMeasures)
        {
            this._nbPlaces = nbPlaces;
            _nbWorker = 0;
        }

        /// <summary>
        /// Définit s'il y a encore de la place dans le lieu.
        /// </summary>
        /// <returns>S'il a de la place ou non.</returns>
        public bool IsHiring()
        {
            bool result = false;
            if (_nbWorker < _nbPlaces)
            {
                _nbWorker++;
                result = true;
            }
            return result;
        }
    }
}
```

Listing 34 – Code source de $\mathsf{F}_{Simulation/Fsites/Store.cs}$

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 29.04.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste tel qu'une ville.
 */

namespace CovidPropagation
{
    class Store : WorkSite
    {
        private const int LENGTH = 10;
        private const int WIDTH = 10;
        private const int HEIGHT = 3;
        private const int NUMBER_OF_WORK_PLACE = 5;
```

```
        private static SiteType[] storeTypes = new SiteType[] {
            SiteType.WorkPlace };
        public Store(double length = LENGTH,
                             double width = WIDTH,
                             double height = HEIGHT,
                             double ventilationWithOutside =
                                 GlobalVariables.BUILDING_VENTILATION_WITH_OUTSIDE,
                             double additionalControlMeasures =
                                 GlobalVariables.BUILDING_ADDITIONAL_CONTROL_MEASURES)
                                 :
                    base(storeTypes, length, width, height,
                        ventilationWithOutside, additionalControlMeasures,
                        NUMBER_OF_WORK_PLACE)
        {

        }

        public Store() : this(GlobalVariables.BUILDING_LENGTH,
                    GlobalVariables.BUILDING_WIDTH,
                    GlobalVariables.BUILDING_HEIGHT,
                    GlobalVariables.BUILDING_VENTILATION_WITH_OUTSIDE,
                    GlobalVariables.BUILDING_ADDITIONAL_CONTROL_MEASURES)
        {

        }
    }
}
```

Listing 35 – Code source de $F_Simulation/F_Sites/School.cs$

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste représentant une ville.
 */
namespace CovidPropagation
{
    public class School : WorkSite
    {
        private const int LENGTH = 50;
        private const int WIDTH = 70;
        private const int HEIGHT = 5;
        private const int NUMBER_OF_WORK_PLACE = 20;

        private static SiteType[] schoolTypes = new SiteType[] {
            SiteType.Eat, SiteType.WorkPlace };
        public School(double length = LENGTH,
                             double width = WIDTH,
                             double height = HEIGHT,
                             double ventilationWithOutside =
                                 GlobalVariables.BUILDING_VENTILATION_WITH_OUTSIDE,
```

```
                            double additionalControlMeasures =
                                GlobalVariables.BUILDING_ADDITIONAL_CONTROL_MEASURES)
                                :
                    base(schoolTypes, length, width, height,
                        ventilationWithOutside, additionalControlMeasures,
                        NUMBER_OF_WORK_PLACE)
        {

        }

        public School() : this(GlobalVariables.BUILDING_LENGTH,
                               GlobalVariables.BUILDING_WIDTH,
                               GlobalVariables.BUILDING_HEIGHT,
                               GlobalVariables.BUILDING_VENTILATION_WITH_OUTSIDE,
                               GlobalVariables.BUILDING_ADDITIONAL_CONTROL_MEASURES)
        {

        }
    }
}
```

Listing 36 – Code source de $F_Simulation/F_Sites/Hospital.cs$

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 27.04.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste tel qu'une ville.
 */
using System.Collections.Generic;

namespace CovidPropagation
{
    /// <summary>
    /// Lieu pouvant accueillir des cas de covid pour les traiter.
    /// </summary>
    public class Hospital : WorkSite
    {
        private const int LENGTH = 100;
        private const int WIDTH = 50;
        private const int HEIGHT = 10;
        private const int NUMBER_OF_WORK_PLACE = 50;
        private const int NUMBER_MAX_OF_COVID_PATIENT = 50;
        private const int NUMBER_MAX_OF_COVID_PATIENT_EXTREME = 10;

        int nbCovidPatientMax;
        int nbExtremeCovidPatientMax;
        int nbSumCovidPatientMax;
        List<Person> covidPatient;
        private static SiteType[] hospitalTypes = new SiteType[] {
            SiteType.Hospital, SiteType.Eat, SiteType.WorkPlace };

        public Hospital(double length = LENGTH,
```

```
                        double width = WIDTH,
                        double height = HEIGHT,
                        double ventilationWithOutside =
                            GlobalVariables.BUILDING_VENTILATION_WITH_OUTSIDE,
                        double additionalControlMeasures =
                            GlobalVariables.BUILDING_ADDITIONAL_CONTROL_MEASURES)
                            :
            base(hospitalTypes, length, width, height,
                ventilationWithOutside, additionalControlMeasures,
                NUMBER_OF_WORK_PLACE)
    {
        nbCovidPatientMax = NUMBER_MAX_OF_COVID_PATIENT;
        nbExtremeCovidPatientMax =
            NUMBER_MAX_OF_COVID_PATIENT_EXTREME;
        nbSumCovidPatientMax = nbCovidPatientMax +
            nbExtremeCovidPatientMax;
        covidPatient = new List<Person>();
    }

    public Hospital() : this(GlobalVariables.BUILDING_LENGTH,
                             GlobalVariables.BUILDING_WIDTH,
                             GlobalVariables.BUILDING_HEIGHT,
                             GlobalVariables.BUILDING_VENTILATION_WITH_OUTSIDE,
                             GlobalVariables.BUILDING_ADDITIONAL_CONTROL_MEASURE
    {

    }

    /// <summary>
    /// ID Documentation : Enter_Hospital
    /// Fait entrer un individu pour cause de covid dans le
        bâtiment s'il y a de la place
    /// </summary>
    /// <param name="patient">Patient qui essai d'entrer.</param>
    /// <returns>Si le patient a pu entrer ou non.</returns>
    public bool EnterForCovid(Person patient)
    {
        bool result = false;
        if (covidPatient.Count < nbSumCovidPatientMax)
        {
            covidPatient.Add(patient);
            result = true;
        }
        return result;
    }

    /// <summary>
    /// Fait quitter un individu étant admis pour cause de covid.
    /// </summary>
    /// <param name="patient">Patient qui quitte l'hôpital.</param>
    public void LeaveForCovid(Person patient)
    {
        covidPatient.Remove(patient);
    }
```

```
        /// <summary>
        /// Traite le patient. Réduit le nombre de maladies plus
           rapidement que la normale.
        /// Recalcile la résistance au virus du patient.
        /// Vérifie s'il doit quitter l'hôpital car il est remit.
        /// </summary>
        public void TreatPatients()
        {
            covidPatient.ForEach(p => {
                p.GetHospitalTreatment();
                p.RecalculateVirusResistance(2);
                if (p.CurrentState < PersonState.Infected)
                    p.MustLeaveHospital = true;
            });
        }

        /// <summary>
        /// Compte le nombre de patients covid.
        /// </summary>
        /// <returns>Nombre de patient covid.</returns>
        public int CountPatients()
        {
            return covidPatient.Count;
        }
    }
}
```

Listing 37 – Code source de F$_S$*imulation*/*F*$_S$*ites*/*SitePersonStatut.cs*

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste représentant une ville.
 */
namespace CovidPropagation
{
    /// <summary>
    /// Raison pour un individu de se déplacer dans un lieu.
    /// Permet de lui appliquer différentes mesures.
    /// </summary>
    public enum SitePersonStatus
    {
        Client,
        Worker,
        Other
    }
}
```

Listing 38 – Code source de F$_S$*imulation*/*F*$_S$*ites*/*Restaurant.cs*

```
/*
 * Nom du projet : CovidPropagation
```

```
 * Auteur          : Joey Martig
 * Date            : 11.06.2021
 * Version         : 1.0
 * Description     : Simule la propagation du covid dans un environnement
    vaste représentant une ville.
 */

namespace CovidPropagation
{
    class Restaurant : WorkSite
    {
        private const int LENGTH = 13;
        private const int WIDTH = 30;
        private const int HEIGHT = 5;
        private const int NUMBER_OF_WORK_PLACE = 5;

        private static SiteType[] restaurantTypes = new SiteType[] {
           SiteType.Eat, SiteType.WorkPlace };
        public Restaurant(double length = LENGTH,
                          double width = WIDTH,
                          double height = HEIGHT,
                          double ventilationWithOutside =
                              GlobalVariables.BUILDING_VENTILATION_WITH_OUTSIDE,
                          double additionalControlMeasures =
                              GlobalVariables.BUILDING_ADDITIONAL_CONTROL_MEASURES)
                              :
                  base(restaurantTypes, length, width, height,
                     ventilationWithOutside, additionalControlMeasures,
                     NUMBER_OF_WORK_PLACE)
        {

        }

        public Restaurant() : this(GlobalVariables.BUILDING_LENGTH,
                           GlobalVariables.BUILDING_WIDTH,
                           GlobalVariables.BUILDING_HEIGHT,
                           GlobalVariables.BUILDING_VENTILATION_WITH_OUTSIDE,
                           GlobalVariables.BUILDING_ADDITIONAL_CONTROL_MEASURES)
        {

        }
    }
}
```

Listing 39 – Code source de $\mathsf{F}_{Simulation/Fsites/SiteType.cs}$

```
/*
 * Nom du projet : CovidPropagation
 * Auteur          : Joey Martig
 * Date            : 11.06.2021
 * Version         : 1.0
 * Description     : Simule la propagation du covid dans un environnement
    vaste représentant une ville.
 */
```

```
namespace CovidPropagation
{
    /// <summary>
    /// Permet de catégoriser un lieu.
    /// </summary>
    public enum SiteType
    {
        Home,
        Store,
        Eat,
        Transport,
        Hospital,
        WorkPlace,
        School
    }
}
```

Listing 40 – Code source de $\mathsf{F}_{S}simulation/Simulation.cs$

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste représentant une ville.
 */
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Threading.Tasks;

namespace CovidPropagation
{
    public delegate void GUIDataEventHandler(int[] personsNewSite,
       int[] personsNewState);
    public delegate void InitializeGUIEventHandler(DataPopulation
       populationDatas, DataSites siteDatas);
    public delegate void DataUpdateEventHandler(SimulationDatas e);
    public delegate void DispalyChangeEventHandler(SimulationDatas e,
       bool isDisplayChange);

    /// <summary>
    /// ID Documentation : Simulation_Class
    /// </summary>
    public class Simulation : EventArgs
    {
        private const double PROBABILITY_OF_BEING_A_COMPANY = 0.7909d;
        private const double PROBABILITY_OF_BEING_A_STORE = 0.1d;
        private const double PROBABILITY_OF_BEING_A_RESTAURANT = 0.1d;
        private const double PROBABILITY_OF_BEING_A_SCHOOL = 0.0045d;
        private const double PROBABILITY_OF_BEING_A_HOSPITAL = 0.0004d;
        private const double PROBABILITY_OF_BEING_A_SUPERMARKET =
           0.0042d;
```

```
        private const int NUMBER_OF_STORE_PER_PERSON = 3;
        private const int NUMBER_OF_PLACE_TO_EAT_PER_PERSON = 3;

        private const double DEFAULT_PROBABILITY_OF_BEING_MINOR = 0.22d;
        private const double DEFAULT_PROBABILITY_OF_BEING_RETIRED =
            0.14d;

        private const double PROBABILITY_OF_USING_A_CAR = 0.36d;
        private const double PROBABILITY_OF_USING_A_BIKE = 0.37d;
        private const double PROBABILITY_OF_USING_A_BUS = 0.15d;
        private const double PROBABILITY_OF_WALKING = 0.27d; //
            Normalement 0.11 avec les bus
        private const double PERCENTAGE_OF_BUS = 0.085d;

        private const double
            PROBABILITY_OF_BEING_VACCINATED_PER_TIMEFRAME =
            0.00004472222d;
        private const int NUMBER_OF_INFECTIOUS_VALUES_FOR_REPRODUCTION
            = 10;

        private int DELAY_BEFORE_CHECKING_IF_SIMULATION_SHOULD_START =
            100;
        private int MAX_TIME_BEFORE_DISPLAYING_AGAIN = 1000;


        public event DataUpdateEventHandler OnDataUpdate;
        public event DispalyChangeEventHandler OnDisplay;
        public event GUIDataEventHandler OnGUIUpdate;
        public event InitializeGUIEventHandler OnGUIInitialize;

        private Random rdm = new Random();
        private double _probabilityOfBeingInfected;
        private int _nbPersons;

        private Dictionary<SiteType, List<Site>> _sitesDictionnary;
        private List<Site> _sites;
        private Dictionary<Site, int> _sitesIds;
        private Outside _outside;
        private List<Person> _population;
        private Stopwatch _sp;
        private bool _startStop;
        private bool _isInitialized;
        private int _interval;

        SimulationDatas chartsDatas;

        public int Interval { get => _interval; set => _interval =
            value; }
        public bool IsInitialized { get => _isInitialized; }

        public Simulation()
        {
            _sites = new List<Site>();
            _sitesIds = new Dictionary<Site, int>();
```

```
            _sitesDictionnary = new Dictionary<SiteType, List<Site>>();
            _outside = new Outside();
            _population = new List<Person>();
            _sp = new Stopwatch();
            _isInitialized = false;
        }

        /// <summary>
        /// ID documentation: Initialize_Simulation
        /// Initialise la simulation, créé les bâtiments, créé les
            individus, Réinitialise le TimeManager et set les trigger
            des mesures.
        /// </summary>
        /// <param name="probabilityOfBeingInfected">Probabilités qu'un
            individu a d'être infecté lors de sa création</param>
        /// <param name="nbPersons">Nombre d'individus total dans la
            simulation</param>
        public void Initialize()
        {
            _sites.Clear();
            _sitesDictionnary.Clear();
            _population.Clear();
            _isInitialized = true;
            _probabilityOfBeingInfected =
                SimulationGeneralParameters.ProbabilityOfInfected;
            _nbPersons = SimulationGeneralParameters.NbPeople;
            _startStop = true;

            TimeManager.Init();
            Virus.Init();

            // Récupérer depuis les paramètres
            double minorProbability =
                DEFAULT_PROBABILITY_OF_BEING_MINOR;
            double retirementProbability =
                DEFAULT_PROBABILITY_OF_BEING_RETIRED;

            Stopwatch speedTest = new Stopwatch();
            speedTest.Start();
            CreateBuildings();
            speedTest.Stop();
            Debug.WriteLine("Building" + speedTest.ElapsedMilliseconds
                + "  Count" + _sites.Count);
            speedTest.Restart();
            CreatePublicTransports();
            speedTest.Stop();
            Debug.WriteLine("Transport" +
                speedTest.ElapsedMilliseconds);
            speedTest.Restart();

            CreatePopulation(_nbPersons, retirementProbability,
                minorProbability);

            speedTest.Stop();
            Debug.WriteLine("Population " +
```

```
            speedTest.ElapsedMilliseconds + "  Count " +
            _population.Count);
    }

    /// <summary>
    /// ID Documentation : Simulation_Iteration
    /// Itère la simulation.
    /// Créé un "timer" à l'aide des stopwatch pour une meilleur
       précision.
    /// Ordonne à la population de changer d'activité,
    /// aux bâtiment de calculer les chances d'attraper le virus
       dans ceux-ci,
    /// à la population de calculer si elle a été infectée.
    /// </summary>
    public async void Iterate()
    {
        chartsDatas = new SimulationDatas();
        chartsDatas.Initialize();
        int[] personsNewSite = new int[_population.Count];
        int[] personsNewState = new int[_population.Count];

        // Informe si une mesure est active ou non
        bool isMaskMeasureOn = false;
        bool isDistanciationMeasureOn = false;
        bool isQuarantineMeasureOn = false;
        bool isVaccinationMeasureOn = false;

        int nbOfInfectious = 0;
        int nbOfIncubating = 0;
        int nbOfImmune = 0;
        int nbOfHealthy = 0;
        int nbOfDead = 0;
        int indexPerson = 0;

        // Initialise les données du GUI
        OnGUIInitialize?.Invoke(new DataPopulation(
                _population.Count,
                _population.Where(p => p.CurrentState ==
                   PersonState.Infected).Select(p => p.Id).ToArray()
                ),
            new DataSites(
                _sitesDictionnary[SiteType.Home].Count,
                _sitesDictionnary[SiteType.WorkPlace].Where(s =>
                   s.GetType() == typeof(Company)).Count(),
                _sitesDictionnary[SiteType.Hospital].Count,
                _sitesDictionnary[SiteType.Eat].Where(s =>
                   s.GetType() == typeof(Restaurant)).Count(),
                _sitesDictionnary[SiteType.WorkPlace].Where(s =>
                   s.GetType() == typeof(School)).Count(),
                _sitesDictionnary[SiteType.Store].Where(s =>
                   s.GetType() == typeof(Store)).Count(),
                _sitesDictionnary[SiteType.Store].Where(s =>
                   s.GetType() == typeof(Supermarket)).Count()
                )
            );
```

```
int sumEllapsedTime = 0;
Stopwatch sp = new Stopwatch();
// Boucle d'itération de la simulation
while (true)
{
    // Défini si la simulation est en pause ou non
    if (_startStop)
    {
        _sp.Start();

        // Réinitialise les compteur de types d'individus
        nbOfInfectious = 0;
        nbOfIncubating = 0;
        nbOfImmune = 0;
        nbOfHealthy = 0;

        // Applique les mesures ou les retires.
        int nbInfected = _population.Where(p =>
            (int)p.CurrentState >=
            (int)PersonState.Infected).Count();
        if
            (SimulationGeneralParameters.IsMaskMeasuresEnabled)
             isMaskMeasureOn = SetMaskMeasure(nbInfected,
                 isMaskMeasureOn);

        if
            (SimulationGeneralParameters.IsDistanciationMeasuresEnabled)
             isDistanciationMeasureOn =
                 SetDistanciationMeasure(nbInfected,
                 isDistanciationMeasureOn);

        if
            (SimulationGeneralParameters.IsQuarantineMeasuresEnabled)
             isQuarantineMeasureOn =
                 SetQuarantineMeasure(nbInfected,
                 isQuarantineMeasureOn);

        if
            (SimulationGeneralParameters.IsVaccinationMeasuresEnabled)
             isVaccinationMeasureOn =
                 SetVaccinationMeasure(nbInfected,
                 isVaccinationMeasureOn);

        TimeManager.NextTimeFrame();

        // Change l'activité de la population, change de
            lieu pour le GUI, applique la vaccination si
            celle-ci est appliquée.
        indexPerson = 0;
        _population.ForEach(p => {
             Site newSite = p.ChangeActivity();
             if (_sitesIds.ContainsKey(newSite))
                 personsNewSite[indexPerson] =
                     _sitesIds[newSite];
```

```
            if (isVaccinationMeasureOn && rdm.NextDouble()
                <=
                PROBABILITY_OF_BEING_VACCINATED_PER_TIMEFRAME
                && (int)p.CurrentState <
                (int)PersonState.Infected)
                 p.GetVaccinated();
            indexPerson++;
        });
        // Calcul les probabilités d'infections dans les
            lieux et traites les patients à l'hôpital.
        _sites.ForEach(p =>
            p.CalculateprobabilityOfInfection());
        _sitesDictionnary[SiteType.Hospital].ForEach(h =>
            ((Hospital)h).TreatPatients());

        // Change l'état de l'individu s'il a été infectés
            et change l'état pour le GUI.
        indexPerson = 0;
        _population.ForEach(p => {
            personsNewState[indexPerson] =
                (int)p.ChechState();
            indexPerson++;
            switch (p.CurrentState)
            {
                case PersonState.Dead:
                    nbOfDead++;
                    break;
                case PersonState.Healthy:
                    nbOfHealthy++;
                    break;
                case PersonState.Immune:
                    nbOfImmune++;
                    break;
                case PersonState.Infected:
                    nbOfIncubating++;
                    break;
                case PersonState.Asymptomatic:
                case PersonState.Infectious:
                    nbOfInfectious++;
                    break;
                default:
                    break;
            }
        });
        // Retire les individus décédés au cours de
            l'itération.
        _population.RemoveAll(p => p.CurrentState ==
            PersonState.Dead);

        // Trigger l'évènement OnTick qui va mettre à jour
            le GUI et met à jour ses données.
        if (OnGUIUpdate != null)
        {
            OnGUIUpdate(personsNewSite, personsNewState);
```

```
                }

                chartsDatas.AddDatas(GetAllDatas(nbOfDead,
                    nbOfHealthy, nbOfImmune, nbOfIncubating,
                    nbOfInfectious));
                // Affiche au maximum une fois par seconde
                if (sumEllapsedTime >=
                    MAX_TIME_BEFORE_DISPLAYING_AGAIN)
                {
                    // Trigger les évènements qui vont mettre à
                        jour les graphiques
                    OnDisplay?.Invoke(chartsDatas, false);
                    OnDataUpdate?.Invoke(chartsDatas);

                    // Trigger l'évènement qui va mettre à jour le
                        GUI et met à jour ses données.
                    OnGUIUpdate?.Invoke(personsNewSite,
                        personsNewState);

                    sumEllapsedTime = 0;
                }

                _sp.Stop();
                // Calule le temps à attendre pour correspondre au
                    données du slider. (1 secondes par itération -->
                    si l'itération se fait en 100ms alors on attend
                    900ms)
                if (_sp.ElapsedMilliseconds < Interval)
                {
                    long interval = Interval;
                    int delay = (int)(Interval -
                        _sp.ElapsedMilliseconds);
                    await Task.Delay(delay);
                    sumEllapsedTime += delay;
                }
                sumEllapsedTime += (int)_sp.ElapsedMilliseconds;
                _sp.Reset();
            }
            else
            {
                await
                    Task.Delay(DELAY_BEFORE_CHECKING_IF_SIMULATION_SHOULD_START)
                    // Ajoute un délai pour réduire la consommation
                    CPU
            }
        }
    }


    /// <summary>
    /// Trigger l'évènement qui réaffiche les graphiques.
    /// </summary>
    public void TriggerDisplayChanges()
    {
        OnDisplay?.Invoke(chartsDatas, true);
```

```
        }

        #region measures
        /// <summary>
        /// Définit si la mesure du masque doit être appliquée.
        /// </summary>
        /// <param name="nbInfected">Le nombre d'infectés dans la
            simulation.</param>
        /// <param name="isOn">Si la mesure est déjà active.</param>
        /// <returns>Si le mesure est active ou non.</returns>
        private bool SetMaskMeasure(int nbInfected, bool isOn)
        {
            // Si la mesure est déjà active, elle n'est pas réactivée.
            if (isOn == false && nbInfected >
                SimulationGeneralParameters.NbInfecetdForMaskActivation)
            {
                _sites.ForEach(s => s.SetMaskMeasure(true, true));
                isOn = true;
            }
            if (isOn == true && nbInfected <
                SimulationGeneralParameters.NbInfecetdForMaskDeactivation)
            {
                _sites.ForEach(b => b.SetMaskMeasure(false, false));
                isOn = false;
            }
            return isOn;
        }

        /// <summary>
        /// Définit si la mesure des distanciations doit être appliquée.
        /// </summary>
        /// <param name="nbInfected">Le nombre d'infectés dans la
            simulation.</param>
        /// <param name="isOn">Si la mesure est déjà active.</param>
        /// <returns>Si le mesure est active ou non.</returns>
        private bool SetDistanciationMeasure(int nbInfected, bool isOn)
        {
            if (isOn == false && nbInfected >
                SimulationGeneralParameters.NbInfecetdForDistanciationActivation)
            {
                _sites.ForEach(s => s.SetDistanciations(true));
                isOn = true;
            }
            if (isOn == true && nbInfected <
                SimulationGeneralParameters.NbInfecetdForDistanciationDeactivation)
            {
                _sites.ForEach(s => s.SetDistanciations(false));
                isOn = false;
            }
            return isOn;
        }

        /// <summary>
        /// Définit si la quarantaine doit être mise en place.
        /// </summary>
```

```
        /// <param name="nbInfected">Le nombre d'infectés dans la
            simulation.</param>
        /// <param name="isOn">Si la mesure est déjà active.</param>
        /// <returns>Si le mesure est active ou non.</returns>
        private bool SetQuarantineMeasure(int nbInfected, bool isOn)
        {
            if (isOn == false && nbInfected >
                SimulationGeneralParameters.NbInfecetdForQuarantineActivation)
            {
                foreach (var person in _population)
                {
                        person.SetQuarantine(QuarantineParameters.IshealthyQuarantin
                                             QuarantineParameters.IsInfectedQuaranti
                                             QuarantineParameters.IsInfectiousQuaran
                                             QuarantineParameters.IsImmuneQuarantine
                }
                isOn = true;
            }
            if (isOn == true && nbInfected <
                SimulationGeneralParameters.NbInfecetdForQuarantineDeactivation)
            {
                foreach (var person in _population)
                {
                    person.SetQuarantine(false, false, false, false);
                }
                isOn = false;
            }
            return isOn;
        }

        /// <summary>
        /// Définit si la vaccination doit être mise en place ou non.
        /// </summary>
        /// <param name="nbInfected">Le nombre d'infectés dans la
            simulation.</param>
        /// <param name="isOn">Si la mesure est déjà active.</param>
        /// <returns>Si le mesure est active ou non.</returns>
        private bool SetVaccinationMeasure(int nbInfected, bool isOn)
        {
            if (isOn == false && nbInfected >
                SimulationGeneralParameters.NbInfecetdForVaccinationActivation)
            {
                isOn = true;
            }
            if (isOn == true && nbInfected <
                SimulationGeneralParameters.NbInfecetdForVaccinationDeactivation)
            {
                isOn = false;
            }
            return isOn;
        }

        #endregion

        #region Datas
```

```
        /// <summary>
        /// Récupère les données lors d'un évènement qui est déclanché à
            chaque itération de la simulation.
        /// </summary>
        /// <returns>Données actuelles de la simulation.</returns>
        public SimulationDatas GetAllDatas(int nbOfDead, int
            nbOfHealthy, int nbOfImmune, int nbOfIncubating, int
            nbOfInfectious)
        {
            SimulationDatas datas = new SimulationDatas();
            datas.Initialize();

            datas.NumberOfPeople.Add(_population.Count);
            datas.NumberOfInfected.Add(nbOfInfectious + nbOfIncubating);
            datas.NumberOfInfectious.Add(nbOfInfectious);
            datas.NumberOfIncubation.Add(nbOfIncubating);
            datas.NumberOfImmune.Add(nbOfImmune);
            datas.NumberOfHospitalisation.Add(GetNumberOfHospitalisation());
            datas.NumberOfDeath.Add(nbOfDead);
            datas.NumberOfContamination.Add(GetNumberOfContamination());
            datas.NumberOfHealthy.Add(nbOfHealthy);
            datas.NumberOfReproduction.Add(GetNumberOfReproduction());
            return datas;
        }

        // Datas
        public int GetNumberOfHospitalisation()
        {
            return _sitesDictionnary[SiteType.Hospital].Sum(b =>
                ((Hospital)b).CountPatients());
        }

        /// <summary>
        /// Compte le nombre de décès en soustrayant le nombre
            d'individus actuellement présents et les individus
            actuellement healthy.
        /// </summary>
        /// <returns>Nombre de décès dans le timeframe.</returns>
        public int GetNumberOfDeath()
        {
            return SimulationGeneralParameters.NbPeople -
                _population.Count;
        }

        /// <summary>
        /// Compte le nombre de contamination en soustrayant le nombre
            d'infectés du dernier timeframe et celui actuel.
        /// </summary>
        /// <returns>Nombre de contaminations dans le
            timeframe.</returns>
        public double GetNumberOfContamination()
        {
            double result = 0;
            if (chartsDatas.NumberOfInfected != null &&
```

```
            chartsDatas.NumberOfInfected.Count > 1)
        {
            double currentNbOfInfected =
                chartsDatas.NumberOfInfected[chartsDatas.NumberOfInfected.GetLast
            double lastNbOfInfected =
                chartsDatas.NumberOfInfected[chartsDatas.NumberOfInfected.GetLast
                - 1];
            if (lastNbOfInfected == 0)
                result = currentNbOfInfected;
            else
                result = currentNbOfInfected - lastNbOfInfected;

            if (result < 0)
                result = 0;
        }
        return result;
    }

    /// <summary>
    /// Compte le nombre de reproduction en faisant la somme des
       dix dernières données de contaminations et en les divisant
       par le nombre de personnes infectieuses.
    /// </summary>
    /// <returns>Nombre de reproduction dans le timeframe.</returns>
    public double GetNumberOfReproduction()
    {
        double result = 0;
        if (chartsDatas.NumberOfInfectious != null &&
            chartsDatas.NumberOfInfectious.Count >
            NUMBER_OF_INFECTIOUS_VALUES_FOR_REPRODUCTION)
        {
            double avgRecentContamination = 0;
            double currentNbOfInfectious =
                chartsDatas.NumberOfInfectious[chartsDatas.NumberOfInfectious.Ge

            for (int i = chartsDatas.NumberOfContamination.Count -
                NUMBER_OF_INFECTIOUS_VALUES_FOR_REPRODUCTION; i <
                chartsDatas.NumberOfContamination.Count; i++)
            {
                avgRecentContamination +=
                    chartsDatas.NumberOfContamination[i];
            }

            if (currentNbOfInfectious != 0)
            {
                result = avgRecentContamination /
                    currentNbOfInfectious;
            }
        }
        return result;
    }

    #endregion

    /// <summary>
```

```
        /// Démarre/Redémarre la simulation.
        /// </summary>
        public void Start()
        {
            _startStop = true;
        }

        /// <summary>
        /// Arrête/Pause la simulation.
        /// </summary>
        public void Stop()
        {
            _startStop = false;
        }

        /// <summary>
        /// ID documentation: Create_Buildings
        /// Créé les bâtiments de la simulation en fonction du nombre
           de personnes.
        /// Peut importe le nombre de personnes, il existe un bâtiment
           de chaque.
        /// Les bâtiments sont créé proportionnelement à la taille de
           la population et à leur type.
        /// Ne créé par les maisons de la population.
        /// </summary>
        private void CreateBuildings()
        {
            // 6.8d est le rapport bâtiment / personne sans compter les
               habitations.
            int nbBuildings = (int)Math.Ceiling((6.8d - 100) / 100 *
               _nbPersons + _nbPersons);
            KeyValuePair<object, double>[] companyType = new
               KeyValuePair<object, double>[] {
                    new KeyValuePair<object, double>(typeof(Company),
                       PROBABILITY_OF_BEING_A_COMPANY),
                    new KeyValuePair<object, double>(typeof(Store),
                       PROBABILITY_OF_BEING_A_STORE),
                    new KeyValuePair<object,
                       double>(typeof(Restaurant),
                       PROBABILITY_OF_BEING_A_RESTAURANT),
                    new KeyValuePair<object, double>(typeof(School),
                       PROBABILITY_OF_BEING_A_SCHOOL),
                    new KeyValuePair<object, double>(typeof(Hospital),
                       PROBABILITY_OF_BEING_A_HOSPITAL),
                    new KeyValuePair<object,
                       double>(typeof(Supermarket),
                       PROBABILITY_OF_BEING_A_SUPERMARKET),
            };
            _sites.Add(_outside);
            // Permet d'ajouter les éléments dans le bon ordre pour
               permettre d'utiliser leur index comme id dans unity
            List<Site> hospitals = new List<Site>();
            List<Site> schools = new List<Site>();
            List<Site> stores = new List<Site>();
            List<Site> restaurants = new List<Site>();
```

```
List<Site> supermarkets = new List<Site>();
List<Site> companies = new List<Site>();

#region list
_sitesDictionnary.Add(SiteType.Hospital, new List<Site>());
_sitesDictionnary.Add(SiteType.Store, new List<Site>());
_sitesDictionnary.Add(SiteType.WorkPlace, new List<Site>());
_sitesDictionnary.Add(SiteType.Eat, new List<Site>());
_sitesDictionnary.Add(SiteType.School, new List<Site>());
//buildingSitesDictionnaryArray.Add(typeof(Company), new
   Site[nbBuildings]);

while (nbBuildings > 0)
{
    Type result = (Type)rdm.NextProbability(companyType);
    Site site;

    if (result == typeof(Company))
    {
        site = new Company();
        _sitesDictionnary[SiteType.WorkPlace].Add(site);
        companies.Add(site);
    }
    else if (result == typeof(Store))
    {
        site = new Store();
        _sitesDictionnary[SiteType.Store].Add(site);
        _sitesDictionnary[SiteType.WorkPlace].Add(site);
        stores.Add(site);
    }
    else if (result == typeof(Restaurant))
    {
        site = new Restaurant();
        _sitesDictionnary[SiteType.Eat].Add(site);
        _sitesDictionnary[SiteType.WorkPlace].Add(site);
        restaurants.Add(site);
    }
    else if (result == typeof(School))
    {
        site = new School();
        _sitesDictionnary[SiteType.School].Add(site);
        _sitesDictionnary[SiteType.WorkPlace].Add(site);
        schools.Add(site);
    }
    else if (result == typeof(Hospital))
    {
        site = new Hospital();
        _sitesDictionnary[SiteType.Hospital].Add(site);
        _sitesDictionnary[SiteType.WorkPlace].Add(site);
        hospitals.Add(site);
    }
    else
    {
        site = new Supermarket();
        _sitesDictionnary[SiteType.Store].Add(site);
```

```
                _sitesDictionnary[SiteType.WorkPlace].Add(site);
                supermarkets.Add(site);
            }
            nbBuildings--;
        }
        #endregion

        #region missingBuildings
        Site missingSite;
        if (_sitesDictionnary[SiteType.Hospital].Count == 0)
        {
            missingSite = new Hospital();
            _sitesDictionnary[SiteType.Hospital].Add(missingSite);
            companies.Add(missingSite);
        }

        if (_sitesDictionnary[SiteType.Store].Count == 0)
        {
            missingSite = new Store();
            _sitesDictionnary[SiteType.Store].Add(missingSite);
            stores.Add(missingSite);
        }

        if (_sitesDictionnary[SiteType.Eat].Count == 0)
        {
            missingSite = new Restaurant();
            _sitesDictionnary[SiteType.Eat].Add(missingSite);
            restaurants.Add(missingSite);
        }

        if (_sitesDictionnary[SiteType.School].Count == 0)
        {
            missingSite = new School();
            _sitesDictionnary[SiteType.School].Add(missingSite);
            schools.Add(missingSite);
        }

        #endregion

        _sites.AddRange(hospitals);
        _sites.AddRange(schools);
        _sites.AddRange(stores);
        _sites.AddRange(restaurants);
        _sites.AddRange(supermarkets);
        _sites.AddRange(companies);
    }


    /// <summary>
    /// Créé les transports publiques de la simulation.
    /// </summary>
    private void CreatePublicTransports()
    {
        double nbOfBus = (int)Math.Ceiling((PERCENTAGE_OF_BUS -
            100) / 100 * _nbPersons + _nbPersons);
```

```
            _sitesDictionnary.Add(SiteType.Transport, new List<Site>());

            for (int i = 0; i < nbOfBus; i++)
            {
                Bus bus = new Bus();
                _sites.Add(bus);
                _sitesDictionnary[SiteType.WorkPlace].Add(bus);
                _sitesDictionnary[SiteType.Transport].Add(bus);
            }
        }

        /// <summary>
        /// Créé la population, définit l'âge de la personne ainsi que
            les lieux dans lesquelles elle va aller.
        /// Définit si la personne est infectée dès le départ ou si
            elle est saine.
        /// </summary>
        /// <param name="nbPeople">Nombre de personnes à créer.</param>
        /// <param name="retirementProbability">pourcentage de chance
            que la personne soit retraitée.</param>
        /// <param name="minorProbability">Pourcentage de chance que la
            personne soit mineur.</param>
        private void CreatePopulation(int nbPeople, double
            retirementProbability, double minorProbability)
        {
            _sitesDictionnary.Add(SiteType.Home, new List<Site>());
            List<Site> houses = new List<Site>();
            while (nbPeople > 0)
            {
                int age;
                int nbWorkDays;
                Home home = new Home();
                Hospital hospital =
                    (Hospital)_sitesDictionnary[SiteType.Hospital][rdm.Next(0,
                    _sitesDictionnary[SiteType.Hospital].Count)];
                Dictionary<SiteType, List<Site>> personSites = new
                    Dictionary<SiteType, List<Site>>();
                PersonState personState;

                // Calcul la probabilité que l'individu soit infecté
                    dès le départ.
                if (rdm.NextBoolean(_probabilityOfBeingInfected))
                    personState = PersonState.Infectious;
                else
                    personState = PersonState.Healthy;

                // ID Documentation: Sites_Attribution, Create_Person
                // Sélectionne les lieux dans lesquels l'individu va se
                    déplacer en fonction des probabilité qu'il a d'être
                    soit retraité soit mineur soit en âge de travailler.
                if
                    (GlobalVariables.rdm.NextBoolean(retirementProbability))
                {
                    personSites = new Dictionary<SiteType,
                        List<Site>>() {
```

```
                { SiteType.Home, new List<Site>{home} },
                { SiteType.Store,
                    CreateTypePersonSites(SiteType.Store,
                    NUMBER_OF_STORE_PER_PERSON) },
                { SiteType.Eat,
                    CreateTypePersonSites(SiteType.Eat,
                    NUMBER_OF_PLACE_TO_EAT_PER_PERSON) },
                { SiteType.Transport, new List<Site>{
                    GetVehicle() } }
            };
            age = rdm.NextInclusive(60, 100);
            nbWorkDays = 0;
        }
        else if
            (GlobalVariables.rdm.NextBoolean(minorProbability))
        {
            personSites = new Dictionary<SiteType,
                List<Site>>() {
                { SiteType.Home, new List<Site>{home} },
                { SiteType.Store,
                    CreateTypePersonSites(SiteType.Store,
                    NUMBER_OF_STORE_PER_PERSON) },
                { SiteType.Eat,
                    CreateTypePersonSites(SiteType.Eat,
                    NUMBER_OF_PLACE_TO_EAT_PER_PERSON) },
                { SiteType.Transport, new List<Site>{ _outside
                    } },
                { SiteType.WorkPlace,
                    CreateTypePersonSites(SiteType.School, 1) },
            };
            age = rdm.NextInclusive(5, 24);
            nbWorkDays = 5;
        }
        else
        {
            personSites = new Dictionary<SiteType,
                List<Site>>() {
                { SiteType.Home, new List<Site>{home} },
                { SiteType.Store,
                    CreateTypePersonSites(SiteType.Store,
                    NUMBER_OF_STORE_PER_PERSON) },
                { SiteType.Eat,
                    CreateTypePersonSites(SiteType.Eat,
                    NUMBER_OF_PLACE_TO_EAT_PER_PERSON) },
                { SiteType.Transport, new List<Site>{
                    _outside, GetVehicle() } },
                { SiteType.WorkPlace, new List<Site>{
                    FindWorkPlace() } }
            };
            age = rdm.NextInclusive(18, 70);
            nbWorkDays = 5;
        }
        // Créé le planning avec les lieux créés et créé
            l'individu
        houses.Add(home);
```

```
                _sitesDictionnary[SiteType.Home].Add(home);
                Planning planning = new Planning(personSites,
                    nbWorkDays);
                _population.Add(new Person(planning, hospital, age,
                    personState));
                nbPeople--;
            }
            // Insère les maison des individus au début de la liste
                pour garder un ordre précis dans le GUI.
            _sites.InsertRange(0, houses);
            // Tous les lieux sont entrés, les ids peuvent être créés.
            for (int i = 1; i < _sites.Count; i++)
            {
                _sitesIds.Add(_sites[i], i);
            }
        }

        /// <summary>
        /// Créé le nombre de lieu demandé du type demandé dans une
            liste.
        /// </summary>
        /// <param name="type">Type de lieux</param>
        /// <param name="quantity">Nombre de lieux à créer.</param>
        /// <returns>Liste de lieux demandé à la quantité
            demandée.</returns>
        private List<Site> CreateTypePersonSites(SiteType type, int
            quantity)
        {
            List<Site> sites = new List<Site>();
            for (int i = 0; i < quantity; i++)
            {
                sites.Add(_sitesDictionnary[type][rdm.Next(0,
                    _sitesDictionnary[type].Count)]);
            }
            return sites;
        }

        /// <summary>
        /// Méthode récursive permettant de trouver un lieu de travail
            pour un individu.
        /// </summary>
        /// <returns>Lieu de travail.</returns>
        private Site FindWorkPlace()
        {
            WorkSite workplace =
                (WorkSite)_sitesDictionnary[SiteType.WorkPlace][rdm.Next(0,
                _sitesDictionnary[SiteType.WorkPlace].Count)];
            return workplace.IsHiring() ? (Site)workplace :
                FindWorkPlace();
        }

        /// <summary>
        /// Récupère le moyen de transport qu'un individu adulte
            utilisera.
        /// </summary>
```

```
        /// <returns></returns>
        private Site GetVehicle()
        {
            KeyValuePair<object, double>[] transportsProbability = new
                KeyValuePair<object, double>[] {
                new KeyValuePair<object, double>(new Car(),
                    PROBABILITY_OF_USING_A_CAR),
                new KeyValuePair<object, double>(_outside,
                    PROBABILITY_OF_WALKING),
                new KeyValuePair<object, double>(new Bike(),
                    PROBABILITY_OF_USING_A_BIKE),
                new KeyValuePair<object,
                    double>(_sitesDictionnary[SiteType.Transport][rdm.Next(0,
                    _sitesDictionnary[SiteType.Transport].Count)],
                    PROBABILITY_OF_USING_A_BUS),
            };
            return (Site)rdm.NextProbability(transportsProbability);
        }
    }
}
```

Listing 41 – Code source de F*simulation/SimulationDatas.cs*

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste représentant une ville.
 */
using System.Collections.Generic;

namespace CovidPropagation
{
    public struct SimulationDatas
    {
        public void Initialize()
        {
            CentralizedDatas = new List<KeyValuePair<string,
                List<double>>>();
            NumberOfPeople = new List<double>();
            NumberOfInfected = new List<double>();
            NumberOfInfectious = new List<double>();
            NumberOfIncubation = new List<double>();
            NumberOfImmune = new List<double>();
            NumberOfReproduction = new List<double>();
            NumberOfHealthy = new List<double>();
            NumberOfHospitalisation = new List<double>();
            NumberOfDeath = new List<double>();
            NumberOfContamination = new List<double>();

            CentralizedDatas.Add(new KeyValuePair<string,
                List<double>>("Nombre de personnes", NumberOfPeople));
            CentralizedDatas.Add(new KeyValuePair<string,
```

```
            List<double>>("Nombre d'infectés", NumberOfInfected));
        CentralizedDatas.Add(new KeyValuePair<string,
            List<double>>("Nombre d'infectieux",
            NumberOfInfectious));
        CentralizedDatas.Add(new KeyValuePair<string,
            List<double>>("Nombre en incubation",
            NumberOfIncubation));
        CentralizedDatas.Add(new KeyValuePair<string,
            List<double>>("Nombre d'immunisé", NumberOfImmune));
        CentralizedDatas.Add(new KeyValuePair<string,
            List<double>>("Nombre de reproductions",
            NumberOfReproduction));
        CentralizedDatas.Add(new KeyValuePair<string,
            List<double>>("Nombre de sains", NumberOfHealthy));
        CentralizedDatas.Add(new KeyValuePair<string,
            List<double>>("Nombre d'hospitalisation",
            NumberOfHospitalisation));
        CentralizedDatas.Add(new KeyValuePair<string,
            List<double>>("Nombre de décès", NumberOfDeath));
        CentralizedDatas.Add(new KeyValuePair<string,
            List<double>>("Nombre de contamination",
            NumberOfContamination));
    }
    public List<KeyValuePair<string, List<double>>>
        CentralizedDatas { get; set; }

    public List<double> NumberOfPeople { get; set; }

    public List<double> NumberOfInfected { get; set; }
    public List<double> NumberOfInfectious { get; set; }
    public List<double> NumberOfIncubation { get; set; }

    public List<double> NumberOfImmune { get; set; }

    public List<double> NumberOfReproduction { get; set; }

    public List<double> NumberOfHealthy { get; set; }

    public List<double> NumberOfHospitalisation { get; set; }

    public List<double> NumberOfDeath { get; set; }

    public List<double> NumberOfContamination { get; set; }

    /// <summary>
    /// Ajoute des données à la structure.
    /// </summary>
    /// <param name="newDatas">Données à ajouter à la structure
        provenant d'une autre structure.</param>
    public void AddDatas(SimulationDatas newDatas)
    {
        NumberOfPeople.AddRange(newDatas.NumberOfPeople);
        NumberOfInfected.AddRange(newDatas.NumberOfInfected);
        NumberOfInfectious.AddRange(newDatas.NumberOfInfectious);
        NumberOfIncubation.AddRange(newDatas.NumberOfIncubation);
```

```
            NumberOfImmune.AddRange(newDatas.NumberOfImmune);
            NumberOfReproduction.AddRange(newDatas.NumberOfReproduction);
            NumberOfHealthy.AddRange(newDatas.NumberOfHealthy);
            NumberOfHospitalisation.AddRange(newDatas.NumberOfHospitalisation);
            NumberOfDeath.AddRange(newDatas.NumberOfDeath);
            NumberOfContamination.AddRange(newDatas.NumberOfContamination);
        }

        /// <summary>
        /// Récupère la liste correspondant a la valeur du enum.
        /// </summary>
        /// <param name="enumData">Valeurs à récupérer.</param>
        /// <returns>Valeurs correspondants aux valeurs à
           récupérer.</returns>
        public List<double> GetDataFromEnum(ChartsDisplayData enumData)
        {
            List<double> result;
            switch (enumData)
            {
                default:
                case ChartsDisplayData.Persons:
                    result = NumberOfPeople;
                    break;
                case ChartsDisplayData.Cases:
                    result = NumberOfInfected;
                    break;
                case ChartsDisplayData.Infectious:
                    result = NumberOfInfectious;
                    break;
                case ChartsDisplayData.Incubations:
                    result = NumberOfIncubation;
                    break;
                case ChartsDisplayData.Immune:
                    result = NumberOfImmune;
                    break;
                case ChartsDisplayData.Re:
                    result = NumberOfReproduction;
                    break;
                case ChartsDisplayData.Healthy:
                    result = NumberOfHealthy;
                    break;
                case ChartsDisplayData.Hospitalisation:
                    result = NumberOfHospitalisation;
                    break;
                case ChartsDisplayData.Death:
                    result = NumberOfDeath;
                    break;
                case ChartsDisplayData.Contamination:
                    result = NumberOfContamination;
                    break;
            }
            if (result == null)
                result = new List<double>();
            return result;
        }
```

```
    }
}
```

Listing 42 – Code source de $\mathsf{F}_{S}simulation/GlobalVariables.cs$

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste représentant une ville.
 */
using System;

namespace CovidPropagation
{
    static class GlobalVariables
    {
        public static readonly Random rdm = new Random();

        public const int DURATION_OF_TIMEFRAME = 30; // En minutes
        public const int MINUTES_PER_DAY = 1440;
        public const int NUMBER_OF_TIMEFRAME = MINUTES_PER_DAY /
            DURATION_OF_TIMEFRAME;
        public const int NUMBER_OF_DAY = 7;

        public const double ILNESS_INFECTION_PROBABILITY = 0.01d;

        public const int DEFAULT_INTERVAL = 500;

        // Symptomatic / Asymptomatic
        public const int PERCENTAGE_OF_ASYMPTOMATIC = 5;
        public const int ASYMPTOMATIC_MIN_RESISTANCE = 90;
        public const int ASYMPTOMATIC_MAX_RESISTANCE = 101; // 101
            exclus

        public const int SYMPTOMATIC_MIN_RESISTANCE = 80;
        public const int SYMPTOMATIC_MAX_RESISTANCE = 90; // 90 exclus

        // Person
        public const int DEFAULT_PERSON_AGE = 30;

        public const double AVERAGE_QUANTA_EXHALATION = 5.6d;

        #region SitesDefaultParameter
        public const double OUTSIDE_TEMPERATURE = 30;

        // Bâtiments
        public const double BUILDING_WIDTH = 15;
        public const double BUILDING_LENGTH = 15;
        public const double BUILDING_HEIGHT = 2;

        public const double BUILDING_PRESSURE = 0.95d;
        public const double BUILDING_TEMPERATURE = 23;
```

```
        public const double BUILDING_CO2 = 415;
        public const double BUILDING_VENTILATION_WITH_OUTSIDE = 0.7d;
        public const double BUILDING_ADDITIONAL_CONTROL_MEASURES = 0;

        // Véhicules
            // Voiture
        public const double CAR_WIDTH = 30;
        public const double CAR_LENGTH = 30;
        public const double CAR_HEIGHT = 3;

        public const double CAR_VENTILATION_WITH_OUTSIDE = 0.7d;
        public const double CAR_ADDITIONAL_CONTROL_MEASURES = 0;
            // Bus
        public const double BUS_WIDTH = 2.5d;
        public const double BUS_LENGTH = 19;
        public const double BUS_HEIGHT = 3.3d;

        public const double BUS_VENTILATION_WITH_OUTSIDE = 3d;
        public const double BUS_ADDITIONAL_CONTROL_MEASURES = 0;
        #endregion
    }
}
```

Listing 43 – Code source de $F_{S}imulation/F_{U}nityDatas/DataPopulation.cs$

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste représentant une ville.
 */

namespace CovidPropagation
{
    /// <summary>
    /// ID Documentation : Data_Population
    /// Permet de transférer les données de la population au GUI.
    /// </summary>
    public class DataPopulation
    {
        private int nbPersons; // Le nombre d'individus
        private int[] indexOfInfected; // L'index des individus
            infectés.

        public int NbPersons { get => nbPersons; set => nbPersons =
            value; }
        public int[] IndexOfInfected { get => indexOfInfected; set =>
            indexOfInfected = value; }

        public DataPopulation(int nbPersons, int[] indexInfected)
        {
            NbPersons = nbPersons;
```

```
            IndexOfInfected = indexInfected;
        }
    }
}
```

Listing 44 – Code source de $\mathsf{F}_{simulation/F_{U}nityDatas/DataSites.cs}$

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste représentant une ville.
 */

namespace CovidPropagation
{
    /// <summary>
    /// ID Documentation : Data_Sites
    /// Données des sites qui seront transmisent par pipeline au GUI.
    /// </summary>
    public class DataSites
    {
        private int _nbHouse;
        private int _nbCompany;
        private int _nbHospital;
        private int _nbRestaurant;
        private int _nbSchool;
        private int _nbStore;
        private int _nbSupermarket;

        public int NbHouse { get => _nbHouse; set => _nbHouse = value; }
        public int NbCompany { get => _nbCompany; set => _nbCompany =
            value; }
        public int NbHospital { get => _nbHospital; set => _nbHospital
            = value; }
        public int NbRestaurant { get => _nbRestaurant; set =>
            _nbRestaurant = value; }
        public int NbSchool { get => _nbSchool; set => _nbSchool =
            value; }
        public int NbStore { get => _nbStore; set => _nbStore = value; }
        public int NbSupermarket { get => _nbSupermarket; set =>
            _nbSupermarket = value; }

        public DataSites(int nbHouse, int nbCompany, int nbHospital,
            int nbRestaurant, int nbSchool, int nbStore, int
            nbSupermarket)
        {
            _nbHouse = nbHouse;
            _nbCompany = nbCompany;
            _nbHospital = nbHospital;
            _nbRestaurant = nbRestaurant;
            _nbSchool = nbSchool;
            _nbStore = nbStore;
```

```
            _nbSupermarket = nbSupermarket;
        }
    }
}
```

Listing 45 – Code source de $\mathsf{F}_{Simulation/FUnityDatas/DataIteration.cs}$

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste représentant une ville.
 */

namespace CovidPropagation
{
    /// <summary>
    /// ID Documentation : Data_Iteration
    /// Données qui seront envoyé au GUI à chaque itération.
    /// </summary>
    public class DataIteration
    {
        private int[] personsNewSite;
        private int[] personsNewState;

        public int[] PersonsNewSite { get => personsNewSite; set =>
            personsNewSite = value; }
        public int[] PersonsNewState { get => personsNewState; set =>
            personsNewState = value; }

        public DataIteration(int[] personsNewSite, int[]
            personsNewState)
        {
            this.PersonsNewSite = personsNewSite;
            this.PersonsNewState = personsNewState;
        }
    }
}
```

Listing 46 – Code source de $\mathsf{F}_{Simulation/FVirus/Transmission.cs}$

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste représentant une ville.
 */
using System;
using System.Collections.Generic;
using System.Text;
```

```
namespace CovidPropagation
{
    /// <summary>
    /// Interface permettant le regroupement des différents moyens de
       transmissions.
    /// </summary>
    public interface Transmission
    {

    }
}
```

Listing 47 – Code source de $\mathsf{F}_{Simulation/F_Virus/Symptom.cs}$

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste représentant une ville.
 */
namespace CovidPropagation
{
    /// <summary>
    /// Interface permettant de regrouper les symptômes.
    /// </summary>
    public interface Symptom
    {

    }
}
```

Listing 48 – Code source de $\mathsf{F}_{Simulation/F_Virus/F_Symptoms/CoughSymptom.cs}$

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste tel qu'une ville.
 */

namespace CovidPropagation
{
    class CoughSymptom : Symptom
    {
        private int _quantaAddedMin;
        private int _quantaAddedMax;

        public CoughSymptom(int quantaAddedMin, int quantaAddedMax)
        {
            _quantaAddedMin = quantaAddedMin;
            _quantaAddedMax = quantaAddedMax;
```

```
        }

        /// <summary>
        /// Récupère le nombre de quanta que ce symptôme ajoutera.
        /// </summary>
        /// <returns>Valeur aléatoir entre le minimum et le maximum de
            quanta.</returns>
        public double QuantaAddedByCoughing()
        {
            return GlobalVariables.rdm.Next(_quantaAddedMin,
                _quantaAddedMax);
        }
    }
}
```

Listing 49 – Code source de $\mathsf{F}_{Simulation/F_Virus/F_Transmission/AerosolTransmission.cs}$

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste représentant une ville.
 */
using System;

namespace CovidPropagation
{
    class AerosolTransmission : Transmission
    {
        private double _probabilityOfOneInfection;
        private double _probabilityOfInfection;

        private double _nOfInfectivePersons;
        private double _virusAraisingCases;

        public AerosolTransmission()
        {

        }

        /// <summary>
        /// Calcule les risques de propagation d'aérosols.
        /// </summary>
        public AerosolTransmissionData CalculateRisk(int nbPersons, int
            infectivePersons, double fractionOfImmune, int
            nbPersonsWithMask, double inhalationMaskEfficiency, double
            sumFirstOrderLossRate, double volume, double
            quantaExhalationRateOfInfected, double
            probabilityOfBeingInfective, double quantaInhaledPerPerson)
        {
            int nbSusceptiblePersons = ((nbPersons - infectivePersons)
                * (1 - (int)fractionOfImmune));
```

```
            // Infection probability
            _probabilityOfOneInfection = 1 -
               Math.Exp(-quantaInhaledPerPerson);
            _probabilityOfInfection = (1 - Math.Pow((1 -
               _probabilityOfOneInfection *
               probabilityOfBeingInfective), nbSusceptiblePersons)) /
               10;

            _nOfInfectivePersons = (infectivePersons +
               nbSusceptiblePersons) * probabilityOfBeingInfective;
            _nOfInfectivePersons = _nOfInfectivePersons.SetValueIfNaN();
            _virusAraisingCases = (nbSusceptiblePersons -
               _nOfInfectivePersons) * _probabilityOfInfection;
            _virusAraisingCases = _virusAraisingCases.SetValueIfNaN();

            return new AerosolTransmissionData(
                _probabilityOfOneInfection,
                _probabilityOfInfection,
                _nOfInfectivePersons,
                _virusAraisingCases
            );
        }
    }
}
```

Listing 50 – Code source de $\mathsf{F}_{S}imulation/F_{V}irus/F_{T}ransmission/AerosolTransmissionData.cs$

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste tel qu'une ville.
 */
namespace CovidPropagation
{
    /// <summary>
    /// Classe utilisée dans le transfère de données de la transmission
       du virus.
    /// </summary>
    public struct AerosolTransmissionData
    {
        public double ProbabilityOfOneInfection { get; set; }
        public double ProbabilityOfInfection { get; set; }
        public double NOfInfectiousPersons { get; set; }
        public double VirusAraisingCases { get; set; }

        public AerosolTransmissionData(
                double probabilityOfOneInfection,
                double probabilityOfInfection,
                double nbOfInfectivePersons,
                double virusAraisingCases)
        {
```

```
            ProbabilityOfOneInfection = probabilityOfOneInfection;
            ProbabilityOfInfection = probabilityOfInfection;
            NOfInfectiousPersons = nbOfInfectivePersons;
            VirusAraisingCases = virusAraisingCases;
        }
    }
}
```

Listing 51 – Code source de $\mathsf{F}_{Simulation/F_Virus/Virus.cs}$

```
/*
 * Nom du projet : CovidPropagation
 * Auteur        : Joey Martig
 * Date          : 11.06.2021
 * Version       : 1.0
 * Description   : Simule la propagation du covid dans un environnement
    vaste représentant une ville.
 */
using System;
using System.Collections.Generic;
using System.Xml;
using System.Linq;

namespace CovidPropagation
{
    /// <summary>
    /// ID Documentation : Virus_Class
    /// Class contenant les données du virus qui est utilisé pour
       simuler le/la covid.
    /// </summary>
    public static class Virus
    {
        private const string XML_LOCATION = "./CovidData.xml";
        private const string TRANSMISSION_XML_NAME = "Transmissions";
        private const string SYMPTOMS_XML_NAME = "Symptoms";
        private const string AEROSOL_XML_NAME = "Aerosol";
        private const string COUGH_XML_NAME = "Cough";

        private const int MIN_QUANTA = 100;
        private const int MAX_QUANTA = 200;

        private static List<Symptom> commonSymptoms;
        private static List<Symptom> rareSymptoms;
        private static List<Transmission> transmissions;

        private static int _incubationDurationMin;
        private static int _incubationDurationMax;
        private static int _durationMin; // En jours
        private static int _durationMax; // En jours

        private static int _immunityDurationMin; // En jours
        private static int _immunityDurationMax; // En jours
        private static double _immunityEfficiency;

        private static double _decayRateOfVirus;
```

```
        private static double _depositionOnSurfaceRate;

        public static int IncubationDurationMin { get =>
            _incubationDurationMin; }
        public static int IncubationDurationMax { get =>
            _incubationDurationMax; }
        public static int IncubationDuration { get =>
            GlobalVariables.rdm.NextInclusive(_incubationDurationMin,
            _incubationDurationMax); }
        public static int DurationMin { get => _durationMin; }
        public static int DurationMax { get => _durationMax; }
        public static int Duration { get =>
            GlobalVariables.rdm.NextInclusive(_durationMin,
            _durationMax); }
        public static double DecayRateOfVirus { get =>
            _decayRateOfVirus; }
        public static double DepositionOnSurfaceRate { get =>
            _depositionOnSurfaceRate; }
        public static int ImmunityDurationMin { get =>
            _immunityDurationMin; set => _immunityDurationMin = value; }
        public static int ImmunityDurationMax { get =>
            _immunityDurationMax; set => _immunityDurationMax = value; }
        public static double ImmunityEfficiency { get =>
            _immunityEfficiency; set => _immunityEfficiency = value; }

        /// <summary>
        /// Récupère les données se trouvant dans le fichier XML
            contenant les données du covid et intialise le virus.
        /// </summary>
        public static void Init()
        {
            XmlDocument xmlDatas = new XmlDocument();
            xmlDatas.Load(XML_LOCATION);

            XmlNodeList transmissionsNode =
                xmlDatas.GetElementsByTagName(TRANSMISSION_XML_NAME);
            XmlNodeList symptomsNode =
                xmlDatas.GetElementsByTagName(SYMPTOMS_XML_NAME);

            transmissions = new List<Transmission>();
            commonSymptoms = new List<Symptom>();
            rareSymptoms = new List<Symptom>();

            _incubationDurationMin =
                VirusParameters.IncubationDurationMin;
            _incubationDurationMax =
                VirusParameters.IncubationDurationMax;
            _durationMin = VirusParameters.DurationMin;
            _durationMax = VirusParameters.DurationMax;
            _decayRateOfVirus = VirusParameters.DecayRateOfVirus;
            _depositionOnSurfaceRate =
                VirusParameters.DepositionOnSurfaceRate;
            _immunityDurationMin = VirusParameters.ImmunityDurationMin;
            _immunityDurationMax = VirusParameters.ImmunityDurationMax;
            _immunityEfficiency = VirusParameters.ImmunityEfficiency;
```

```
            // Parcours les moyens de transmissions et les ajoutent au
                virus.
            for (int i = 0; i < transmissionsNode.Count; i++)
            {
                for (int j = 0; j <
                    transmissionsNode[i].ChildNodes.Count; j++)
                {
                    if (transmissionsNode[i].ChildNodes[j].Name ==
                        AEROSOL_XML_NAME &&
                        VirusParameters.IsAerosolTransmissionActive)
                    {
                        transmissions.Add(new AerosolTransmission());
                    }
                }
            }

            // Parcours les symptômes et les ajoutent au virus.
            for (int i = 0; i < symptomsNode.Count; i++)
            {
                for (int j = 0; j < symptomsNode[i].ChildNodes.Count;
                    j++)
                {
                    if (symptomsNode[i].ChildNodes[j].Name ==
                        COUGH_XML_NAME &&
                        VirusParameters.IsCoughSymptomActive)
                    {
                        int minQanta = MIN_QUANTA;
                        int maxQanta = MAX_QUANTA;

                        if
                            (symptomsNode[i].ChildNodes[j].ChildNodes.Count
                            >= 1 && VirusParameters.CoughMinQuanta < 0)
                            minQanta =
                                Convert.ToInt32(symptomsNode[i].ChildNodes[j].ChildNo
                        else if(VirusParameters.CoughMinQuanta > 0)
                            minQanta = VirusParameters.CoughMinQuanta;

                        if
                            (symptomsNode[i].ChildNodes[j].ChildNodes.Count
                            >= 2 && VirusParameters.CoughMaxQuanta < 0)
                            maxQanta =
                                Convert.ToInt32(symptomsNode[i].ChildNodes[j].ChildNo
                        else if(VirusParameters.CoughMaxQuanta > 0)
                            maxQanta = VirusParameters.CoughMaxQuanta;

                        commonSymptoms.Add(new CoughSymptom(minQanta,
                            maxQanta));
                    }
                }
            }
        }

        /// <summary>
        /// Définit si le virus est transmissible par le type de
```

```
            transmission demandée.
        /// </summary>
        /// <param name="typeOfTransmission">Type de transmission à
            vérifier.</param>
        /// <returns>Si le virus est transmissible par le type demandé
            ou non.</returns>
        public static bool IsTransmissibleBy(Type typeOfTransmission)
        {
            bool result = false;
            if (transmissions.Where(t => t.GetType() ==
                typeOfTransmission).Any())
                result = true;

            return result;
        }

        /// <summary>
        /// Récupère un certain type de transmission du virus.
        /// </summary>
        /// <param name="typeOfTransmission">Le type de transmission à
            récupérer.</param>
        /// <returns>Objet du type demandé.</returns>
        public static Transmission GetTransmission(Type
            typeOfTransmission)
        {
            return transmissions.Where(t => t.GetType() ==
                typeOfTransmission).First();
        }

        /// <summary>
        ///  Récupère la liste de symptômes du virus
        /// </summary>
        /// <returns>Liste de symptômes</returns>
        public static List<Symptom> GetCommonSymptoms(){
            return commonSymptoms;
        }
    }
}
```