Table des matières

Résumé

Covid propagation est une application permettant de visualiser l'évolution du covid au sain d'un environnement peuplé d'individus uniques. La visualisation se fait à l'aide de graphiques (Colonne, courbe, circulaire) et de données telles que le nombre d'infecté actuellement ou le nombre de rétablissements ainsi qu'une interface graphique permettant la visualisation des individus, des lieux, des véhicules et de leurs déplacements.

L'utilisateur a une certaine liberté dans les paramètres de la simulation comme pour la création de la population en choisissant leur nombre et l'âge moyen.

Des mesures contre le virus peuvent être prises, limitant ou stoppant sa propagation.

Abstract

Introduction

Dans le cadre du cursus technicien, nous sommes amenés à réaliser un travail de diplôme qui dure du 19 avril aux 11 juin. Durant cette période, plusieurs évaluations intermédiaires sont prévues, la première se situant 10 jours après le début du travail de diplôme. La seconde évaluation est prévue le 17 mai, la troisième le 31 mai et la dernière est le rendu final le 11 juin et dure 9 jours contrairement aux autres sprints qui durent 10 jours.

Il est nécessaire de réaliser un poster pour ce travail ainsi que de remplir un journal de bord comprenant nos activités et nos réflexions.

Le but de mon projet est de simuler une propagation du covid dans une simulation dite individu centré. Ce qui signifie que des individus sont simulés et agissent selon leur planning. S'ils sont infectés, c'est en cas de contact avec une autre personne infectée. Les données utilisées sont des données officielles et sont maintenues à jour aussi souvent que possible.

WPF

Le programme WPF est le coeur de l'application, il réunit toutes les sections du projet et les gère.

Simulation

La simulation génère tous les objets nécessaires au fonctionnement de celle-ci. Ses paramètres peuvent être modifiés depuis L'UI. Ses paramètres concernant le virus sont écrits dans un fichier XML. La simulation gère

aussi la temporalité permettant la propagation et les déplacements.

UI

L'interface utilisateur est gérée par les grilles WPF permettant un affichage responsive. Elle permet à l'utilisateur de modifier les paramètres de la simulation ainsi que les paramètres d'affichage des graphiques.

XML

Les paramètres jugés fixes du virus sont stockés dans un fichier XML.

Graphiques

Les graphiques sont créés par la librairie liveCharts qui permet l'affichage de nombreux type de graphiques ainsi qu'un grand contrôle sur ceux-ci. Les données sont mises à jour en temps réel et des animations intégrées à la librairie.

Unity

Le programme Unity s'occupe de gérer l'interface graphique qui comprend les bâtiments, véhicules et individus. L'interface est animée en fonction de la temporalité de l'application WPF. La simulation et l'interface graphique avancent donc ensemble. La communication s'effectue à travers un pipeline nommé. Les données de la simulation sont envoyées par celui-ci. Le programme Unity est intégré directement dans le projet WPF.

Maquettes

UI

Page Simulation

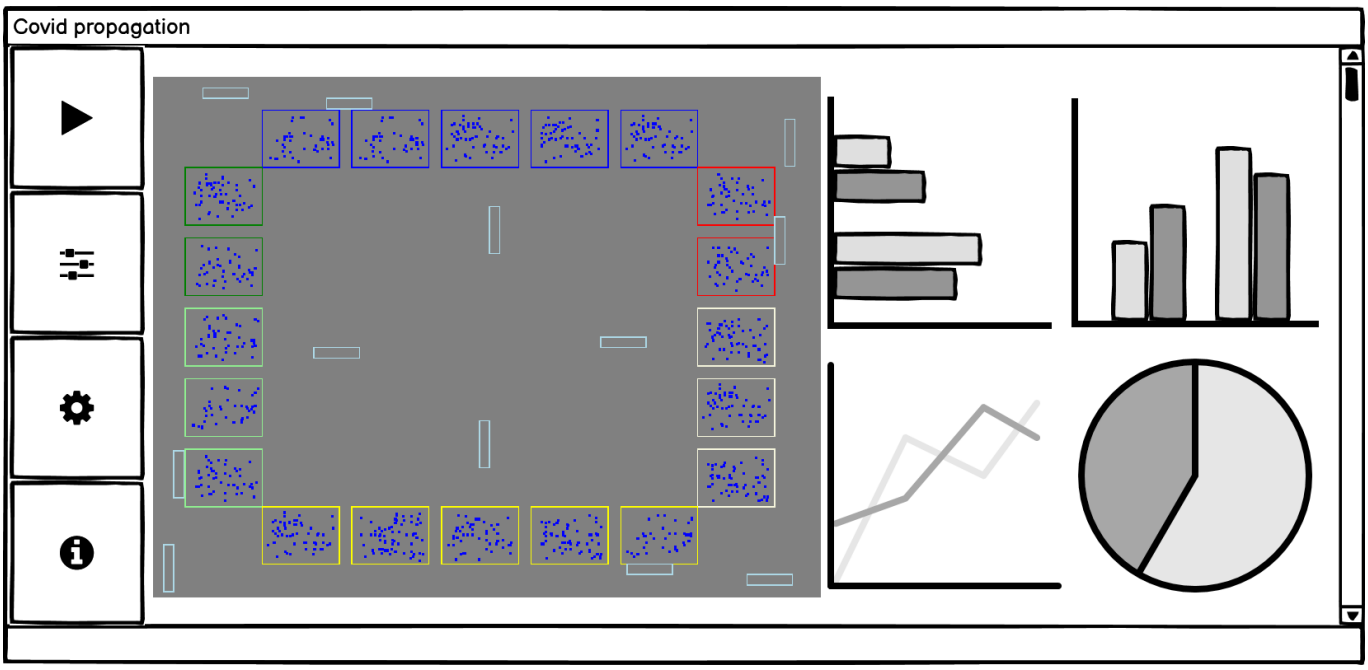


Figure 1: Maquette page de simulation

La page simulation permet la visualisation de la simulation via les graphiques, l'interface graphique ou des données brutes.

Page Paramètres graphiques

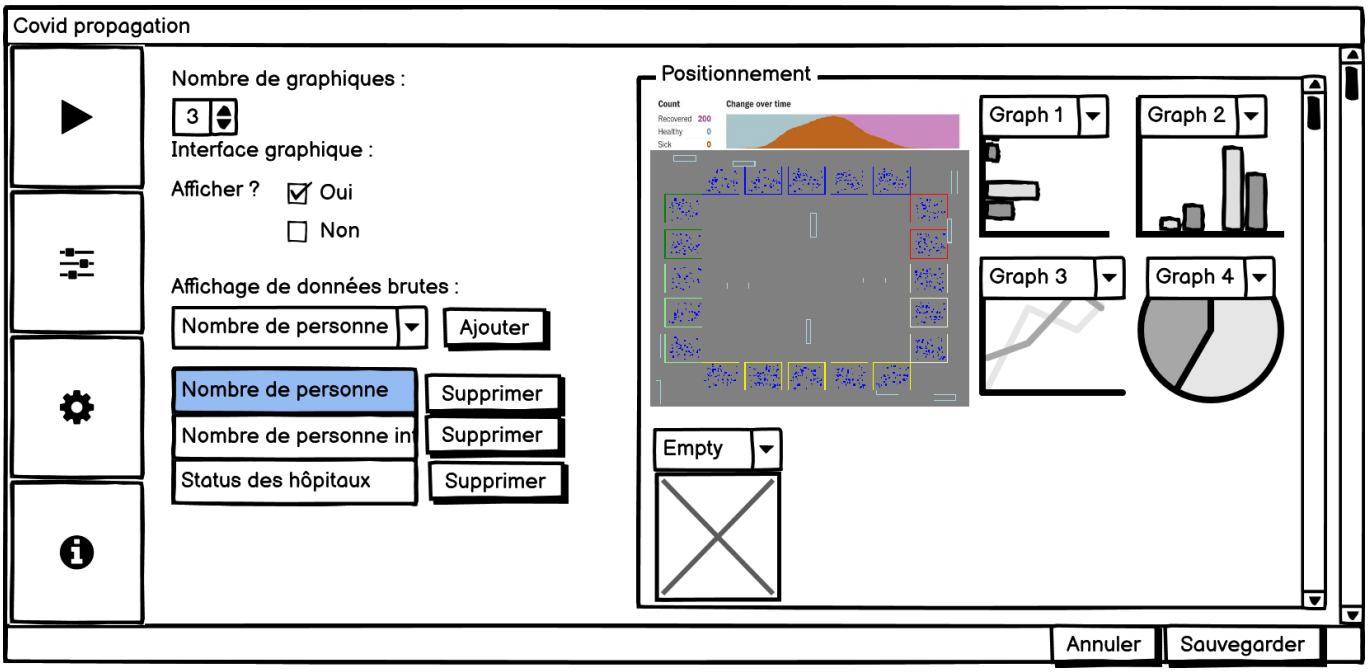


Figure 2: Maquette page de paramètres graphiques 1

Cette page permet à l'utilisateur de modifier l'interface graphique qui s'affichera dans la page simulation.

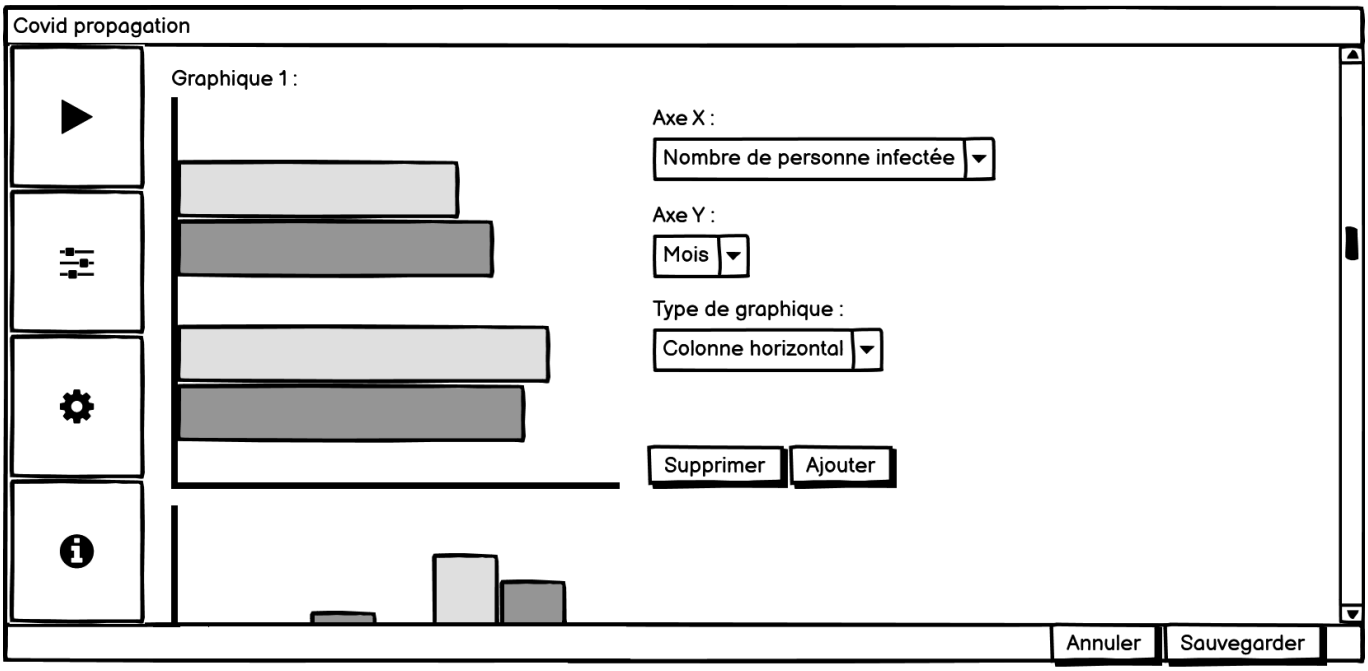


Figure 3: Maquette page de paramètres graphiques 2

Cette section permet de modifier en détail les paramètres d'un graphique.

Page Paramètres

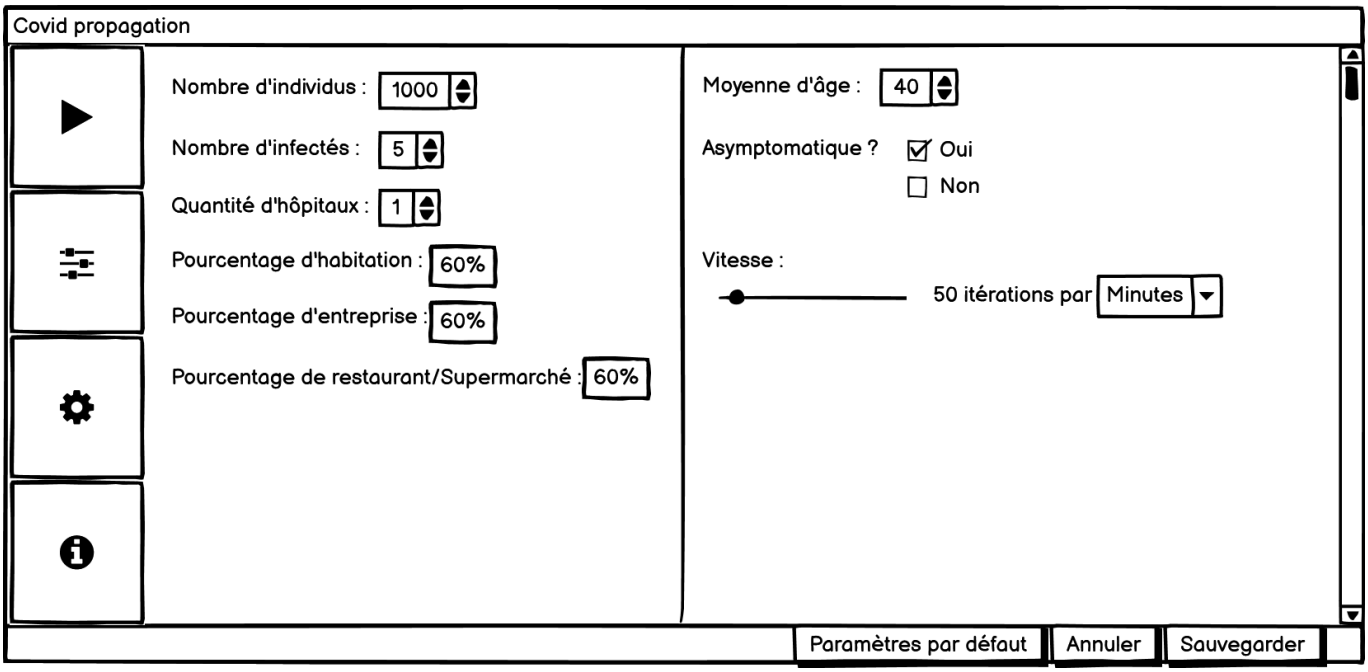


Figure 4: Maquette page de paramètres

Cette page permet à l'utilisateur de modifier les paramètres de la simulation.

Page Informations

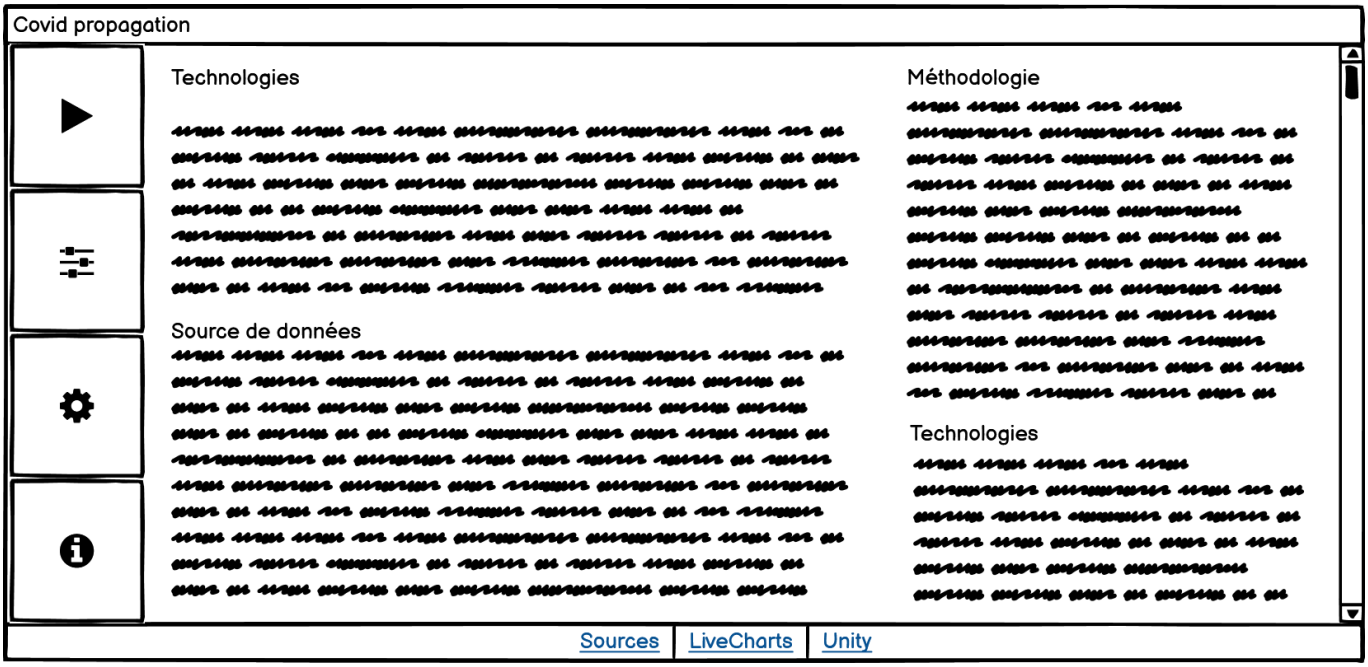


Figure 5: Maquette page d'informations

Cette page sert d'aide à l'utilisateur sur le fonctionnement de l'application et contient aussi des informations sur sa logique et ses sources.

Interface Graphique

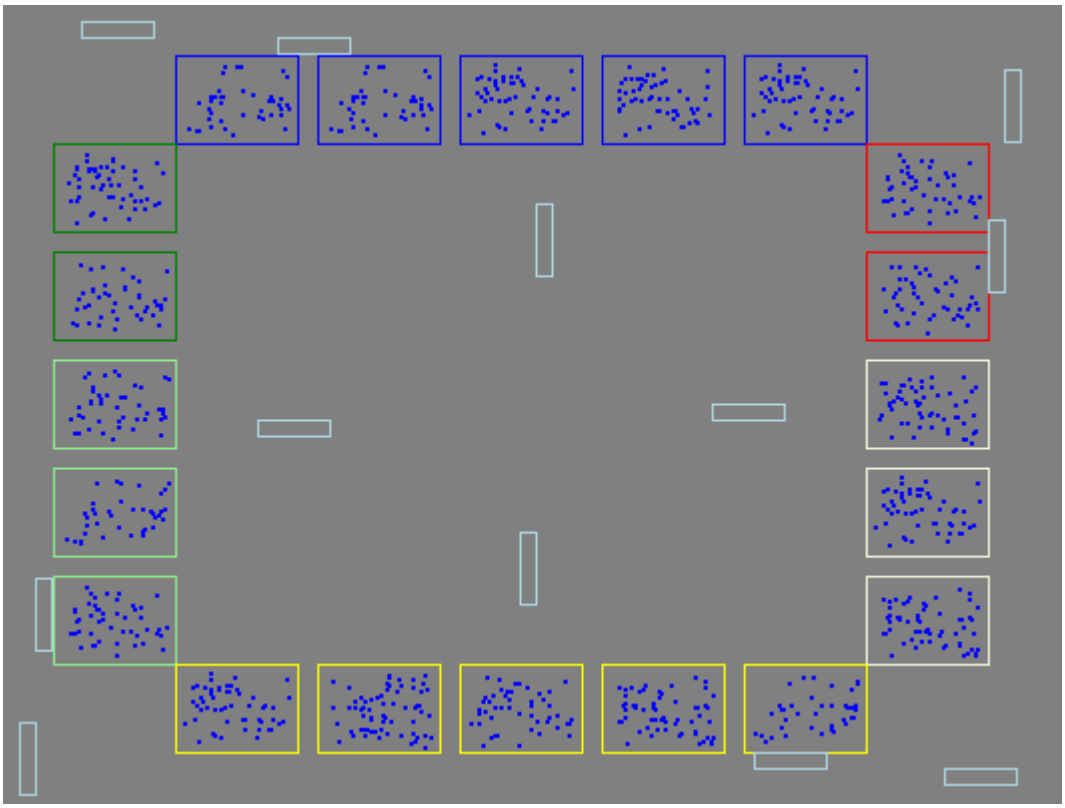


Figure 6: Maquette interface graphique

L'interface graphique permet de visualiser ce qui se passe dans la simulation. Les bâtiments, les véhicules et les individus sont affichés ainsi que leur déplacement et leur statut.

Organisation

Planification

Pour la planification du travail de diplôme, j'ai décidé d'utiliser Excel qui permet de réaliser un planning simple et très compréhensible. Le planning me servant de fil conducteur et de moyen d'organiser l'ordre d'exécution des tâches que j'ai créé.

Tâches

Le traçage des tâches s'effectue sur github en suivant le modèle de scrum. Les tâches à effectuer sont dans une section "To Do" les tâches qui sont en cours, sont dans la section "In progress" et finalement les tâches terminées sont dans la section "Done".

Les sprints sont tous séparés ayant des tâches différentes.

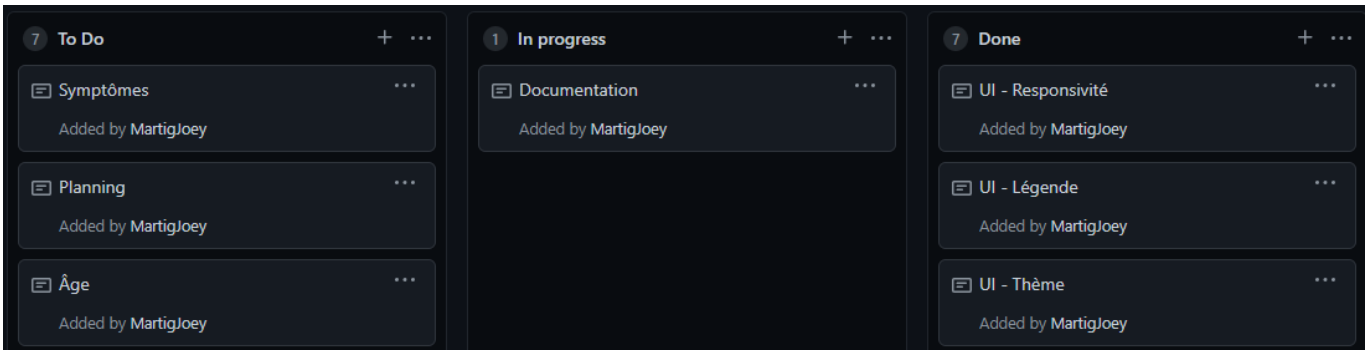


Figure 7: Gestion des tâches

Versionning - Backup

Le versionning est fait à l'aide de github. Au moins deux sauvegardes sont faites chaque jour. Une à midi et une en fin de journée. En cas de perte de données, je ne perds qu'une demi-journée dans le pire des cas.

Ayant eu des problèmes avec git par le passé. (Corruptions de fichiers - conflits) J'ai décidé de faire une sauvegarde supplémentaire sur un disque dur externe. La fréquence de sauvegarde étant plus faible, mais suffisante étant donné qu'il s'agit d'une sauvegarde de secours.

Communications

Nous nous sommes mis d'accord sur le fait de se contacter au moins une fois par semaine pour vérifier l'avancement du projet ou poser différentes questions. Sachant que nous pouvons nous contacter à tout moment par mails. Nous avons établi des intervalles réguliers chaque semaine:

- Mardi --> Meet
- Vendredi --> En personne

Technologies utilisées

C#

C# est un langage de programmation orienté objet développé dans les années 2000 par Microsoft. Sa première version a été adoptée comme standard international en 2002 par Ecma. Il est régulièrement mis à jour, des versions majeures sont publiées tous les 2 à 3 ans environ. La dernière version de C# est la version 8.0 et c'est avec celle-ci que j'ai développé l'application. Son environnement de développement Visual Studio permet de créer des applications Windows facilement.

Microsoft Visual studio

Microsoft Visual Studio est une suite de logiciels disponibles sur Windows et mac. La dernière version qui est la version utilisée dans la réalisation de ce projet est la version 2019.

Il permet de générer des services web XML, des applications web ASP .NET, des applications Visual basic, Visual C++, Visual C#. C#, C++ et basic utilisent tous les mêmes IDE, ce qui permet de partager certaines ressources.

WPF

Windows Presentation Foundation (WPF) ou nom de code Avalon est une spécification graphique de .NET 3.0. Il utilise le XAML qui le rapproche d'une page HTML avec un système de balise. Il est apparu en 2006.

WPF comparé à WinForms permet par exemple l'affichage d'une interface responsive et l'utilisation du GPU pour certaines fonctionnalités.

Unity

Unity est un moteur de jeu développé par Unity Technologies. Il est majoritairement utilisé par des petits studios et des indépendants pour la création de jeux. Il est compatible avec le C# et le JavaScript qui permet de réaliser les scripts. Il permet de développer des jeux compatibles avec Windows, Mac OS X, iOS, Android, TV OS, PlayStation 3, PlayStation Vita, PlayStation 4, Xbox 360, Xbox One, Xbox One X, Windows Phone 8, Windows 10 Mobile , PlayStation Mobile, Tizen, Oculus Rift, Wii U, Nintendo 3DS, Nintendo Switch, WebGL.

XML

XML qui est un acronyme pour Extensible Markup Language. C'est un langage de balises et fait parti du sous-ensemble du standard Generalized Markup Language (SGML). Il a été créé en 1999.

Le bute premier du XML étant de permettre au SGML d'être utilisé sur le web de la même manière que le HTML. Dans mon cas, il permettra de stocker certaines données du programme.

LiveCharts

LiveCharts est une librairie C# permettant de créer des graphiques. Il permet d'inclure une grande quantité de graphiques à des projets, de lier les données au code. Lorsque les données changent, les graphiques s'adaptent automatiquement et sont animés. Les graphiques sont personnalisables et interactable. Il est même possible d'importer des cartes composées de régions en tant que graphique.

En plus d'ajouter énormément d'éléments graphiques et animations, LiveCharts est très performant et peut par exemple afficher des graphiques contenant plus de 100'000 points tout en restant fluides.

JSON

JavaScript Object Notation (JSON) est un format de données dérivé de la notation des objets JavaScript. Il permet d'afficher la structure d'une information comme par exemple un objet C# ainsi que ses données. C'est le format de données qui me permet de communiquer avec le programme Unity depuis le programme WPF.

Cahier des charges

Titre

Covid propagation

Fonctionnalités

- Simulation
  - Population
    - Mesures

        - Hôpitaux
        - individus
            - Patient à risque
            - Âge
            - Décès dû au virus
            - Famille
            - Cercle d'amis
            - "Vie" `Calendrier`
    - Virus
        - propagation
        - effets sur les individus
            - De “Aucun”
            - À “Grave”
    - Hôpitaux
        - Places limitées
    - Mesures de sécuritées
        - Port du masque
        - Quarantaine
        - Confinement global
        - Distanciation
- Graphiques
    - Informations sur la population
        - Décès
        - Rétablissements
        - Infecté
        - Sains
    - Informations sur le virus
        - Dangerosité

Matériel et logiciels

- PC techniciens
- Visual studio 2019
- Une connexion internet
- Github

Prérequis

- C#
- Visual studio 2019

Descriptif complet du projet

Méthodologie

Scrum

Description de l’application

Simuler un grand nombre de personnes possédant toutes des variables différentes (âge, résistance immunitaire, etc.), y introduire le virus et observer sa propagation. Il est possible d'affecter des mesures de sécurité, telles que le port du masque ou la distanciation pour observer la possible réduction de la propagation. L'affichage permet de voir en temps réel la propagation du virus et permet de visualiser chaque individu distinctement au besoin. Des graphiques sont aussi présents pour avoir une idée en chiffres de ce que signifie l'affichage.

Graphique

Les données des graphiques sont choisies par l'utilisateur et donc personnalisables. Plusieurs graphiques peuvent être affichés en même temps. Leur position est définie par l'utilisateur au sein de la page de l'application.

L'interface graphique est fournie par LiveChart. Les données sont directement fournies par l'application ainsi que les échelles de grandeurs qui sont ajustées automatiquement. Les graphiques à courbes et en forme camembert sont disponibles.

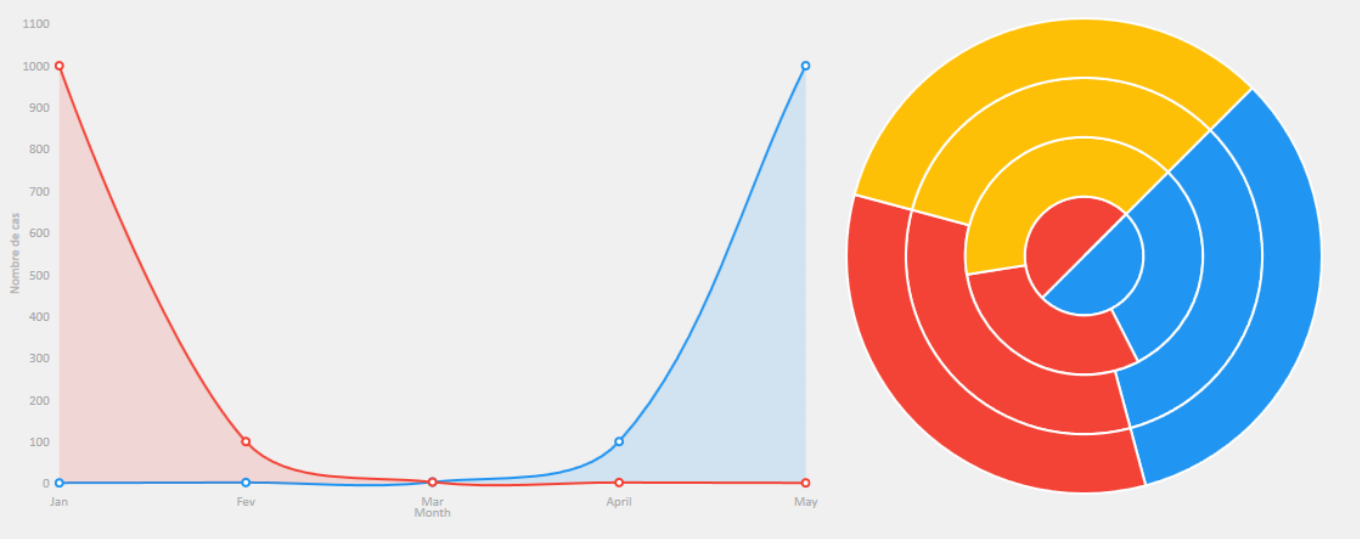
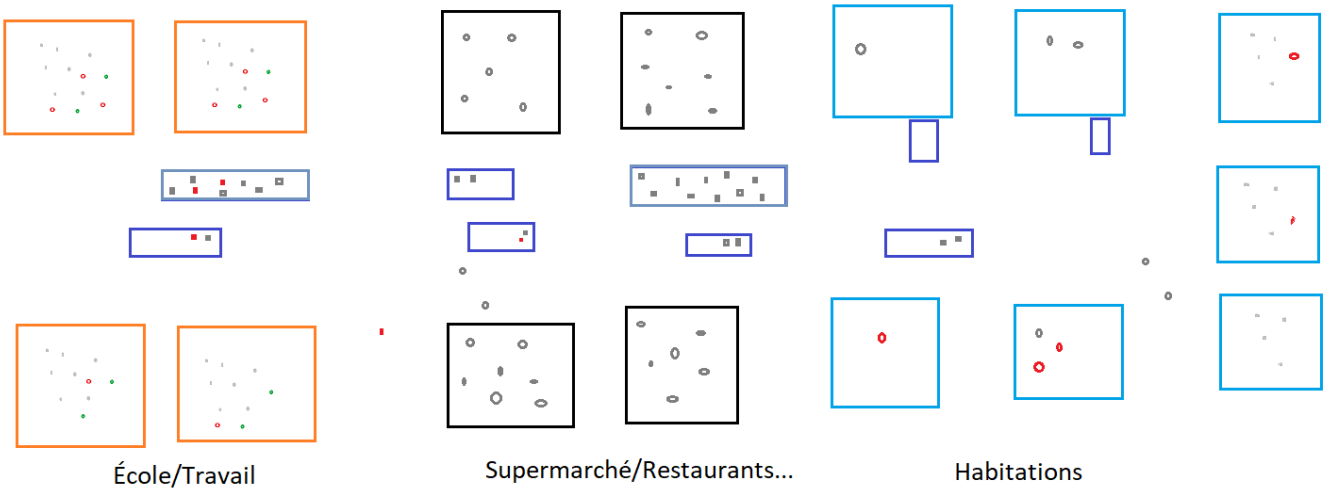


Figure 8: Exemple de graphiques

Interface graphique

En plus des graphiques, une interface graphique affichant les individus ainsi que leur lieur de travail, habitation et déplacement est disponible. Elle permet d'avoir une visualisation plus naturelle de la situation. Elle est très simple, car simuler une ville est une tâche trop complexe et longue pour être ajoutée au projet. Il s'agit donc d'une aide visuelle simple de la simulation. Il n'y a donc pas de routes ou autres éléments

complexe similaire. Voici deux exemples d'interface graphique :



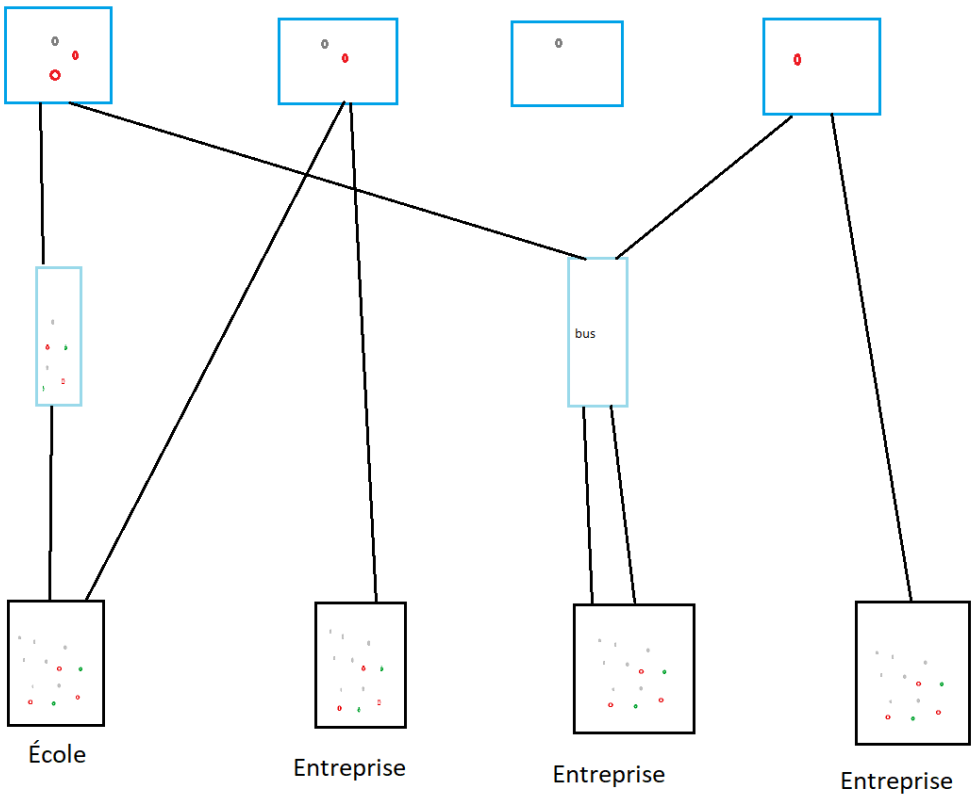


Figure 9: Exemple d'interface graphique

Propagation

La propagation se fait à l'aide de calcul et de différentes variables. 1000 m^2 contenant 10 individus à l'intérieur aura de faibles chances de transmettre le virus. Le même nombre de personnes dans un espace clos de 10 m^2 aura des résultats totalement différents.

La température est prise est compte ainsi que les mesures telles que le masque. Le masque réduit les chances de transmettre le virus. La température, elle fait varier la durée de vie du virus à l'extérieur d'un hôte. La complexité de ce type de calcul étant d'une difficulté largement supérieure aux compétences acquises en tant

que technicien, je me base sur cette fiche Excel réalisée par des professionnels. Elle est très bien documentée et sourcée.

Fiche Excel

Population

La population est constituée d'objets C# générés partiellement, aléatoirement en fonction des paramètres de la simulation. Ils informent la simulation en cas de changement d'état (sain, infecté, etc.). Des itérations sont faites dans la simulation pour calculer si un individu est infecté ou non durant le temps écoulé. Il a un planning simple à suivre dans sa journée qui peut être constituée de par exemple :

- Être dans son habitation
- Prendre le bus
- Travailler
- Prendre le bus
- Faire les courses dans un supermarché
- Prendre le bus
- Et finalement rentrer chez sois

Ce planning est différent en fonction des individus même si vaguement le même. Durant sa journée, il croisera d'autres individus et à chaque itération, il aura des chances d'être infecté si des personnes aux alentours le sont. En fonction du lieu, il rencontrera des personnes différentes, parfois les mêmes comme dans son travail où ses collègues sont fixes. Dans le bus, des variations seront possibles. Son cercle d'amis ainsi que sa famille, lorsqu'il se trouve dans son habitation, seront les individus risquant de le contaminer.

Temporalité

Le quotidien des individus est défini par la simulation lors de leur création. Elles peuvent évoluer avec l'âge des individus.

Une itération est équivalente à ~30min dans la simulation. À chaque itération, chaque individu calcul ses chances d'attraper le virus en fonction de son environnement et des mesures prises. Elle permet aussi à un individu d'évoluer dans son quotidien en passant d'une tâche à une autre par exemple. Leur permettant aussi de changer de lieu et tous les événements liés à l'agenda des individus ainsi que la propagation du virus. La "durée" de la simulation est définie par l'utilisateur et peut donc durer plusieurs jours.

Individus

Les individus possèdent différents paramètres qui vont modifier leur quotidien ainsi que leur résistance au virus. La valeur la plus essentielle est l'âge de ces personnes. L'âge permet de contribuer à la modification de la résistance au virus. Il modifie aussi le quotidien en définissant si la personne va travailler, va à l'école, est libre de faire ce qu'il souhaite ou rien si trop jeune. L'âge évolue avec le temps de la simulation.

Chaque individu a un entourage qui peut le contaminer. Il possède un cercle d'amis avec lequel il peut y avoir des contacts à domicile, et avec lequel il y aura des contacts en extérieur. Il a aussi une famille avec qui les

contacts se font majoritairement à domicile même s'il peut y avoir des déplacements groupés. Par exemple déposer des enfants à l'école, aller au restaurant en famille. Finalement, il a des collègues/camarades qui sont des contacts qui se trouve dans les écoles ou lieu de travail et qui sont ne définit pas ceux-ci.

Les moyens de transport des individus sont choisis par la simulation en fonction des paramètres de celle-ci. Un individu possédant une voiture aura beaucoup moins de risque de propager le virus qu'en prenant les transports publics. Il est cependant possible que d'autres personnes du cercle familial ou du cercle d'amis utilisent le même véhicule. De ce fait, il n'est pas forcement 100% sécurisé. Les transports publics eux ont des risques élevés, car beaucoup de monde se situe dans le même véhicule de taille moyenne. En plus de cela, les individus sont en contact avec des étrangers qui peuvent varier en fonction des jours augmentant encore plus les chances de contagion.

La résistance au virus des individus défini si la personne a des symptômes en cas d'infection, si elle est asymptomatique, ou si elle a besoin de soins. Ce paramètre est défini par pourcentage. De 100% à 90% de résistance, l'individu est asymptomatique. De 90% à 50% de résistance, l'individu a des symptômes tels que la toux. De 50% à 10% de résistance, la personne est hospitalisée. Et finalement, à moins de 10%, l'individu est hospitalisé et risque la mort.

- Plus ce paramètre est haut moins les effets du virus sont présents
- 90-100 => asymptomatiques
- 90-50 => symptômes normaux
- 50< => hospitalisations
- 10< => décès

Chaque individu créé commence avec une valeur entre 80 et 100. Sachant qu'environ 5% de ces individus ont plus de 90 de résistance. Des maladies peuvent entrer en compte et baisser la résistance naturelle. Plus l'âge est élevé, plus l'individu sera impacté par un grand nombre de maladies et celles-ci seront plus dangereuses.

Les maladies sont inspirées de maladie réelle impactant l'effet du covid. Cependant, dans la simulation, elle n'affecte que la résistance au virus. Ces maladies apparaissent de façons aléatoires et plus fréquemment sur les individus dont l'âge est élevé. Elles ne se propagent pas. Elles sont en partie assignées au départ par la simulation puis apparaissent avec le temps. Elles réduisent la résistance au covid de 1% à 20% en fonction de la maladie et de l'âge de la personne.

Hôpitaux / écoles / entreprise

Ces différents lieux fonctionnent de façon similaire. Ils ont tous des individus en leurs seins qui peuvent se transmettre le virus. Ils ont des tailles différentes en fonction du nombre de personnes pouvant être à l'intérieur.

Les hôpitaux fonctionnent légèrement différemment. Ils ont des patients ainsi que des membres du staff de l'hôpital. Il y a donc des différences de mesures et quantités. Les patients sont là de manière temporaire en fonction du nombre de personnes attrapant le covid.

Les écoles ont une situation similaire en ayant des élèves ainsi que des profs qui ont des mesures et quantités différentes.

Les entreprises elles fonctionnent en groupe d'individus, similaire aux classes des écoles, mais sans personnel ayant des mesures différentes des autres.

Protocole de tests

Ce projet étant en c#, je vais utiliser les tests unitaires intégrés dans visual studio.

Les tests unitaires ne garantissant pas qu'il n'y ait aucun bug dans l'application, je vais créer des scénarios que je testerais avant et après chaque implémentation de fonctionnalités. Ces scénarios auront pour but de couvrir un maximum de possibilités pour éviter l'apparition de bug dû à une modification du code ou l'ajout d'une fonctionnalité. Ils permettent aussi de trouver d'éventuels des problèmes d'ergonomie en me plongeant à la place d'un utilisateur.

Persona

Utilisateur expérimenté

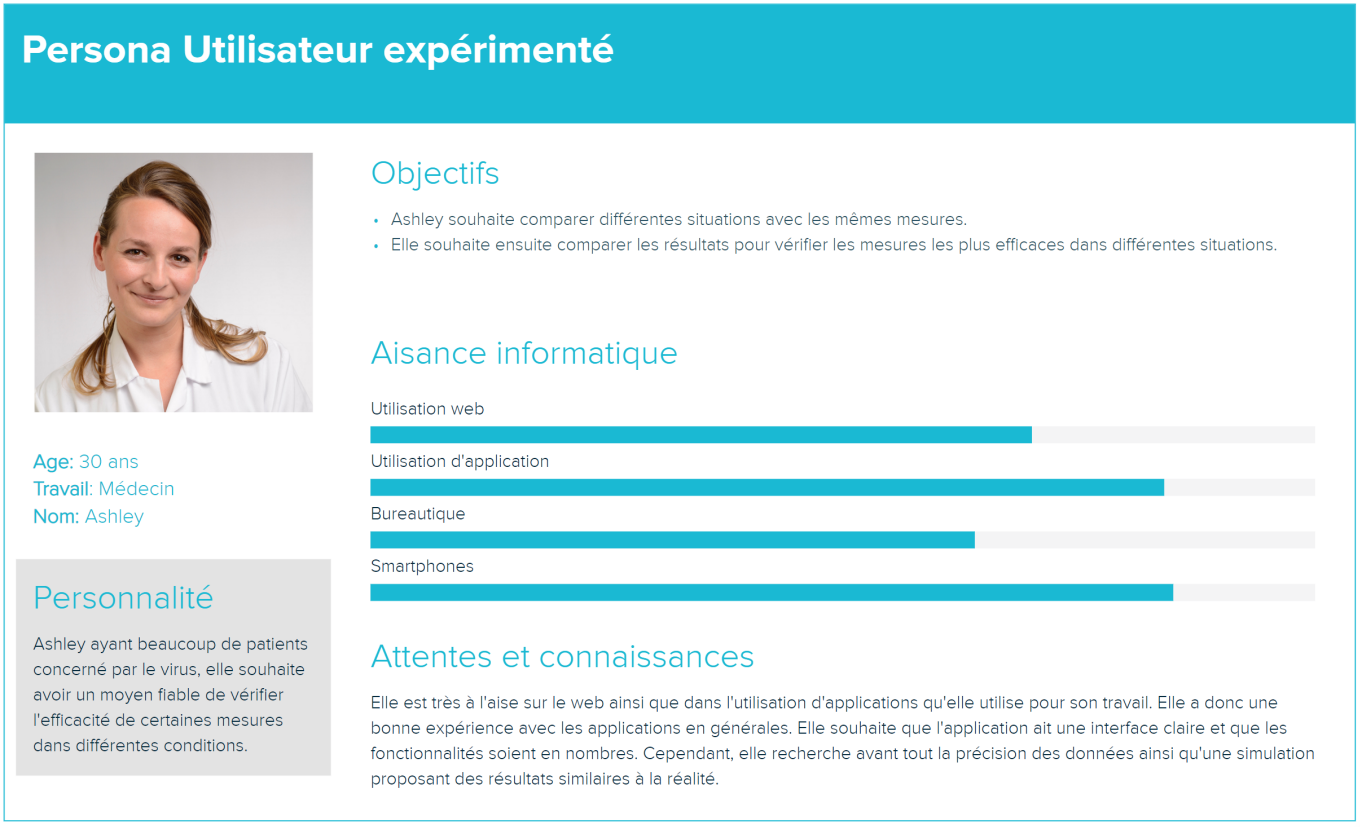


Figure 10: Persona experimenté

Utilisateur inexpérimenté

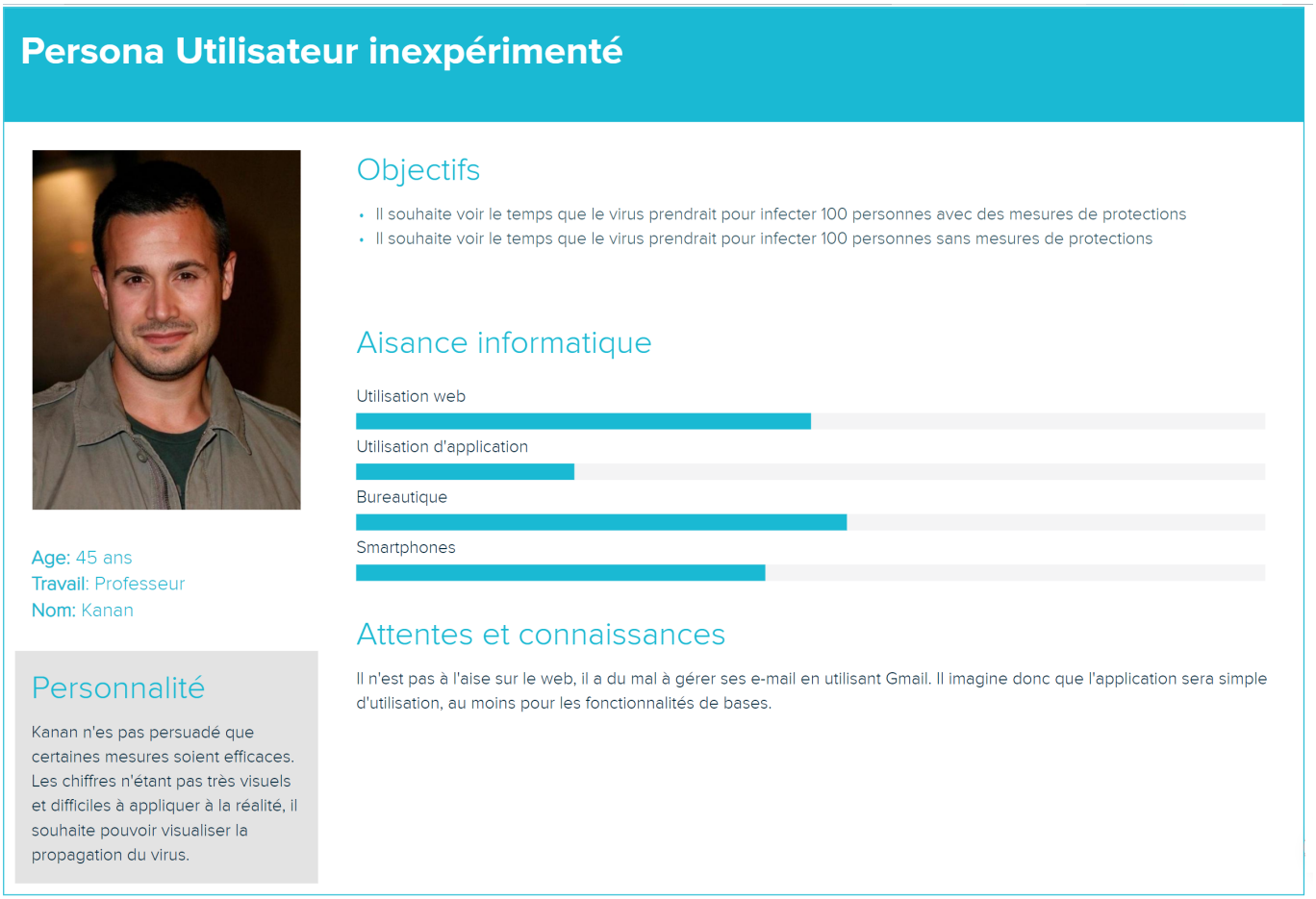


Figure 11: Persona inexpérimenté

User stories

Ashley

En tant que Ashley
Je veux comparer différentes situations avec différentes personnes en prenant des mesures identiques
Afin de pouvoir observer les différences et déterminer quelles mesures est efficaces dans quelle situation.

scénarios Je crée sans soucis une situation à l'aide de l'application. Pour ce faire, j'entre différents paramètres, tel que le nombre de personnes, les mesures prisent pour limiter la transmission ainsi que d'autres paramètres.
J'observe la simulation et prends note des résultats.
Une fois terminée, j'en lance une autre avec certains paramètres différents et prends note des résultats.
Je compare les résultats avec la simulation précédente et effectue ma conclusion.

Kanan

En tant que Kanan
Je veux vérifier l'efficacité de différentes mesures prisent pour éviter la propagation du covid
Afin de afin de me donner une idée concrète et visuelle de l'efficacité de ses mesures.

scénarios Je lance l'application et cherche à créer une simulation. Une fois trouvé, je peux voir les mesures qui apparaissent clairement, d'autres paramètres sont disponibles, mais je n'y touche pas.
Une fois la simulation lancée, je vois un message m'indiquant que celle-ci commence.

Des aides sont disponibles me permettant de comprendre les données qui sont affichées.
Après avoir terminé cette simulation, j'en lance une autre en désactivant les mesures.
Je relance la simulation et observe la différence entre les deux simulations.

Diagramme d'activité

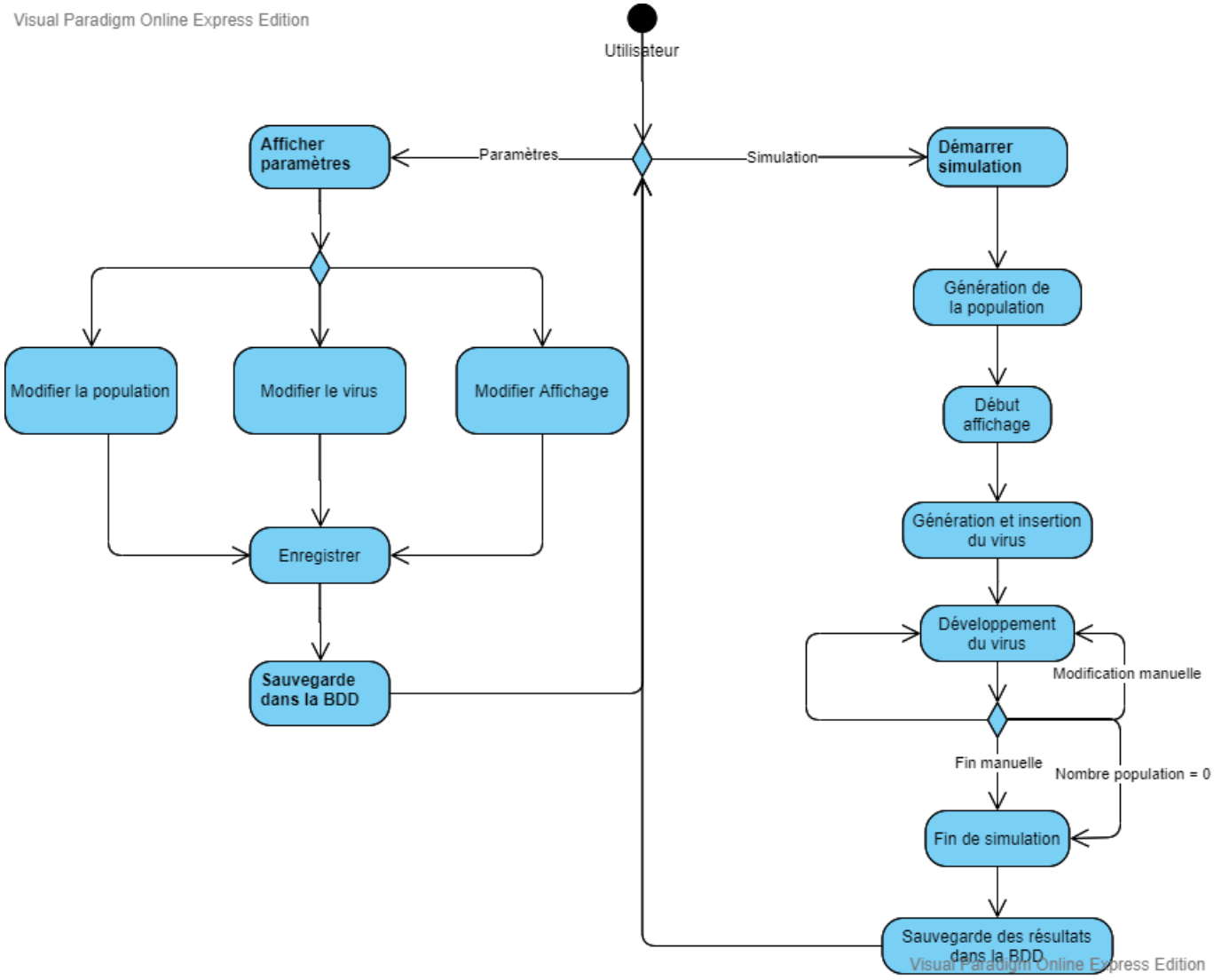


Figure 12: Diagramme d'activité

Planning

https://docs.google.com/spreadsheets/d/1tSpIbcDVvGnzMhEN71UDwPOxEy0oapQSSbxzjqXt3RA/edit?usp=sharing

Diagramme de classe initial

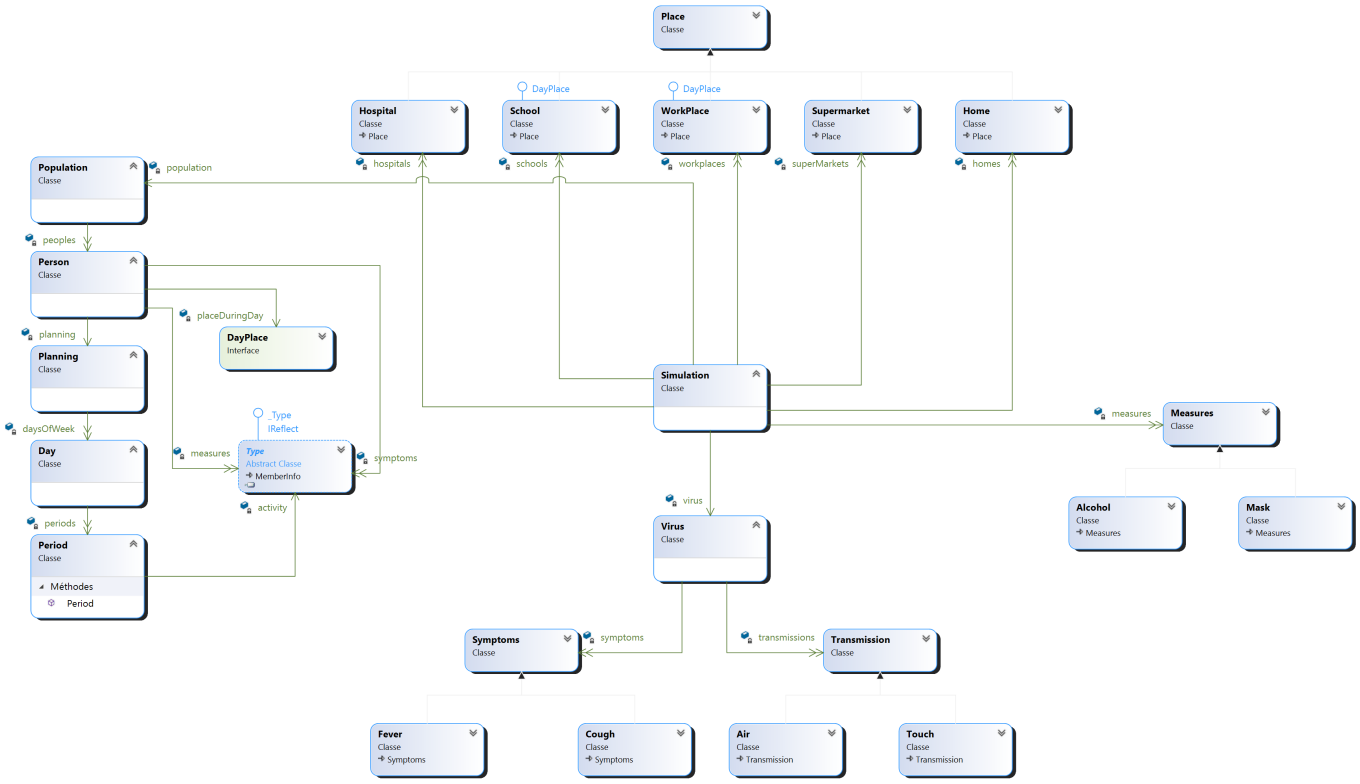


Figure 13: Diagrame de classe initiale

Interactions

Menu principal

- Affiche un preview de l'affichage de la simulation
- Btn Paramètres
  - Population
    - Remplace l'affichage actuel se situant à droite pour afficher les paramètres de la population
  - Virus
    - Remplace l'affichage actuel se situant à droite pour afficher les paramètres du virus
  - Affichage
    - Remplace l'affichage actuel se situant à droite pour afficher les paramètres de l'affichage
- Btn lancer la simulation
  - Change l'affichage de la totalité de l'application, affiche une barre de chargement indiquant l'état de création de la simulation.

Population

Affiche une page avec les paramètres suivant :

- Écoles / lieux de travail
  - Différentes selon l'âge
  - Zone de transmission
- Familles / Cercles d'amis
  - Transmission
- Moyenne d'âge de la population

    - Permet de modifier la moyenne d'âge de la population de 1 à ~100
    - Permet de délimiter une limite d'âge maximal et minimal
    - Il est possible de le laisser par défaut
- Nombre d'individus
    - Le nombre d'individus simulé dans une population
    - La limite n'est pas définie par le programme
    - L'utilisateur connaît les limites de sa machine
- Mesures
    - Permet de sélectionner plusieurs mesures
    - Les mesures ont un pourcentage d'efficacité
    - Permet de réduire les chances de propagation du virus
        - Affecte différemment le virus en fonction de la mesure
        - Pourrait totalement contrer un virus
    - Peut-être modifier par l'utilisateur jusqu'à un niveau de 100% de protection
    - Valeur par défaut défini par des études sur le sujet
    - Appliquer uniquement sur certaines parties de la population
        - Infectés
        - Sains
        - À risques
- Cercle social
    - Ami
    - Famille
    - Collègues
    - ...
    - Transmissions accrues
    - Rencontres inclussent dans le planning journalier des individus
- Hôpitaux
    - Il y a plusieurs hôpitaux avec les options :
        - Copier
        - Coller
        - Appliquer sur tout
    - Permet de modifier le nombre d'hôpitaux
    - Permet de modifier le nombre de places
    - Stabilise les individus y étant admis
        - Réduis leurs chances de décès
    - Nécessite du personnel qui peut être infecté pour fonctionner
        - Mesures du personnel :
            - Permet de sélectionner plusieurs mesures
            - Les mesures ont un pourcentage d'efficacité
            - Permet de réduire les chances de propagation du virus
                - Affecte différemment le virus en fonction de la mesure
                - Pourrait totalement contrer un virus
            - Peut-être modifier par l'utilisateur jusqu'à un niveau de 100% de protection
- Btn annuler
    - Annule les modifications faites à l'hôpital
    - Réaffiche les données précédemment affichées

- Btn sauvegarder
    - Sauvegarde les paramètres choisis par l'utilisateur

Virus

Affiche une page avec les paramètres suivant :

- Effet sur le corps
    - Permet de modifier le pourcentage de propagation en fonction du symptôme (toux)
    - Les effets mortels nécessitant une hospitalisation
- Moyens de transmissions
    - Sont impacté par les symptômes (en incrémentant l'efficacité)
    - Sont impacté par les mesures (en décrémentant l'efficacité)
- Durée
    - Permet de définir la durée durant laquelle le virus prend effet
- Asymptomatique
    - Permet de définir si oui ou non il y a des asymptomatiques
    - Permet de définir le pourcentage d'asymptomatiques

Affichage

Affiche une page avec les paramètres suivant :

- Graphiques
    - Permet de sélectionner différents styles de graphiques à afficher
        - Permet de sélectionner une donnée au choix en X et en Y
        - Un exemple du graphique avec les données est affiché à côté de la barre de sélection
    - Plusieurs graphiques possibles à sélectionner
- Affichage d'une "carte" permettant une visualisation plus simple

Simulation

Affiche une page :

- Affichage d'une barre de chargement lors de la génération de la simulation
    - Évolue en fonction du nombre d'individus créé
- Affiche les graphiques sélectionnés
    - Onglets permettant de sélectionner quel graphique affiché
    - Possibilité d'afficher jusqu'à 4 graphiques sur le même onglet
- S'actualise toutes les secondes (environ)

Livrables

- Mind Map
- Planning
- Rapport de projet
- Manuel utilisateur
- Journal de travail ou LogBook
- Résumé / Abstract

Environnement

L'environnement de travail est composé d'un pc technicien, 3 écrans, clavier, souris et d'un SSD amovible avec Windows 10 pro version 10.0.19042 Build 19042. Le code est réalisé à l'aide de visual studio 2019 versions 16.9.2. La documentation et le logbook sont réalisés à l'aide de visual studio code et des extensions Markdown All in One et Mardown PDF.

Le projet WPF utilise .core 3.1 qui est la version lts. La version d'Unity est la 2020.3.4f1 qui est aussi la version lts.

Architecture

Arborescence

```
├── CovidPropagation
│   ├── .vs
│   ├── CovidPropagationGraphicInterface
│   └── CovidPropagationGraphicInterface.sln
├── Documentation
│   ├── Medias
│   │   ├── LogBook
│   │   └── Rapport
│   ├── LogBook.md
│   └── Rapport.md
└── POC
    ├── TestUnity_WPF
    └── testWPF_Unity.sln
```

Structure

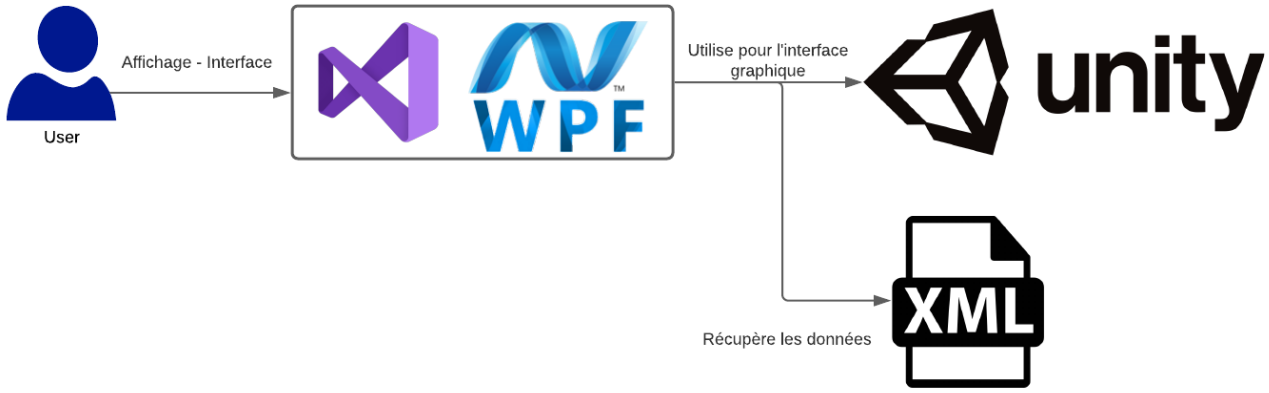


Figure 14: Diagramme de fonctionnement

Analyse interface graphique

Cette analyse concerne l'interface graphique et le choix de la technologie à utiliser pour réaliser celle-ci.

Comparaison technologies

WinForm (Windows Forms)

Lors du CFC ainsi que de l'apprentissage technicien, nous avons toujours utilisé cet interface pour réaliser l'entièreté de nos projets. Je connais donc bien cet environnement contrairement au WPF. En plus de cela, l'interface graphique réalisée dans le poc est en WinForm. Me permettant donc de simplement importer le projet déjà existant.

Cependant, WinForm ainsi que l'interface graphique déjà existante apportent de gros problèmes tels que les timers. Lorsqu'il y a une charge CPU trop lourde, les timers perdent leurs rythmes et n'arrivent plus à suivre. Le résultat de cette surcharge est que plus rien ne fait de sens. Les animations n'ont plus le temps de s'effectuer rendant les individus immobiles ou presque.

WPF (Windows Presentation Foundation)

WPF est plus récent que WinForms et a donc certains avantages non négligeables en comparaison. Il est beaucoup plus complet en termes d'esthétique et donc d'UI que WinForms. En plus de cela, il est possible de créer des objets en 2D ou 3D. Ces objets contrairement à WinForms sont gérés par le GPU plutôt qu'être entièrement basé sur le CPU. Cette différence à elle seule fait pencher la balance pour WPF.

La liaison entre la vue et les données est aussi plus efficace ce qui est très important dans mon cas.

Cependant, une application WPF ne peut pas être lancée sur un mac ou sur Linux. C'est un gros désavantage, mais dans le cadre de ce travail, il me semble négligeable.

Le possible problème de timer bien que probablement réduit du au fait que la charge du CPU est allégée par la carte graphique, risque d'être toujours présent.

Il faut aussi noter que je n'ai aucune expérience en WPF et vais donc devoir m'y habituer durant un certain temps avant d'être efficace à 100%.

Unity

Unity est un moteur de jeu en 2D et 3D. Il est possible de l'intégrer directement à une application WPF. Ça me semble être le meilleur choix si l'on prend en compte les problèmes de timer des deux autres technologies. Unity possède de façon native des méthodes qui sont appelées à chaque frame permettant le bon déroulement de la simulation.

En plus de cela, j'ai beaucoup d'expérience avec ce logiciel, ayant réalisé mon TPI avec celui-ci. Je peux donc affirmer qu'il est beaucoup plus simple de réaliser l'interface graphique avec Unity.

Cependant un autre problème est présent. La liaison des données. Il m'est impossible, sans le tester, de savoir si ce modèle de fonctionnement est compatible avec mon projet. Je sais qu'il est possible de transférer des informations de WPF à Unity cependant, je ne sais pas si la fréquence d'envoi est suffisante ou même si la quantité de données envoyées que je souhaite atteindre est possible.

Communication

Pour communiquer entre WPF et Unity, j'ai essayé plusieurs méthodes fonctionnant différemment et surtout de complexité différente.

Unity Controller

Mon premier essai fut avec Unity Controller qui permet de créer un serveur qui communique entre une application C# et Unity.

Pour l'installer il faut d'ajouter le paquet nuget "Unity Controller" au projet ainsi qu'un using "UnityController". Son implémentation est la plus simple des solutions testées sachant qu'elle ne prend que quelques lignes au total.

Le code dans un script Unity ne comprend que deux lignes. La première étant le démarrage du serveur.

```
UnityCommands.StartServer("008");
```

La deuxième s'updatant à chaque image, permettant de recevoir la commande et de l'appliquer.

```
UnityCommands.ReceiveMessage();
```

Maintenant, dans le projet Windows. Dans l'initialisation de la form, il faut démarrer le serveur en localhost.

```
UnityCommands.StartClient("localhost", "008");
```

La dernière ligne située dans un événement click d'un bouton permet de modifier l'élément texte du GameObject "GameObjectText" en lui ajoutant la valeur "Texte".

```
UnityCommands.UpdateText("GameObjectText", "Texte");
```

Cette implémentation de la communication est extrêmement simple à mettre en place cependant, les possibilités sont très limitées. Les seules actions possibles sont le fait de changer le texte d'un GameObject, sa couleur, son image, etc. Il est impossible d'envoyer un message de code à code puis de l'interpréter. Cette façon de faire ne peut donc pas servir à la réalisation de mon projet qui demande un traitement des données.

PipeLines

Contrairement à UnityController, les pipelines laissent plus de liberté, mais leur complexité est bien supérieure. J'ai rencontré divers problèmes en implémentant cette fonctionnalité.

Dans mon cas, la communication se fait à sens unique, WPF donnant les informations à l'interface graphique se trouvant sur Unity. Il faut donc commencer par créer un serveur du côté WPF.

Écriture

L'écriture se situe dans le projet WPF. Cette méthode permet de créer le serveur, le démarrer le serveur et d'établir une connexion avec le client qui est Unity. La méthode ServerThread sera appelée lors de l'appelle de la méthode sever.Start().

```
Thread server;
server = new Thread(ServerThread);
server.Start();
```

Lors du démarrage du Thread, le pipeline est créé et le serveur attend que le client se connecte. Une fois qu'il est connecté, un objet StreamString est créé permettant l'écriture de message pouvant être transféré via le pipeline.

```
NamedPipeServerStream pipeServer = new NamedPipeServerStream("testpipe",
                                        PipeDirection.InOut, numThreads);
pipeServer.WaitForConnection();
ss = new StreamString(pipeServer);
```

Le constructeur de l'objet StreamString récupère le pipeline créé et le transforme en Stream qui est à son tour transformé en BinaryWriter qui permettra l'envoi des données. Créé un objet UnicodeEncoding permettant la

conversion de string en bytes pour le transfert.

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

WriteString est la méthode appelée à chaque fois que des données doivent être envoyées. Elle convertit le message qui lui est fourni en byte et envoie celui-ci dans le pipeline.

```
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 >> 8));
        dataToSend.Add((byte)(len >> 0));
        dataToSend.AddRange(outBuffer.ToList());
        stream.Write(dataToSend.ToArray(), 0, dataToSend.Count);
        stream.Flush();
    });
}
```

Lecture

L'ouverture de la connexion avec le serveur s'effectue dans la méthode "Start" d'Unity qui s'effectue au démarrage du projet. Puis, il appelle la méthode ConnectToServer(). Si la connexion à échouée, un nouvel essai sera effectué à chaque frame du projet jusqu'à que celle-ci soit effectuée.

```
pipeClient = new NamedPipeClientStream(".", "testpipe", PipeDirection.In,
PipeOptions.Asynchronous);
ConnectToServer();
ConnectToServer();
```

ConnectToServer() essai donc de ce connecter, si la connexion est effectuée, un objet SreamString est créé et la lecture du flux commence.

```
pipeClient.Connect();
if (pipeClient.IsConnected)
{
    ss = new StreamString(pipeClient);
    Thread.Sleep(250);
```

```
    ReadPipeData();
}
```

ReadPipeData() est une méthode récursive et asynchrone. Elle permet de lire le résultat reçu par le pipeline. Elle attend la réception d'un message. Une fois qu'elle en reçoit un grâce à ReadStringAsync(), elle le lit et finit par s'appeler elle-même et recommence le cycle.

```
string result = await ss.ReadStringAsync();
ChangingText.GetComponent<Text>().text = result;
ReadPipeData();
```

La lecture du résultat s'effectue exactement comme l'écriture, mais dans le sens inverse. La longueur du message est récupérée et utilisée pour le lire dans son entièreté. Une fois le message reçu, le résultat est converti en string et retourné à ReadPipData() qui pourra effectuer son cycle.

```
public async Task<string> ReadStringAsync()
{
    return await Task.Run(() =>
    {
        int len;
        len = stream.ReadByte() << 8;
        len += stream.ReadByte();
        byte[] inBuffer = new byte[len];
        stream.Read(inBuffer, 0, len);
        return streamEncoding.GetString(inBuffer);
    });
}
```

Intégration

L'intégration permet d'avoir un rectangle au sein de la page WPF qui sera constitué d'une application .exe. Dans ce cas, il s'agit d'Unity. Ça ne permet pas de commander le contenu de la fenêtre, mais uniquement sa taille, position et quand démarrer le .exe.

Cette méthode permet de charger et démarrer le projet Unity qui a été buildée au préalable. UnityGrid étant une grille crée dans la vue du code WPF.

```
<Grid x:Name="unityGrid" Width="454" Height="319" VerticalAlignment="Top"
HorizontalAlignment="Right" Margin="0,10,327,0"></Grid>
```

Cette grille est ensuite transformée en unityHandle qui permet de donner au programme la grid ou il va devoir s'afficher. Le process récupère l'emplacement du programme à lancer. Les arguments permettent de définir où le programme doit se lancer, sans les arguments, le programme se lance dans une fenêtre

indépendante. Ensuite, le process est lancé ce qui démarre le programme. EnumChildWindows (user32.dll) permet de lier le programme lancé à la fenêtre, permettant la modification de sa taille en fonction de la taille du programme WPF.

```
Process process;
HwndSource source = PresentationSource.FromVisual(unityGrid) as HwndSource;
IntPtr unityHandle = source.Handle;

process = new Process();
process.StartInfo.FileName = @".\UnityBuild\testWPF_Unity.exe";
process.StartInfo.Arguments = "-parentHWND " + unityHandle.ToInt32() + " " +
                              Environment.CommandLine;
process.Start();
```

Choix de la solution

Mon attention se porte premièrement sur Unity qui me semble être la solution avec le meilleur rendue et permet de contourner certains problèmes présents dans les deux autres options. Le premier test que j'ai effectué ne permet pas de transmettre des données complexes, uniquement des strings ou images, mais pas de liste c# ou autres éléments que je pourrais utiliser.
Le second test que j'ai effectué avec les pipelines permet de transférer une grande quantité de données (Int32) en string. Il est donc possible de transférer des objets en json d'un projet à l'autre.
C'est cette deuxième option que j'ai donc choisie pour réaliser l'interface graphique. Permettant donc d'utiliser Unity qui est beaucoup plus simple à utiliser pour la réalisation de ce genre de fonctionnalité.

Problèmes rencontrés

Pipeline

Durant l'implémentation des pipelines, j'ai rencontré divers problèmes, le premier étant que la structure originale des pipelines utilise une communication synchrone. Lors de l'attente de données, le programme Unity s'arrêtait complètement jusqu'à la réception de la donnée attendue. Une fois reçue, une exécutait une frame puis attendait à nouveau des données. Le rproblèmes était similaire dans le code WPF qui, après avoir envoyé des données, attendait qu'Unity les ait réceptionnées pour continuer.

Pour pallier à ce problème, j'ai opté pour l'implémentation de l'asynchrone dans la réception et dans l'envoi des données. Concernant l'envoie, j'ai rencontré un léger problème qui m'empêchait d'accéder à une méthode en asynchrone, car j'envoyais le contenu d'un textbox appartenant donc au thread principal. Ce problème a été réglé avec l'utilisation de Dispatcher.Invoke. Ce problème de thread m'a malgré tout pris un certain temps à régler du au fait que WPF, Unity et WinForms utilisent tous une façon d'invoque différente rendant les recherches plus compliquées.

```
await Task.Run(() =>
{
    Dispatcher.Invoke((Action)(() =>
    {
```

```
            ss.WriteString(tbxValue.Text);
        }));
    });
```

Le gros poisson concerne la réception des données. Comme étant un code bloquant, j'ai cherché différentes façons d'appeler mon code de manière constante. Le placer dans la méthode update(appelée chaque frame) ne fonctionnant tout simplement pas, j'ai opté pour la méthode "InvokeRepeating" qui permet de donner un interval dans lequel une méthode sera exécutée.

InvokeRepeating couplé avec de l'asynchrone m'a permis d'éviter le programme de s'arrêter à chaque attente de données tout en étant capable d'en recevoir. Cependant, les données reçues n'étaient pas fidèles aux données envoyées.

Par exemple:
la première réception de données est fidèle à celles envoyées. Lors de la deuxième réception, le message est tronqué et certaines lettres du début de la transmission sont manquantes. La troisième réception est encore plus corrompue, recevant donc un message en caractère chinois. Après cette réception il était courant de ne recevoir des données vides.

Figure 15: Fidélité des données

Pour régler ce problème, j'ai pensé à remplacer InvokeRepeating par une méthode asynchrone récursive. Cette méthode est appelée une première fois au démarrage du script puis s'appelle une fois qu'elle a réceptionné des données. Permettant de recevoir les données correctes et sans bloquer le code.

```
string result = await ss.ReadStringAsync();
ChangingText.GetComponent<Text>().text = result;
ReadPipeData();
```

WPF UI

N'ayant encore jamais travaillé avec WPF, la structure du projet m'a déconcerté au départ, mais j'ai rapidement pu prendre la main. Cependant, pour réaliser une apparence spéciale, j'ai dû modifier certains

outils mis à disposition par WPF. Les boutons étant très similaires à WinForms ne m'ont pas posé de problèmes.

Ce n'est qu'après avoir eu envie de modifier l'affichage d'un slider et après des recherches, que je me suis rendu compte qu'il était impossible de modifier le slider existant pour satisfaire aux paramètres que j'avais choisis. Pour le modifier, il m'a fallu créer un template du slider qui revient à récupérer le code XAML du slider et le modifier à la main. Il m'a fallu un certain temps pour comprendre chaque composant ainsi que leurs paramètres. Même si ça permet de modifier dans le moindre détail l'outil, j'ai été étonné qu'il n'existe par de paramètres facilement modifiables comme le background des boutons par exemple.

Après avoir eu du mal à modifier le slider, j'ai pu modifier les autres outils (RadioBoutons, Checkbox) avec aisance.

Simulation

Structure

Diagramme de classe final - Interactions entre objets

Fonctionnement

Création - Évolution (Itérations)

GUI

Structure

Diagramme de classe final - Interactions entre objets - Réception des données

Fonctionnement

Création - Évolution (Itérations) - Réception des données

UI

Thème

Création - Difficultés

Pages

Simulation - paramètres graphiques - paramètres - légende - Informations

Planning

Prévisionnel

Dans le cadre de ce projet, je vais commencer par réfléchir à la structure générale de celui-ci et de ces interactions entre les différentes sections (Simulation - Graphique - etc.) ainsi que toujours trouver le meilleur moyen d'optimiser le code et la structure pour permettre la simulation d'un plus grand nombre d'individus.

Les timers de Visual studio étant très aléatoire dès lors que le programme nécessite une trop grande charge de travail, je vais essayer de trouver une alternative ou de corriger ce problème en modifiant le timer. Une fois la structure réfléchis et le problème de timer règle, je vais commencer par créer l'interface de l'application où viendront s'ajouter tous les autres composants. Je vais ensuite commencer à créer la population et vérifier que tout fonctionne correctement. Pour la population, j'aurais tout de même besoin d'une esquisse des bâtiments.

Après la population, et pour le second sprint, je pourrais commencer à générer la propagation ainsi que les bâtiments et leurs différents paramètres pour compléter la simulation.

Pour le troisième sprint, je vais m'attaquer à la partie graphique en commençant par les graphiques et la librairie Live Charts. Et je finirais par adapter l'interface graphique déjà existante au projet en y apportant des modifications majeures.

Le dernier sprint est consacré entièrement aux finitions du projet ainsi qu'à l'optimisation et si le temps est suffisant, aux améliorations prévues dans le cahier des charges. Les deux derniers jours étant consacrés entièrement à la documentation.

Planning: Covid propagation

Jour	SPRINT 1	19.04.2021	20.04.2021	21.04.2021	22.04.2021	23.04.2021	26.04.2021	27.04.2021	28.04.2021	29.04.2021	30.04.2021
Tâches :											
Général:											
Structure du code											
Optimisation (Thread - GPU - WPF)											
Temporalité (Timer - StopWatch - Planning)											
Interface utilisateur											
Paramètres											
Simulation											
Thème											
Légende											
Responsivité											
Population:											
Résistance											
Maladies											
Moyen de transport											
Cercles sociaux (amis / famille)											
Âge											
Planning											
Symptômes											
Propagation											
Données XML											
Changements états											
Calcul pourcentage de chance											
Contamination du lieu											
Bâtiments											
Type											
Taille											
Staff											
Clients											
Hopitaux											
Graphiques											
Intégration Librairie											
Mise à jour des données											
Différents type de graphiques											
Sliders et autres paramètres visuels											
Interface graphique											
Adaptation POC au nouvel environnement											
Déplacements											
Véhicules											
Batiments											
Individus											
Responsivité											
Adaptation des données de la simulation											
Autres :											
Documentation											
Journal de bord											
Rapport											

Jour	03.05.2021	04.05.2021	05.05.2021	06.05.2021	07.05.2021	10.05.2021	11.05.2021	12.05.2021	14.05.2021	17.05.2021
Tâches :										
Général:										
Structure du code										
Optimisation (Thread - GPU - WPF)										
Temporalité (Timer - StopWatch - Planning)										
Interface utilisateur										
Paramètres										
Simulation										
Thème										
Légende										
Responsivité										
Population:										
Résistance										
Maladies										
Moyen de transport										
Cercles sociaux (amis / famille)										
Âge										
Planning										
Symptômes										
Propagation										
Données XML										
Changements états										
Calcul pourcentage de chance										
Contamination du lieu										
Bâtiments										
Type										
Taille										
Staff										
Clients										
Hopitaux										
Graphiques										
Intégration Librairie										
Mise à jour des données										
Différents type de graphiques										
Sliders et autres paramètres visuels										
Interface graphique										
Adaptation POC au nouvel environnement										
Déplacements										
Véhicules										
Batiments										
Individus										
Responsivité										
Adaptation des données de la simulation										
Autres :										
Documentation										
Journal de bord										
Rapport										

Jour	18.05.2021	19.05.2021	20.05.2021	21.05.2021	24.05.2021	25.05.2021	26.05.2021	27.05.2021	28.05.2021	31.05.2021
Tâches :										
Général:										
Structure du code										
Optimisation (Thread - GPU - WPF)										
Temporalité (Timer - StopWatch - Planning)										
Interface utilisateur										
Paramètres										
Simulation										
Thème										
Légende										
Responsivité										
Population:										
Résistance										
Maladies										
Moyen de transport										
Cercles sociaux (amis / famille)										
Âge										
Planning										
Symptômes										
Propagation										
Données XML										
Changements états										
Calcul pourcentage de chance										
Contamination du lieu										
Bâtiments										
Type										
Taille										
Staff										
Clients										
Hopitaux										
Graphiques										
Intégration Librairie										
Mise à jour des données										
Différents type de graphiques										
Sliders et autres paramètres visuels										
Interface graphique										
Adaptation POC au nouvel environnement										
Déplacements										
Véhicules										
Batiments										
Individus										
Responsivité										
Adaptation des données de la simulation										
Autres :										
Documentation										
Journal de bord										
Rapport										

Jour	01.06.2021	02.06.2021	03.06.2021	04.06.2021	07.06.2021	08.06.2021	09.06.2021	10.06.2021	11.06.2021
Tâches :									
Général:									
Structure du code									
Optimisation (Thread - GPU - WPF)									
Temporalité (Timer - StopWatch - Planning)									
Interface utilisateur									
Paramètres									
Simulation									
Thème									
Légende									
Responsivité									
Population:									
Résistance									
Maladies									
Moyen de transport									
Cercles sociaux (amis / famille)									
Âge									
Planning									
Symptômes									
Propagation									
Données XML									
Changements états									
Calcul pourcentage de chance									
Contamination du lieu									
Bâtiments									
Type									
Taille									
Staff									
Clients									
Hopitaux									
Graphiques									
Intégration Librairie									
Mise à jour des données									
Différents type de graphiques									
Sliders et autres paramètres visuels									
Interface graphique									
Adaptation POC au nouvel environnement									
Déplacements									
Véhicules									
Batiments									
Individus									
Responsivité									
Adaptation des données de la simulation									
Autres :									
Documentation									
Journal de bord									
Rapport									

Effectif

Bilan personnel

Conclusion

Table des figures

Bibliographie

19.04.2021

- Utilisés dans la compairson entre les différentes technologies de l'interface graphique
  - c-sharpcorner - Sandeep Mishra - WPF vs WinForms 1
  - wpf-tutorial - WPF vs WinForms 2
  - educba - Priya Pedamkar - WPF vs WinForms 3
  - stackoverflow - Litisqe Kumar - WPF vs WinForms 4
- Utilisés dans le programme de test WPF et Unity
  - stackoverflow - Programmer - Intégration d'Unity en WPF
  - youtube - Anousha - Communication
  - Packet NuGet sur Unity

20.04.2021

- Utilisés dans la création des pipelines
  - MSDN - Auteurs disponibles sur la page - Création des pipelines nommés
  - Forum Unity - WylieFoxxx - Modification du code msdn pour fonctionner sur Unity
- 21.04.2021
  - Utilisé dans la création de la documentation des pipelines
    - Stackoverflow - usr - Modification du code MSDN

26.04.2021

- Utilisé dans la propagation du virus

Annexes

- Projet C#
- Images
    - Diagramme de classe
    - Planning prévisionnel
    - Planning effectif
- Journal de bord

Livrables

- Documentation
- Logbook
- Programme C#
- Poster