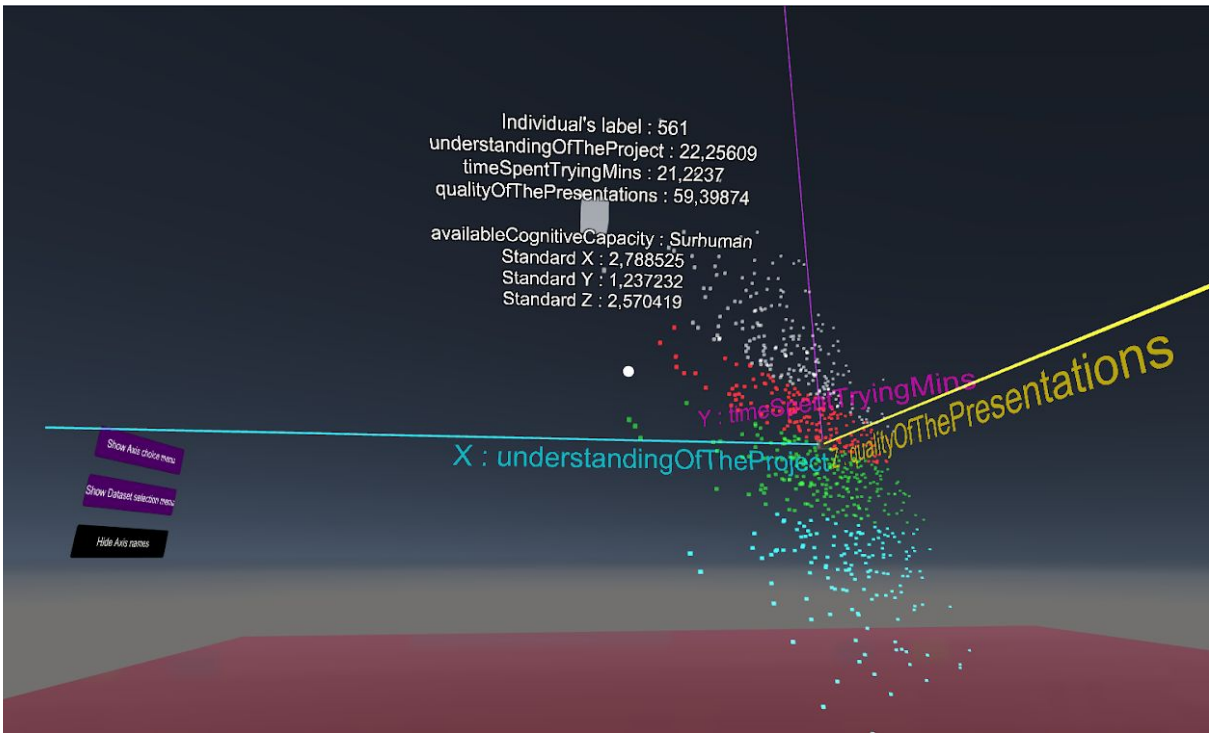
Rapport de projet informatique individuel

DataVRiz, visualisation de données en VR



Professeur tuteur : Jérôme Saracco
Professeur encadrant : Jean-Marc Salotti

Auteur et développeur : Hugo Fournier

Table des matières

I. Introduction

A. Problématique

Les données (“Big Data”) et la réalité virtuelle sont deux problématiques à la mode, et on peut en comprendre les raisons en voyant le potentiel des technologies associées. Particulièrement, les statistiques jouent un rôle prépondérant dans le flux continu d’informations que l’on peut capter sur les médias, dans les publications scientifiques, dans les décisions politiques. Développer des outils permettant de comprendre le plus facilement possible ces données, pour le plus grand nombre, représente ainsi un enjeu démocratique important. L’intuitivité de la réalité virtuelle et sa troisième dimension spatiale pourrait alors participer à la réponse.
Ce sont sur ces enjeux que je positionne mon projet d’application de visualisation de données (ou Dataviz) en réalité virtuelle (VR pour *Virtual Reality* ou Réalité Virtuelle), DataVRiz, avec deux objectifs principaux qui sont en fait deux hypothèses que j’essaie de tester :

- permettre une visualisation de données agréable et simple
- permettre de voir des structures de données en 3D qui soient invisibles en 2D

B. Contexte et technologies

Ce projet est un projet individuel d’informatique de 2ème année à l’ENSC. Il a commencé le 27 Janvier 2020 et est livré le 22 avril 2020 (date prévue initialement : 9 avril). Pour un travail prévu initialement à environ 4h/semaine avec 13 semaines de travail, il a au final représenté environ 88h de travail, soit une moyenne de 6h40 par semaine mais avec en réalité une accélération importante sur les deux dernières semaines pour raison de libération d’emploi du temps.

DataVRiz est développé avec le moteur de jeu Unity, version 2018.4.01f, avec des scripts en C#, outils que j’utilisais déjà avant le projet mais dont je souhaitais approfondir la maîtrise. L’application était initialement conçue pour fonctionner sur PC avec un casque VR Oculus Touch avec 2 manettes, matériel gracieusement mis à disposition par l’ENSC. Mais durant cette période a frappé la crise sanitaire du Covid-19, demandant une ré-adaptation du projet puisque je n’avais plus accès à ce matériel. J’ai alors commandé une visionneuse VR pour téléphone, le Google Cardboard, visible sur la photo ci-dessous. Cet outil, avec son application smartphone associée et les builds Unity, m’ont permis de tester facilement et régulièrement le projet.

Toute la transition pour pouvoir à nouveau *build* (construire une version exécutable sur la machine visée) et tester sur une dispositif VR m’a demandé environ 6h de travail. Il s’agissait d’abord d’adapter le projet à un build pour Cardboard, mais surtout à un build pour smartphone, une configuration nouvelle pour moi. Cela a aussi entraîné 2 semaines de report de planning pour la

recherche de solution et l'attente de la livraison du Cardboard, en plus changements dans la conception que j'expliquerai dans la partie suivante.

C. Application livrée

Deux builds de l'application livrée sont disponibles dans le dossier "builds", un .apk qui peut s'installer sur un smartphone Android (mais ne s'utilise bien qu'avec Google Cardboard), et l'autre pour Windows avec Oculus touch qui n'a donc pas pu être testé pour les dernières fonctionnalités mais qui peut s'exécuter depuis un ordinateur, sans avoir les contrôles de la VR pour autant.

II. Paradigme de conception

A. Spécifications

Le tableau de spécifications ci-dessous vise à répondre aux objectifs expliqués dans la problématique, avec des fonctionnalités séparées en trois catégories : traitement des données "DATA", interactions VR "CTRL" et portabilité et performance "PORT". Elles ont été conçues au lancement du projet puis mises à jour grâce aux précieux retours de mon tuteur et aux tests sur le projet en avancement. La colonne "Etat" donne un retour sur la fonctionnalité à la livraison de ce projet.

Version 1		**Etat**
F1_DATA_V1	Les données affichées sont issues d'un fichier .csv ou .txt avec un séparateur spécifique enregistrant des données à 3 dimensions quantitatives et présent dans le répertoire de l'application.	Testé V3
F2_DATA_V1	Les données sont affichées sont la forme d'un nuage de points 3D centré réduit.	Testé V3
F1_CTRL_V1	L'utilisateur peut, par une action sur le contrôleur, quitter l'application.	Testé V3
F2_CTRL_V1	Lorsque l'utilisateur tourne la tête, sa projection dans l'environnement 3D tourne en accord avec le regard.	Testé V3
F3_CTRL_V1	L'utilisateur équipé d'un casque VR peut se déplacer dans l'environnement 3D virtuel.	Testé V1 sur Occulus et V3 Cardboard
F_PORT_V1	L'application fonctionne pendant plus de 5 minutes sur le matériel VR du hall technologique de l'école pour des données avec au moins 100 individus.	Testé V1
Version 2		

F1_DATA_V2	Les données sont affichées avec des axes visibles en indiquant la signification et l'échelle.	Testé V3
F_CTRL_V2	L'utilisateur peut pointer un point et ainsi obtenir l'affichage de ses coordonnées, de son écart à la moyenne, de ses valeurs dans les unités des données.	Testé V3
F2_DATA_V2	L'application peut aussi exploiter des données à 2 dimensions.	Testé V3
F_PORT_V2	L'application fonctionne pour des données d'au moins 1000 individus.	Testé V3 (/!\ facteurs et perfs)
Version 3		
F_DATA_V3	Les données affichées peuvent représenter une 4ème dimension qualitative par une coloration distincte et non graduée des points.	Testé V3
F1_CTRL_V3	L'utilisateur peut, sans quitter l'application, lancer l'affichage de n'importe lequel fichier de données présent dans le répertoire. Les fichiers non valides ne sont pas affichés.	Testé V3
F2_CTRL_V3	En activant un mode jeu, les points deviennent soumis à une gravité et sont poussables par l'utilisateur.	Abandonnée car trop coûteux (demande de repenser le mouvement et tout)
F3_CTRL_V3	L'utilisateur peut choisir les variables à afficher et sur quels axes les placer.	Testé V3
F_PORT_V3	L'application fonctionne sur au moins un autre dispositif matériel.	Testé V3
Version 4		
F_DATA_V4	Les données utilisées peuvent contenir plus de 4 variables.	Testé V3
F1_CTRL_V4	L'utilisateur peut visualiser un fichier de données existant sur sa machine sans quitter l'application.	Non livrée
F2_CTRL_V4	L'utilisateur peut zoomer ou dézoomer sur les données.	Non livrée, très peu utile
F3_CTRL_V4	L'utilisateur peut faire tourner les 3 axes du nuage de points, s'il a mal au cou par exemple.	Non livrée, très peu utile

F_PORT_V4	L'application fonctionne sans soucis de performance pendant plus de 20 minutes.	Testé V3

A. Statistique

Le projet DataVRiz comporte deux enjeux majeurs liés à la statistique : que l'application soit exacte et cohérente avec les notions de statistique, et qu'elle puisse avoir une utilité pour la compréhension des données, que l'application facilite le plus possible un potentiel usage cohérent avec les outils statistiques pré-existants.
Dans ce paradigme là, le projet a suivi les choix de développement et de conception suivants :

- priorisation sur la compatibilité avec les fichiers de données au format .txt séparateur virgule et .csv, qui sont communs, et simples à lire
- génération de données de test avec le langage R pour pouvoir comparer les résultats avec un standard du milieu
- mettre à disposition un maximum d'informations pertinentes et précises, sans surcharger pour autant
- représenter en nuage de points, représentation qui profite effectivement de la troisième dimension
- représenter les données par un nuage de points réduits et centrés sur le centre de la scène pour pouvoir toujours voir dans le même volume l'ensemble des individus du jeu de données
- permettre de visualiser des jeux de données avec une taille d'échantillon la plus grande possible sans que cela cause de soucis de performance

B. Réalité virtuelle

Utiliser la réalité virtuelle apporte des opportunités et des contraintes, notamment :

- les contrôles et le déplacement peuvent y être bien plus intuitifs mais il y a beaucoup moins de touches disponibles
- la performance est très importante, car si le nombre d'image par secondes (ips ou *fps pour frames per second*) baisse en dessous du taux de rafraîchissement du matériel VR (qui est 90 fps pour la majorité des dispositifs), cela peut provoquer des nausées à l'utilisateur
- l'interface utilisateur est forcément à l'intérieur du monde virtuelle de l'application (les interfaces extra-diégétiques, très courantes en 2D, causent très vite de la cybersickness)
- les textes 2D, qui doivent donc être placés dans le monde, ne sont pas toujours lisibles
- un résultat pas testable directement et moins prévisible par rapport aux applications Unity classiques

De plus, les différents dispositifs de VR ont quelques spécificités. Pour les deux qui nous intéressent, l'Oculus touch et le Google Cardboard, on a notamment :

- un poids du casque léger pour le Cardboard avec téléphone, comparé au poids plus important de l'Oculus touch (qui est lui-même plus léger que plusieurs autres casques)
- deux manettes avec 4 ou 5 boutons chacune pour l'Oculus touch, contre un seul bouton pour le Cardboard, qui simule en fait un contact digital sur l'écran
- pas de capteurs de position de l'utilisateur pour le Cardboard (seulement le gyroscope du smartphone pour la rotation de la tête) contrairement à l'Oculus touch
- possiblement une carte graphique et un processeur très puissants sur l'ordinateur de l'Oculus touch, contrairement au smartphone qui est limité bien plus vite

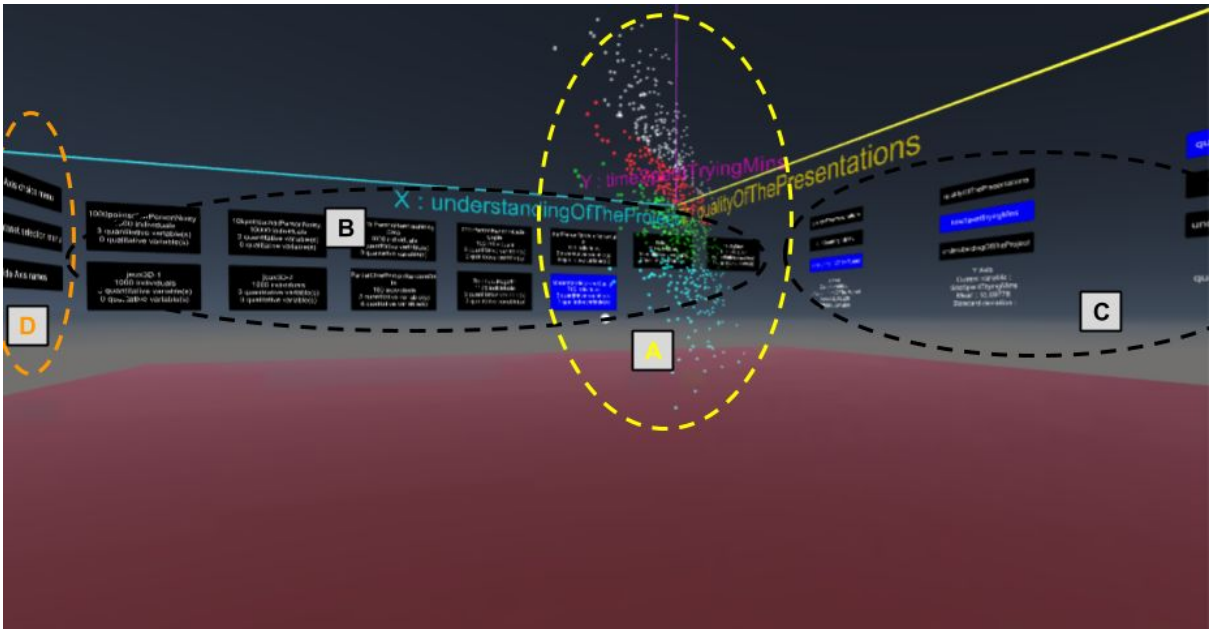
Tout ceci, qui pose des exigences importantes sur l'expérience utilisateur (UX pour *user experience*), a entraîné les choix suivants pour ce projet, avec certaines modifications à la moitié du projet suite à la perte de l'Oculus touch avec PC :

- ⇒ priorisation encore plus importante des contraintes de performance
- déplacement initialement prévu avec une combinaison de déplacement dans la zone des capteurs et des manettes ⇒ déplacement dans la direction du regard avec l'unique bouton du Cardboard
- priorisation des contrastes de couleur pour améliorer la lisibilité
- plusieurs tests par session sur le dispositif de VR
- beaucoup d'autres choix de conception plus mineurs et de réglages faits et modifiés au fur et à mesure de l'avancement du développement et des tests possibles

III. Architecture de l'application

A. Explication de la scène Unity

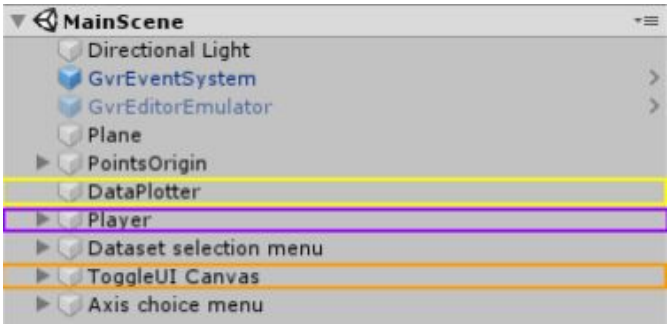
Dans un projet Unity, les éléments sont organisés en GameObjects, qui comportent plusieurs Components paramétrables (dont la Transform qui déterminent leur position dans le monde et les Scripts qui permettent de programmer leurs comportements). Ces GameObjects peuvent avoir des enfants, et donc des parents, cela permet notamment d'organiser la scène et grouper les objets. Enfin, ces GameObjects sont répartis dans des Scenes, qui sont des ensembles de GameObjects paramétrés et positionnés.
Dans DataVRiz, il n'y a qu'une scène principale, qui ressemble au screenshot ci-dessous ceci lorsqu'elle est lancée. Malheureusement, les screenshots en VR ne sont pas possibles. Ce qui est dommage, car la perception de la profondeur, l'angle de vue supérieur et la liberté de rotation permettent une vue plus lisible et compréhensible :



Invisible, devant la caméra, se trouve bien sûr l'utilisateur, représenté par l'objet "Player", qui porte la caméra pour qu'elle suive ses déplacements.
Au centre de la scène, en A, se trouvent l'origine des points (x=0, y=0 ,z=0 dans l'espace de représentation des données) qui est en hauteur pour pouvoir voir les points dont le Y centré réduit

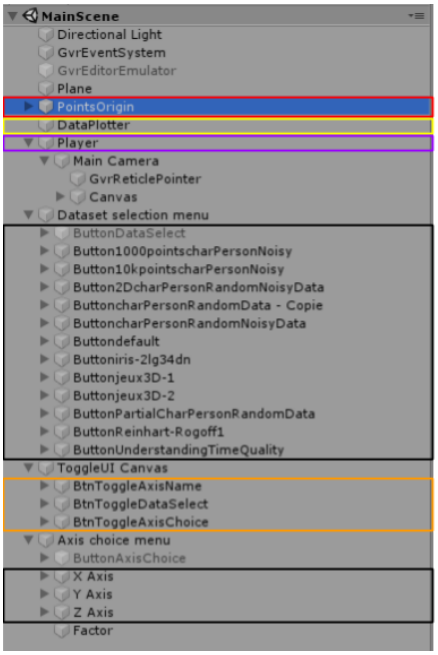
est négatif. On y voit aussi le nom des variables représentées sur chaque axe, représentés eux-mêmes par un rayon de couleur dans la direction associée, vers le sens positif. Enfin, on y voit tous les DataCubes, représentant chacun un individu du jeu de données visualisé.
En B se trouve l'ensemble des boutons qui permettent de lancer la visualisation d'un autre jeu de données. Les boutons sont générés au lancement de l'application, un pour chaque jeu de données valide détecté dans le dossier "Resources/DataFiles" . Sur chaque bouton est lisible le nom du fichier (sans son extension), le nombre de variables quantitatives et qualitatives détectées, et le nombre d'individus.
En C, un autre ensemble de boutons en 4 colonnes qui permettent de choisir la variable représentée sur chaque axe (dans l'ordre de gauche à droite : axe X, axe Y, axe Z, facteur représenté par la couleur des points). En bas de chaque colonne, un texte affiche un résumé de la variable actuellement affichée avec son nom, sa moyenne et son écart-type empirique.
En D est un menu pour cacher ou ré-afficher les deux précédents, mais aussi le nom des axes, pour pouvoir mieux voir les DataCubes et leurs détails si besoin.

Cette scène fonctionne avec l'ensemble de GameObjects formant la scène principale, dont l'architecture est visible sur le screenshot ci-dessous. Les objets encadrés d'une couleur sont ceux qui implémentent au moins un Script. Les couleurs, qui vont rester les mêmes ensuite pour l'architecture des scripts, permettent de voir l'utilité principale de ces GameObjects scriptés dans la scène visible ci-dessus.



On voit donc assez peu de GameObjects différents quand on ne déroule pas leurs enfants lorsque l'application n'est pas lancée. En jaune, l'objet "DataPlotter" est invisible mais supporte un script important qui sera expliqué par la suite.

Mais lorsque la scène est active et qu'on regarde aussi les enfants (sauf ceux de PointsOrigin qui sont aussi nombreux qu'il y a d'individus), on voit cette hiérarchie, qui contient maintenant les boutons générés de B et de C, et les DataCubes visibles en A :



B. Architecture des scripts Unity

Le diagramme ci-dessous (qui prend une page entière) représente l’ensemble des classes C# que j’ai implémenté dans le projet et leurs relations. Les classes implémentées sur des GameObjects sont encadrées avec les mêmes couleurs correspondantes. Les flèches en vert représentent des associations qui ne se font pas directement par le code mais en récupérant le script sur le GameObject associé. Les classes qui ne sont pas encadrées ne sont pas implémentées sur des objets et n’utilisent pas les méthodes de l’API Unity.
Sans trop avancer les détails qui vont suivre dans les parties suivantes, ces classes peuvent se séparer en trois groupes :

i. Traitement statistique

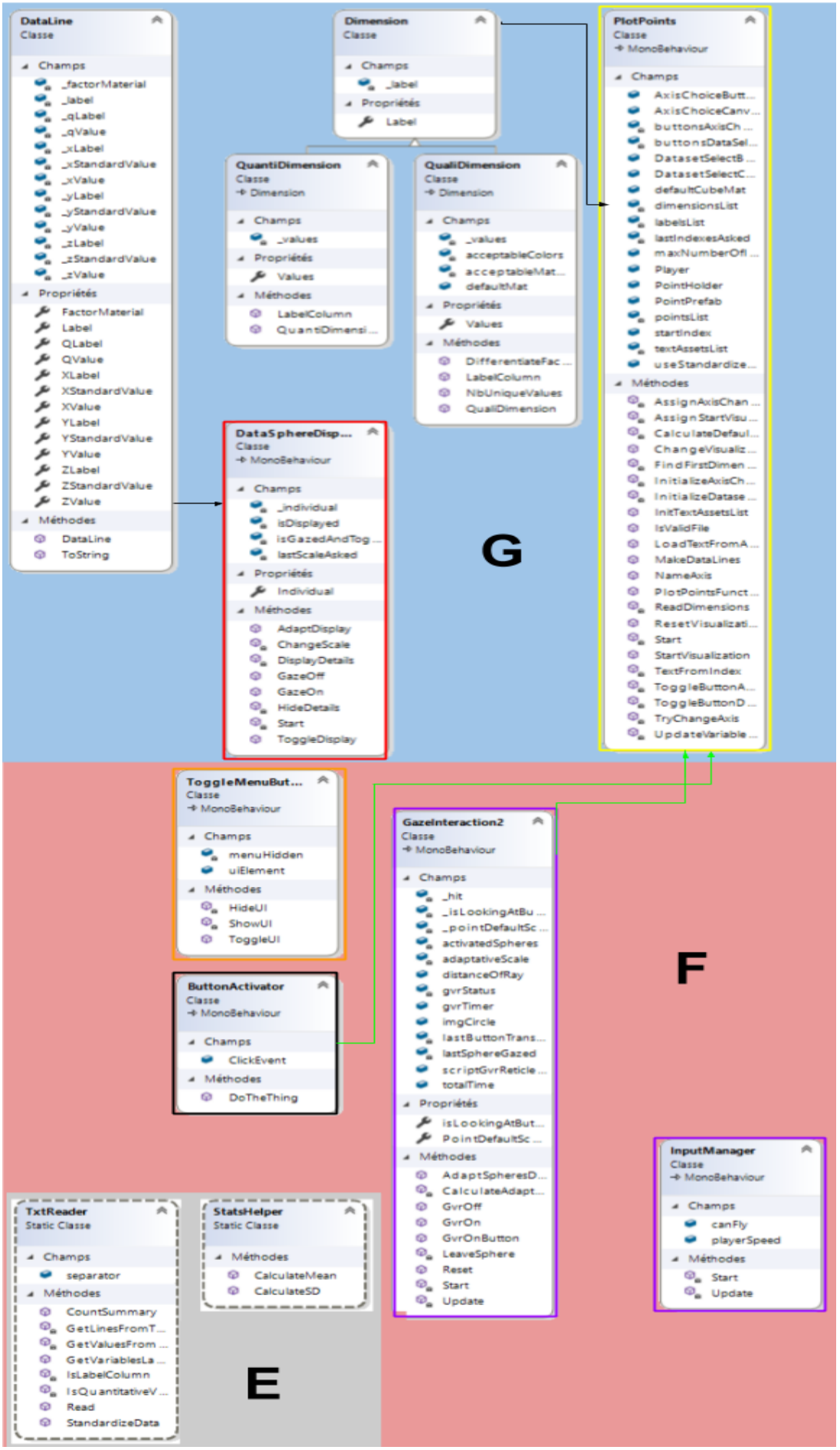
En zone E, les deux classes principales qui servent au traitement des données et traitement statistique. Elles sont toutes deux statiques et sont en fait utilisées comme une librairie de fonctions, notamment par la classe PlotPoints (même s’il n’y a pas de flèche).

ii. Interactions VR

En zone F, ce sont les classes qui implémentent les interactions VR et la gestion des évènements associés, avec les boutons et le pointeur de regard notamment.

iii. Classes intermédiaires

En zone G, ce sont les classes intermédiaires qui servent notamment en tant que support d’informations et/ou ou de création d’objets. On y trouve notamment la massive classe PlotPoints qui centralise beaucoup de traitement et d’informations des autres classes.


DataLine
Classe
Champs
_factorMaterial
_label
_qLabel
_qValue
_xLabel
_xStandardValue
_xValue
_yLabel
_yStandardValue
_yValue
_zLabel
_zStandardValue
_zValue
Propriétés
FactorMaterial
Label
QLabel
QValue
XLabel
XStandardValue
XValue
YLabel
YStandardValue
YValue
ZLabel
ZStandardValue
ZValue
Méthodes
DataLine
ToString
Dimension
Classe
Champs
_label
Propriétés
Label
QuantiDimension
Classe
Dimension
Champs
_values
Propriétés
Values
Méthodes
LabelColumn
QuantiDimensi...
QualiDimension
Classe
Dimension
Champs
_values
acceptableColors
acceptableMat...
defaultMat
Propriétés
Values
Méthodes
DifferentiateFac...
LabelColumn
NbUniqueValues
QualiDimension
PlotPoints
Classe
MonoBehaviour
Champs
AxisChoiceButt...
AxisChoiceCanv...
buttonsAxisCh...
buttonsDataSel...
DatasetSelectB...
DatasetSelectC...
defaultCubeMat
dimensionsList
labelsList
lastIndexesAsked
maxNumberOfl...
Player
PointHolder
PointPrefab
pointsList
startIndex
textAssetsList
useStandardize...
Méthodes
AssignAxisChan...
AssignStartVisu...
CalculateDefaul...
ChangeVisualiz...
FindFirstDimen...
InitializeAxisCh...
InitializeDatase...
InitTextAssetsList
IsValidFile
LoadTextFromA...
MakeDataLines
NameAxis
PlotPointsFunct...
ReadDimensions
ResetVisualizati...
Start
StartVisualization
TextFromIndex
ToggleButtonA...
ToggleButtonD...
TryChangeAxis
UpdateVariable...
DataSphereDisp...
Classe
MonoBehaviour
Champs
_individual
isDisplayed
isGazedAndTog...
lastScaleAsked
Propriétés
Individual
Méthodes
AdaptDisplay
ChangeScale
DisplayDetails
GazeOff
GazeOn
HideDetails
Start
ToggleDisplay
G
ToggleMenuBut...
Classe
MonoBehaviour
Champs
menuHidden
uiElement
Méthodes
HideUI
ShowUI
ToggleUI
GazeInteraction2
Classe
MonoBehaviour
Champs
_hit
_isLookingAtBu...
_pointDefaultSc...
activatedSpheres
adaptativeScale
distanceOfRay
gvrStatus
gvrTimer
imgCircle
lastButtonTrans...
lastSphereGazed
scriptGvrReticle...
totalTime
Propriétés
isLookingAtBut...
PointDefaultSc...
Méthodes
AdaptSpheresD...
CalculateAdapt...
GvrOff
GvrOn
GvrOnButton
LeaveSphere
Reset
Start
Update
F
ButtonActivator
Classe
MonoBehaviour
Champs
ClickEvent
Méthodes
DoTheThing
InputManager
Classe
MonoBehaviour
Champs
canFly
playerSpeed
Méthodes
Start
Update
TxtReader
Static Classe
Champs
separator
Méthodes
CountSummary
GetLinesFromT...
GetValuesFrom...
GetVariablesLa...
IsLabelColumn
IsQuantitativeV...
Read
StandardizeData
StatsHelper
Static Classe
Méthodes
CalculateMean
CalculateSD
E

IV. Exploiter les données

A. Statistique

Dans la classe StatHelper (**zone E** du diagramme ci-dessus) sont simplement implémentés un calcul de moyenne et un calcul d'écart-type empirique, qui est un estimateur biaisé, ce qui représente peut-être un mauvais choix, même si la différence n'est pas flagrante pour un échantillon de grande taille.
Cette classe est notamment utilisée par la méthode StandardizeData() du TxtReader qui permet de calculer pour les DataLines (représentées par les DataCubes) leurs coordonnées en centré-réduit.

Tous les jeux de données utilisés pour le test et utilisables avec la version livrée de l'application sont disponibles au chemin suivant, en partant de la racine du dossier livré :
"UnityProject-DataVRiz\Assets\Resources\DataFiles".

B. Analyser les fichiers

Cette partie là de l'implémentation est sûrement celle qui a demandé le plus de refactorisation. Elle commence dans PlotPoints qui, à son initialisation, scanne tous les fichiers présents dans le répertoire nommé ci-dessus avec InitTextAssetsList(). Ils sont ajoutés à la liste si ils sont comptés comme valides par IsValidFile(), ce qui veut dire que le TxtReader.CountSummary() a détecté au moins 2 variables quantitatives, que le fichier peut être un TextAsset, et qu'il a une taille d'échantillon n inférieur à un seuil fixé à 10000 (pour des soucis de performance).
Ensuite, il faut lancer la visualisation avec StartVisualization(int index) qui va enchaîner d'autres méthodes de PlotPoints() pour créer le nuage de points et les axes correspondants à l'index demandé dans la liste des jeux de données scannés. Notamment, cela demande d'utiliser le TxtReader.Read() qui lit le texte en entier du fichier puis crée une liste d'objets Dimension, correspondant chacun à une variable, quantitative ou qualitative. Au lancement de l'application, l'index 0 est donné par défaut, et chaque jeu de données est directement lancé sur ses premières variables compatibles grâce à la méthode FindFirstDimensions(int index).
En résultat, la version livrée peut afficher sans soucis sur un smartphone Galaxy S7 (performances actuellement milieu de gamme) un jeu de données de 1000 points sans soucis, ou 1000 points avec un facteur (donc des couleurs différentes qui coûtent beaucoup à traiter) avec des légers ralentissements. Le jeu de données sur les iris à 150 points avec facteur ne pose par contre aucun soucis à afficher. Les fichiers compatibles sont les fichiers en .txt avec séparateur virgule, ou les .csv, avec moins de 10 000 individus et même avec des données vides ou incorrectes car les individus seront simplement écartés du jeu de données.

C. Les informations au bon endroit

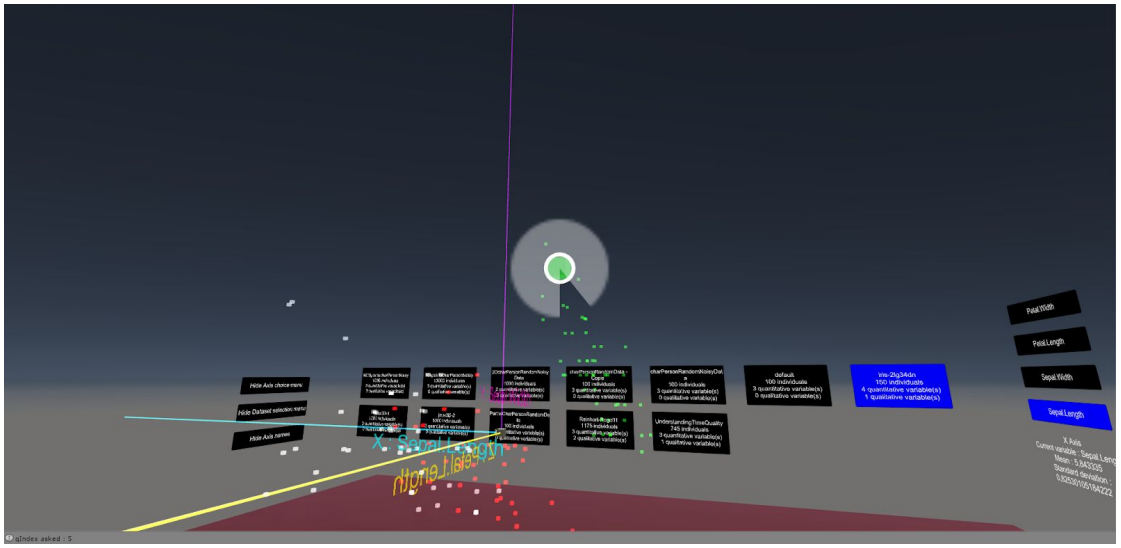
Un dernier enjeu important était celui de pouvoir récupérer et afficher les bonnes informations, au bon endroit, sans traitement superflu.
Cela est permis notamment par la classe intermédiaire PlotPoints et les deux classes DataLine et Dimension (**zone G**). Une DataLine stocke toutes les informations d'un point, aussi bien celles nécessaires à son affichage dans le nuage qu'à l'obtention de ses détails. En somme, c'est un individu du jeu de données qui est affiché selon maximum 3 variables quantitatives et une quantitative. Pour faciliter et optimiser le traitement, les individus ne sont pas traités en entier, ces DataLines sont nettoyées et recrées quand on change les variables affichées.

La classe Dimension, apparue plus tard dans le développement, stocke toutes les valeurs d'une variable. C'est soit une QuantiDimension, soit une QualiDimension, selon si elle doit représenter des valeurs numériques ou un facteur. Elle permet d'abord de pouvoir changer les variables affichées d'un même jeu de données, mais aussi de pouvoir associer des couleurs aux différents niveaux d'un facteur pour pouvoir les associer aux DataCubes.

V. Adaptation à la VR

A. Pointeur lié au regard

Pour adapter l'application à un casque plus léger mais avec un seul bouton, la plupart des interactions ont été implémentées avec le regard. Comme l'illustre le screenshot ci-dessous, un point blanc indique ou est centré le regard, et lorsqu'il touche un objet interactible (Bouton ou DataCube), ce point s'écarte pour former un cercle, et se déclenche alors un rapide timer représenté par ce disque blanc qui se remplit progressivement. Également, les DataCubes grossissent temporairement lorsqu'ils sont regardés, pour permettre de maintenir le regard malgré des tremblements de la main tenant le Cardboard. Ils restent grossis lorsqu'ils sont activés pour permettre de les désactiver plus facilement et de visualiser lesquels sont activés.



Ceci est en partie implémenté grâce au GvrPointer, fourni par le package Google VR proposé pour fonctionner avec le Google Cardboard. Mais pour s'adapter à ces interactions spécifiques, la majorité du comportement de ce pointeur est codée dans la classe GazeInteraction2 (zone F du diagramme). Cette classe permet d'activer et désactiver le timer en fonction des objets regardés, de déclencher les événements associés aux objets lorsque le timer est complété, et de grossir/dégrossir correctement les DataCubes regardés ou quittés du regard.

B. DataCubes et boutons

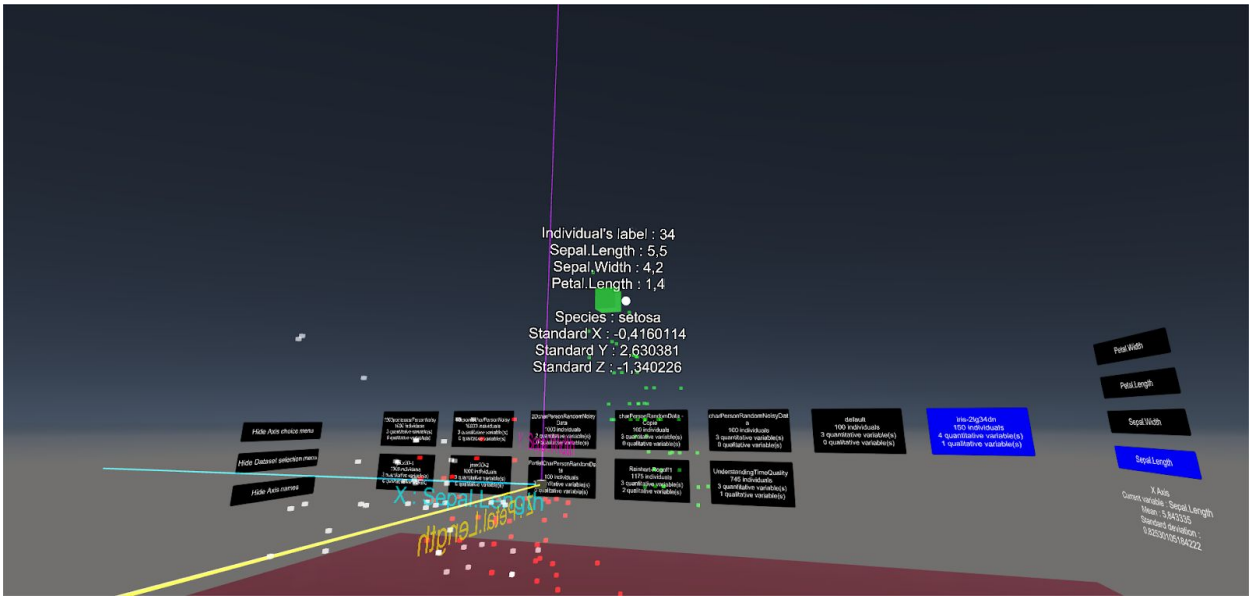
Les DataCubes peuvent afficher leurs détails grâce à la classe DataSphereDisplayer visible en zone G (renommer un script est risqué dans Unity, que reposent en paix les DataSpheres). Elle utilise les données de son unique DataLine associée pour activer/désactiver et adapter l'affichage de ses détails selon les appels de GazeInteraction2.
Les boutons sont divisés en 3 types, comme vu dans la partie II.A. : ceux pour cacher/afficher, pour changer dataset, ou changer de variable sur un axe. Les premiers implémentent le script simple ToggleMenuButton (zone F) alors que les deux autres dépendent plus de méthodes de PlotPoints.

Mais les trois implémentent ButtonActivator, un script très simple qui permet de stocker un évènement et a une méthode pour le déclencher.
Ceci est utilisé dans PlotPoints pour pouvoir assigner à ces boutons des délégués (sorte d'encapsulation de méthodes) en tant que *listener*, prêts à être déclenchés quand le bouton est activé, avec des paramètres précis correspondant à chaque bouton (par exemple l'axe correspondant pour tel bouton, ou l'index du dataset qui doit être affiché). Ceci est nécessaire car sinon le UnityEvent du ButtonActivator ne peut pas prendre d'actions avec des paramètres. Ces paramètres sont assignés lors de la création des boutons dans PlotPoints, dans les méthodes InitAxisChoiceButton() et InitDataselectButton(), ou à la main dans l'éditeur pour les trois boutons d'affichage qui ne sont pas générés automatiquement.

C. Mouvement et lisibilité

Dans DataVRiz, le déplacement se fait par une pression (continue ou non) du bouton du Cardboard. Très simplement, l'utilisateur avance dans la direction de son regard tant qu'il appuie. Ce déplacement est simple mais efficace

En laissant le regard sur un cube, il affiche donc ses détails, de façon zoomée tant qu'on ne le quitte pas des yeux. Et pour permettre de facilement se rapprocher et s'éloigner, de tourner autour du nuage pour prendre différents points de vues, les détails affichés par un DataCube, visibles dans le screenshot ci-dessous, sont toujours lisibles.



En effet, après chaque déplacement (= relâchement du bouton, donc une simple pression suffit) , et à chaque nouveau détail affiché, l'échelle du texte s'adapte à la nouvelle position de l'utilisateur et il se tourne dans la bonne direction. Ceci est permis par plusieurs interactions entre DataSphereDisplayer et GazeInteraction2, selon une formule établie empiriquement dans GazeInteraction2.CalculateAdaptativeScale().

Enfin, la taille des points est aussi adaptée pour faciliter la lisibilité, mais selon la taille de l'échantillon. Ainsi, les cubes d'un échantillon de taille n=1000 vont avoir une arête deux fois plus petite que les cubes pour n=100, pour permettre de naviguer même dans un grand jeu de données. Ceci est implémenté grâce à une formule établie empiriquement et codée dans PlotPoints.CalculateDefaultScale().

D. Optimisation

Après tests de plusieurs réglages grâce à l'onglet "Stats" de l'éditeur Unity, dont notamment les réglages lumières, ombres, réflexions et matériaux des nombreux DataCubes, en plus de la conception initiale minimaliste, les modifications suivantes, nécessaires au vu de la triple contrainte de performance, ont permis de significativement baissé la consommation de l'application sur le processeur et la carte graphique :

- activation du GPU instancing sur le matériau des DataCubes et assignation du matériau depuis un script pour permettre au processeur de regrouper les cubes en "batches" de même couleur et ainsi fluidifier le traitement
- désactivation des réflexions sur les cubes et des ombres par la lumière principale
- abandon des DataSpheres pour les DataCubes, bien qu'elles étaient plus esthétiques

La dernière modification a divisé le nombre de sommets de polygone *(vertices*) et de triangles traités par 50, passant d'environ 500 000 pour un jeu de données de 1000 individus, à environ 10 000. Tout ceci, d'après les tests sur mon ordinateur (processeur I5 3.30 GHz), a divisé par 2 la consommation des ressources du processeur. Sur smartphone, cela a permis de rendre fluide l'affichage d'un jeu de données de 1000 points. La mémoire RAM n'a jamais posé problème, dépassant rarement les 600 Mo.

VI. Gestion du projet

A. Risques

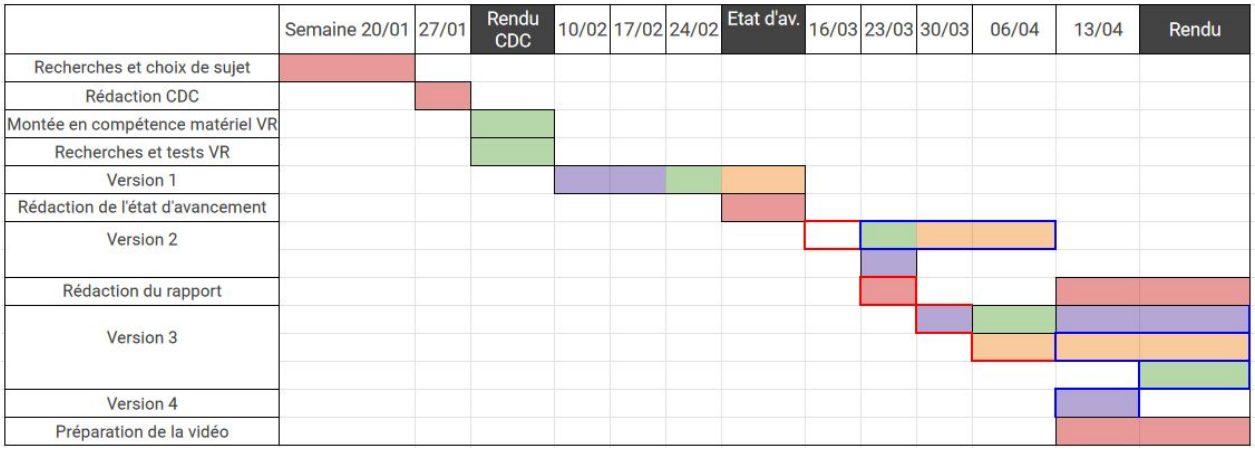
Dès le choix du sujet du projet, j'avais identifié les risques suivants :

1. soucis de matériel VR
2. le développement spécifique à la VR qui explose en temps de développement
3. des soucis de performance bloquants

Pour mitiger leur impact, il s'agissait surtout de faire des recherches préliminaires pour ensuite contourner au mieux les problèmes, tout en ayant un plan B si le risque s'avère bloquant. Pour le matériel VR, ca a été par exemple l'achat d'un Google Cardboard, auquel j'avais déjà pensé très tôt dans le prochain. Pour le risques 2 et 3, qui se sont avérés mais plus légèrement, il s'agissait surtout de prendre de l'avance et d'anticiper dès la conception. En rétrospective, ils n'ont jamais été bloquants.

B. Planning

Ci-dessous se trouve le planning de travail que j'ai suivi, avec adaptation à l'extension des deadlines. Les cellules encadrées en rouge représentent du travail que j'avais initialement prévu de faire, mais que je n'ai pas effectué sur cette semaine. Les cellules en bleues en sont souvent la conséquence puisqu'elles représentent du travail effectué qui n'était pas prévu sur cette semaine là.
On voit notamment l'explosion du temps du développement sur les interactions en VR avec d'abord l'affichage détaillé d'un point de la version 2 puis les boutons de la version 3. Ceci a fait prendre du retard à la version 2, rattrapé par l'accélération à la fin. On voit aussi les travaux en portabilité et performance ajoutés par le changement de matériel.



C. Carnet de bord, état des fonctionnalités et commits

Trois démarches m'ont été très utile dans la gestion de ce projet.
D'abord celle de tenir un carnet de bord, sur lequel j'ai écrit brièvement à chaque avancée ce que je viens d'implémenter ou concevoir, et ce sur quoi je vais travailler ensuite. Il m'a permis notamment de reprendre rapidement à chaque début d'une nouvelle session de travail, en cohérence avec mon travail de la session précédente. Cela m'a permis aussi de revenir sur telle architecture pour comprendre pourquoi je l'avais conçue ainsi. Et enfin, c'est avec ça que j'ai pu compter mes heures de travail et préciser mes écarts au planning.
Ensuite celle de tenir à jour un répertoire git avec un ou deux commits par session, pour pouvoir annuler des changements mais surtout voir les changements à une date donnée (en s'aidant du carnet de bord notamment), et avoir une sauvegarde.
Enfin, je mettais à jour l'état des fonctionnalités sur mon cahier des charges et cela était très utile pour continuer à avancer, voir la suite, et visualiser l'avancement du projet.

D. Tests et débogage

Pour ce projet, il a été important de tester très souvent les fonctionnalités, pour s'assurer que ça fonctionne, mais aussi que ça ait l'air utilisable, car les contraintes UX étaient fortes. Souvent, c'était au moins un build VR par session en fin ou au milieu pour tester les nouveaux ajouts.
Et beaucoup plus régulièrement les tests se faisaient directement sur la Game view de l'éditeur Unity avec un émulateur de contrôle.
Pour ne pas avoir de mauvaises surprises, les tests étaient toujours menés dans l'optique de pousser vers des cas limites.
Pour le débogage, j'étais un peu moins bien armé que sur un projet C# classique avec les outils de Visual Studio qui ne fonctionnent pas tous pour un projet Unity, mais le UnityEngine.Debug.Log(message) a énormément servi pour voir les valeurs des variables et quels traitements étaient effectués ou non.

Bilan

A. Rétrospective

Commençons cette rétrospective par les leçons que je retiendrais de ce projet. Globalement, sur plusieurs points, je n'ai pas vu assez loin et assez clair dans ma conception et mon architecture, ce qui a ensuite entraîné des refactorisations coûteuses en temps de développement. Le double enjeu statistique et réalité virtuelle, deux problématiques assez nouvelles pour moi, a accentué ce problème. Pour donner quelques exemples :

- j'ai fait dès le départ l'hypothèse que les jeux de données en .csv ou .txt séparateur virgule était toujours générés avec une colonne d'étiquettes pour chaque individu; cela a entraîné des modifications importantes dans le TxtReader quand je m'en suis rendu compte très tard grâce aux jeux de données générés par mon tuteur
- PlotPoins et TxtReader étaient mal construits pour pouvoir changer les variables affichées, là aussi il a fallu une refactorisation importante et tardive
- dans la méthode d'importation des jeux de données, j'ai d'abord utilisé le classique System.IO qui permet de manipuler des fichiers en C#, puis refactorisé pour utiliser les fonctions Unity et leur objet TextAsset pour stocker les données car les fichiers ne se retrouvaient pas dans le build de l'application. Mais avec ces TextAsset, il n'est pas possible de vérifier les extensions de fichiers, et c'est très compliqué d'aller récupérer un fichier qui n'a pas été build dans le projet. La solution optimale que j'ai vu trop tard dans le projet est expliquée dans la dernière partie.

Globalement, même si les algorithmes étaient plutôt bien prévus, faire des interactions viables en VR s'est avéré bien plus coûteux en développement que prévu, ce qui a demandé d'accélérer le travail sur la fin pour amener DataVRiz à l'état de finition que je souhaitais.
Sur ce projet qui me plaisait, et sur cet environnement ou le test est graphique, immédiat, plutôt stimulant, et où on peut voir beaucoup de possibilités, je me suis souvent dispersé en travaillant sur plusieurs fonctionnalités en parallèle. C'est un point qui mérite d'y faire attention car il diminue beaucoup mon efficacité, et je pense qu'il explique en partie ces défauts de réflexion long terme. Ici je noterais que tenir un carnet de bord me permettant de voir rapidement ce sur quoi je travaille et travaillais, et avoir un cahier des charges avec des fonctionnalités priorisées m'a beaucoup aidé à ne pas trop me disperser.
Plus personnellement, c'était un projet stimulant, mais travailler seul implique très peu de deadlines et de responsabilités vis-à-vis d'autres personnes, ce qui explique en partie quelques ventres mous de motivation et donc de travail durant le projet; surtout au moment où je devais chercher une solution à la perte soudaine de l'Oculus touch. S'ajoute à ça l'impression d'un projet énorme où il reste toujours beaucoup à faire. Mais là aussi, voir le tableau des fonctionnalités se remplir progressivement, et pouvoir tester cela facilement, a été très gratifiant.

Enfin, je suis très satisfait de la version livrée de ce projet, qui est même sur certains points meilleure que ce que j'imaginais au départ. Pour autant, DataVRiz ne permet que des nuages de points parmi toutes les représentations possibles, et ne permet pas d'importer des nouveaux jeux de données sans devoir faire un nouveau build par Unity. Puis reste le doute de l'utilité que peut vraiment avoir cette application sur lequel je ne peux conclure sans une démarche que je vais expliquer dans la partie suivante.

B. Perspectives

Je vois beaucoup de possibilités intéressantes pour l'application DataVRiz, notamment au niveau du côté esthétique de l'application et de l'optimisation mémoire, mais voici ce qui, je pense, apporterait le plus au projet.
D'abord, un vrai système de traitement des fichiers et de récupération des datasets. C'est primordial pour un vrai usage de pouvoir importer facilement des nouveaux jeux de données, dans des formats plus variés, et avec des vérifications qui permettent de savoir facilement ce qui marche ou pas, et pourquoi. Ceci passerait très probablement par l'utilisation des StreamingAssets de Unity, des fichiers qui se retrouvent dans le build de l'application sans être transformés en formats Unity. Cependant, sur smartphone uniquement, ce dossier est en fait compressé et l'exploiter dans les scripts Unity demanderait d'effectuer des requêtes web.
Ensuite se pose la question de la pertinence de la VR pour la Dataviz. Lire des graphiques en 2D n'est pas toujour simple, alors en 3D on pourrait tout à fait risquer la surcharge cognitive. Là-dessus, DataVRiz demande pas mal de lecture, et en VR c'est particulièrement fatigant. Pour répondre à ces questions là, le projet bénéficierait beaucoup d'une recherche approfondie en sciences cognitives et recherche utilisateur.
Enfin, toujours dans un soucis de pertinence, se pose une problématique connexe, celle des usages. Est-ce que les structures révélées par la 3D ne peuvent pas mieux se voir par la combinaison d'autres outils en 2D ? Comment des personnes qui commencent l'analyse d'un jeu de données se serviraient de DataVRiz ? Une étude des usages avec des statisticiens, des analystes, mettrait en valeur des nouvelles fonctionnalités intéressantes et apporterait des réponses pertinentes à ces questions et pourrait permettre de conclure quant à l'utilité d'un tel outil.