Gabriel :

Difficulté n°1 : Afficher le graphe dans l'interface graphique.

Grâce à matplotlib, j'ai vu qu'on pouvait exporter un graphe sous format .png, donc j'ai fai tout un système pour exporter le graphe en image, puis grâce au module python Pillow, j'ouvrais cette image que j'afficher grâce à Custom Tkinter. Ce sytème posait problème pour plusieurs raisons :
- peu efficace et rapide : l'étape d'exportation en image puis son ouverture demandait pas mal de temps ;
- peu fiable car pour l'ouverture de l'image, il était nécessaire d'utiliser son chemin d'accès, ce qui était une source de bug supplémentaire car la disposition des fichiers changeaient d'un ordinateur à un autre ;
- nécessitait une installation supplémentaire pour le module Pillow qui n'est pas à natif à python.

La solution à ce problème a été d'utiliser l'objet de matplotlib : les Figures. Cet objet peut être défini comme un canvas sur lequel on dessine les graphiques dont on a besoin. L'avantage de ce sytème est que comme matplotlib.Figure est un objet python : on peut le renvoyer grâce à une fonction, et il peut être affiché par Custom Tkinter sans module tierce. L'affichage du graphique se fait maintenant plus rapidement et plus sécuritairement car il n'y a plus d'utilisation de chemin d'accès à des fichiers et d'exportation en image.

Difficulté n°2 : Requête pour connaître les rangs (classements) d'un prénom au fil des années

Pour étudier et donner une image de la popularité/proportion de personne née avec tel prénom, nous avons eu l'idée de faire un graphe qui affiche les classements d'un prénom donné au fil des années. Pour ce faire, j'ai premièrement utilisé en SQL le mot clé **RANK() OVER** qui permet, qu'une fois que les données soit triées selon le nombre de naissances, de récupérer le rang du prénom voulu. Cette première requête ne fonctionnait que sur une année, donc à l'aide d'une boucle **for** j'exécutais cette requête pour chaque année possible, soit 123 requêtes au total pour un graphe. Le problème était l'efficacité de la création du graphe qui prenait 5 secondes par prénom pour s'afficher. Pour résoudre le problème, j'ai d'abord fait un dictionnaire contenant les informations nécessaires à la création du graphe pour les prénoms déjà sélectionnés pour éviter de récuperer une nouvelle fois des données déjà obtenue. Je me suis en suite renseigné sur la documentation SQL, et j'ai découvert le mot le clé **PARTITION BY** qui permet de partition le résultat d'une requête, et d'effectuer le même tri de donnée sur chacune des partition. J'ai donc partitionné selon les années, puis récupérer le rang du prénom sur chaque année. Cela à permit de rendre la création du graphe plus rapide (plus qu'une seconde par prénom).

Difficulté n°3 : Mauvais affichage du graphe selon les prénoms choisis

Lorsque l'on choisissait des prénoms avec des parties en commum (ex : Jean-Michel et Michel, Jean et Jean-Gabriel), le graphique devenait aléatoire et incompréhensible avec beaucoups de lignes à l'horizontal car les abscisses étaient désorganisée passant de 2009 à 1904. Le problème a été de comprendre pouquoi il y avait ce désordre dans les abscisses. Pour résoudre le problème, je me suis rendu compte que les années (qui servent d'abscisses aux graphes) sont des chaînes de caractères, et donc leur ordre qui est pris en charge par matplotlib était mauvais. Donc au préalable, toutes les années sont converties entier dans une liste triée, puis on récupère dans le dictionnaire les naissances associées à chaque anné.

Points positif :
Projet très complet qui m'a permis de faire des choses qui m'intèressent beacoup (graphes et traîtement de données)
Prise de plaisir de rendre une base de données interactive car on peut partager ce projet avec des personnes qui ne font pas de la programmation

Points négatif :
On avance un peu à l'aveugle sur certains points car le barème et les exigences ne sont pas explicitement et rigoureuresement décrits
Rendre se projet au milieu de la deuxième semain des vacances fait que je n'ai ni pu profiter de mes vacances la première semaine, ni pu profiter de participer au projet comme j'aurais pu le faire

Yavor :

Travail fait :
-Tout le traitement et importation des données –que ce soit csv, txt ou zip (partie utilitaires).
-Ceux-ci utilisés par la suite pour mettre en place un système d'auto-installation/installation du programme à partir du petit fichier de base.
-Mise en place du config.ini et son utilisation pour offrir un service d'initialisation encore plus dynamique, performant et customizable
-Gestion des chemins absolus pour permettre l'utilisation de pyinstaller et donc offrir la possibilité de télécharger le programme en exe
-Gestion des actions/workflows GitHub pour justement générer ces exe/ « artifacts ».
-La gui d'intialisation qui propose plusieures options et s'occupe de lancer la partie GUI
-Résolution de petits problèmes diverses au sein de leur GUI
-Un peu de threading
-Le README.md
i.e. Plus ou moins toutes les fonctionnalités pas directement visibles à l'utilisateur

Difficulté n°1 : Importation des bases de données « decès »

La première des sous-difficultés de celle-ci fût le fait que ces données bien que disponibles sur data.gouv.fr étaient fournies au format txt et non pas csv. De ce fait, elles n'étaient pas simplement importables en utilisant un reader qui départage les valeurs selon des séparateurs fixés mais simplement par un nombre variable d'espaces séparant les données.
Pour la résoudre j'ai donc développé un pattern **Regex** très complèt pouvant être appliqués à toutes les données de decès, qu'importe le nombre ou l'année, et de les séparer correctement pour par la suite pouvoir les utiliser et les importer dans la Base De Données.

La deuxième sous-difficulté concerne les txt et csv. En effet, alors que je croyais avoir réussi à tout importer correctement avec ce pattern **Regex**, je fût informé qu'il était impératoire de passer par un CSV lors de l'import vers la base de données. En conséquence, j'ai dû premièrement extraire les données des fichiers txt **puis** les écrire dans des fichiers csv **puis** les importer dans la base de données, résolution qui, comme vous le verrez a affecté la suite.

Difficulté n°2 : Trop grand éspace que prennaient les données

A cette étape du développement, nous utilision déjà plusieurs datasets diverses en relation avec les prénoms sur plusieures années. La conséquence : le projet feaisait à présent plusieurs Go et non seulement son utilisation sur les PC du Lycée était difficile en raison des restrictions, il était clairement impossible de le partager via e-mail, ajoutez à cela l'étape intérmediaire de txt vers csv on dépassait largement l'espace de stockage.

La solution était donc de créer un système d'initialisation de la base de données executable à partir d'un plus petit script python ou directement depuis l'exe. Ce système propose 3 options principales : Charger une base de données existante (au cas où on en a généré une auparavant mais a changé de chemin), en générer une au chemin par défaut, choisir un chemin pour la générer. Si on prend l'option 2 ou 3 alors le programme recupère les fichiers dont nous avons besoin sur le web (configuration des liens disponible dans config.ini), il importe ensuite les fichiers et leurs données un par un. Selon cas, soit on télécharge le zip pour les naissances, on unzip, importe les données dans la base de données et on renettoie par la suite, soit on télécharge les txt, les converti en csv, les importe et renettoie. Ce nettoyage permet de fair en sorte que la base de données ne prenne qu'environ 1/3 ou 1/4 de ce qu'elle prend normalement et d'être totalement fonctionnelle peu importe l'environnement (e.g. les pc du lycée disposant de stockage insuffisant).

Difficulté n°3 : L'executabilité du programme

En tant que programme python, son execution nécessite une certaine version avec laquelle il a été conçu pour également fonctionner son problème, de librairies, de marcher sur tous les OS (ou du moins Windows et le notre (Linux)). Etant donnée que nous avions nous-même quelques problèmes quant au téléchargement de certaines librairies/modules et soucis d'incompatiblité entre système, il fallait trouver une solution. Il y avait de plus quelques problèmes avec le fait que notre programme se feaisait « flag » comme Trojan 100 % sur Windows.

Pour le problème des systèmes d'exploitation c'était assez simple, il a simplement suffit de faire des « checks » à quelques endroits du code pour s'assurer d'executer le bon morceau de code selon système (« checks » avec le module os).

Pour s'assurer qu'il y ait tout de même au moins une version du programme normalement executable sur Windows sans contraintes de librairies ou de version python (parfois problématiques à télécharger) il a fallu compiler le programme en .exe. Pour ce faire j'ai utilisé le module Pyinstaller qui a lui-même quelques problèmes avec les anti-virus et les modules :

- Manquait des données (comme les icônes utilisées ou autre ressources nécessaires au programme). Résolu avec des –add-data en paramètres.
- Le module CustomTkinter dont nous nous sommes servis est un peu spécial dans le sens où il n'est pas automatiquement inclus dans l'exe. La solution fût de trouver son chemin et « forcer » pyinstaller à l'inclure dans son entiereté et ses propres dépendances
- L'exe généré se feaisait « flag » comme Trojan par l'antivirus windows (encore plus que le programme python de base). La solution pour tous les problèmes fût quelques changements à de diverses endroits du code et surtout l'utilisation de quelques paramètres compileur comme –noupx –clean –onedir… jusqu'a obtenir un exe totalement « sûr »

- Aucun de nous disposait vraiment de système Windows alors on ne pouvait pas générer l'exe nous-mêmes. Solution : J'ai configuré github actions et workflows pour nous générer un zip contenant l'exe grâce à pyinstaller, qui se fait directement upload sur le github.

Points Positifs :
- Projet m'a permis de me remémorer des outils et compétences précedement employés en train de tomber dans l'oubli comme par exemple : CustomTkinter, le Threading, le regex, pyinstaller, cross-platform, actions github…
- Renforce les compétences python en général comme la modularization, programmation fonctionelle
- Première fois que je fais un système d'auto-initialisation (et il marche correctement).
- Très satisfeaisant de venir à bout d'un problème

Points négatifs :
- Il y avait trop de problèmes, très diverses et qui prennent trop de temps (j'en ai cité 3 mais ce ne sont que 3 parmi un bon nombre)
- Temps de développement assez long
- Problèmes inattendus : le fait que ce soit obligatoire de reconvertir les txt en csv avant d'importer les données ajoutant de la compléxité non-nécessaire de mon coté (back-end), le fait que le programme se fasse détecter comme cheval de Troie alors qu'il ne l'est point
- Le fait que certaines choses ne marchent tout simplement pas comme prévu (faute des modules non adaptés à tout environment) devant également être résolus à chaque fois, ou réactions entre modules causant plus de problèmes.
- Règle d'or de ce projet : un problème en cache deux autres : nombreuses sont les fois où je me suis trouvé à résoudre un problème, que ce soit dans ma partie ou en aidant les autres sur leur GUI, et ce problème revèle qu'en verité il y en avait d'autres sous-jacents.