POLYTECH SORBONNE

SCIENCES SORBONNE UNIVERSITÉ

Rapport projet : There is no planet B

UE : C++ - UML MAIN

Étudiants :
Tahar AMAIRI
Hamza RAIS

Enseignante :
Cécile BRAUNSTEIN

15 janvier 2022

Table des matières

1 Introduction

À quoi peut donc correspondre un thème intitulé "There is no planet B" ? Il peut y avoir une multitude d'interprétations pour un sujet si vaste, mais au sein de notre groupe cela rime avec essentiellement **l'écologie**.

Aujourd'hui, il est devenu très difficile de définir ce terme et tous les enjeux qui y gravitent autour. En effet, le sujet de l'écologie s'est immiscé dans de nombreux domaines (politique, sociétal, économique, marketing etc...) au point d'en oublier l'essence même : l'étude des interactions entre des êtres vivants et leur milieu. C'est pourquoi nous allons donc aborder l'impact des activités humaines sur l'environnement, ce qui se résume à étudier les émissions anthropiques mondiales de CO_2.

Pour cela, nous avons développé une application graphique se basant sur l'équation de Kaya. Celle-ci permet de relier les émissions de CO_2 à certains paramètres économico-sociaux comme la population, la richesse et la technologie. En d'autres termes, cela permet de voir comment l'évolution de ces paramètres affectent celle des émissions de CO_2.

Finalement, au delà de ce concept initial, l'idée sera de pouvoir prédire les émissions de CO_2 durant les prochaines années selon plusieurs scénarios que l'utilisateur pourra fixer à l'aide de l'application.

2 L'équation de Kaya

2.1 Présentation

L'équation de Kaya est une simple égalité mathématique :

$$CO_2 = POP \times \frac{PIB}{POP} \times \frac{E}{PIB} \times \frac{CO_2}{E}$$

Avec :

- CO2 : les émissions anthropiques mondiales de CO2.
- POP : la population mondiale.
- PIB : le produit intérieur brut mondial.
- E : la consommation mondiale d'énergie.

On remarque l'apparition de différents rapports :

- $\frac{PIB}{POP}$: la production par personne (richesse, croissance).
- $\frac{E}{PIB}$: l'intensité énergétique de l'économie (coût énergétique pour créer un bien ou un service).
- $\frac{CO_2}{E}$: le contenu en gaz à effet de serre de l'énergie (quantité de CO_2 émise pour disposer d'une quantité d'énergie donnée).

Ainsi, cette équation lie le niveau du CO_2 à différents paramètres économico-sociaux permettant ainsi de voir l'impact de l'évolution de ces rapports sur les émissions de CO_2 car tout changement effectué sur le côté droit de l'équation (rapports) se répercute sur celui de gauche (CO_2).

2.2 Utilisation

Afin d'utiliser l'équation, il faut tout d'abord modéliser ces différents paramètres afin de pouvoir prédire leur évolution future. Pour cela, nous allons donc utiliser une **régression linéaire simple** qui sera basée sur des données allant de 1965 jusqu'à 2019 (voir le dossier data de notre git). Par souci de simplicité, nous allons modéliser tout cela sous **R** (voir le script).

En ayant maintenant un moyen de prédire les prochaines valeurs des paramètres de l'équation de Kaya, on pourra mettre en place un outil de prédiction se basant sur des scénarios fixés. De cette manière, on pourra comparer l'évolution des émissions de CO_2 sous deux scénarios différents :

- le scénario par défaut, donné par la régression linéaire.
- le scénario fourni par l'utilisateur après avoir modifié l'évolution des différents rapports.

Afin d'effectuer cette évolution, on utilisera un pourcentage p d'évolution par année de manière à ce que les rapports suivent la suite :

$$y_{n+1} = y_n \times (1 + \frac{p}{100}) \text{ avec } y_0 = \text{la valeur prédite du rapport en 2020.}$$

Ainsi, pour un p positif on obtiendra une croissance et à l'inverse pour un p négatif, une décroissance. Un scénario sera donc défini par ces différents pourcentages qu'on pourra fixer pour chacun des rapports et ce via notre application.

3 KayaGUI : l'application graphique

3.1 Procédure d'installation

Notre application utilise la librairie graphique **SFML** et utilise **C++17**. Il faut donc un compilateur supportant C++17 et avoir installé au préalable la version **2.5** (minimum) de SFML (plus de détails **ici**).

3.2 Présentation

Tout d'abord, voici l'interface principale de notre application :

FIGURE 1 – KayaGUI

On remarque tout d'abord le rappel de l'équation de Kaya et la définition des différents paramètres la composant. Ensuite, l'interface se sépare en deux sections :

— à **gauche** : nous avons la partie "plot" permettant d'afficher les graphiques contenant les données et les droites des moindres carrés pour chacun des paramètres à l'aide des boutons `PLOT`.

— à **droite** : nous avons le pronostiqueur de scénario. Afin de l'utiliser, il suffit de remplir les cases des pourcentages (si rien n'est indiqué, la valeur prise par défaut est 0 %/année). De plus, il faudra aussi indiquer l'année de début et de fin : afin d'obtenir des résultats cohérents, il est vivement recommandé de prendre une année de départ supérieure ou égale à 1965 (car la régression linéaire débute en cette valeur). Par ailleurs, si un intervalle incohérent est donné, par exemple avec une année de début supérieure à celle de la fin, alors l'intervalle pris par défaut est $[2022; 2023]$. Finalement, il suffit d'appuyer sur le bouton `COMPUTE` pour obtenir le résultat en forme de graphique.

3.3 Utilisation

Pour illustrer comment notre application fonctionne, prenons un exemple cohérent. D'après le dernier rapport du GIEC, afin de respecter la limite des 1.5 °C de l'accord de Paris, les émissions de CO_2 doivent diminuer de 45 % d'ici 2030 par rapport à leur niveau de 2010. En d'autres termes, en 2030, nous devons émettre 30 Gt de CO_2 au lieu de 47 Gt. Afin d'appliquer cette réduction nous allons prendre un scénario cohérent :

- POP et $\frac{PIB}{POP}$: pour des raisons d'éthique, nous n'allons pas décroître le niveau de la population ni celui de la richesse. Bien plus, sur une échelle de temps si courte et avec la pandémie actuelle, ces deux paramètres ne risquent pas d'évoluer fortement d'où $p_{POP} = 1\%/\text{année}$ et $p_{\frac{PIB}{POP}} = 5\%/\text{année}$.
- $\frac{E}{PIB}$: avec une possible amélioration des processus industriels, on peut imaginer qu'on puisse produire un bien de manière beaucoup plus efficiente durant les années à venir ($p = -3\%/\text{année}$).
- $\frac{CO_2}{E}$: avec la démocratisation des énergies renouvelables, on peut estimer une réduction du contenu en CO_2 dans l'énergie ($p = -6\%/\text{année}$).
- Et concernant l'intervalle de temps, nous prendrons $[2020; 2030]$.

Voyons donc maintenant ce que l'on obtient :

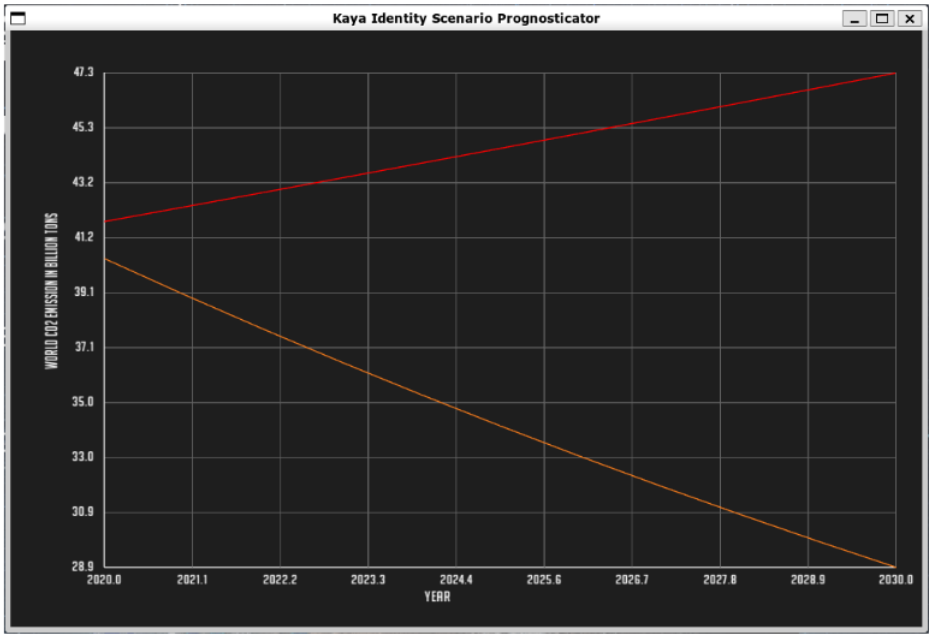

FIGURE 2 – Résultat du pronostiqueur de scénario

Ici, la courbe rouge représente le scénario par défaut et celle en orange le scénario qu'on a indiqué. Afin de respecter la recommandation du GIEC, il faudrait donc augmenter la part des énergies renouvelables de 60% dans la production d'énergie mondiale et utiliser 30% d'énergie en moins pour produire un bien ou un service. Cela parait cohérent surtout que ces résultats suivent les objectifs fixés par l'UE concernant le changement climatique.

4 Description du code

4.1 Les contraintes

Dans cette partie nous allons discuter des différentes contraintes du projet :

— **8 classes** : l'application utilise un total de 9 classes, qu'on peut classer comme suit :
  — pour afficher les fenêtres : `Window`, `WindowPlot`, `WindowMain`.
  — pour afficher les graphiques : `Plot`, `PlotAxes`, `PlotData`.
  — pour réagir aux inputs de l'utilisateur : `CheckPlot`, `CheckCompute`.
  — pour enregistrer les inputs de l'utilisateur : `TextField`.

— **3 niveaux de hiérarchie** : cette contrainte permet aux classes `WindowPlot/Main` de manipuler les propriétés des fenêtres SFML directement via la classe abstraite `Window` héritant de `sf::RenderWindow`.

— **2 fonctions virtuelles** : pour afficher les graphiques, on override la fonction virtuelle `draw` de la classe `sf::Drawable` et pour exécuter les fenêtres on override la fonction virtuelle `run` de la classe `Window` :

```
//to draw
void draw(sf::RenderTarget& target,sf::RenderStates states) const override
//Open the main or a plot window
void run() override
```

— **2 surcharges d'opérateurs** : l'une permettant d'accéder directement aux données enregistrées au sein de la classe `PlotData` et l'autre de vérifier si le curseur de la souris est sur un bouton :

```
//access data operator with index i
sf::Vector2<double> operator()(size_t i) const
//check if the mouse is on a button
bool operator[](sf::Vector2f mouse_coord) const;
```

— **2 conteneurs STL** : pour stocker tous les objets qu'on manipule (`sf::Sprite, sf::Text, std::string, std::shared_ptr`), on utilisera le conteneur `std::vector` car il est très flexible (gestion de la mémoire automatique, accepte les classes SFML). Pour représenter l'intervalle des données qu'on manipule, on utilisera le conteneur `std::pair` car il ne peut contenir que deux éléments.

— **Code bien commenté** : pour cela nous avons commenté notre code en utilisant le format **Doxygen**.

— **Pas d'erreurs avec Valgrind** : nous nous sommes basés sur l'idiome "**Resource acquisition is initialization (RAII)**" en utilisant des `std::shared_ptr`, évitant ainsi toute fuite de mémoire. Cependant, Valgrind envoie tout de même des erreurs mais celles-ci sont relatives aux drivers graphiques.

— **Rendu par dépôt git** : nous avons utilisé GitHub et complété un `README`.

— **Utilisation d'un makefile** : nous avons deux makefiles : l'un permettant de compiler le programme et l'autre les tests. Bien plus, les deux possèdent des règles `clean`.

— **Utilisation de tests unitaires** : avec l'aide de la librairie `Catch`, nous avons mis en place 40 tests afin de garantir le bon fonctionnement de nos classes.

4.2 Diagramme UML

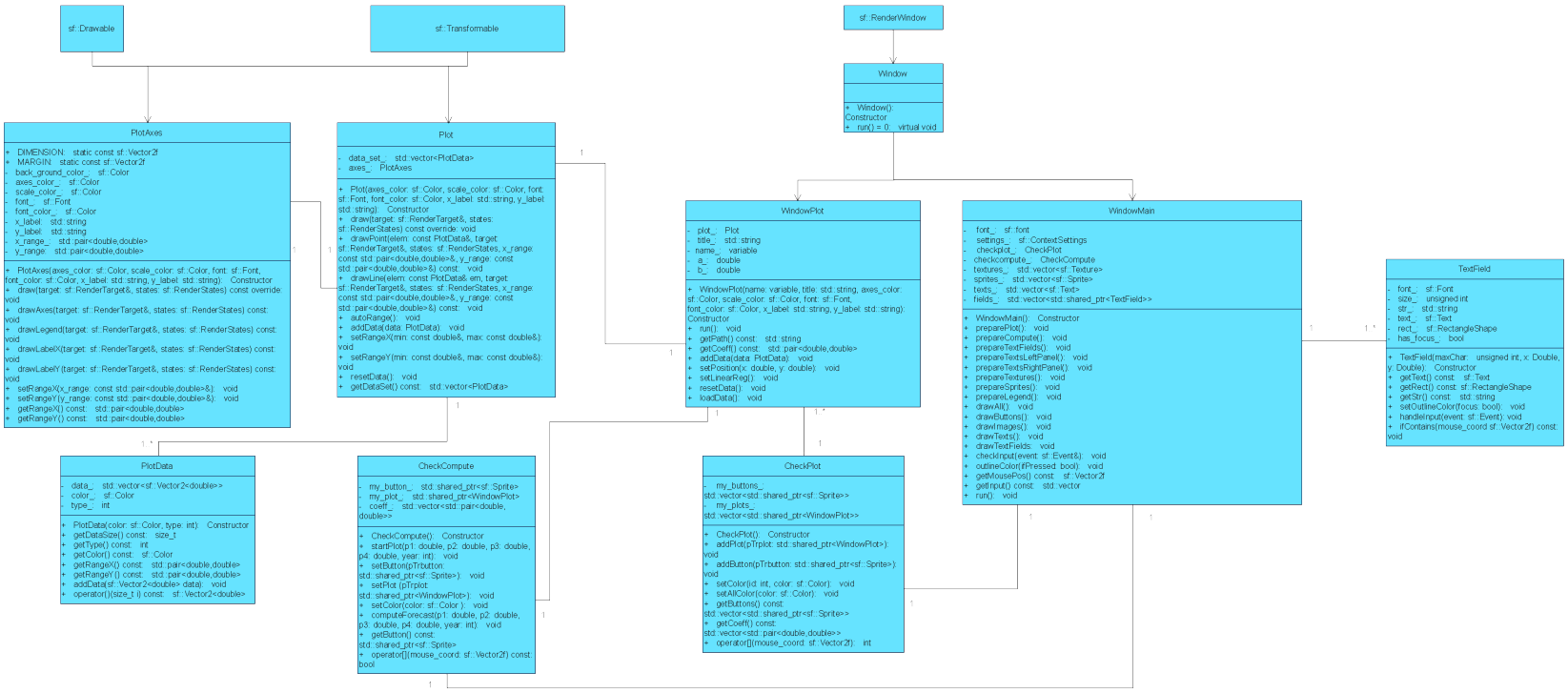

FIGURE 3 – Diagramme UML

5 Fiertés

5.1 L'équation de Kaya

Comme vous avez pu le constater, l'équation de Kaya est très simple et n'implique aucune notion complexe. Néanmoins, c'est dans cette simplicité que réside sa force : pouvoir lier de façon directe les émissions de CO_2 à des paramètres dont l'humanité peut avoir une influence dessus. Accompagner donc cette équation d'une interface et d'un modèle avec lesquels nous pouvons étudier les émissions de gaz à effet de serre nous montrent qu'en théorie l'humanité PEUT résoudre le problème du réchauffement climatique. Selon le scénario que nous avons mis en place, il suffirait d'investir massivement dans les énergies renouvelables et d'améliorer l'efficience avec laquelle nous produisons des biens. Il nous manque que de la volonté et c'est donc satisfaisant d'arriver à cette conclusion car rien n'est encore joué.

5.2 Régression linéaire

Toute la partie de modélisation et de prédiction avec le modèle de régression linéaire sur R est essentielle pour le fonctionnement de notre application, car sans elle on ne pourrait obtenir les résultats du scénario fourni par l'utilisateur. Tout le travail effectué lors de la conception de cette partie provient de l'enseignement qu'on a eu lors de notre cours de statistique inférentielle. Ainsi, pouvoir appliquer les connaissances qu'on a acquise lors de deux cours différents dans un même projet est une satisfaction pour nous car c'est certainement une compétence dont on aura besoin pour plus tard.

5.3 Partie graphique

Sous SFML, il n'existe pas de classes ou de fonctions permettant d'effectuer un plot comme sur Python avec le module matplotlib. Nous étions donc obligés de tout programmer depuis le début afin de pouvoir afficher des graphiques. L'affichage se fait via 3 classes : `Plot`, `PlotAxes` et `PlotData`. Celles-ci n'ont aucune indépendance avec le reste du code (à part avec SFML) et peuvent donc être très bien utilisées en tant que librairie externe dans un autre projet. Bien plus, nous devions prendre en compte le caractère aléatoire de nos données ce qui nous a obligé de programmer cette partie d'une manière dynamique pour qu'elle puisse se scale automatiquement sur les données (par exemple pour l'intervalle des axes).

La deuxième implémentation dont nous sommes fiers est la zone d'input pour le pronostiqueur de scénario. En effet, ce n'était pas chose facile de mettre à jour l'affichage à chaque modification de l'utilisateur. Dans le même cadre, il y a aussi le changement de couleur lorsqu'un utilisateur passe la souris par dessus un élément dont il peut intéragir avec (comme les boutons `PLOT` et `COMPUTE` mais aussi les rectangles d'input).

Finalement, pour une application graphique utilisant une librairie très simple telle que SFML (à l'inverse de Qt par exemple, qui offre beaucoup plus de choix), nous sommes tout de même fiers de son esthétique et de la clarté de l'interface.

6 Conclusion

En conclusion, ce projet a été un moyen de découvrir la bibliothèque SFML et reconsolider tous les acquis qu'on a développé lors de ce semestre en C++ en passant du paradigme de la programmation objet à celui du RAII. Bien plus, cela ne se limite pas uniquement à la programmation car c'était aussi l'occasion de mettre en pratique les connaissances de nos cours de statistiques inférentielles.

Au delà des compétences acquises, nous avons pensé à certaines pistes d'améliorations futures :

- implémenter du multi-threading afin de pouvoir manipuler l'interface principale tout en ayant ouvert une deuxième fenêtre graphique. En effet, notre programme ne le permet pas actuellement car il tourne sous un unique thread. Bien plus, de cette manière nous pourrons afficher plusieurs scénarios en simultané. Il est important de noter que nous avons essayé de le faire mais nous avons rencontré d'importants conflits avec SFML.
- utiliser différents modèles que la régression linéaire afin de modéliser les différents paramètres de l'équation de Kaya : AR, ARMA pour les séries temporelles, le cycle du carbone, le modèle Verhulst pour la population etc...

Finalement, c'est toujours si satisfaisant de travailler sur un projet et de le voir évoluer en partant d'une idée abstraite pour finir avec une application concrète.