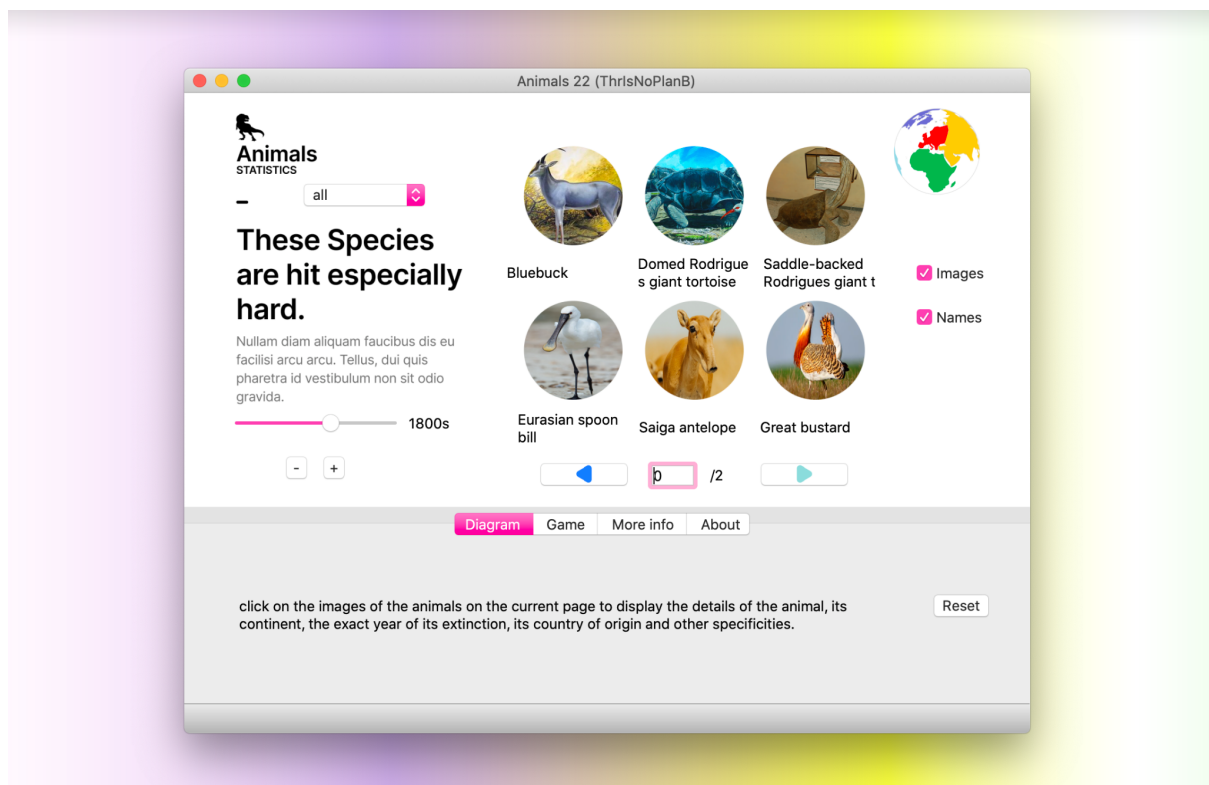


RAPPORT DE PROJET

Programmation orientée objet en C++



Rio & Genty

23/01/2022

C++ – EISE 4

I. Prémices du projet

1. Réflexion et mise en place du projet

Lors de notre premier échange, nous avons rapidement envisagé la création d'une infographie. Tous deux sensibles au changement climatique et à l'extinction des espèces à travers le monde, il nous a semblé évident de profiter de ce projet et son thème "There is no planet B" pour montrer la quantité d'espèces disparues. Notre infographie a pour but de montrer que l'espèce humaine est loin d'être la seule sur Terre.

Nous nous sommes lancés sur une infographie pouvant montrer à l'utilisateur les espèces disparues selon le continent, le règne animal et la décennie de disparition

2. Recherches effectuées

Afin de trouver des espèces disparues selon les continents et les décennies, nous avons alors cherché une base de données à lire et à traiter. Après recherches, nous avons trouvé une base de données sous forme de fichier csv (Comma Separated Value). Ce genre de fichier permet de stocker de nombreuses données par ligne. Sur chacune de ces lignes, on trouve des valeurs utiles pour la description des données attendues, séparées par des virgules.

```
Common name,Scientific name,Extinction date,Decades,Range,Continent,Category
Giant aye-aye,Daubentonia robusta,1000 AD,1000s,Madagascar,Africa,Mammals
Malagasy hippopotamus,Hippopotamus sp.,1000 AD,1000s,Madagascar,Africa,Mammals
Microgale macpheeii,Type of shrew tenrec,100 AD,100s,Madagascar,Africa,Mammals
Voay robustus,,100 AD,100s,Madagascar,Africa,Reptiles
Giant fossa,Cryptoprocta spelea,Before 1400,1300s,Madagascar,Africa,Mammals
Archaeoindris,,Before 1500,1400s,Madagascar,Africa,Mammals
Archaeolemur,,Before 1500,1400s,Madagascar,Africa,Mammals
```

Format des données dans un .csv

Cette base de données retenue contient le nom commun et le nom scientifique de l'espèce, la date et la décennie de sa disparition, et le lieu et le continent de sa disparition, ainsi que son règne animal (Oiseaux, Mammifères, Reptiles, ...)

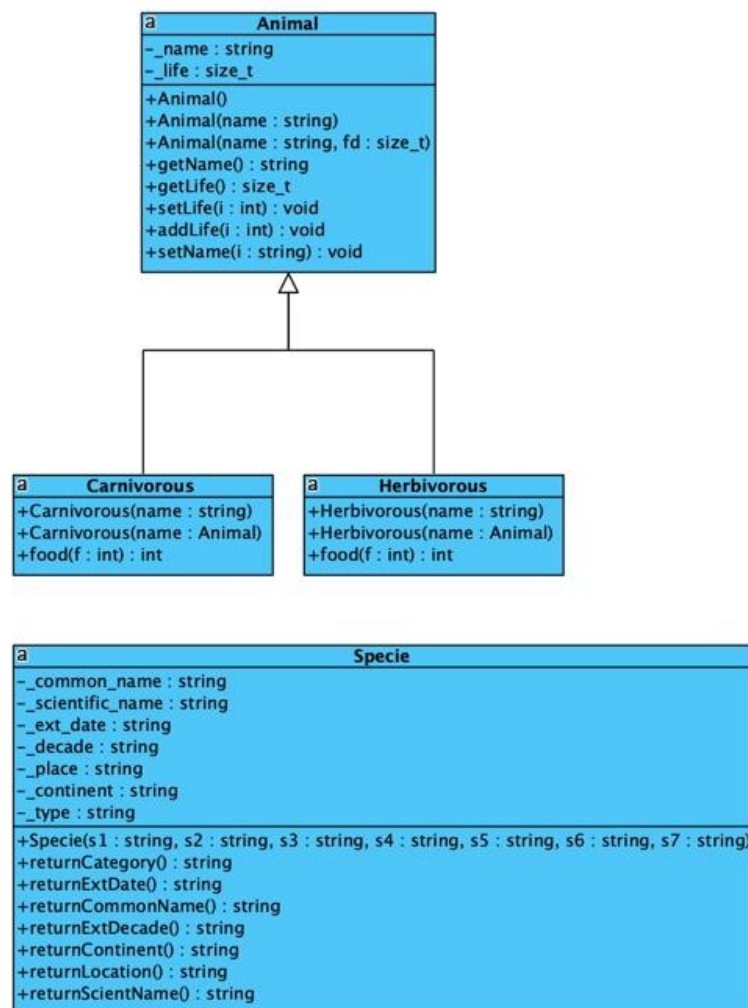
Elle a été trouvée sur un site de partage de ressources pour le traitement de données, Kaggle.com. Vous trouverez le lien vers la base de données dans les sources. Le fichier a été modifié par rapport à l'original, afin de correspondre au format de données que nous allions utiliser.

3. Diagramme UML

Vous trouverez ci-dessous le diagramme UML du projet, avec les classes utilisées dans le code, pour la partie backend.

La classe 'Specie', permet de décrire une espèce contenue dans le fichier csv, avec tous les attributs attendus.

Les classes 'Carnivorous' et 'Herbivorous', héritant de 'Animal', permettent d'avoir des objets pour le petit jeu implémenté dans l'infographie.



Au niveau de la programmation front-end nous avons utilisé la librairie Qt qui fonctionne sur un système de programmation objet aussi avec des classes déjà faites qui ne rentrent pas dans notre UML car nous ne les avons pas créés, simplement quelques redéfinitions de fonctions.

II. Description du code backend

1. Lecture du fichier CSV

La première chose à effectuer dans le code est la lecture du fichier .csv afin de récupérer toutes les espèces. La fonction permettant la lecture du CSV est décrite dans le fichier 'CSVReader.hh'. Cette dernière retourne un vecteur de vecteur de chaîne de caractère, ce qui nous donne un tableau à 2 dimensions. Nous ouvrons le fichier à l'aide de la fonction 'open' de la librairie <fstream>. Ensuite, nous lisons le fichier ligne par ligne à l'aide de la fonction 'getline' de la librairie <string>, qui prend en argument un stream et une string pour lire la ligne du stream et la place dans la string. Ensuite pour chaque ligne lue, on réutilise la fonction 'getline', mais en ajoutant un argument, un caractère (ici ',') pour lire chaque champ de l'espèce dans le .csv

Chaque information sera stockée dans le tableau 2D retourné par la fonction.

2. Classes, attribut et méthodes

- Classe Specie

Au niveau backend, on a tout d'abord une classe 'Specie' définie dans le fichier 'Specie.hh'. Cette classe décrit chaque espèce lue dans le .csv.

Elle comprend différents attributs définissant les espèces, comme le nom commun, le nom scientifique, l'année et la décennie de disparition, le continent et le lieu de disparition ainsi que la catégorie dans laquelle l'espèce se trouve.

Cette classe possède des méthodes de 'setter', permettant de récupérer chaque attribut privée de la classe, sans avoir la possibilité de les modifier.

La classe possède également une surcharge de l'opérateur '<<' pour l'affichage de chaque espèce, surtout utilisé pour le débogage du code.

On peut par conséquent créer un vecteur de 'Specie' pour créer un tableau de ces espèces et donc manipuler plus facilement les espèces dans l'infographie.

- Classe 'Animal', et des ses classes filles 'Carnivorous' et 'Herbivorous'

Pour le code du petit jeu ajouté à l'infographie, nous avons implémenté une classe 'Animal' qui donne lieu à deux classes filles 'Carnivorous' et 'Herbivorous'

Ces classes permettent d'avoir deux animaux, un carnivore et un herbivore.

On a tout d'abord 'Animal', une classe abstraite donnant le nom de l'animal et une méthode 'food' qui permet de retourner si l'utilisateur a entré le bon type d'alimentation.

Cette méthode 'food' est donc virtuelle dans 'Animal' et redéfinie dans les classes filles 'Carnivorous' et 'Herbivorous'. En effet, la viande et les végétaux sont représentés par des entiers, qui retourneront 0 ou 1 si cela correspond à la bonne nourriture pour, respectivement, le carnivore ou bien l'herbivore.

Nous avons, pour ces classes, effectué des tests unitaires.

3. Structure du code

Au niveau backend, on commence tout d'abord par lire le .csv, que l'on stocke dans un vecteur de vecteur de string, que nous ne ferons que lire dans le programme. Par la suite, nous nous plaçons dans un boucle infinie pour mettre à jour la liste d'espèce "active", celle à afficher dans l'infographie. On effectue 3 tris, un tri prenant en compte le continent, ensuite la catégorie puis la décennie. A chaque tri, on recompose un vecteur temporaire, et le dernier vecteur composé permettra au final de créer le vecteur d'espèce.

III. Description du code graphique

1. Explication globale

L'interface graphique de l'application a été créée en majeure partie sur l'Ide Qt et la fenêtre Qt designer. La librairie Qt contient de nombreuses classes avec beaucoup de méthodes très pratiques et surtout très bien documentées.

L'outil Qt Designer permet de gagner du temps sur le placement des widgets (objets graphiques comme les images ou les boutons) car c'est une utilisation graphique à base de drag & drop. Il faut commencer par créer une fenêtre principale (Mainwindow) dans laquelle on ajoute tous les éléments graphiques. Tout est regroupé au même endroit, ce qui rend l'analyse du code plus fastidieuse. Les fonctions relatives à la partie graphique se situent dans le fichier mainwindow.cpp. Le constructeur de l'objet mainwindow est composé d'une suite d'initialisations des objets et de l'interface ainsi que des variables locales qui servent à la sauvegarde des paramètres et le timer du mini-jeu.

L'application contient un widget QTabWidget qui permet de créer des onglets indépendants. Aussi les background des onglets sont des images au format .png qui ont été désignées via Figma (un logiciel de graphisme orienté web) à partir d'un template de la communauté et libre de droit et d'édition.

2. Interface de l'onglet Diagram

Cet onglet est contenu dans le widget QTabWidget sur la première page. Il est composé de plusieurs boutons, d'images, de labels (sections de texte), et de conteneurs à choix multiples. Les images et le textes varient en fonctions des paramètres de tri qui ont été effectués (continent, familles d'espèces, décennies, pages)

Notre plus grande difficulté ici a été de gérer les images et leur actualisation. La base de données que nous avons récupérée ne contenait pas les images des animaux et vu le nombre d'espèces, il aurait été trop fastidieux de télécharger manuellement toutes les images. Nous avons alors eu l'idée d'utiliser un script python de quelques lignes pour le faire à notre place. Le code est disponible dans le dossier de notre projet ; il suffit de créer un tableau des noms des animaux qu'on souhaite récupérer l'image, on a fait une petite boucle sur le tableau csv en ajoutant les " ; " pour créer notre liste d'animaux et ainsi la copier dans le script python qui télécharge la première image correspondant au nom de l'animal et la place dans un dossier du même nom. Ensuite, l'actualisation de l'interface à chaque interaction, avec des éléments graphiques, se fait avec la longue fonction `update()` qui est une méthode que nous avons ajoutée à la classe mainwindow. Cette fonction effectue un

reset des sections de texte et des images, puis effectue le tri en fonction des paramètres choisis par l'utilisateur qui sont rangés dans des variables locales. une fois le tri effectué, il faut afficher les bonnes images et les bons textes dans les widgets correspondants.

Pour savoir comment est fait l'appel de cette fonction il faut comprendre que les objets Qt disposent de slot (emplacements qui s'activent en fonction de l'interaction de l'utilisateur) ces slots activent des méthodes privées de la classe mainwindow qui ont été implémentées par nous. à l'intérieur de ces différentes méthodes (il y en a 19 pour cet onglet) il y a des opérations de base sur les variables de la classe mainwindow qui vont servir à effectuer le tri ainsi que l'appel à la fonction `update()`, certaines de ces méthodes servent juste d'affichage (je pense au clic sur les images) qui font seulement une mise à jour de la section texte en bas de page qui donne les détails sur les espèces.

3. Interface de l'onglet Game

Pour la partie jeu l'affichage est beaucoup plus basique et contient seulement quelques boutons, une barre de progression qui est désactivée pour l'utilisateur et qui permet d'afficher le niveau de vie de l'animal.

Règles et fonctionnement du jeu :

- On peut choisir deux types d'animaux, carnivore et herbivore via le bouton à choix multiple sur la gauche.
- Ensuite on peut lancer la partie et le timer se lance, diminuant alors la vie de l'animal au cours du temps.
- Pour éviter que le niveau de vie tombe à zéro et que l'animal meurt, il suffit de le nourrir avec la nourriture correspondant à son régime alimentaire.
- Si la nourriture donnée ne correspond pas, alors la vie de l'animal chute encore plus vite.

Il a fallu créer deux instances de la classe animal, un herbivore et un carnivore, en variable globale de la source mainwindow.cpp (il n'y avait pas d'autres solutions plus élégantes...) avec chacun un niveau de vie max et puis ensuite, les boutons agissent soit sur le timer avec `start` & `stop`, soit sur la vie de l'animal avec `meat` & `plants`

4. Onglet More info & About

Pour ces onglets, il s'agit globalement d'informations contenues dans l'image provenant de figma où nous avons ajouté des informations qui nous paraissaient pertinentes et intéressantes. Toutefois, l'interaction utilisateur est très limitée puisqu'il n'y a que deux boutons sur l'onglet about. La méthode appelée effectue une ouverture du navigateur par défaut jusqu'à la page de nos profils linkedin respectifs.

IV. Installation et lancement du code

Le projet est fourni avec un dossier complet où nous avons rangé tous les fichiers de ressource nécessaire. Il y a d'abord toutes les images des animaux (300 environ pour éviter d'avoir un projet trop lourd), puis les images des éléments graphiques ainsi que le .csv qui contient la base de données.

Pour l'installation en ligne de commande pour Mac & Linux :

⚠ Il faut utiliser la librairie Qt suivant le tutoriel suivant [install qt library](#)


Ensuite il suffit de se placer dans le répertoire du projet à l'aide de la commande `@cd`. Il faut à présent compiler le projet avec la commande `qmake` en tapant `@qmake` dans le terminal. La prochaine étape est de lancer l'application (avec la commande `open` pour Mac OS), seulement il faut ajouter un argument cet argument est le chemin où se trouve le dossier du projet avec des guillemets aux extrémités par exemple : `"/Users/thomasrio/1FO/C++/PROJET/TEST"`

la commande pour lancer l'application sera donc :

Mac OS : `@open TEST.app --args "/Users/thomasrio/1FO/C++/PROJET/TEST"`

Linux : `@TEST "/Users/thomasrio/1FO/C++/PROJET/TEST"`

Pour l'installation via l'IDE de Qt (Qt Creator) :

- Installez l'IDE qt creator en cliquant sur le lien suivant [qt_download_webpage](#)
- Ouvrez le projet qt en cliquant sur le fichier .pro
- Ensuite rendez vous dans le fichier source main.cpp, à la ligne 9 changez le chemin d'accès au dossier contenant le projet et ensuite vous pouvez lancer le projet en appuyant sur le symbole  en bas à gauche de qt creator.

Problèmes possibles d'affichages :

1. Si aucun affichage de texte s'effectue à l'écran il faut passer l'interface de votre OS en mode jour (clair) pour Mac OS notamment.
2. Si jamais vous n'arrivez vraiment pas, nous avons upload une vidéo en non répertorié sur youtube qui permet quand même de voir les différentes fonctionnalités du jeu : <https://youtu.be/ZG5CSITHEZM>

V. Difficultés rencontrées & fiertés

1. Les contraintes imposées

Du fait de la nature de notre projet, la création d'une infographie, une des difficultés majeures était de respecter les demandes explicitées dans le barème du projet ; soit les classes et les hiérarchies. Évidemment, l'attente des 8 classes n'est pas respectée du fait que nous n'ayons pas voulu créer de classes qui ne nous auraient pas servies. Faire des

classes pour ne faire que des classes n'aurait pas eu d'intérêt. Le faible nombre de classes implique que nous n'ayons pas les trois niveaux de hiérarchies. Nous avons cependant réussi à en faire deux pour le petit jeu.

Rapport /5					Code /20												Fonctionnalités /14						
Mise en forme	Install	Description de l'appli	Fiertés	diag UML	Propre	Commenté	erreur valgrind	Compil sans err/sans warning	Makefile	8 classes	3 niveaux de hiérarchie	2 fonctions virtuelles	2 surcharges d'opé	conteneurs	Test Unitaires	Plus	Jouable	Beau	Intéressant	Complex	Drôle	Conception (hiérarchie a un sens)	
1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	3	2	2	2	2	2	1	5	

2. Absolute path

L'un des plus gros problèmes que nous ayons rencontré se situait dans Qt. Lors de la compilation dans Qt, le programme ne réussissait à compiler que si on lui fournissait le chemin absolu pour retrouver les fichiers tel que le fichier .csv. Le chemin absolu est le chemin du dossier en partant du dossier racine /home/. Cela peut s'avérer problématique puisque celui-ci diffère selon l'OS et le nom de l'utilisateur, et ce chemin est paradoxalement relatif selon l'utilisateur. La solution trouvée a été de rajouter le chemin absolu du fichier lors de la compilation en tant qu'argument, par l'utilisateur comme montré dans la section install mais cette solution nous ennuie un peu du fait de sa complexité pour l'utilisateur lambda.

3. Fiertés

L'une des difficultés majeures a été la mise à jour des images des animaux, c'est donc un point sur lequel nous sommes particulièrement fiers. Autrement graphiquement, nous sommes très satisfaits de l'apparence de l'application et de sa facilité d'utilisation par tout utilisateur (si on ne prend pas en compte l'installation qui aurait pu être simplifiée).

Autrement, la dimension sérieuse du projet le rendait très intéressant et peut être utilisé par beaucoup de curieux du fait de sa vocation pédagogique. De plus, ce projet peut être facilement amélioré avec sa construction en onglets ce qui pourrait permettre par exemple de diversifier les tris et peut-être ajouter des options pour trier les plantes disparues etc.

VI. Sources utilisées

- Base de données utilisées:
<https://www.kaggle.com/junyaozhang/extinct-animals-database>
- Correction des problèmes et questions diverses :
<https://stackoverflow.com/>
- Généralités C++, description des fonctions :
<https://en.cppreference.com/w/>
<https://www.cplusplus.com/reference/>
- Lien vers le git du projet : https://github.com/jugen667/infographic22_RIO_GENTY