|  |
| --- |
| ESIEE Paris |
| Rapport projet IHM |
|  |

|  |
| --- |
| BORDIER Kévin et CAROY Elodie  01/01/2017 |

Table des matières

[Avant-propos : 2](#_Toc498267285)

[Contexte : 3](#_Toc498267286)

[Personas : 4](#_Toc498267287)

[Maquette : 5](#_Toc498267288)

[Justification de nos choix de maquette : 6](#_Toc498267289)

[Projet : 7](#_Toc498267290)

[Le menu : 7](#_Toc498267291)

[La sauvegarde : 7](#_Toc498267292)

[La carte : 8](#_Toc498267293)

[Pistes d’amélioration : 10](#_Toc498267294)

# Avant-propos :

Nous avons enregistré notre projet dans github dans le répertoire suivant : <https://github.com/KevinBordier/IHM_MBDA2.git> .

Ayant eu des soucis lors de nos différents merge (erreur de destination), la branche principale est “master2” et non “master”.

Toute notre application ne fonctionne pas. Nous n’avons implémenté que les deux premiers types d’indicateur dans le menu. Nous avons un bug dans notre menu que nous n’avons pas réussi à résoudre : lorsque nous réaffichons la carte, celle-ci se dessine au-dessus du menu, il faut donc changer la taille de l’IHM pour que le menu repasse au-dessus et ainsi changer d’indicateur. Nous n’avons pas implémenté la barre de recherche.

# Contexte :

Le but de ce projet est de créer une IHM de visualisation de données géographiques. Le contexte est le suivant :

“Le responsable informatique des lycées d’Annecy vous a demandé de réaliser une application d’aide à l’analyse de données géographiques.

L’idée est de pouvoir offrir un outil aux professeurs (d’histoire géographie plutôt) et aux élèves pour afficher dynamiquement certaines données selon des critères définis par eux.

Les contraintes sont d’avoir un client lourd, de fonctionner sans connexion et de proposer une vue cartographique.

Vous n’avez pas pu obtenir d’accréditation pour rencontrer des professeurs ou des élèves.”

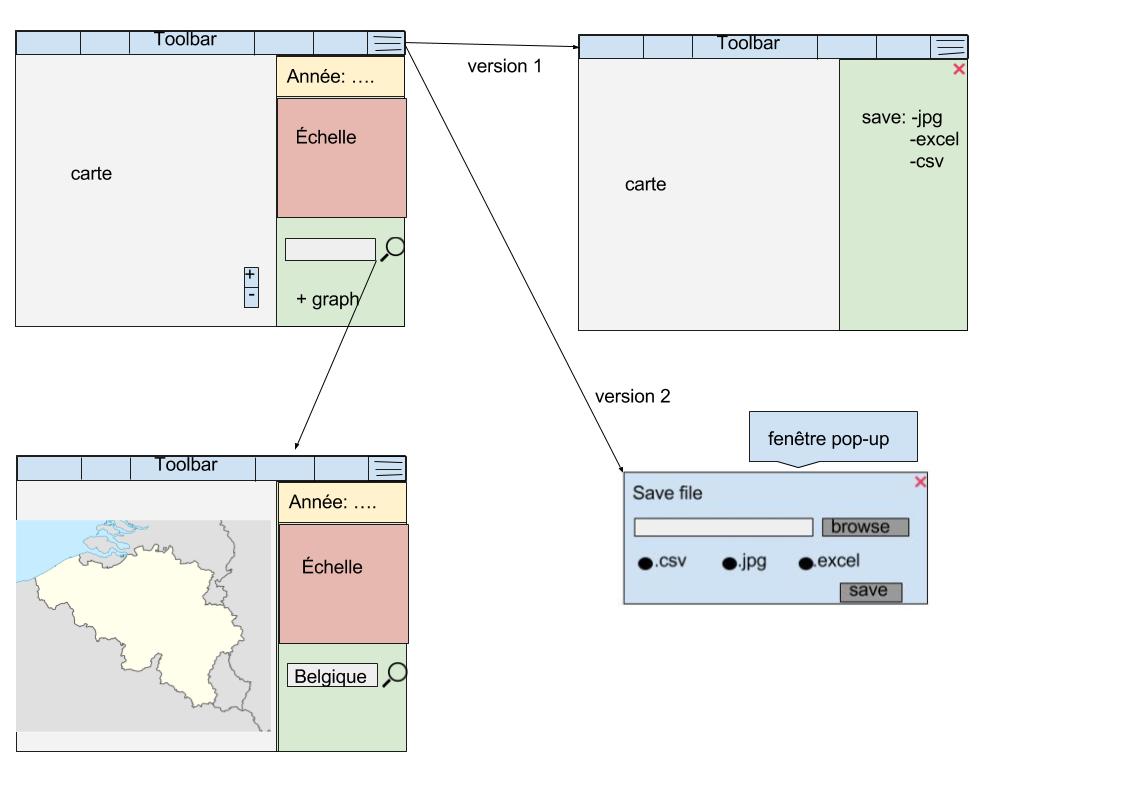
Le code fourni permet déjà de récupérer et de parser les données. Il suffit donc de créer une maquette et de l’implémenter à partir des données fournies.

# Personas :





# Maquette :



# Justification de nos choix de maquette :

Les données fournies étant en anglais, nous avons choisi d’avoir des personas anglophone.

La lycéenne ayant envie de sauvegarder facilement et rapidement son travail, nous avons créé un bouton de sauvegarde pour le faire automatiquement. Nous avons choisi de faire une légende à 5 couleurs car nous pensons que plus de couleur rendrait la lecture difficile. Le menu permet de trouver les indicateurs plus facilement selon nous : on devine plus facilement la catégorie d’un indicateur que le nom de l’indicateur lui-même lorsque l’on cherche une information. L’espace vide en dessous de la barre de recherche a été gardé dans l’éventualité d’une zone de création de graphe lors de l’évolution de l’IHM.

Pour la zone de configuration de la sauvegarde, nous avions pensé faire un panel recouvrant le panel existant sur la droite. Cependant, nous nous sommes aperçu que c’était trop compliqué, nous avons donc opté pour une popup à la place.

# Projet :

### Le menu :

Pour faire le menu, nous avons utilisé la fonction drag & drop de netbeans. Pour garder l’espace disponible pour faire celui-ci, nous l’avons créé en premier avant d’ajouter la carte.

Nous avons créé un JMenuBar dans lequel nous avons ajouté des JMenu contenant des JMenuItem. Seules les parties “Education”, “Health” et le bouton au bout ont été implémentés.

Le bouton du fond permet la sauvegarde automatique de la recherche effectuée. Nous pensions permettre la sauvegarde des données au format excel, csv et une sauvegarde de l’aperçu de l’application au format jpg. L’aperçu permettrai ainsi une visualisation graphique des résultats et de la légende.

Le clique sur le bouton ouvre un popup implémenté dans la classe saveDialog.

### La sauvegarde :

Dans ce popup figure trois radioButton permettant de choisir le format de sauvegarde, une textBox associé à un bouton permettant de choisir l’emplacement de sauvegarde (nous n’avons jamais compris pourquoi la fonction getCurrentDirectory nous retourne le dossier supérieur), un bouton “save” et un bouton “cancel”. Les radioButton sont configurés pour n’avoir qu’un seul d’entre eux coché en même temps.

Cependant, nous n’avons eu le temps d’implémenter que la sauvegarde au format jpg.

Pour sauvegarder l’aperçu de l’application, nous faisons une capture de l’écran au niveau de la fenêtre de l’application. Nous n’avons jamais compris pourquoi une capture simple de l’image de la fenêtre n’a pas fonctionné. De ce fait, nous avons opté pour une capture d’un rectangle de l’écran : ce rectangle est de la taille de la fenêtre et se situe au coin haut gauche de celle-ci. Pour ne pas voir la popup, nous la fermons et nous mettons en place un temps d’attente pour lui laisser le temps de se fermer parfaitement (afin qu’elle n’apparaisse pas en transparence).

|  |
| --- |
| this.dispose();  Container content = this.getParent();  long l = 1000;  Thread.sleep(l);  Point p = content.getLocationOnScreen();  Rectangle capt = new Rectangle(p.x, p.y, content.getWidth(), content.getHeight());  BufferedImage image = new Robot().createScreenCapture(capt);  ImageIO.write(image, "jpg", new File(namefile)); |

### La carte :

La carte s’affiche grâce à une classe qui hérite de JPanel. Cela nous permet de la drag & drop aisément sur un panel de la fenêtre. Pour dessiner la carte, nous parcourons la liste des pays contenu dans DataManger.INSTANCE, pour chacun de ces pays on récupère le tracé du pays (getGeometry().getPolygons()). Puis lorsque le pays pays courant a fini de dessiner ses frontières, on le rempli de noir dans le cas où aucun indicateur n’a été sélectionné. Dans le cas contraire, en fonction de la valeur de l’indicateur choisi pour le pays courant, une des couleurs de l’échelle va lui être associée en fonction de l’intervalle dans laquelle se trouve cette valeur. Cela s’effectue via un ColorProvider (cette classe sera expliqué par la suite).

La carte est équipée de différents ActionListener. Le premier est le zoom avec la molette de la souris (addMouseWheelListener() qui permet d’appeler mouseWheelMoved()) et le second est le fait de pouvoir déplacer la carte avec un drag & drop de la souris (addMouseListener() qui permet d’appeler (mouseDragged()).

Dans le cas du zoom, on utilise un facteur que l’on multiplie par 1,1 si on veut dézoomer. Si on veut zoomer, on divise le facteur zoom par 1,1.

La classe ColorProvider a pour but de retourner une couleur pour un pays donné via la méthode getColorForCountry(). Cette fonction recalcule l’indicateur si celui-ci a été changé. Ensuite les data associées à l’indicateur sont chargées ainsi que certaines valeurs remarquables (le minimum et le maximum qui vont nous servir pour déterminer les différents intervalles de l’échelle). On récupère la valeur associée au pays pour l’indicateur choisi et en fonction de celle-ci et des intervalles préalablement définis, on retourne une couleur pour colorier le pays.

Vient ensuite le petit panel EditYear. Ce dernier est une classe qui hérite de JPanel. Il a pour but de pouvoir modifier l’année à la main avec un JTextField ou avec un JSlider.

Rappel : l’année doit être comprise entre 1965 et 2015 inclus. Chacun de ces deux composants on leur ActionListener associé.

Le LegendProvider est un panel tout comme EditYear, classe héritant donc de JPanel. C’est dans cette classe que sont définies les couleurs de la légende. Ce panel ne fait que de l’affichage. Il s’occupera d’afficher les valeurs qui définissent les intervalles.

Dans la dernière case se trouve un panel EditZoom, créé de la même façon que précédemment. Celui-ci se compose d’un JSpinner, permettant de zoomer ou dézoomer sur la carte en cliquant sur les boutons correspondants. Le niveau de zoom s’affiche. Il est aussi possible de changer ce niveau en inscrivant directement un nombre dans la zone dédiée. Attention, le nombre ne peut être négatif. Cela pose un problème avec le BasicStroke de la carte.

Des Listeners spécifiques ont été créé pour la carte dans le but qu’elle se redessine avec des nouvelles valeurs entrées et/ou modifiées par l’utilisateur avec les panels présentés ci-dessus. Pour se faire, des interfaces ont été créé pour chacun des panels nécessitant une modification par un utilisateur. Ces interfaces ont été implémenté dans la classe de la carte. Dans les classes EditYear et EditZoom la fonction addListener() ainsi qu’une liste de Listener ont été ajouté. La fonction addListener() permet d’inscrire la carte pour pouvoir la modifier. L’attribut quant à lui permet d’appeler une des méthodes implémentées sur la carte lorsqu’une valeur est modifiée.

Nous avons fait cette même gymnastique entre LegendProvider et la Map mais dans le sens inverse. Cela permet à LegendProvider de récupérer les valeurs minimums et le pas qui permettent de calculer les intervalles et de les afficher.

## Pistes d’amélioration :

Les premières améliorations à mettre en place sont la fin de l’implémentation des fonctionnalités visibles que nous n’avons pas eu le temps de créer telles que la sauvegarde au format csv et excel.

Ensuite, nous avons pensé à ajouter une fonctionnalité permettant de générer des graphes de comparaison. Nous avons aussi pensé à une barre de recherche permettant de zoomer sur un pays. Ainsi, l’utilisateur peut ensuite dézoomer pour se concentrer sur une région : par exemple chercher la Belgique pour se concentrer sur l’Europe ...