Projet d'Informatique
Coloriage d'une carte avec 4 couleurs

2017

1 Objectif

Le projet consiste à développer un algorithme de coloriage d'une carte géographique plane dans laquelle chaque pays est de forme rectangulaire. Les pays sont adjacents mais ne se recouvrent pas. Leurs côtés sont parallèles à l'axe des abscisses ou à l'axe des ordonnées

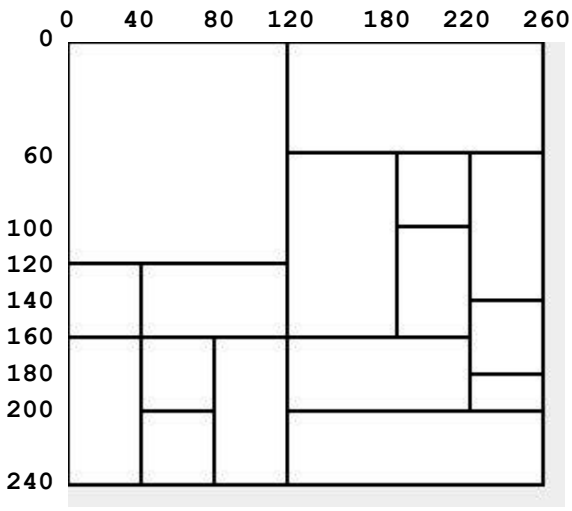
Le but du programme est de colorier chaque pays de sorte qu'aucun voisin n'ait la même couleur et que seulement quatre couleurs soient utilisées.

Initialement, pour donner la description d'une carte, on fournit au programme un fichier textuel (avec l'extension **`.txt`**) dans lequel chaque ligne représente un pays composé d'un nom et de ses coordonnées séparées par des espaces :

`nom xmin ymin xmax ymax`

Par exemple voici, à gauche, le texte de la **`Carte_18.txt`** :

```
P0 0 0 120 120
P1 120 0 260 60
P2 120 60 180 160
P3 180 60 220 100
P4 220 60 260 140
P5 180 100 220 160
P6 120 160 220 200
P7 220 140 260 180
P8 0 120 40 160
P9 40 160 80 200
P10 40 120 120 160
P11 80 160 120 240
P12 0 160 40 240
P13 40 200 80 240
P14 120 200 260 240
P15 220 180 260 200
```



Dans cette représentation, la coordonnée (**`0`**, **`0`**) est située en haut à gauche.

Sur la plateforme **`Minerve`**, vous disposez d'un grand nombre de cartes décrites textuellement et représentées graphiquement (vides).

Vous travaillerez en groupe de projet (environ cinq personnes) pour développer une solution originale.

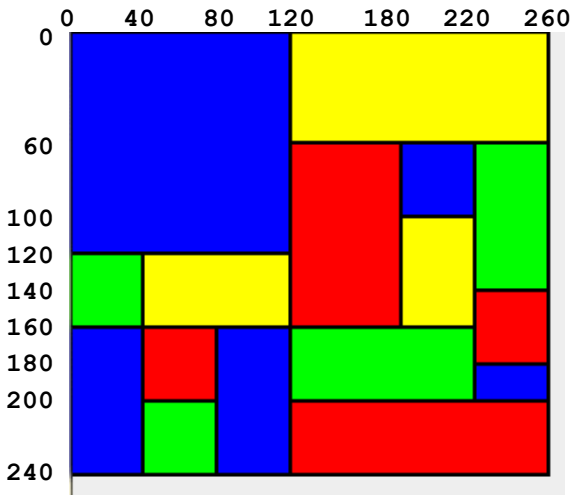
2 Premier Livrable

La première version du logiciel se contente d'afficher dans la console, sous forme textuelle, le résultat du coloriage. Le programme compose un texte identique à celui de départ avec l'ajout de la couleur en fin de chaque ligne :

`nom xmin ymin xmax ymax couleur`

Pour faciliter la visualisation du résultat vous disposez, aussi sur **Minerve**, de deux programmes d'affichage de la carte ; l'un est un fichier **Excel** avec une macro qui traduit le texte en un tableau de cellules colorées et l'autre est un programme, écrit en **Java**, qui dessine une carte dans une fenêtre graphique.

```
P0 0 0 120 120 bleu
P1 120 0 260 60 jaune
P2 120 60 180 160 rouge
P3 180 60 220 100 bleu
P4 220 60 260 140 vert
P5 180 100 220 160 jaune
P6 120 160 220 200 vert
P7 220 140 260 180 rouge
P8 0 120 40 160 vert
P9 40 160 80 200 rouge
P10 40 120 120 160 jaune
P11 80 160 120 240 bleu
P12 0 160 40 240 bleu
P13 40 200 80 240 vert
P14 120 200 260 240 rouge
P15 220 180 260 200 bleu
```



3 Travail demandé

Ce projet peut être résolu soit en utilisant le langage **C** soit en utilisant le langage **Java**. Le langage est choisi librement.

Néanmoins ce choix est un *a priori* car il va impacter de façon significative la démarche globale d'analyse, de codage et de restitution du résultat.

Les quelques étapes proposées ci-dessous sont des indications et forcément il y aura de nombreuses itérations avant que vous arriviez un algorithme satisfaisant.

3.1 Découverte manuelle d'une stratégie

Pour vous aider à découvrir une stratégie de coloriage, vous disposez d'un grand nombre de cartes déjà préparées et disponibles sur la plateforme **Minerve**. C'est à force d'entrainement manuel que vous dégagerez un algorithme pour automatiser ce coloriage.

3.2 Analyse du programme

Du point de vue informatique, avant de coder dans le langage informatique, vous devez concevoir l'architecture de votre programme. Cela consiste à dessiner des diagrammes (**SADT** ou **UML**) qui aident à conceptualiser l'ensemble des algorithmes. C'est sur la base de cette analyse que vous déciderez de la répartition des tâches au sein du groupe de projet.

3.3 Travail en groupe de projet

Afin de constituer un groupe de projet informatique vous devez mettre en place les outils de gestion de version proposés par **Git**. C'est sur la base de l'analyse du projet que vous identifierez les membres du groupe et leurs identifiants seront créés dans le dépôt **Git** partagé. Pendant tout le déroulement du projet vous travaillerez avec **Git** afin, localement d'explorer des voies de solution et sur le dépôt commun, la mise à disposition des branches qui méritent d'être partagées.

3.4 Conventions de nommage

Afin de partager les mêmes façons de coder les identifiants dans les textes des programmes, voici le rappel de quelques règles :

- Types de données (**`typedef`**, **`structure`**, **`classe`**, **`interface`**) s'écrivent en minuscules avec la lettre capitale en majuscule. Les identifiants composés de plusieurs mots s'écrivent avec pour chaque mot la capitale en majuscule. Exemples :
    - **`Pays`**
    - **`VoisinsProches`**
- Les variables et les fonctions (ou méthodes) ont des noms écrits en minuscules. Les identifiants composés de plusieurs mots s'écrivent avec pour chaque mot la capitale en majuscule. Exemples
    - **`index`**
    - **`nombreDeVoisins`**
- les constantes symboliques s'écrivent en majuscules. Les identifiants composés de plusieurs mots sont séparés par le caractère souligné **`_`**. Exemples :
    - **`XMAX`**
    - **`UNITE_HORIZONTALE`**
- Les identifiants des fonctions (ou méthodes) sont des verbes à l'infinitif. Exemples :
    - **`afficher`**
    - **`lireLeFichier`**
    - **`compterLeNombreDeVoisins`**

3.5 Le glossaire des termes utilisés

La gestion de version avec **`Git`** peut vous permettre, entre autre, de partager un glossaire de tous les termes employés dans le projet informatique. Il s'agit d'un tableau que vous définirez au fur et à mesure et qui recense tous les mots en précisant leurs contextes de définition et d'usage.

4 Synthèse de la solution en langage C

Cette version écrite en langage **C** doit fournir le **Premier livrable** tel que défini ci-dessus.

4.1 Analyse du programme développé en langage C

A l'aide du diagramme **SADT**, vous identifierez les données que doivent échanger les différents blocs fonctionnels.

Très rapidement vous aurez soin de définir quelles seront les représentations informatiques des données (tableau, structure, liste chaînée, etc.).

C'est sur cette base de travail (**SADT**, identification des blocs fonctionnels, définition des données) que vous pourrez répartir le travail au sein du groupe de projet ; ainsi chacun saura :

- quelles sont les données dont il dispose (leur type, leur nom, leur domaine de définition, etc.)
- quels algorithmes il doit coder
- quelles sont les données qu'il doit produire (leur type, leur nom, leur domaine de définition, etc.)
- et enfin comment il doit tester sa solution pour valider son travail

4.2 Développement du programme

Les développements informatiques se feront dans l'IDE **Visual Studio Community 2015**.

Pendant toute la phase de développement (codage et tests unitaires) vous aurez soin de travailler avec **Git**, aussi bien en local, sur votre machine, qu'avec le dépôt distant partagé par tout le groupe.

La phase d'intégration doit permettre d'aboutir au programme final et à la production du livrable.

Au-delà des cartes proposées pour s'entraîner, vous devez proposer d'autres cartes originales qui toutes doivent tenter de mettre votre solution en défaut.

4.3 Indications pour le codage en langage C

Les fichiers doivent être définis en couple, l'en-tête **.h** et le fichier source **.c** associé.

Rappelez-vous de la règle selon laquelle un fichier d'en-tête a deux vocations :

- Permettre la compilation du fichier source qui porte le même nom
- Donner l'ensemble des prototypes des fonctions définies dans le fichier source

Souvenez-vous aussi de la nécessité de définir des *directives de garde*.

Le codage unitaire impose de créer (au moins) un deuxième fichier source qui a pour vocation de tester le bon fonctionnement de toutes les fonctions définies dans le fichier source sous test. Par convention il porte le même nom que le fichier source mais avec le préfixe **test_** . Exemple :

- **cartographie.h**, **cartographie.c** et **test_cartographie.c**

5 Synthèse de la solution en langage Java

Cette version en **Java** permet d'aller au-delà du premier livrable (sortie textuelle sur la console) ; en effet ce langage permet si facilement de définir une interface graphique que l'on peut prévoir un livrable :

- qui permet la saisie avec la souris de la carte et produit entre autre le texte de départ
- et l'affichage de la carte colorée suite à la production du texte de sortie.

Autrement dit, on garde comme principe de base, la lecture d'un fichier texte en entrée et la production d'un texte en sortie. Cela permettra aussi des tests croisés avec les solutions proposées en langage **C**.

5.1 Analyse du programme développé en langage Java

A l'aide d'**UML** vous dessinerez les diagrammes qui permettent de modéliser l'ensemble de la solution. Toutes les entités du programme seront alors identifiées (package, classe, interfaces, exceptions, etc.).

C'est sur cette base de travail que vous pourrez répartir le travail au sein du groupe de projet ; ainsi chacun saura :

- quelles sont les données dont il dispose (leur type, leur nom, leur domaine de définition, etc.)
- quels mécanismes il doit utiliser (agrégation, composition, héritage)
- quel algorithme il doit coder au sein de quelles classes (ou interfaces)
- quelles sont les données qu'il doit produire (leur type, leur nom, leur domaine de définition, etc.)
- et enfin comment il doit tester sa solution pour valider son travail

5.2 Développement de la solution

Les développements informatiques se feront dans l'IDE **Eclipse Neon**.

Pendant toute la phase de développement (codage et tests unitaires) vous aurez soin de travailler avec **Git**, aussi bien en local, sur votre machine, qu'avec le dépôt distant partagé par tout le groupe.

La phase d'intégration doit permettre d'aboutir au programme final et à la production du livrable. Au-delà des cartes proposées pour s'entraîner, grâce à la saisie graphique, vous pourrez proposer d'autres cartes originales qui toutes doivent tenter de mettre votre solution en défaut.

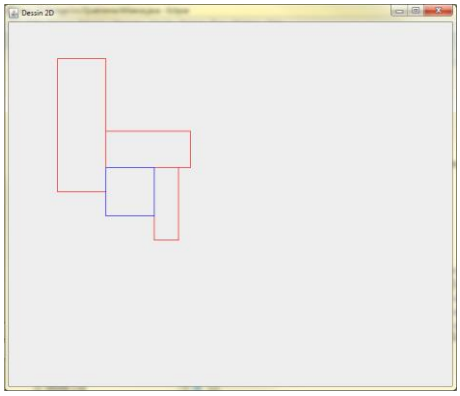
5.3 Création d'une interface graphique de saisie de la carte

Le but de cette partie est de créer graphiquement la carte et de produire le fichier texte qui la décrit. Afin que cet outil soit ergonomique on préconise :

- Dans la fenêtre, la souris permet d'étirer un rectangle dont les sommets[1] sont contraints sur une grille de 10 x 10 pixels.
- Les rectangles sont contraints à ne pas se chevaucher.
- Le rectangle en cours de tracé est de couleur bleu tandis que le reste de la carte est de couleur rouge.
- Il existe quelques commandes au clavier pour aider à finaliser ce dessin :
    - **Z** ou **z** : pour effacer le dernier rectangle dessiné

[1] A tout instant un Rectangle est toujours défini par (x, y, largeur, hauteur) avec x, y coordonnées du sommet supérieur gauche. Ce qui est différent des coordonnées utilisées dans les textes représentatifs de la carte.

- **E** ou **e** : pour enregistrer. A chaque commande d'enregistrement c'est un autre fichier texte qui est enregistré avec un indice qui évolue.
- **C** ou **c** : pour tout effacer



5.4 Indications relatives à l'API Java

Voici une liste non exhaustive des packages Java qui peuvent aider à la réalisation de ce programme

JFrame
JPanel
Color
Point
Rectangle et/ou **Rectangle2D**
Graphics
BasicStroke
LinkedList
ListIterator
MouseAdapter
KeyListener
JMenu, **JMenuBar**, **JMenuItem**

La classe **LinkedList** permet de créer une liste chaînée que l'on peut très facilement parcourir avec une instance de **ListIterator**. Ces deux classes sont paramétrables, on écrit, par exemple **LinkedList<Rectangle>** (idem pour **ListIterator<Rectangle>**) si l'on veut créer une liste de **Rectangle** (et la parcourir).

BasicStroke permet de choisir les propriétés de dessin des figures ; par exemple choisir l'épaisseur du trait. Le dernier choix affecte les nouveaux tracés (de rectangles par exemple).

Graphics est la classe qui permet d'identifier le contexte graphique d'une fenêtre ; c'est via ce contexte graphique que l'on fixe la couleur de la figure et/ou que l'on met en œuvre une instance de **BasicStroke** pour appliquer les choix de tracé.

KeyListener est une interface pour la gestion des évènements liés au clavier. Attention il convient de rendre la fenêtre réceptive à ces évènements en ajoutant dans la définition du **JPanel** :

```
setFocusable(true);
requestFocusInWindow() ;
```