Information géographique

Dans ce cours, tous les algorithmes et programmes seront implémentés en PYTHON. Nous verrons comment, en utilisant ce langage, réaliser diverses opération liées à la géographie :

— créer des classes permettant de représenter différents types d'objets géographiques ;
— représenter ces objets dans un document (ici de type svg ou pdf) ;
— lire des fichiers géographiques ;
— faire du changement de projection ;
— accéder à des bases de données géographiques ;
— générer des graphes ou des histogrammes.

Nous utiliserons des bibliothèques de PYTHON pour nous aider. Cependant, nous pourrions faire la même chose en utilisant d'autres langages : C, C++, JAVA, etc.

1 Construction d'un fond de carte

Durant le premier cours, nous verrons comment lire un fichier contenant des informations géographiques, comment les représenter en mémoire et enfin comment générer un document SVG à partir de ces données. Pour commencer, nous définirons quelques classes permettant de représenter des objets géographiques.

1.1 Les objets géographiques

Chaque objet géographique possède des propriétés. Pour une commune, il pourra s'agir de la population ou du taux de natalité. Pour un fleuve, il s'agira plutôt du débit ou du taux de pollution par exemple. Les propriétés (ou données alphanumériques) sont codées dans les fichiers de données géographiques. Elles peuvent être utilisées pour réaliser une carte en coloriant les régions en fonctions de leurs valeurs. En plus des propriétés, les données géographiques contiennent des informations relatives à la géométrie des objets. Dans notre cas, ces géométries seront de 3 types différents : polygones, polylignes et objets ponctuels.

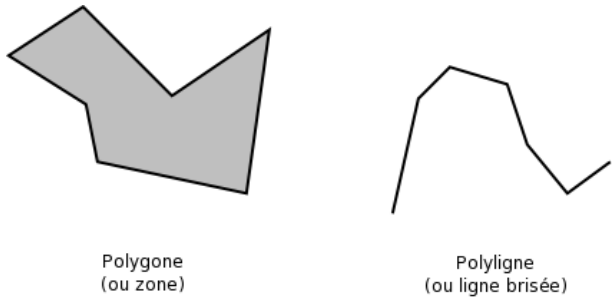

FIGURE 1 – Des exemples de formes géométriques

Les polygones

Ces objets géographiques peuvent servir à représenter des communes, des pays, des bois ou des arrêts de bus par exemple. Ils peuvent être codés en mémoire sous la forme d'une liste de coordonnées (x, y). Parfois, il est possible de les trouver encodés sous la forme de deux listes : une liste pour les x, une autre pour les y. Dans certains cas, le premier point du polygone est explicitement réécrit à la fin de la liste, mais ce n'est pas toujours

le cas. Certains systèmes de coordonnées peuvent aussi tenir compte de l'altitude. Dans ce cas, il faut rajouter une composante z.

Exemple de polygone (format *WKT*[1]) :

```
POLYGON((527998 1785478,527983 1785518,527973 1785583))
```

Les polylignes

Les polylignes permettent de représenter entre autres des routes, des fleuves, des pipelines ou des lignes EDF. Ces objets peuvent également être représentées sous la forme d'une liste de coordonnées. Exemple de polyligne (format *WKT*) :

```
LINESTRING(527998 1785478,527983 1785518,527973 1785583)
```

Les objets ponctuels

Les objets ponctuels permettent de représenter des communes, des arrêts de bus, des mairies, etc. Ils sont représentés en mémoire par des coordonnées : (x, y). Pour leur représentation visuelle nous utiliserons des cercles ou des carrés par exemple. Exemple de point (format *WKT*) :

```
POINT(527998 1785478)
```

Les autres géométries

La norme *OpenGIS* décrit d'autres types de géométrie : le multi-polygone, la multi-polyligne, le multi-point et les collections d'objets. Ces dernières sont des compositions de différentes géométries hétérogènes. À titre d'exemple, une région telle que la bretagne qui est composée d'une région principale et de plusieurs îles peut être représentée par un multi-polygone.

Remarque importante

Certains objets tels que les communes ou les arrêts de bus peuvent être vus sous la forme de polygones ou de points. Cela dépend essentiellement de ce que l'on désire représenter et de l'échelle à laquelle on se place.

Les structures de données

Une structure de données représentant un objet géographique doit prendre en compte à la fois la géométrie de l'objet et ses propriétés alphanumériques. La figure 2 donne un exemple de structure de données permettant de représenter des objets géographiques.

Question de TD

Faites le code PYTHON permettant de représenter les classes d'objets géographiques.

Question de TP

Implémentez les structures de données de la question précédente.

1.2 Lecture de données géographiques

Les données géographiques proviennent pour la plupart d'instituts géographiques (tels que l'IGN pour la France).

Les informations peuvent être données sous la forme de coordonnées :

— géodésiques (latitude, longitude) exprimées en degrés ;

— cartographiques (aussi appelées projetées).

Pour l'instant nous nous intéressons aux coordonnées projetées.

1. http ://dev.mysql.com/doc/refman/5.0/en/gis-wkt-format.html

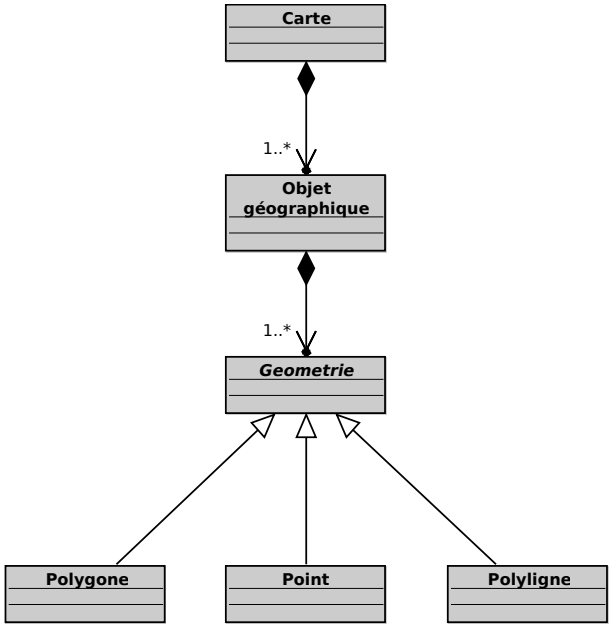

FIGURE 2 – Nos différentes classes d'objets géographiques.

Les formats de fichiers

Il existe plusieurs formats de fichiers tels que MIF/MID [2], Shapefile [3], NTF [4], GML [5]. Certains formats sont binaires (NTF ou ShapeFile) d'autres sont directement lisible en texte (MIF/MID ou GML).

Le contenu des fichiers

Les fichiers géographiques sont composés de différentes informations :
— informations sur la projection ;
— méta-informations sur les données ;
— géométries des objets ;
— propriétés des objets (données alphanumériques).

Dans certains formats, les informations sont réparties dans plusieurs fichiers différents. Dans le cas des fichiers MIF/MID, le fichier MIF contient les informations sur les projections, les méta-informations sur les données et les géométries des objets. Le fichier MID associé comprend les données des objets. Dans certains formats, il peut y avoir une décomposition par couches (*layers*). Chaque couche est composée d'un ensemble d'objets.

La lecture de fichiers géographiques

Pour la plupart des formats de fichiers, il est possible de trouver des spécifications qui permettent de les lire en écrivant un programme adapté. Les applications de cartographie telles que *mapinfo* permettent de visualiser rapidement le contenu des fichiers géographiques. Cela permet de connaitre rapidement le contenu des fichiers avant de créer un programme permettant de les lire.

Il existe aussi des bibliothèques de fonctions libres qui permettent de lire différents formats de fichiers. En C++, C, PYTHON nous pouvons utiliser la bibliothèque GDAL [6] et plus particulièrement sa sous-biliothèque OGR. Cette bibliothèque permet d'ouvrir un grand nombre de types de fichiers différents (entre autre MID/MIF, NTF, GML, Shapefiles).

2. `http://www.safe.com/reader_writerPDF/mif.pdf`
3. `http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf`
4. `http://www.ordnancesurvey.co.uk/`
5. `http://opengis.net/gml/`
6. Geospatial Data Abstraction Library

Le format de fichiers MIF

Nous allons à titre d'exemple étudier le format MIF qui est un format utilisé par l'IGN pour distribuer ses données. La figure 4 montre un exemple de ce que peut contenir un de ces fichiers.

```
1  VERSION 300
2  DELIMITER ","
3   CoordSys Earth Projection 1, 104
4  COLUMNS 6
5      NAME_ENGLI char(32)
6      NAME_ISO char(32)
7      NAME_LOCAL char(32)
8      NAME_FRENC char(42)
9      POP2000 float
10     CODE char(4)
11 DATA
12 REGION 9
13 639
14 -61.465175 10.315097
15 -61.467902 10.323668
16 -61.464006 10.324447
17 -61.45972 10.330292
18 -61.461668 10.336136
19 -61.460889 10.343928
20 -61.464785 10.347045
21 ...
22 -61.689983 10.700426
23 -61.687645 10.703543
24 -61.683359 10.702764
25 -61.677905 10.705881
26 5
27 -60.51724 11.30472
28 -60.516851 11.302382
29 -60.519188 11.298096
30 -60.521136 11.299655
31 -60.51724 11.30472
32 BRUSH(1,0)
33 REGION 9
34 538
35 -61.051793 14.456319
36 -61.055299 14.453592
37 -61.063092 14.45554
38 -61.067377 14.459436
39 ...
```

FIGURE 3 – Exemple de fichier MIF

Nous nous intéressons pour l'instant à la partie `Data`. Dans l'exemple, elle commence à partir de la ligne 11 jusqu'à la fin du fichier. La partie `DATA` contient la liste des géométries des objets. Chaque objet commence par une ligne contenant le type de géométrie et le nombre de géométrie de l'objet (`REGION 9`). Ensuite, pour chaque géométrie, nous avons une ligne représentant le nombre de points (`639`) suivi de la liste des points en coordonées $x\ y$ (`-61.465175 10.315097`).

Exercice de TD. Nous prendrons comme hypothèse que le fichier MIF que nous voulons lire ne contient que des polygones (par exemple les régions françaises). Écrivez une classe `Lecteur` permettant de lire ce type de fichier et d'instancier une carte. Vous pouvez vous inspirer de l'algorithme de la figure 4 et des fonctions décrites dans la figure 6 ainsi que du diagramme de la figure 9.

Exercice de TP. Implémentez les questions précédentes. Vous pourrez utiliser les fichiers disponibles à l'url [7] comme bases pour vos tests.

7. http ://users.info.unicaen.fr/∼jfroment/ens0910/l2/infogeo

```
Fonction: "LireFichierMIF"
Argument
    fich: le nom du fichier MIF a ouvrir
Retourne
    Une liste d objet geographiques
Debut
    f = OuvrirFichier(fich)
    Boucler
        l = f.LireLigne()
        Si l.commencePar("DATA")
            SortirDeLaBoucle()
    l = f.LireLigne()
    listeObjets = listeVide()
    Boucler
        objGeographique = lireObjetGeo(f)
        Si objGeographique == None
            SortirDeLaBoucle()
        listeObjets.ajouter(objGeographique)
    Retourner listeObjets
Fin

Fonction: "LireObjetGeo"
Argument
    f: le fichier dans lequel lire
Retourne
    Un objet geographique ou None si c est la fin du fichier
Debut
    l = f.lireLigne()
    Si l == ""
        Retourner None
    type, nbGeom = l.decouper()
    objGeographique = ObjetGeo()
    Pour i entre 0 et int(nbGeom)
        polygone = Polygone()
        nbPoints = f.lireLigne()
        Pour j entre 0 et int(nbPoints)
            l = f.lireLigne()
            x, y = l.decouper()
            polygone.ajouterPoint(int(x), int(y))
        objGeographique.ajouterGeom(zone)
    lireLigne(f) #nous sautons la ligne "BRUSH (...)"
    Retourne objGeographique
Fin
```

FIGURE 4 – Algorithme de lecture d'un fichier MIF

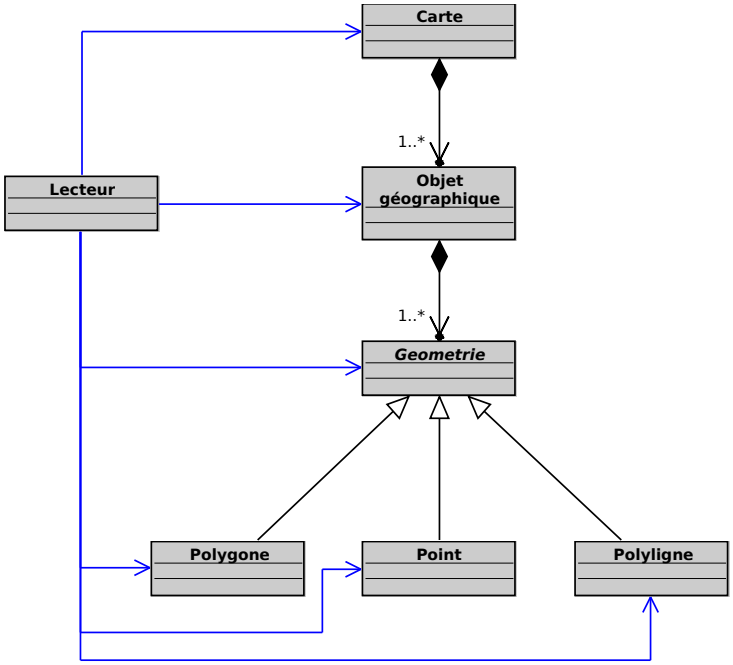

FIGURE 5 – Ajout de la classe `Lecteur`.

```
>>> help(file)
file(name) -> file object

Open a file.

>>> help(file.readline)
readline(...)
    readline([size]) -> next line from the file, as a string.

    Retain newline.  A non-negative size argument limits the
    maximum number of bytes to return (an incomplete line may
    be returned then). Return an empty string at EOF.

>>> help(str.startswith)
startswith(...)
    S.startswith(prefix[, start[, end]]) -> bool

    Return True if S starts with the specified prefix, False
    otherwise. With optional start, test S beginning at that
    position. With optional end, stop comparing S at that
    position.

>>> help(str.split)
split(...)
    S.split([sep [,maxsplit]]) -> list of strings

    Return a list of the words in the string S, using sep as
    the delimiter string.  If maxsplit is given, at most
    maxsplit splits are done. If sep is not specified or is
    None, any whitespace string is a separator.
```

FIGURE 6 – Quelques fonctions utiles

1.3 La génération de cartes

Générer une carte nécessite d'avoir déjà acquis des données et de les avoir représentées en mémoire. Nous travaillerons sur des données déjà projetées.

Les différentes parties d'un document électronique

La plupart des formats de représentation de fichiers peuvent être décomposés en *Header* (entête) et *Content* (contenu). L'entête et le contenu sont très dépendants du format de fichier. L'entête va contenir les infos relatives aux fichiers. Par exemple :

— largeur de la page ;
— hauteur de la page ;
— nombre de pages ;
— encodage des caractères.

Le contenu est constitué d'un ensemble de commandes qui sont interprétées par le logiciel de visualisation pour faire le rendu visuel.

La génération de SVG

Nous allons générer des documents en SVG. Ce format est un format XML développé par le w3c [8]. Il peut être ouvert par un grand nombre d'applications, dont, entre autres, le navigateur internet firefox. Le format SVG permet de représenter des données vectorielles en 2 dimensions. Il permet de représenter des plygones, des lignes, des cercles, etc.

Le SVG est un format XML, donc un format texte avec des balises. Il peut s'incorporer relativement facilement à de l'HTML. Il est possible d'inclure du code javascript au sein du code SVG ou du code HTML pour faire interagir les deux type de documents et les rendre dynamique par exemple. Le site du w3c donne une bonne description de ce format et de ses possibilités. À titre d'exemple, la figure 7 donne un exemple de code SVG. La figure 8 montre le résultat du code SVG

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <desc>Example polygon01 - star and hexagon</desc>
  <!-- Show outline of canvas using 'rect' element -->
  <rect x="1" y="1" width="1198" height="398"
        fill="none" stroke="blue" stroke-width="2" />
  <polygon fill="red" stroke="blue" stroke-width="10"
           points="350,75  379,161 469,161 397,215
                   423,301 350,250 277,301 303,215
                   231,161 321,161" />
  <polygon fill="lime" stroke="blue" stroke-width="10"
           points="850,75  958,137.5 958,262.5
                   850,325 742,262.6 742,137.5" />
</svg>
```

FIGURE 7 – Un exemple de code SVG tiré du site du W3C.

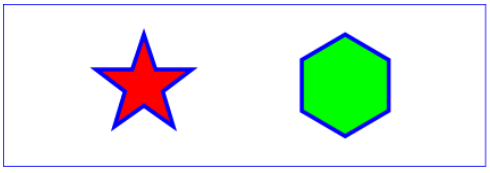

FIGURE 8 – Résultat du code SVG ci-avant.

8. http ://www.w3.org/TR/SVG11/

Exercice de TD. Comme pour la classe `Lecteur`, écrire une classe `Ecrivain` permettent d'afficher une carte dans un document SVG. Le document en question pourra être un fichier passé en paramètre de la méthode.

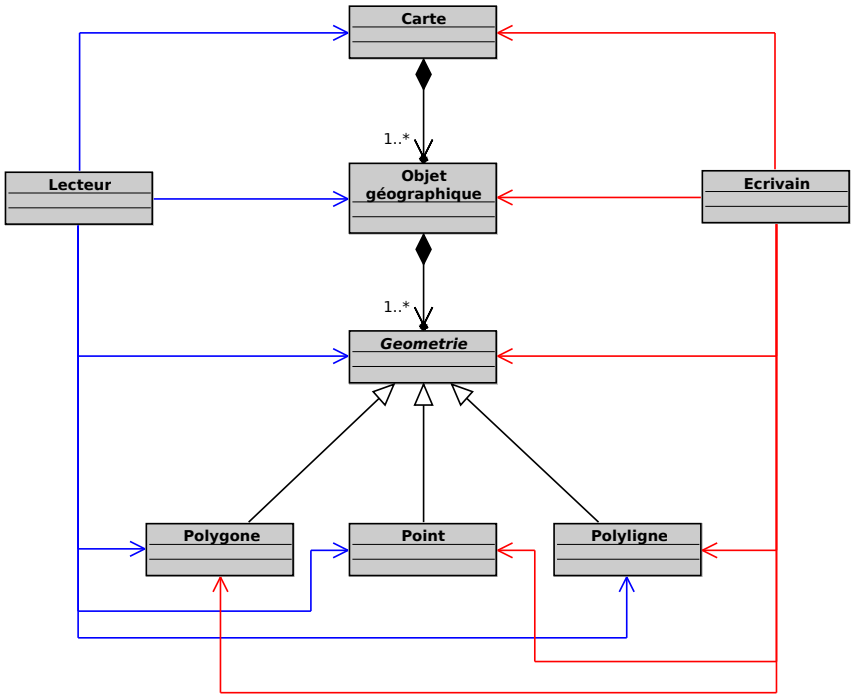

FIGURE 9 – Ajout de la classe `Ecrivain`.

Exercice de TP. À partir du programme écrit lors des exercices de TD, écrivez les méthodes qui permettent de générer un SVG en fonction d'un fichier MIF donné

2 Afficher de l'information

Comme nous l'avons vu au cours précédent, les objets géographiques sont définis à la fois par leurs géométries, mais aussi par leurs propriétés. Le thème de ce cours sera de lire et de représenter les propriétés des objets géographiques.

En nous basant sur les fonctions de lecture de fichiers MIF que nous avons construites au cours précédent, nous allons maintenant lire les propriétés des objets.

2.1 Lectures des propriétés

Les propriétés des objets peuvent être vues comme des couples *(nom, valeur)*. Par exemple, un département français peut avoir les propriétés nommées *Nom_Departement* et *Code_Departement* avec, respectivement, les valeurs *Calvados* et *14*. La structure de données PYTHONque nous allons utiliser pour représenter les propriétés sera donc le dictionnaire. Dans les données que nous allons utiliser, les noms et les valeurs sont dans deux fichiers différents :

— les noms de propriétés sont stockées dans l'entête du fichier MIF ;
— les valeurs sont stockées dans le fichier MID associé.

Dans un premier temps, il faut modifier la fonction de lecture des fichiers MIF pour qu'elle permette de lire les noms de propriétés. Ensuite, il faut ajouter une fonction pour lire les valeurs dans le fichier MID.

Parfois, les propriétés peuvent être définies avec un type et une unité. Pour ce travail, nous ne tiendrons pas compte de ces paramètres.

Lecture des noms de propriétés

Le début de la partie du fichier contenant noms de propriétés est identifiée par la ligne commençant par *Columns*. La deuxième information de cette ligne est le nombre de propriétés.

Chaque propriété est ensuite codée sur une ligne. La première information de chaque ligne est son nom et la seconde information est son type.

Pour simplifier le programme, le type des propriétés ne sera pas utilisé. Nous considérerons qu'il s'agit toujours de propriétés textuelles.

```
1  version 300
2  Charset "WindowsLatin1"
3  CoordSys Earth Projection 3, 1002, "m", 0.000000000000, 46.800000000000,
4  45.898918964419, 47.696014502038, 600000.000, 2200000.000
5  Columns 9
6  Code_International char (200)
7  Code_Region char (200)
8  Nom_Region char (200)
9  Code_Departement_County_UA char (200)
10 Nom_Departement_County_UA char (200)
11 Statut_Administratif char (200)
12 Superficie_km2 char (200)
13 Population_Totale_Sans_Double_Compte char (200)
14 Densite char (200)
15 Data
16 Region 62
17 13
18 176700.00 2448800.00
19 176800.00 2448900.00
20 176900.00 2448900.00
21 176900.00 2449000.00
22 176900.00 2449100.00
23 177000.00 2449100.00
```

FIGURE 10 – Début d'un fichier MIF

Lecture des valeurs des propriétés

Les valeurs des propriétés sont stockées dans le fichier MID associé. A chaque objet géographique du fichier MIF correspond une ligne dans le fichier MID. Chaque ligne est composée de différentes valeurs des propriétés de l'objet séparées par des tabulations. Ces valeurs sont ordonnées de la même façon que dans le fichier MIF.

Pour lire ce genre fichier, le plus simple est d'utiliser la classe *reader* du module *csv*. Cette classe permet d'itérer sur les lignes du fichier en retournant la liste des valeurs de chaque ligne.

Exercice de TD. Modifiez les fonctions de lecture de fichiers MIF/MID de façon à ajouter aux objets géographiques des propriétés.

2.2 Le clustering

À partir des valeurs des propriétés, nous allons modifier la couleur des zones. Pour cela, il faut pouvoir attribuer une couleur à chaque zone. Il existe différentes méthode pour attribuer une couleur en fonction d'un attribut. Nous utiliserons la méthode des quartiles et nous découperons les données en quatre classes différentes (A, B, C, D).

Cette méthode se base sur un calcul de médianes. C'est à dire une valeur q telle que pour liste donnée il y ait autant de valeurs plus petites que de valeurs plus grandes que q.

Les quatre groupes sont définis tels que :

— $objgeo \in A$ ssi $val < q1$
— $objgeo \in B$ ssi $q1 \leq val < q2$
— $objgeo \in C$ ssi $q2 \leq val < q3$
— $objgeo \in D$ ssi $q3 \leq val$

En considérant que *val = objgeom.getProp(nomProp)*

```
>>> print csv.reader.__doc__
    csv_reader = reader(iterable [, dialect='excel']
                        [optional keyword args])
    for row in csv_reader:
        process(row)

The "iterable" argument can be any object that returns a line
of input for each iteration, such as a file object or a list.  The
optional "dialect" parameter is discussed below.  The function
also accepts optional keyword arguments which override settings
provided by the dialect.

The returned object is an iterator.  Each iteration returns a row
of the CSV file (which can span multiple input lines)

>>> s = '"albert"\t"10"\t"12"\n"bernard"\t"11"\t"15"\n"cedric"\t"13"\t"9"'

>>> print s
"albert"        "10"    "12"
"bernard"       "11"    "15"
"cedric"        "13"    "9"

>>> f = StringIO.StringIO(s)

>>> for l in csv.reader(f, delimiter="\t", lineterminator="\n"):
>>>     print l
>>>
['albert', '10', '12']
['bernard', '11', '15']
['cedric', '13', '9']
```

FIGURE 11 – Aide de la classe "reader" du module csv

```
"4EUFR" "53"    "Bretagne"      "22"    "C\^otes d'Armor" "D\'epartement"
"6878"  "550610"        "80.0"
"4EUFR" "11"    "Ile-de-France" "92"    "Hauts-de-Seine"        "D\'epartement"
"176"   "1462347"       "8309.0"
"4EUFR" "52"    "Pays de la Loire"      "49"    "Maine-et-Loire"        "D\'epartement"
"7166"  "742703"        "104.0"
"4EUFR" "11"    "Ile-de-France" "77"    "Seine-et-Marne"        "D\'epartement"
"5915"  "1220221"       "206.0"
```

FIGURE 12 – Début d'un fichier MID

Exercice de TD. Définissez une classe *Quartiles* qui permet de faire un découpage en quatre groupes. Le constructeur de la classe prendra en paramètre une liste d'objets géographiques, le nom de la propriété, et une liste de couleurs. Cette classe possédera une méthode *getCouleur* qui prendra en argument un objet géographique et qui retournera sa couleur.

Exercice de TP.
- — Implémentez la lecture de fichiers MID et la classe Quartiles.
- — Générez une carte de la zone transmanche représentant les densités de population. Vous utiliserez les fichiers *transmanche3* disponibles son le wiki. Les quatre couleurs devront être en dégradé, par exemple : marron, rouge, orange et jaune. Vous pouvez utiliser la balise "g" du svg pour factoriser des propriétés de différents polygone (par exemple leur couleur de remplissage).
- — Ajoutez une méthode *toSVG* à la classe *Quartiles* qui dessine une légende des valeurs à partir des informations de la classe *Quartiles*.

3 La génération de PDF

À partir du programme qui permet de générer du SVG, nous voulons maintenant construire un document PDF. Nous réutiliserons les classes d'objets géographiques, la lecture des données au format MIF et le découpage en quartiles vus lors des cours précedents. Nous ajouterons des méthodes permettant d'afficher les objets géographiques au format PDF ainsi que l'utilisation de rectangles englobants et le changement de repère.

3.1 Pourquoi générer du PDF ?

Le PDF est un format binaire créé par Adobe Systems qui permet d'intégrer des images bitmap, du dessin vectoriel et du texte dans le même document. Il s'agit d'un format propriétaire dont les spécifications sont publiques. Le format PDF est utilisable librement.

Les deux formats SVG et PDF sont assez complémentaires. Pour intégrer un document vectoriel dans une page web, pour réutiliser le contenu d'un document ou pour intégrer de la dynamicité dans le document, il est préférable d'utiliser le SVG. Cependant, le format PDF constitue une meilleur solution dans le cas où il faut que le document soit imprimable, visualisable simplement et que le contenu soit visible par tous de la même façon,

Pour un site internet utilisant la cartographie, il pourrait être intéressant d'utiliser les deux formats. SVG et HTML pour visualiser les informations et pour interagir avec l'utilisateur et PDF pour une sortie d'un document à imprimer.

Le format RML (ReportML) de ReportLab permet de décrire un document en XML et le transformer automatiquement en PDF, cela peut constituer un niveau intermédiaire de document qui possède les avantages du XML tout en pouvant être transformé en PDF.

3.2 Le format PDF

Nous allons utiliser la bibliothèque REPORTLAB pour générer directement des documents PDF. Le site web *reportlab.org* donne accès à une documentation très complète de cette bibliothèque. Dans ce cours, nous n'utiliserons qu'une petite partie des fonctions mises à notre disposition.

Le code de la figure 13 permet de voir comment générer un fichier PDF de taille A4 dans lequel il est écrit *Hello World* à l'emplacement (100mm, 100mm) à partir du coin en bas à gauche de la page.

Dans ce code, $myCanvas$ représente le document PDF. La ligne 5 permet de déclarer ce document. La ligne 6 affiche la chaine de caractère dans le document. La ligne 7 déclare une fin de page (ce qui n'est pas nécessaire pour ce document car il ne contient qu'une seule page). La ligne 8 génère le fichier *hello.pdf* dans le répertoire courant.

L'exemple de la figure 14 illustre l'utilisation des chemins (*paths*) pour dessiner des polygones. La méthode `beginPath` créée un nouveau chemin. Le début du chemin est défini grâce à l'intruction `moveTo` (ligne 8). Ensuite les lignes sont traçées grâce à l'instruction `lineTo` (ligne 13). La méthode `close` de la ligne 14 permet de fermer le chemin. Un chemin qui reste ouvert représenterait une polyligne. Enfin, la méthode `drawPath` permet de dessiner le chemin dans le canvas.

La ligne 23 change la couleur de remplissage. Les trois paramètres sont de nombres entre 0 et 1 qui définissent respectivement les trois composantes : rouge, vert et bleu. Toutes les couleurs peuvent être définies à partir de ces trois composantes. Les lignes 24 et 25 servent respectivement à dessiner le disque et le cadre.

La figure 15 montre le résultat de l'exécution du code précédent.

3.3 Le changement de repère

L'inconvénient du format PDF est qu'il faut transformer les coordonnées originales (en Lambert 2) vers les dimensions de la feuille de travail (qui peut être par exemple au format A4). Dans ce cas, il faut donc procéder à un changement de repère.

Rectangles englobants et changement de repère

Le changement de repère sera effetué en 3 étapes :

— calcul des dimensions du cadre à représenter ;
— calcul des dimensions du cadre vers lequel représenter les données ;
— création d'un objet permettant de transformer les coordonnées d'un objet en fonction des dimensions précedemment calculées.

Pour représenter les dimensions, nous utiliserons des rectangles englobants aussi appelé *bounding box* (cf : figure 16). Les cotés d'un rectangle englobant sont parallèles aux axes. Ils peuvent donc être définis par 4 valeurs : $x_{min}, y_{min}, x_{max}$ et y_{max}.

L'interface de la classe *BoundingBox* est décrite dans la figure 17. Ces *BoundingBox* doivent pouvoir être calculés par les objets géographiques et les géométries. Par conséquent, il faut rajouter à la classe `OjbetGeographique`

et aux classes `Geometrie` des méthodes `getBoundingBox` qui retournent les rectangles englobants de ces objets.

La boundingbox de la zone dans laquelle dessiner peut être définie manuellement en utilisant par exemple sur la taille du format A4 (21cm * 29.7cm). En se basant sur les deux rectangles englobants, il est possible de définir un objet effectuant le changement de repére automatiquement. Ce transformeur de coordonnées (cf. figure 18 implémente une méthode `transforme` qui prend en argument une coordonnée dans le repère défini par la première *boundingbox* et retourne l'image de cette coordonnée dans le repère défini par la seconde *boundingbox*. La méthode `transforme` préserve les proportions originales. L'algorithme de cette méthode est donné par la figure 19

La génération du document PDF

L'algorithme de la figure 20 décrit une méthode qui permet de dessiner un ensemble d'objets géographiques sur une feuille au format A4. Cet algorithme se base les méthodes `getBoundingBox` et `toPDF` de la classe `ObjetGeographique`.

Exercice de TD. Écrivez le code :

- de la classe `BoundingBox`;
- de la classe `Transformeur`;
- de la fonction `ecrirePDF`;
- des méthodes `getBoundingBox` pour la classe `ObjetGeographique` et pour les classes de type `Geometrie` (surtout pour la classe `Polygone`);
- des méthodes `toPDF` pour la classe `ObjetGeographique` et pour les classes de type `Geometrie` (surtout pour la classe `Polygone`).

Exercice de TP. Implémentez les questions de la partie précedente.

Rajoutez le titre, le nom de l'auteur, et la direction du nord dans le document généré.

Vous pourrez aussi rajouter l'échelle du document et bien entendu la génération automatique de la légende.

Optimisez la méthode de transformation des coordonnées de la classe `Transformeur`.

```
1 from reportlab.pdfgen import canvas
2 from reportlab.lib.pagesizes import A4
3 from reportlab.lib.units import mm
4
5 myCanvas = canvas.Canvas("hello.pdf", pagesize=A4)
6 myCanvas.drawCentredString(100*mm,100*mm,"Hello World")
7 myCanvas.showPage()
8 myCanvas.save()
```

FIGURE 13 – Génération d'un *hello world* en pdf.

```
from reportlab.pdfgen import canvas
from reportlab.lib.pagesizes import A4
from reportlab.lib.units import mm
from math import pi, cos, sin

def pentagone(aCanvas, xcenter, ycenter, radius, fill=False):
    p = aCanvas.beginPath()
    p.moveTo(xcenter+radius,ycenter)
    for i in xrange(5):
        angle = (2*pi*i)/5
        x = xcenter+cos(angle)*radius
        y = ycenter+sin(angle)*radius
        p.lineTo(x, y)
    p.close()
    aCanvas.drawPath(p, fill=fill)

myCanvas = canvas.Canvas("pentagone.pdf", pagesize=(120*mm,120*mm))
pentagone(myCanvas, 60*mm, 60*mm, 50*mm)
pentagone(myCanvas, 60*mm, 60*mm, 40*mm)
pentagone(myCanvas, 60*mm, 60*mm, 30*mm)
pentagone(myCanvas, 60*mm, 60*mm, 20*mm)
pentagone(myCanvas, 65*mm, 65*mm, 5*mm, fill=True)
myCanvas.setFillColorRGB(0.5,0.5,0.5)
myCanvas.circle(15*mm, 15*mm, 5*mm, fill=True)
myCanvas.rect(0, 0, 120*mm, 120*mm)
myCanvas.showPage()
myCanvas.save()
```

FIGURE 14 – Génération de pentagone en pdf

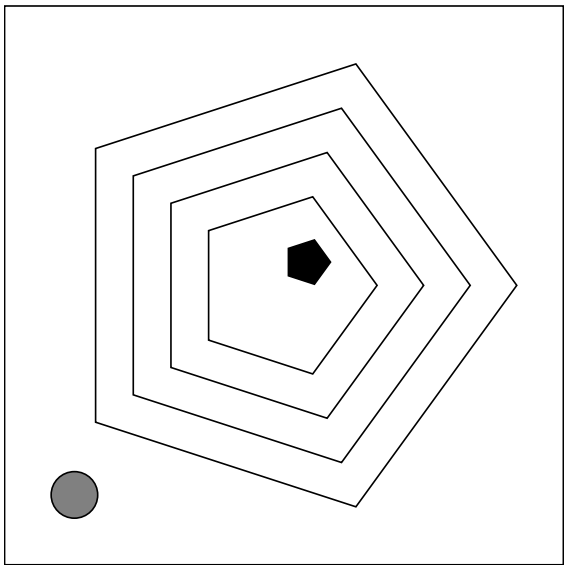

FIGURE 15 – Résultat de la génération des pentagones

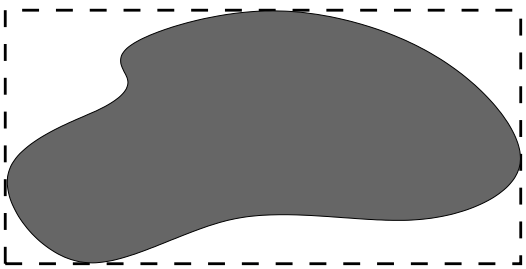

FIGURE 16 – Le rectangle englobant associé à une figure (en pointillé)

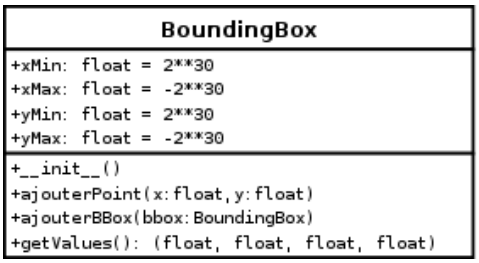

FIGURE 17 – La classe BoundingBox

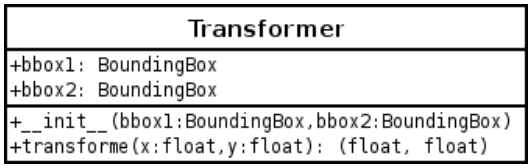

FIGURE 18 – La classe Transformer

```
Fonction: "conversion"
Argument
    bbox1 = la bounding-box representant le repere d'origine
    bbox2 = la bounding-box representant le repere cible
    x, y = la coordonnee a convertir
Retourne
    xc, yc = la coordonnee convertie
Debut
    x1, y1, l1, h1 = bbox1.getValues()
    x2, y2, l2, h2 = bbox2.getValues()
    si l1/h1 > l2/h2 alors
       ratio = l2/l1
    sinon
       ratio = h2/h1
    xc = (x-x1)*ratio + x2
    yc = (y-y1)*ratio + y2
    Retourne (xc, yc)
Fin
```

FIGURE 19 – Algorithme de changement de repère

```
Fonction: "ecrirePDF"
Argument
    documentGeographiques: La liste des objets geographiques a representer.
    methodeDeColoration: La methode de coloration des objets geographiques
                (par exemple par des quartiles)
    nomFichPDF: le nom du fichier PDF a generer
Retourne
    rien
Debut
    bbox1 = BoundingBox()
    PourChaque objGeographique dans lstObjGeographiques:
        bbox = objGeographique.getBoundingBox()
        bbox1.ajouterBBox(bbox)
    bbox2 = BoundingBox()
    bbox2.ajouterPoint(0,0)
    bbox2.ajouterPoint(21cm, 29.7cm)
    transformeur = Transformer(bbox1, bbox2)
    myCanvas = Canvas(nomFich,pagesize=(21cm, 29.7cm))
    PourChaque objGeographique de documentGeographique:
        objGeographique.toPDF(myCanvas, methodeDeColoration, transformeur)
    myCanvas.save()
Fin
```

FIGURE 20 – Algorithme de dessin d'une carte en PDF