François Espinet Clément Lestrelin

Promotion X2011

INF 431

Carte du ciel

Sujet de Jean-Christophe Filliâtre

École Polytechnique ParisTech

Table des matières

1 Descriptif du projet

Le but de ce projet est de construire, à partir d'informations sur les étoiles, une carte du ciel. Cette dernière prendra en compte la date, l'heure, et la position de l'observateur. Partant d'une base de données textuelles, nous convertissons pas à pas les coordonnées jusqu'à reconstituer virtuellement la sphère céleste. Elle est ensuite projetée sur un disque que l'utilisateur pourra regarder et analyser. Ce dernier aura la possibilité de modifier à loisir les paramètres exposés précédemment. Notre seul point de départ est le fichier `stars.dat` [1]. Celui-ci répertorie des informations concernant les 3140 étoiles les plus visibles depuis la Terre.

Il nous semble important de préciser dès à présent quelques conventions que nous nous sommes fixées, et que nous avons suivies tout au long du projet :

- Les nombres sont représentés soit par des champs de type `int`, soit par des champs de type `double`. Bien qu'il ne soit pas nécessaire de coder certains champs sur 64 bits, nous avons voulu instaurer une uniformité pour tous les flottants afin éviter les opérations de cast et les éventuelles pertes de données. De plus, les fonctions mathématiques Java de la classe `Math` renvoient, pour la plupart, des objets de type `double`.
- Tous les angles sont exprimés en radians. Les méthodes de la classe `Math` fonctionnent avec des angles en radian. C'est pourquoi nous avons choisi cette unité. Imposer une unicité de l'unité angulaire nous permet, comme dans le cas précédent, d'évier les opérations de conversion, ou le calcul biaisé par les fonctions trigonométriques, qui sont des erreurs difficiles à repérer. Seuls les angles de longitude et latitude proposés à l'utilisateur sont en degrés (décimaux).

1. Fichier fourni avec le projet

1.1 Architecture

Le langage de programmation utilisé est Java 7. Notre projet se décompose en une vingtaine de classes, elles-mêmes regroupées en packages. L'arborescence est détaillée dans la figure 1.

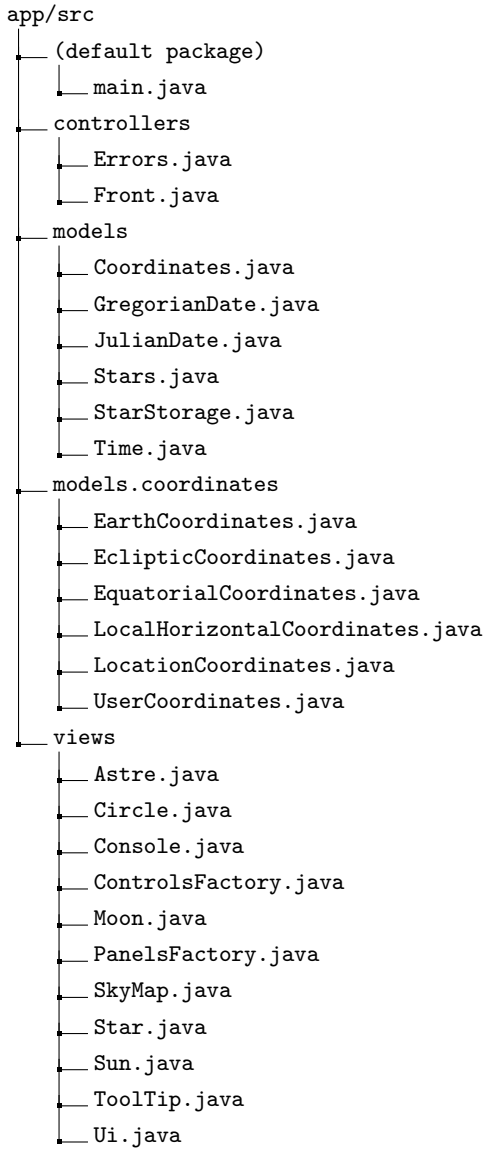

FIGURE 1 – Arborescence du projet

Voici une brève description des packages :

`(default package)` ne contient que le fichier `main.java` qui est exécutée au lancement du programme.

`controllers` contient les classes qui gèrent le programme. C'est à dire qui commande le dessin des graphismes, qui répond aux demandes de l'utilisateur...

`models` contient les classes qui modélisent les objets mathématiques : les dates et heures dans le calendrier grégorien et julien, l'ensemble des étoiles, mais également celles qui contiennent les méthodes capables de convertir ces objets.

`models.coordinates` contient les classes représentant les différents types de coordonnées.

`views` contient toutes les classes nécessaires à l'affichage, ainsi que les étoiles et les astres spécifiques : la lune et le soleil.

Des informations plus détaillées peuvent être trouvées dans la JavaDoc du projet (/docs/javadoc) ;

1.2 Représentation de la date et l'heure

Le temps est représenté à l'aide de trois classes :

`GregorianDate` représente une date et une heure – en fraction de jours – dans le calendrier grégorien. La présence de cet objet est essentielle car il s'agit du calendrier proposé à l'utilisateur afin de choisir ses paramètres.

`JulianDate` représente une date – en jours juliens – et une heure – en fraction de jour – dans le calendrier julien. Ce calendrier est très utilisé en astronomie. Il permet de calculer la position des astres. Au delà du simple stockage de données, cette méthode permet le calcul d'heures et d'angles variant en fonction du temps, et nécessaires à la conversion de coordonnées. Les méthodes

```
public double getCentury()
public double getDecimalCentury()
```

```
public double obliquityEcliptic()
public double siderealTime()
```

renvoient respectivement la date en siècles juliens à 0h UT, notée T ; la date en siècles juliens à la date et l'heure contenues dans l'objet `JulianDate` ; l'obliquité de l'écliptique, notée ϵ ; et le temps sidéral à Greenwich, noté Θ_0.

`Time` fait le lien entre les deux classes précédentes. Elle ne contient que deux méthodes

```
private static double ent(double x)
public static JulianDate gregorianToJulian(GregorianDate d)
```

La première calcule la "fausse" partie entière d'un `double x`. La seconde renvoie, à partir d'une instance de la classe `GregorianDate`, une instance de `JulianDate`. Autrement dit, elle convertit une date du calendrier grégorien en une date du calendrier julien. Aucune méthode n'est prévue pour effectuer la conversion inverse, car celle-ci n'est pas utile dans le projet. Dans le programme, l'heure sélectionnée par l'utilisateur est convertie pour effectuer les calculs, et non l'inverse.

1.3 Représentation des coordonnées

Les coordonnées des astres peuvent être exprimées dans trois systèmes différents : local horizontal, équatorial, et écliptique. Afin de réaliser au mieux les transformations, nous avons choisi de représenter ces coordonnées par trois classes différentes :

`EclipticCoordinates` n'est utilisée que pour le stockage du résultat du calcul des coordonnées du soleil et de la lune. À la différence des autres astres, leurs coordonnées sont calculées à partir de formules spécifiques, qui les fournissent dans le système de coordonnées écliptiques.

`EquatorialCoordinates` permet le stockage en mémoire des coordonnées des étoiles fournies dans le fichier `stars.dat`. Le grand avantage de ce système de coordonnées est le fait que la position des étoiles y est constante.

`LocalHorizontalCoordinates` permet enfin de représenter une carte du ciel. Ces coordonnées tiennent compte de la position de l'observateur sur terre, de la date et de l'heure. Elles sont donc parfaitement adaptées au repérage d'un point sur la sphère céleste.

Ces trois classes, issues du même package `models.coordinates` sont assez similaires si on se place du point de vue de l'objet Java. Elles contiennent toutes deux champs privés de type `double`, représentant deux angles. Les méthodes, toutes dynamiques, ne sont que les constructeurs, getters et setters. La classe `LocalHorizontalCoordinates` se différencie avec la présence d'une méthode supplémentaire `isVisible()`. Elle permet la différenciations des étoiles visibles ou non, i.e. celles situées ou non sous l'horizon.

Suite à la lecture de ce paragraphe, on mesure toute la nécessité de méthodes de transformation entre les différents systèmes de coordonnées. Le sujet n'en fournit que deux :

$$\begin{aligned} \text{Coordonnées écliptiques} &\longrightarrow \text{Coordonnées équatoriales} \\ \text{Coordonnées équatoriales} &\longrightarrow \text{Coordonnées locales horizontales} \end{aligned}$$

Aucune autre n'est nécessaire. Celles-ci sont regroupées dans la classe `Coordinates` sous forme de 5 méthodes statiques. Une sixième est présente. Il s'agit de la "vraie" fonction arc tangente, indispensable à l'obtention du signe du résultat. Ces méthodes sont les suivantes :

```
public static LocalHorizontalCoordinates equatorialToLocalHorizontal(↩
    EquatorialCoordinates c, UserCoordinates uc)

private static LocalHorizontalCoordinates equatorialToLocalHorizontal(double ascention,↩
     double declinaison, double Theta, double longitude, double latitude)

public static EquatorialCoordinates eclipticToEquatorial(EclipticCoordinates c, ↩
    UserCoordinates uc)

private static EquatorialCoordinates eclipticToEquatorial(double latitude, double ↩
    longitude, double epsilon)
```

```
public static LocalHorizontalCoordinates eclipticToLocalHorizontal(EclipticCoordinates ↩
    c, UserCoordinates uc)

private static double atan(double y, double x)
```

On remarque la présence de méthodes publiques et privées. Afin d'éviter toute erreur de type inversion des deux coordonnées, les méthodes publiques ne prennent en argument que des objets du package `models.coordinates`. Ces méthodes font ensuite appel à leur homonyme privée en leur fournissant les champs des coordonnées en argument.

1.4 La classe `UserCoordinates`

La classe `UserCoordinates` se différencie des trois autres classes du package `models.coordinates`. Elle représente un observateur à la surface de la Terre. Elle comprend des champs `double latitude` et `double longitude` représentant les coordonnées géographiques. Mais aussi un champ `JulianDate`. Enfin, pour éviter les calculs répétés et inutiles du temps sidéral à Greenwich et de l'obliquité de l'écliptique, deux champs `double Theta` et `double epsilon` stockent respectivement ces deux valeurs.

1.5 Représentation des astres

Tous les astres héritent de la classe abstraite `Astre`. Celle-ci contient principalement des méthodes et des champs servant au dessin. Trois classes héritent de celle-ci :

`Star` représente les étoiles du fichier `stars.dat`. Le fichier est parsé et toutes les informations disponibles sont stockées dans les différents champs de la classe. La méthode

```
public static Color color(double magnitude, String spectralType)
public void draw(Graphics g, UserCoordinates userCoordinates, Circle mapBounds)
```

permettent respectivement de calculer la couleur d'une étoile à partir de son type spectral et de sa magnitude, et de la dessiner à partir des coordonnées de l'observateur et du cercle représentant le ciel.

`Sun` et `Moon` permettent le calcul des coordonnées écliptiques à partir de la date et l'heure exprimées en siècles juliens, à partir des formules fournies dans le sujet.

Les instances de la classe `Star` sont créées, importées et stockées grâce aux deux méthodes

```
public void importFromFile(final String filename)
public StarStorage getStars()
```

de la classe `Stars`. La première parse le fichier `stars.dat` et stocke les instances de `Star` dans une instance de la classe `StarStorage`. La seconde méthode renvoie cette dernière.

La classe `StarStorage` n'est en fait qu'une simple liste. Le nombre d'étoiles [2] étant relativement peu important, nous pouvons nous permettre de transformer les coordonnées de toutes et de déterminer, suite à ce calcul, si elles sont visibles ou non. Un nombre plus important d'étoiles nous imposerait l'implémentation d'une structure de données plus complexe, de laquelle on ne retirerait que les étoiles potentiellement visibles depuis la position de l'observateur.

1.6 Interface graphique

Pour construite notre interface graphique, nous avons utilisé le GUI [3] WindowBuilder. Celui-ci nous a permis de créer rapidement et efficacement le GUI de notre programme, à l'aide d'une interface graphique intuitive, semblable à cette proposée pour le développement d'applications Android. Celui-ci repose sur l'utilisation de 4 classes :

2. les 3140 du fichier `Stars.dat`
3. Graphical User Interface

`Ui` contient toutes les informations sur le fenêtre principale telles que : le nombre d'onglets, la position des boutons et des spinners, la position des cartes... Cette classe a été générée automatiquement à partir du plug-in WindowBuilder.

`ControlsFactory` contient deux méthodes calibrant les spinner – valeurs limites, pas, affichage – de l'onglet "Carte du monde".

`SkyMap` contient toutes les informations nécessaires au dessin de la carte du ciel : une instance de `StarStorage`, de `Sun`, et de `Moon`, ainsi qu'un champ `mapBounds` représentant les limites de la représentation de la carte du ciel (centre et bords).

1.7 La classe `Front`

La classe `Front` permet de faire le lien entre tous les composants de l'application. En particulier elle est à l'interface entre les objets de vue (du package `views`) et les objets "modèles" (package `models`). Cette architecture est inspirée du modèle MVC [4] de programmation objet.

Par exemple, le constructeur de cette classe effectue les bindings entre les boutons de la vues (Spinner, Button, toggleButton) est les actions à effectuer. Donnons deux exemples :

- Le bouton "Mettre à jour" de l'interface qui permet de mettre à jour la carte du ciel : ce bouton est récupéré dans la classe `Front` qui lui ajoute un "listener". Ce listener consiste à appeler la méthode `updateDate` de la classe `Front` qui instancie les modèles dates correspondants et les donne à l'objet SkyMap pour qu'il dessine à nouveau les étoiles.
- Le bouton "Faire défiler" est connecté de la même façon, mais cette fois-ci, son click lance un Thread qui permet de faire défiler le ciel au dessus de l'utilisateur.

1.8 Fonctionnement global

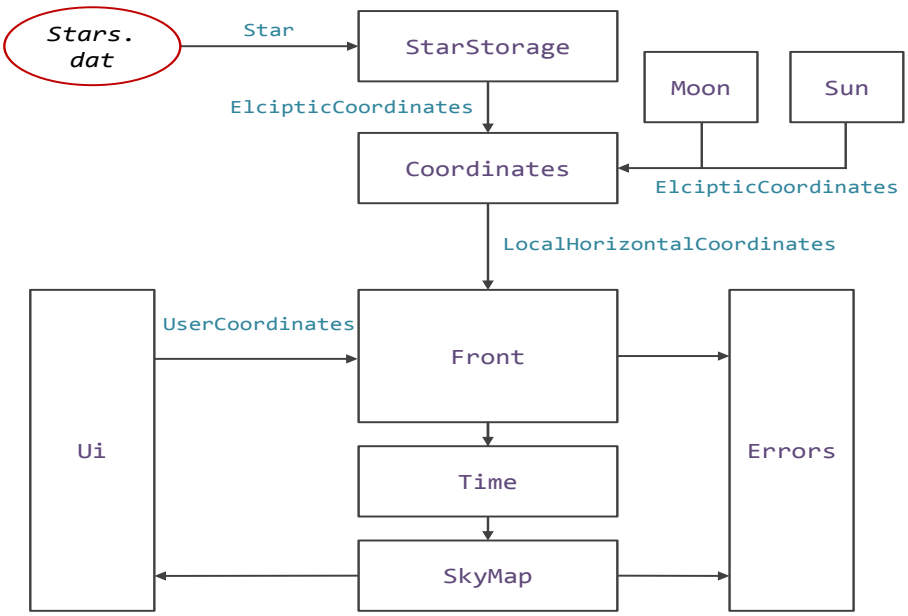

FIGURE 2 – Fonctionnement interne de l'application

4. Model Vue Controller : http ://fr.wikipedia.org/wiki/Modèle-Vue-Contrôleur

2 Problèmes rencontrés

Le défi majeur de ce projet était la bonne compréhension du sujet, des nombres, des types, des conversions d'angle, et des transformations dans les différents systèmes de coordonnées. Des soucis sont apparus quant à certains manquement aux conventions que nous nous étions fixés. Par exemple, lors de la phase de développement, le soleil a été longtemps un astre invisible. Les coordonnées de l'observateur désignaient par défaut la commune de Palaiseau. Un oubli de conversion degrés → radians nous localisait en réalité proche de pôle nord. Ce qui nous empêchait de voir celui-ci. Ce type d'erreur est difficile à détecter.

Un autre point crucial était la compréhension de la signification réelle des objets. Nous avions par exemple omis de convertir le temps sidéral à Greenwich Θ_0 en heure dans la transformation des coordonnées équatoriales en locales horizontales.

La question d'inclure ou non les heures, en fraction de jour, dans le calcul de certains temps tels que la date en siècles juliens T, nous est souvent apparue. Nous le calculions à l'origine toujours à 0h UT. Ce qui ne convient pas au calcul des coordonnées de soleil et de la lune. En effet, tous les jours à midi, la date, exprimée en siècle juliens, fait un saut de valeur. Cette discontinuité provient de la définition même du jour julien : nombre de jours et fraction de jour écoulés depuis une date conventionnelle fixée au 1er janvier -4712 à 12 heures. Si les jours juliens sont exprimés à 0h UT, tous les jours à midi, ce nombre est incrémenté d'une unité. Cette discontinuité apparaissait alors au niveau des positions de la lune et du soleil, qui faisaient un bond tous les jours à cette même heure.

3 Manuel d'utilisation

3.1 Lancement du programme

L'utilisateur pourra lancer le programme à l'aide de la commande après l'avoir compilé :

```
java -cp app/bin main
```

Au lancement du programme l'utilisateur se trouve face à une fenêtre semblable à celle de la figure 3.1. Elle se décompose en deux parties. Sur la gauche, l'utilisateur dispose de quelques contrôles, que nous détaillerons par la suite. Sur la droite, il peut consulter une carte du ciel. On remarquera la présence d'un menu Localisations qui permet de se placer sur des lieux prédéfinis.

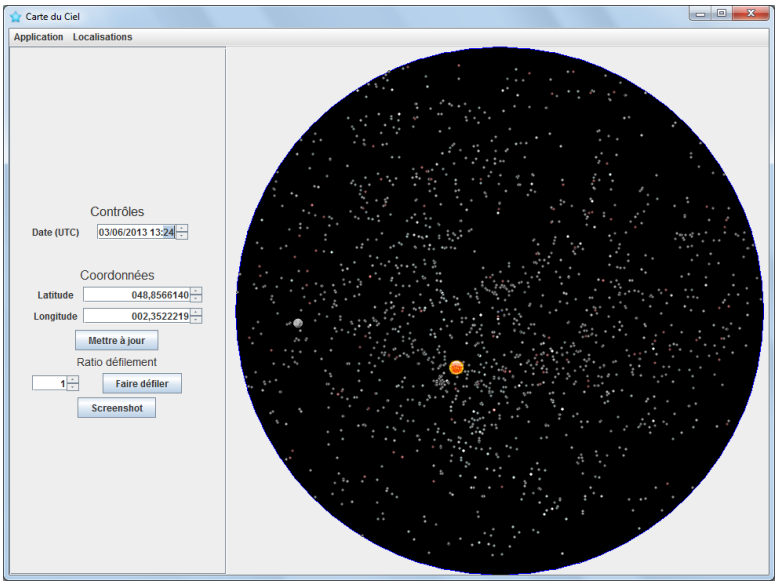

FIGURE 3 – Fenêtre au lancement

3.2 Paramètres

Du haut vers le bas, l'utilisateur dispose de :

Un spinner de sélection de la date et l'heure qui, comme son nom l'indique, permet de choisir la date et l'heure pour lesquelles la carte du ciel sera dessinée. Dès qu'une date est saisie, manuellement ou à l'aide des flèches, la carte est automatiquement mise à jour. Par défaut, ce spinner est réglé à l'heure de lancement du programme.

Deux spinners de sélection des coordonnées qui permettent de se positionner sur la surface du globe. Par défaut, les latitudes et longitudes sont celles de la commune de Palaiseau.

Le bouton Mettre à jour permet de redessiner la carte du ciel, notamment quand de nouvelles coordonnées sont saisies.

Un spinner Ratio défilement permet de sélectionner un entier définissant le ratio suivant : $\frac{\text{Temps logiciel}}{\text{Temps reel}}$

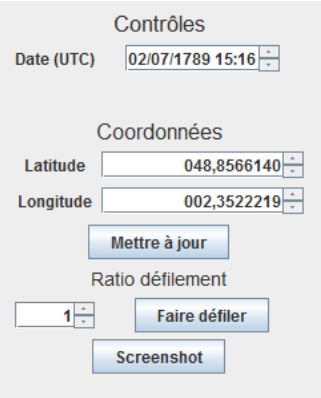

FIGURE 4 – Paramètres disponibles

Un bouton basculant Faire défiler permet d'activer ou non le défilement des étoiles en changeant la date logiciel selon le ratio défini précédemment.

Un bouton Screenshot permet d'enregistrer une capture d'écran de la carte au format PNG, dans le dossier `/screenshots`.

3.3 Carte du ciel

La carte du ciel située sur la partie gauche représente une projection de la sphère céleste, telle qu'elle peut être admirée à la date et aux coordonnées sélectionnées, sur un disque. Lorsque l'utilisateur clique sur cette carte, un tooltip [5] affiche des informations sur l'étoile la plus proche. La couleur de fond est celle de l'étoile correspondante [6]. Quatres tooltips peuvent être affichés simultanément à l'écran. Au delà, chaque nouveau clic efface le plus ancien. Lors d'un changement de date ou d'un redimensionnement de la fenêtre, ils sont tous effacés, pour ne pas conserver de tooltips décorrélés de leur étoile.

Les informations affichées sont :
- Le nom de l'étoile ;
- Ses coordonnées équatoriales, qui sont constantes ;
- Sa magnitude ;
- Son type spectral.

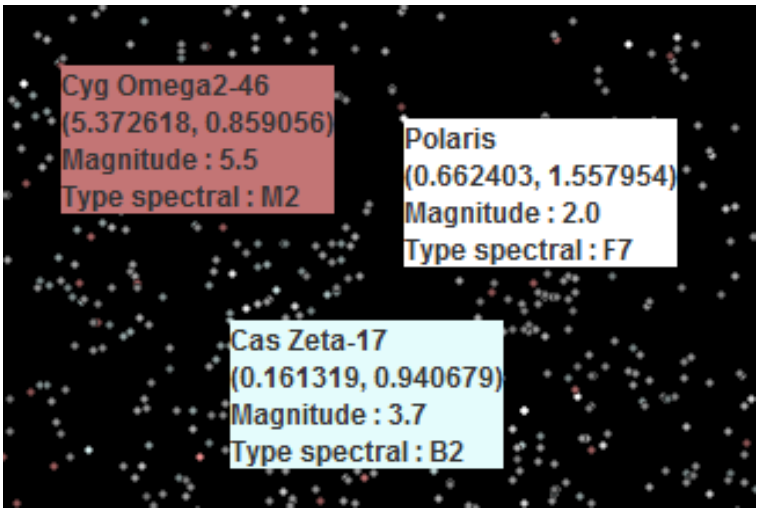

FIGURE 5 – Exemples de ToolTips

5. Info-bulle en français
6. Pour cette utilisation spécifique, elle a été éclaircie afin d'assurer une meilleure lisibilité

3.4 Menus

Deux menus sont présent sur la fenêtre principale :

Application ne contient qu'un bouton quitter, qui ferme l'application.

Localisations propose une dizaine de points remarquables sur la surface du globe : le pôle nord, l'équateur [7], ainsi que quelques villes célèbres. Étretat, petit village Normand, a été sélectionné car il constitue un emplacement idéal d'observation de l'éclipse du 11 août 1999.

4 Conclusion

Ce projet a été l'occasion pour nous d'appliquer nos connaissances à un projet complet. Nous avons construit l'application et son architecture ce que les TD nous permettent rarement de faire. Ce fut aussi l'occasion de se familiariser avec un gestionnaire de version (Mercurial) qui nous a permis un travail en équipe facile.

Nous avons procédé par étapes et améliorés notre produit au fur et à mesure. En effet, le menu localisations, le bouton Screenshot et Faire défiler sont venus en fin de développement.

Nous avions pensé à d'autres évolutions pour l'application, mais le fait que son fonctionnement était satisfaisant ne nous a pas poussé à les développer. En effet, nous avions pensé à utiliser plusieurs threads, tant pour charger les étoiles que pour exécuter les dessins afin de ne pas bloquer l'interface utilisateur, cependant nos connaissances en multithreading lors de la réalisation du projet ne nous ont pas permis de réaliser cette évolution. Comme mentionné plus haut, nous aurions pu utiliser de meilleurs moyens de stockage des étoiles et de recherche de ces dernières. Enfin, nous aurions pu rendre l'application plus paramétrable (en permettant d'éditer les Localisations prédéfinies ou le fichiers de base des étoiles par exemple).

7. point d'intersection entre l'équateur et le méridien de Greenwich