

Rapport de Projet L3 Informatique : Carte du Système Solaire

Alexandre Jean

Damyia Achour

Nathan Gallone

3 juin 2022

Remerciements

Avant tout, un merci à Jean-Michel Hufflen pour nous avoir aiguillé dans la réalisation de ce projet, pour ses conseil et sa relecture de ce rapport.

Table des matières

1	Introduction	3
2	Difficultés préliminaires	4
3	Récupération et stockage des données : API REST et JSON	5
3.1	Récupération de données : V1	6
3.2	Récupération de données : V2	6
3.3	Stockage des données : JSON	6
3.4	Stockage des données : Application	6
3.5	Transfert des données entre Ruby et Java	7
3.5.1	Organisation général du transfert de données entre les parties de l'application	7
4	Calculs des coordonnées des planètes	8
4.1	prémices de cours d'astronomie	8
4.2	Les paramètres orbitaux kepleriens	8
4.3	Les données fournies par l'API	11
4.3.1	La méthode Newton-Raphson pour l'anomalie excentrée	11
4.3.2	Calcul de l'anomalie vraie à partir de l'anomalie excentrée	13
4.4	Passage aux coordonnées cartésiennes	13
4.4.1	Distance entre le corps de référence et le satellite	13
4.4.2	Coordonnées cartésiennes dans le plan de rotation du satellite	13
4.4.3	Rotation dans le repère de référence	14
5	Une application graphique avec Java SWING	15
5.1	Java Swing et Java2D	15
5.2	Modèle MVC	16
5.2.1	Le modèle	17
5.2.2	La vue	17
5.2.3	Les contrôleurs	18
5.3	Calcul des échelles	18
5.3.1	Distances	18
5.3.2	Diamètres	20
5.3.3	Référence et échelle	20
5.4	Changement de référence et événements	21
6	Résultats	24
6.1	État actuel de l'application	24
6.2	pistes d'améliorations	24
6.3	Conclusions personnelles	24

Chapitre 1

Introduction

Dans le cadre du module *Projet Tuteuré* de Licence 3 en informatique, nous avons été amenés à réaliser une application de carte du système solaire. Sous la tutelle de Jean-Michel Hufflen, enseignant chercheur à l'université de Franche-Comté, nous avons codé SolarSys. L'objectif était de réaliser une carte permettant l'exploration du système solaire. Au démarrage du programme, est affichée la carte des planètes existant dans ce système : On peut alors observer les différentes planètes qui le composent à des positions bien particulières sur leur orbite et avoir une idée des différences de tailles entre tous ces astres. Ensuite, un "clic" sur une planète provoque l'affichage d'une carte à plus grande échelle montrant la planète ainsi que ses satellites, suivant le même principe que précédemment. D'autres manipulations permettent l'affichage de renseignements à propos de l'objet sélectionné : son diamètre, sa masse, sa vitesse de rotation, sa date de découverte...

Il s'agit d'un projet très complet qui fait intervenir de nombreuses notions : Des traitements sur des bases de données, de la résolution approchée d'équations différentielles, des décisions à prendre concernant le format d'affichage final de la carte, des planètes, des informations à faire figurer ou non... Tout ceci nous a permis d'explorer des aspects de l'informatique peu travaillés dans le cadre de la licence.

Ce document présente les différentes étapes par lesquelles nous sommes passés pour en arriver à une version viable d'un tel programme.

Chapitre 2

Difficultés préliminaires

Avant de nous lancer dans la programmation de l'application. Une phase de recherche a été nécessaire. N'ayant de prime abord aucune notion d'astronomie, nous avons parcouru quelques documents nous permettant de nous familiariser avec le sujet de notre projet. Ensuite, il a fallu se poser plusieurs questions.

- Quelles données utiliser ?
- Quels traitements sont nécessaires pour utiliser efficacement ces données ?
- Quels objets afficher ou non et comment choisir l'échelle¹ avec laquelle présenter les planètes et autres astres du système solaire ?

Ce sont ces mêmes questions qui ont déterminé les différents modules composant le programme final, à savoir :

- Un script Ruby de récupération de données "officielles" depuis une API
- Un script Ruby permettant la conversion des coordonnées de l'API, alors sous forme keplériennes, en coordonnées cartésiennes bien plus pratiques pour l'affichage.
- un modèle MVC² en Java pour la représentation graphique des différents astres qui constituent notre système solaire.

La suite de ce rapport décrit donc chacun de ces modules séparément pour donner une idée la plus précise possible des différents problèmes rencontrés et solutions y étant apportées.

1. Dans les faits, il serait plus pertinent de parler d'*échelles* au pluriel. Mais nous aborderons cette question en partie 5.3

2. Model View Controller, un patron de conception de logiciel répandu pour produire des applications graphiques

Chapitre 3

Récupération et stockage des données : API REST et JSON

L'acquisition des données était la première étape du projet, nous avons décidé d'utiliser une API afin d'automatiser la récupération des données concernant les corps célestes mais aussi afin d'avoir des données officielles et vérifiées. Cette API est [Le-système-solaire.net](#)[2].

Les données suivante sont fournies :

- name :
Le nom français du corps céleste.
- englishName :
Le nom anglais du corps céleste.
- isPlanet :
Attribut booléen indiquant si le corps céleste est une planète
- moons :
La liste des satellites avec le format suivant :
 - moon : Le nom du satellite
 - rel : Un lien vers le chemin de l'API pour récupérer les données du satellite.
- Mesures du corps céleste :
 - masse
 - volume
 - ...
- DiscoveredBy :
Nom de la personne ayant découvert le corps céleste.
- DiscoveryDate :
Date de découverte du corps céleste format : jj/mm/aaaa.
- avgTemp :
Température moyenne à la surface du corps céleste.
- rel :
Lien vers le chemin permettant d'obtenir les données sur l'API.

Pour les Lunes :

- aroundPlanet contient :
 - planet : Nom de la planète en français
 - rel : Lien vers le chemin permettant d'obtenir les données sur l'API pour la planète.

1. La liste complète est plus longue mais non décrite ici car expliquée plus loin ou simplement ignoré dans l'application

3.1 Récupération de données : V1

Une fois décidé sur l'API utilisée, la première étape fut d'écrire un script afin de pouvoir récupérer les données utiles à l'application. Le script a été écrit en Ruby et permet d'automatiser la récupération des données en envoyant automatiquement les requêtes pour chaque planète et chaque lune. En effet cette tâche serait beaucoup trop longue à effectuer à la main de par le grand nombre de satellites de certaines planètes.

Le premier script de récupération des données utilisait des chemins de l'API permettant de récupérer les données spécifique à un corps céleste en l'incluant dans le chemin par exemple :

```
https://api.le-système-solaire.net/rest/bodies/jupiter
```

L'avantage de cette méthode vient de la facilité à récupérer les données liées aux satellites des planètes en récupérant la liste des satellites directement depuis le résultat de chaque requête pour les planètes. L'inconvénient de cette méthode est lié au nombres de requêtes nécessaire pour récupérer toute les données. En effet, il faut faire une requête par planète mais aussi une requête par satellite ce qui peut devenir coûteux pour certaines planètes tel que Saturne avec 82 satellites et donc 83 requêtes. Un deuxième problème relevé pendant la création du script de récupération de données vient de certaines requête de l'API renvoyant un fichier JSON invalide et obligeant des vérifications d'erreurs non nécessaire dans la deuxième version.

Cette version a été utilisé pendant le début de la création de l'application graphique puis réécrite dans une version deux utilisant un autre chemin de l'API afin de récupérer les données.

3.2 Récupération de données : V2

Le deuxième script de récupération des données, écrit plus tard utilise un chemin de l'API permettant de récupérer toute les données de l'API en une seule fois. Les données n'étant pas organisée une fois récupérée, le script s'occupe de grouper chaque planète et ses satellites dans une structure de données avant d'écrire les données en JSON.

Le deuxième script devait également être compatible avec JRuby pour pouvoir être intégré plus facilement à l'application mais JRuby possède encore aujourd'hui un bug avec les connexions SSL² et cette deuxième version du script désactive donc les vérifications SSL sur les requêtes envoyées.

3.3 Stockage des données : JSON

Les données récupérées ont été stockée dans le format JSON car l'API rentrait les données dans ce format mais aussi car ce format est facilement manipulable en Ruby. Les données sont stockées selon la structure suivante :

- bodies : dossier contenant tout les dossiers des planètes.
 - NomPlanete : Dossier contenant le fichier JSON relatif à la planète et le dossier moons.
 - moons : dossier contenant les fichiers JSON relatif aux lunes et satellites de la planète.
- La sauvegarde des données en tant que fichier procède à quelque réécritures pour le noms de certaines lunes et satellites dont le nom contient des espaces, remplacé par des _ et des \ remplacée par _BS_ afin d'éviter les problèmes de nom de fichiers invalide à l'enregistrement des fichiers. Un exemple d'un tel satellite est S\2004 S 4 l'une des nombreuses lunes de Saturne.

3.4 Stockage des données : Application

Le format de stockage des données interne à l'application a connu deux version : Un premier format stockant les objets dans la classe `CelestialBody`³ avec une `referenceFrame` étant elle même un `CelestialBody` et une liste de satellites contenant des `CelestialBody`. Ce format de stockage a été conservée pendant une grande partie du développement de part sa simplicité d'utilisation et de

2. SSL est un protocole de chiffrement des données à l'envoi de requêtes

3. Classe Ruby permettant le stockage des données JSON

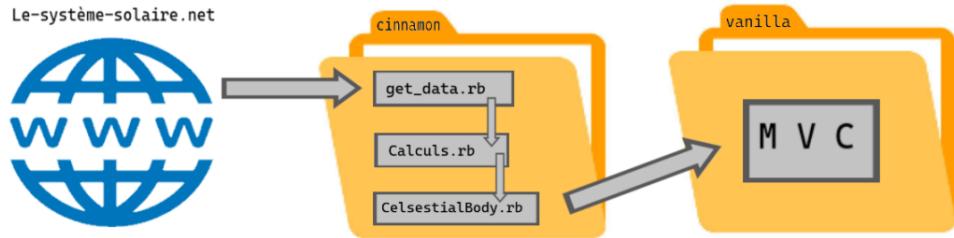


FIGURE 3.1 – Architecture global du transfert des données.

compréhension. Le problème de ce format est lié à l'affichage des planètes. En effet, l'affichage des informations liés au planètes déclenchaient des erreurs de `StackOverflow`. Ces erreurs étaient liées au fait que le stockage des données dans ce premier format était récursif (Chaque planète faisait référence au soleil qui faisait référence aux planètes) faisant boucler l'affichage. Ce problème a été résolu en changeant les références aux planètes par des chaînes de caractères désignant le nom de la planète et par une réécriture de l'algorithme de récupération des données.

3.5 Transfert des données entre Ruby et Java

La dernière étape de récupération des données consiste à transformer les données récupérées en Ruby en données Java exploitable par l'application graphique. Les données sont simplement transmises à une version Java de la classe `CelestialBody` à l'aide de la méthode `toJava` de la classe `CelestialBody` Ruby. Cette méthode initialise tous les attributs de la classe Java à partir de la classe Ruby et convertit également les positions polaires en positions cartésiennes comme expliqué dans la prochaine partie.

3.5.1 Organisation général du transfert de données entre les parties de l'application

Le transfert des données entre les différentes parties de l'application suit le schéma de la figure 3.1. En effet, les données sont récupérées depuis l'API par le fichier `get_data.rb`, puis envoyées à `Calculs.rb` afin de transformer les coordonnées obtenues depuis l'API et de calculer les coordonnées réelles des objets célestes. Ces données sont ensuite transmises à `CelestialBody.rb` afin d'être transmises à l'application Java (Le dossier `vanilla` sur le schéma) et afin de pouvoir être affichées.

Chapitre 4

Calculs des coordonnées des planètes

Une fois les données acquises, il convenait de les traiter. L'API fournit des coordonnées au format *keplerien*¹. Il convenait donc de convertir les coordonnées kepleriennes en coordonnées cartésiennes², bien plus confortable au développement de notre carte.

4.1 prémices de cours d'astronomie

Étant donné que nos seules notions d'astronomie remontaient au Lycée, nous nous sommes quelque peu trouvés noyé sous la grande quantité d'informations que fournissait l'API. Nous avons donc cherché des documents pour nous familiariser avec ces notions complexes. Dans un premier temps avec des vidéos Youtube de la chaîne OrbitNerds pour mieux comprendre comment interpréter les coordonnées kepleriennes[3]. Cette chaîne présente de manière claire quelques uns des concepts des bases à l'utilisation des paramètres orbitaux des planètes que nous avons récupéré. Cependant, cette ressource s'est avérée insuffisante pour effectuer les opérations dont nous avions besoin, à savoir convertir les coordonnées du format keplerien à cartesien. Pour comprendre comment ceci fonctionne, nous avons eu recours aux Memorandum de Rene Schwarz [4].

4.2 Les paramètres orbitaux kepleriens

Avant d'entrer dans les détails, il convient de préciser un point particulier. Les coordonnées kepleriennes sont principalement constituées d'angles. Un angle étant la donnée de 3 points, nous devrions spécifier trois points de l'espace A , B et C pour chaque angle \widehat{ABC} dont nous parlerons. Sachant que la plupart du temps, notre point B pour les angles en questions sera l'objet céleste autour duquel nous décrivons une orbite, nous omettrons fréquemment de le mentionner et parlerons simplement de l'angle entre A et C . Cela dit, revenons à nos paramètres orbitaux. Pour donner une idée de la position des astres les uns par rapport aux autres sur leurs orbites elliptiques, un set de coordonnées kepleriennes comprend 6 paramètres :

— a : la longueur du demi grand axe de l'ellipse.

Le demi grand axe est en quelque sorte le *rayon le plus long* de l'ellipse (voir figure 4.1).

— e : l'excentricité de l'ellipse.

Il s'agit de d'un paramètre décrivant à quel point l'ellipse est *aplatie*. Quand $e = 0$, c'est un cercle et tant que $e < 1$, c'est une ellipse telle qu'on en a l'habitude (voir figure 4.2).

— i : l'inclinaison de l'ellipse.

Il s'agit de l'angle formé par le plan dans lequel s'inscrit notre ellipse et un autre plan de référence. Par exemple, en ce qui concerne les plans de références, celui utilisé pour la Terre est le

1. Johannes Kepler (1571-1630) est un astronome allemand à l'origine des lois portant son nom. Ces dernières affirment que les astres en orbite autour du soleil ont une trajectoire *elliptique* et non parfaitement *circulaires*.

2. Du nom de René Descartes (1596-1650), philosophe, mathématicien et physicien français.



FIGURE 4.1 – Représentation du demi grand axe d'un satellite en orbite autour de la Terre.

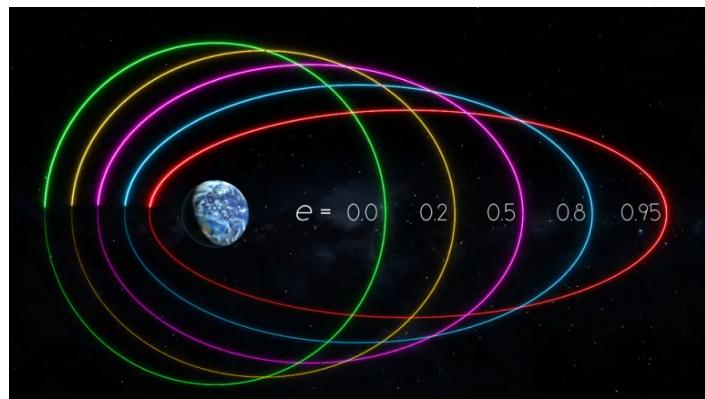


FIGURE 4.2 – Représentations des trajectoires de satellites avec différentes excentricités.

plan équatorial et celui utilisé pour le système solaire est le plan de la trajectoire terrestre.(voir figure 4.3)

- Ω : la longitude du noeud ascendant.
C'est l'angle à partir duquel le satellite entame la partie de sa trajectoire se trouvant au dessus du plan de référence. (voir figure 4.4)
- ω : l'argument de périhélie.
Angle formé entre le noeud ascendant et la périhélie. (voir figure 4.5)
- M_0 : l'anomalie vraie.
Correspond à l'angle entre la périhélie et la position courante du satellite. (voir figure 4.6)

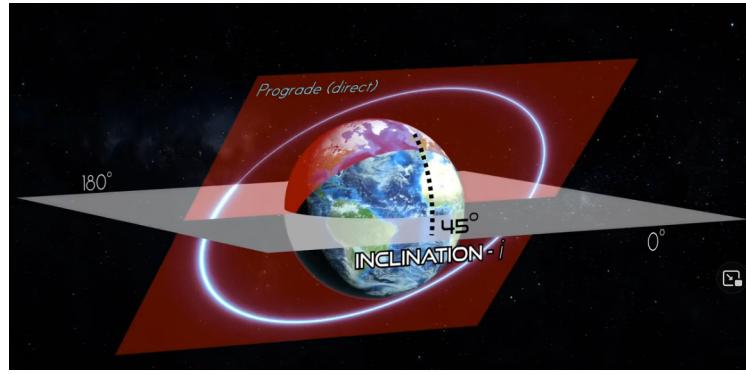


FIGURE 4.3 – Représentation de l'inclinaison entre l'orbite d'un satellite et le plan équatorial terrestre.

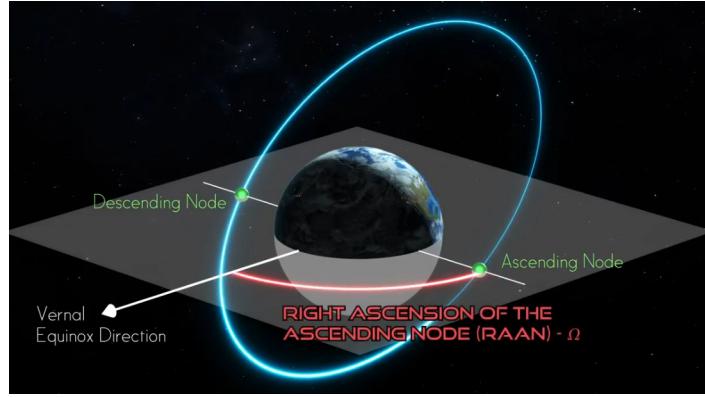


FIGURE 4.4 – Angle de noeud ascendant d'un satellite (en rouge).

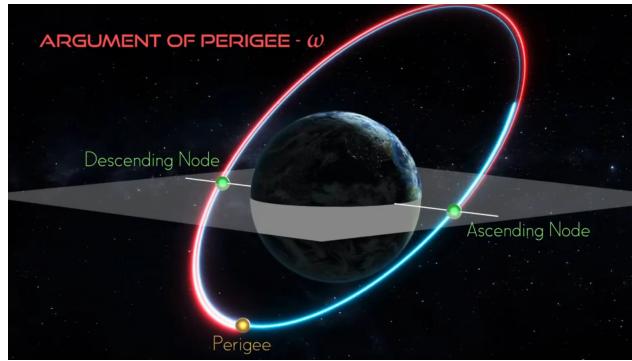


FIGURE 4.5 – Argument de périhélie d'un satellite (en rouge).

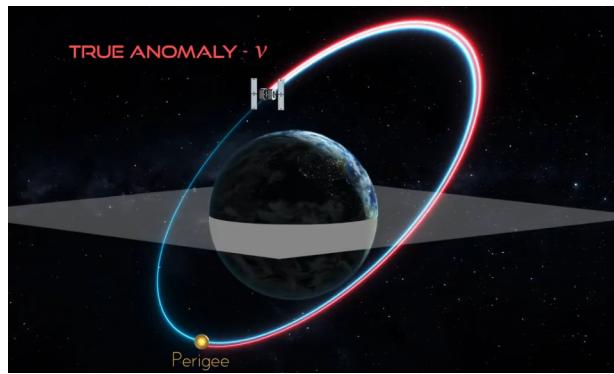


FIGURE 4.6 – Anomalie vraie d'un satellite à orbite circulaire (en rouge).

4.3 Les données fournies par l'API

Parmi toutes ces données requises pour calculer la position d'un objet, l'API nous fournit a le demi grand axe, e l'excentricité, i l'inclinaison, Ω la longitude du noeud ascendant et ω l'argument de périhélie. Pas d'anomalie vraie M_0 à priori. Il nous a donc fallu l'obtenir par d'autres moyens.

Ce que nous laisse savoir l'API, en revanche, est l'anomalie moyenne. Ce paramètre de même grandeur (il s'agit toujours d'un angle) pourrait être décrit comme l'anomalie vraie du satellite si l'on rapportait sa trajectoire elliptique à une trajectoire circulaire qu'il mettrait autant de temps à parcourir. En des termes plus techniques, l'anomalie moyenne M correspond au mouvement sur une orbite circulaire de même période que l'orbite elliptique. Sachant que la vitesse d'un satellite est plus élevée à la périhélie qu'à l'aphélie, l'anomalie vraie croît tantôt plus vite, tantôt plus lentement que l'anomalie moyenne dont la croissance est constante.

Passer directement de l'anomalie moyenne à l'anomalie vraie n'est pas chose aisée et on utilise souvent l'artifice de l'anomalie excentrée E . Cette donnée intermédiaire correspond à l'angle du projeté orthogonal de notre satellite selon le grand axe de l'ellipse sur le cercle circonscrit à notre orbite. Voilà une phrase bien compliquée qui se représente mieux par la figure 4.7. Sachant qu'il existe un lien entre

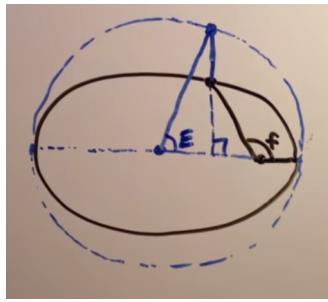


FIGURE 4.7 – Passage de la vraie anomalie f à l'anomalie excentrée E .

l'anomalie moyenne et l'anomalie excentrée. Le travail était d'implémenter la résolution des équations nous donnant l'anomalie vraie, M_0 , en fonction de l'anomalie excentrée, E , en fonction de l'anomalie moyenne, M .

Ces angles étant relatifs au temps de rotation écoulé depuis un instant t_0 , nous les représenterons par des fonctions pour s'en rappeler. Les équations de Kepler finissent par nous donner la relation suivante pour l'anomalie excentrée $E(t)$ en fonction de l'anomalie moyenne $M(t)$:

$$M(t) = E(t) - e \sin E(t)$$

Comme il s'agit d'une équation différentielle non-linéaire, impossible d'utiliser les méthodes de calculs classiques pour trouver la fonction E . Nous utilisons donc la méthode de Newton-Raphson afin de calculer la valeur de $E(t)$. Là encore, nous avons dû apprendre comment fonctionnait un tel algorithme dont l'usage est plus courant en physique ou en mathématiques qu'en informatique, et ce grâce à un cours de mathématiques de la *University of British Columbia* dispensé par Richard Antee [1].

4.3.1 La méthode Newton-Raphson pour l'anomalie excentrée

Soit une fonction $f : \mathbb{R} \rightarrow \mathbb{R}$ dérivable sur \mathbb{R} . Si l'on cherche à trouver la valeur approchée de l'une de ses racines, il est possible d'employer la méthode de Newton-Raphson³ qui consiste, à partir d'une première estimation, à calculer une valeur plus proche encore de la solution de l'équation $f(x) = 0$. On s'appuie sur des calculs de dérivées successifs. Soit r une racine de f , soit x_0 une estimation assez proche de r . On peut alors écrire $r = x_0 + h$. Tout le problème est de trouver la valeur de h qui est donc l'écart entre x_0 et la racine. Nous avons alors la relation :

$$0 = f(r) = f(x_0 + h) \approx f(x_0) + h \times f'(x_0)$$

³. Isaac Newton (1623-1747) est un physicien anglais à l'origine des fondements de la mécanique classique. Joseph Raphson (vers 1648 - vers 1715) est un mathématicien anglais. La méthode porte le nom de chacun bien qu'ils n'aient jamais collaboré. Chacun aurait développé cette méthode de calcul indépendamment de l'autre, plus ou moins à la même période.

Ainsi, à moins que $f'(x)$ ne soit proche de 0, on a donc :

$$h \approx -\frac{f(x_0)}{f'(x_0)}$$

Ce qui nous permet d'écrire :

$$r \approx x_0 - \frac{f(x_0)}{f'(x_0)}$$

Si l'on pose :

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

on peut alors répéter ce processus afin de calculer une valeur plus rapprochée encore de notre racine. Cette méthode possède deux principaux avantages. Le premier est sa facilité à mettre en place : elle nécessite peu de calculs et peut donc s'exécuter très rapidement par un ordinateur. L'autre point fort de la méthode de Newton-Raphson est qu'elle produit soit une valeur très proche du résultat escompté, soit une valeur très éloignée. Il est donc simple de discerner quand les calculs s'éloignent du résultat et de recommencer l'algorithme avec une meilleure première estimation x_0 .

Dans le cas de notre calcul de l'anomalie excentrée $E(t)$ à partir de l'anomalie moyenne $M(t)$ et de l'excentricité e , on prend pour fonction f :

$$\begin{aligned} f &: \mathbb{R} \rightarrow \mathbb{R} \\ E &\mapsto E - e \sin E - M \end{aligned}$$

Pour ce qui concerne la première estimation x_0 , la littérature indique qu'il est cohérent d'estimer x_0 à $M(t)$ si l'excentricité est faible (< 0.8) sinon à π . Enfin, il ne nous reste plus qu'à décider arbitrairement d'une précision pour le calcul et d'un nombre maximum d'itérations pour s'assurer que l'algorithme termine. C'est ainsi que l'on en vient à la fonction Ruby décrite par la figure 4.8.

```
def eccentricAnomaly(meanAnomaly,eccentricity)
  meanAnomaly = meanAnomaly/360.0
  meanAnomaly = 2*Math::PI*(meanAnomaly-meanAnomaly.floor)

  def F(estimatedExcentricAnomaly,meanAnomaly)
    |   estimatedExcentricAnomaly-eccentricity*Math::sin(estimatedExcentricAnomaly)-meanAnomaly
  end

  def derOff(estimatedExcentricAnomaly,eccentricity)
    |   1-eccentricity*Math::cos(estimatedExcentricAnomaly)
  end

  if (eccentricity < 0.8)
    |   guess=meanAnomaly
  else
    |   guess=Math::PI
  end
  precision =  0.0000001

  # Methode de Newton-Raphson pour calculer les coords courantes des planètes
  max_iteration = 30
  current_iteration = 0
  x = guess
  f = F(x,meanAnomaly)
  while(f.abs() > precision && current_iteration < max_iteration)
    |   x = x - (f / derOff(x,eccentricity))
    |   f = F(x,meanAnomaly)
  end
  x
end
```

FIGURE 4.8 – Algorithme Ruby de calcul de l'anomalie excentrée E

4.3.2 Calcul de l'anomalie vraie à partir de l'anomalie excentrée

L'anomalie vraie M_0 se calcule ensuite par la relation :

$$M_0(t) = 2 \times \arctan 2(\sqrt{1+e} \sin \frac{E(t)}{2}, \sqrt{1-e} \cos \frac{E(t)}{2})$$

Où $\arctan 2$ est la fonction arc tangente à deux arguments. On obtient ainsi le dernier paramètre nécessaire pour compléter les coordonnées keplériennes décrivant l'orbite de nos objets.

4.4 Passage aux coordonnées cartésiennes

Ces coordonnées keplériennes sont très pratiques pour la communication de positions d'un satellite. On peut⁴ mentalement se représenter les angles et faire tourner les objets pour savoir où en est un objet dans sa trajectoire. Cependant, elles ne sont pas très adaptées à une représentation en 2D pour une carte. Il nous a donc fallu les convertir. Il existe heureusement des relations pour passer du 6-uplet des coordonnées keplériennes à un 6-uplet de coordonnées cartésiennes :

$$\begin{bmatrix} a \\ e \\ i \\ \Omega \\ \omega \\ M_0 \end{bmatrix} \iff \begin{bmatrix} x \\ y \\ z \\ x' \\ y' \\ z' \end{bmatrix}$$

où (x, y, z) correspond à la position d'un objet dans l'espace et (x', y', z') sa vitesse dans chacune des trois dimensions.

Cette opération de conversion s'effectue en 3 temps :

1. calcul de la distance $d(t)$ entre le corps de référence et le satellite,
2. calcul des coordonnées cartésiennes dans le plan de rotation du satellite,
3. rotation de ces dernières coordonnées dans le repère d'origine en prenant en compte Ω , ω et i .

4.4.1 Distance entre le corps de référence et le satellite

Pour ce qui est de trouver la distance $d(t)$ entre un satellite et son astre de référence quand la trajectoire est circulaire, il ne s'agit pas d'une difficulté car, par définition, tout point d'un cercle se trouve à égale distance du centre. Donc pour un cercle de rayon a le demi grand axe de la trajectoire elliptique associée, on aurait tout simplement la relation :

$$d(t) = a$$

On se doute bien que, pour ce qui est d'une distance dans une trajectoire elliptique, ce n'est pas aussi aisément de déterminer la distance. Cependant, l'analogie avec le cercle de rayon a n'est pas si absurde (on peut se rappeler la figure 4.7) et finalement, il ne manque à la relation précédente que de prendre en compte l'anomalie excentrée $E(t)$ et l'excentricité e :

$$d(t) = a \times (1 - e \cos(E(t)))$$

4.4.2 Coordonnées cartésiennes dans le plan de rotation du satellite

Une fois la distance $d(t)$ proprement déterminée, et sachant que l'angle formé par la périhélie et le satellite (l'anomalie vraie M_0 donc), si l'on souhaite placer le satellite dans son plan de rotation, nous suffit de le situer sur le cercle de rayon 1 avec un angle M_0 puis de multiplier ses coordonnées par la distance $d(t)$:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = d(t) \times \begin{bmatrix} \cos(M_0) \\ \sin(M_0) \\ 0 \end{bmatrix}$$

4. avec un peu de gymnastique tout de même

En ce qui concerne la vitesse. Il faut prendre en compte des paramètres supplémentaires notamment les constantes gravitationnelles μ relatives aux masses des astres de références autour desquels tournent les satellites (voir figure 4.9), mais aussi, à nouveau, l'excentricité e , l'anomalie Excentrée $E(t)$ et le demi grand axe a :

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \frac{\sqrt{\mu a}}{d(t)} \times \begin{bmatrix} -\sin(E(t)) \\ \sqrt{1-e^2} \cos(E(t)) \\ 0 \end{bmatrix}$$

$$\begin{aligned} \mu_{Soleil} &: 1.3271 \times 10^{20} \\ \mu_{Mercure} &: 2.2032 \times 10^{13} \\ \mu_{Venus} &: 3.24859 \times 10^{14} \\ \mu_{Terre} &: 3.98604 \times 10^{14} \\ \mu_{Mars} &: 4.28284 \times 10^{13} \\ \mu_{Jupiter} &: 1.26687 \times 10^{17} \\ \mu_{Saturne} &: 3.79312 \times 10^{16} \\ \mu_{Uranus} &: 5.79394 \times 10^{15} \\ \mu_{Neptune} &: 6.83653 \times 10^{15} \\ \mu_{Pluton} &: 8.71 \times 10^{11} \end{aligned}$$

FIGURE 4.9 – Constantes gravitationnelles relatives aux astres pouvant servir de référence

4.4.3 Rotation dans le repère de référence

Désormais, le satellite est placé dans sa trajectoire. Il ne reste plus qu'à relier sa trajectoire au repère de l'astre de référence. Tous les calculs restants ne sont finalement plus que des rotations. Ce qui implique sa dose de multiplications par des $\sin \Omega$ et $\cos i$ assez indigestes. Voici néanmoins le détail des calculs en figure 4.10.

```
def bodyCentricFrameCartesianCoord(orbCoords,i,omega,OMEGA)
    x = orbCoords.x*(Math::cos(omega)*Math::cos(OMEGA) - Math::sin(omega)*Math::cos(i)*Math::sin(OMEGA))
    | - orbCoords.y*(Math::sin(omega)*Math::cos(OMEGA) + Math::cos(omega)*Math::cos(i)*Math::sin(OMEGA))
    y = orbCoords.x*(Math::cos(omega)*Math::sin(OMEGA) + Math::sin(omega)*Math::cos(i)*Math::cos(OMEGA))
    | + orbCoords.y*(Math::cos(omega)*Math::cos(i)*Math::cos(OMEGA) - Math::sin(omega)*Math::sin(OMEGA))
    z = orbCoords.x*(Math::sin(omega)*Math::sin(i)) + orbCoords.y*(Math::cos(omega)*Math::sin(i))

    xSpeed = orbCoords.xSpeed*(Math::cos(omega)*Math::cos(OMEGA) - Math::sin(omega)*Math::cos(i)*Math::sin(OMEGA))
    | - orbCoords.ySpeed*(Math::sin(omega)*Math::cos(OMEGA) + Math::cos(omega)*Math::cos(i)*Math::sin(OMEGA))
    ySpeed = orbCoords.xSpeed*(Math::cos(omega)*Math::sin(OMEGA) + Math::sin(omega)*Math::cos(i)*Math::cos(OMEGA))
    | + orbCoords.ySpeed*(Math::cos(omega)*Math::cos(i)*Math::cos(OMEGA) - Math::sin(omega)*Math::sin(OMEGA))
    zSpeed = orbCoords.xSpeed*(Math::sin(omega)*Math::sin(i)) + orbCoords.ySpeed*(Math::cos(omega)*Math::sin(i))

    CartesianCoord.new(x,y,z,xSpeed,ySpeed,zSpeed)
end
```

FIGURE 4.10 – Fonction Ruby de rotation des coordonnées de la trajectoire dans le repère de référence

Suite à ces calculs, on stocke les coordonnées dans un attribut de la classe `CelestialBody` tel qu'expliqué au § 3.5.1.

Chapitre 5

Une application graphique avec Java SWING

5.1 Java Swing et Java2D

Une fois les données récupérées et les coordonnées calculées, nous devions nous concentrer sur l'aspect graphique de l'application.

Le langage Java nous permet de réaliser une interface graphique simple et ergonomique tout en combinant la facilité de développement offerte par le langage Ruby afin d'obtenir l'application finale.

C'est ainsi que notre intérêt s'est porté sur Java Swing qui est une API Java qui propose de nombreux composants dont certains possèdent des fonctions étendues, une utilisation des mécanismes de gestion d'événements performants et une apparence modifiable à la volée par exemple.

Le premier objectif a été de prendre en main cet outil et ensuite d'adapter son utilisation à l'objectif de notre projet. Notre fenêtre graphique peut être vue comme un ensemble de composants graphiques agencés les uns par rapport aux autres. Ces composants sont appelés Panels ou JPanels et nous permettent de traiter différemment les parties de notre fenêtre.

Dans notre cas, un JPanel principale correspond à la carte du système à afficher. (figure 5.1)

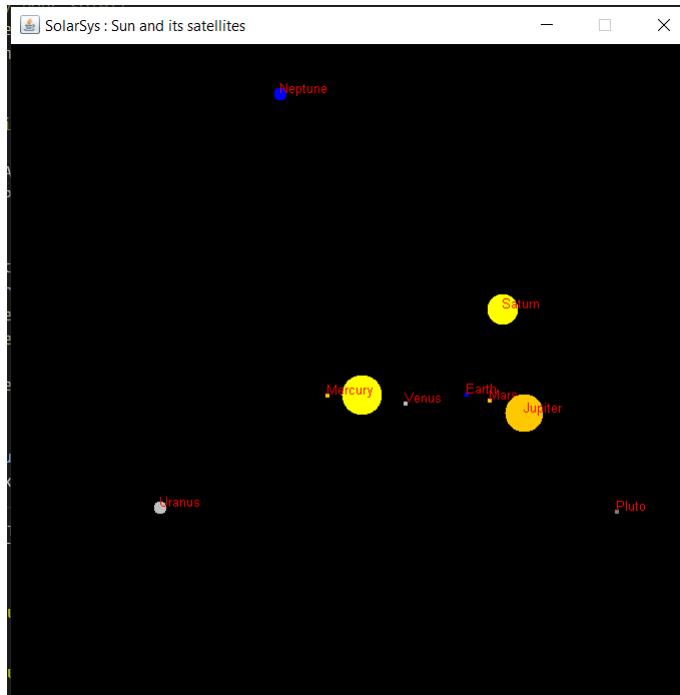


FIGURE 5.1 – Fenêtre graphique composée d'un JPanel : la carte du système solaire.

Java Swing permet d'obtenir une simple fenêtre vide. Pour ajouter des éléments, donc nos planètes, nous nous sommes concentrés aussi sur une bibliothèque graphique java appelée Java2D. Cette dernière permet de tracer toute forme géométrique en deux dimensions.

L'utilisation de cet outil est très intuitif. En effet, elle consiste à une suite d'actions ou d'opérations résumant un remplissage d'une forme géométrique à l'aide d'une peinture (couleur) sur une surface graphique (notre écran). (figure 5.2)

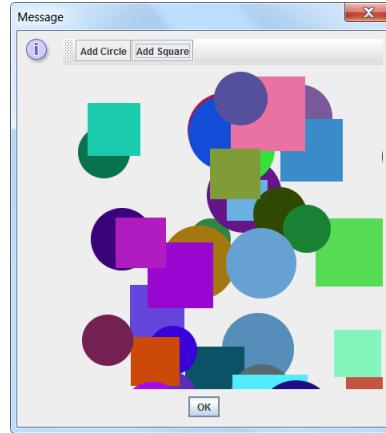


FIGURE 5.2 – Les différentes formes possibles avec la bibliothèque Java2D.

5.2 Modèle MVC

Le choix de l'architecture de notre application est très important et décisif pour la suite du projet. Il faut choisir le modèle adéquat à l'utilisation de notre application. Il fallait être capable de modifier et agencer facilement nos composants à l'écran en fonctions des données récupérées plus tôt, tout en laissant la possibilité d'interagir avec ses composants.

Le Modèle-vue-contrôleur ou MVC est un motif d'architecture logicielle destiné aux interfaces graphiques et est composé de trois types de modules ayant trois responsabilités différentes : les modèles, les vues et les contrôleurs. Cette décomposition permet une meilleure répartition du travail et une maintenance améliorée. (figure 5.3)

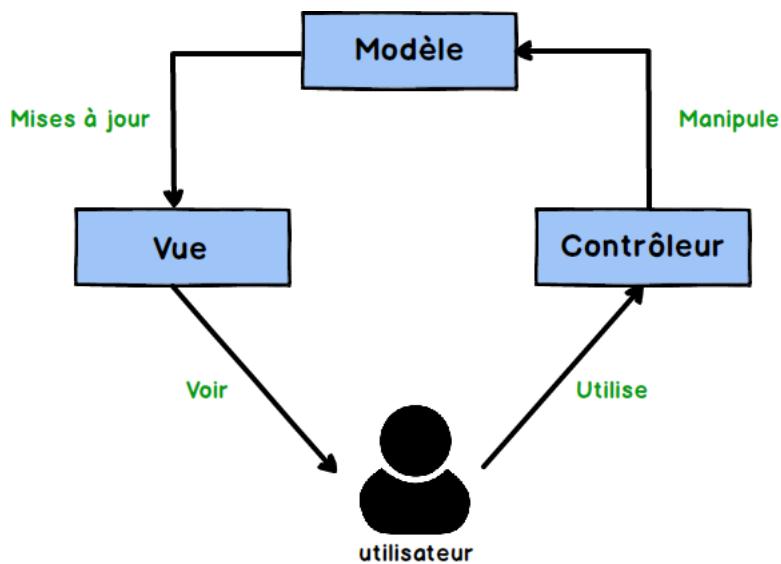


FIGURE 5.3 – Composants du modèle MVC.

5.2.1 Le modèle

Dans ce module nous nous intéressons aux données réelles des corps célestes. Nous y retrouvons, leur diamètre, leur distance par rapport à la référence, leur position, ainsi que les satellites orbitant autour de chacun d'eux (figure 5.6). Ces données nous permettront de trouver les échelles de calcul des distances et de diamètre que nous utiliseront plus tard pour leur affichage dans notre repère.

5.2.2 La vue

Ce module gère la disposition et l'affichage. Il comprend 3 classes :

1. notre fenêtre graphique "Window" ;
2. le "MapPanel" représentant le repère sur lequel placer les planètes ;
3. les éléments appelés "VisibleBody" représentant chacun une planète à afficher.

Ces trois composants auront chacun leurs propriétés et leur utilité.

Window

La première étape est de créer notre fenêtre graphique (figure 5.4). Nous devons préciser la taille de celle-i qui sera ensuite utilisée plus tard pour le calculs des échelles 5.3.

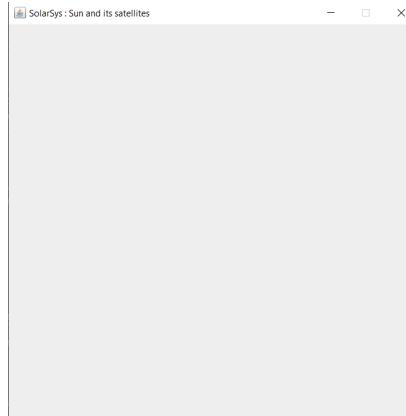


FIGURE 5.4 – Première fenêtre graphique obtenue.

MapPanel

Une fois notre fenêtre graphique ouverte, nous créons une classe appelée MapPanel qui représentera la surface qui sert à représenter une portion du système solaire (figure 5.5). C'est dans cette classe que nous allons stocker la planète/étoile de référence à afficher ainsi que tous les satellites/planètes orbitant autour de cette référence. C'est ici aussi que nous stockons les informations relatives à l'échelle appliquée aux distances et diamètres.

VisibleBody

Un VisibleBody est la classe permettant de représenter un corps céleste. On y retrouve sa position, son diamètre, ainsi que sa couleur (figure 5.6).

Cette classe respecte un patron de conception Wrapper de la classe CelestialBody. Le wrapper permet d'adapter la classe représentant un corps céleste à l'interface graphique de l'application. Ce patron de conception est utile pour faire fonctionner les différentes classes du modèle.

Grâce au VisibleBody, on peut, pour chaque astre, mieux séparer les données relatives à l'objet lui-même et celles relatives uniquement à son affichage, et tout cela sans surcharger la class CelestialBody. Cela offre également un moyen de stocker les données d'affichage et de les mettre à jour sans modifier le modèle.

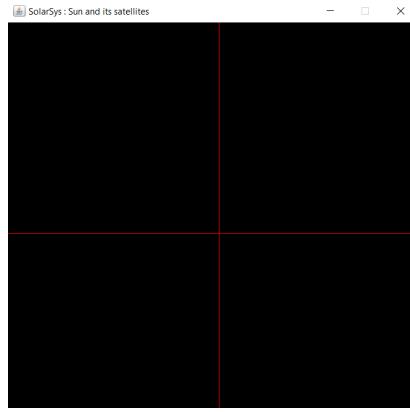


FIGURE 5.5 – Repère tracé sur la fenêtre graphique.

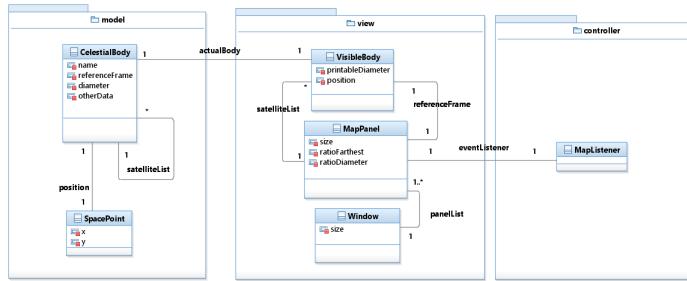


FIGURE 5.6 – Diagramme de classe de l'application.

Cette hiérarchisation des composants graphiques de notre application permet de manipuler plus facilement les objets. De plus, pour permettre le changement de vue et de référence, il suffit de mettre à jour les données présentes dans notre MapPanel.

5.2.3 Les contrôleurs

La section contrôleur gère les événements associés à nos éléments graphiques pour pouvoir interagir avec notre application.

Il est ainsi possible de cliquer sur une planète pour rafraîchir notre affichage et placer celle-ci en référence.

L'application permet aussi d'afficher les informations relatives à une planète au survol de la souris.

5.3 Calcul des échelles

Pendant la réalisation du projet, nous nous sommes attardés sur l'un des problèmes majeurs définis dès le début. Il nous fallait faire un choix sur la méthode de calcul des échelles de distances et de diamètres des corps célestes.

5.3.1 Distances

Les distances réelles entre les planètes sont astronomiques, d'où l'intérêt d'une représentation graphique du système solaire. Le but est de réaliser une représentation simplifiée et intuitive de celui-ci en se rapprochant au plus près des rapports de distances de la réalité.

Une première version

Un première version permettait le calcul de cette échelle en fonction du satellite le plus éloigné de notre référence.

Prenons le cas du système solaire. Une première étape consistait à récupérer la planète la plus éloignée du soleil, ici Pluton. Ensuite il nous faisons le choix de placer celle ci à une distance de 90% de la largeur de notre fenêtre. Ce choix nous permet ensuite de trouver un ratio entre les distances réelles et les distances d'affichage. C'est donc en fonction de ce ratio que nous trouvons les positions d'affichage des autres planètes (figure 5.7).

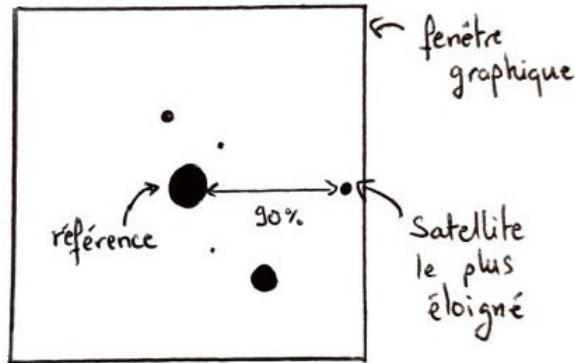


FIGURE 5.7 – Première version de calcul d'échelle de distances.

Cette version nous permettait de respecter les rapports de distances réelles. Cependant au fil du projet, nous nous sommes rendu compte que cette méthode pouvait poser problème dans certains cas, comme par exemple lorsque une planète est très isolée par rapport aux reste des planètes. C'est ce que nous avons observé dans le cas du système solaire en incluant Pluton dans la liste des planètes. Cela créait un amas de planètes au centre de la fenêtre qui compromettait la bonne visualisation du système (figure 5.8).

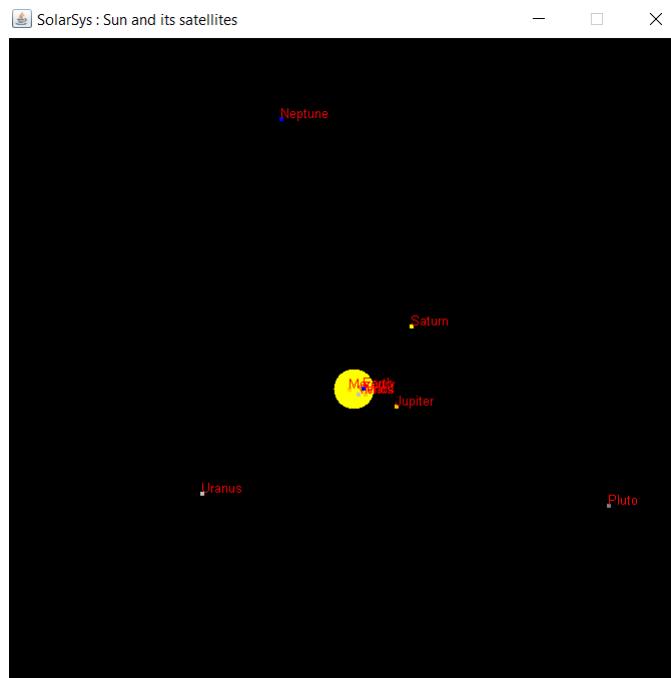


FIGURE 5.8 – Pluton isolée et amas de planètes au centre.

Version actuelle

Pour pallier ce problème nous avons décidé de définir une distance constante qui sera égale pour toutes les planètes. Ainsi nous ne parlerons pas d'échelle de distance mais plutôt d'espacement fixe

(figure 5.9). Cette distance entre chaque planète est calculée en divisant la largeur de notre fenêtre par le nombre de satellite de notre vue actuelle. Cette solution nous a permis de mieux visualiser les planètes affichées et de se concentrer plutôt sur l'échelle des diamètres des planètes.

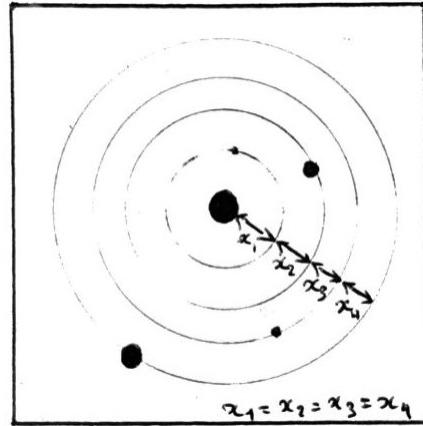


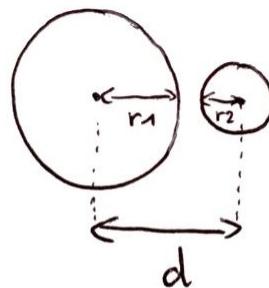
FIGURE 5.9 – Espacement fixe entre les planètes

5.3.2 Diamètres

Une fois la distance entre les planètes fixée, nous pouvons trouver une échelle pour les diamètres de celle ci. Pour se faire, Nous avons opté pour la méthode suivante :

- nous parcourons deux à deux l'ensemble de nos planètes de la plus proche à la plus éloignée du centre.
- nous calculons le rapport entre la somme des deux rayons et la distance séparant les deux planètes.
- nous stockons le plus petit rapport trouvé au cours du parcours.

Cette méthode permet d'éviter tout chevauchement entre deux planètes. Elle adapte ensuite le diamètre des autres planètes en fonction. Le *ratio* trouvé est donc stockée et sera considéré comme notre échelle de diamètre (figure 5.10). Les diamètres de toutes les planètes seront donc calculée en les multipliant par cette échelle.



$$\text{ratioDiameter} = \frac{r_1 + r_2}{d}$$

FIGURE 5.10 – Méthode de calcul de l'échelle de diamètre.

5.3.3 Référence et échelle

La gestion des paramètres du corps céleste de référence est particulière. Pour la distance, c'est une évidence, la référence est placée au centre de notre repère. Cependant, le diamètre de celui ci pose

problème. Deux options s'offrait alors :

- Adapter la référence aux autres planètes, donc appliquer l'échelle calculée précédemment au corps céleste de référence.
- Fixer la taille du corps de référence quelque soit le système affiché.

Malheureusement, dans les deux cas proposés, un problème apparaissait.

Pour la première méthode, la référence pouvait être trop grande par rapport aux autres planètes. L'intérêt d'une échelle de diamètre était donc perdu puisque les autres planètes étaient excessivement petites (figure 5.11).

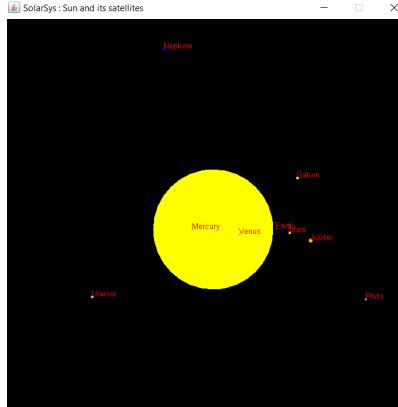


FIGURE 5.11 – Référence trop grande lorsqu'elle respecte l'échelle.

Pour la deuxième méthode, dans certains cas la référence était trop petite par rapport à ses satellites. Prenons le cas de la Terre et la Lune, la Lune était deux fois plus grande que la Terre sur notre représentation. Il n'est donc pas représentatif des vrais rapport de grandeur entre la référence et ses satellites (figure 5.12).

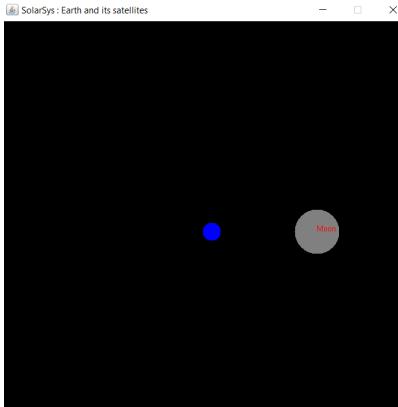


FIGURE 5.12 – Référence trop petite lorsqu'elle ne respecte pas l'échelle.

Pour pallier ces problèmes nous avons combiner ces deux méthodes. La référence respecte l'échelle de diamètre lorsque la différence de son diamètre avec celui du plus grand de ses satellites n'est pas très élevée (figure 5.13). Si celle ci est trop importante, le diamètre de la référence est fixé pour pouvoir se concentrer sur la représentation des autres planètes (figure 5.14).

5.4 Changement de référence et événements

Il est possible dans notre application d'interagir avec ses composants. Ainsi la gestion des événements permet à l'utilisateur de changer de référence ou aussi d'afficher des informations.

Un simple clic sur une des planètes permet d'afficher celle-ci en référence ainsi que tous les satellites qui orbitent autour.

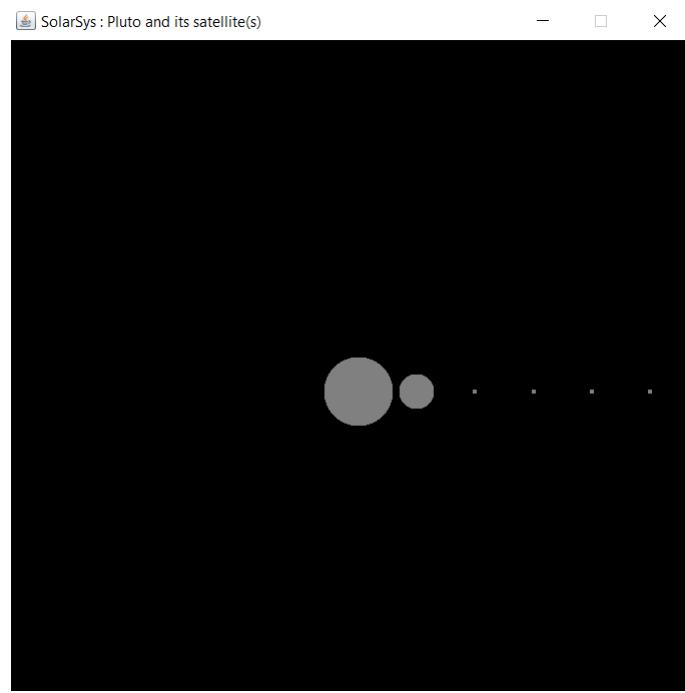


FIGURE 5.13 – Référence à l'échelle.

Au survol de la souris ce sont ses informations qui sont affichées. informations comme la vitesse de rotation, la distance par rapport à la référence, ou encore le diamètre (figure 5.15).

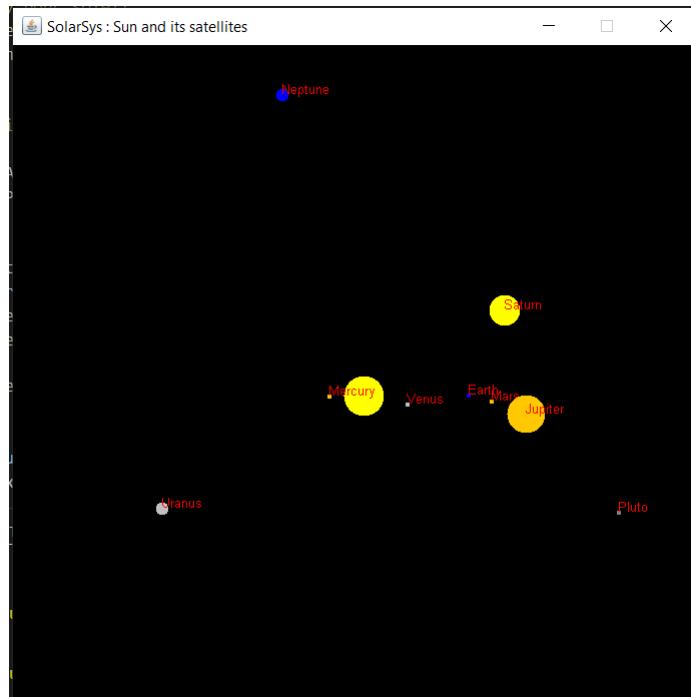


FIGURE 5.14 – Référence avec taille fixe.

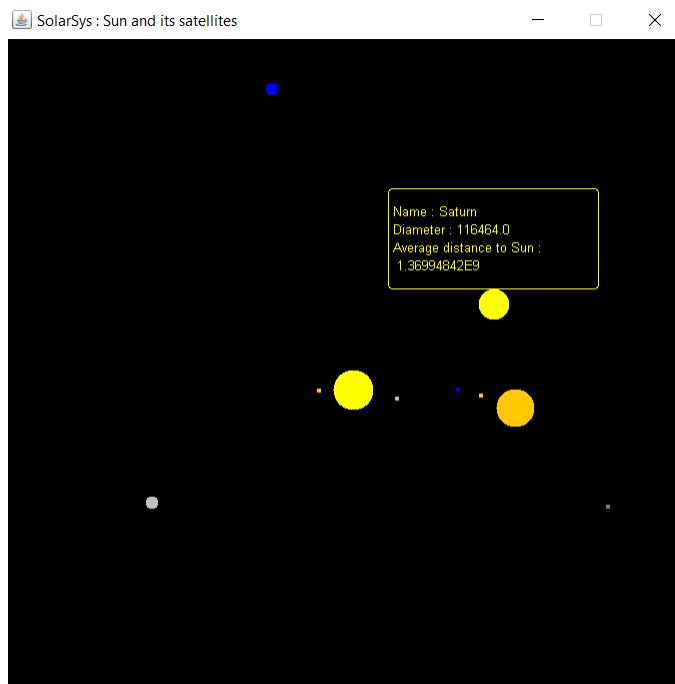


FIGURE 5.15 – Informations affichées au survol de la souris.

Chapitre 6

Résultats

6.1 État actuel de l'application

Aujourd’hui, l’application *SolarSys* permet de visualiser l’ensemble des planètes de notre système solaire. De par sa conception, et c’était l’une de nos premières intentions en commençant ce projet, il serait assez aisément de modifier notre application afin qu’elle puisse traiter et afficher des données en provenance d’une autre API. De plus, si l’organisation des données JSON respecte la même architecture que l’actuelle, on pourrait tout à fait envisager de représenter d’autres systèmes planétaires.

L’application n’est cependant pas complète. Il s’agit plus d’un prototype que d’un projet réellement achevé. Néanmoins, une bonne partie de son code est tout à fait réutilisable et on pourrait tout à fait s’en servir comme base au développement de d’une version plus ambitieuse.

6.2 pistes d’améliorations

Malgré le développement de toutes ces fonctionnalités, beaucoup de pistes restent encore à explorer. Pour aller des modifications les plus légères aux plus lourdes, nous pourrions encore, avec plus de temps devant nous :

- Augmenter le nombre d’informations affichées au survol de la souris,
- Faire se déplacer les planètes et satellites autour de leur astre de référence,
- Passer d’un modèle 2D à un modèle 3D de la carte,
- Permettre à l’utilisateur de zoomer/dézoomer et déplacer son point de vue sur la carte (cela permettrait également d’afficher les objets célestes avec les mêmes échelles de distances),
- Remplacer les disques colorés par de véritable images de planètes et les animer avec des threads permettant aux interfaces Ruby et Java de communiquer en temps réel des positions pour les astres affichés…

6.3 Conclusions personnelles

Avec ces 6 mois de l’année qui nous ont été laissés pour réaliser ce projet, nous avons eu quelques difficultés pour nous organiser ensemble et avancer sérieusement. Il a été assez rare que nous trouvions du temps en commun tous les trois pour discuter du développement du projet. Si bien que l’application avançait petit pas par petit pas au fil des remarques que l’un ou l’une faisait à l’autre entre deux cours. Néanmoins, chacun d’entre nous a pu trouver son compte puisque nous n’avions pas tous la même affinité pour l’astronomie, les représentations graphiques ou le traitement de fichiers JSON et ce fut pour chacun une occasion de sortir de sa zone de confort et de découvrir une facette de l’informatique un peu moins scolaire que ce que nous avions connu jusqu’alors. Nous trouvons également assez satisfaisant d’avoir pu produire une application de A à Z et en tirons une certaine fierté.

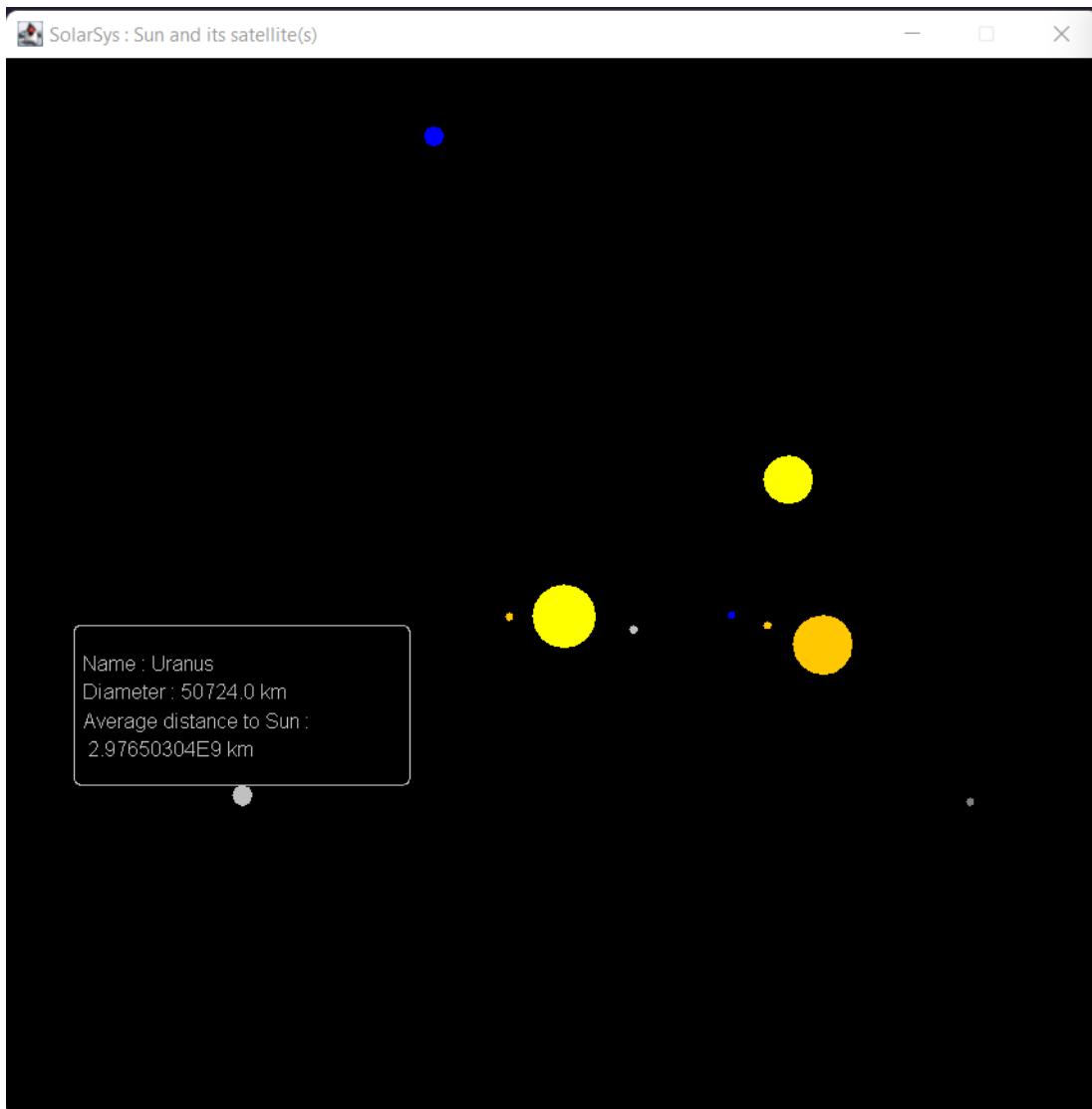


FIGURE 6.1 – Rendu final de l'application : Le système solaire

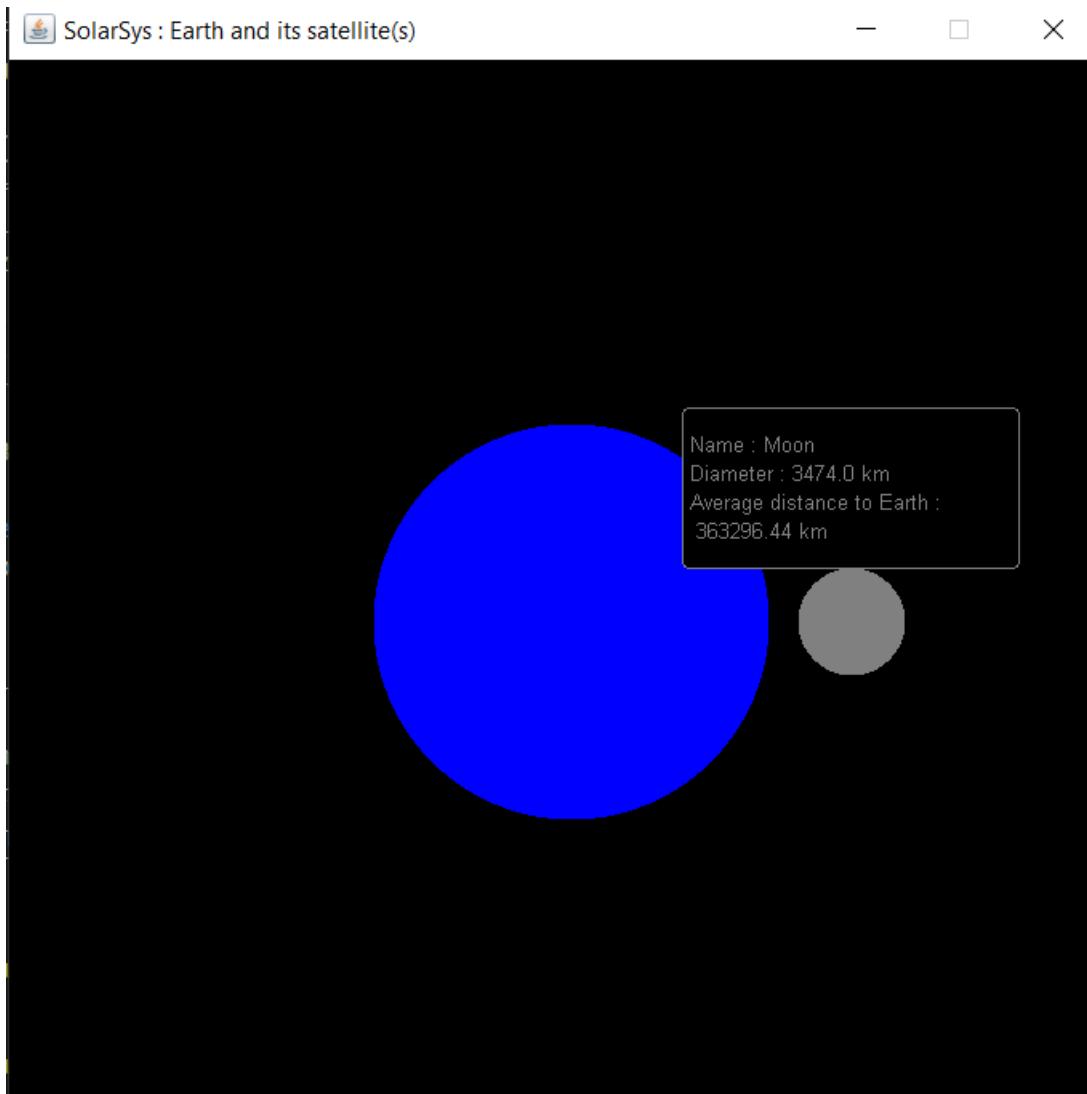


FIGURE 6.2 – Rendu final de l'application : La Terre

Bibliographie

- [1] Richard ANSTEE. *the Newton-Raphson Method*. URL : <https://personal.math.ubc.ca/~anstee/math104/newtonmethod.pdf>.
- [2] CHRISTOPHE. *L'OpenData du Système Solaire*. 2019-2022. URL : <https://api.le-systeme-solaire.net/>.
- [3] ORBITNERDS. *Cartesian vs Keplerian Elements*. URL : <https://www.youtube.com/watch?v=A-wAiFFSY6o>.
- [4] Rene SCHWARZ. *Memorandum n°1 : Keplerian orbit elements to Cartesian state vectors*. 2017. URL : https://www.rene-schwarz.com/web/Science/Memorandum_Series.

Résumé

Ce document fait le rapport de notre projet tuteuré de L3 Informatique à l'université de Franche-Comté. Au cours de cette année de notre formation, nous avons réalisé une application de carte en 2D du système solaire. Le programme a été réalisé en JRuby et se fournit en données astronomiques à l'aide d'une API en ligne.

Mots clefs— API REST, JSON, Éléments orbitaux, JRuby, Java SWING, Java2D, Astronomie, Carte