LIMBALLE Pierre - PEREZ Quentin
UFR-ST - Licence informatique 3ème année
Groupe TP 2B

4 janvier 2016

Méthodes et Outils pour la Programmation
Mini Projet 2015

Rapport

Contacts : pierre.limballe@edu.univ-fcomte.fr, quentin.perez@edu.univ-fcomte.fr
Professeurs : GREFFIER Françoise, BONNEVILLE François

Table des matières

1 Sujet

1.1 Contexte

Dans le cadre du module de Méthodes et Outils pour la Programmation (MOP), la réalisation d'un projet nous a été demandé afin de mettre en application les connaissances théoriques acquisent tout au long du semestre 5.
La conception de l'application a pour objectifs :

- l'utilisation du langage de programmation Java en Orienté Objet
- l'application du paradigme MVC (Modèle - Vue - Contrôleur)
- l'approche du travail collaboratif par l'utilisation de logiciels de gestion de versions
- la manipulation d'éléments graphiques de la bibliothèque graphique JavaSwing

1.2 But de l'application

La finalité de ce projet est la conception d'une application Java qui réalise une animation graphique en 2D, représentant un ensemble de planètes et leurs satellites en orbite autour d'une ou plusieurs étoiles. L'utilisateur doit disposer des fonctionnalités suivantes :

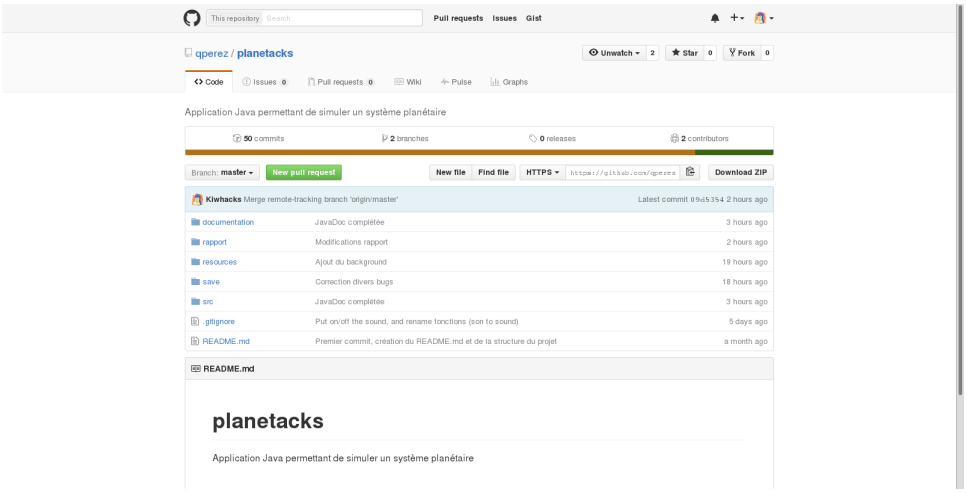
- l'ajout et la suppression d'astres (étoiles ou satellites) dans l'application
- l'affichage des astres dans une fenetre graphique animée
- la possibilité de sauvegarder un système planétaire

La contrainte d'ergonomie est également importante quant à l'utilisation du logiciel.

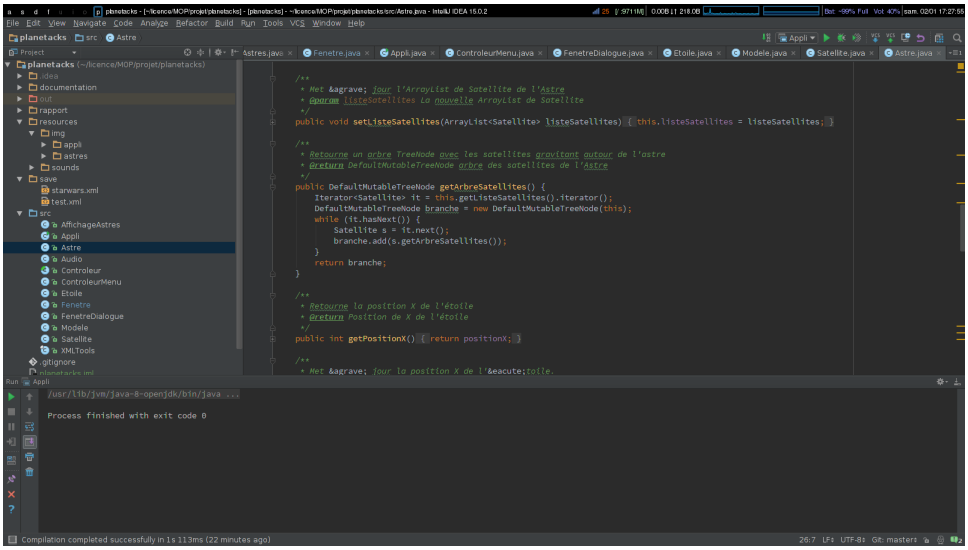
2 Conception

2.1 Introduction

Planethacks est le nom choisi pour l'application. Du fait d'un travail en binôme, l'utilisation d'un logiciel de gestion de version fut priviligiée. Nous avons pour cela utilisé Git, en lien avec le serveur d'hébergement GitHub (sources GitHub de Planethacks : `https://github.com/qperez/planetacks`).



L'IDE (*Internal Development Environement*) IntelliJ IDEA fut utilisé durant toute la phase de développement et de tests, et ce afin d'accélérer et de faciliter le développement de l'application ainsi que son déboguage.



2.2 Structures de données au sein de l'application

La définition de classes permettant de structurer les données de notre programme fut la première étape de conception. Ainsi 3 classes nous permettent de modéliser un astre :

— Astre (classe abstraite)
— Etoile
— Satellite

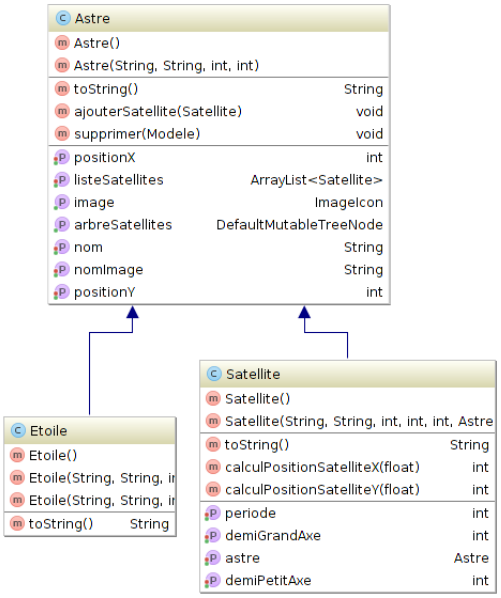
Planethacks repose sur une conception basée sur le paradigme MVC. Ce paradigme impose l'utilisation à minima de 3 classes (Modèle - Vue - Contrôleur) permettant de séparer les données de l'affichage et des interactions utilisateur.
Dans un soucis de respect du modèle MVC, nous disposons donc des classes suivantes :

— Fenetre
— Modele
— Controleur (classe abstraite)
— ControleurMenu
— AffichageAstres

Afin de réaliser la fonctionnalité de sauvegarde, une classe spécifique contenant les outils nécessaires fut créée : *XMLTools*.
Le point d'entrée de l'application est assurée par la classe *Appli*.

2.2.1 Modélisation d'un astre (Etoile ou Satellite)

Le diagramme UML ci-dessous représente la modélisation choisie pour un astre :

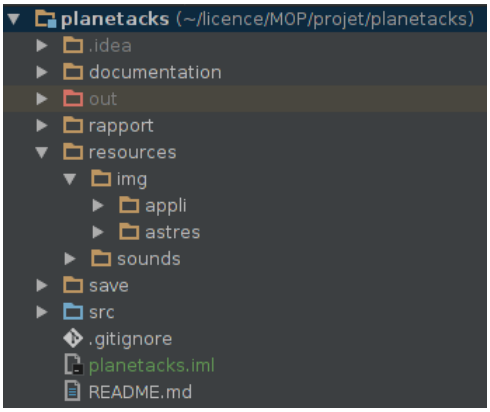


Les méthodes *calculPositionSatelliteX(float)* et *calculPositionSatelliteX(float)* présentes dans la classe Satellite permettent de calculer la position en X et en Y du Satellite en fonction du temps. La méthode *ajouterSatellite(Satellite)* permet de mettre en orbite un satellite autour d'un astre. Enfin, la méthode *supprimer(Modele)* permet de supprimer récursivement un astre et ses satellites à partir du modèle donné.

2.3 Structure générale de l'application

2.3.1 Emplacement des fichiers

Nous avons utilisé une structure de fichiers qui nous parraissait être la plus simple et la meilleure pour le fonctionnement de l'application. Ci-dessous, une représentation de l'arborescence :



Nous pouvons retrouver les dossiers de production suivants, dans lesquels nous manipulions directement les fichiers :

- `src`
  → dossier des sources (.java)
- `ressources`
  → dossier des différents fichiers image ou audio
  - `img`
    → dossier contenant toutes les images du projet
    - `appli`
      → dossier des images de l'application en général
    - `astres`
      → dossier des images des astres (.png)
  - `sounds`
    → dossier contenant tous les sons du projet
- `rapport`
  → dossier contenant les fichiers permettant de réaliser ce rapport (fichiers LaTeX)

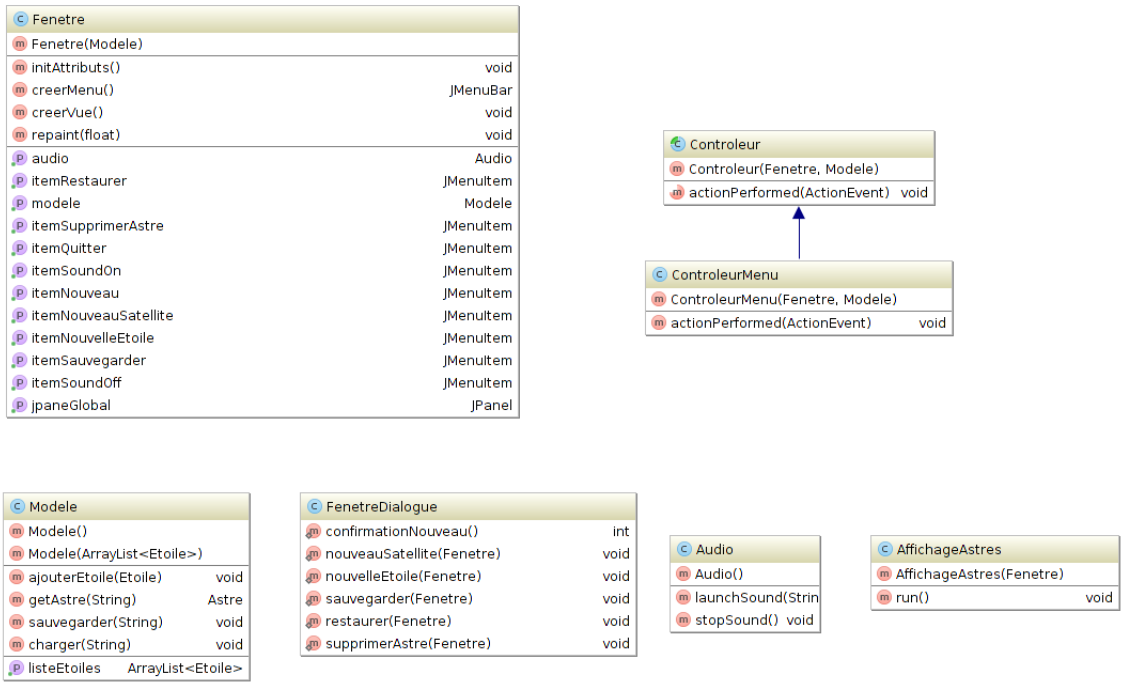
Mais également des dossiers dont le contenu est généré automatiquement :

- `out`
  → dossier des fichiers java compilés (.class)
- `save`
  → dossier des fichiers de sauvegarde générés par l'application (.xml)
- `documentation`
  → dossier contenant la documentation du projet (.html)
- `.idea`
  → dossier contenant les fichiers de configuration du logiciel *IntelliJ IDEA*

Ainsi que des fichiers seuls :

- `.gitignore`
  → fichier permettant d'ignorer l'envoi de fichiers sur GitHub (ex : dossier *out*)
- `README.md`
  → fichier MarkDown de présentation du projet sur GitHub
- `planetacks.iml`
  → fichier de configuration du projet pour *IntelliJ IDEA*

2.3.2 Architecture MVC



Comme dit précédemment, nous avons utilisé le paradigme MVC afin de faciliter le développement. Dans cette architecture, nous pouvons retrouver 3 principales classes qui sont les représentantes du modèle MVC :

— *Modele* abstraite implémentant la classe *ActionListener*
— *Fenetre* étendue de la classe *JFrame*
— *Controleur*

La classe *Modele* nous permet de gérer la liste des astres, comme par exemple ajouter une Etoile, ainsi que l'enregistrement et la restauration d'un système planétaire. Plus généralement, elle est le modèle des données de l'application.

La class *Fenetre* s'occupe de tout ce qui touche à l'interface utilisateur, sauf la gestion des différents pop-up que nous verrons dans la classe *FenetreDialogue*. Elle regroupe la présentation et le design de l'application, comme le menu, l'image de fond...

La classe *Controleur* quant à elle, permet de faire le lien entre les deux classes précédente. Elle permet donc la gestion des événements, ainsi que la synchronisation entre les données et l'affichage.

Ici, le contrôleur connait la vue et le modèle, la vue ne connait que le modèle, tandis que le modèle n'a pas d'informations sur les deux autres.

La classe *ControleurMenu* ne contrôle uniquement le menu et les actions qu'il peut avoir. Elle est étendue de la classe *Controleur*.

La classe *FenetreDialogue* prend en compte toutes les interactions avec l'utilisateur, comme l'ajout d'un astre, ou encore la sauvegarde d'un système planétaire.

La classe *AffichageAstres* étendue de la classe *Thread*, permet de lancer une boucle infinie pour desssiner les astres sur la fenêtre principale.

La classe *Audio* nous permet de gérer plus facilement la lancement et l'arrêt du son de l'application.

2.4 Les algorithmes intéressants

Algorithm 1 méthode repaint de la classe Fenetre

function REPAINT(t temps)
  Supprimer l'ensemble des composants graphique du JPanel
  for chaque Etoile e de la liste d'étoiles du Modele **do**
    $jlabastre \leftarrow$ nouveau JLabel avec l'image de e
    Afficher $jlabastre$ en fonction de la position X_e et Y_e
    Ajouter le $jlabastre$ au JPanel
    Appel de la fonction $repaintSatellite$ avec $e.getListeSatellite()$ et t
  end for
  Appel de la méthode $repaint()$ sur le JPanel
end function
function REPAINTSATELLITE(listeSatellite l, temps t)
  for chaque Satellite s de la liste des satellites l **do**
    $jlabsat \leftarrow$ nouveau JLabel avec l'image de s
    Afficher $jlabsat$ en fonction de $s.calculPositionSatelliteX(t)$ et $s.calculPositionSatelliteY(t)$
    Ajouter le $jlabsat$ au JPanel
    Appel récursif de $repaintSatellite$
  end for
end function

Algorithm 2 méthode encodeFromFile de la classe XMLTools

function DECODEFROMFILE(Object $object$, String $fileName$)
  $fos \leftarrow$ nouveau FileOutputStream initialisé avec $fileName$
  $decoder \leftarrow$ nouveau XMLEncoder initialisé avec fos
  écrire $object$ dans le buffer I/O avec $encoder.writeObject(object)$
  flush du buffer I/O avec $encoder.flush()$
end function

Algorithm 3 méthode decodeFromFile de la classe XMLTools

```
function DECODEFROMFILE(Object object, String fileName)
    fis ← nouveau FileInputStream initialisé avec fileName
    decoder ← nouveau XMLEncoder initialisé avec fis
    object ← l'objet renvoyé par decoder.readObject()
    la ← transtypage de object en ArrayList d'Astre
    it ← nouvelle itérateur d'ArrayList d'Astre
    while it retourne qu'il reste un élément à parcourir do
        astre ← élément suivant Astre pointé par it
        mise à jour de l'image de l'image de astre à l'aide de a.setImage()
    end while
return la
end function
```

2.5 Points intéressants et supplémentaires du programme

Afin d'être original, avons décidé de créer un "mode" *StarWars*.

Tout d'abord, il a fallu recréer des nouvelles planètes tirées de la saga, comme *Tatooine*, *Naboo*, *Jakku*, l'*étoile noire* ainsi qu'un *fighter* (vaisseau côté obscur de la force) ou encore le *Faucon Millenium* d'Han Solo. Ceci nous a fait découvrir l'utilisation de zones transparentes dans une image PNG, ainsi que le logiciel Gimp que nous n'avions pas utilisé dans notre formation jusqu'à présent.

De plus, nous avons implémenté la gestion du son, que l'on peut activer ou désactiver grâce à un menu prévu à cet effet. Celui-ci s'active automatiquement lorsque l'on charge la sauvegarde de l'environnement *StarWars*, avec le thème original des 7 films accompagné de plusieurs bruitages célèbres. L'intégration de son dans une application était inconnue pour nous auparavant, c'est pourquoi nous avons décidé de l'ajouter à notre projet pour la découvrir.

2.6 Tests

Les tests ainsi que la documentation ont été réalisés tout au long du développement. Ceci nous a permis de pouvoir déboguer facilement et de ne pas perdre de temps dans la production à cause d'erreurs de programmation. Les outils utilisés pour la construction du projet nous ont aussi facilité les tests.

En effet, *IntelliJ IDEA* propose des fonctionnalités permettant d'éviter un grand nombre d'erreurs (auto-complétion inter-classes, renommage des variables et fonctions sur tout le projet, messages explicites lors d'une erreur...) et de passer moins de temps sur des problèmes de syntaxe.

De plus, grâce au gestionnaire de version GitHub, il est aisé de retrouver les modifications apportées à un fichier, et pourquoi elles ont été faites. Ici encore, cela permet de gagner du temps et de pouvoir retrouver un projet fonctionnel même après avoir rencontré un problème lourd lors du développement.

Pour finir, ces deux logiciels fonctionnent l'un avec l'autre, ce qui rend la construction du projet très confortable.

Une phase de tests fonctionnels plus conséquente a été réalisée à la fin du projet. Dans un souçis de professionnalisation et de rigueur quant à la démarche de tests, plusieurs fiches de vérification d'aptitude au bon fonctionnement de l'application ont été réalisées.

Pour ce qui est des tests des entrées utilisateur lors du déroulement du programme, nous testons si le type des données rentrées est bon. Par exemple, si nous attendons un entier et que l'utilisateur rentre "toto", un message est affiché à l'écran. De même, si l'utilisateur a oublié de remplir un champ, il lui sera annoncé à l'écran et il ne pourra pas valider son ajout ou sa suppression d'astre tant que les données ne sont pas conformes.

Vérification d'aptitude au bon fonctionnement

Scénario de test numéro 1

Profil d'utilisateur visé : Étudiant

Titre du scénario : Création et sauvegarde d'un système planétaire

Conditions préalables :
- Disposer d'un ordinateur ayant Java Runtime Environement installé
- Disposer du logiciel Planethacks
- Aucunes aides à la prise en main ou à la décision durant la durée du test

Description : on souhaite vérifier que l'application permet l'ajout de 2 étoiles ainsi que de 3 satellites. On désire ensuite sauvegarder l'ensemble du système planétaire créé dans un fichier "test.xml". Le test s'effectue sans jeu de données préalables.

Instructions du scénario

1. Ne pas ouvrir de nouveau système planétaire à l'ouverture de l'application
2. Ajouter une étoile se nommant Tatooine en position X=250 et Y=300 et ayant pour image "tatooine.png"
3. Ajouter un satellite à l'étoile Tatooine se nommant Asteroide et ayant un demi-grand axe=50, un demi-petit axe = 50, une période de 150 et ayant pour image "asteroide.png"
4. Ajouter un satellite à l'étoile Tatooine se nommant Faucon et ayant un demi-grand axe=50, un demi-petit axe = 50, une période de 150 et ayant pour image "faucon_millenium.png"
5. Ajouter une étoile se nommant Terre en position X=600 et Y=400 et ayant pour image "terre.png"
6. Ajouter un satellite à l'étoile Terre se nommant Lune et ayant un demi-grand axe=200, un demi-petit axe=50, une période de 80 et ayant pour image "lune.png"
7. Sauvegarder le système planétaire sous le nom "test"
8. Quitter l'application
9. Vérifier que test.xml a été créé

Vérification d'aptitude au bon fonctionnement

Scénario de test numéro 2

Profil d'utilisateur visé : Étudiant

Titre du scénario : Chargement et modification d'un système planétaire

Conditions préalables :

- Disposer d'un ordinateur ayant Java Runtime Environement installé
- Disposer du logiciel Planethacks
- Disposer d'un jeu de données correpondant à une sauvegarde d'un système planétaire
- Aucunes aides à la prise en main ou à la décision durant la durée du test

Description : on souhaite vérifier que l'application puisse charger correctement un système planétaire enregistré au format XML mais également qu'il puisse être modifié, c'est à dire que 2 astres puissent être supprimés et un autre ajouté. Le test s'effectue avec jeu de données "starwars.xml".

Instructions du scénario

1. Choisir d'ouvrir un nouveau système planétaire lors du démarrage de l'application, choisir "starwars.xml"
2. Supprimer l'astre "tatooine"
3. Ajouter une étoile se nommant Terre en position X=600 et Y=400 et ayant pour image "terre.png"
4. Supprimer l'astre "etoile noire"
5. Quitter l'application

Vérification d'aptitude au bon fonctionnement

Scénario de test numéro 3

Profil d'utilisateur visé : Étudiant

Titre du scénario : Vérification de l'activation de la désactivation de l'environnement sonore et de la fonction de fermeture de l'application

Conditions préalables :

- Disposer d'un ordinateur ayant Java Runtime Environement installé
- Disposer du logiciel Planethacks
- Disposer d'enceintes stéréo (le surround est en option dans certaines salles équipées)
- Aucunes aides à la prise en main ou à la décision durant la durée du test

Description : on souhaite vérifier que l'on puisse activer ou désactiver l'environnement sonore de l'application mais également que la fonction permettant de quitter l'application fonctionne correctement. Le test s'effectue sans jeu de données préalables.

Instructions du scénario

1. Ne pas ouvrir de nouveau système planétaire à l'ouverture de l'application
2. Activer le son et vérifier que le thème original de Star Wars est entendu
3. Désactiver le son
4. Activer le son et vérifier que le thème original de Star Wars est entendu
5. Quitter l'application à l'aide du menu

3 Conclusion

3.1 Bilan

Nous avons pu répondre aux différents points demandés à savoir :
— Créer un astre qui est ajouté dynamiquement à l'ensemble des astres de l'appli
— Supprimer un astre dynamiquement de l'application
— Enregistrer sur le disque un fichier comprenant une étoile et ses satellites
— Ouvrir un fichier enregistré comprenant une étoile et ses satellites
— JavaDoc générée du programme
— .jar exécutable

Toutes les classes ont été testées afin de limiter au maximum les bogues et les problèmes lors de l'éxécution.

3.2 Les acquis

Nous avons acquéris diverse connaissance durant ce projet. Premièrement, nous avons approfondi nos connaissances sur le MVC ainsi que sur la manière de l'utiliser. Des problèmes de conception et nottamment d'accessibilité des variables ont été rencontrés et par la suite corrigés, ce qui a été formatteur.

Par le fait d'avoir implémenté un mode *StarWars*, nous avons pu découvrir plusieurs technologies que nous ne savions pas utiliser. La première, la retouche photo qui a été effectuée avec *Gimp* (logiciel gratuit et libre) afin de créer différentes planètes à partir d'images, en ajoutant un filtre de transparence. La deuxième, la prise en charge de la musique dans l'application a été amusante à mettre en place, d'autant plus que nous ne l'avions jamais fait auparavant lors de projets diverses.

Le travail en groupe n'était pas une déouverte, mais nous avions pu confirmer notre méthode pour parvenir à fournir un travail efficace avec le moins de problèmes possibles.

3.3 Améliorations apportées

Globalement, nous n'avons pas apporté d'importantes fonctionnalités au projet, ne serait-ce la possibilité d'enregistrer plusieurs étoiles dans un même fichier de sauvegarde qui n'était pas demandé. Autre détail, nous pouvons aussi lancer ou couper de la musique, mais nous n'avons pas implémenté le fait de choisir celle que l'on veut. Pour ce qui est de la conception, nous avions pensé que le paradigme MVC était le meilleur moyen pour réaliser le projet, c'est pourquoi nous nous somme tourné vers cette solution.

Afin de mieux visualiser le travail fourni par chacun d'entre nous, nous avons généré un rapport d'analyse statistique du dépôt *GitHub*, où vous pourrez retrouver les différentes modifications apportées au cours du développement, le nombre de lignes codées ainsi que les responsabilités de chacun

3.4 Améliorations futures

Pour ce qui est des améliorations possibles de l'application, nous avons pensé à un système de "Drag and drop" pour le placement des étoiles et satellites, à la place d'un menu où l'on doit rentrer nos coordonnées à la main. Ceci permetterai de créer des systèmes planétaires plus rapidement. Toute fois, il reste important de conserver la possibilité de rentrer les coordonnées manuellement afin de pouvoir construire de manière précise les astres.

Pour ce qui est du réalisme de l'application, nous avions pensé à plusieurs améliorations. Tout d'abord, nous pourrions prendre en compte la masse des astres, composante qui est prise en compte dans la loi de Kepler. Ceci donnerai un aspect plus vrai sur la rotation des astres.

Au niveau de l'affichage de ceux-ci, nous pensions qu'établir une relation d'échelle entre les astres permettrais de construire un univers plus probable, afin de ne pas avoir de satellites plus gros que leur astre référent. Il en est de même pour la masse. De plus, pour avoir un réalisme plus

prononcé, il serait intéressant de modéliser un astre par plusieurs images qui représenteraient chacune un angle de vue différent, et de les adapter en fonction de la position engendrée par sa rotation.

Enfin, pour gagner en souplesse d'enregistrement, il pourrait être intéressant de pouvoir fusionner deux sauvegardes afin d'avoir plusieurs systèmes créés indépendamment sur le même écran. Ceci implique d'autres fonctionnalités, comme le fait de zoomer et dézoomer sur la vision générale des astres, ou encore de pouvoir déplacer une étoile ainsi que tous ses satellites récursivement à l'aide d'un drag and drop.