

LIMBALLE Pierre - PEREZ Quentin

UFR-ST - Licence informatique 3ème année
Groupe TP 2B

2 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	2
1.1	Contexte	2
1.2	But de l'application	2
2	Conception	3
2.1	Introduction	3
2.2	Structures de données au sein de l'application	3
2.2.1	Modélisation d'un astre (Etoile ou Satellite)	4
2.3	Structure générale de l'application	4
2.3.1	Emplacement des fichiers	4
2.3.2	Architecture MVC	6
2.4	Les algorithmes intéressants	6
2.5	Points intéressants et supplémentaires du programme	6
2.6	Tests	7
3	Conclusion	8
3.1	Bilan	8
3.2	Les acquis	8
3.3	Améliorations apportées	8
3.4	Améliorations futures	8

1 Sujet

1.1 Contexte

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

La conception de l'application a pour objectifs :

- l'utilisation du langage de programmation Java en orienté objets
- 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 Java Swing

1.2 But de l'application

La finalité de ce projet est la conception Java une application 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 :

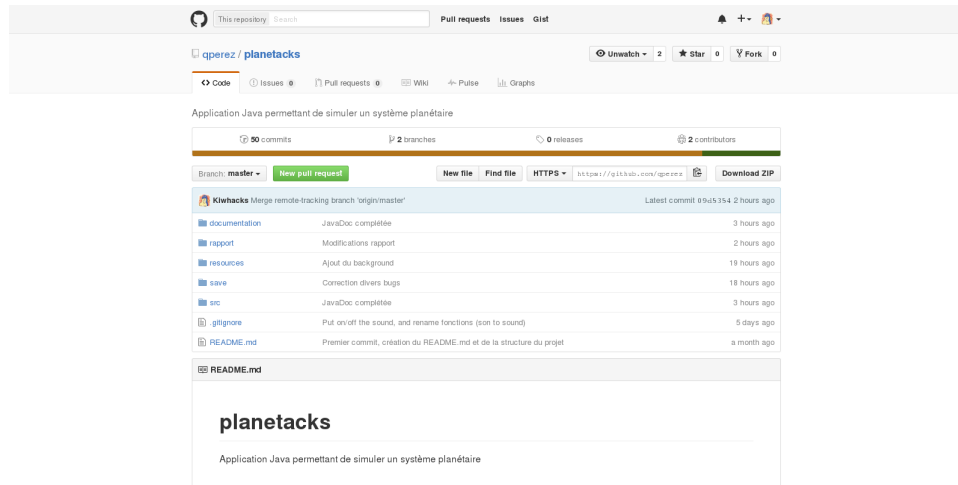
- l'ajout et la suppression d'astres (étoiles ou satellites) dans l'application
- l'affichage des astres dans une fenêtre graphique animée
- la possibilité de sauvegarder un système planétaire

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

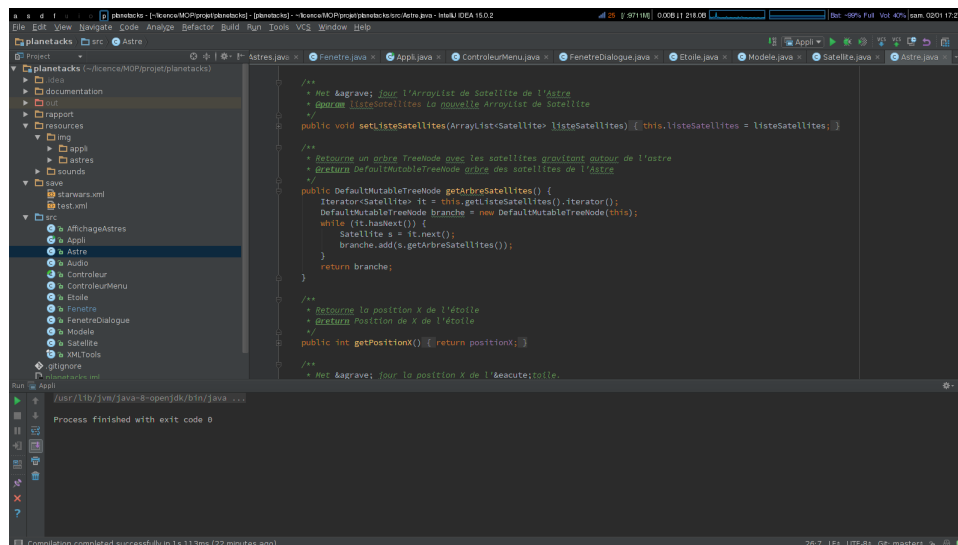
2 Conception

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 privilégiée. 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 Environnement*) IntelliJ IDEA fut utilisé durant toute la phase de développement et de tests et ce afin d'accélérer et faciliter le développement de l'application ainsi que son débogage.



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 :

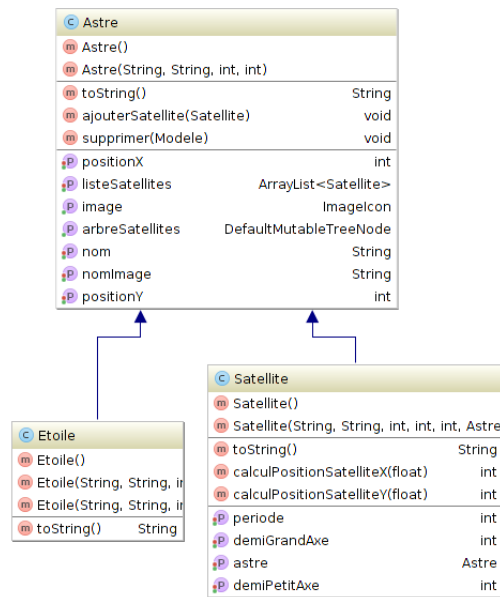
- Fenetre
- Modele
- Controleur (classe abstraite)
- ControleurMenu
- AffichageAstres

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 assuré 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 :

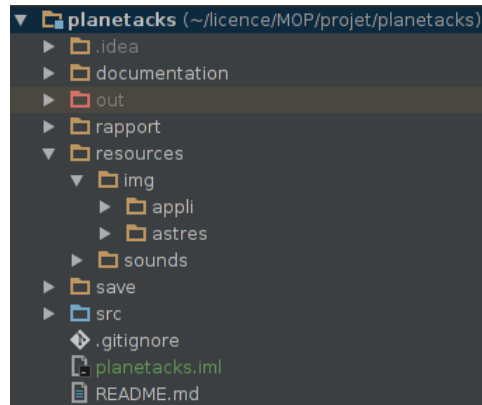


Les méthodes *calculPositionSatelliteX(float)* et *calculPositionSatelliteY(float)* présentes dans la classe *Satellite* permettent de calculer la position en X et en Y du *Satellite* en fonction du temps. La méthode *ajouterSatellite(Satellite)* permet de mettre en orbite un satellite autour d’un astre. 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 paraissait 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 manipulons 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 $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$)

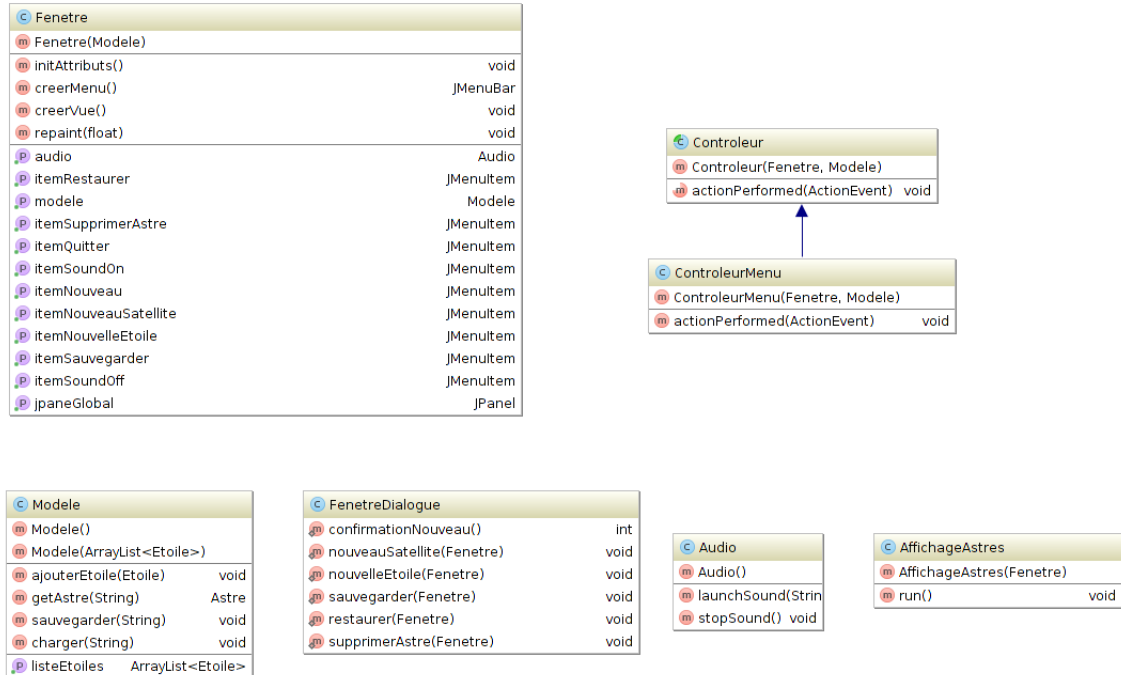
Mais également des dossiers dont le contenu est généré automatiquement :

- **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 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 connaît la vue et le modèle, la vue ne connaît 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 dessiner 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

2.5 Points intéressants et supplémentaires du programme

Afin de ne pas rester dans la banalité, avons décidé de créer un "mode" *StarWars*.

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(l listeSatellite, t temps)
    for chaque Satellite  $s$  de la liste des satellites  $l$  do
         $jlabst \leftarrow$  nouveau JLabel avec l'image de  $s$ 
        Afficher  $jlabst$  en fonction de  $s.calculPositionSatelliteX(t)$  et
         $s.calculPositionSatelliteY(t)$ 
        Ajouter le  $jlabst$  au JPanel
        Appel récursif de repaintSatellite
    end for
end function
```

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 pour. 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 plus conséquente a été réalisée à la fin du projet, pour trouver d'éventuels conflits entre les différentes fonctionnalités et au fur et à mesure du déroulement du programme.

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

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

3.2 Les acquis

Les acquis au cours de ce projet ont été divers. Premièrement, nous avons approfondi nos connaissances sur le MVC ainsi que sur la manière de l'utiliser. Des problèmes de conception et notamment 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 découverte, mais nous avons 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 avons pensé que le paradigme MVC était le meilleur moyen pour réaliser le projet, c'est pourquoi nous nous sommes tournés vers cette solution.

3.4 Améliorations futures