Développement d'une application Web de représentation 3d des systèmes planétaire sur le portail exoplanetes de l'Observatoire de Paris

Thomas Bédrine, Département informatique

20 décembre 2021

Remerciements

Avant de présenter le produit de mon stage, je tiens à remercier l'Observatoire de Paris - notamment le Laboratoire d'Études Spatiales et d'Instrumentation en Astrophysique, LESIA - de m'avoir reçu pour ce stage d'assistant ingénieur. Travailler pour un tel organisme est une véritable fierté pour le passionné d'astrophysique que je suis, et j'ai peut-être trouvé ma vocation grâce à l'Observatoire.

Je souhaite ensuite remercier tout particulièrement Françoise Roques, à la tête du projet Exoplanètes, de m'avoir permis d'intégrer son équipe. Elle a été toujours très compréhensive devant mon travail et m'a donné des pistes sérieuses pour l'avancement de l'application sur laquelle j'ai travaillé. Je remercie également Quentin Kral, membre de l'équipe, qui m'a fourni une aide précieuse pour la partie calculatoire entre autres et sans qui je n'aurais jamais pu mettre au point le coeur de mon application.

Mon collègue Ulysse Chosson, récemment diplômé ingénieur et ayant rejoint le projet en même temps que moi, s'est montré très bienveillant et sympathique tout au long du stage. Je l'ai beaucoup cotoyé car nous travaillions toujours dans le même petit groupe au sein du projet, bien que sur des aspects différents. J'ai toutefois eu recours à son travail pour les besoins de mon application, et je le remercie de m'avoir épaulé sur cet aspect du projet. De manière générale, mon stage n'aurait pas été aussi léger sans nos discussions tout aussi diverses qu'intéressantes.

Et enfin mes plus grands remerciements vont à mon tuteur de stage, Pierre-Yves Martin. Réel passionné comme moi - aussi bien en informatique qu'en astrophysique - je me suis retrouvé en lui dès le début et notre collaboration a été extrêmement positive pour moi, comme pour lui je l'espère. Il m'a appris tout ce que vous allez lire dans ce rapport, en étant toujours très patient et pédagogue. Très porté sur la rigueur, il m'a transmis l'envie de faire mon travail le plus finement possible tout en étant libre de tester tous les choix possibles. Pierre-Yves a été un mentor parfait, et je souhaite à toute personne de travailler un jour avec quelqu'un d'aussi impliqué et jovial.

Contents

Introduction

Origine du projet, création de la base de données par Jean Schneider.

Le site exoplanet.eu reste aujourd'hui une référence mondiale pour répertorier et consulter les découvertes liées aux exoplanètes. Dans une volonté de moderniser le site et de le rendre plus accessible au plus grand nombre (étudiants, enseignants, chercheurs...), un remaniement général du site a été entamé quelques mois avant mon arrivée. L'introduction d'une nouvelle gestion de la base de données, avec le format EXODAM - sur lequel a travaillé mon collègue Ulysse - est l'un des points-clés pour centraliser la réception et la consultation des données sur les exoplanètes. Le site doit également se doter d'une nouvelle charte graphique et d'une navigation allégée, grâce au travail de mon tuteur Pierre-Yves.

Un troisième aspect a été envisagé pour le nouveau site : la modélisation en 3D des exosystèmes planétaires. La NASA a réalisé une application similaire, qui reste tout de même très sommaire et assez lourde à manipuler. C'est ainsi qu'a été pensé mon sujet de stage, je devrai réaliser une application permettant de visualiser des exoplanètes quelconques à partir des données fournies par le site exoplanet.eu - application qui sera implantée dans le site lui-même par mon tuteur. L'objectif s'accompagne d'une volonté de réaliser un produit plus performant et plus pédagogue que celui déjà existant de la NASA. C'est avec ces informations en tête que je démarrai mon stage d'assistant ingénieur auprès de Pierre-Yves.

Quels outils choisir ?

J'avais beaucoup échangé avec Pierre-Yves pendant les vacances, à propos de la marche à suivre pour le déroulé du projet et des potentiels outils à utiliser. Nous souhaitons intégrer l'application au site Web, aussi le choix le plus évident est le langage JavaScript, qui apporte le support 3D au format HTML/CSS (entre autres fonctionnalités, le support 3D reste notre priorité).

JavaScript et librairies 3D

Le JS possède du support natif pour la gestion d'environnements 3D, cependant il est très peu accessible et difficilement malléable. Il est préférable de ne pas réinventer la roue, et de se tourner vers des librairies existantes qui proposent des fonctions et classes plus faciles à utiliser. Ainsi, nos deux candidats pour le développement de l'application sont Three et Babylon. La première est la plus large, elle permet une manipulation totale de l'environnement 3D, moyennant une connaissance pointue de la librairie. La seconde est construite à partir de la première, et propose des outils plus intuitifs et regroupant davantage d'options ; cela implique un contrôle moindre sur l'environnement car Babylon gère beaucoup d'interactions en interne, on gagne cependant en facilité de prise en main et d'utilisation.

Sous les conseils de Pierre-Yves, j'ai procédé à une comparaison des deux librairies en réalisant plusieurs mini-applications avec chaque langage. Je n'ai en réalité eu aucun succès avec Three en raison de problèmes d'import, de plus Babylon s'est révélé plus efficace en pratique que Three ne l'est en théorie, grâce aux manipulations de caméras dans l'environnement 3D notamment. Un extrait de mes notes est disponible en annexe pour détailler ce résultat. J'ai donc choisi d'utiliser l'outil Babylon pour développer l'application.

Babylon.js : le moteur et cerveau de l'application

(*Présentation de la librairie en bref.*)

Il est important de noter que de nombreuses décisions concernant le développement ont été faites en faveur de Babylon et non de l'exactitude scientifique requise - il va de soi que nous avons fait le maximum pour concilier les deux, mais quand ce n'était pas possible, c'est la nécessité de se rapprocher du fonctionnement de Babylon qui l'a emporté. Vous en verrez un exemple très concret lors de la gestion du temps au sein de la simulation, où les formules mathématiques complexes se sont heurtées à un fonctionnement hermétique de Babylon.

A posteriori, le choix de Babylon a été grandement valorisé par sa très vaste documentation. Des tutoriels pour tous les niveaux y sont fournis, elle est à jour avec les dernières versions de Babylon et le Git de la librairie est accessible au public (ce qui m'a été très pratique pour l'étude approfondie des fonctions mathématiques que la librairie propose). J'ai toujours au moins un quart de mes

onglets ouverts pour de la documentation Babylon, et je suis très satisfait de la qualité et la quantité qu'elle propose.

Git : espace de travail et planification

Git est un système bien connu des développeurs, il est indispensable à tout projet bien organisé car il sert aussi bien de journal de bord que d'espace de travail. J'avais déjà entendu parler de Git mais je ne m'en étais jusqu'alors servi que très brièvement, laissant le soin à d'autres camarades de projet sa gestion. Ici, je suis le développeur principal de ce projet et j'ai donc dû apprendre à utiliser cet outil pour garder mon code organisé et soigné. Pierre-Yves m'a donc présenté divers concepts : les branches, les 'commit', les 'push', les 'merge'... Nous allions également superviser l'avancement de mon travail via un GitLab associé à l'Observatoire. Nous pourrons y recenser les 'issues', les 'milestones' et les 'merge requests'.

Nous avons alors imaginé un grand découpage du projet en trois étapes :

- première version : réaliser une simulation du système solaire en guise d'exemple, avec une majorité de fonctionnalités.
- deuxième version : étoffer manuellement l'application avec la base de données de l'Observatoire, en créant des systèmes exoplanétaires quelconques.
- troisième version : intégrer complètement l'application au site exoplanet.eu, en automatisant la création de systèmes par l'application.

Je devrai alors travailler ces grandes étapes en les divisant en plus petits blocs (des 'issues'), et je créerai une branche sur le Git pour chaque bloc ainsi traité. Chaque petite avancée dans ces blocs devra être marquée et archivée par un 'commit', et une fois le bloc terminé, nous fusionnerons ma branche de travail avec le 'repository', c'est-à-dire la branche principale du projet.

Choix de l'IDE : Visual Studio Code

VSCode est un environnement de développement que je connais depuis longtemps, il est simple à maîtriser et très personnalisable. C'est également l'IDE de mon tuteur, et il possède du support très pertinent pour le branching de Git - par exemple l'affichage des fichiers modifiés et qui n'ont pas été 'commit'. VSCode est très flexible comme je l'ai mentionné, grâce à ses nombreux modules améliorant la qualité du code et rendant le développement plus agréable. C'est donc un choix naturel vis-à-vis de nos besoins et attentes pour ce projet.

Coder le plus rigoureusement possible : pre-commit, Standard et Prettier

Pierre-Yves a insisté tout au long du stage sur l'importance de conserver un code lisible, cohérent et en règle avec tous les standards de développement. C'est un point auquel je n'étais pas sensible au début de mon stage, et bien que j'ai pu en

voir les bénéfices directs après quelques semaines, il a fallu encadrer mon travail dès le départ pour que je m'y habitue.

Tout d'abord nous avons mis en place 'pre-commit', un intermédiaire entre mon code et le Git. Il analyse mon travail selon des critères précis - longueur des lignes, présence de caractères interdits, non-respect d'une règle du JavaScript... - et m'empêche d'ajouter mon travail à la branche si je ne respecte pas la totalité de ces critères. Pour analyser mon JavaScript et le comparer à des règles existantes, nous nous basons sur le Standard : la norme la plus fine sur l'utilisation du JS en mode strict (*développer les explications sur le mode strict*).

La base de modules de VSCode fournit un autre outil pour m'aider dans ce sens : Prettier. Lorsqu'il est configuré pour respecter le Standard, il corrige automatiquement toutes les parties du code qui ne sont pas conformes lors de la sauvegarde du fichier. Ainsi, le croisement de 'pre-commit' et de Prettier configurés au Standard garantit un code irréprochable sur la forme. Toutefois le fond reste la responsabilité de la personne derrière le clavier : c'est à moi de connaître les usages et les bonnes pratiques du JavaScript. Heureusement, je peux compter sur l'expérience de mon tuteur pour m'améliorer dans ce sens.

Et bien d'autres outils...

Nous avons eu recours à un grand nombre d'outils intermédiaires, ajoutés au fur et à mesure dans le projet. Ils ont tous leur importance, comme 'jest' pour tester l'exactitude des calculs dans le code, cependant ils sont trop nombreux pour être recensés ici. Ils sont toutefois tous listés et documentés dans le fichier 'CONTRIBUTING.md' du projet, que vous trouverez également en annexe (*est-ce pertinent de le mettre en annexe ?*).

Annexes

Notes sur la comparaison Three/Babylon

Deux solutions sont possibles pour l'intégration d'éléments 3D dans le site : three.js et babylon.js, le second étant un package avancé basé sur le premier. Il faut donc tester lequel des deux est le plus adapté à notre problème, en tachant de réaliser une scène basique avec un objet (de préférence un cube) avec des faces de couleur différentes, qui se déplace dans l'espace et qui possède éventuellement un éclairage. Il faut aussi que l'on puisse déplacer la caméra avec la souris.

J'ai donc obtenu le résultat escompté avec babylon, mais je n'ai pas pu dépasser le stade du cube se déplaçant dans l'espace avec three (le problème se posant lors des imports dans la librairie). Jusqu'à résolution du problème de three.js, babylon.js sera mon matériel d'expérimentation.

Avantages de three.js :

- c'est le matériau de base, quoi que l'on décide de faire, on peut le faire précisément avec three.js
- il y a de la doc et énormément d'exemples commentés
- la communauté est plus développée, donc beaucoup de pros pour nous aider si besoin

Inconvénients de three.js :

- demande de placer la totalité de la librarie dans le projet, et de faire beaucoup d'imports
- il faut programmer chaque petit élément, même le plus basique (bouger la caméra avec la souris par exemple)

Avantages de babylon.js :

- regroupe les fonctions de base de three.js pour créer des objets beaucoup plus rapidement
- permet donc notamment de créer une caméra dynamique contrôlable par la souris, sans devoir importer d'autres modules ou librairies
- doc très détaillée, tutoriels complets

Inconvénients de babylon.js :

- comme les petites opérations sont faites automatiquement par la librairie, il est possible que l'on soit tôt ou tard confronté à un problème qui nécessite un paramétrage et/ou une intervention plus ciblée que ce que permet babylon (donc d'avoir recours à three.js pur)
- la communauté est moins nombreuse donc probablement moins de pros capables de nous aider