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

(*Tout texte entre parenthèses et emphasé est une de mes notes pour toi PYM. C'est ce que je compte ajouter avec ton aide, ou modifier selon ce qui te paraît le plus cohérent.*)

(*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 - développé par Ulysse Chosson, ingénieur sur le projet Exoplanet.eu - 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 Martin, ingénieur d'études sur le même projet.

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.

Lexique et terminologie

Termes d'origine	Domaine rattaché	Définition
Sprint	Méthode agile	
Main	Git & GitLab	La version officielle et principale du projet. C'est celle qui est accessible au public (lecture seule).
Branche	Git & GitLab	Une copie sur du projet sur laquelle les développeurs peuvent travailler. Elle peut ensuite être intégrée au main avec l'accord des gestionnaires du Git.
Commit	Git & GitLab	Une sauvegarde locale d'une portion de branche. Plusieurs commits permettent de recenser des modifications régulières sur cette branche.
Push	Git & GitLab	L'action de sauvegarder toutes les modifications d'une branche sur la version en ligne du Git. Un push doit contenir au moins un commit, et peut être suivi par un merge request.
Merge (request)	Git & GitLab	L'action de fusionner une branche avec le main. Le request est la demande de cette action adressée aux gestionnaires.
Issue	Git & GitLab	Une fonctionnalité ou un problème à régler dans le cadre d'un sprint, elle doit être traitée dans une branche.
Milestone	Git & GitLab	Un ensemble d'issues à terminer pour achever un objectif majeur.
V1 / V2 / V3	Spécifique au projet	Ces termes désignent les grandes catégories du projet sous formes de milestones. Chaque V (version) correspond à une ou plusieurs fonctionnalités-clés à mettre en oeuvre avant de passer à la suivante.
Refactoring	Développement	Un processus de réécriture du code pour le rendre plus propre et/ou conforme à un standard de développement particulier. Un refactoring ne doit pas modifier le fonctionnement du programme, seulement la structure de son code.
Mesh (pl: meshes)	Développement 3D	Un ensemble de formes géométriques assemblées pour représenter une forme en 3D.

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 ?*).

Avant le développement du projet : apprentissage et prototype

Mon organisation de travail générale

Tous les outils sont prêts, cependant je débute à peine mon stage et je ne maîtrise pas le JavaScript. Par ailleurs, il faut encore découper le projet en de multiples sous-parties - des 'sprints' - grâce une méthode agile. C'est un processus itératif, qui s'assure que l'application fait exactement ce qui est attendu avant de passer à la suite, orientant ainsi le développement en fonction des obstacles qui se présentent.

En pratique, je choisis une 'issue' à traiter selon un ordre arbitraire (le plus souvent, j'ai choisi des fonctionnalités qui m'intéressaient et/ou qui me semblaient vitales pour le projet). Après avoir prévenu mon tuteur, nous en discutons ensemble s'il faut prévoir un design particulier - nécessaire pour les fonctionnalités majeures. Ensuite, je développe ce design jusqu'à obtenir une première version fonctionnelle. Je la teste - à l'aide d'outils tels que Jest s'il y a le moindre calcul mathématique, entre autres - et je passe en revue ce qui devrait être corrigé et ce qui peut être conservé. Lorsque cette 'issue' est traitée, je note les conséquences qui découlent de ce nouveau morceau de l'application (bugs éventuels, modifications d'autres fonctionnalités, implémentation de nouvelles fonctionnalités...) et je demande à Pierre-Yves de relire mon travail. S'il est irréprochable sur le fond et la forme, le travail est validé et ma branche de développement subit un 'merge', ce qui la rattache à la branche principale. Dans le cas contraire, je corrige autant de fois que nécessaire les coquilles dans mon travail. Je peux ensuite passer à la branche suivante - en pratique, il m'est arrivé de travailler sur plusieurs branches en parallèle, toutefois j'évite au maximum cette pratique car elle a bien trop souvent causé des problèmes de retard de version entre certaines branches. Cette méthode de travail permet d'avoir un projet toujours fonctionnel, même s'il est incomplet.

Il n'est pas rare que terminer une 'issue' en ouvre deux nouvelles - sans compter bien sûr les 'issues' qui nous sont venues en tête plus tard - ainsi nous sommes passés d'à peine 10 'issues' au mois de septembre à près de 50 au mois de décembre. Nous avons donc utilisé l'outil GitLab pour les lister au fur et à mesure, en fermant les 'issues' traitées. C'est également lors de la première réflexion avec mon tuteur que j'ai convenu d'un nom pour cette partie du projet Exoplanet.eu : je travaillerai sur le développement de l'application 'Exo3D'. J'ai alors commencé par développer un Proof Of Concept.

Les débuts d'Exo3D avec un PoC

D'une façon similaire à la phase de sélection de la librairie, je fais mes premiers pas en JavaScript avec des fonctions très simples. On commence donc par placer une sphère au centre de l'environnement graphique, puis on introduit une autre sphère en mouvement autour de la première. La première simulation d'orbite est un succès immédiat, qui a été notamment facilité par mon expérience en la matière. En effet, j'avais déjà réalisé des calculs de trajectoire de corps célestes lors d'un projet de découverte en astrophysique (dans le cadre de l'UV AC20 enseignée à l'UTBM).

L'un de nos objectifs principaux est la représentation des exoplanètes au sein de leur système, aussi il est crucial de bien choisir notre représentation de ces objets. Parmi la liste des premières 'issues' importantes, on retrouve la gestion de la vitesse de simulation, l'aspect des objets spatiaux et la gestion des angles de vue de la scène. Dès la fin de la deuxième semaine de stage, j'avais réussi à modéliser un pseudo-soleil avec une Terre gravitant autour, avec la possibilité d'accélérer (vitesse doublée), ralentir (vitesse réduite de moitié) ou arrêter ce mouvement. Ce mouvement restait toutefois très sommaire, l'objet se téléportant d'un point à l'autre de sa trajectoire.

Nous avions également implémenté deux nouvelles caméras, en plus de celle basique centrée sur le Soleil. La première peut suivre l'unique planète du système, la seconde est libre et n'a pas de cible particulière - elle se déplace librement grâce aux contrôles du clavier. À partir de là, Pierre-Yves m'a montré comment passer nos fichiers .js en fichiers .mjs : ce sont maintenant des modules, et ils sont intégrés comme tels par le navigateur (*davantage d'explications sur l'utilité des modules*). Ceci a donc demandé de nettoyer un peu le code et de l'adapter pour qu'il gère mieux cette nouvelle particularité, ce fut le premier 'refactoring'. C'est également à partir de cet instant que j'ai pu constater une légère amélioration des performances graphiques : utiliser des méthodes de programmation optimisées et strictes avait un impact bien visible sur notre application.

J'ai ensuite amélioré le rendu général du Soleil et de la Terre - le Soleil "émet" maintenant de la lumière - en plus d'ajouter la Lune comme satellite. Le tout n'avait pas encore d'environnement, il n'y avait qu'un fond gris. J'ai donc introduit une texture de fond étoilé de la Voie Lactée. Une question avait été soulevée à ce sujet : devrait-on tenter de changer le fond étoilé pour chaque système, en fonction des étoiles qui devraient visibles depuis cet endroit ? La réponse est non, car nous ne pouvons pas voir certaines étoiles dont la lumière ne nous est pas encore parvenue. Nous ne pourrions donc que deviner et inventer des étoiles selon l'exosystème choisi, le choix de la Voie Lactée a donc

été accepté à l'unanimité y compris en dehors du Système Solaire. Ce genre de problèmes doit toujours être discuté avec Françoise Roques et Quentin Kral, car notre application doit avant tout refléter la réalité des observations astronomiques : on doit donc penser à trouver une alternative cohérente lorsque certaines données nous manquent.

Toutes ces fonctionnalités ont démontré un certain succès de l'application jusque-là, et Pierre-Yves a choisi ce moment pour clore le PoC et débuter pleinement la première version. C'est ainsi que le 6 octobre 2021, j'ai entamé la V1 en passant tout le projet sous forme de classes et d'objets : c'était le deuxième 'refactoring'.

Version 1.0.0 : Simulation du système solaire

L'objectif de ce 'milestone' est d'avoir une application presque aboutie, mais dont les seules données sont celles du système solaire. Le prototype s'orientait déjà dans cette direction pour faciliter le développement. D'ailleurs pour être parfaitement honnête, je n'avais pas conscience de travailler sur un PoC ; cela avait bien été mentionné mais il me semblait être dès le début sur le développement de la V1. C'est Pierre-Yves qui a souligné que ce que j'avais travaillé jusqu'ici était une base solide, mais ce n'était pas encore la V1. Étant donné que nous travaillions sur un prototype, il était acceptable d'être imprécis ou de choisir des raccourcis, ceci afin de s'assurer qu'une solution semble viable avec des paramètres simples. Si en revanche une fonctionnalité était bancale ou difficile travailler, et ce dès cette phase de prototypage, alors cela nous permettait d'économiser du temps en mettant de côté cette option. Il était donc temps de commencer la véritable application.

Conflits entre rigueur scientifique et limites de programmation

Pour entamer la V1, j'ai re-travaillé les calculs de trajectoires. Jusqu'ici, il ne s'agissait que d'une formule basique pour créer un cercle avec une centaine de points. Il nous fallait à partir de maintenant des ellipses quelconques, ainsi qu'un mouvement des planètes conforme à la réalité.

L'un des gros problèmes a été la gestion du temps et d'éventuelles fonctions paramétriques pour le mouvement des planètes. En effet, nous souhaitions obtenir un mouvement continu et fluide visuellement, mais cela s'est avéré incompatible avec le fonctionnement de Babylon. Cette solution aurait nécessité d'utiliser un grand nombre de points (de l'ordre du million pour certains trajectoires très larges), ou d'avoir beaucoup de mémoire disponible pour calculer les trajectoires "en temps réel". De plus, Babylon ne permet de déplacer les objets 3D (les 'meshes') de façon continue, pas de cette manière en tout cas. Le résultat est donc une trajectoire imprécise, beaucoup plus polygonale qu'elliptique, et avec des objets qui se "téléportent" de point en point.

(*Créer schéma sur l'explication de ce type de trajectoires.*)

Il a donc fallu poser un dilemme crucial au sein de ce développement : doit-on tenter de respecter les formules mathématiques et les concepts physiques - quitte à s'éterniser sur le développement - ou bien doit-on renoncer à certaines d'entre elles au profit d'un développement plus efficace ? Nous avons testé en tout quatre solutions situées à des points différents de cette opposition, et attribué un vote à chacune d'entre elles en fonction de critères communs (respect des formules, simplicité de développement, rendu visuel pertinent...) C'est finalement la solution suivante qui a été retenue : le système d'animation interne à Babylon. Celui-ci applique moins bien les formules que les trois autres, cependant il gagnait haut la main sur tous les autres critères : il est facile à mettre en place et à manipuler, les mouvements qu'il permet sont parfaitement fluides, et il peut simuler certains comportements physiques de façon assez propre. Le détail de son fonctionnement est disponible dans le fichier `docs/detailled_doc.md` du projet.

Et bien que j'ai parlé d'inexactitude scientifique dans cette solution, il faut doser cette information : le mouvement simulé des planètes est si proche de leur véritable ellipse qu'il est impossible de détecter la moindre différence à moins de les comparer à échelle microscopique. Ce n'est donc pas la pure réalité physique, mais je pense qu'on peut estimer qu'elle s'en approche à plus de 95%. De réelles différences se manifesteraient en cas de trajectoires plus inhabituelles, comme dans des problèmes à trois corps et autres mouvements chaotiques. J'ai déjà travaillé sur ces cas en deuxième année de Tronc Commun, et je suis certain que mon modèle ne tiendrait pas la route face à ces situations, toutefois mon tuteur m'a assuré que nous ne ferions pas face à des problèmes de ce type.

Huit planètes au lieu d'une : complétion du système

Jusqu'ici, seule la Terre gravitait autour du Soleil. Il a donc fallu commencer à ajouter les autres, et ce fut la partie suivante du développement. Je l'ai peu souligné jusqu'ici, mais la rigueur et les connaissances de Pierre-Yves m'ont été très utiles pour créer un projet très malléable. Ajouter plusieurs planètes à une application prévue pour une seule planète a été un jeu d'enfant, car tout avait été pensé dès le deuxième 'refactoring' et le passage en classes. De

manière générale, une fois qu'une nouvelle fonctionnalité était créée dans son ensemble, elle était alors très facile à modifier ultérieurement et à relier à de nouvelles options. Bien que j'ai été très réticent à l'idée d'être constamment relu par mon tuteur, j'ai vite compris ce qu'il attendait de moi et j'ai pu travailler bien mieux que je ne l'ai jamais fait.

Une fois les sept autres planètes ajoutées au système, il a fallu modifier toutes les autres fonctionnalités qui leurs sont dédiées. Ainsi, de meilleures caméras ont été mises au point pour pouvoir suivre chaque planète plus efficacement. Nous avons aussi ajouté une première version des anneaux de Saturne, bien que le rendu ne soit pas très convaincant à cet instant.

J'ai également passé la totalité du projet en anglais, car jusqu'ici la documentation et les commentaires étaient en français. En effet le site exoplanet.eu a une portée internationale, et il est préférable de choisir l'anglais par défaut en attendant une traduction dans plusieurs langues (c'est une possibilité qui a été évoquée, toutefois cela ne sera pas une de mes missions). Ce fut un bon exercice pour travailler mon niveau d'anglais et m'assurer que je n'avais pas régressé depuis mon entrée en branche - cela faisait plus d'un an que je n'avais pas pratiqué l'anglais couramment, car je m'étais tourné vers d'autres langues après l'obtention de mon BULATS.

Enfin, j'ai créé une véritable interface utilisateur grâce aux composants Bootstrap, remplaçant les boutons placés au jugé sur la page par un panneau de contrôle propre et organisé. La façon dont est présenté ce panneau vise à rappeler un lecteur de vidéo - comme celui de Youtube - avec des boutons Pause/Lecture en bas à gauche et le reste des fonctionnalités sur le reste du panneau. C'est une idée de Pierre-Yves, que j'ai tout de suite trouvé très intéressante : cela ajoute un côté plus moderne et plus intuitif à l'application.

Nous avons donc les huit planètes du système solaire et la majeure partie des outils à implémenter sont présents. Un autre défi s'oppose maintenant à nous : la gestion des échelles dans l'espace.

Système réaliste et système didactique : les échelles à un autre niveau

Pour une première approche de la complétion du système, j'avais arbitrairement choisi des tailles et des distances proches les unes des autres. À présent, il faut proposer des échelles correctes et réalistes, en effet c'est un autre grand objectif du projet Exo3D. J'ai donc récolté les données dont j'ai besoin à propos du système solaire et les ai intégrés à l'application. C'est à cet instant que j'ai pris conscience d'un problème majeur : mon programme fonctionne parfaitement, tellement bien que les distances entre les planètes les rendent impossible à voir. Il y a une comparaison que je donne toujours pour expliquer ceci : pouvez-vous voir une bille d'un centimètre de diamètre à une distance de 117 mètres ? Vous avez alors la réponse à la question "pourquoi ne peut-on pas voir la Terre depuis le Soleil", car les planètes et les étoiles sont ridicules face à l'espace qui se trouvent entre elles. L'un des objectifs majeurs d'Exo3D est donc atteint : montrer aux gens à quel point l'Univers est démesurément grand.

Vous conviendrez qu'il n'est pas très intéressant de regarder un fond étoilé avec quelques trajectoires apparaissant au loin, sans pouvoir examiner les planètes dans leur ensemble. Nous avons une solution à ce problème, c'est la vue didactique. Cet aspect a été pensé dès la création du projet, car nous savions bien à quoi nous attendre à propos des échelles dans l'espace. L'autre grand objectif qui va de pair avec la vue réaliste des planètes, c'est bien la vue didactique : une vue volontairement tronquée d'un système, qui ne respecte pas certaines échelles afin de proposer une observation plus intuitive du système. Il faudra donc idéalement voir toutes les planètes et l'étoile du système, même à très grande distance. Nous avions le choix entre deux échelles à modifier : les distances ou les tailles. Après réflexion, nous avons choisi les tailles, car il est risqué de modifier les trajectoires des objets une fois qu'ils ont été créés : leur animation est définie et immuable dès qu'ils sont créés, il faudrait donc les réécrire partiellement.

Nous souhaitions conserver les proportions des objets entre eux même une fois agrandis, il faudrait donc idéalement choisir un facteur d'agrandissement commun. Quelques tests avec une interface de debug faite par mes soins ont vite soulevé un problème de taille (encore un) : le Soleil est bien plus grand que la moyenne des planètes dans le système solaire. J'ai estimé qu'au-delà d'un facteur d'agrandissement de 100, le Soleil dévorait la trajectoire de Mercure. Le Soleil ne peut donc pas être agrandi au-delà de quelques dizaines de fois sa taille. Peut-être cela suffit-il pour voir les autres planètes ? Malheureusement non, elles restent toutes quasi invisibles avec un facteur de 50.

Il a donc fallu faire un nouveau compromis : les échelles de planètes seraient respectées entre elles, mais elles ne seront pas en phase avec celle de l'étoile. Ceci dit, il s'agit d'un compromis pour une fonctionnalité qui est intrinsèquement irréaliste, donc ce n'est pas vraiment grave. Il faudra juste retenir sa surprise en voyant que Jupiter peut devenir plus grande que le Soleil...

J'ai donc imaginé un calcul simple, qui compare deux planètes aux trajectoires voisines. L'objectif de prendre la plus petite distance qui les sépare et de la diviser par la somme de leur rayon. Le résultat est un facteur d'agrandissemment qui permet aux deux planètes de grossir au point de se toucher. En appliquant cette formule à tout le système et en prenant la plus petite valeur, on s'assure que toutes les planètes sont agrandies sans entrer en collision. Pour éviter que les deux planètes engendrant ce ratio ne se touchent, on descend celui-ci à environ 40 ou 50% de sa valeur. Le résultat pour le système solaire est un facteur d'agrandissement de presque 1600 pour les planètes, et de 55 pour le Soleil. Et je vous laisse juger par vous-même de la qualité d'une telle vue :

(*Insérer capture d'écran du système.*)

Une fois ce problème d'échelle spatiale résolu, je me suis attaqué à l'échelle temporelle. Toutes les planètes ont des vitesses de révolution et de rotation différentes, et pour certaines l'écart est immense. Mercure met environ 87 jours à faire un tour complet autour du Soleil, tandis que Neptune met plus de 160 ans. En choisissant de simuler la période de révolution de Mercure en 5 secondes réelles, cela correspond à 3420 secondes pour Neptune, soit 57 minutes. En prenant ce problème à l'envers et en faisant tourner Neptune en 5 secondes, Mercure se retrouve à faire un tour toutes les 7 millisecondes. Jusqu'ici notre manipulation de la vitesse de simulation ne couvrait que de 0% à 200%, et cela ne suffit pas pour s'adapter à de tels écarts. Peut-être faudrait-il autoriser la vitesse à monter à de très grandes valeurs ? Très mauvaise idée, car nous programmons pour le long terme et nous ne savons pas quelles planètes pourront être découvertes dans le futur, il se pourrait qu'elles aient des périodes encore plus grandes que celles que nous connaissons actuellement. Et les utilisateurs n'ont pas envie de défiler la vitesse d'animation au hasard jusqu'à trouver que 12 200% est la vitesse correcte pour observer Jupiter à un rythme de 5 secondes par tour (je mentionne souvent la période de 5 secondes : c'est en effet la période standard simulée lorsque l'on veut observer le mouvement d'une planète).

La bonne solution est de prendre des planètes de référence temporelle (PRT, TRP en anglais : 'Time-Reference Planet', terme que j'ai créé spécifiquement pour ce projet). Lorsqu'une PRT est choisie, elle est accélérée ou ralentie pour que sa période simulée devienne 5 secondes. Le facteur d'accélération/décélération qui lui est appliqué est uniformisé - on le calcule à partir des écarts de périodes entre toutes les planètes - ainsi on peut aussi l'appliquer à toutes les planètes. Leur période sera donc toujours correctement proportionelle à celle de la PRT. J'ai alors intégré de nouveaux contrôles à l'UI pour permettre de choisir n'importe quelle planète comme PRT, une à la fois. Les contrôles de ralenti (50%), accéléré (200%) et pause (0%) sont toujours présents et s'appliquent par-dessus les facteurs d'accélération d'animation. Par exemple dans le cas de Neptune qui a une période environ 680 fois supérieure à celle de Mercure, si je passe de Mercure (PRT par défaut, vitesse de référence : 100%) à Neptune, celle-ci sera la nouvelle TRP et augmentera donc la vitesse d'animation de tout le système à 68 000%. Si je change de PRT pour choisir la Terre à la place, on reviendra à un ratio moins chaotique de 415%.

Ainsi, il y a toujours de gros écarts entre les planètes car l'échelle temporelle est respectée, cependant il est toujours possible de choisir un point de vue plus ou moins rapide. Avec ces derniers réglages d'échelle, il ne me restait plus qu'à corriger quelques bugs mineurs et je pourrais définitivement clore la V1. Nous sommes alors au mois de décembre, après trois mois de développement sur la V1, j'allais pouvoir commencer la V2. Ou du moins, je le pourrais si je ne comptais pas la documentation. Je dois donc me plonger dans les commentaires du code pour m'assurer que tout est aussi irréprochable que le code lui-même.

Deux niveaux de documentation : JSDoc et documentation détaillée

Je ne l'ai pas mentionné jusqu'ici, mais je documentais mon code depuis le début du stage. J'ai commencé par de simples phrases placées un peu au hasard, pour décrire tout ce qui ne semblait pas évident, puis j'ai commencé à retirer le superflu en ne laissant que des descriptions entières pour un bloc. C'est peu de temps après le 'refactoring' pour passer le projet en classes que j'ai commencé à documenter d'une façon bien spécifique : celle de la JSDoc. La JSDoc est un ensemble de mots-clés et de règles de placement de commentaires qui permettent un accès universel à la documentation d'un morceau de code. En plaçant par exemple tous les membres d'un objet juste avant la classe elle-même, il est possible de les consulter depuis n'importe quel morceau de code utilisant cette classe. J'ai donc fourni cet en-tête à toutes les classes, en plus d'un en-tête pour leurs constructeurs - afin de bien spécifier les paramètres attendus en entrée - et un devant les méthodes. Il est ensuite possible de générer un fichier markdown qui repère certains balises (les mots-clés, leur position, la nature de ce qu'ils documentent...) et fournit une page écrite détaillée qui donne la totalité des informations à savoir sur la classe documentée. Il existe donc un fichier markdown pour chaque module que nous avons créé, fournissant la documentation sous forme de tableaux et de titres hiérarchisés.

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