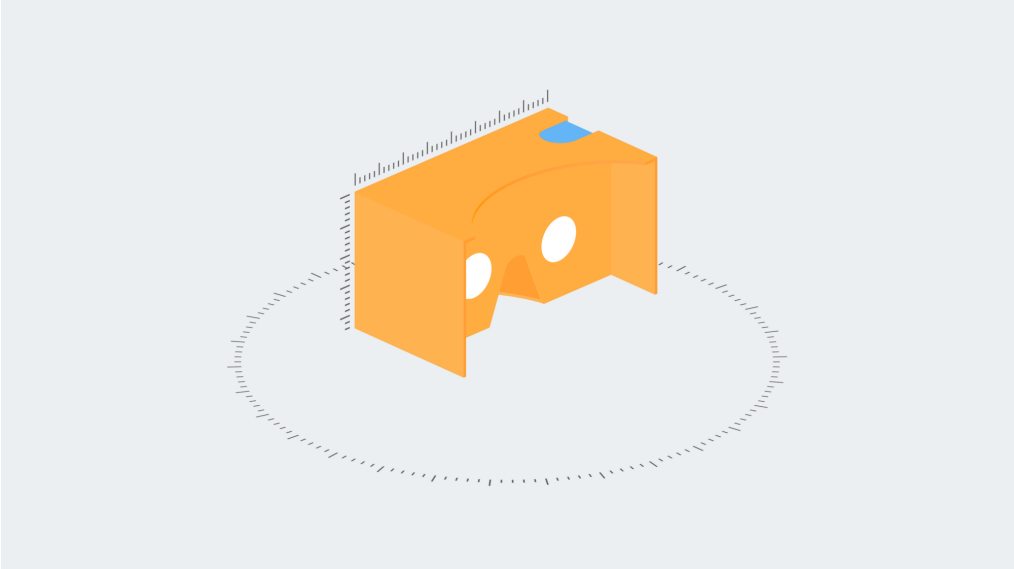
DUBS Victor

GI03 - TN09

03 septembre 2018 au 15 février 2019

RAPPORT DE STAGE

RÉALITÉ VIRTUELLE ET TRAITEMENT DU LANGAGE NATUREL EN ASTRONOMIE



CNRS - Observatoire Astronomique de Strasbourg

11 rue de l'Université

67000 Strasbourg

Suiveur Entreprise : André SCHAAFF

Suiveur UTC : Ghislaine GAYRAUD

utc Université de Technologie Compiègne

CENTRE DE DONNÉES ASTRONOMIQUES DE STRASBOURG

Observatoire astronomique de Strasbourg

I. REMERCIEMENTS

Je souhaite remercier l'ensemble des membres de l'Observatoire astronomique de Strasbourg qui m'ont accueilli et qui m'ont permis de réaliser ce stage. Tout d'abord je souhaite en particulier remercier André SCHAAFF, qui m'a supervisé pendant toute la durée du stage et qui m'a aidé à de nombreuses reprises. Également François-Xavier PINEAU et Sébastien DERRIERE avec qui j'ai pu partager un bureau.

Je remercie toutes les personnes ayant assisté à ma soutenance blanche et qui ont pu m'aider à me préparer.

Je remercie aussi M. Pierre-Alain DUC, directeur de l'observatoire astronomique de Strasbourg ainsi que M. Mark ALLEN, directeur du CDS de m'avoir permis d'effectuer ce stage.

Sommaire

II. RÉSUMÉ TECHNIQUE

Le CDS propose régulièrement des sujets de stage orientés R&D, notamment autour des grandes masses de données et des technologies permettant de les stocker, les visualiser ou d'y accéder. Le stage avait pour but de créer une plus-value sur les différents développements déjà réalisés au sein du CDS sur les sujets de compréhension du langage naturel et de réalité virtuelle. L'objectif est alors d'intégrer de la reconnaissance vocale, d'un côté sur un Chatbot utilisant déjà le traitement du langage naturel en version texte, et de l'autre de trouver une solution pour s'affranchir des moyens de contrôle habituels en réalité virtuelle grâce à la reconnaissance vocale. De nombreuses contraintes sont apparus dû à DialogFlow qui n'est pas encore dans une situation stable, il a été en revanche possible de réaliser des développements intéressants en utilisant uniquement des Google Cardboards. Dans ce rapport sera d'abord présenté le contexte du stage, puis le sujet et enfin les différentes étapes de la réalisation.

III. Présentation de l'entreprise

1. Observatoire astronomique de Strasbourg

L'Observatoire astronomique de Strasbourg a été fondé en 1881, c'est un établissement conjuguant recherche et enseignement situé sur le campus de l'Université de Strasbourg.

Illustration 1: Photo du bâtiment principal de l'observatoire : la coupole

L'Observatoire est composé de trois bâtiments historiques tous reliés entre eux par des couloirs couverts qui permettaient à l'époque aux chercheurs de se protéger des intempéries.

Lorsque l'observation au sol a commencé à montrer ses limites, l'observatoire fut contraint de s'adapter : Deux pôles se sont formés, le premier consacré à la recherche astronomique nommé *GALHECOS* et le second orienté sur l'archivage des données, donnant lieu par la suite au Centre des Données Stellaires qui deviendra finalement le Centre de Données astronomiques de Strasbourg (CDS), c'est dans cette entité qu'a lieu mon stage.

L’Observatoire est une unité mixte de recherche du Centre National de la Recherche Scientifique et de l’Université de Strasbourg, il participe notamment à la formation du parcours astrophysique du Master de physique.

L’Observatoire est amené à collaborer avec un grand nombre d'organismes tels que le l’IVOA (International Virtual Observatory Alliance) dont il est l'un des acteurs majeurs. L’IVOA est une alliance dont l’activité principale est le développement de standards techniques. Ces standards permettent de déployer des outils et des systèmes nécessaires au développement mais aussi l’utilisation internationale des archives astronomiques.

2. CNRS

Le 19 octobre 1939, le Président de la République Française Albert Lebrun acheva la création du Centre National de la Recherche Scientifique (CNRS) par la signature d’un décret-loi. A cette époque, le CNRS eut essentiellement pour but de réunir les moyens de recherche disponibles et de concentrer les recherches sur des solutions militaires tout en mettant les savants à l’abri du conflit armé. A la fin de la Seconde Guerre Mondiale, le CNRS fut réorganisé et commença alors à concentrer ses efforts sur la recherche fondamentale. C’est à partir de 1966 que se met en place l’idée d’unités associées, qui ne sont autres que des laboratoires universitaires liés par contrat au CNRS. Ce dernier les soutiendra grâce à ses moyens humains et financiers.

Au fil des ans, la notion d’unité associée évoluera et les structures opérationnelles de recherche se diversifieront. Parmi elles, les unités mixtes de recherche (UMR) émergent vers les années 1990. Elles représentent l’association contractuelle d’un ou plusieurs laboratoires de recherche d’un établissement d’enseignement supérieur ou d’un organisme de recherche avec le CNRS. Une UMR dispose d’un budget qui lui est propre, d’un personnel affecté à la fois par le CNRS et l’établissement d’enseignement supérieur (ainsi que des contractuels) et est administré par un directeur et un conseil de laboratoire qui définit sa stratégie de recherche de manière autonome.

3. Centre de données astronomiques de Strasbourg

Le CDS a pour but de récolter et de traiter les données astronomiques, notamment dans l'optique de créer une plus-value sur ces données. Trois pôles existent au sein du CDS : les ingénieurs en informatique, les documentalistes et enfin les astronomes. Toutes ces personnes sont liées : les développeurs créent des programmes pour la base, les documentalistes remplissent les bases et enfin les astronomes aident à vérifier les connaissances utilisées.

Le CDS présente plusieurs services :

SIMBAD

Ce service est une base de données fournissant des données basiques, des mesures, des bibliographies ainsi que des « cross-identification[1] » pour les objets astronomiques situés en dehors de notre système solaire. Il est ainsi possible de retrouver ces objets grâce à leurs noms ou identifiants, ou leurs coordonnées. Simbad repose notamment sur un besoin en astronomie : Chaque personne peut choisir l'identifiant de son choix lors de l'observation d'un objet astronomique, il existe alors de multiples identifiants pour le même objet, Simbad permet donc de consulter les différents identifiants d'un même objet. Le regroupement de ces identifiants est une des tâches des documentalistes au sein du CDS.

1 Identification croisée. Consiste à identifier dans deux catalogues de sources astronomiques celles qui sont détectées dans les deux, et à les relier.

Identifiers (32) :

An access of full data is available using the icon Vizier near the identifier of the catalogue

M 31	IRC +40013
2C 56	K79 1C
DA 21	LEDA 2557
2FGL J0042.5+4114	2MASX J00424433+4116074
3FGL J0042.5+4117	2MAXI J0043+412
GIN 801	MCG+07-02-016
IRAS F00400+4059	NAME Andromeda
IRAS 00400+4059	NAME Andromeda Galaxy

Illustration 2: Différents identifiants de l'objet M31 sur Simbad

Le 8 Janvier 2019 Simbad contenait 10 230 948 objets et 34 191 473 identifiants

VIZIER

C’est une collection de catalogues astronomiques et de tables publiées dans les journaux académiques. Elle a été créée en 1992 et contrairement à Simbad, elle ne contient pas directement des données brutes mais des catalogues. Un catalogue est un regroupement de données qui ont été récupérées par des outils au sol ou dans l’espace, ces données sont ainsi rangées sous forme de tables. Chaque catalogue est en soi une petite base de données sachant que le contenu de chaque catalogue peut contenir de nombreuses tables avec un contenu pouvant varier d’un catalogue à l’autre.

ALADIN

C'est le service le plus intuitif du CDS, il permet de visualiser des images HiPS[2] (Hierarchical Progressive Survey). L'idée est d'afficher des relevés d'image pris par des instruments de mesure et de permettre de zoomer avec une grande précision : au fil du zoom de nouvelles images sont générées pour donner l'impression d'une image satellite de très haute qualité. Ainsi il est possible de zoomer et d'avoir des informations sur les objets observés, il est également possible de changer le type de lumière de ce qui est observé. Il se décline notamment en deux outils : Aladin Desktop et Aladin Lite. Le premier est une version logicielle assez lourde mais qui permet de faire des traitements plus complexes, comme par exemple comparer plusieurs vues d'un même objet sous différentes longueurs d'ondes. Le second est un service accessible sur une page web, utilisable comme widget n'importe où. C'est un outil de visualisation extrêmement puissant qui est utilisable par le grand public.

SESAME

Sesame est un petit service permettant de déterminer sur quel service du CDS un nom d'objet astronomique est présent. Cela est notamment utile pour ne pas multiplier des requêtes inutilement sur chaque service.

2 Mécanisme de pavage hiérarchique permettant d'accéder, de visualiser et de parcourir de manière transparente les données d'images et de catalogues

IV. LA MISSION

1. SUJET

L'objectif du stage est principalement la valorisation des travaux réalisés sur la réalité virtuelle/augmentée ainsi que le traitement automatique du langage naturel. Il y a donc en réalité trois objectifs :

- Reprendre le Chatbot[3] utilisant DialogFlow développé par le stagiaire précédent pour l'améliorer en mettant notamment en place une reconnaissance vocale et obtenir une architecture plus pérenne.

- Développer une application de réalité virtuelle utilisant les Google Cardboard qui permet de visualiser des relevés d'image sur une vue 360°, le focus du regard permettrait ensuite d'obtenir des informations sur les objets observés.

- Implémenter le système du Chatbot à l'application Cardboard : Il doit être possible de contrôler l'affichage et les actions de l'application via la voix, toujours en utilisant DialogFlow.

Il n'y a pas de cahier des charges réellement défini, le développement peut s'adapter et prendre une différente direction en fonction des technologies trouvées et des possibilités

3 Programme simulant une conversation avec un humain

2. PLANNING

Il n'y pas eu de planning défini lors du stage, l'objectif était de partir sur une piste de développement et d'ensuite étudier dans quelle direction la faire évoluer.

Ci-dessous le planning réalisé :

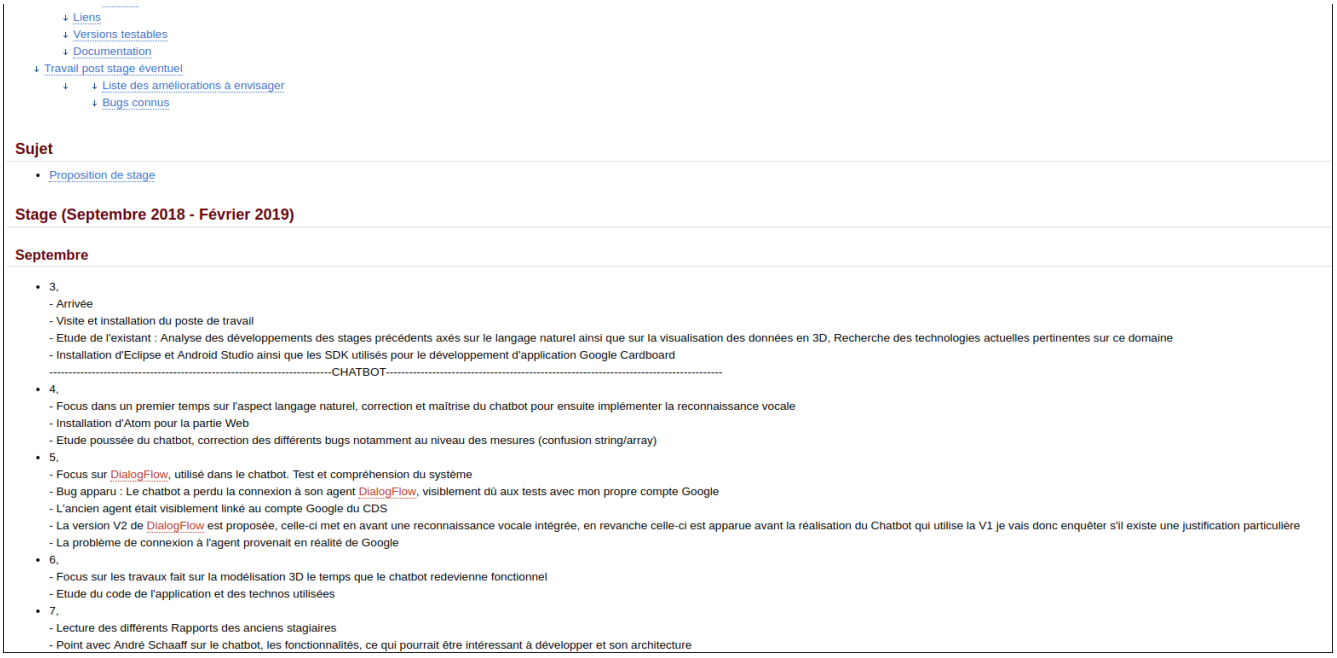
3/09/2018	Début du stage et découverte de l'observatoire
	Documentation sur l'existant et les technologies
14/09/2018	Début du développement sur le Chatbot
8/10/2018	Changement d'orientation du stage vers l'aspect réalité virtuelle
1/11/2018	Affichage d'une image en réalité virtuelle, et utilisation d'un Hotspot permettant d'afficher une interface
22/11/2018	Appel serveur fonctionnel
15/12/2018	Documentation sur les conversions de coordonnées, et création d'une zone de focus n'utilisant plus les hotspots
22/12/2018 – 2/01/2019	Vacances
15/01/2019	Conversion des coordonnées corrigées et fonctionnelles, application fonctionnelle
4/02/2019	Entretien Skype avec suiveur UTC ainsi qu'André Schaaff
5/02/2019	Soutenance blanche au sein du CDS
15/02/2019	Fin du stage

3. CONTRIBUTIONS

Au départ du stage existait notamment l'application Chatbot réalisée par Alexis Guyot, celle-ci était dans un état fonctionnel et faisait l'œuvre de démonstration lors de présentations.

Le travail a majoritairement été réalisé seul, André Schaaff venait régulièrement faire un point, donner des pistes pour débloquer un problème et réfléchir sur les directions à prendre sur le projet. J'ai par moment reçu un peu d'aide de la part de mon voisin de bureau François-Xavier notamment sur les conversions de coordonnées astronomiques.

Un site faisant office de carnet de bord est utilisé pour suivre l'avancement du stage, il est nécessaire d'y noter les différentes réalisations pour avoir une trace écrite de l'avancement du stage. Ce site s'appelle Twiki et est disponible au sein de l'intranet du CDS.



Liens
Versions testables
Documentation
Travail post stage éventuel
Liste des améliorations à envisager
Bugs connus

Sujet

- Proposition de stage

Stage (Septembre 2018 - Février 2019)

Septembre

- 3,
  - Arrivée
  - Visite et installation du poste de travail
  - Etude de l'existant : Analyse des développements des stages précédents axés sur le langage naturel ainsi que sur la visualisation des données en 3D, Recherche des technologies actuelles pertinentes sur ce domaine
  - Installation d'Eclipse et Android Studio ainsi que les SDK utilisés pour le développement d'application Google Cardboard

  ----------CHATBOT----------
- 4,
  - Focus dans un premier temps sur l'aspect langage naturel, correction et maîtrise du chatbot pour ensuite implémenter la reconnaissance vocale
  - Installation d'Atom pour la partie Web
  - Etude poussée du chatbot, correction des différents bugs notamment au niveau des mesures (confusion string/array)
- 5,
  - Focus sur DialogFlow, utilisé dans le chatbot. Test et compréhension du système
  - Bug apparu : Le chatbot a perdu la connexion à son agent DialogFlow, visiblement dû aux tests avec mon propre compte Google
  - L'ancien agent était visiblement linké au compte Google du CDS
  - La version V2 de DialogFlow est proposée, celle-ci met en avant une reconnaissance vocale intégrée, en revanche celle-ci est apparue avant la réalisation du Chatbot qui utilise la V1 je vais donc enquêter s'il existe une justification particulière
  - La problème de connexion à l'agent provenait en réalité de Google
- 6,
  - Focus sur les travaux fait sur la modélisation 3D le temps que le chatbot redevienne fonctionnel
  - Etude du code de l'application et des technos utilisées
- 7,
  - Lecture des différents Rapports des anciens stagiaires
  - Point avec André Schaaff sur le chatbot, les fonctionnalités, ce qui pourrait être intéressant à développer et son architecture

Illustration 3: Carnet de bord Twiki

4. OUTILS ET TECHNOLOGIE

Ce projet a été réalisé avec une méthode se rapprochant du développement agile, des périodes de développements sont réalisées qui peuvent être également liées à des veilles technologiques. Des points sont fait régulièrement pour faire un point sur les problèmes et les avancées du projet.

La première partie du développement était consacré au développement Web, on retrouve donc les principaux langages tels que HTML, CSS et JavaScript.

HTML, CSS et JavaScript, Ces trois langages sont la base du développement web et seront donc naturellement utilisés dans ce projet. JavaScript en particulier parce que le précédent stagiaire n'a utilisé que ce langage pour le back-end du chatbot.

NodeJs est une plateforme logicielle permettant de créer un environnement côté serveur, on peut alors diviser l'application en une partie client et une partie serveur. Le côté client a pour rôle de modifier la page web et donc d'interagir avec l'utilisateur, le côté serveur dans notre situation a pour rôle de gérer l'aspect traitement du langage naturel. NodeJS permet également d'effectuer des entrées/sorties de manière asynchrone ce qui est extrêmement utile pour un Chatbot qui implique de nombreuses interactions avec l'utilisateur

Atom est un éditeur de texte libre, utile notamment pour ses nombreux plugins très utiles, j'ai également choisi cet éditeur par préférence personnelle.

DialogFlow est un outil appartenant à Google permettant de traiter des requêtes en langage naturel notamment grâce à une API[4] qui peut être incorporée dans une application.

4 Application Programming Interface – Ensemble de fonctions qui facilitent l'accès au service d'une application

Gitlab est un service d'hébergement et de gestion de développement de logiciels. Celui-ci a permis de diffuser les différentes versions des projets, de sauvegarder nos fichiers et de conserver une trace du développement. C'est un service qui est notamment libre et open-source. Celui-ci est hébergé en local au sein du CDS.

CARDBOARD

Unity est un moteur de jeu présent sur plusieurs plateforme, sa particularité est qu'il utilise 2 langages : C# et JavaScript. Il est possible de développer des applications Android, iOS et PC. L'asset store est également un aspect important, ce « store » regroupe tous les assets[5] créés et partagés, ceux-ci étant gratuits ou non. Les assets sont souvent d'excellentes qualités et permettent de récupérer de grosses parties de code pour n'importe quel type de développement.

C# est un langage de programmation orienté objet développé en 2002 par Microsoft. Il est très similaire au langage Java avec lequel j'ai une certaine expérience. Il est nécessaire pour développer des scripts sur Unity.

Google Cardboards est un casque de réalité virtuelle créé par Google et dévoilé en 2014 qui utilise notre smartphone, il est fait de carton et de deux lentilles ce qui en fait du matériel très peu coûteux. Il est également possible de construire sa propre Cardboard grâce aux patrons disponible sur Internet. Ce choix a été fait pour limiter l'aspect trop coûteux des casques de réalité virtuelle

5 Représentation d'un objet qui peut être utilisé au sein d'un projet, cela peut être n'importe quel type de fichier supporté par Unity

ADQL (Astronomical Data Query Language) est un langage similaire à SQL mais qui comprend des fonctions géométriques. Il est utilisé par l'observatoire virtuel pour lancer des requêtes aux services de l'IVOA (International Virtual Observatory Alliance).

5. PRISE DE RECUL

Le travail réalisé lors de ce stage est lié de très près à des développements en cours sur des technologies récentes : le principal exemple étant DialogFlow. Sa seconde version étant sortie récemment, les compatibilités avec les autres outils ne sont plus assurés. Il est possible que dans un futur proche un asset pour DialogFlow devienne disponible sur Unity, il est nécessaire pour ses projets de s'intéresser de près aux nouvelles technologies et mises à jour. La réalité virtuelle est également une technologie qui se développe pour bientôt devenir accessible au grand public, les outils qui y sont liés sont en train d'évoluer.

L'application réalisée peut potentiellement devenir obsolète du jour au lendemain suite à une mise à jour et il serait nécessaire de vérifier si celle-ci reste utilisable. Ce fut notamment un des enjeux du développement de faire une application disponible pour un grand nombre et pouvant perdurer un minimum dans le temps. L'application sera peut-être reprise par un autre stagiaire pour explorer de nouvelles possibilités.

V. RÉALISATION

1. CHATBOT

La première étape du stage fut de travailler sur le Chatbot et le traitement du langage naturel. Il était donc nécessaire de tester le Chatbot développé, quelques bugs étaient notamment présents ce qui ne correspondait pas à la description d'un programme fini. Le développement n'était en réalité pas la dernière version en revanche il était tout de même possible de saisir l'architecture du programme.

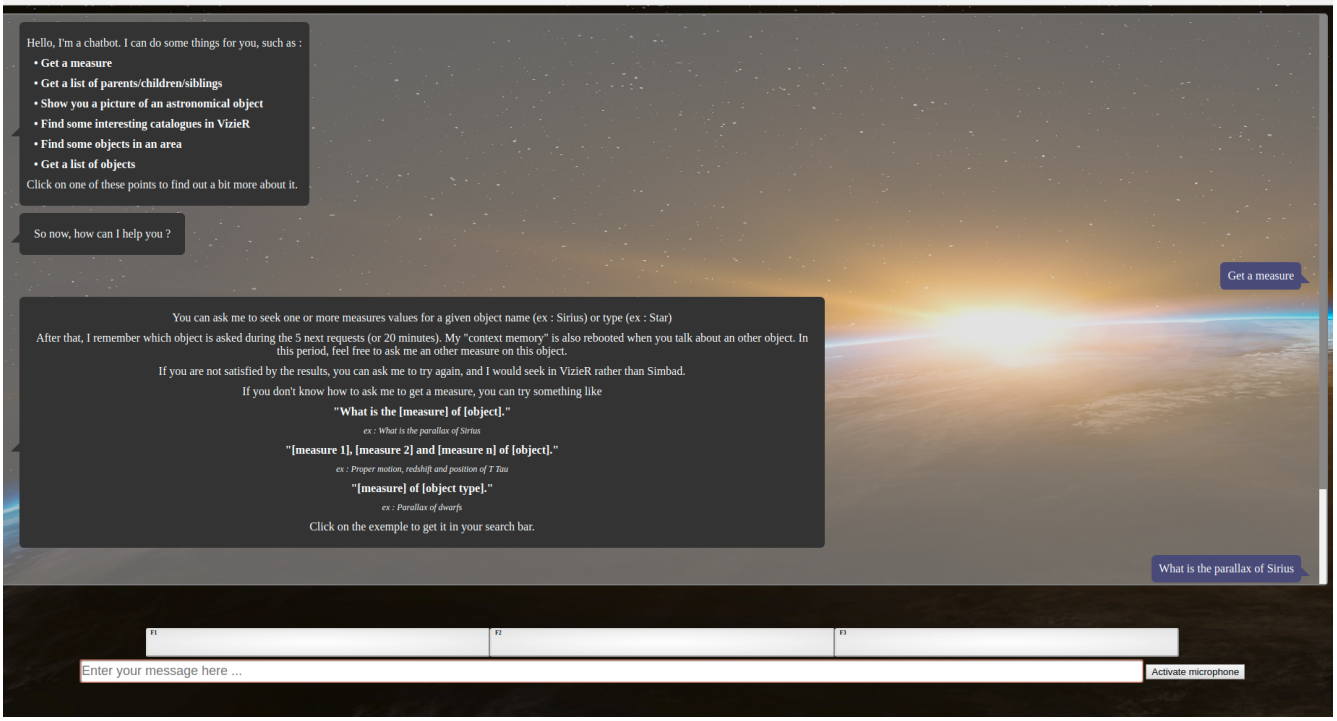


Illustration 4: Affichage du Chatbot

Ci-dessus la fenêtre du Chatbot, on écrit donc les messages que l'on veut envoyer sur la barre en bas de l'écran. Des propositions de requêtes sont proposées et cliquable par l'utilisateur notamment pour les premières utilisations du Chatbot et ainsi mieux comprendre son fonctionnement. En réalité il s'agit d'un moyen abordable d'interroger les services du CDS, les requêtes doivent donc être orientées vers des données susceptibles d'être mises à disposition dans une des bases de données.

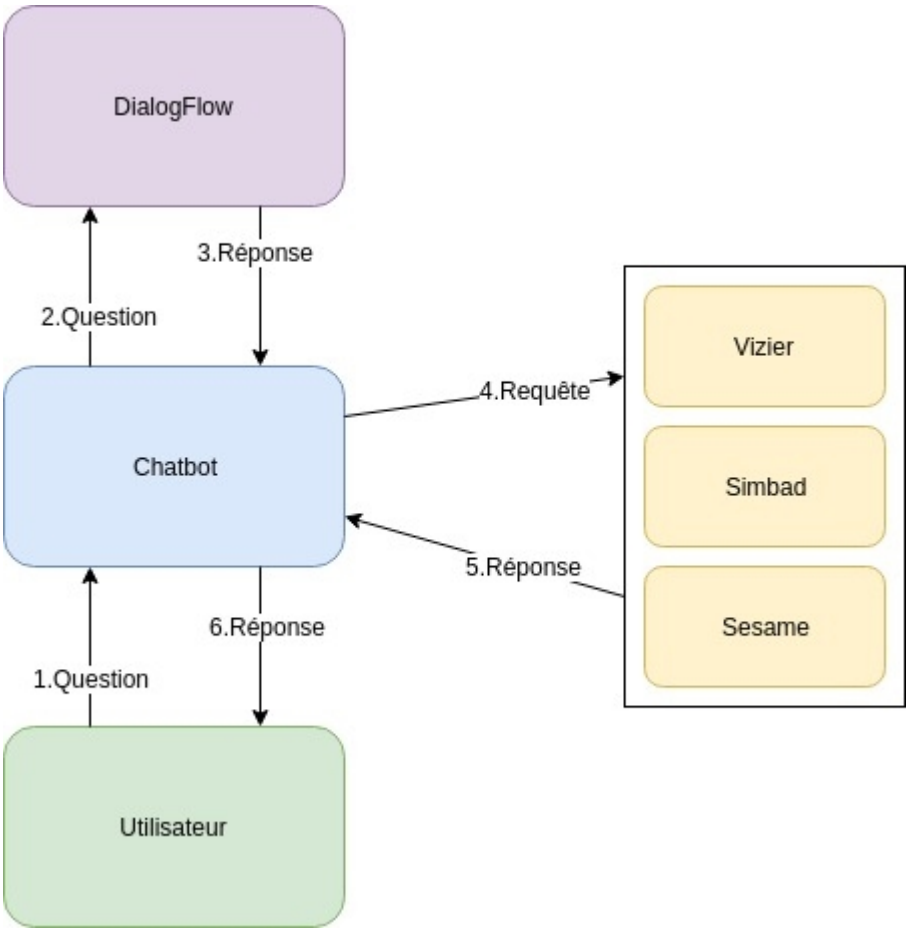


Illustration 5: Schéma de fonctionnement du Chatbot

Le Chatbot est donc une application Web utilisant principalement du JavaScript, avec comme unique framework[6] jQuery. L'architecture interne du Chatbot suit cette logique : un module regroupant les actions sur l'interface, un autre pour les requêtes vers DialogFlow, un pour le traitement des demandes, les calculs et les requêtes sur les services du CDS et un dernier pour les fonctions outils utilisables sur différents modules.

6 Architecture logicielle permettant de faciliter le développement

DialogFlow

DialogFlow était connu à l'origine sous le nom de API.AI, suite au rachat par Google il a été renommé par son nom actuel en 2017.

DialogFlow est donc basé sur le « Natural Language Processing » ou **NLP** qui repose lui-même sur le « machine learning ». Il permet de créer un **agent**, ce dernier est un module de « Natural Language Understanding » ou **NLU** qui sert à transformer une requête en données utilisables.

Il utilise des **intent** (soit des intentions), le but étant de trouver une intention qui correspond à la requête envoyé. L'agent doit être « entrainé » notamment en lui fournissant un maximum de requête type correspondant à cette intention.

Une **entity** (soit entité) correspond à un type de données, ce qui est utile notamment pour savoir quelles données sont demandées à l'application.

Fulfilment est un service permettant de créer du code personnalisé pour définir des actions sur les intentions avec des fonctions du cloud. Ce système n'est pas utilisé dans le cadre de l'application, les traitements étant trop complexes.

Le **contexte** permet de conserver des variables sur un certain nombre de requêtes.

Exemple de requête

Comme exemple nous allons prendre une question de l'utilisateur : On veut obtenir la parallaxe de l'objet astronomique nommé Sirius. On pose alors la question « What is the parallax of Sirius ? », la requête est alors directement envoyée vers DialogFlow. Ce dernier va alors traiter la phrase pour déterminer son « intention », l'intention est alors renvoyé avec les éléments voulus : Ici, on obtiendra comment intention *get_measure*, les entités seront alors *meas* : « parallax » et *oid* : « Sirius ».

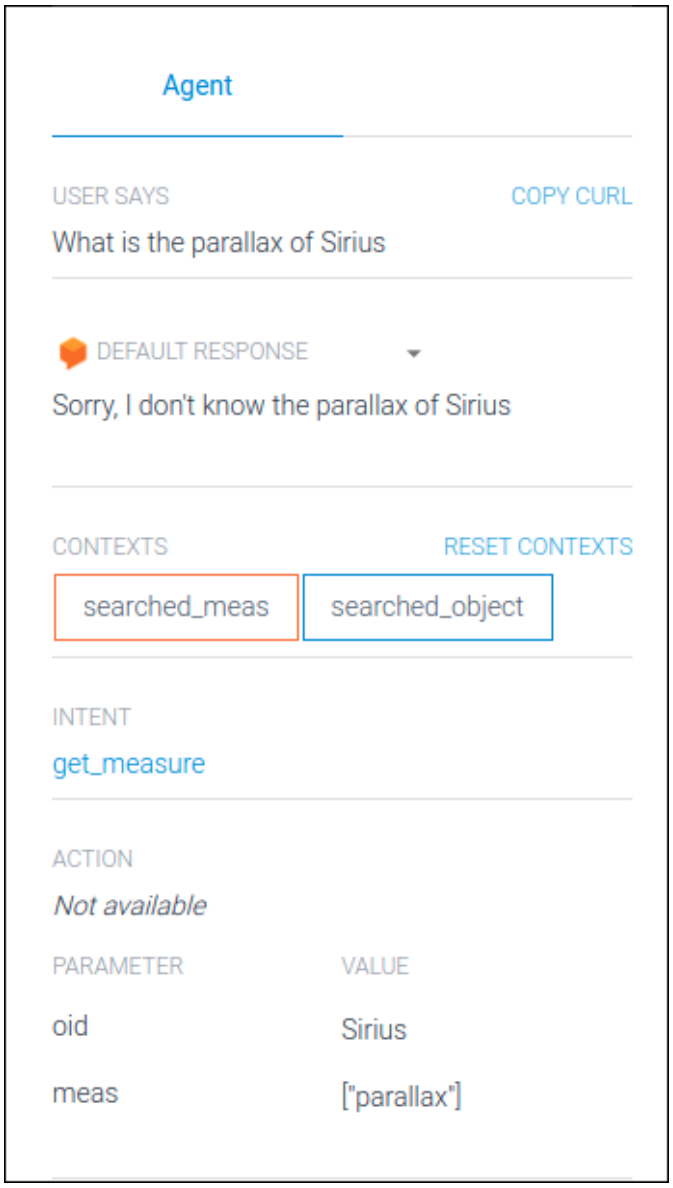


Illustration 6: Résultat de la console de DialogFlow

Le résultat ci-contre dans le cas de la requête directement tapée dans la console de DialogFlow.

On remarque bien que l'intent trouvé est *get_measure* et que les paramètres sont les bons, on peut également noter que *meas* correspond à un tableau, il est effectivement possible de chercher plusieurs mesures au sein d'une même requête. En revanche ici il n'y a pas d'action et DialogFlow ne sait pas quoi répondre à cette demande, effectivement il est nécessaire via le programme de récupérer ces informations pour nous même les traiter et faire ainsi les requêtes aux serveurs du CDS.

Un détail est la catégorie Contexte qui permet de créer un contexte lors d'une requête, on pourra ensuite poser une autre question de façon implicite comme par exemple : «What is its position ? ». *meas* sera bien « position » et *oid* restera « Sirius ». Un contexte perdure un nombre défini de requête, suite à ça il est effacé.

On retourne alors ces 3 informations au Chatbot qui va déterminer quel service devra être utilisé pour récupérer les informations.

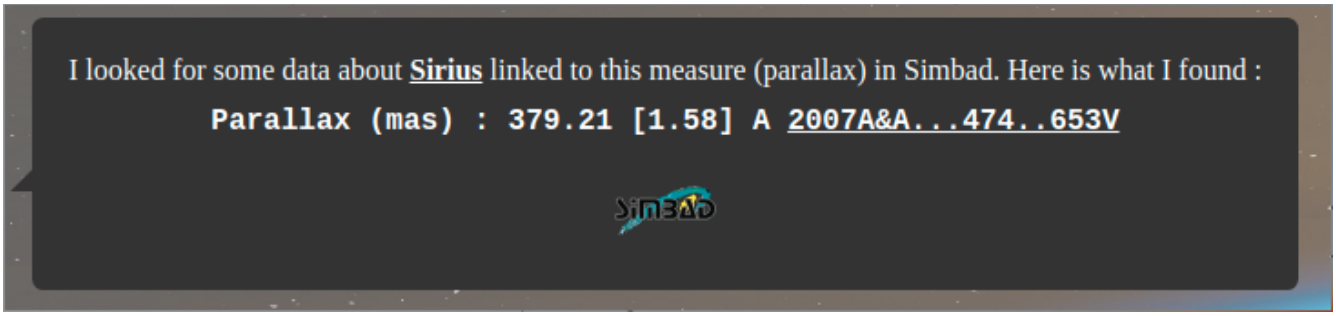


Illustration 7: Résultat obtenu sur le Chatbot

La réponse a été trouvée grâce à Simbad, on obtient alors la parallaxe avec notamment un lien vers la page donnant ce résultat.

AMÉLIORATIONS

Le Chatbot est aujourd'hui assez complet et utilisable, en revanche il reste de nombreuses pistes à explorer :

1. Utilisation d'un système de reconnaissance vocale.

2. Passer la version d'API de DialogFlow utilisé de la v1 à la v2, ce qui est notamment requis pour le traitement des messages vocaux par DialogFlow. Aujourd'hui ce changement est fait partiellement, les messages textes sont envoyés en v2 et pourront perdurer lorsque la v1 ne sera plus supportée.

3. Finaliser la transition vers NodeJS pour l'application, notamment pour obtenir une architecture pérenne avec un meilleur découpage client/serveur mais également car cela est requis par la v2 de DialogFlow pour l'envoi des messages vocaux.

Le principal développement effectué pour le Chatbot était la conversion vers une architecture NodeJS pour pouvoir mettre en place la reconnaissance vocale. Pour pouvoir mettre ça en place il était nécessaire de séparer la partie serveur de la partie client.

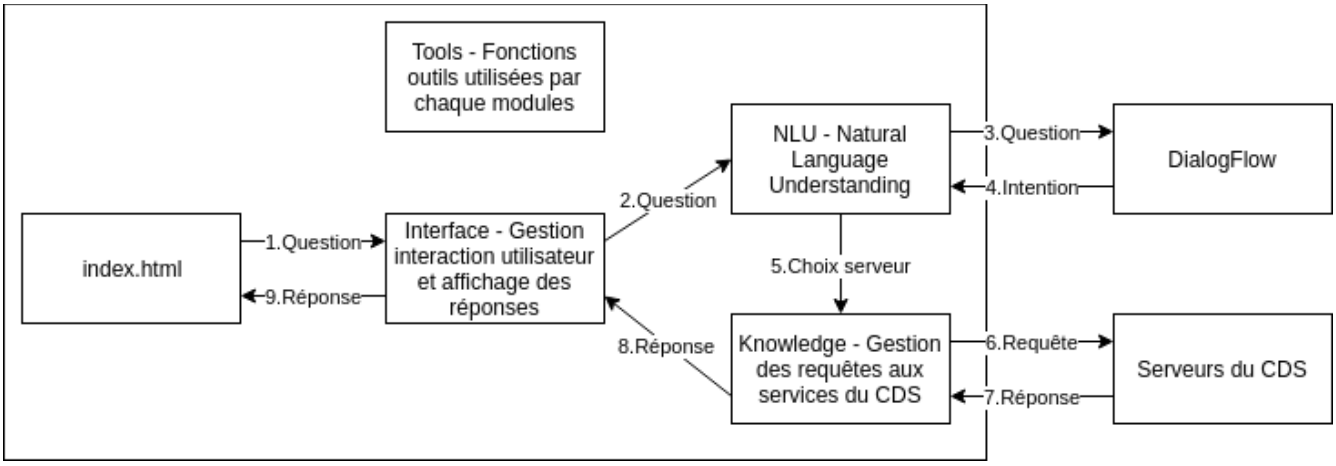


Illustration 8: Schéma détaillé des modules du Chatbot

Ci-dessus le Chatbot tel qu'il est présenté d'après son architecture, d'après celle-ci il aurait été de créer cette découpe :

Partie client : index.html + Interface + Une partie du module Tools avec les fonctions utilisées dans cette partie client.

Partie serveur : NLU + Knowledge + Une partie du module Tools avec les fonctions utilisées dans cette partie serveur.

La réalité est en fait plus complexe car le développement qui a été réalisé ne respecte pas vraiment cette architecture, chaque module peut appeler n'importe quel autre module dans de nombreuses fonctions : Un exemple est une fonction du module knowledge qui va appeler une fonction tools, cette dernière doit ensuite elle-même appeler une fonction de knowledge pour donner un résultat.

Il est donc nécessaire de reprendre entièrement l'application pour pouvoir utiliser convenablement NodeJS. Étant donné la nature relativement longue et peu intéressante de la tâche il m'a été proposé de passer à l'aspect réalité virtuelle du stage.

DIFFICULTÉS

- DialogFlow dépend des serveurs de Google, ce service est relativement jeune et en constant développement, il arrive donc régulièrement que les serveurs ne soient pas fonctionnels, ainsi le Chatbot devient inutilisable par moments.
- NodeJS est une architecture particulière reposant sur des modules, chaque module appelant d'autres modules cela engendre une application plus imposante et moins rapide.
- DialogFlow dans sa v2 implique un temps de traitement plus élevé que la v1 pour les requêtes texte. Il a été alors trouvé comme solution de continuer à utiliser la v1 uniquement pour les requêtes texte, conservant ainsi une certaine fluidité dans la conversation avec le Chatbot. Cette solution n'est en revanche pas définitive car la v1 deviendra sûrement obsolète et ne sera plus supportée dans le futur.
- API utilisée pour la reconnaissance vocale très complexe, présentant une énorme quantité de fonctions peu utiles, rendant le développement plus complexe.
- Travail long et fastidieux pour la migration vers NodeJS, les fonctions s'appellent entre fichier sans propre encapsulation, il est donc nécessaire de parcourir chaque fichier pour retrouver les appels de chaque fonction pour les exporter et importer convenablement comme le requiert NodeJS.

2. CARDBOARD

CHOIX DU SUPPORT

Mon premier choix de support de développement fut Android Studio, ayant déjà développé une application Android au cours de mes études j'étais assez familier de cet outil. Un exemple d'application était déjà disponible et qui permet de visualiser une image avec un Google Cardboard à 360°, il était alors notamment possible de tester l'image 360° fournie par le CDS. La difficulté rencontrée fut de rajouter un « calque » permettant d'afficher des infos sur les éléments observés, en effet Android Studio ne gère pas plusieurs couches d'affichage sans utiliser des traitements bas niveau de type OpenGL.

Google préconisait notamment sur son site l'utilisation de Unity pour le développement d'applications utilisant le SDK de réalité virtuelle de Google. Unity étant notamment un moteur de jeu donnant une interface facilitant l'affichage d'éléments dans l'espace, il est alors facile de créer les « calques » permettant d'afficher les informations. En revanche il n'existe pas de moyen natif de créer une image 360° et l'afficher en RV comme désiré. Suite à des recherches il existe de nombreux assets permettant de faire ce traitement, beaucoup sont payantes mais je me dirige naturellement vers celles étant gratuites en priorité.

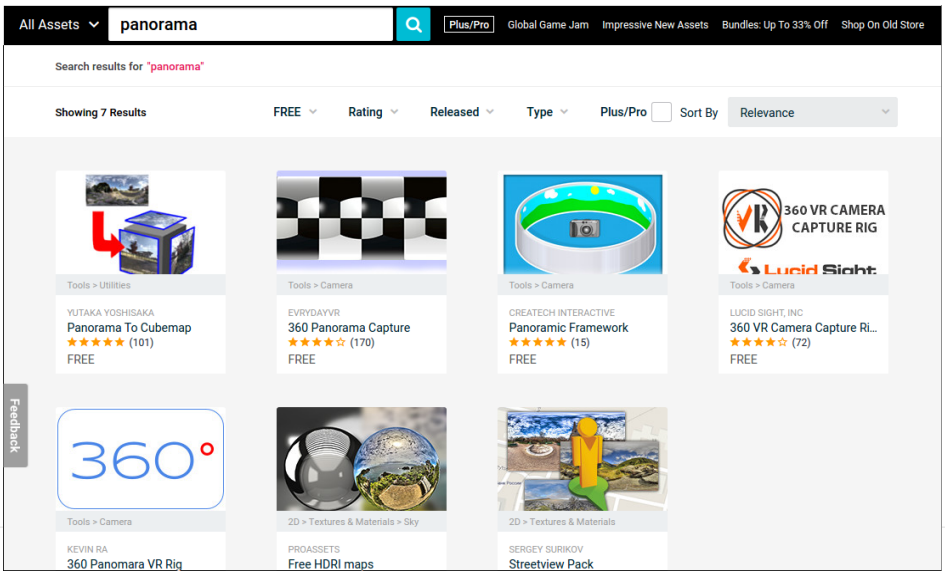


Illustration 9: Asset store de Unity

FONCTIONNEMENT GLOBAL DE UNITY

Ayant quelques bases sur Unity j'étais déjà un peu familier avec, en revanche cela reste un outil puissant avec des possibilités énormes et je n'avais pas d'expérience sur des traitements complexes hors-tutoriel. Le programme fonctionne sur deux axes possibles : 2D ou 3D, dans notre cas nous utiliserons uniquement l'interface 3D car nous développons une application VR.

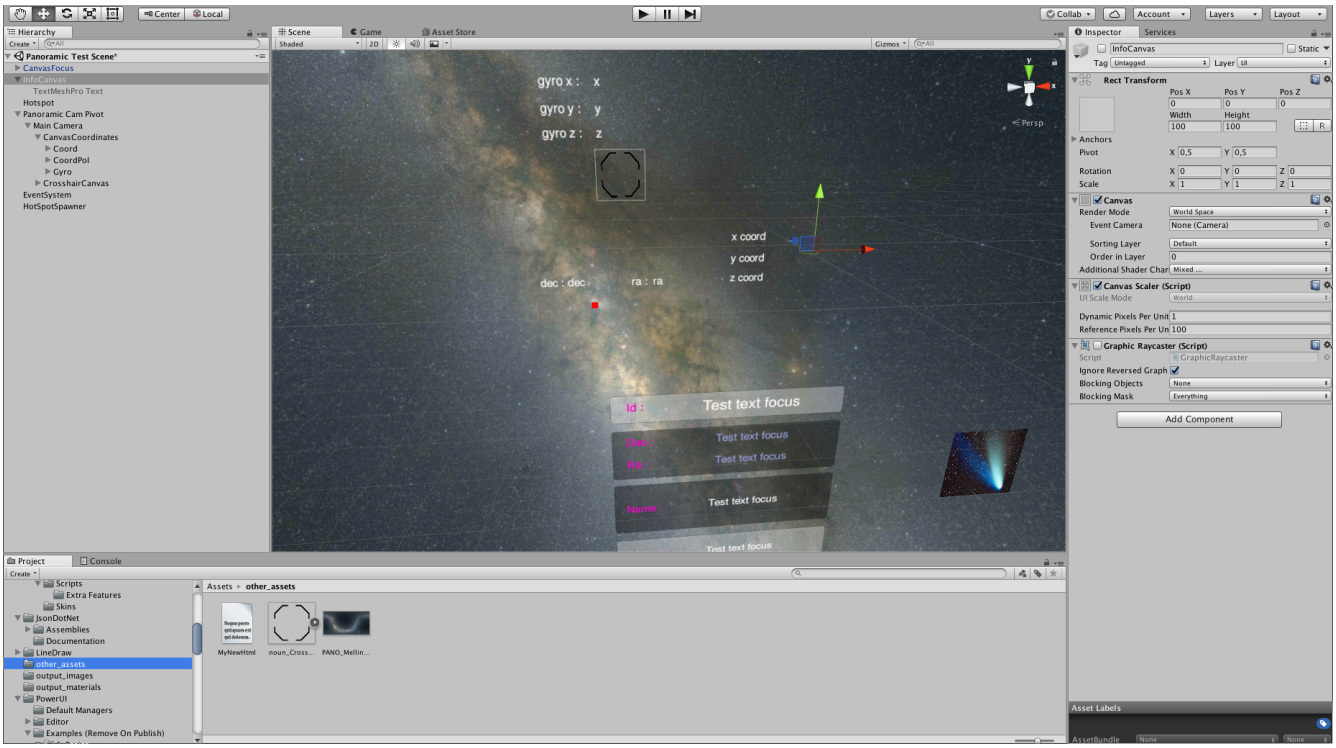


Illustration 10: Fenêtre de travail de Unity

Assets

Les assets sont un des éléments les plus importants de Unity, chaque ressource informatique utilisé dans un projet est un asset soit une image, du texte, un script ou même un fichier audio.

Prefabs

Les prefabs sont extrêmement utiles, c'est en fait un assemblage d'assets avec des paramètres définis. Ceux-ci sont alors liés entre eux de façon à créer un prefab qui pourra par la suite être instancié à volonté et dupliqué. Un exemple classique pourrait être un personnage dont chacun de ses membres est un asset, ainsi on peut instancier différents personnages identiques.

Scripts

Les scripts permettent de programmer toutes les interactions, les événements entre les objets du jeu ainsi que modifier les paramètres des composants. Ceux-ci seront écris en C#, un langage se rapprochant beaucoup du Java. MonoBehaviour est un concept important lorsqu'on crée des scripts, la plupart des scripts héritent de la classe MonoBehaviour, elle permet notamment d'utiliser des fonctions qui se déclenchent à des moments clés comme la création du script, chaque instant (ou « tick ») de la vie du script ou bien lors de sa destruction. Un script doit forcément être attaché à un GameObject dans Unity.

GameObject

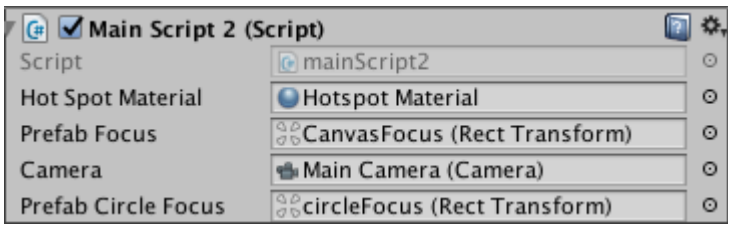
Un GameObject est la classe de base pour toute entité dans Unity. Celui-ci peut-être un objet 2D ou 3D, une lumière, une caméra ou bien un texte d'interface. Lorsqu'on souhaite ajouter un asset à notre scène, on doit l'attacher à un GameObject.

Références

Dans Unity pour gérer les références à d'autres objets au sein du projet, on déclare au sein d'un script la variable de façon classique :

```
public Material HotSpotMaterial;
public Transform prefabFocus;
public Camera m_Camera;
public Transform prefabCircleFocus;
```

Suite à ça un cadre apparaîtra dans l'inspector, pour chaque variable il sera donc nécessaire de déplacer un objet du type correspondant, provenant de la scène ou des assets, jusqu'à ce cadre.



PANORAMIC FRAMEWORK

C’est un framework gratuit créé par Createch Interactive. Il permet de nombreuses choses qui correspondent avec le projet :

- Transformation d’une image en une *cubemap,* en effet il est nécessaire d’avoir une cubemap pour l’afficher en tant que *skybox*. Une cubemap est simplement un cube avec pour chaque face un morceau de l’image, chaque morceau se connectant avec les autres. Enfin une skybox est un procédé graphique permettant de représenter le ciel dans un espace 3D, c’est en réalité une cubemap qui est placé tout autour de la caméra et qui semble inatteignable.

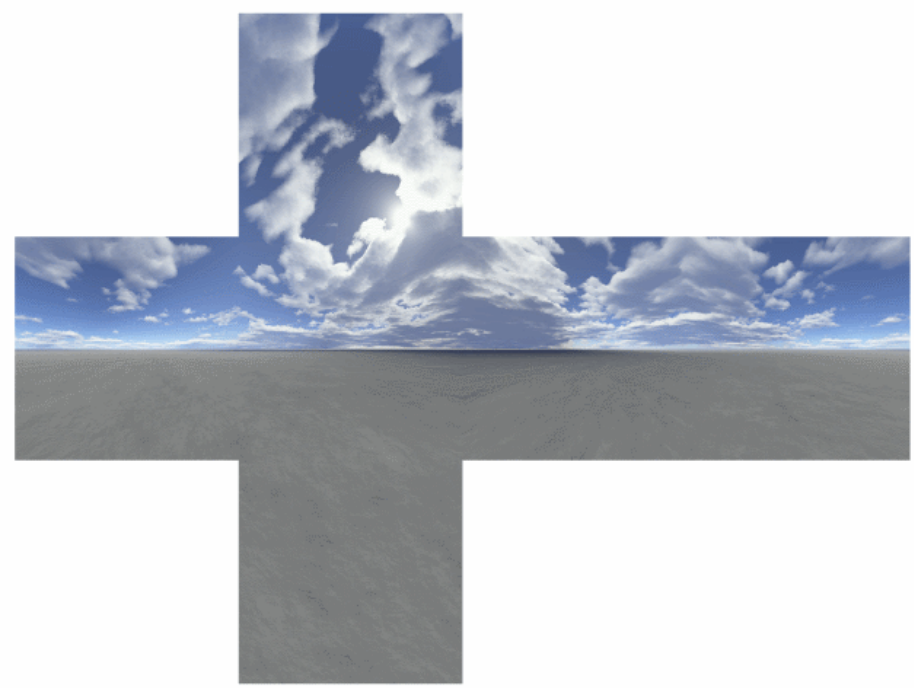


Illustration 11: Exemple de Cubemap

- Création d’un Hotspot : C’est une forme dans l’espace, comme par exemple un carré. Son but sera de déclencher un événement lorsqu’il subit une interaction, ici l’interaction proposée par le framework en démonstration est soit un clic, soit un **raycast** passant par le hotspot. Le raycast représente en réalité un rayon partant de la caméra qui est en général centré au milieu du regard, donc il sera ici centré au milieu de notre écran, si ce rayon vient traverser une partie du cube alors la condition est remplie pour déclencher une action. Ce mécanisme est très intéressant notamment pour une application de VR voulant limiter au maximum les actions avec l’interface.

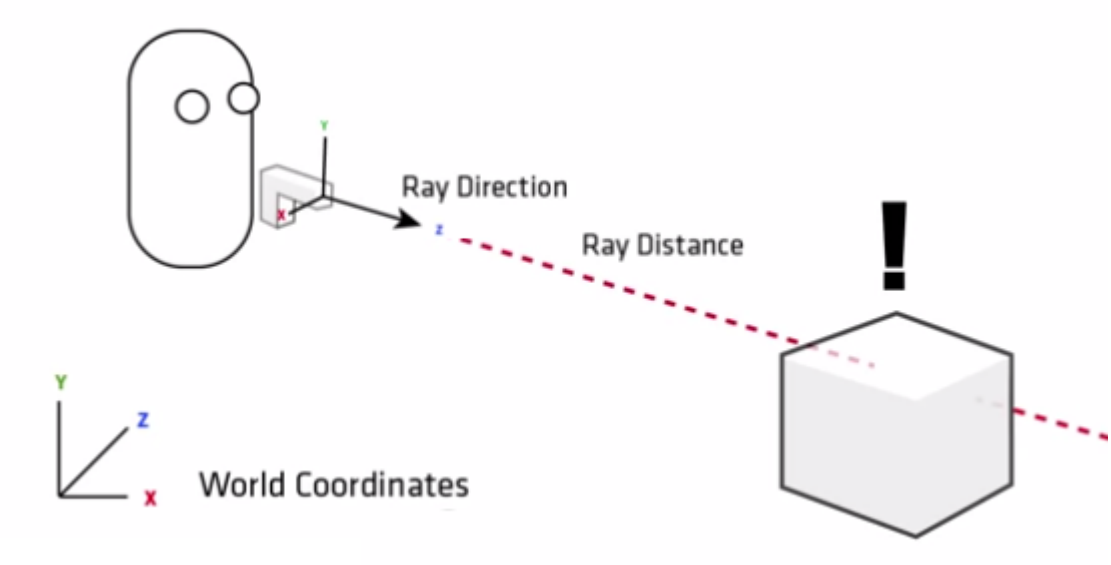


Illustration 12: Schéma d'un raycast

- Création de la caméra pivot qui sera centrée dans l’espace. Cette caméra est la base d’une application VR 360° : dans le repère 3D de Unity elle est placée aux coordonnées (0,0,0) et est inamovible, elle peut uniquement pivoter dans tous les sens. Le framework fournit également un système de déplacement de la caméra via des boutons cliquables, pratique notamment pour faire des tests de calibrage.

CRÉATION DU CUBEMAP

La première étape fut d'importer un .jpeg d'une image HiPS :

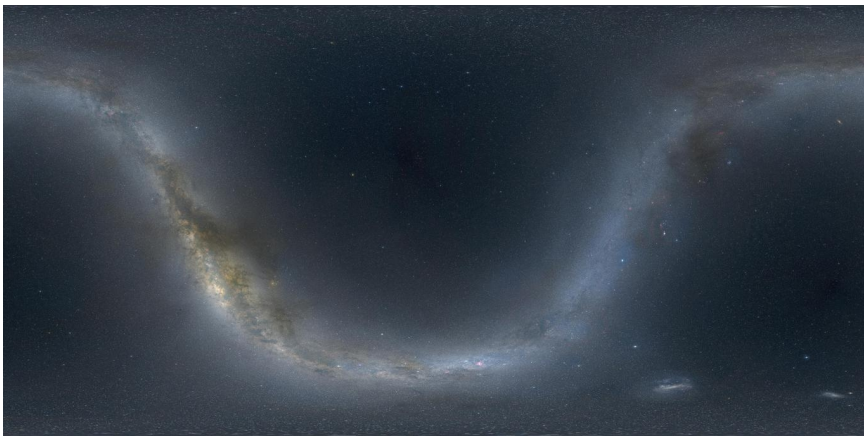


Illustration 13: Image panoramique utilisée dans l'application

Cette image est disponible sur Aladin pour l'utiliser avec une application Cardboard, notamment l'application officielle créée par Google uniquement pour visualiser n'importe quelle image 360°.

On importe donc l'image en tant qu'Asset et on l'intègre dans le traitement du framework pour la transformer en cubemap. Il est notamment possible d'augmenter la résolution de chaque face du cubemap jusqu'à une résolution de 1024*1024. En rendu cette résolution donne un résultat très moyen en terme de visuel dans l'application et rend l'application assez désagréable à utiliser, il est en revanche possible d'augmenter cette résolution jusqu'à 4096*4096 en modifiant directement le code du framework.

Cette modification est effectué assez tard dans le développement et donne un rendu tout à fait acceptable pour une application Cardboard. Un seul problème important est apparu : il n'est pas possible de faire ce traitement pendant que l'application fonctionne.

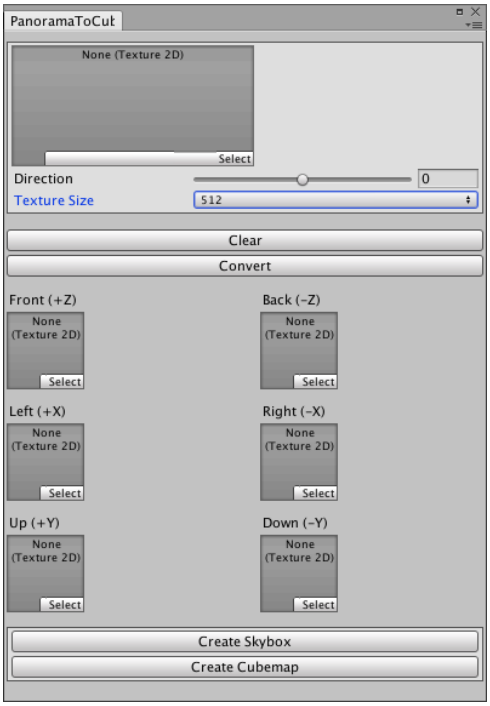


Illustration 14: Interface de création d'un cubemap

C’est une importante contrainte car cela oblige à créer au préalable une série d’images pré-converties dans l’application. L’utilisation idéale aurait été de demander à l’utilisateur de choisir parmi une importante série d’image qui serait traité au cas par cas.

On obtient alors une application utilisable équivalente à l’application VR de base de Google permettant de visualiser des images 360°.

TESTS DE L’APPLICATION

Unity permet de tester une application conçue pour Android directement sur PC sans besoin d’émulation. Cela s’avère extrêmement pratique pour les phases de test, il est en revanche très important de tester l’application directement sur Smartphone pour obtenir un vrai résultat, en effet la zone utilisable de l’écran avec la Cardboard est très réduite par rapport à l’écran de test sur Unity. Il est également important de faire attention à la netteté de l’affichage, sur une Cardboard les textes sont souvent assez flous et doivent avoir une taille conséquente, tester l’affichage est primordial. L’idéal est de tester l’application pour différents Smartphone, celle-ci étant supposé être utilisable par un maximum de smartphone, c’est en revanche complexe car Unity ne dispose pas d’émulateur de Smartphone comme Android Studio il serait donc nécessaire de disposer de différents smartphone de test. En pratique cela nécessiterait un gros budget ce qui n’est pas réalisable pour ce projet, on utilisera donc mon smartphone personnel : un OnePlus 3 ainsi que celui d’André Schaaff : un Samsung S7. Ne disposant pas d’appareil Apple il n’est pas possible de développer l’application pour iOS, la contrainte étant requise pour compiler l’application.

RAYCAST ET ZONE DE FOCUS

Le raycast est la principale mécanique de l’application, le but sera donc de créer des hotspot aux différents endroits où se situent des objet astronomiques intéressants. Le déclenchement du Hotspot créerait alors une interface affichant différentes informations intéressantes.

Voici le code qui permet de comprendre la création d’un raycast :

```
Ray ray = Camera.main.ViewportPointToRay (new Vector3 (0.5f, 0.5f, 0f));
```

On récupère la caméra principale (Camera.main) puis on définit le rayon suivant un vecteur. Ce vecteur est dans un repère relatif à la caméra : Si la caméra se déplace ou pivote, le vecteur sera toujours à la même position du point de vue de la caméra. Le vecteur utilisé correspond donc à un vecteur qui a pour origine le centre de la caméra et est orienté dans la direction de la caméra. On obtient alors notre rayon qui permet de réaliser nos traitements.

Pour détecter avec ce rayon le contact d'un objet, on vérifie que le rayon est en contact avec un *Rigidbody* qui représente un objet physique. Si ce *Rigidbody* appartient à un objet de type Hotspot alors on peut déclencher l'action.

Un problème apparaît alors avec ce système : Il est nécessaire de placer des Hotspot au préalable, cela créerait une quantité d'objets énorme. La question se pose alors, est-il vraiment nécessaire d'utiliser ce système de Hotspot ? De nombreux astres sont visibles sur l'image, comment définir lesquels sont les plus pertinents ?

Finalement le choix a été fait de pouvoir obtenir des informations sur des objets sur n'importe quel point de l'image : Lorsque l'utilisateur reste concentré sur un endroit, l'application affiche alors des informations sur l'objet le plus pertinent de cette zone.

Créer une zone de focus :

Pour créer une zone de focus il est nécessaire de créer un point de focus, chaque fois que l'utilisateur déplace la tête d'un angle défini le point de focus est redéfini de nouveau au centre de l'écran. Il est nécessaire de trouver une distance qui soit équilibrée car la zone de focus doit être permissive pour les légers mouvements de tête. Un temps pour que le focus se fasse est défini, si l'utilisateur reste fixe pendant ce temps alors l'interface s'affiche. Pour réaliser cela on récupère le raycast et on additionne les changements de positions du rayon, notamment en utilisant le vecteur du rayon et en vérifiant son déplacement à chaque « update ».

APPEL SERVEUR

Lorsque le focus est réalisé on réalise un appel au serveur Simbad, celui-ci met à disposition des informations basiques ce qui semble le plus judicieux pour un premier affichage d'informations. Pour faire la requête on utilise le service TAP[7] Simbad avec une requête en ADQL :

```
SELECT TOP 1 main_id AS "Main identifier", otype_longname, RA, DEC,
DISTANCE(POINT('ICRS',RA,DEC),POINT('ICRS', 0.0, 0.0)) AS "Distance"
FROM basic JOIN otypedef ON basic.otype=otypedef.otype
WHERE CONTAINS(POINT('ICRS', RA, DEC), CIRCLE('ICRS', 340.5098,
13.84546, 1)) = 1
ORDER BY "Distance" ;
```

On récupère l'identifiant, le nom complet, les coordonnées ra et dec exactes ainsi que la distance nous séparant de l'objet. On doit notamment effectuer une jointure pour récupérer le nom complet. Ensuite on prend les éléments qui sont contenus dans un cercle de rayon 1 aux coordonnées désirées : 340.5098 et 13.84546 sont les coordonnées ra et dec récupérées et préalablement converties sur Unity. On prend la première valeur en les classant par distance, on choisit donc d'afficher les informations de l'astre le plus proche. Le calcul des coordonnées est une partie critique du programme et est explicitée dans la partie suivante.

Suite à cette requête on récupère une réponse en JSON, il a alors été fait le choix d'utiliser l'asset Json.NET qui est l'outil le plus complet sur Unity pour gérer le JSON : En effet la réponse JSON est sous forme d'un premier tableau de *métadata* qui donne une description de tous les attributs demandés dans la requête. Un second tableau contient les objets demandés dans la requête. Unity possède un système de gestion du JSON par défaut qui ne peut pas traiter les tableaux, d'où la nécessité de trouver un asset plus complet.

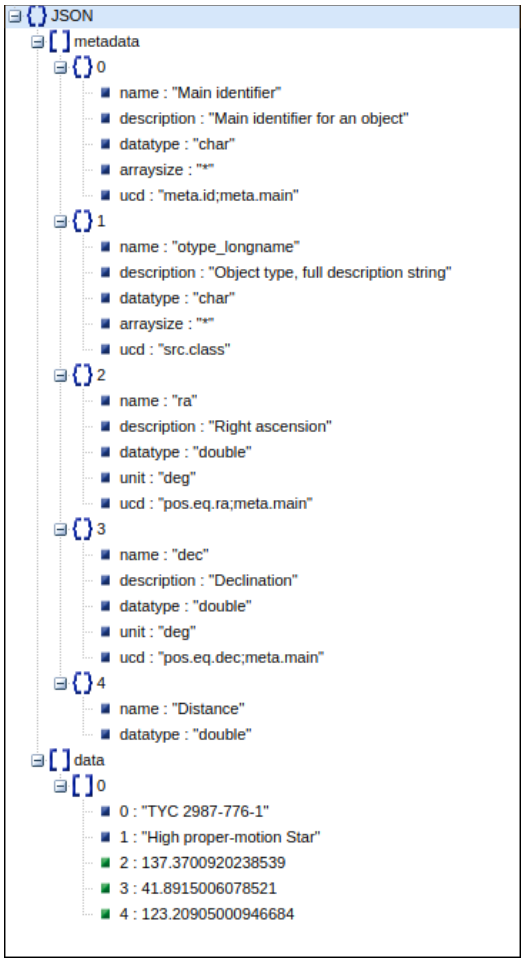


Illustration 15: Visualisation d'une réponse JSON

7 Protocole de l'IVOA qui permet de récupérer des des tableaux de données

Json.NET utilise un système d'objet anonyme (Jobject), cet objet est flexible et permet ici d'être traité comme un tableau pour récupérer par exemple le nom complet :

```
jo ["data"] [0] [1].Value<string> ();
```

Il serait probablement plus judicieux dans des développements futures d'utiliser les métadonnées pour récupérer toutes les données de façon dynamique, on pourrait par exemple avoir un Dictionnaire (équivalent d'un Hashmap en C#) pour répertorier tous les attributs et valeurs. En revanche pour notre utilisation qui est d'afficher pour l'instant que les informations de base cela ne semble pas nécessaire.

Pour faire la requête au sein de Unity on utilise un module nommé WWW qui est assez basique et sert à faire des requêtes HTTP GET ou POST. On utilise également une coroutine qui est un système très important sur Unity : cela permet de paralléliser une tâche. En effet sur Unity chaque fonction appelée dans la fonction Update() doit être exécutée toutes les *frame*. Le déroulement d'une telle fonction ne doit pas dépasser la durée de la frame : si c'est le cas alors la frame dure plus longtemps et donnera une sensation de ralentissement. On utilise alors une coroutine pour ce genre d'appel serveur car ce n'est pas un traitement qui a un effet immédiat sur l'affichage. Le déroulement de la coroutine se poursuit alors sur les frame suivantes.

CALCUL DES COORDONNÉES

Les coordonnées dans Unity sont récupérées grâce au Raycast de la caméra : on prend le vecteur ray.direction. A noter que dans Unity les vecteurs sont un des outils les plus importants : un type de vecteur sera défini en fonction de son nombre de composantes : Vecteur2 pour un vecteur en deux dimension et Vecteur3 pour trois dimensions.

Notre objectif est de convertir ce vecteur de coordonnées en coordonnées équatoriales. Le seul outil de repère que nous avons est Aladin qui permet de charger la même image et de comparer les coordonnées trouvées.

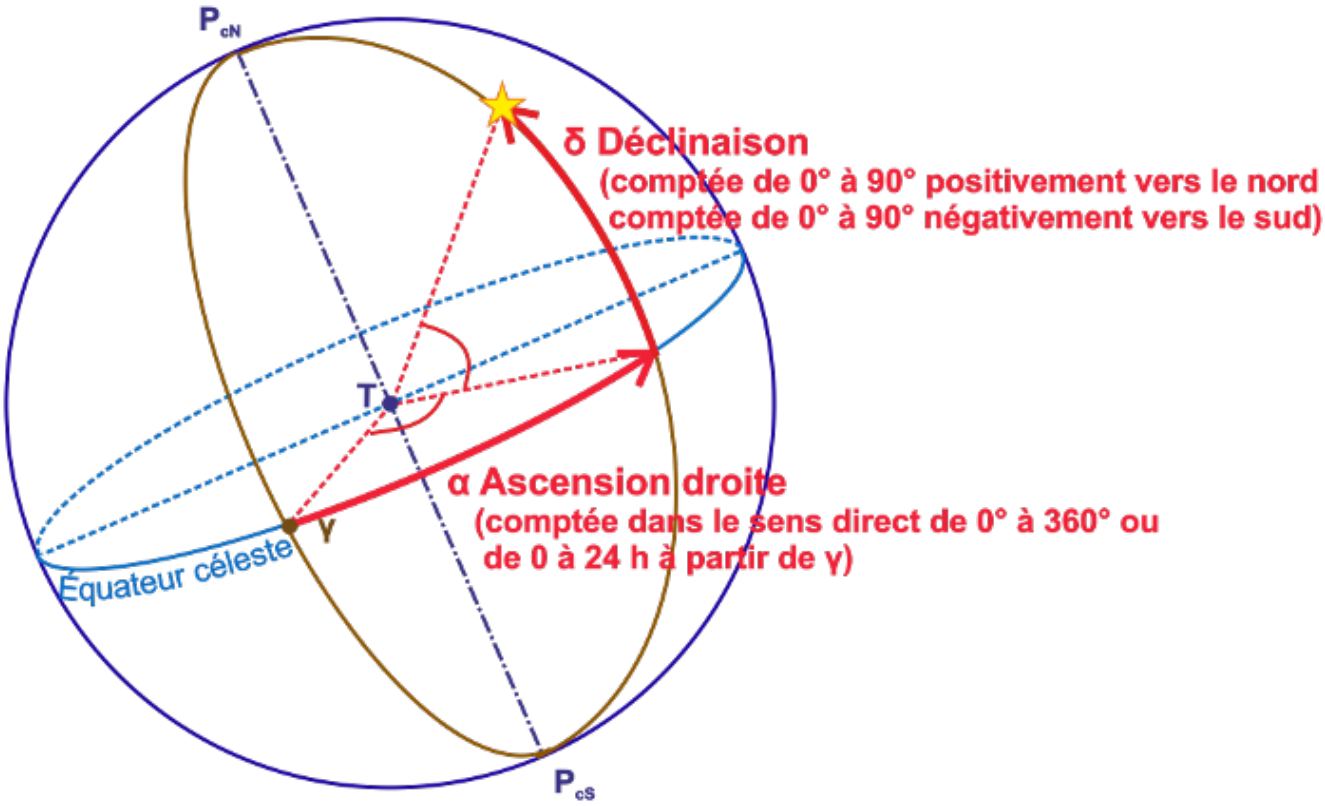


Illustration 16: Schéma des coordonnées équatoriales

On cherche à obtenir la déclinaison et l'ascension droite. Pour ce faire nous avons utilisé cette formule :

```
ra = Mathf.Atan2(point.x,point.z);
```

Cela permet de calculer l'ascension droite (soit la longitude).

```
float xzLen = new Vector2(point.x,point.z).magnitude;
```

Ce calcul est l'équivalent de

```
sqrt(pow(x,2), pow(y,2))
```

```
dec = Mathf.Atan2(-point.y,xzLen);
```

On obtient également la déclinaison (soit la latitude). On a alors les coordonnées ra et dec convertis en revanche elles ne correspondent pas du tout à celles que l'on peut observer sur Aladin, il est donc nécessaire de calibrer grâce à un répère. J'ai choisi de prendre le centre galactique comme repère, ses coordonnées dans Aladin sont : ra = 266.4168 et dec = -29.0078

L’opération va être de rechercher visuellement le même point via Unity et de comparer les chiffres obtenus.

On observe notamment que contrairement à Aladin, l’ascension droite obtenue va de -180° à +180°, or on aimerait obtenir un axe allant de 0° à +360°, il est alors requis d’ajuster :

```
ra += 180f;
```

La déclinaison quant à elle semble être déjà à la bonne valeur seulement l’axe est inversé, on cherche -29° en revanche on obtient +29°, on inverse donc facilement l’axe :

```
dec *= (-1);
```

Il ne reste plus qu’à ajuster l’ascension droite : On calcule notamment la différence entre les deux mesures et on obtient une différence de 82.4668f. On va donc ajouter cette valeur aux coordonnées, en revanche il est important de noter que la valeur ne doit pas dépasser +360°, et si c’est le cas il devient nécessaire de faire un modulo de 360.

```
if ((ra + 82.4668f) > 360)
{ ra = (ra +  82.4668f) % 360; }
else { ra +=  82.4668f; }
```

Le calcul n’est en réalité pas terminé car l’axe de l’ascension droite se révèle être inversé par rapport à Aladin, le traitement est en revanche plus complexe que pour la déclinaison :

```
if (ra > 266.4168f) {
     float diff = ra - 266.4168f;
     ra -= diff*2;
     if (ra < 0)
          ra = 360 + ra;
} else {
     float diff = 266.4168f - ra;
     ra += diff*2;
     if (ra > 360)
          ra = ra % 360;
}
```

Il faut noter qu'on compare l'ascension droite à 266.4168 qui est l'ascension droite du centre galactique, on inverse effectivement l'axe par rapport à ce point sur lequel on a décidé de centrer le calibrage.

Le calcul est enfin terminé et nous obtenons exactement les mêmes valeurs que ce qu'on peut voir sur Aladin. Ce travail fut assez complexe car il n'est pas facile de réaliser ces calculs sans avoir une bonne représentation mentale du problème. A noter qu'au départ je n'avais pas remarqué qu'Aladin disposait des coordonnées équatoriales, j'ai donc converti ces coordonnées en galactique, qui implique encore d'autres calculs et une plus grande possibilité d'erreur. En revanche ces calculs ont été réalisés et restent très intéressants et utiles, les coordonnées galactiques pouvant éventuellement être affichées sur l'interface.

Il a été ensuite nécessaire de reconvertir les coordonnées équatoriales en coordonnées cartésiennes utilisables par Unity, ce calcul reprend le même principe que les calculs précédents. Cette étape est nécessaire notamment car dans notre requête nous envoyons un cercle avec pour coordonnées le centre de notre écran, l'objet retourné ne se situe pas forcément au centre et on doit donc connaître sa position exacte.

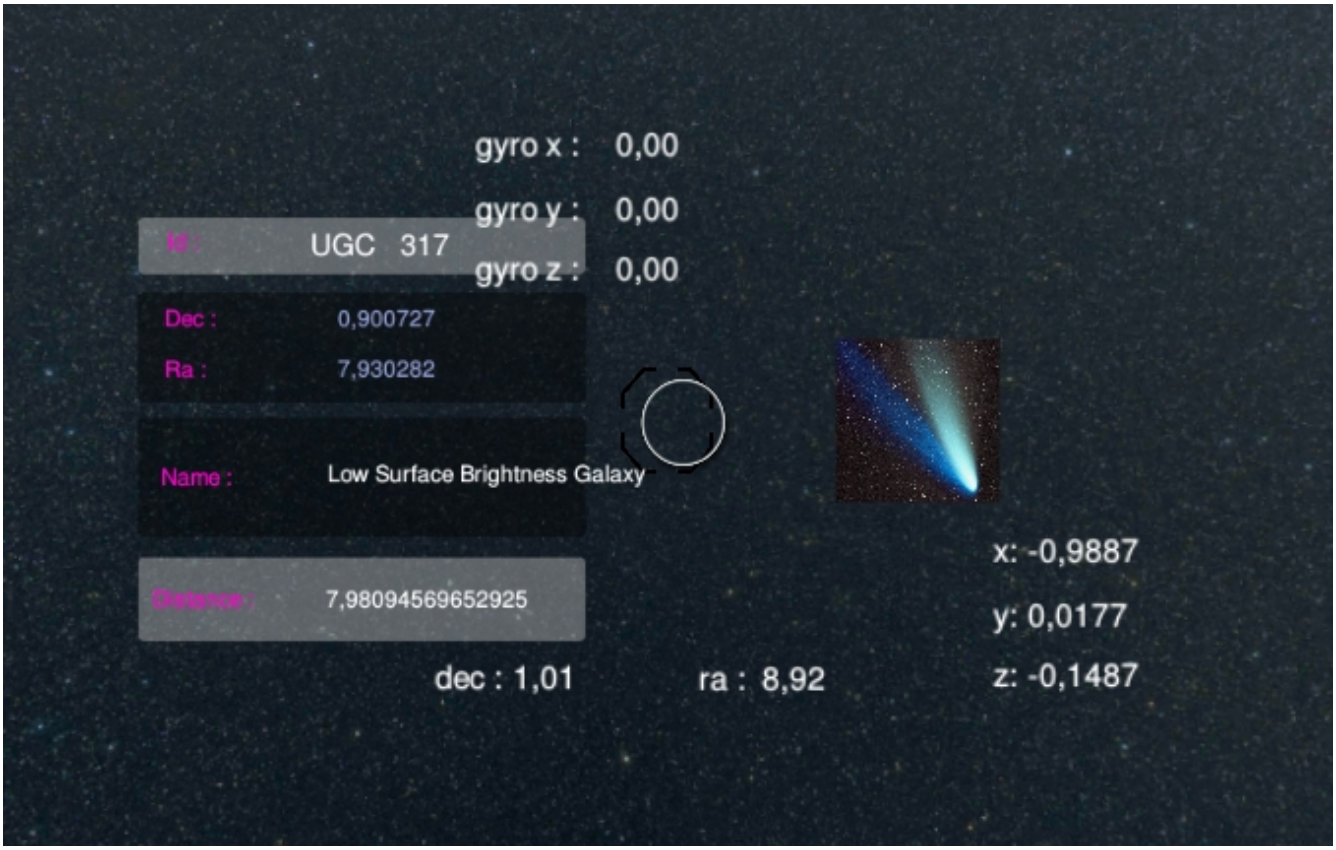


Illustration 17: Affichage de l'interface après focus

Sur l'illustration ci-dessus on peut voir l'affichage tel qu'on l'obtient dans l'application, le réticule noir sert de « viseur » car dans un affichage de réalité virtuelle il est en réalité assez complexe de discerner le centre exact de ce que l'on observe. Le cercle blanc est donc centré sur l'objet astronomique qui a été trouvé.

INTERFACE

L'interface est en pratique représenté comme une feuille dans l'espace qui sera toujours dirigée vers la caméra centrale. Il y a dans l'application deux types d'affichage différents : l'affichage de la caméra, et l'affichage de focus. Dans le premier cas on aura les éléments qui restent toujours fixes par rapport à la caméra : Le réticule noir, et les informations de calibrages en blanc. Ces éléments sont tous positionnés sur un **canvas**, ce dernier est un outil très utile car il sert à afficher tous les éléments qu'on qualifiera d'interface utilisateur. Un canvas peut être lié à l'écran (comme une feuille transparente collé à la caméra avec des éléments posés dessus) comme il peut être lié à l'espace.

Dans le cadre d'une application de réalité virtuelle l'usage veut qu'on n'utilise que des canvas lié à l'espace. Un canvas dans cette configuration est complexe à visualiser, on peut imaginer un groupe d'éléments d'interface qui seront placés dans l'espace mais tous seront liés par le même canvas : ils obéiront tous aux propriétés de ce dernier.

DÉVELOPPEMENTS NON ABOUTIS

Widget Aladin

Lors de l'affichage de l'interface sur les objets recevant le focus, je souhaitais notamment pouvoir afficher un visuel plus agréable. En effet malgré que la skybox soit dans une bonne résolution il n'est pas évident de clairement voir l'objet observé, j'avais donc souhaité pouvoir afficher une image de l'objet en question si elle existait. L'idée était alors d'utiliser Aladin Lite : On récupère le widget, on centre Aladin sur l'astre voulu grâce aux informations données par le serveur et on récupère une image de l'objet grâce à une fonction d'Aladin Lite.

Illustration 18: Interface d'Aladin Lite

Pour ce faire il était nécessaire d'intégrer du HTML/CSS/JS au sein de Unity ce qui n'est pas possible par défaut. J'ai donc pu trouver un asset nommé « PowerUI » qui permet notamment d'afficher une interface réalisée en langage web. Cette solution semblait répondre à mes besoins jusqu'à ce que je réalise qu'il ne supportait que les fonctions JavaScript les plus basiques. Cet asset était en réalité utilisé principalement pour pouvoir personnaliser de l'interface axé texte avec les outils du développement web.

Cette réalisation ne semble pas impossible car il existe des assets plus puissants mais payants sur l'asset store, certains seraient probablement capable d'afficher l'interface d'Aladin dans Unity.

DialogFlow

Une tentative a été faite d'intégrer DialogFlow dans Unity pour mettre en place le système de commande vocale. Un asset existe pour l'ancienne version de DialogFlow nommé API.AI mais n'est plus supporté depuis quelques temps. En effet beaucoup d'erreurs apparaissent lors de l'importation de l'asset et n'est pas fonctionnelle, de plus il utilisait la V1 de DialogFlow annoncée comme bientôt obsolète.

Une autre possibilité était notamment de récupérer le package DialogFlow pour C# prévu pour une architecture .NET. Le premier obstacle fut notamment que le package n'était plus disponible et les solutions proposées par le site officiel de DialogFlow n'étaient pas à jour, il était alors impossible de récupérer le package. Enfin je n'ai aucune expérience en .NET et il était notamment nécessaire de créer un .DLL pour ensuite l'importer dans Unity.

Le traitement semblait complexe et relativement lié au bon vouloir de Google. D'après moi les deux seules solutions seraient soit d'attendre un outil réellement utilisable par Google dans Unity, soit de trouver un système semblable à DialogFlow mais qui nécessiterait d'apprendre à maîtriser un outil supplémentaire.

VI. CONCLUSION

Lors de ce stage un travail a été réalisé sur les différentes méthodes de compréhension du langage naturel avec DialogFlow ainsi que l'apprentissage de nodeJS qui n'a pas pu aboutir à un résultat fini. Une application de Réalité virtuelle sur Android a pu être réalisée via Unity permettant de visualiser des relevés d'images. La suite logique de ce développement serait de réussir à intégrer DialogFlow au sein de l'application Unity pour pouvoir interagir avec l'interface dans un environnement de réalité virtuelle. Le Chatbot est également un développement pouvant être amélioré indépendamment.

Ce stage m'a permis de découvrir le monde de la recherche et de découvrir des technologies actuelles. C'est un environnement très serein et reposant qui laisse place à l'imagination et à l'expérimentation dans un superbe cadre. Je pense en revanche que le fait d'avoir moins de contrainte ne correspond pas réellement à ma manière de travailler, je suis plus stimulé par des grosses contraintes de temps et de résultat. Tenter de réaliser de nouvelles choses me semblait à l'origine comme le travail idéal, cela m'a donc permis de réaliser que je m'orienterai probablement vers le monde de l'entreprise.

Ce stage reprend des bases du développement logiciel, celui-ci n'a pas réellement influencé mon choix de filière. Ayant déjà une bonne expérience en développement j'ai déjà une bonne idée de mon projet professionnel et ce stage m'a permis d'élargir cette expérience.

VII. BIBLIOGRAPHIE

Formules conversion de coordonnées :

https://en.wikipedia.org/wiki/Celestial_coordinate_system

1/02/2019

Fonctionnement du Raycast :

https://gamedev.stackexchange.com/questions/93592/graphics-raycaster-of-unity-how-does-it-work

ADQL :

http://tapvizier.u-strasbg.fr/adql/help.html

Coordonnées :

http://www.cosmovisions.com/CTreperages.htm

Sphère céleste :

https://fr.wikipedia.org/wiki/Sph%C3%A8re_c%C3%A9leste

1/02/2019

CDS :

http://cdsweb.u-strasbg.fr/about

VO/IVOA :

http://www.euro-vo.org/

VIII. GLOSSAIRE

VO : Virtual Observatory – Initiative internationale permettant l'accès aux base de données de l'espace et des différents observatoires.

IVOA : International Virtual Observatory Alliance – Organisation qui débat et s'accorde sur les standards techniques qu'utilise l'Observatoire Virtuel.

Natural Language Processing / Natural Language Understanding : L'application de processus automatiques permettant l'analyse et la compréhension du langage naturel.

Machine Learning : Procédé permettant de résoudre des problèmes complexes automatiquement en repérant des schémas dans de grosses quantités de données.

API : Application Programming Interface – Ensemble de fonctions qui facilitent l'accès au service d'une application.

ADQL : Astronomical Data Query Language – Langage semblable au SQL utilisé par l'observatoire virtuel pour envoyer des requêtes aux services de l'IVOA.

SDK : Software Development Kit – Ensemble d'outils d'aide à la programmation (Kit de développement).

Cross-identification : Identification croisée. Consiste à identifier dans deux catalogues de sources astronomiques celles qui sont détectées dans les deux, et à les relier.

Image HiPS : Hierarchical Progressive Surveys. Mécanisme de pavage hiérarchique permettant d'accéder, de visualiser et de parcourir de manière transparente les données d'images et de catalogues

Asset : Représentation d'un objet qui peut être utilisé au sein d'un projet, cela peut être n'importe quel type de fichier supporté par Unity

Framework : Architecture logicielle permettant de faciliter le développement

IX. ANNEXES

1. ANNEXE A

CALCUL DES COORDONNÉES GALACTIQUES

Le calcul des coordonnées se base sur une équation récupérée sur Wikipédia, elle correspondait notamment à d'autres sources j'ai donc implémenté cette équation :

$$\begin{aligned}\sin(\alpha - \alpha_{\mathrm{G}})\cos(\delta) &= \cos(b)\cos(l - l_{\mathrm{NCP}})\\ \cos(\alpha - \alpha_{\mathrm{G}})\cos(\delta) &= \sin(b)\cos(\delta_{\mathrm{G}}) - \cos(b)\sin(\delta_{\mathrm{G}})\sin(l - l_{\mathrm{NCP}})\\ \sin(\delta) &= \sin(b)\sin(\delta_{\mathrm{G}}) + \cos(b)\cos(\delta_{\mathrm{G}})\sin(l - l_{\mathrm{NCP}})\end{aligned}$$

A noter qu'ils y a 3 constantes dans l'équation :

$$\alpha_G = 192.85948^\circ \qquad \delta_G = 27.12825^\circ \qquad l_{\mathrm{NCP}} = 122.93192^\circ$$

Elles correspondent respectivement à l'ascension droite puis à la déclinaison du pôle nord galactique et finalement la longitude galactique du pôle nord céleste.

Les pôles célestes correspondent aux pôles de la sphère céleste, celle-ci étant centrée autour de la terre. Ce sont également les pôles du système équatorial.

Le plan galactique utilise le système solaire comme centre et le centre galactique correspond à son point zéro.

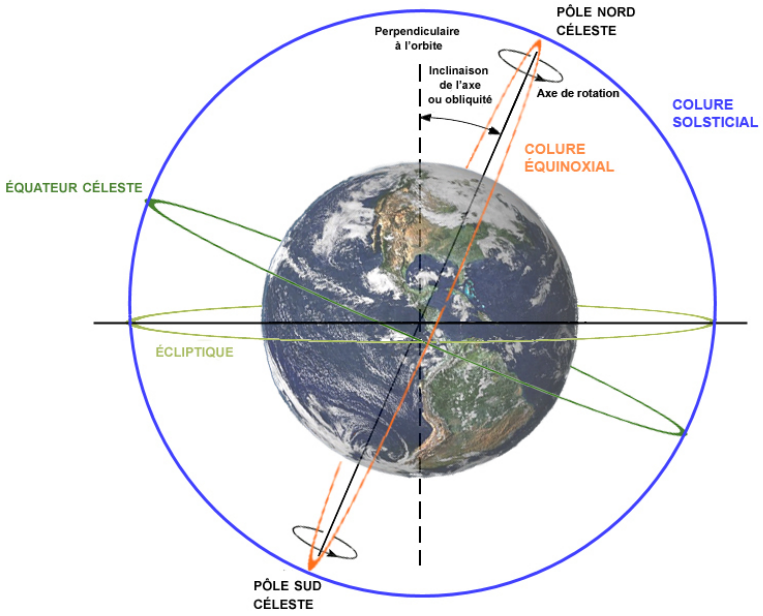


Illustration 19: Schéma de la sphère célèste

```
Vector2 EquatorialToGalactic(float dec, float ra)
{
    float deltaG = 27.12825f * Mathf.Deg2Rad;
    float alphaG = 192.85948f * Mathf.Deg2Rad;
    float lNCP = 122.93192f * Mathf.Deg2Rad;

    float delta = dec * Mathf.Deg2Rad;
    float alpha = ra * Mathf.Deg2Rad;

    float b = Mathf.Asin(Mathf.Sin (delta) * Mathf.Sin (deltaG) + Mathf.Cos (delta) * Mathf.Cos (deltaG) * Mathf.Cos (alpha - alphaG));

    float l = Mathf.Asin((Mathf.Cos (delta) * Mathf.Sin (alpha - alphaG))/Mathf.Cos(b)) + lNCP;

    return new Vector2 (l*Mathf.Rad2Deg, b*Mathf.Rad2Deg);
}
```

Il est important de noter que les valeurs doivent systématiquement être converties en radian pour pouvoir appliquer des formules de trigonométrie.

```
Vector2 GalacticToEquatorial(float l, float b)
{
    float deltaG = 27.12825f*Mathf.Deg2Rad;
    float alphaG = 192.85948f*Mathf.Deg2Rad;
    float lNCP = 122.93192f*Mathf.Deg2Rad;

    float longl = l * Mathf.Deg2Rad;
    float latb = b * Mathf.Deg2Rad;

    //delta = dec, alpha = ra
    float alpha = Mathf.Asin ((Mathf.Cos (latb) * Mathf.Cos (longl - lNCP)) / Mathf.Cos (latb)) + alphaG;


    float delta = Mathf.Asin (Mathf.Sin (latb) * Mathf.Sin (deltaG) + Mathf.Cos (latb) * Mathf.Cos (deltaG) * Mathf.Sin (longl - lNCP));

    //polar.x = dec / polar.y = ra
    return new Vector2 (delta*Mathf.Rad2Deg, alpha*Mathf.Rad2Deg);
}
```

La conversion des coordonnées galactiques vers équatorial ne semble pas avoir de véritable utilité mais reste possible.

X. TABLE DES FIGURES

XI. Table des matières