ENSIIE STRASBOURG

1 RUE JEAN-DOMINIQUE CASSINI

67400 ILLKIRCH-GRAFFENSTADEN

Rapport de Stage

Affichage d'images astronomiques à 360 degrés avec l'atlas astronomique Aladin

Création d'une interface de contrôle d'Aladin en mode Planétarium

Observatoire astronomique de Strasbourg

AUTEUR : ARNAUD STEINMETZ

Elève ingénieur en informatique à l'ENSIIE

MAITRE DE STAGE : BONNAREL FRANÇOIS

Ingénieur de recherche au CNRS

STRASBOURG, LE 02 SEPTEMBRE 2016

UNIVERSITÉ DE STRASBOURG

REMERCIEMENTS

Tout d'abord, je tiens à remercier l'ensemble de l'équipe de l'Observatoire de Strasbourg, qui m'a fait vivre une expérience professionnelle enrichissante et de passer mon stage dans de très bonnes conditions. Je n'ai pas vu passer ces 10 semaines de stage, tellement j'ai été captivé par un travail intéressant qui m'a permis de développer mes compétences.

Je tenais à remercier tout particulièrement mon maître de stage, François BONNAREL, ainsi que mes encadrants Pierre FERNIQUE et André SCHAAFF pour la confiance qu'ils m'ont accordé tout au long du stage. J'ai pu réaliser ce projet de manière autonome tout en ayant des personnes référentes à qui m'adresser en cas de besoin.

J'adresse également mes remerciements à Laurent MICHEL, pour sa disponibilité pour les explications du fonctionnement de l'outil ArchesWalker. Il a aussi été de bon conseil quant à la conception de l'interface de contrôle d'Aladin.

Mes remerciements s'adressent également à l'équipe du Planétarium dont :

- Milène Wendling, directrice du Planétarium
- Benjamin Rota, responsable technique du Jardin des Sciences
- Jean-Yves Marchal, médiateur scientifique au Planétarium

TABLE DES MATIERES

INTRODUCTION

Dans l'optique de valider ma première année en école d'ingénieur à l'ENSIIE, j'ai effectué un stage d'une durée de 10 semaines à l'Observatoire astronomique de Strasbourg. Le but de ce stage est la mise en contact avec le monde du travail tout en appliquant les connaissances acquises durant la formation.

L'intitulé du stage était : « Affichage d'images astronomiques à 360 degrés avec l'atlas d'images Aladin ». Il a finalement été convenu que le stage se porterait plus sur la conception d'une interface permettant de contrôler l'affichage du logiciel Aladin en mode Planétarium. C'est-à-dire une interface de contrôle utilisable à partir d'un ordinateur principal, tout en projetant Aladin sur le dôme.

Ce rapport a pour objectif de résumer et d'expliquer au mieux la manière dont s'est déroulé ce stage selon la structure suivante :

Dans un premier temps, je présenterai l'Observatoire Astronomique de Strasbourg, les différentes équipes et services qui le composent ainsi que le Planétarium et ses activités. Cela aura pour but de présenter le contexte dans lequel j'ai travaillé et permettra également de comprendre la démarche qui a donné naissance au sujet du stage.

Je détaillerai ensuite le déroulement du stage, en commençant par souligner les différentes applications déjà existantes et actuellement utilisées. Suivra un état de l'art, qui permettra de mieux cerner le domaine et les outils qui pourront être utilisés. J'aborderai ensuite la partie développement qui a nécessité le plus de temps, et dans laquelle seront précisées les différentes démarches mises en place pour répondre au besoin du projet.

Et je terminerai par un bilan qui fera la liaison entre l'outil réalisé et les possibilités d'améliorations futures.

I) PRESENTATION DU CADRE

1) L'OBSERVATOIRE ASTRONOMIQUE DE STRASBOURG

1.1) PRESENTATION GENERALE

L'Observatoire astronomique de Strasbourg est un Observatoire des Sciences de l'Univers (OSU) ainsi qu'une Unité Mixte de Recherche de l'Université de Strasbourg et du CNRS. Il est actuellement dirigé par Hervé Wozniak.

Historiquement, Strasbourg a accueilli trois Observatoires astronomiques. Le premier a vu le jour en 1673 grâce à l'astronome Julius Reichlet et se situait sur les tours d'enceinte de la ville. Le second à quant à lui été construit en 1828 sur le site de l'Académie de Strasbourg. Ce n'est qu'en 1881 que l'Observatoire Astronomique de Strasbourg rejoint son emplacement actuel. Celui-ci fait partie, avec le jardin botanique voisin, des actions mises en œuvre par l'empereur Guillaume Ier pour faire de Strasbourg une vitrine suite à la guerre de 1870.

Actuellement, l'Observatoire astronomique de Strasbourg est composé de trois bâtiments dont le principal est celui de la Grande Coupole, visible ci-contre. Ce dernier contient la 3e plus grande lunette astronomique de France. Outre ses propres bâtiments, l'Observatoire héberge également le Planétarium de Strasbourg dont il a eu la responsabilité de 1986 à 2008. Celui-ci fait désormais partie du Jardin des Sciences de l'Université de Strasbourg.

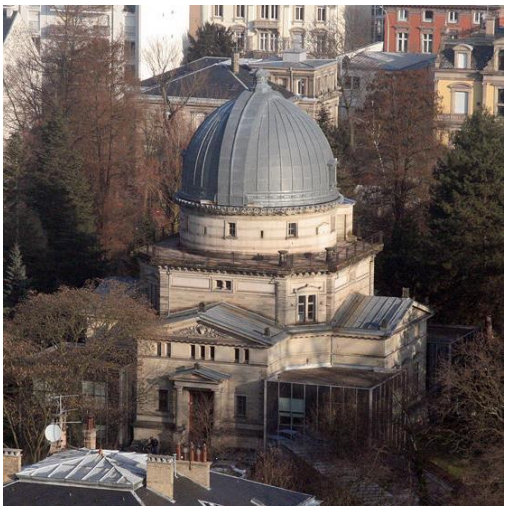


Figure 1 - Bâtiment de la Grande Coupole

L'Observatoire astronomique de Strasbourg est constitué de trois équipes de recherches — Galaxies, Hautes énergies et le CDS — et de deux services d'observation de l'Institut National des Sciences de l'Univers — SSC-XMM et le CDS.

Si à l'origine, l'Observatoire avait pour quasi unique vocation l'observation de comètes, de météorites et d'étoiles variables, ses missions sont bien plus diversifiées à l'heure actuelle. En effet, l'Observatoire astronomique de Strasbourg a pour mission de contribuer au progrès de la connaissance par l'acquisition de données d'observation, le développement de moyens appropriés ou encore l'élaboration des outils théoriques nécessaires. De plus, l'Observatoire est également chargé d'assurer la diffusion des connaissances, d'assurer la formation des étudiants et des personnels de recherche, de prendre part à des activités de coopération internationale, et bien plus encore.

1.2) LES EQUIPES DE RECHERCHES

EQUIPE DE RECHERCHE GALAXIES

Le champ d'action de l'équipe Galaxies regroupe tout ce qui concerne la formation des galaxies ainsi que l'étude et le recensement des populations stellaires qui composent ces galaxies. Plus particulièrement, cette équipe va s'intéresser à notre propre galaxie, la Voie Lactée, ainsi qu'à ses voisines du Groupe Local. Outre les recherches sur les populations stellaires, l'équipe Galaxies s'intéresse également à la dynamique gravitationnelle qui régit les étoiles et les matériaux à l'intérieur des galaxies. Le but de cela est de réunir suffisamment d'informations afin de pouvoir établir l'histoire précise de l'évolution d'un système stellaire et de reconstituer les événements clés dans la vie d'une galaxie.

En plus de cela, l'équipe Galaxies mène des recherches sur les amas stellaires — composants fondamentaux d'une galaxie — afin de mieux comprendre la formation de celles-ci.

Enfin, l'équipe s'implique également dans des missions satellitaires telles que Gaia, mission lancée en décembre 2013 par l'Agence Spatiale Européenne (ESA) ayant pour but de fournir des relevés sur la Voie Lactée.

EQUIPE DE RECHERCHE HAUTES ENERGIES

L'équipe Hautes énergies s'intéresse aux sources émettrices de rayon X, aux objets compacts — étoiles à neutrons par exemple — et aux noyaux actifs des galaxies. Elle est impliquée dans le Survey Science Center d'XMM (SSC-XMM), un consortium international de laboratoires sélectionnés par l'ESA qui est en charge de fournir les 8 catalogues complets d'objets observés par le satellite XMM-Newton. L'équipe Hautes énergies participe également à des projets communautaires tels que le projet européen Arches, en collaboration avec le CDS.

CENTRE DE DONNEES ASTRONOMIQUES DE STRASBOURG (CDS)

Le Centre de Données astronomiques de Strasbourg est à la fois un service d'observation et une équipe de recherche. Le CDS a entre autre développé des services d'accès aux données astronomiques — Simbad, VizieR — et de visualisation de ces mêmes données — Aladin — qui sont utilisés par l'ensemble de la communauté internationale. C'est aussi l'un des acteurs majeurs du développement de l'International Virtual Observatory Alliance (IVOA). En 2008, le CDS a été labellisé « Très Grande Infrastructure de Recherche », ce qui le place au même niveau que des infrastructures européennes comme l'European Southern Observatory (ESO).

2) LE CDS

2.1) PRESENTATION GENERALE

Crée en 1972, le CDS est d'abord connu sous le nom de Centre de Données Stellaires avant de devenir le Centre de Données astronomiques de Strasbourg en 1983. Hébergeant entre autres la base de données de référence pour l'identification d'objets astronomiques, ces services — Simbad, VizieR et Aladin — sont largement utilisés par la communauté astronomique internationale.

Figure 2 – Logo du CDS

Ses missions sont nombreuses et comprennent entre autres :

- le rassemblement des informations utiles concernant les objets astronomiques,
- la mise à jour de ces données,
- la distribution de ces données à la communauté internationale,
- la mise en place de recherche à propos de ces données.

Le CDS emploie actuellement 33 personnes réparties de la façon suivante :

- 8 chercheurs/astronomes-adjoints
- 1 chercheur postdoctoral
- 11 informaticiens
- 10 documentalistes
- 3 administratifs.

2.2) LES DIFFERENTS SERVICES DU CDS

SIMBAD

Simbad est la base de données de référence mondiale en ce qui concerne la nomenclature et la bibliographie d'objets astronomiques. Possédant l'équivalent de plus de 40 ans de données, Simbad permet aux astronomes de trouver facilement des informations sur plus de 8 millions d'objets grâce à plus de 300 000 références bibliographiques.

Simbad dispose également d'un résolveur de noms permettant de retrouver n'importe quel objet désigné par l'un des 22 millions d'identifiants que la base contient.

Simbad a reçu en moyenne 360 000 requêtes par jour pour l'année 2014, soit plus de 4 requêtes par secondes.

Figure 3 – Logo de Simbad

VIZIER

VizieR est une base de données regroupant des catalogues d'objets astronomiques. Ces catalogues sont constitués de données relevées durant des missions d'observation et sont ajoutés à VizieR par les documentalistes du CDS. A l'heure actuelle, la base est constituée de plus de 13 000 catalogues.

Interrogeable sur de nombreux critères (longueur d'ondes, nom de la mission, etc.), VizieR permet de rassembler et d'homogénéiser les données astronomiques afin de pouvoir les comparer et les exporter.

En 2014, le nombre de requêtes faites sur la base de données ont été en moyenne de 300 000 par jour, avec des pics à plus de 2 millions.

Figure 4 – Logo de VizieR

Aladin est un atlas interactif du ciel permettant de visualiser et de comparer des images du ciel. Il a été entièrement développé en Java par Pierre Fernique, Thomas Boch et François Bonnarel.

Aladin utilise des images provenant d'observatoires au sol ou spatiaux et génère des cartes du ciel en trois dimensions (technologie HEALPix). Ces images peuvent être issues de la base de données inclus à Aladin, d'autres bases telles que la NASA Extragalactical Database, ou encore provenir de l'utilisateur lui-même. A l'heure actuelle, Aladin est disponible sous quatre formes : une application téléchargeable, une applet Java, une version JavaScript et un simple previewer en ligne.

Figure 5 – Logo d'Aladin

2.3) INTERNATONAL VIRTUAL OBSERVATORY ALLIANCE

L'International Virtual Observatory Alliance (IVOA) est une organisation internationale regroupant 19 projets d'Observatoire Virtuel (VO) provenant de 17 pays différents : Arménie, Australie, Brésil, Canada, Chine, France, Allemagne, Hongrie, Inde, Italie, Japon, Russie, Ukraine, Espagne, Royaume-Uni, Argentine et États Unis.

Ce projet international a un but similaire au World Wide Web Consortium : mettre en place des standards et décider quels seront les travaux les plus importants à faire ainsi que les stratégies à adopter. Ceci permet de faciliter les échanges internationaux et la collaboration nécessaire à l'élaboration de standards d'interopérabilité et au déploiement d'outils, de systèmes et de structures organisés nécessaires pour permettre une utilisation internationale des archives astronomiques (images, catalogues, etc..). Telle est la mission confiée à l'IVOA lors de sa création en juin 2002.

Le CDS participe activement à ce projet notamment par l'implémentation des standards du VO dans les services cités précédemment : Aladin, Simbad et VizieR.

Figure 6 – Logo de l'IVOA

3) LE PLANETARIUM DE STRASBOURG

3.1) DEFINITION GENERALE

Le terme Planétarium désigne le dôme présentant une reproduction du ciel avec ses constellations et ses étoiles, et par métonymie le projecteur spécifique permettant de simuler le ciel sur un écran hémisphérique, de même que la salle où fonctionne ce projecteur. Comme l'on peut le voir ci-contre avec celui de Bretagne.

En France le Planétarium désigne un lieu de culture scientifique consacré à l'astronomie et pouvant comporter différents systèmes de projection du ciel étoilé.

Figure 7 - Salle de spectacle du Planétarium de Bretagne

Le Planétarium de Strasbourg a été créé en 1981 et inauguré en janvier 1982. Ce planétarium fait partie du site astronomique de Strasbourg qui possède notamment une grande lunette astronomique de 487 mm, abrité sous une coupole de 34 tonnes. Il est situé au cœur de l'Observatoire astronomique de Strasbourg qui en a eu la responsabilité de 1986 à 2008. Depuis 2008, il dépend du Jardin des Sciences de l'université de Strasbourg.

3.2) RELATIONS

Le Planétarium de Strasbourg fait notamment parti de l'APLF (Association des Planétarium de Langue Française), qui regroupe 22 planétariums en France et dont le siège social est l'Observatoire de Strasbourg. Cette association a pour but de renforcer les liens entre le monde des planétariums et les astronomes professionnels, et de développer les planétariums en favorisant les rencontres et les échanges.

Au niveau international, le planétarium prend également part à l'IPS (International Planetarium Society) rassemblant environ 700 membres à travers 35 pays et dont le but premier est de partager des idées à travers diverses rencontres, conférences et publications

3.2) ACTIVITES

Le planétarium propose des spectacles qui permettent de rêver et de s'informer sur l'actualité astronomique et spatiale, sur les connaissances actuelles sur l'Univers. On y découvre le ciel étoilé et les images les plus fascinantes du cosmos.

Figure 8 – Salle de projection du Planétarium de Strasbourg

Outre les spectacles à thèmes à destination de différents publics, le planétarium propose des visites guidées de la Grande Coupole de l'Observatoire ainsi qu'une visite libre de la salle d'exposition permanente baptisée « La Crypte aux étoiles ».

Régulièrement, des soirées d'observations sont proposées dans les jardins de l'Observatoire ainsi qu'avec la Grande Lunette.

Le Planétarium de Strasbourg possède également un planétarium mobile. Celui-ci se présente sous la forme d'un dôme gonflable, de quatre mètres cinquante de diamètre, alimenté par un ventilateur silencieux. Il possède un système de projection Cosmodyssée IV de 1500 étoiles permettant de simuler les constellations, le ciel à différentes latitudes et les mouvements de la Terre. Installé dans une salle de classe, une salle polyvalente, une bibliothèque, il permet d'initier le jeune public à l'astronomie sans contraintes météorologiques ou de pollution lumineuse.

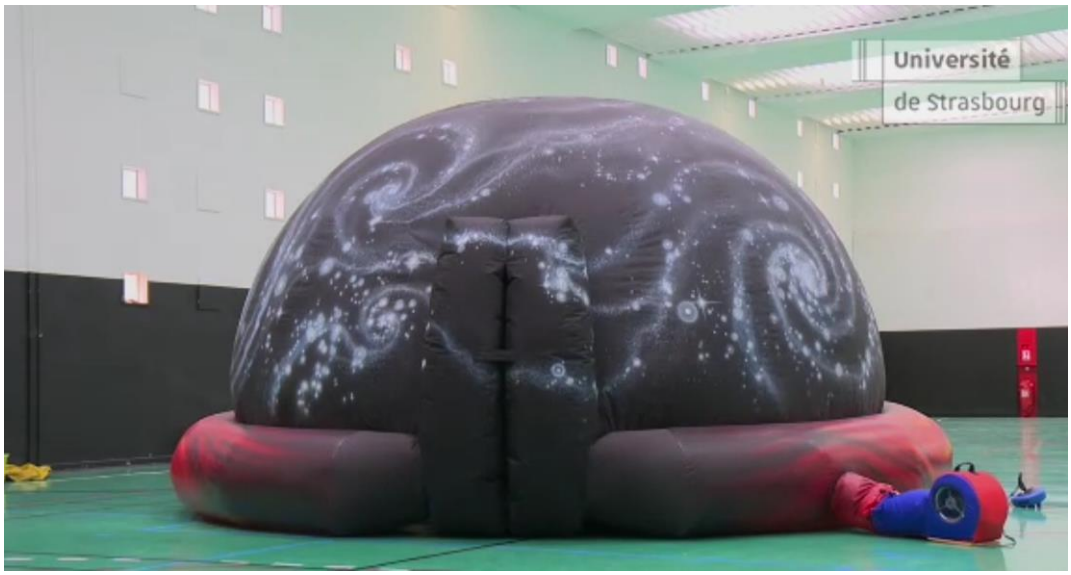


Figure 9 – Dôme gonflable du Planétarium de Strasbourg

II) DEROULEMENT DU STAGE

1) MISE EN CONTEXTE

1.1) NAISSANCE DU SUJET

Le Centre de Données astronomiques de Strasbourg propose plusieurs services à la communauté dont Aladin, un atlas interactif du ciel.

Cette application, développée en Java, permet d'afficher des images du ciel et offre également une navigation progressive dans de grands relevés d'images issus des missions spatiales. Elle est utilisée aussi bien dans un cadre professionnel que dans une démarche éducative.

Les nouveaux planétariums numériques fournissent de nouvelles possibilités d'affichage avec l'utilisation d'un projecteur permettant d'afficher des vues du ciel sur un dôme.

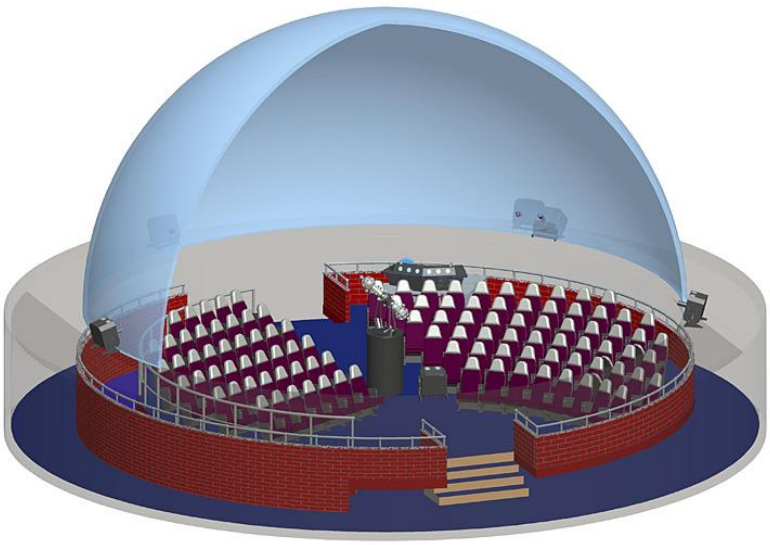


Figure 10 – Illustration d'un dôme utilisant un projecteur SKYMASTER

Le Planétarium de Strasbourg est quant à lui doté d'une demi-sphère de 8.20m de diamètre et d'un projecteur Barco F50 doté d'une résolution de 2560*1600 pixels.

Figure 11 – Projecteur Barco F50

Dans l'optique de pouvoir exploiter Aladin au planétarium, le CDS a commencé le prototypage d'un mode ''fisheye'' qui est une projection ARC (Zénithale équidistante), permettant de déformer l'image pour qu'une fois projetée sur le dôme elle apparaisse correctement.

Le stage a donc consisté à créer une interface en java permettant de gérer ce qu'Aladin projette sur le dôme à partir d'un moniteur de contrôle. Ce travail a été effectué en collaboration avec des personnes du Planétarium de Strasbourg afin de définir leurs besoins et également dans le but de réaliser des tests validant les différentes fonctionnalités.

1.2) L'EXISTANT

Pour comprendre au mieux les besoins du planétarium, j'ai commencé par effectuer un petit inventaire des outils actuellement utilisés pour leurs présentations. Sur leur site internet, 4 logiciels sont cités :

CELESTIA

Il s'agit d'un logiciel libre, multiplateforme, open source qui affiche des rendus en temps réel, tridimensionnels et photo-réalistes du système solaire, de la galaxie et même de l'univers. C'est un outil de valeur pour l'enseignement de l'astronomie utilisé chez les particuliers, dans les écoles, les musées et les planétariums du monde entier.

Figure 12 – Vue de la Terre avec le logiciel Celestia

AVL

L'Atlas Virtuel de la Lune est un logiciel qui ravira les amateurs de l'observation sélénique autant que les astronomes amateurs, en les invitant à découvrir notre satellite naturel sous toutes ses coutures.

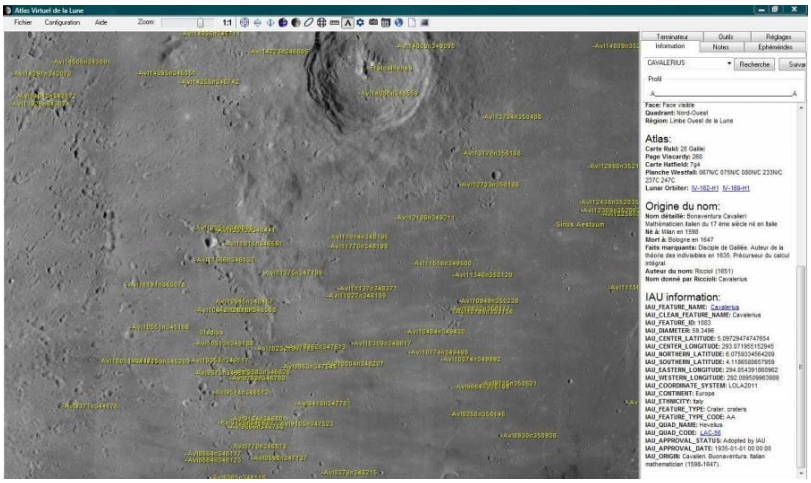


Figure 13 – Ecran principale d'ATLUN

WORLD WIND

Ce logiciel qui permet à partir d'images satellites de la NASA récupérées en ligne de voir notre planète avec une qualité et une finesse de détails tout à fait remarquables.

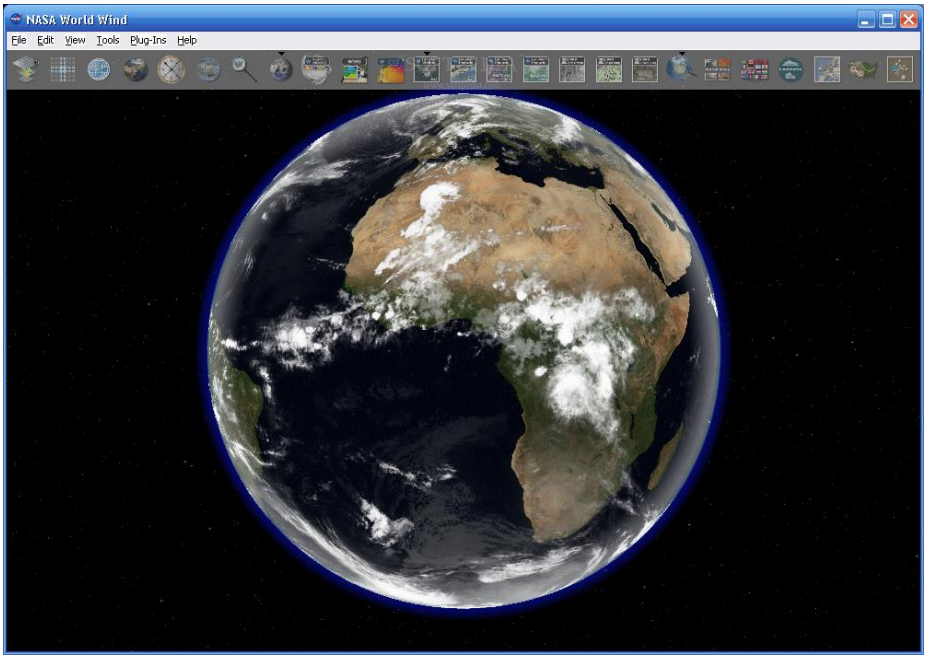


Figure 14 – Vue de la Terre avec World Wind

STELLARIUM

Cet outil est un logiciel pour planétarium open source. Il permet d'afficher un ciel réaliste en 3D, comme si vous le regardiez à l'œil nu, aux jumelles ou avec un télescope. Il permet d'observer la voûte céleste en temps réel en fonction de votre lieu géographique, avec la position des planètes, les dessins des constellations, plus quelques informations sur les étoiles brillantes. Ce logiciel est compatible avec les rétroprojecteurs capables de projeter une image à 360° sur voûte hémisphérique, ce qui permet donc une immersion toujours plus grande. Sur la figure 9 présente ci-contre, nous avons un exemple de rendu.

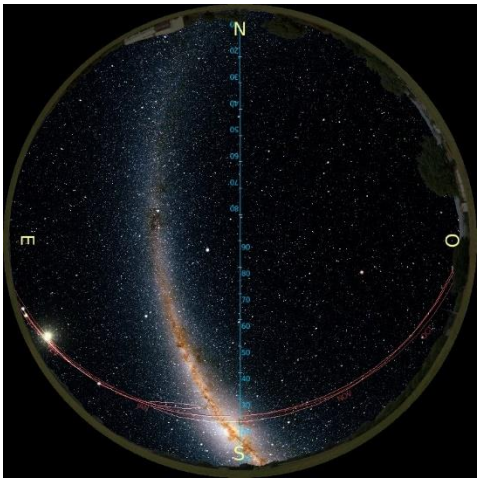


Figure 15 – Voie lactée par Stellarium 360

C'est ce dernier auquel je me suis le plus intéressé, car c'est celui le plus utilisé des 4 et qui s'apparente le plus à Aladin.

1.3) LES BESOINS

Pour se familiariser avec le contexte d'utilisation ainsi qu'aux besoins du planétarium, j'ai eu l'occasion de participer à une de leur démonstration publique. Celle-ci s'est déroulée en deux parties, la première a été effectuée avec Stellarium et la seconde a été un film documentaire.

Dans cette première partie j'ai remarqué un nombre abondant de fonctionnalités intéressantes. L'intervenant s'est servi du logiciel pour présenter des objets célestes qu'il pouvait mettre en avant en encerclant et parfois même en zoomant. Il était également possible de relier les éléments pour faire apparaître les constellations.

Parmi les fonctionnalités les plus pertinentes, j'ai noté :

- La possibilité de mettre en avant des objets célestes.
- La possibilité d'apporter des modifications à l'image (pour ajouter les constellations ou d'autres informations).
- La possibilité de visualiser l'état de la voûte céleste à partir d'un point de vue terrestre, à une date et heure précise (Stellarium permet également d'accélérer le temps pour voir le mouvement des étoiles).
- La possibilité de lancer des scénarios effectuant des opérations prédéfinies.

Aladin possède déjà un grand nombre de ces fonctionnalités de base ce qui permettrait de l'utiliser de manière similaire à Stellarium. Parmi celles présentées précédemment, il ne permet juste pas de faire intervenir la notion de temps.

Par contre Aladin présente plusieurs autres avantages :

- Beaucoup plus d'informations avec plus de 300 Surveys HiPS.
- La grande profondeur des surveys permet d'observer les objets de très près.
- Accès à des données provenant de Simbad et VizieR.
- La possibilité d'observer la voûte céleste dans plusieurs longueurs d'ondes.

Après un premier test d'Aladin sur le dôme, nous avons cependant rapidement réalisé que l'interface existante, représentée ci-dessous, n'était pas du tout adaptée à telle utilisation.

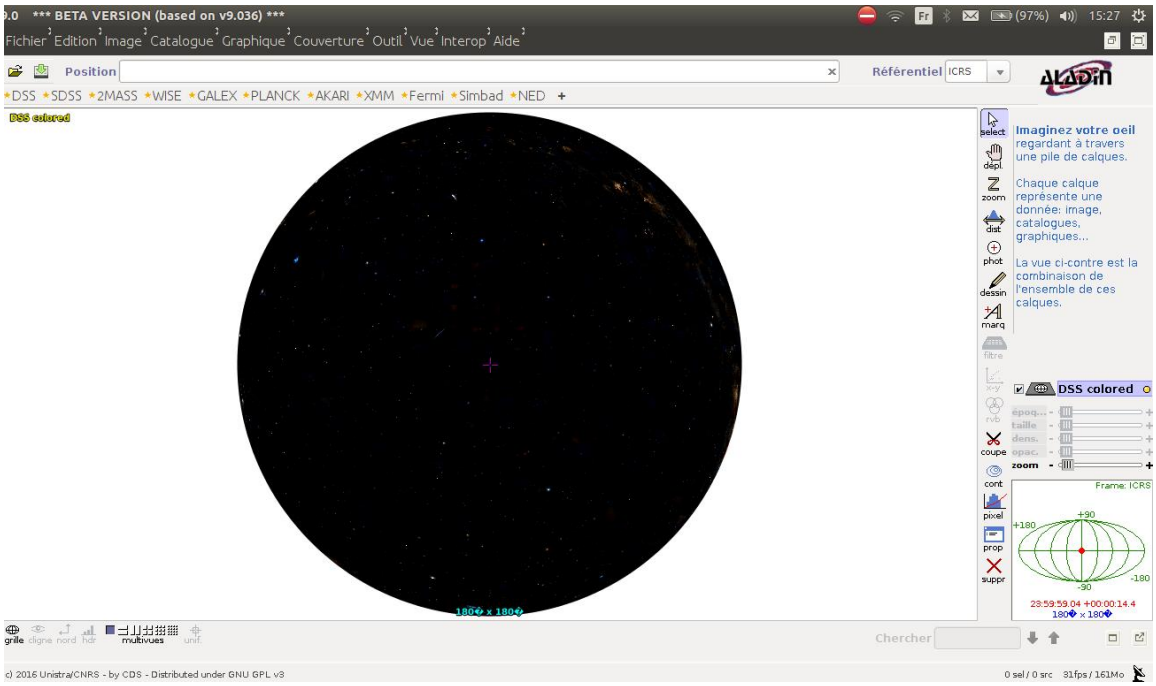


Figure 16 – Interface habituelle d'Aladin

Le fait de projeter le logiciel sur le dôme rendait très difficile l'accès aux menus d'Aladin, car il est conçu pour être utilisé sur un seul écran. Sur le moniteur principal, il n'y avait donc aucune possibilité de contrôle.

Malgré la mise en place d'une projection d'un mode ''Cinéma'', spécifiquement conçu par Pierre FERNIQUE pour le Planétarium, il manquait toujours un système de contrôle.

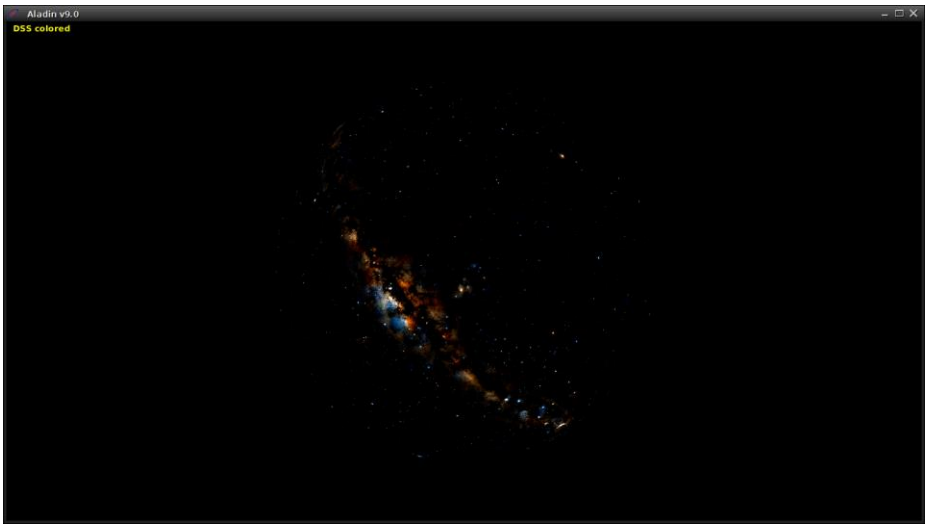


Figure 17 – Rendu mode ''Cinéma'' d'Adadin

Au niveau de la projection, les résultats était corrects mais dès que l'on zoomait, des déformations se faisaient voir sur les objets célestes. Cette partie du travail ne me concernait cependant pas.

Après cette première analyse des besoins nous avons défini qu'il fallait créer une interface permettant :

- La création et l'exécution de scénarios.
- L'envoi de commande à Aladin dont :
    - Le chargement de surveys.
    - La sélection du survey actuellement visible.
    - Un slider permettant de zoomer.
- Un lancement simple pour que l'interface s'affiche directement sur l'écran principal et que la projection d'Aladin se fasse sur le dôme.

2) SOLUTIONS ENVISAGEES

2.1) ETAT DE L'ART

Pour répondre aux besoins, j'ai tout d'abord relevé les différentes possibilités pouvant permettre d'envoyer des instructions à Aladin.

SCRIPTS PYTHON/PERL

Dans la documentation d'Aladin, on y retrouve des scripts Python et PERL décrivant la manière de transmettre des commandes. Ci-dessous, un exemple de code Python que j'ai rédigé pour effectuer une rotation dans le but d'imiter le déplacement de la voûte céleste en fonction du temps.

```
1  #!/usr/bin/env python
2  import subprocess
3  import time
4
5  p = subprocess.Popen(['java -jar ./AladinFisheye.jar'],
6                       shell=True,
7                       stdin=subprocess.PIPE
8                         )
9  p.stdin.write('get hips(P/DSS2/color)\n');
10 p.stdin.write('zoom 180 deg\n');
11
12 movePosition='0 47';
13 x=0.0
14 while x<360:
15     movePosition=str(x) + ' 47\n';
16     p.stdin.write(movePosition);
17     #time.sleep(0.00016);
18     x+=0.1;
19 #time.sleep(30);
20
21 #p.stdin.write('quit\n')
22 p.wait()
```

Figure 18 – Extrait de code Python communiquant avec Aladin

Une seconde solution aurait été la mise en place d'un serveur TomCat traitant les requêtes utilisateurs et renvoyant la réponse obtenue d'Aladin.

On peut apercevoir la structure sur le schéma suivant :

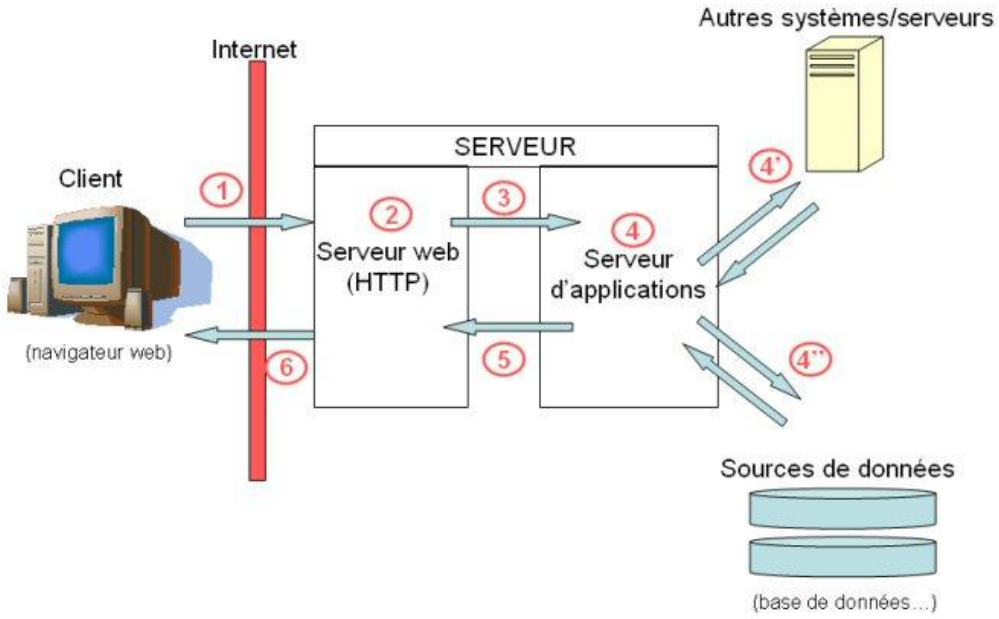


Figure 19 – Architecture TomCat

A l'étape 1 le client émet une requête à partir de l'interface pour demander une ressource au serveur (par exemple un survey dans notre cas).

A la seconde un serveur web (Apache) traite la requête entrante. Cependant celui-ci ne connait pas de lui-même la réponse attendue. Il la transmet alors via un connecteur (étape 3).

Le serveur d'application (TomCat) prend le relais en récupérant les données nécessaires en interrogeant les sources de données (Aladin dans notre cas). Puis il reconstitue une réponse qu'il repasse au serveur HTTP, qui la renvoie à son tour au client.

L'avantage de cette solution est que l'on peut faire abstraction du type d'interface utilisée par le client. Celle-ci peut tout aussi bien être une application tablette ou une application web.

Dans le cas d'une application web, il ne serait donc pas nécessaire d'effectuer une quelconque installation préalable.

J'ai échangé avec Laurent MICHEL à propos de cette solution qu'il avait utilisée pour réaliser une application tablette ainsi qu'une interface web exploitant Aladin et nommée ArchesWalker visible sur la figure 20.

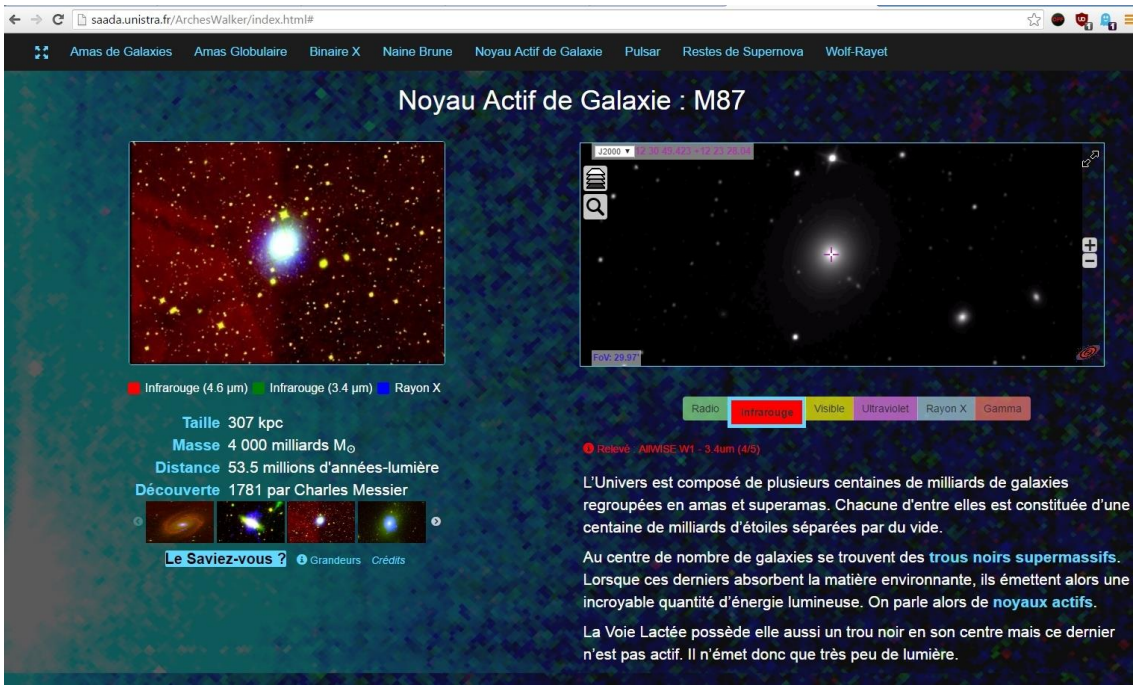


Figure 20 – Interface WEB d'ArchesWalker

ArchesWalker possédait de nombreuses fonctionnalités intéressantes pouvant correspondre à l'idée que je me faisais de l'interface à réaliser pour le planétarium, comme par exemple le fait de pouvoir choisir des types de longueur d'onde différentes pour un même objet.

Malheureusement, la mise en place d'une telle architecture côté serveur en plus de la création de l'interface utilisateur aurait pris bien plus de temps. De plus, durant la présentation à laquelle j'ai assisté, l'intervenant n'avait pas besoin d'être mobile lorsqu'il donnait ses explications, ce qui écartait également le besoin d'une application tablette. Cette piste a donc été écartée.

PLUGIN INTEGRE

La troisième solution explorée a été la création d'un plugin dédié à Aladin. L'interface se présenterait donc sous la forme d'un simple jar à placer dans le dossier ''.plugin'' d'Aladin.

Dans ce cas-là, le développement se ferait dans le même langage, c'est-à-dire le Java. L'implémentation de plugin était très bien documentée et la présence d'un nombre important d'exemples a été d'une très grande aide.

Pour la création d'un plugin, il suffisait de créer une classe héritant de la classe mère AladinPlugin et de compléter les attributs et méthodes héritées.

Cela donnait alors accès à une multitude d'attribut permettant d'interagir avec Aladin. Il était par exemple possible d'envoyer des requêtes scripts directement à Aladin avec la commande :

aladin.execCommand(''Commandes ;'') ;

Le manuel présentant la liste des commandes disponibles est présent dans la bibliographie.

De plus pour lancer le plugin, il suffisait d'attribuer une commande script à ce dernier afin de l'appeler à l'exécution d'Aladin :

java –jar Aladin.jar –script=''AladinController''

La simplicité d'utilisation et de développement du plugin m'ont conforté dans mon choix d'utiliser cette solution pour la création de mon interface.

2.2) OUTILS

Durant le stage, j'ai disposé d'une machine sous le système d'exploitation Linux avec un dual screen pour simuler le dôme avec un écran et l'interface de contrôle sur l'autre.

Pour le développement du plugin en Java, au départ j'avais simplement opté de compiler avec un script bash, mais avec la complexification du projet lorsque j'ai commencé à utiliser des bibliothèques externes, j'ai choisis d'utiliser un IDE. J'ai principalement utilisé Intel IJ que j'ai déjà eu l'occasion de manipuler à plusieurs reprises durant ma formation. J'ai cependant aussi dû apprendre à maîtriser l'IDE Eclipse, car lorsque j'ai voulu apporter des modifications au code d'Aladin, celui-ci était déjà configuré avec ce dernier.

A ces IDE, j'ai également ajouté les plugins correspondant pour effectuer la gestion de versions avec SVN.

Au départ j'ai également pensé à utiliser des plugins pour faciliter la création d'interface java SWING, mais finalement je n'en ai pas eu l'utilité.

3) DEVELOPPEMENT

3.1) EBAUCHE GRAPHIQUE

Pour pouvoir organiser le développement de la meilleure façon, j'ai commencé à réaliser une ébauche de l'interface telle que je la voyais avec les fonctionnalités que l'on m'avait indiquées.

J'ai décidé de réaliser deux dispositions bien distinctes, dont je détaillerai le fonctionnement dans la partie dédiée aux modules.

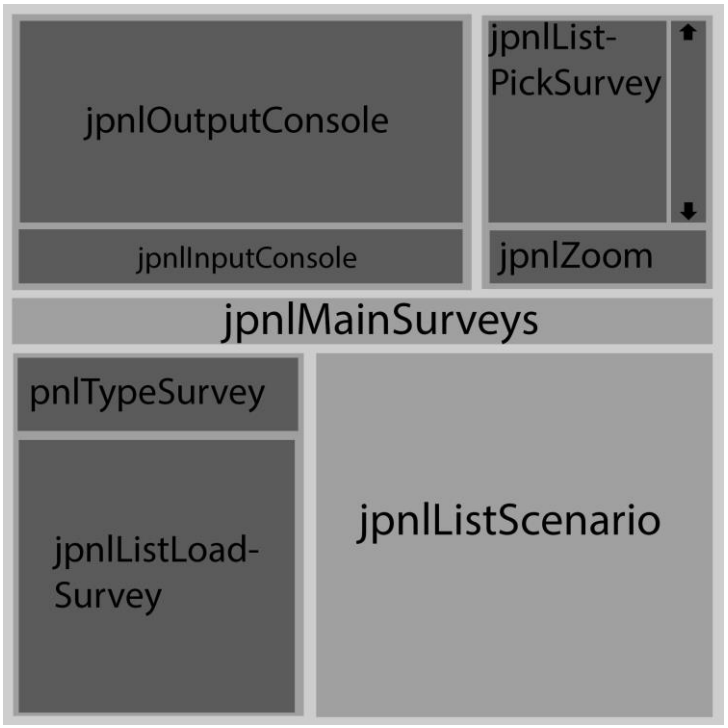


Figure 21 – Ebauche du panneau principal

La première partie, visible ci-dessus, permettrait d'interagir principalement avec Aladin.

Les éléments **jpnlOuputConsole** et **jpnlInputConsole** forment le module Console permettant d'envoyer des requêtes respectant les indications du manuel de commande (URL en bibliographie). Ce module fonctionne de manière très similaire à une console linux.

L'élément **jpnlMainSurveys**, contiendrait une liste de boutons permettant de charger les surveys les plus intéressants.

Les éléments **pnlTypeSurvey** et **jpnlListLoadSurvey** seraient utilisés pour charger de nouveaux surveys. L'élément **jpnlTypeSurvey** permettrait de choisir le type de longueur d'onde et le **jpnlListLoadSurvey** afficherait les surveys correspondants au type sélectionné.

L'élément **jpnlZoom** comme son nom l'indique serait simplement un slider permettant de zoomer.

Le **jpnlListPickSurvey** contiendrait la liste des surveys actuellement chargé par Aladin, et permettrait de switcher entre ceux-ci.

Et finalement le **jpnlListScenario** serait une simple liste de boutons permettant de lancer les scénarios.

Le panneau secondaire, visible ci-dessous, serait plutôt tourné vers la création de scénarios.

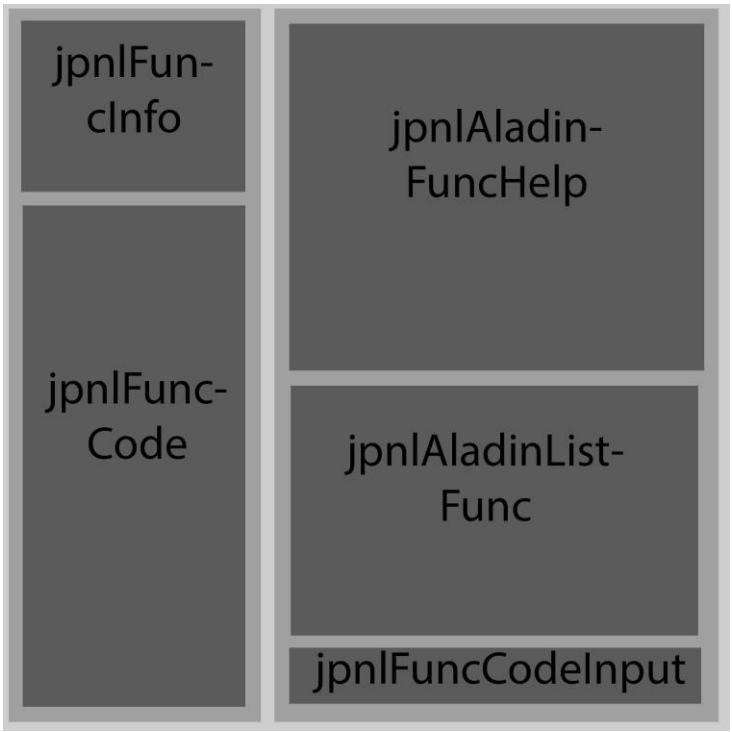


Figure 22 – Ebauche du panneau secondaire

L'élément **jpnlFuncInfo** serait divisé en plusieurs champs pour indiquer les informations importantes sur le scénario comme le nom, les arguments et la description.

L'élément **jpnlFuncCode** contiendrait la liste de toutes les instructions du scénario.

L'élément **jpnlFuncCodeInput** permettrait de rajouter des instructions au scénario en cours.

L'élément **jpnlAladinListFunc** contiendrait la liste des commandes disponible, et **jpnlAladinFuncHelp** afficherait l'aide correspondante à la commande sélectionnée.

En résumé, l'utilisation de la première partie (hormis le **jpnlListScenario**) serait entièrement dédiée à réaliser des tests de commandes pour voir comment réagit Aladin, dans l'optique de créer des scénarios. La seconde partie concernerait donc la création et l'édition de ces scénarios, de manière à confectionner des démonstrations préétablies pour une utilisation au planétarium.

3.2) CHOIX GRAPHIQUES

STYLE GRAPHIQUE

En étudiant le code source d'Aladin durant les premières semaines, j'ai appris qu'il était possible d'utiliser des styles prédéfinis appelés LookAndFeels. L'utilisation de cette propriété des interfaces graphiques de Java permet d'attribuer un style cohérent à une GUI très simplement.

J'ai remarqué pendant la séance que le moniteur de l'intervenant était recouvert d'un filtre plastique rouge pour limiter l'émission de lumière parasite. J'ai donc décidé de garder un style sobre et sombre pour que le moniteur de l'intervenant ne fasse pas trop de lumière durant la séance.

Pour cela j'ai épluché les différents LookAndFeel présents de base dans Java, mais aucun n'a retenu mon attention. J'ai donc utilisé une bibliothèque externe appelée JTattoo, qui regroupe un grand nombre de styles différents. Parmi ceux-ci j'ai opté pour le ''HiFiLookAndFeel'' qui répondait à tous mes critères.

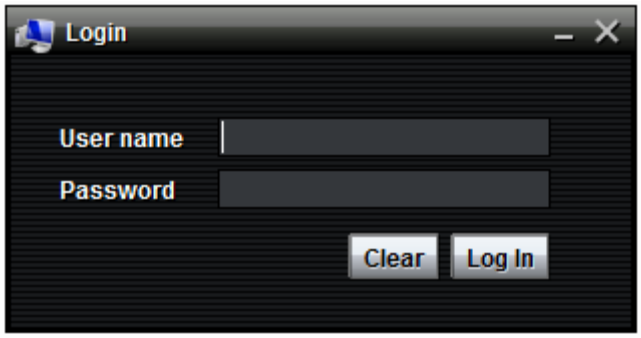


Figure 23 – Apparence du style HiFiLookAndFeel

DISPOSITION DES ELEMENTS

Durant le stage j'ai effectué beaucoup de modifications graphiques, tant au niveau de la disposition des éléments qu'au niveau de la manière de les disposer.

Au départ j'ai eu beaucoup de difficultés à trouver une manière simple de réaliser l'interface, car je n'avais pas tellement d'expérience dans ce domaine. J'ai donc essayé au début de trouver un plugin pour Intel IJ permettant de réaliser une GUI en déplacement des éléments, mais cela n'a pas été convainquant. J'ai donc simplement opté pour l'utilisation de SWING de base, avec en plus la bibliothèque ''SpringUtilities'' d'Oracle.

3.3) DEVELOPPEMENT DES MODULES

Cette partie concernera la façon dont j'ai réalisé les modules composant mon interface de contrôle. Celle-ci abordera donc une approche plus technique. Pour pouvoir visualiser l'interface dans sa globalité veuillez-vous référer aux annexes.

MODULE CONSOLE

Durant le développement Pierre FERNIQUE m'a suggéré d'inclure à l'interface un moyen d'envoyer des commandes à Aladin tout en pouvant facilement récupérer l'historique des requêtes effectuées. Dans le but par exemple de les intégrer par après dans un scénario.

Pour répondre à ce besoin, j'ai réalisé un module, visible ci-dessous, fonctionnant de manière similaire à une console Linux.

Figure 24 – Module Console

Ce module est très simple d'utilisation. En bas, il s'agit d'un TextArea pour laisser la possibilité à l'utilisateur d'écrire des enchaînements de requêtes. Au-dessus j'ai également utilisé une textArea avec un StyleDocument pour pouvoir utiliser des couleurs différentes selon les types de messages renvoyés par Aladin.

De la même manière qu'une console, il est possible d'accéder aux commandes récemment effectuées en effectuant la combinaison CTRL+↑ ou CTRL+↓. Pour réaliser cela, je stocke les commandes dans une ArrayList à laquelle j'accède ensuite lorsque la combinaison de touches est utilisée.

J'ai décidé de faire transiter toutes les opérations effectuées par mon interface par la console, c'est pourquoi j'ai intégré une méthode ''doCommand()'' à ma classe ConsolePanel. De cette manière, dès qu'il y avait besoin de communiquer avec Aladin j'utilisais cette méthode, qui faisait appelle à ''aladin.execCommand(''*Commande ;*'')'' utilisable lorsqu'on implémente un plugin. Grâce à cela je pouvais afficher les réponses ou les avertissements d'Aladin directement dans l'output de ma console.

Pour la création de ce module, visible ci-dessous, j'ai dû m'y prendre à plusieurs reprises. Car au départ j'avais réalisé un système fonctionant de la même manière que celui d'ArchesWalker, où l'on cliquait sur des types de fréquences (infrarouge, visible, etc…) et cela tirait au sort un survey correspondant au type.

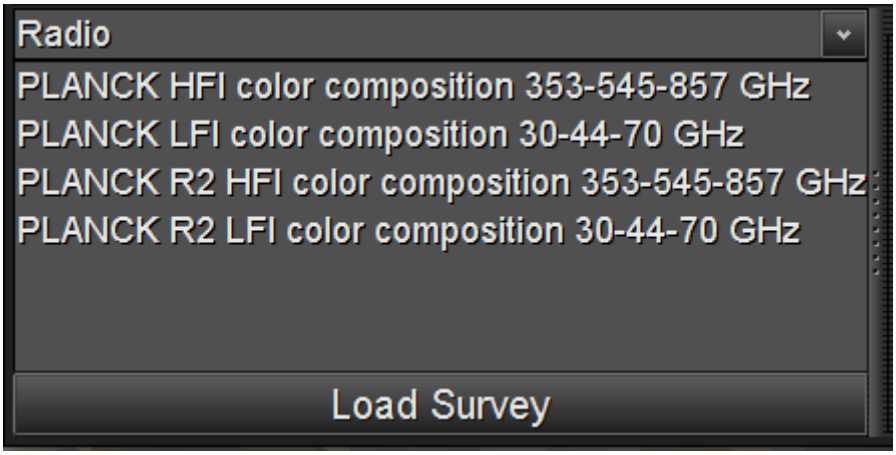


Figure 25 – Module Chargement Survey

Finalement je suis arrivé à ce rendu, ou la sélection du type de fréquences se fait par l'intermédiaire d'un menu déroulant. Cette JComboBox propose les valeurs suivantes : Radio, Infrared, Optical, UV, X-ray et Gamma.

Cela permet à l'utilisateur de facilement sélectionner le type de survey qui l'intéresse.

La partie la plus complexe de ce module a été la manière de récupérer les ID des surveys au lancement du plugin. Plusieurs solutions s'offraient à moi, comme l'écriture en dur ou l'émission d'une requête à Aladin. J'ai finalement recouru à un site nommé dans la bibliographie auquel j'envoyais des requêtes GET spécifiant les critères des surveys que je souhaitais utiliser.

Il a fallu ensuite traiter les réponses obtenues au format JSON. Je me suis servi pour cela de la bibliothèque open source Gson développée par Google. Celle-ci permet de convertir un objet Java dans sa représentation JSON et vice versa.

Une fois Gson maitrisé, il fut très simple de récupérer mes surveys dans une ArrayList pour les classer selon leur type. Pour cela la création d'une classe Survey (visible à l'annexe 4) fut nécessaire.

Ce module est très important, car c'est celui qui met en avant toute la spécificité d'Aladin. Il permet de switcher entre les surveys actuellement chargés.

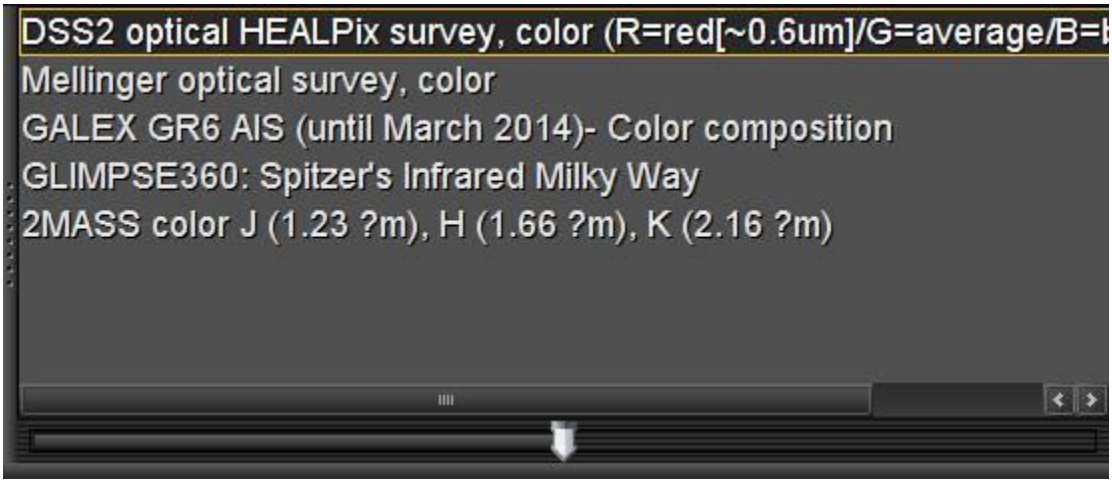


Figure 26 – Module Survey Picker

Pour la création de ce module, j'ai rajouté un test dans la méthode ''doCommand()'' de la console, qui vérifie si la requête correspond au chargement d'un survey. Si cela est effectivement le cas, alors je stocke l'objet Survey correspondant dans une JList.

Lorsque l'utilisateur clique sur un des surveys listés, la commande ''cview'' avec l'id est envoyé à Aladin ce qui change survey courant.

Pour avoir un aperçu d'utilisation, les annexes 6 à 9 illustrent l'objet NGC1365 provenant de 4 surveys.

On peut apercevoir également au bas de ce module le slider donnant la possibilité de zoomer sur le survey courant. Au départ ce slider envoyait une requête uniquement quand le clic de la souris était relâché. Ce qui évitait d'envoyer trop de commandes à Aladin et qui pouvait éventuellement faire crasher l'application. Mais Pierre FERNIQUE m'a indiqué vouloir garder un slider qui, dès que le curseur bouge, influe directement sur le zoom. J'ai donc répondu à sa demande en implémentant sa méthode. Cela permet entre autre de zoomer progressivement, mais rend également l'application susceptible aux crashs.

Tout d'abord, je souhaite introduire la différence entre ce qu'est une brique et ce qu'est un scénario. Une brique est une suite de commande Aladin qui lorsqu'on l'exécute, traite les commandes une par une. Le scénario est quant à lui constitué de briques et est affiché dans le module de lancement de scénario.

Cette différenciation permet de créer de regrouper des enchaînements de commandes dans une brique qui pourra être par la suite intégrée à plusieurs scénarios.

Ce module est celui m'ayant demandé le plus de temps de réflexion et de développement, celui-ci est visible ci-dessous.

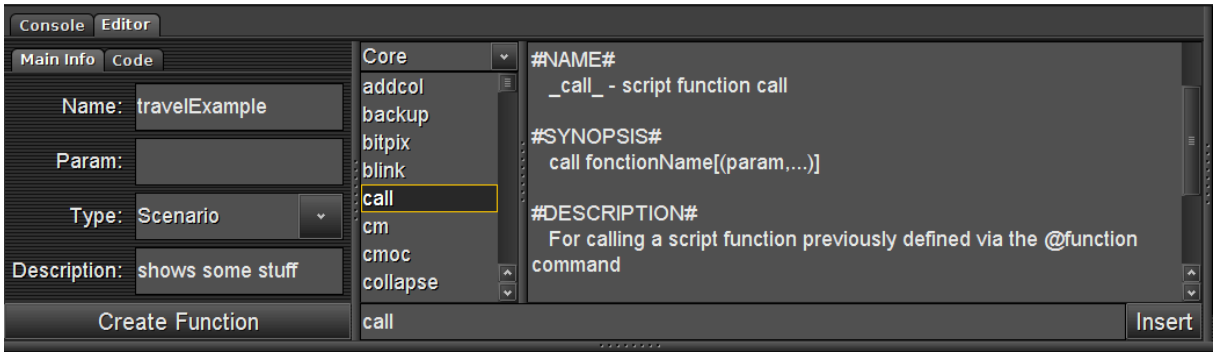


Figure 27 – Module Création Scénario/Brique

Lors de la première itération de ce module, il suffisait de compléter les champs correspondant aux informations sur le scénario, visible sur la partie gauche de la figure 27. Et de rédiger ensuite le code dans une textArea, présente dans l'onglet Code. Mais lorsque j'ai présenté cela au personnel du Planétarium, ils m'ont indiqué à juste titre que ne connaissant pas les commandes Aladin, il leur serait difficile de créer un scénario sans aucune aide.

J'ai donc directement intégré sur la droite une liste des fonctions disponibles classées par type. Les fonctions Core sont les fonctions provenant directement d'Aladin, les scénarios et briques proviennent de l'utilisateur. Comme l'on peut apercevoir sur la figure 27, en sélectionnant une fonction dans la liste on obtient alors toute l'aide nécessaire à sa compréhension.

Pour créer un scénario, rien de plus simple il suffit de remplir les champs de l'onglet MainInfo, et puis accéder à l'onglet Code visible sur la figure 28.

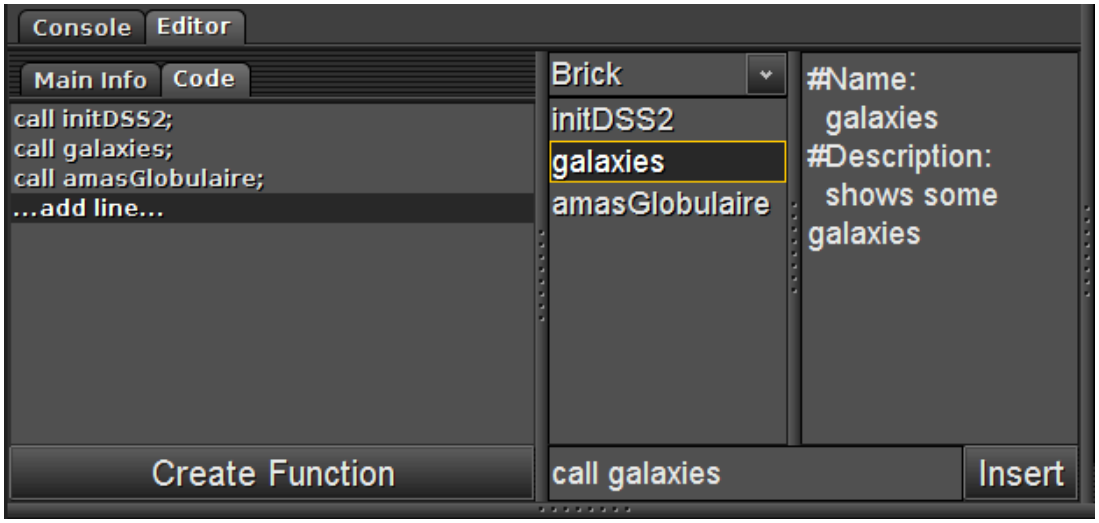


Figure 28 – Module de Création, onglet Code

Pour l'écriture du code j'ai opté pour une JList. Celle-ci convient du fait que les commandes Aladin sont généralement très courtes, un scénario se résume donc à de simples appels de fonctions successives.

Pour ajouter une ligne dans cette JList, il faut sélectionner l'élément ''…add line…'' et écrire dans la textBox en bas à droite la commande à insérer. Dans le cas où l'on souhaite éditer une ligne, il suffit de double-cliquer sur l'instruction. Cela aura pour effet de changer le texte du bouton ''Insert'' en ''Edit'' et permettra de revenir sur le code. Et pour supprimer une instruction, il suffit d'utiliser le clic droit.

Il est également possible d'éditer ou de supprimer des scénarios et briques en utilisant le clic droit de la souris sur l'élément où l'on souhaite agir. Dans le cas d'une suppression un message de confirmation s'affichera.

Le plus gros du travail restait cependant l'enregistrement des scénarios/briques ainsi créés. Il fallait trouver un moyen de pouvoir facilement transférer des scénarios pour les utiliser sur un autre ordinateur par exemple.

Pour répondre à cette problématique, j'ai utilisé une fonctionnalité d'Aladin permettant de créer des fonctions en utilisant une commande appelée ''function''. Celle-ci correspondait exactement à mes problématiques, car Aladin sauvegarde ces fonctions selon un certain format dans le fichier ''Bookmarks.ajs'' présent dans le dossier caché ~/.aladin. Un extrait de ce fichier est présent à l'annexe 10.

MODULE LANCEMENT SCENARIO

Le module de lancement de scénario, visible à l'annexe 2, se présente sous la forme de boutons portant le nom du scénario et affichant la description correspondante lorsque le curseur de la souris passe par-dessus.

Ce module charge l'ensemble des scénarios disponibles à partir du fichier Bookmark.ajs lors du lancement. Lors de la création d'un scénario, le bouton associé est directement ajouté au module.

3.4) MODIFICATIONS APPORTEES A ALADIN

Durant le développement des modules de l'interface, j'ai dû modifier le code d'Aladin de manière à pouvoir accéder à certaines données ou fonctions qui étaient inaccessibles à partir du plugin.

L'ensemble des modifications est listé à l'annexe 13, en voici un résumé :

- Ajout d'un attribut type pour différencier les fonctions scénarios et briques.
- Ajout de fonctions permettant de récupérer les informations des scénarios et briques afin de pouvoir les afficher dans la partie aide de l'utilisateur.
- Modification de l'accès à la méthode ''saveLocalFunction()''de protected à public. Pour sauvegarder les scénarios/briques directement à la création de celle-ci, car Aladin n'effectuait cette sauvegarde qu'à la fermeture du logiciel.
- Modification de l'affichage du mode ''Cinéma'' au lancement. De sorte que l'interface s'affiche sur le moniteur principal et que la vue de la voûte céleste soit directement affichée sur le dôme.

III) BILAN

1) L'INTERFACE CORRESPOND-ELLE AUX ATTENTES

De manière générale, la possibilité de régulièrement tester mon application avec les personnes du planétarium, m'a permis de toujours m'adapter à leurs conseils et leurs demandes. Le fait de développer à l'aide d'un double écran pour me mettre dans les mêmes conditions qu'en production, m'a également aidé à me faire une idée des résultats sur le dôme lorsque je n'y avais pas accès.

De plus André SCHAAFF, François BONNAREL et Pierre FERNIQUE m'ont fait de nombreux retours durant nos réunions hebdomadaires.

Toutes ces conditions réunies m'ont permis de réaliser un plugin répondant aux attentes et besoins initiaux. C'est-à-dire obtenir une interface simple d'utilisation, permettant d'effectuer des requêtes à Aladin et de réaliser des scénarios en vue d'une utilisation au planétarium.

J'ai notamment eu d'excellents retours suite à ma dernière démonstration en octobre, où j'ai présenté la dernière version du projet.

2) PROPOSITION D'AJOUTS

S'il y avait une suite à ce projet, les quelques améliorations simples qui pourraient être apportées seraient les suivantes :

- L'ajout d'une commande dans Aladin mettant en pause le déroulement d'un scénario jusqu'à l'appui d'une touche spécifiée. Cela permettrait à l'intervenant de disposer du temps nécessaire à son explication durant une présentation.

- La correction du zoom qui selon moi ne devrait envoyer une requête à Aladin uniquement au relâchement du clic souris. Ou une autre possibilité serait de créer une commande Aladin zoomTo, similaire à goto qui effectuerait un zoom progressif à partir d'un degré initial jusqu'au degré d'arrivé.

3) CONCLUSION

Durant ce stage, j'ai été confronté à de nouveaux défis auxquels je n'avais pas encore eu l'occasion de faire face. J'ai pu mettre à profit les connaissances acquises lors de ma formation, mais il a fallu également compléter mes lacunes dans le domaine en me renseignant sur les manières de réaliser de tels projets. Il a fallu s'y reprendre à plusieurs fois pour certains modules avant d'obtenir les résultats escomptés.

Au début du stage, il m'a fallu beaucoup de temps pour assimiler le fonctionnement d'Aladin. Car c'est une application très complète et complexe qui a nécessité des années de développement. La présence d'une documentation fournie ainsi que l'aide de mes encadrants m'a donc été précieuse.

Comme j'ai déjà eu l'occasion de le dire durant ce rapport, les retours de mes encadrants ainsi que des personnes du Planétarium, m'ont permis d'adapter l'interface en fonction de leurs attentes. J'ai néanmoins bénéficié d'une certaine liberté quant à la conception globale de l'interface.

Les retours très positifs obtenus lors de la démonstration de la dernière version en octobre m'ont prouvé l'efficacité de cette méthode de travail.

Cela a été globalement une expérience très enrichissante, car ce n'est que le deuxième projet informatique que j'ai eu l'occasion de mener à bien.

GLOSSAIRE

API

« Application Programming Interface », interface de programmation. Ensemble normalisé de classes et de fonctions qui sert de façade par laquelle un logiciel offre des services à d'autres logiciels

BASH

Le « Bourse Again Shell » est le shell Linux par défaut.

BIBLIOTHEQUE

En informatique, une bibliothèque logicielle est une collection de fonctions.

FRAMEWORK

Ensemble cohérent de composants logiciels structurels.

GUI

« Graphical User Interface » traduit par interface homme-machine désigne le dispositif graphique permettant à un utilisateur de communiquer avec une machine.

HIPS

« Hierarchical Progressive Surveys » , c'est un mécanisme permettant d'accéder, de visualiser et de naviguer à travers un catalogue d'image. (A la manière de Google Earth)

HTTP

« Hypertext Transfer Protocol » est un protocole de communication client-serveur développé pour le World Wide Web

IDE

« Integrated Development Environment », Environnement de développement. Ensemble d'outils pour augmenter la productivité des programmeurs qui développent des logiciels.

JAVA

Langage de programmation orienté objet dont l'objectif principal est la création de logiciel très facilement portable sur les différents systèmes d'exploitation (UNIX, Windows, Mac OS).

JSON

« JavaScript Object Notation ». C'est un format de données textuelles dérivé de la notation des objets du langage JavaScript.

PLUGIN

Module d'extension. Paquet qui complète un logiciel hôte pour lui apporter de nouvelles fonctionnalités.

SHELL

Il s'agit d'un interpréteur de commande qui agit comme l'interface entre l'utilisateur et le système d'exploitation.

SURVEY

Catalogue d'image.

WORLD WIDE WEB CONSORTIUM

Egalement connu sous le sigle W3C, le World Wide Web Consortium est un organisme de normalisation à but non lucratif chargé de promouvoir la compatibilité des technologies de World Wide Web.

BIBLIOGRAPHIE

Site officiel d'Aladin :

http://aladin.u-strasbg.fr/

FAQ d'Aladin :

http://aladin.u-strasbg.fr/java/FAQ.htx

Manuel des commandes Aladin :

http://aladin.u-strasbg.fr/java/AladinScriptManual.gml

Création d'un plugin pour Aladin :

http://aladin.u-strasbg.fr/java/nph-aladin.pl?frame=plugins

Liste d'exemples de plugins pour Aladin :

http://aladin.u-strasbg.fr/java/nph-aladin.pl?frame=plugRep

URL permettant de récupérer des surveys :

http://alasky.unistra.fr/MocServer/query?dataproduct_subtype=*color*&hips_service_url=http://alasky.*&fields=ID,obs_title,hips_service_url,obs_regime&fmt=json

Manuel de l'utilisateur Aladin :

http://www.inaoep.mx/~isya28/lecciones/rodolfo_3b.pdf

ArchesWalker :

http://saada.unistra.fr/ArchesWalker/index.html#

Site du Planétarium de Strasbourg :

http://jardin-sciences.unistra.fr/planetarium/

Site de JTattoo :

http://www.jtattoo.net/

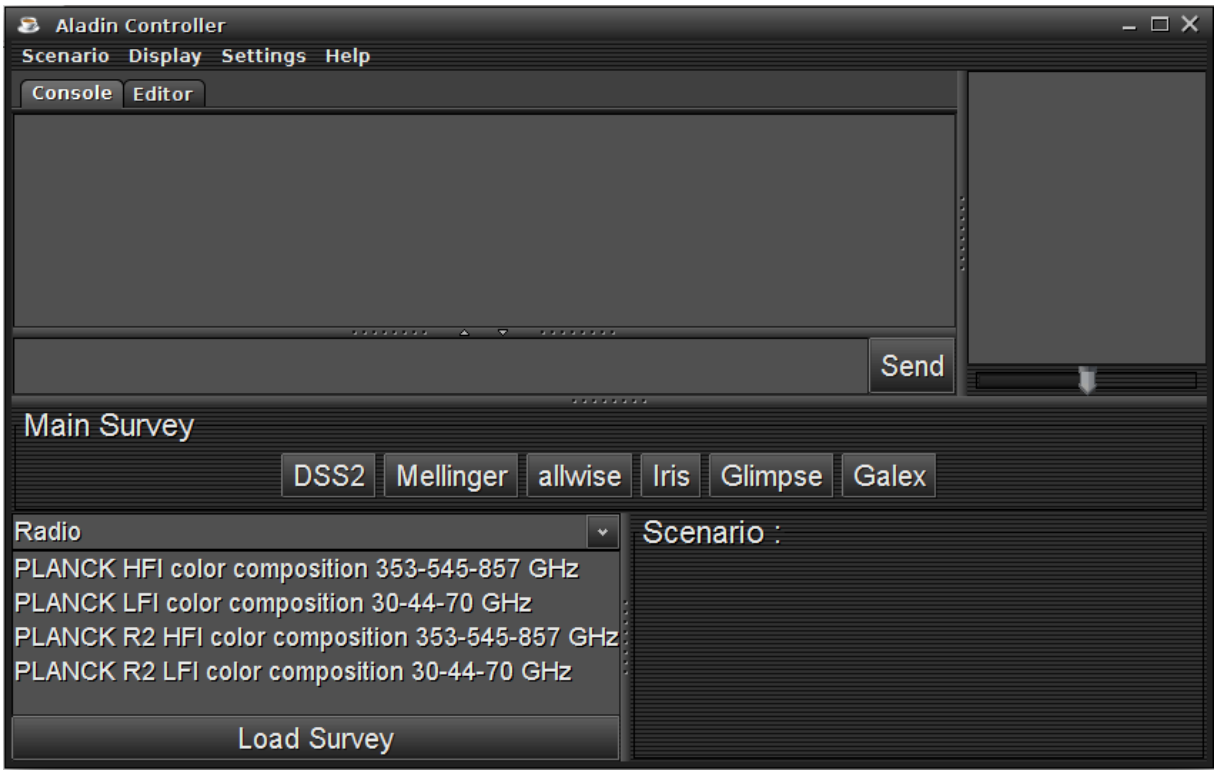
Github de Gson :

https://github.com/google/gson

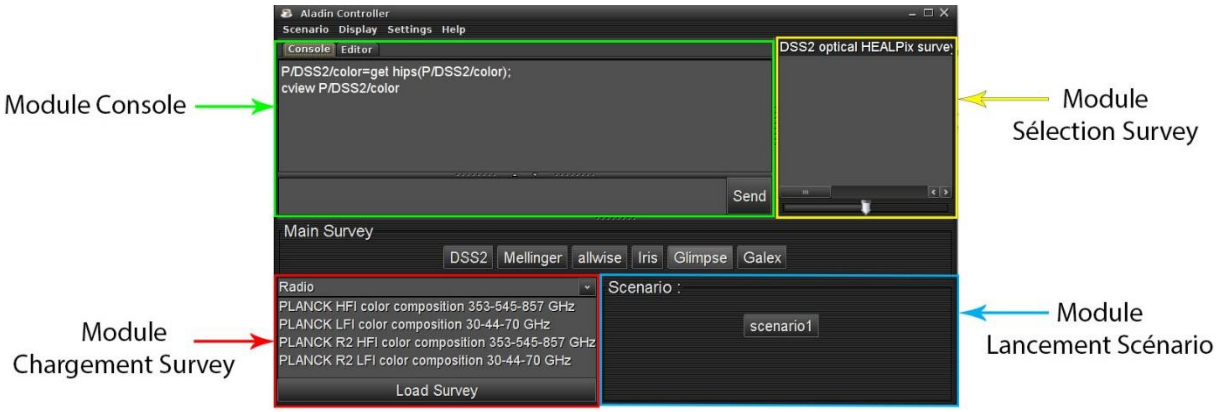
Wiki interne du CDS détaillant mes démarches :

http://cds.u-strasbg.fr/twiki/bin/view/Stages/ArnaudSteinmetz2

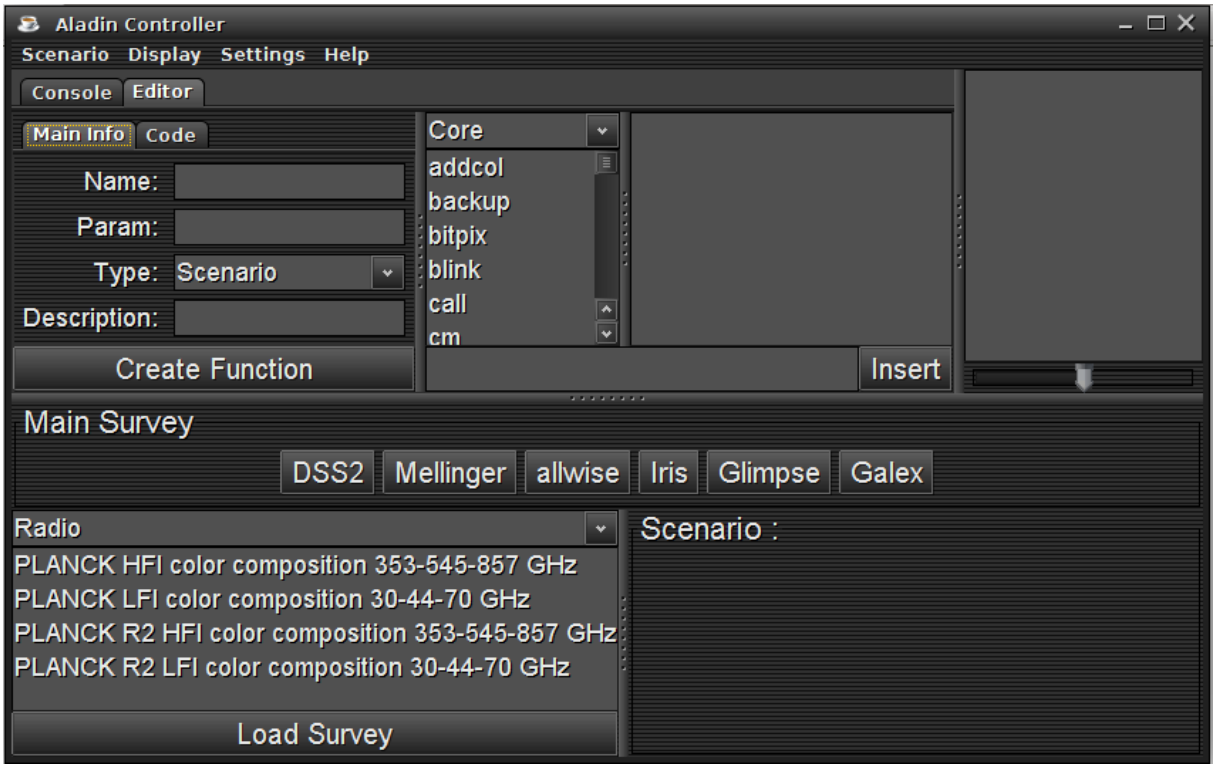
ANNEXES



Annexe 1 – Interface d'AladinController au lancement



Annexe 2 – Interface annotée d'AladinController au lancement



Annexe 3 – Interface dans le menu Editor d'AladinController

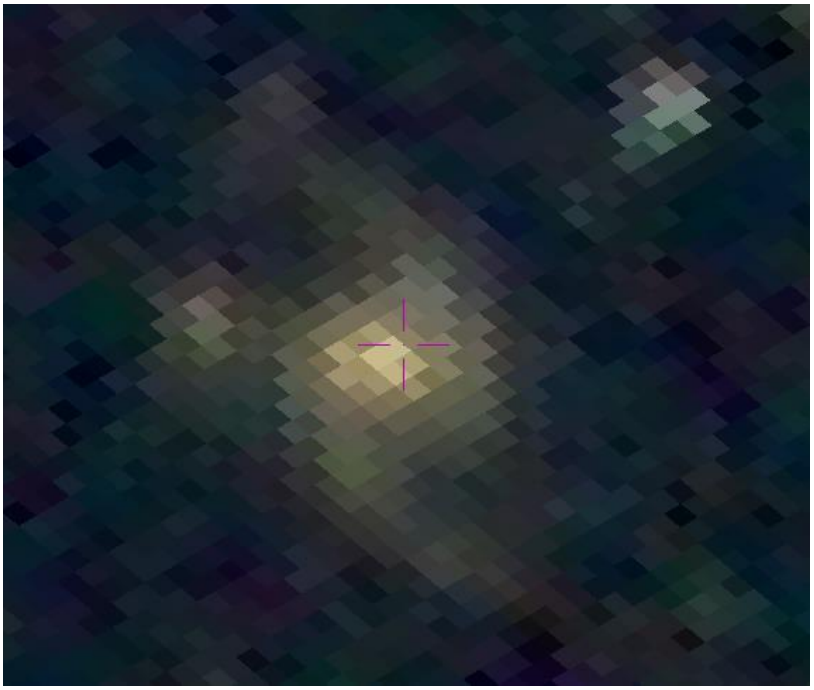
```java
class Survey {
    private String ID;
    private String obs_title;
    private String obs_regime;
    private String obs_id;

    public Survey(String ID, String obs_title, String obs_regime) {
        this.ID = ID;
        this.obs_title = obs_title;
        this.obs_regime = obs_regime;
    }

    public Survey(String ID) { this.ID = ID; }

    public String getID() { return ID; }

    public void setID(String ID) { this.ID = ID; }

    public String getObs_title() { return obs_title; }

    public void setObs_title(String obs_title) { this.obs_title = obs_title; }

    public String getObs_regime() { return obs_regime; }

    public void setObs_regime(String obs_regime) { this.obs_regime = obs_regime; }

    public String getObs_id() { return obs_id; }

    public void setObs_id(String obs_id) { this.obs_id = obs_id; }

    public String getHipsID() { return this.ID.replaceFirst("[^/]*/",""); }

    @Override
    public String toString() { return this.getObs_title(); }
}
```

Annexe 4 – Classe Survey

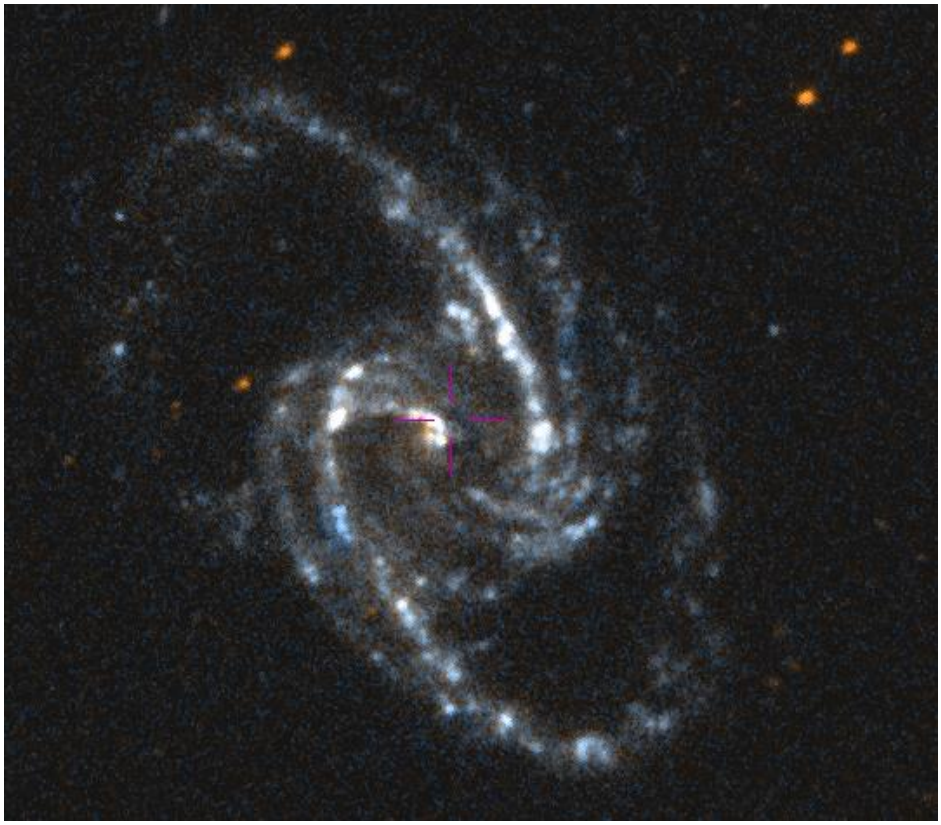
```java
private void initLoadSurvey() {
    Gson gson = new GsonBuilder().create();
    Vector<Survey> listInfoSurveyRadio = gson.fromJson(Planetarium.sendGetinit("Radio"), new TypeToken<Vector<Survey>>(){}.getType());
    Vector<Survey> listInfoSurveyInfrared = gson.fromJson(Planetarium.sendGetinit("Infrared"), new TypeToken<Vector<Survey>>(){}.getType());
    Vector<Survey> listInfoSurveyOptical = gson.fromJson(Planetarium.sendGetinit("Optical"), new TypeToken<Vector<Survey>>(){}.getType());
    Vector<Survey> listInfoSurveyUV = gson.fromJson(Planetarium.sendGetinit("UV"), new TypeToken<Vector<Survey>>(){}.getType());
    Vector<Survey> listInfoSurveyXray = gson.fromJson(Planetarium.sendGetinit("X-ray"), new TypeToken<Vector<Survey>>(){}.getType());
    Vector<Survey> listInfoSurveyGammaray = gson.fromJson(Planetarium.sendGetinit("Gamma-ray"), new TypeToken<Vector<Survey>>(){}.getType());

    surveyData.add(listInfoSurveyRadio);
    surveyData.add(listInfoSurveyInfrared);
    surveyData.add(listInfoSurveyOptical);
    surveyData.add(listInfoSurveyUV);
    surveyData.add(listInfoSurveyXray);
    surveyData.add(listInfoSurveyGammaray);
}
```

Annexe 5 – Code ‘’initLoadSurvey()’’ utilisant gson

Annexe 6 – NGC13565 avec DSS2



Annexe 7 – NGC1365 avec Mellinger

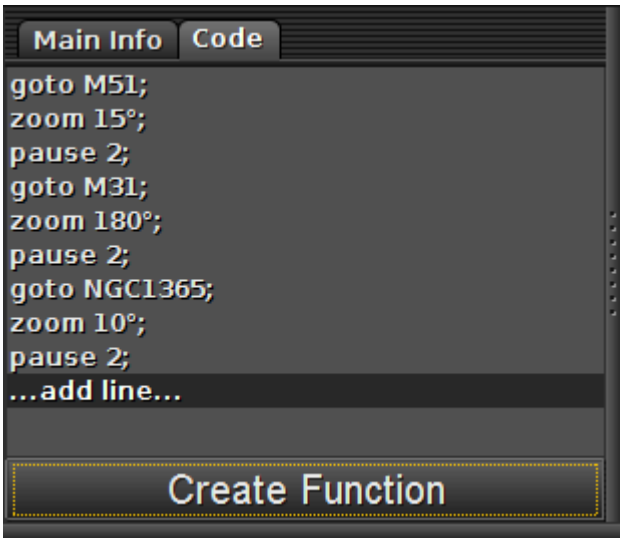


Annexe 8 – NGC1365 avec Galex

Annexe 9 - NGC1365 avec 2MASS

```
25  !Brick
26  #shows some stuff
27  function amasGlobulaire {
28  goto GCL15;
29  zoom 15°;
30  pause 2;
31  goto M15;
32  zoom 10°;
33  pause 2;
34  goto PLEIADE;
35  zoom 180°;
36  pause 2;
37
38  }
39
40  !Scenario
41  #shows some stuff
42  function travelExample {
43  call initDSS2;
44  call galaxies;
45  call amasGlobulaire;
46
47  }
```

Annexe 10 – Extrait de Bookmarks.ajs



Annexe 11 – Exemple de brique, visite de galaxies.

```java
//Event called to create/edit a function by using the information in the FunctionInformationPanel
jbtnCreate.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        Function functionCreated = fip.getFunctionInfo(); //Creating an instance of function depending on the information given in the FunctionInfo
        if(functionCreated!=null) {
            boolean creationBool = true;    //Boolean used to verify if the function already exists.
            for(int i = 1; i<4;i++) {       //Iterating on the local functions (at index 0 are the script functions)
                Vector<Function> functionList = functionData.get(i);

                //Foreach function we check if there is not an already function with the same name
                for(int j=0;j<functionList.size();j++){
                    Function f = functionList.get(j);

                    if(f.getName().equals(functionCreated.getName())){      //If the function already exists
                        functionList.set(j, functionCreated);
                        creationBool=false;
                        break;
                    }
                }
                if(!creationBool) { //if we already found a function with the same name, we can stop the iteration
                    break;
                }
            }

            if(creationBool){
                //If the function is a scenario we create a button
                if(functionCreated.getType().equals("Scenario")){
                    createScenarioButton(functionCreated);
                }

                int indexFunctionType = FunctionType.valueOf(functionCreated.getType()).ordinal();  //Index depending on the type of the function
                functionData.get(indexFunctionType).add(functionCreated);                           //Adding the created function to the functionDa
                if(fhp.jcbFunctionType.getSelectedIndex()==indexFunctionType){                      //If the current index corresponds to the index
                    fhp.jlFunction.setListData(functionData.get(indexFunctionType));                //the function list is refreshed
                }
            }

            aladin.command.addFunction(functionCreated);    //Adding the function to Aladin's local functions
            try {
                aladin.configuration.saveLocalFunction();   //Calling Aladin's saving function so the new created/edited function is saved in the b
            } catch (Exception err) {
                System.out.println("[Exception>FunctionInfoPanel>jbtnCreate]: while trying to save the function the following error has been encour
            }
        }
    }
});
```

Annexe 12 – Extrait de code pour la création d'un scénario

```
Command.java
l.150 & l.162 : Rajout du mot clé public à CMD et NOSYNCCMD pour y accéder à partir du plugin
l.1821 : Changement de privée à public pour la méthode getHelpString

l.2914 : Ajout attribut type, pour pouvoir gérer si c'est un scénario ou une brique
l.2942 : Ajout de code similaire à celui pour les commentaires, pour gérer le type de la fonction
l.3016 : Ajout de code similaire à celui pour les commentaires, pour gérer le type de la fonction
l.3068 : Ajout de code similaire à celui pour les commentaires, pour gérer le type de la fonction
PS:Le fait de gérer le commentaire ici induit que le cas traitant le commentaire dans le parseur ne sera jamais utilisé (dans la classe Function).
J'ai donc ajouté la gestion du type aux deux endroits.

l.4337 : Modification de l'accès en public et ajout de méthode pour accéder aux différentes listes de fonctions
l.4345 : Ajout de mode pour accéder à d'autres listes de fonctions
_____
Function.java
l.38 : Ajout de l'attribut type pour pouvoir faire une différenciation entre une fonction scénario et une fonction brique.
l.57 : Ajout d'un constructeur de Function pouvant prendre un type en entrée
l.79 : Ajout d'un get pour le type
l.82 : Ajout d'une méthode getHelp pour pouvoir l'afficher dans l'éditeur de fonction.
l.93 : Ajout d'un set pour le type
l.133 : Modification de la fonction toString pour pouvoir sauvegarder le type. Celui-ci est précédé par un !
l.207 : Ajout à la méthode parseFunction() de la gestion du type lorsqu'il y en a un.
PS: Comme dis précédemment, ce cas ainsi que celui du commentaire ne seront jamais utilisés car cela est géré dans la classe Command.java
_____
Configuration.java
l.1792 Modification de la méthode saveLocalFunction de protected à public pour sauvegarder les ajouts et suppressions de l'interface de contrôle
directement.
_____
FrameFullScreen.java
l.378: Modification du commentaire en code pour permettre à la vue cinéma d'apparaître sur le dexuième écran en mode cinéma.
```

Annexe 13 – Liste des modifications apportées à Aladin