**Projet Génie Logiciel**

**ENSC 2015-2016**

*Documentation technique*

Contenu

[I. Gestion du projet 3](#_Toc438141977)

[a. Contexte et méthode 3](#_Toc438141978)

[b. Planning 3](#_Toc438141979)

[II. UML 4](#_Toc438141980)

[a. Structure 4](#_Toc438141981)

[b. Justifications 5](#_Toc438141982)

[III. Description du programme 6](#_Toc438141983)

[a. Définition des classes 6](#_Toc438141984)

[b. Fiches Form 6](#_Toc438141985)

[1. Fiche Mission 6](#_Toc438141986)

[2. Fiche SelectAstroDelete 8](#_Toc438141987)

[3. Fiche TaskView 9](#_Toc438141988)

[4. Fiche TaskForm 9](#_Toc438141989)

[5. Fiche DayReport 10](#_Toc438141990)

[6. Fiche MissionMap 11](#_Toc438141991)

[7. Fiche Recherche 11](#_Toc438141992)

[8. Fiche LoadMission 12](#_Toc438141993)

[IV. Tests 13](#_Toc438141994)

[a. Tests unitaires 13](#_Toc438141995)

[b. Tests fonctionnels 13](#_Toc438141996)

[c. Résultats 14](#_Toc438141997)

[V. Résultats 16](#_Toc438141998)

[a. Bilan 16](#_Toc438141999)

[b. Evolutivité 16](#_Toc438142000)

# Gestion du projet

## Contexte et méthode

Ce projet s’inscrit dans l’Unité d’Enseignement de Génie Logiciel, en deuxième année de l’Ecole Nationale Supérieure de Cognitique. L’objectif de ce projet est de réaliser une application sous Windows Form permettant la gestion d’une base martienne abritant des astronautes.

Cette application doit pouvoir être utilisée par les astronautes sur place et par le centre de contrôle sur Terre. Elle doit permettre, entre autre, la gestion des astronautes, de leur emploi du temps et de leurs tâches. Elle doit également être facile et conviviale d’utilisation.

Nous nous sommes servis de GitHub pour travailler ensemble sur ce projet. GitHub est un outil de gestion de développement, qui permet d’accéder au dossier du projet partout et de pouvoir modifier les documents et le code sans conflit. Cela a facilité la répartition des tâches car l’un de nous pouvait se permettre de travailler sur une partie du projet sans modifier ce que l’autre a fait jusque-là.

Nous avons décidé de faire une interface anglaise. Le projet étant dans un contexte international, cela nous semblait pertinent de faire une interface utilisable par le plus grand monde. Néanmoins, pour nous simplifier la tâche et faciliter la communication, le reste du projet est en français.

## Planning

Ce projet a commencé le 7 octobre 2015 et doit être terminé pour le 18 décembre. Le planning détaillé est donné ci-dessous :

|  |  |  |  |
| --- | --- | --- | --- |
| Tâches | Dates | Description | Qui |
| Définition du sujet | 7 – 14 octobre | Définir ce qui est demandé, quelles sont les fonctionnalités attendues et les limites du sujet. Organisation des tâches, définition d’un premier planning. | Jérémy & Florian |
| Définition des classes UML et de l’architecture | 14 octobre – 2 novembre | Etablir l’architecture du projet et des données. Définir l’organisation du fichier XML contenant les données.  Définir l’architecture de l’application. | Jérémy & Florian |
| Rédaction du code | 2 novembre – 13 décembre | Phase de développement de l’interface. Mise en place des fonctionnalités spécifiées et prise en compte des directives formulées lors des définitions et dans le cahier des charges. | Jérémy & Florian |
| Elaboration des tests fonctionnels | 6 – 10 décembre | Création de scénarios visant à tester les fonctionnalités principales et / ou critiques. | Jérémy & Florian |
| Passation des tests fonctionnels | 12– 16 décembre | Passation des tests par plusieurs utilisateurs sélectionnés au préalable. Enregistrement des résultats. | Jérémy & Florian |
| Rédaction du rapport | 6 – 16 décembre | Rédaction de la documentation technique telle que demandée dans le cahier des charges du projet. Reprise des fichiers modifiés pour ajouter des commentaires. | Jérémy & Florian |
| Relecture & Recettage | 16 – 18 décembre | Vérifications pour nettoyer le code et qu’il n’y ait pas de fautes. Vérifier que nos choix et que nos formulations soient cohérentes. | Jérémy & Florian |

# UML

## Structure

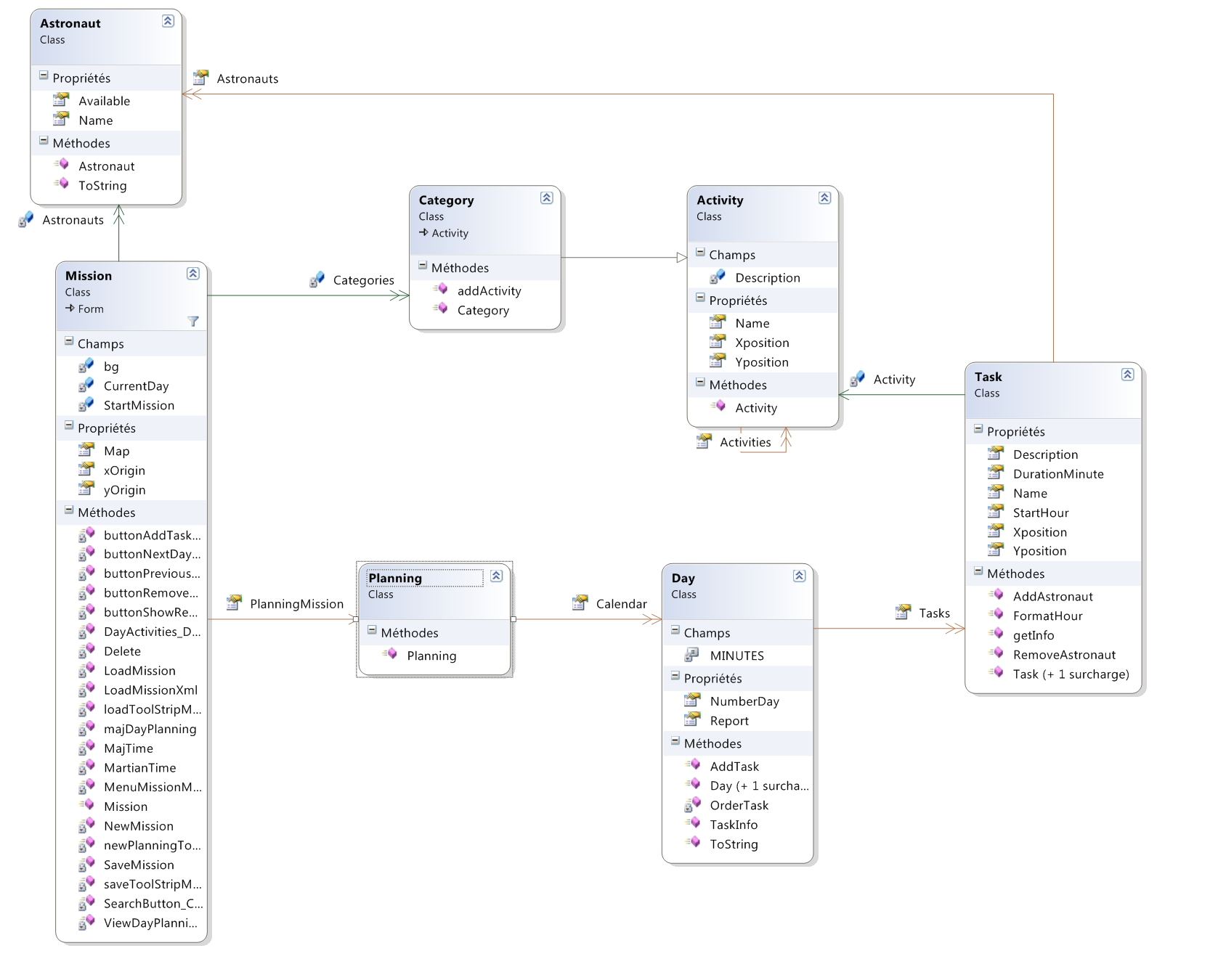


Figure 1 : Structure de l'UML

## Justifications

L’UML de notre projet est relativement simple. La classe centrale est la classe *Task* qui correspond à une tâche, à un moment donné. Ce que les astronautes font correspondent à des tâches, et ce qui remplit leur planning. C’est pour cela que c’est, de loin, la classe avec le plus d’attributs et de méthodes. Une *Task* correspond à l’*Activity* que doivent réaliser un ou plusieurs *Astronaut(s)* un certain *Day.* La définition de la *Task* justifie donc les relations qu’elle a avec les autres classes. Une *Task* est définit par son commencement (*StartHour,* qui correspond à des minutes) et sa durée (*DurationMinute*). Elle a un nom, une description (telle que décrite dans le cahier des charges) et une localisation en X et Y pour la repérer sur la carte.

Une *Activity* correspond aux différentes activités telles que définies dans le cahier des charges du projet. Cela inclut par exemple « Repair », « Medical Act » ou « Private ». Comme nous avons une relation de hiérarchie entre certaines activités (comme Science 🡪 Exploration 🡪 Vehicule), nous avons une relation d’association entre *Activity* et elle-même (formalisée par l’attribut *Activities*). Une *Category* est utilisée pour la gestion de la génération des activités à partir du XML.

Un *Day* correspond aux jours, tels qu’affichés sur le niveau 1 de l’interface. Chaque jour a donc un numéro précis, *NumberDay.* A chaque jour est associé un rapport, *Report,* tel que décrit dans le cahier des charges. L’attribut *MINUTES* correspond à la durée en minutes du jour Martien, ce qui est essentiel pour d’autres méthodes dans le programme. Ces *Day* sont listés dans la classe *Planning,* qui consiste simplement en la liste des jours de la mission. L’intérêt est que c’est cet objet qui sera généré dans l’interface, au lieu d’une liste brute de *Day*.

Un *Astronaut* correspond à un astronaute. Sa propriété « Available » est utilisée dans la fiche *DayReport* pour afficher le bon nombre d’astronautes concernés par la tâche.

Nous n’afficherons pas les fiches dans l’UML de ce programme, mais nous affichons la classe *Mission* car c’est par cette classe que l’on charge le fichier XML du projet. On y créé et définit le planning de la mission, avec les tâches telles que définies dans le XML. Cette fiche pose également le début temporel de la mission, et on y charge la carte où on pourra placer les activités en extérieur.

Vis-à-vis de nos premiers diagrammes UML, on a surtout largement simplifié la gestion des activités, notamment les activités en extérieur. Elles formaient à la base une classe à part entière mais ont été ramené avec les autres activités. Sinon, il y a eu peu d’évolutions de l’UML au cours du projet.

# Description du programme

## Définition des classes

Notre programme est composé de 9 classes qui sont des Forms permettant d’interagir avec l’utilisateur et de 7 classes afin de pouvoir modéliser notre solution.

## Fiches Form

### Fiche Mission

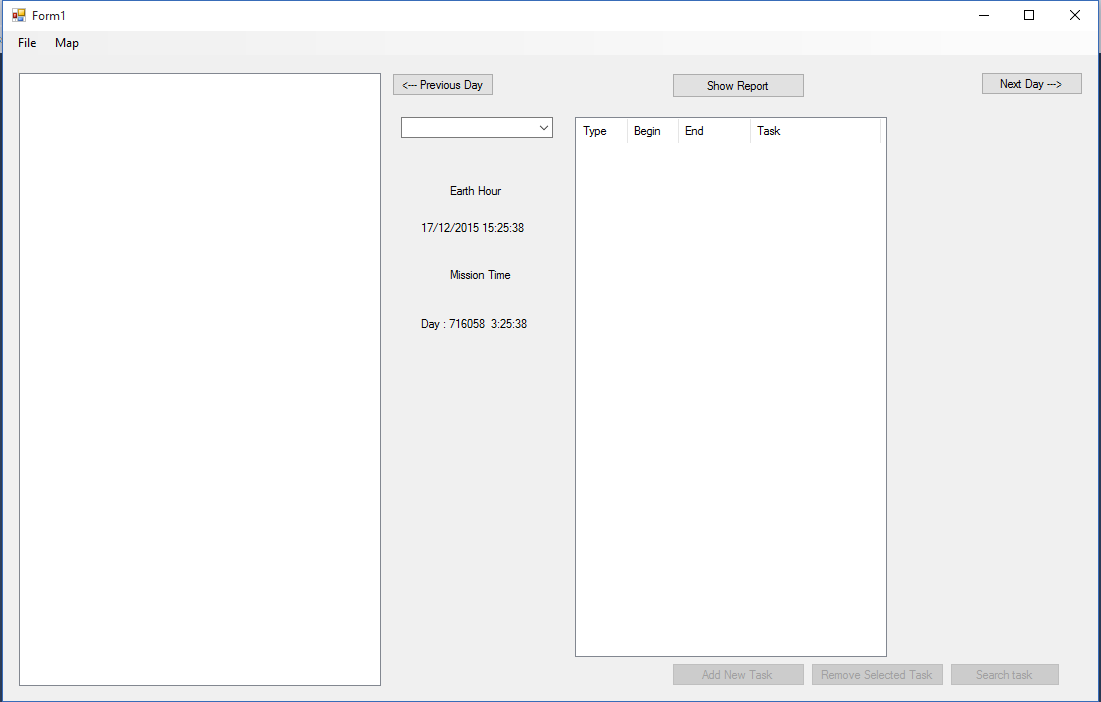


Figure 2: Fiche Mission, sans planning

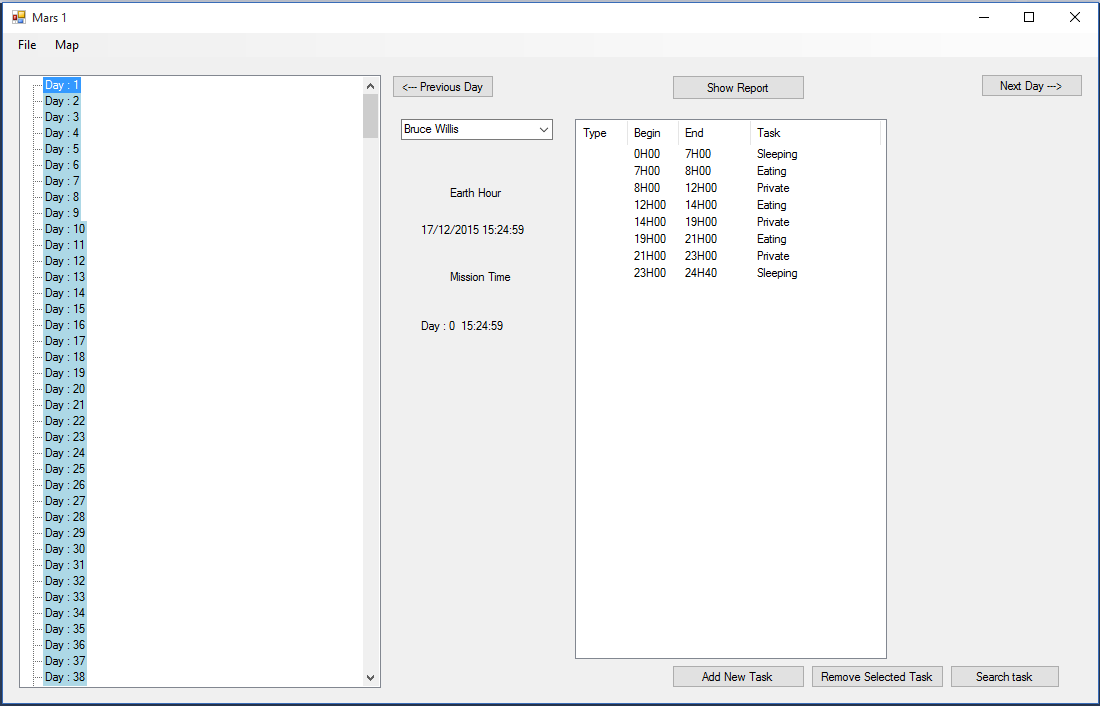


Figure 3: Fiche Mission, avec planning

Ceci est la principale fiche du programme. Elle permet l’affichage des deux premiers niveaux d’interfaces tels que décrits dans le cahier des charges. Cet affichage se fait via deux *SplitContainer,* chacun contenant un panel (et un niveau).

A gauche, il y a le niveau 1 de l’interface, c’est-à-dire l’affichage des jours. Cet affichage est fait par une *TreeView* (nommée ListCalendar), où chaque item – chaque jour donc – est créé lors du chargement de l’XML de mission. Les jours déjà passés sont affichés en gris, le jour actuel est en vert et les jours futurs sont en bleu. Lorsqu’un jour est sélectionné, le niveau 2 de l’interface correspond à ce jour est affiché sur le panel de droite. Ce panel est constitué des boutons de navigation usuels mais surtout d’une *ListView* affichant les différentes tâches de la journée. Y sont affichées :

* Le type d’activité, c’est-à-dire si l’activité est en extérieur (*Outside Activity*) ou non.
* Son début et sa fin
* La tâche en question

Double-cliquer sur une tâche affiche la fiche *TaskView*. Cliquer sur le bouton Add New Task affiche la fiche *TaskForm.* Cliquer sur le bouton Remove Selected Task affiche la fiche *SelectAstroDelete*. Cliquer sur le bouton Show Report affiche la fiche *DayReport* associée au jour sélectionné.

### Fiche SelectAstroDelete

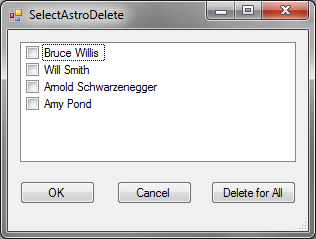


Figure 4 : Fiche SelectAstroDelete

Cette fiche s’ouvre sur confirmation de la suppression d’une tâche depuis le bouton « Delete Selected Task » depuis la fiche *Mission*. Elle permet de gérer la suppression d’une tâche dans l’emploi du temps des astronautes. On peut ainsi supprimer cette tâche pour tous les astronautes avec le bouton Delete for All, ou bien choisir un ou des astronautes manuellement.

### Fiche TaskView

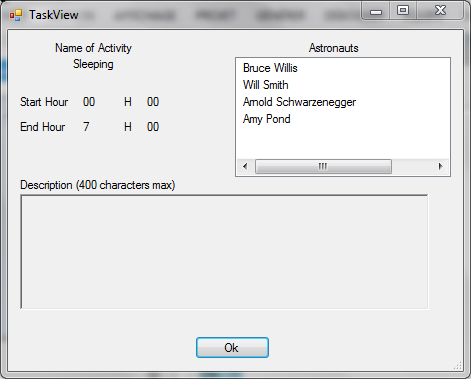


Figure 5 : Fiche TaskView

Cette fiche s’affiche en double cliquant sur une tâche dans le niveau 2 de l’interface. Elle répertorie les informations sur cette activité, sans possibilité de les modifier.

### Fiche TaskForm

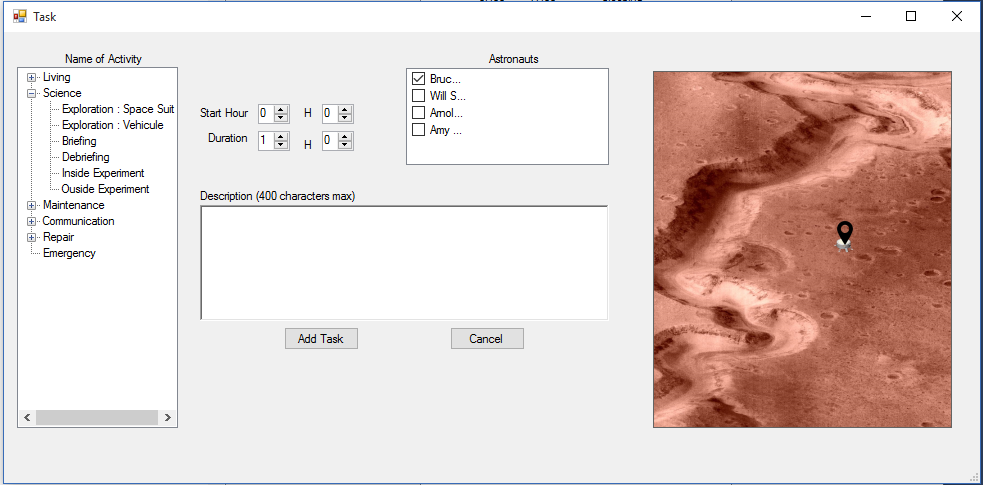


Figure 6: Fiche TaskForm

Cette fiche permet l’ajout d’une tâche et s’ouvre lorsque l’utilisateur clique sur « Add Task ». L’utilisateur peut choisir les différentes options et paramètres de la tâche. Il peut modifier la localisation de la tâche en cliquant sur la carte à droite (par défaut, la tâche se passera dans la base). De plus, le programme vérifie que les astronautes qu’il assigne à cette tâche sont bien disponibles.

### Fiche DayReport

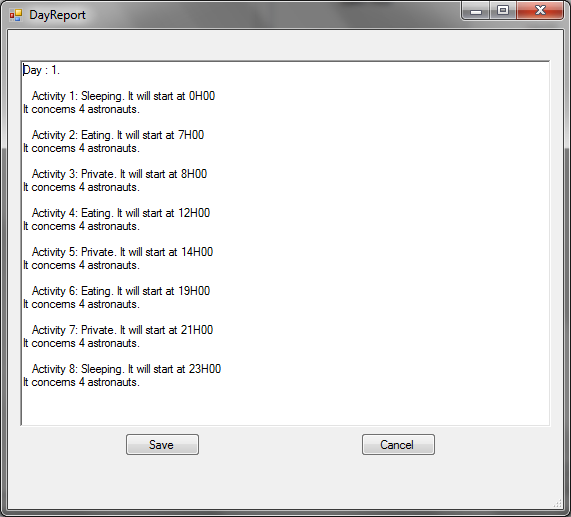


Figure 7 : Fiche DayReport

Cette fiche s’ouvre lorsque l’utilisateur clique sur le bouton « Day Report » dans la fiche *Mission.* Elle correspond à l’attribut « Report » de la classe *Day*. Sur lancement de cette fiche (et à chaque fois que ce rapport est vidé par l’utilisateur), un rapport de base est chargé, tel que montré ci-dessous. L’utilisateur peut le modifier, et ces changements seront sauvegardés.

### Fiche MissionMap

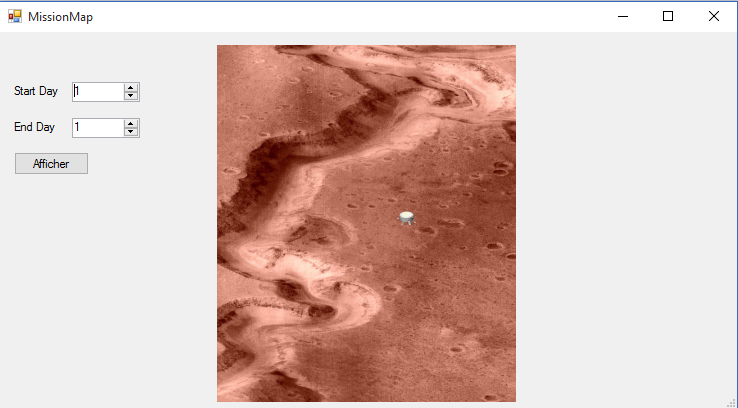


Figure 8: Fiche MissionMap

Cette fiche s’ouvre lorsque l’utilisateur clique « Show Map Mission » et permet l’affichage du lieu des missions se passant à l’extérieur de la base. Les seules missions apparaissant sur cette fiche sont des missions ne se passant pas dans la base. On peut modifier l’intervalle de date dans lequel on souhaite afficher les missions.

### Fiche Recherche

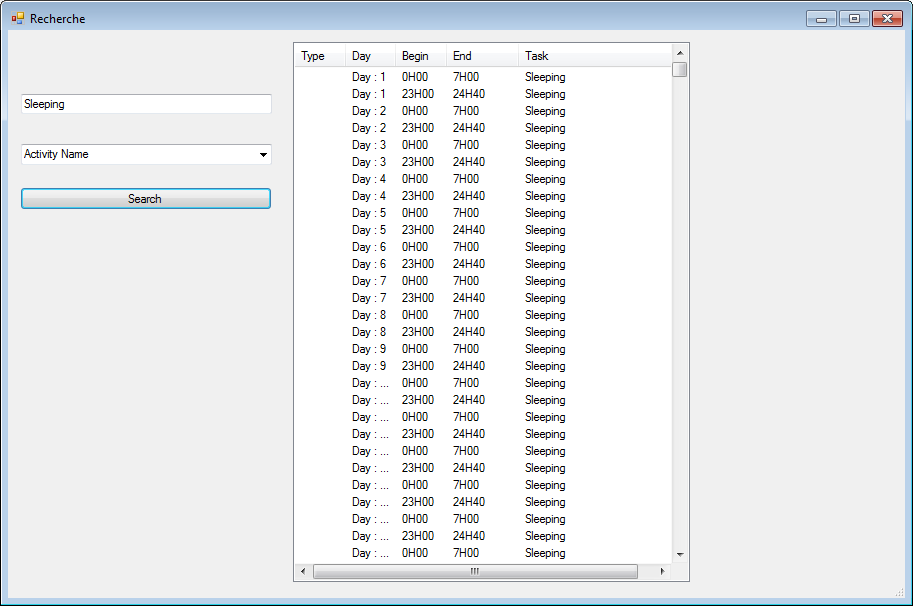


Figure 9: Fiche Recherche

Cette fiche s’ouvre lorsque l’utilisateur clique sur « Search Task ». Elle permet la recherche d’une tâche sur l’ensemble de la période de la mission (pas possible d’affiner la recherche). Les informations affichées sont les mêmes que les informations affichées dans le niveau 2 de l’interface : double-cliquer sur une tâche permet donc aussi l’affichage des informations sur cette tâche dans une fiche *TaskView*.

### Fiche LoadMission



Figure 10 : Fiche LoadMission

# Tests

## Tests unitaires

Nos méthodes ayant général une sortie de type *void*, nous n’avons pu créer des tests unitaires que sur certaines fonctions. Ces fonctions sont (avec le type de la sortie) :

* Astronaut.ToString() : string
* Day.ToString() : string
* Day.TaskInfo(**Task** task) : string[]
* Task.GetInfo[] : string
* Task.FormatHour(**int** Time) : string
* Mission.MartianTime(**double** SecondsBegin, **out** **int** nbDay) : string
* FormatHour(**int** Time) : string

Les tests unitaires ont été générés grâce aux outils de Visual Studio. Ils ont permis de vérifier que le fonctionnement de ces fonctions est conforme à ce que nous attendions.

## Tests fonctionnels

#### Test 1 : Gestion de tâches courantes

Ce qui va être testé :

* Lecture du compte-rendu
* Edition du compte-rendu
* Suppression et ajout d’une tâche
* Gestion des horaires
* Limite de caractères sur le descriptif
* Edition d’une tâche
* Sauvegarde

L’utilisateur lance l’application avec le fichier de base pré-installé. Il se rendra au jour en cours. Il ouvrira le compte-rendu pour ce jour. Il l’éditera en ajoutant : *« Communications avec la terre, compte-rendu oral* ». Il validera et sauvegardera le changement.

Il supprimera l’activité « Private » en 21h00 et 23h00 pour tous les astronautes. Il ajoutera une activité nommée « Debriefing » entre 20h30 et 22h00 pour tous les astronautes, sur la station des astronautes et avec ce descriptif :

*“Simple debriefing with the control center of planet Earth. Regular check of various functions, of the vehicles and of the experiments led during the mission. In case of emergencies, contact the control center as soon as possible.*

*Every member has to be present for this task, so that control center can send news to the astronauts' families. Not all of them have to be present for the whole task though, but we ask for some presence.”*

En cas d’erreur sur l’horaire, il ajoutera cette activité entre 21h00 et 22h30. En cas d’erreur sur la description, il ajoutera cette activité avec le descriptif suivant :

*“Simple debriefing with the control center of planet Earth.”*

Une fois l’activité validée, il sera ramené sur le menu principal. L’utilisateur sauvegardera alors la mission en sélectionnant l’option *Save* dans le menu *File* dans le menu de l’application

#### Test 2 : Chargement Recherches

Ce qui va être testé :

* Chargement d’une mission précédemment enregistrée
* Ajout d’une mission en extérieur
* Recherche sur la carte une période donnée
* Recherche sur les jours pour un mot-clé donné sur une période donnée

L’utilisateur lance l’application et charge la mission précédemment enregistrée en sélectionnant le fichier décrivant la mission et le fichier sauvegardé. Une fois que cela est effectué, l’utilisateur vérifiera que les modifications apportées à l’emploi du temps de base ont bien été prises en compte dans le chargement.

Une fois sur le jour en cours, il supprimera l’activité *Private* entre 14h00 et XXhXX pour les deux Astronautes Amy Pond et Bruce Willis. Ensuite il ajoutera sur cette plage horaire une activité *Outside Experiment* en sélectionnant ces deux Astronautes. Avant de valider, il double-cliquera sur la carte à droite et indiquera un lieu autre que la base de la mission. Une fois cette étape effectuée, l’utilisateur pourra valider l’ajout de la tâche.

Ensuite, l’utilisateur cliquera sur **Map Button**  et sur la nouvelle fenêtre, il sélectionnera une plage de jours comprenant celui où a été vient d’être ajoutée l’activité *Outside Experiment*. En cliquant sur le bouton ***Afficher,*** l’utilisateur verra alors s’afficher un (ou plusieurs) marqueur indiquant la position des missions en extérieurs. En cliquant sur un de ces marqueurs, l’utilisateur verra apparaître une fenêtre donnant les détails de la tâche.

Enfin, l’utilisateur fermera la fenêtre de la carte. Il cliquera ensuite sur le bouton *Recherche* ce qui ouvrira l’interface de Recherche. En écrivant le nom d’une tâche (*Outside Experiment* par exemple) ce qui affichera les tâche portant ce nom, classées par ordre chronologique.

#### Test 3 : Erreur à l’ouverture

Ce qui va être testé :

* Gestion des erreurs en cas d’adresse image invalide

L’utilisateur lance l’application et charge un fichier de mission avec une mauvaise adresse pour l’image. Il vérifie qu’aucune tâche n’apparaisse sur le niveau 2 (droite de l’écran), et que le bouton « Show Map Mission » fonctionne mais affiche une fiche sans map.

## Résultats

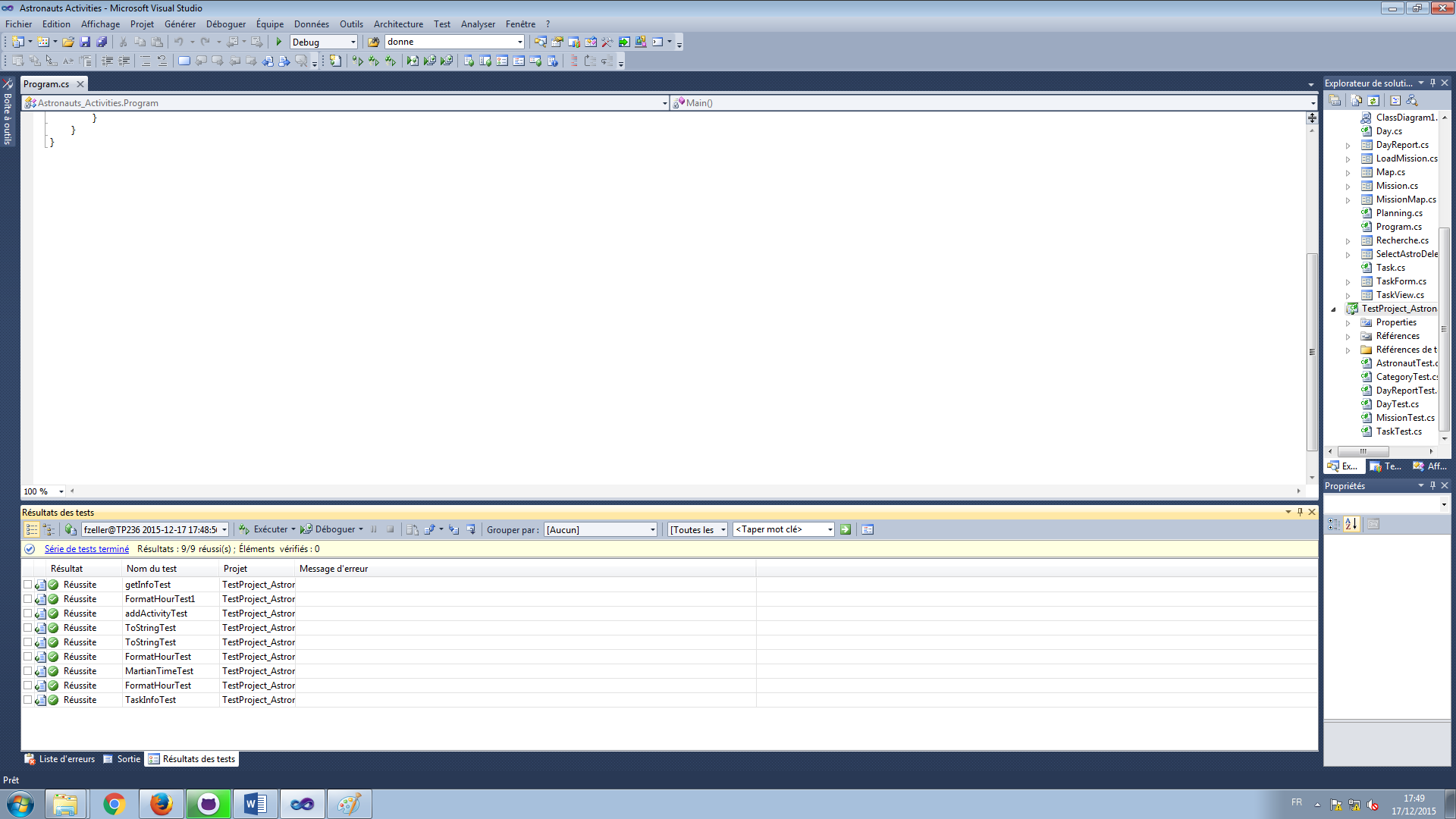


Figure 11 : Rapport des tests unitaires

Les tests fonctionnels quant à eux sont passés avec succès. Ces derniers nous ont bien permis de tester l’ensemble des fonctionnalités de notre application en fournissant les résultats attendus.

# Résultats

## Bilan

Les trois principales fonctionnalités qui manquent à notre application sont la possibilité d’affiner les recherches de tâches sur une période, la possibilité de modifier une tâche et surtout de conserver correctement ces modifications, et l’affichage des activités sur la carte en différentes couleurs, selon que l’étape soit passée ou à venir.

[TODO : Erreurs ?]

La principale originalité de notre application est de regrouper sur une seule fenêtre les niveaux 1 et 2 d’interface. Les utilisateurs ont un seul écran devant eux qui regroupe un maximum d’information tout en restant lisible et facilement à comprendre pour l’utilisateur. Chaque fonctionnalité supplémentaire est ouverte sur une autre fenêtre, pour marquer la différenciation entre ces fonctions et le niveau principal de l’application. Le design en lui-même est sobre, et ne surcharge pas l’utilisateur. Nous nous sommes efforcés de rendre un maximum d’interactions intuitives et de minimiser la profondeur du logiciel. La majorité de l’application est accessible en trois clics au maximum.

Nos tests fonctionnels ont aussi étaient importants pour gérer des erreurs mineurs, mais qui peuvent se révéler agaçantes pour l’utilisateur.

## Evolutivité

Le projet peut être amené à évoluer de nombreuses façons. Tout d’abord, l’ajout d’une fonctionnalité d’édition des tâches est une des choses que l’on pourrait ajouter.

Ensuite la possibilité d’importer des structures missions personnalisée. En effet, actuellement la structure des tâches d’une mission est fixe et ne permet pas l’ajout ou le retrait d’activité. Dans notre code, le chargement prend en compte n’importe quelle architecture d’activité dans le fichier XML de base de mission. Cependant lors de la génération du planning de base, les tâches qui y sont ajoutées sont fixes et ne dépendent pas du fichier XML fourni par l’utilisateur. Il faudrait donc pouvoir rendre le programme plus générique.

En marge de cette option, rajouter une interface afin de pouvoir permettre à l’utilisateur de créer cette architecture d’activités semble être une poste d’évolution pour notre programme.