**Projet informatique**

**Michel Donnet**

1. **Introduction**
   1. **Présentation du projet**

Mon projet dans ce cours d’application informatique consiste à aider au développement d’une application déjà existante utilisée par des chercheurs en neurosciences. Cette application consiste en une caméra se déplaçant à une certaine vitesse dans un environnement 3D, actuellement un chemin dans une forêt. Des drapeaux sont placés de part et d’autre du chemin, et les chercheurs demandent au cobaye, plongé dans cet environnement au moyen de la réalité virtuelle, de trouver les drapeaux respectant certains critères. L’application enregistre où le client a regardé, et les chercheurs exploitent ces données ainsi que les signaux électriques émis par le cerveau lors de l’expérience afin d’étudier le mode de fonctionnement du cerveau dans un environnement 3D, ce qui est plus proche de la réalité que d’étudier son fonctionnement au moyen de tests en 2D, ce qui était fait avant l’arrivée de la réalité virtuelle.

L’application sur laquelle j’ai travaillé permet donc d’améliorer l’étude du cerveau grâce à un environnement 3D. Question de la licence à régler…

* 1. **Client et utilisateurs**

Mon client était une équipe de chercheurs en neurosciences, et les utilisateurs de l’application sur laquelle je devais travailler seront des chercheurs en neurosciences n’ayant pas forcément de grandes compétences en informatique. Mon travail sur l’application devait donc être facile d’utilisation.

* 1. **Besoins identifiés**

Tout d’abord, mon client m’a demandé de faire des drapeaux animés pour un soucis de réalisme. En voyant les résultats obtenus, j’ai proposé de créer plutôt des bannières que des drapeaux car celles-ci permettaient de mieux voir le motif plutôt que le drapeau. Cette proposition a été acceptée par le client après réflexion.

J’avais comme tache suivante de mettre des flashs sur les drapeaux pour attirer l’attention du cobaye sur l’un ou l’autre drapeau, puis enfin d’ajouter des distracteurs afin de déconcentrer le cobaye. Mais ces deux tâches m’ont été retirés plus tard lorsque je reçus de nouvelles tâches plus importantes.

Concernant les drapeaux, on me demanda également d’ajouter deux nouveaux types de drapeaux. Ce qui fut rapidement fait.

Puis après deux semaines environ, comme ils ne trouvaient pas pratique le menu permettant de contrôler les paramètres de l’application, on m’a demandé de refaire l’interface graphique, avec comme besoin principal de supprimer tous les sliders. Il fallait évidement que le nouveau menu permette de faire les mêmes choses que l’ancien menu.

Au fur et à mesure que je progressais, on me demandait de corriger des détails. Par exemple, j’utilisais une liste à scroller pour la configuration des blocks, et on m’a demandé de faire plutôt des pages, car le menu à scroller n’était pas pratique.

On m’a également demandé de modifier la façon d’enregistrer la configuration créée via le menu pour pouvoir relancer l’expérience avec la même configuration : en effet, dans le menu initial, la configuration était sauvegardée en 2 fichiers. Maintenant, la configuration est sauvée dans un unique fichier, et quand on charge une configuration, comme les positions et rotations de chaque drapeau sont enregistrés dans le fichier de configuration, on se retrouve avec exactement la même disposition de notre scène que lorsqu’on avait enregistré notre configuration.

1. **Problèmes rencontrés**

Lors du premier mois, le principal problème rencontré était que le projet était sur Unity3D et en C#, et qu’un code existait déjà : je devais me greffer sur le code existant. Ce fut très difficile au début, et pour surmonter les différents problèmes, je me suis documenté sur Unity3D sur internet et pour C#, j’ai lu et relu les scripts existant en espérant comprendre ce que ceux-ci faisaient, sachant que ces scripts n’étaient presque pas documentés…

Puis, je me suis retrouvé confronté à un autre problème : une de mes tâches était de créer des drapeaux animés pour plus de réalisme dans l’application. Mon idée première était de créer les drapeaux animés sur un logiciel de 3D, Blender, mais je me suis rendu compte que je ne pouvais pas importer de la physique d’objets dans Unity3D… J’ai donc dû créer directement les drapeaux sur Unity3D.

Ensuite, on m’a demandé de refaire la GUI. Comme je n’ai jamais fait de GUI, j’ai tout découvert sur le tas. J’ai donc passé du temps à lire la documentation Unity et à regarder des exemples d’implémentation de GUI.

Puis, je me suis retrouvé confronté à un problème : rendre une application propre pour les prochains développeurs. Comme il y avait des scripts qui traînaient partout, une réorganisation générale et une réécriture des scripts a réglé le problème. Cependant, j’aurais dû ne pas passer d’un extrême à l’autre et tout réunir dans deux scripts, car le client aurait préféré avoir plusieurs scripts de taille moyenne plutôt que deux gros scripts.

Pour faire cette réorganisation générale, un autre problème est venu à moi : j’avais besoin de trouver certains gameObjects de ma scène et pour cela, une fonction unity permet de trouver des gameObjects activés… Mais presque tous mes gameObjects étaient désactivés (pour ne pas apparaître dans l’interface). J’ai donc trouvé une fonction me permettant de charger tous mes gameObjects, puis j’ai fait un tri sur ces gameObjects obtenus (j’en avais 2000 environ, alors que seulement une centaine était utile…). Par la suite, j’ai appris que cela aurait été possible et préférable de tout remplir à la main afin d’éviter la recherche exhaustive de tous les gameObjects…

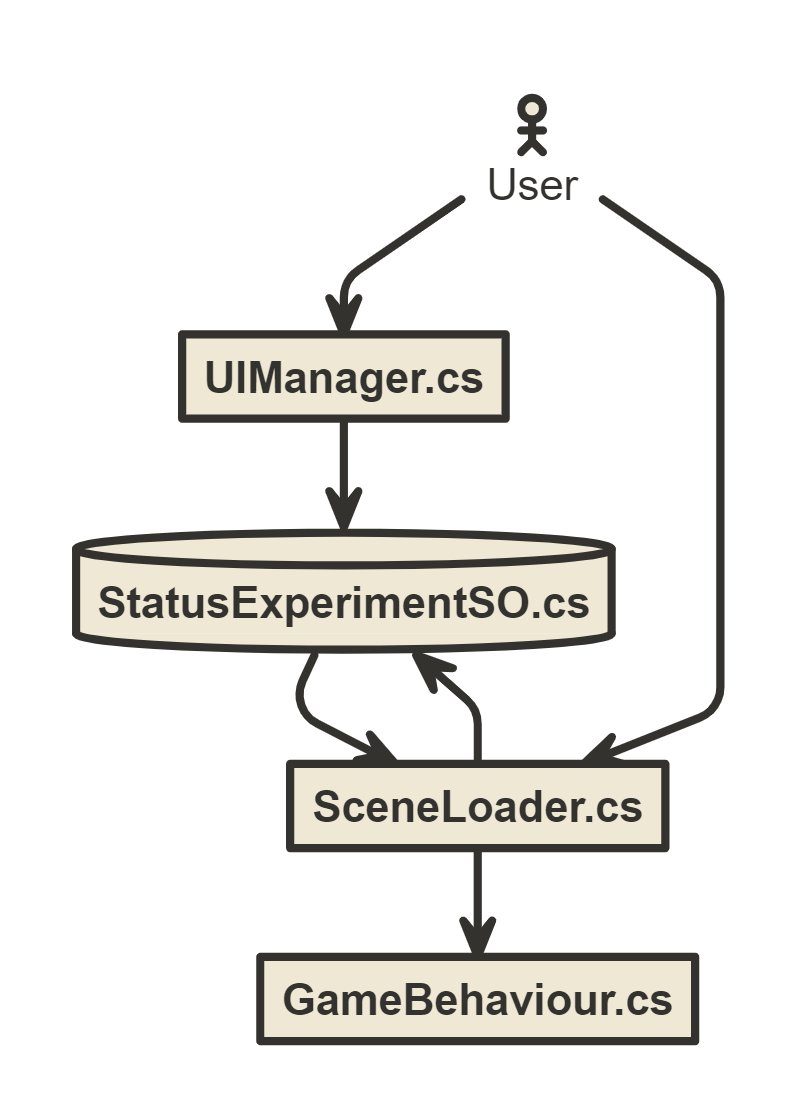
1. **Développement**
   1. **Conception**

Le nouveau menu devait ne plus avoir de sliders, car ce n’était pas pratique. De plus, dans le nouveau menu, on a une configuration de blocks à faire, sachant que chaque block peut être configuré différemment (pour chaque block, on peut changer le nombre de drapeaux, l’emplacement de ceux-ci etc…) Au début, j’avais trouvé judicieux de créer une classe block possédant tous les champs qu’on peut modifier dans un block et de travailler avec une liste de blocks de cette classe. Mais, cela impliquait le fait d’attribuer à chaque gameObject un scriptable object blockclass avec un nom différent. Finalement, j’enregistre toutes les informations des blocks dans des listes bidimensionnelles… Cela est moins clair, mais plus pratique car pour lancer l’expérience, j’ai besoin de listes bidimensionnelles et non de classes blocks car je réutilise la fonction implémentée par le programmeur qui a créé l’application. Je n’ai donc finalement pas utilisé de structures de données ou de classes, mis à part une classe StatusExperiment.cs qui permet de stocker toutes les informations, et toutes ces informations peuvent ainsi être utilisés par d’autres scripts. Sinon, ce sont essentiellement des fonctions appelées par les différents boutons qui se retrouvent dans les scripts créés.

* 1. **Implémentation**

Le schéma de mon code est donc très simple, car je n’ai pas eu besoin de créer beaucoup de classes. Il est donné par le schéma ci-dessous…

Explication du *schéma des relations entre les différents scripts créés* : le script StatusExperimentSO.cs sert de stockage des informations. C’est un scriptable object, il ne possède pas de méthodes, mais seulement des champs à remplir. Notre utilisateur va interagir avec le menu, ce qui va modifier les informations du StatusExperimentSO.cs grâce à l’UIManager.cs et le SceneLoader.cs qui sont composés des fonctions appelées par les différents boutons de l’interface, et qui modifient en général les informations du StatusExperimentSO.cs (mais pas tout le temps… En effet, ma gestion de fenêtre se fait aussi dans l’UIManager, mais ne modifie pas le StatusExperimentSO.cs). Puis le SceneLoader.cs envoie toutes les informations du StatusExperimentSO.cs dans le GameBehaviour.cs lorsqu’on appuye sur le bouton start. Le GameBehaviour.cs est le ‘moteur’ de notre application et a été implémenté par le programmeur qui a créé l’application. Dans le SceneLoader.cs, on appelle des fonctions qui avaient déjà été implémentés par le programmeur dans d’autres scripts.

*Schéma des relations entre les différents scripts créés*

Comme le projet se déroulait sur Unity3D, je n’ai pas eu de choix pour les outils informatiques pour développer l’application. J’ai essayé néanmoins d’utiliser Blender, mais comme je l’ai mentionné précédemment, ce fut un échec. J’ai donc dû tout faire directement sur Unity, et programmer en C#. Unity proposait d’utiliser VSCode pour éditer les scripts… Je ne me suis pas opposé à cette proposition. De plus, le projet qui m’a été fourni était sur Gitlab. Je suis donc resté sur Gitlab pour développer ma propre version du projet, par soucis de simplicité.