**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.

Voici un exemple de ce que la personne voit en réalité virtuelle lors de l’expérience :

Une image contenant plante, plein air, arbre, ciel

Description générée automatiquement

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…

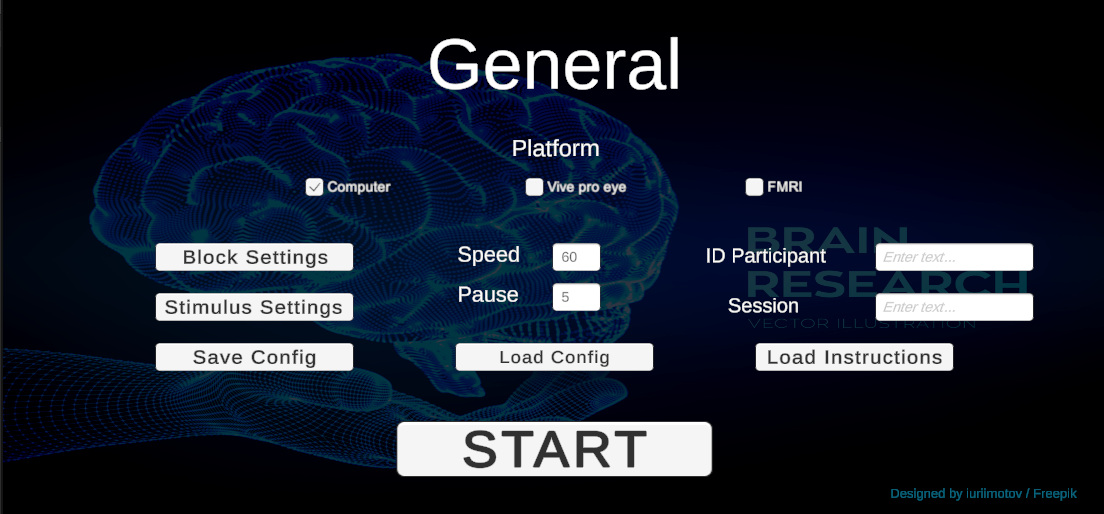
1. **Développement**
   1. **Conception**

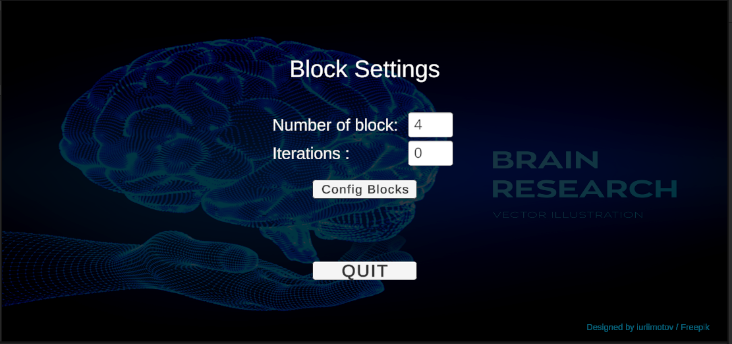
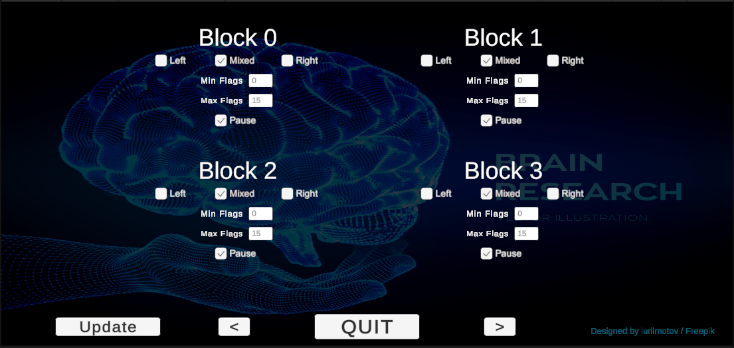
Le nouveau menu devait ne plus avoir de sliders, car ce n’était pas pratique. De plus, dans le nouveau menu, on a une configuration de blocks à faire, sachant que chaque block peut être configuré différemment (pour chaque block, on peut changer le nombre de drapeaux, l’emplacement de ceux-ci etc…) Au 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.

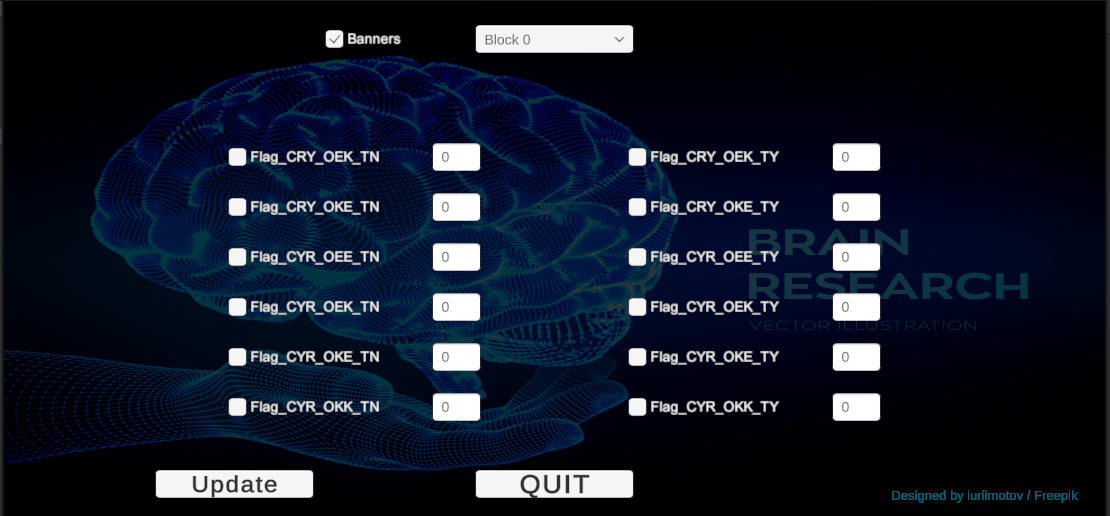
Voici l’ancienne interface qu’il y avait déjà :



Et voici l’interface que j’ai créé (elle est décomposée en plusieurs ‘panels’…) :

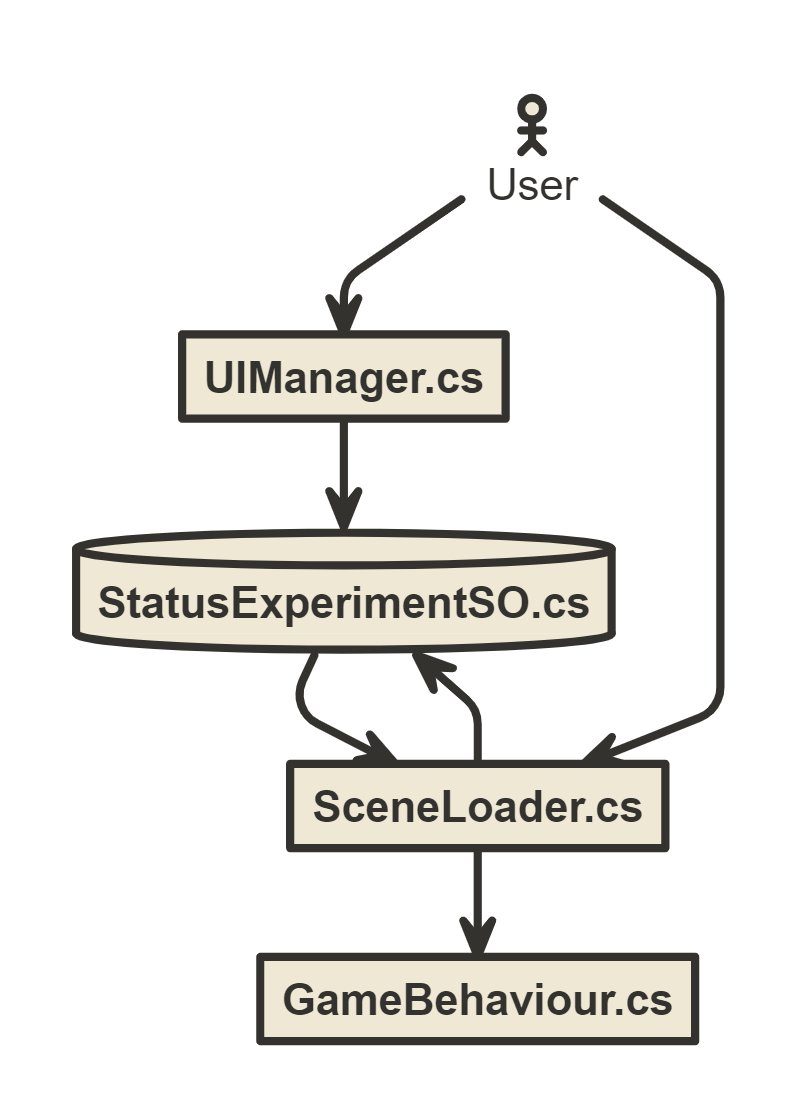


****



* 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…

*Schéma des relations entre les différents scripts créés*

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.

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.

* 1. **Tests et évaluation**

Pour tester mon programme, je n’ai pas utilisé de méthodes particulières, mis à part le fait d’essayer de faire différentes configurations et de voir si le résultat attendu était bien là… Ainsi, j’ai pu trouver quelques erreurs. Pour valider mon programme, j’ai simplement regardé à ce qu’il n’y ait plus d’erreurs qui s’affichent dans la console. Pour arriver à ce résultat, j’ai également dû débugger le code existant qui ne marchait plus avec mon code.

Les pistes pour améliorer le logiciel sont surtout de décomposer mes scripts en plusieurs scripts de taille moyenne. Il faudrait également changer le chargement de tous mes gameObjects en quelque chose d’autre, car ce n’est pas génial de charger 2000 éléments et de faire ensuite le tri sur ces éléments pour qu’il n’en reste plus qu’une centaine. Une chose qui pourrait être également bien serait de faire en sorte que tout le menu se mette à jours lors du chargement d’une configuration pour pouvoir voir un peu quelle configuration on charge …

1. **Organisation**

Le travail a été divisé assez simplement : je devais m’occuper de refaire tout le menu et des drapeaux animés, et l’équipe de développement s’occupait des autres choses, comme de faire des flashs sur les drapeaux.

Les réunions ont été faites avec le client chaque semaine le mercredi de 14h30 à 15h30… Certains jours, elles étaient plus longues que d’autres jours. J’ai trouvé que c’était une bonne idée de prendre ce rythme, comme cela j’étais forcé à travailler régulièrement pour avoir fait quelque chose avant la réunion et avoir des choses à présenter ou des questions à poser. On a fait également 3 réunions avec l’encadrant : une au lancement du projet, une intermédiaire et la dernière à la fin du projet. Et j’ai trouvé que c’était une très bonne idée de faire une réunion vers le début du projet qui m’a permis de rencontrer celle qui s’occupait du développement du programme, comme cela on a pu garder contact et elle a pu me débloquer quand j’étais bloqué dans des problèmes techniques

Le volume horaire que j’ai fourni pour le projet est supérieur à 80 heures… J’estimerais à 100 heures environ, mais après je n’ai pas tenu un compte de mes heures passées sur le projet.

1. **Formation**

Les cours principaux que j’ai utilisés pour réaliser mon projet sont essentiellement les cours de système d’exploitation et de langage orienté objet. En effet, pour programmer en C#, j’ai beaucoup utilisé mes compétences en C développés dans le cours de système d’exploitation, ainsi que les notions données dans le cours de langage orienté objet. Je me suis aussi un peu servi de la programmation en Java qu’on a fait dans ce cours… Et j’ai trouvé que le C# c’était un peu comme un mélange de Java et de C… Lorsque je programmais, je me suis également rendu compte que certaines parties de mon code étaient à optimiser grâce au cours d’algorithmique. Sinon, je n’ai pas utilisé tellement d’autres cours pour le développement de mon projet.

Les notions, principes et techniques que j’ai dû acquérir sont essentiellement d’apprendre seul un nouveau langage de programmation, et pour cela la technique de lire la documentation est très utile, mais déjà conseillé par les professeurs de l’université. J’ai appris à créer une GUI, apprendre des notions de GUI. Personnellement, je dirais que c’est tout à fait possible d’apprendre cela seul, mais ce pourrait être une idée de donner quelques notions dans un cours… J’ai également dû apprendre à maîtriser Unity3D, mais à nouveau, j’estime que ce n’est pas la peine de faire des cours dessus, même si c’est très complexe, car rare sont les personnes utilisant Unity3D.

1. **Feedback**

Personnellement, je trouve que ce cours fait un peu peur aux étudiants car c’est un peu comme un saut dans la vie professionnelle, et je trouve cela très bien. Je dirais qu’il n’y a rien à améliorer, même si ce cours est très chronophage. Mais cela dépend aussi beaucoup de notre client et j’ai eu de bons clients. Le fait de devoir travailler seul sur un projet permet de se rendre compte de comment sera la vie professionnelle plus tard et de qu’est-ce qu’on nous demandera de faire. On peut ainsi se rendre compte que le client a souvent une idée plus ou moins précise de ce qu’il veut, mais que c’est à nous de faire des suggestions qui vont être acceptées ou pas par le client.

En résumé, je trouve que ce cours s’est très bien déroulé, et je dirais qu’il n’y a rien à changer.