Rapport de stage

Equilibre Games

Bastien Giafferi - Prog M1 - 30 Juin au 2 Septembre 2016

Développement de jeux et d'applications avec Unity 5

Frédéric Rolland-Porché

Master

« jeux et médias interactifs numériques »

le cnam

Sommaire

Rapport de stage 1

Equilibre Games 1

Sommaire 2

Remerciements 3

Introduction 3

Développement 4

Numac 4

Camera 7

Mutate to Ghoul 10

Conclusion 12

Annexes 13

Annexe 1 : La « documentation » du lecteur RFID 13

Annexe 2 : La classe de base pour utiliser les Threads dans Unity 14

Annexe 3 : La documentation du GT-511C3 15

Remerciements

J’aimerais remercier Frédéric Rolland-Porché de m’avoir accueilli une nouvelle fois à Equilibre Games ainsi que pour le temps qu’il m’a accordé. J’adresse également mes remerciements à l’ensemble des employés pour leur bonne humeur.

Introduction

Lors de mon stage, j’ai été accueilli par Equilibre Games, un studio de développement installé à Angoulême depuis décembre 2012 et fondé par un ancien ingénieur de France Télécom – Orange, Frédéric Rolland-Porché.

L’entreprise est localisée à proximité du centre ville d’Angoulême, en Charente.

Equilibre Games possède le statut de SASU (Société par Actions Simplifiée Unipersonnelle).

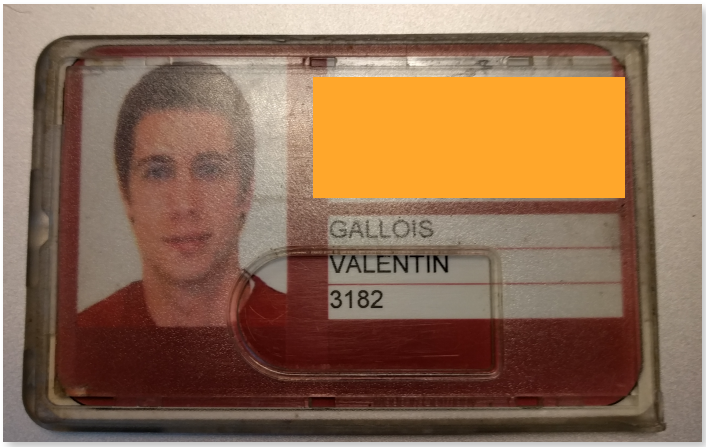
Développement

Lors de ce stage, j'ai travaillé sur de nombreux projets. Je vais maintenant décrire brièvement mes travaux.

Numac

Dans le cadre de l'application « Numac » développé par Equilibre Games pour un client important (dont je ne peux pas donner le nom malheureusement) qui a pour but de remplacer la version « physique » des outils de réunion de l'entreprise sur un écran de 65 pouces tactile, j'ai dû intégrer un lecteur RFID afin de reconnaitre les badges des employés pour s'authentifier, ainsi qu'un lecteur d'empreinte digitale. J'ai donc réalisé le système d'authentification de l'application ainsi qu'une interface de connexion avec login/mdp. Enfin, j'ai dû faire en sorte de pouvoir récupérer des mails d'une boîte mail donnée, de les sauvegarder dans la base de données de l'application afin de remplir dynamiquement la partie « Annonce » de l'application.

Le lecteur RFID



Tous les employés ont des cartes RFID (cf. image ci-contre). L'idée est de faire en sorte que chaque personne puisse se connecter directement en passant leur badge devant le lecteur. Après de nombreux tests nous avons finalement décidé d'utiliser le CSN (Card Serial Number) pour identifier un utilisateur.

On a le lecteur RFID ci-contre à notre disposition. Nous communiquons avec ce dernier via le port série (en réalité le lecteur est branché en USB mais est reconnu comme un port série). La documentation fournie (Annexe 1) est succincte, ne décrit pas l'utilité de chaque commande, ni les codes d'erreurs, ni l'intégralité des commandes disponibles…

De plus, il a fallu utiliser la classe SerialPort de .NET pour communiquer avec le lecteur. Après avoir fait une

première version fonctionnelle sur OSX, j'ai testé le programme sur Windows. Et là surprise, le programme ne marche plus du tout. J'ai finalement découvert cet article (http://www.sparxeng.com/blog/software/must-use-net-system-io-ports-serialport) qui a répondu à une partie de la question. En effet, la propriété `BytesToRead` n'est pas fiable du tout. Pire, certaines fonctions internes à la classe SerialPort semblent déclencher des NullReferenceException et même des NotImplementedException !

Finalement, la seule solution viable que l'on a trouvé pour la lecture des données est :

```
int byteRead = 0;
while((byteRead += serial.Read(ans, byteRead, 1)) < length);
```

avec :

- byte[] ans = le buffer de lecture
- int byteRead = le nombre d'octets lus
- int length = le nombre d'octets à lire (on connait la taille de la réponse)

Cependant, l'opération de lecture est une opération (très) lente, il a fallu faire du multithreading pour ne pas bloquer l'application (cf. Annexe 2 pour le code d'un thread qui fonctionne avec Unity).

Le but du lecteur de badge est qu'à n'importe quel moment dans l'application, l'utilisateur puisse s'authentifier via ce moyen. Pour cela, nous avons donc du écrire un webservice pour faire le lien entre la base de données et l'application Unity.

Le lecteur d'empreintes

Nous devions utiliser le lecteur d'empreinte (GT-511C3, photo ci-contre). Black Swan Technology s'est occupé de la partie hardware et nous a fourni un boîtier complet avec la puce et une connexion USB (encore une fois reconnue comme un port série).

Cette fois-ci, la documentation était de qualité (voir en Annexe 3) ce qui a donc permis d'intégrer plutôt facilement les différents éléments du lecteur (l'enregistrement d'une empreinte, la recherche d'une empreinte, etc).

Quand un employé est connecté, il a la possibilité d'enregistrer son empreinte. Elle est ensuite envoyée à la base de données. Quand l'application démarre, on récupère les empreintes des employés qui ont la même équipe que celle du panneau où l'application tourne (le device ne peut contenir que 200 empreintes différentes).

Le système d'authentification

Le but était de fournir une boîte noire qui s'occupe de toute la partie authentification. J'ai donc fourni deux callbacks :

- Un qui est appelé lorsqu'un utilisateur se connecte
- Un autre qui est appelé lorsqu'il se déconnecte.

Pour accéder aux informations de l'utilisateur actuel (son/ses équipes, son nom, son csn, son rôle, etc.) on peut accéder à une instance de la classe AuthentificationData dans l'AuthentificationManager. Si personne n'est connecté, cette instance vaut null.

Les mails et les annonces

Il a également fallu intégrer OpenPOP.Net à l'application pour pouvoir récupérer des mails à distance. En effet, différentes boites mail vont être utilisées par l'application. En l'occurence, il a fallu se connecter à une boite mail pour récupérer les mails et en faire une annonce dans l'application (il y a une partie annonce, qui contient juste une liste d'annonces avec plus de détails quand on clique dessus). Cependant, une fois le mail récupéré, il est considéré comme traité et on y a plus accès. Il a donc fallu sauvegarder les mails dans la base de données.

Au final, trois classes principales ont été fournies :

- MailBox : représente une boîte mail (il faut donc fournir l'adresse email, le mot de passe, etc). C'est ici qu'OpenPop est utilisé pour récupérer les nouveaux mails distants.
- MailDatabaseManager : permet de faire le lien avec la base de données. On peut demander à sauvegarder un mail dans la base de données en précisant de quel MailBox il vient. On peut également récupérer les mails sauvegardés.
- MailManager : C'est une sorte de contrôleur. Il connait l'ensemble des MailBox de l'application, c'est lui qui fait le lien avec le MailDatabaseManager. Il s'occupe également de stocker localement l'ensemble des mails. C'est également lui qui va notifier le reste de l'application quand un nouveau message est arrivé.

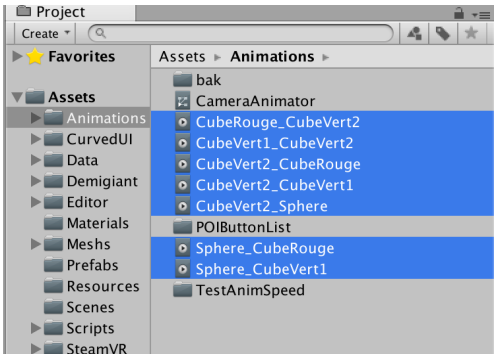
Camera

J'ai aussi travaillé sur un projet de visite virtuelle, également commandé par le même client que le premier projet. Le but est de pouvoir visiter une scène en VR (en l'occurence l'une des future usine de l'entreprise). On a une liste de points d'intérêt et on se rend de l'un à l'autre suivant des animations prédéfinies. On nous fournit la scène 3D, la liste des animations d'un point A à un point B (ils veulent contrôler eux-mêmes les animations) et mon objectif est de pouvoir importer tous les fichiers et générer dynamiquement les lieux (points d'intérêt) pour ne rien avoir à faire. Pour cela, nous avions le droit de définir la nomenclature des fichiers d'animation.

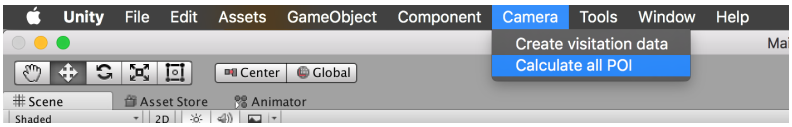
Finalement, il a été décidé que les fichiers d'animation devront être nommés de la manière suivante (POI = point of interest) :

POIDépart_POIArrivé

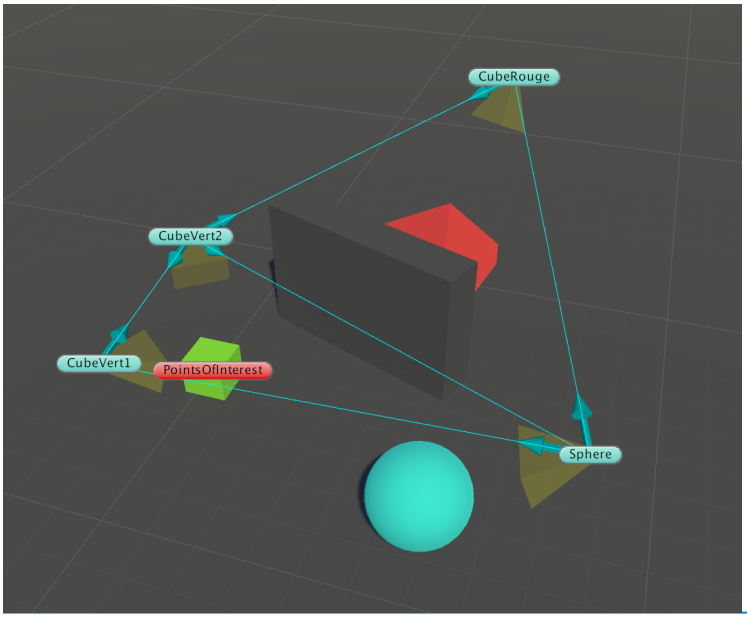
Voici donc le workflow final d'importation :



On sélectionne les fichiers à importer.

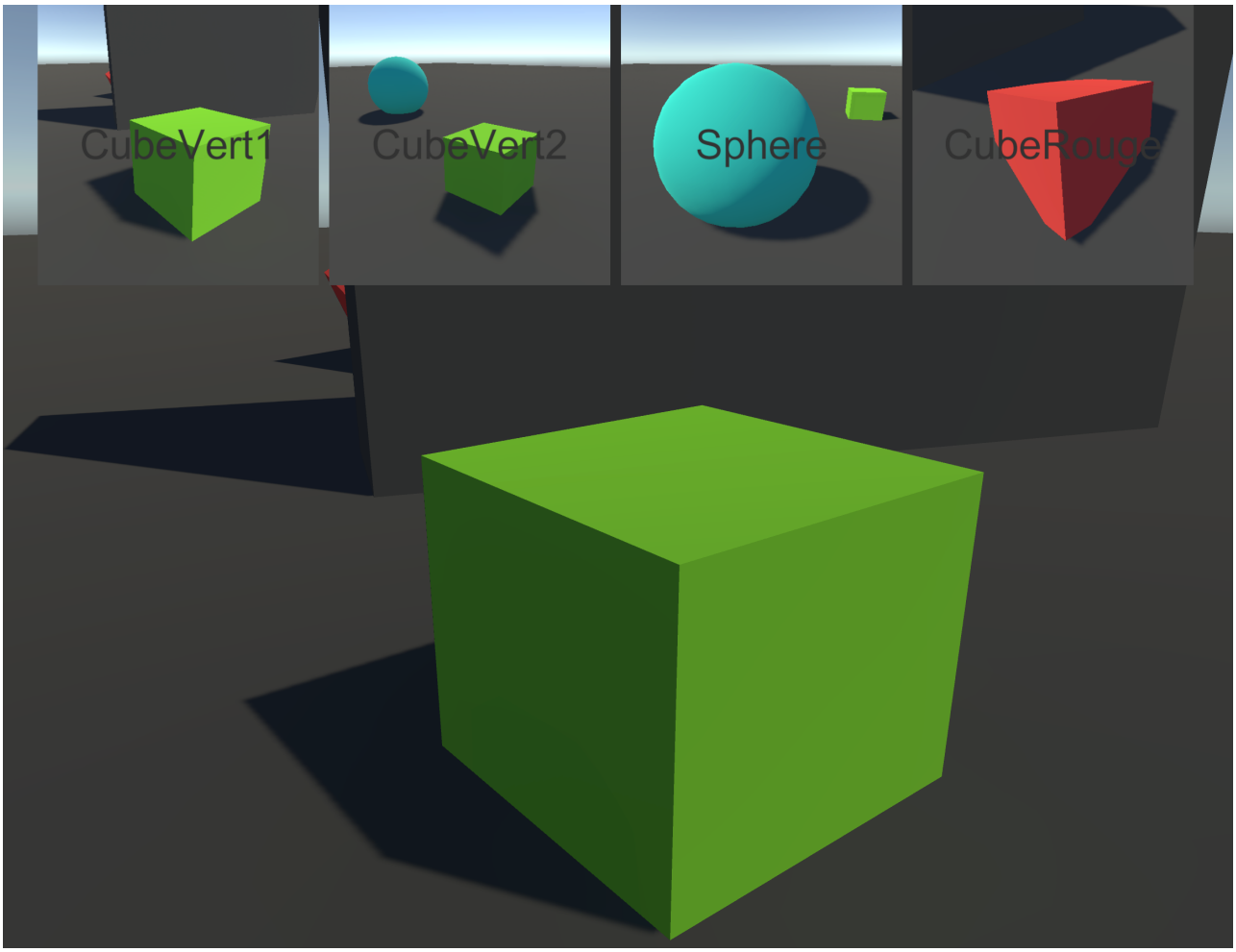


On lance le script.



Les fichiers sont importés, les points d'intérêts créés dans la scène. Des gizmos sont aussi mis en place pour permettre de facilement visualiser les différents points d'intérêt et les transitions.

Pendant l'importation, quand un nouveau point d'intérêt est enregistré, il va être créé.



Lorsque l'application est lancée, chaque POI va configurer sa caméra pour rendre l'image dans une RenderTexture, créer un POIButton et va changer l'image de ce bouton par la RenderTexture de la caméra. Quand on appuie sur un bouton, on va demander au POIManager de nous emmener au POI sélectionné.

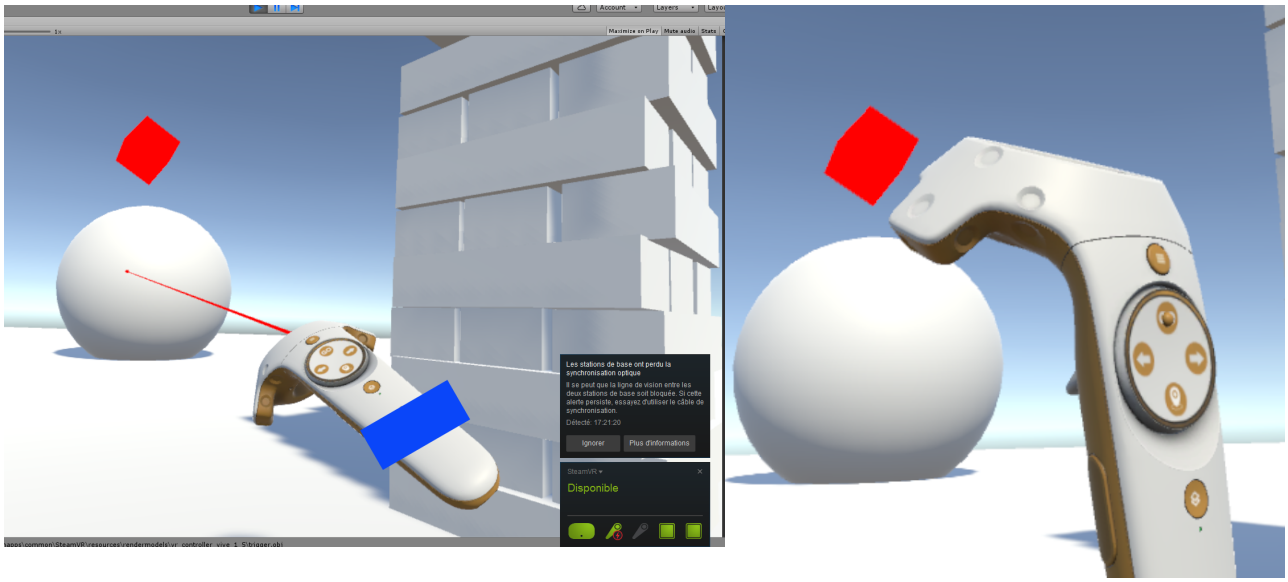
Si on est actuellement au point A et qu'on veut aller au point C, on regarde si une animation directe entre A et C existe. Si ce n'est pas le cas, on recherche dans le graph des chemins entre les différents points d'interêt, le plus court chemin entre A et C. Pour cela, on utilise A* avec une heuristique qui ne surestime pas le coût du but : la distance à vol d'oiseau. Il aurait d'ailleurs été intéressant d'utiliser une heuristique qui prend en compte la durée de l'animation mais dans ce cas il aurait été impossible de ne pas surestimer le coût du but (puisque cela dépend des animations qui vont suivre) et donc d'avoir le chemin le plus court (à moins de toujours utiliser 0, ce qui aurait été équivalent à l'algorithme de Dijkstra).

Cependant, ce genre d'animation n'est pas possible en VR. En effet, comme l'utilisateur reste immobile physiquement, le fait de le faire déplacer virtuellement et pire, de bouger sa « tête » en

bougeant la caméra est extrêmement perturbant. Finalement, l'application a été intégralement repensée. Voici la manette du Vive et le mapping retenu.

Quand on active le laser, on peut pointer différents éléments. Lorsque l'on pointe un PI, il devient sélectionné. Il ne se désélectionne que lorsqu'il quitte notre champ de vision ou si l'on sélectionne un autre PI. Un PI sélectionné affiche différentes informations en plus. Si on appuie sur la partie haute du trackpad, on est téléporté à ce PI, au point de vue le plus proche de nous. Un PI

a plusieurs points de vue. Au moment où je rédige ce rapport, je viens tout juste de commencer à travailler sur ce projet. Voici donc quelques images de l'état actuel du projet.

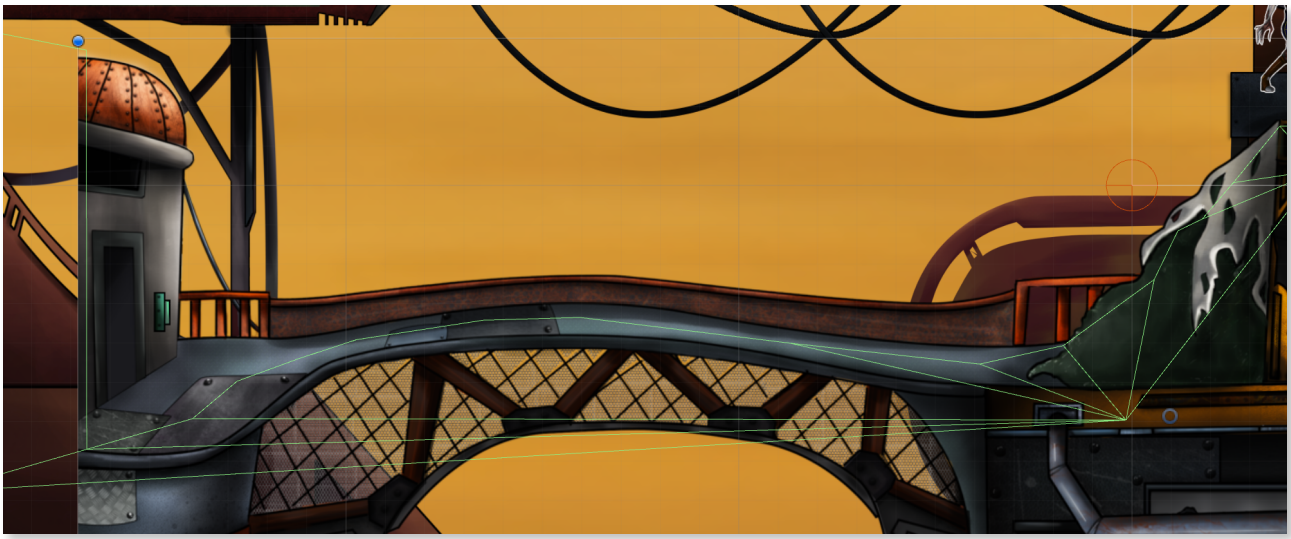


Mutate to Ghoul

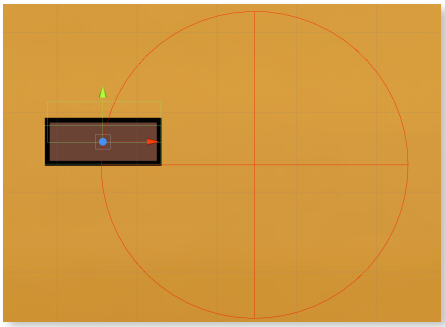
Enfin, dans le cadre de la fin de la période d'apprentissage de Coralie Sanchez, il a fallu programmer son jeu de fin d'étude. C'est un jeu d'aventure classique dans lequel on évolue dans un niveau afin de résoudre des petites énigmes qui nous permette de sauver les goules alliées. Il faut cependant faire attention aux goules ennemies.

Voici, en vrac, une partie des éléments du jeu réalisés :

- CharacterController. Basé sur celui de Sebastian Lague (https://www.youtube.com/playlist?list=PLFt_AvWsXl0f0hqURlhyIoAabKPgRsqjz). Le personnage progresse sur des PolygonCollider.



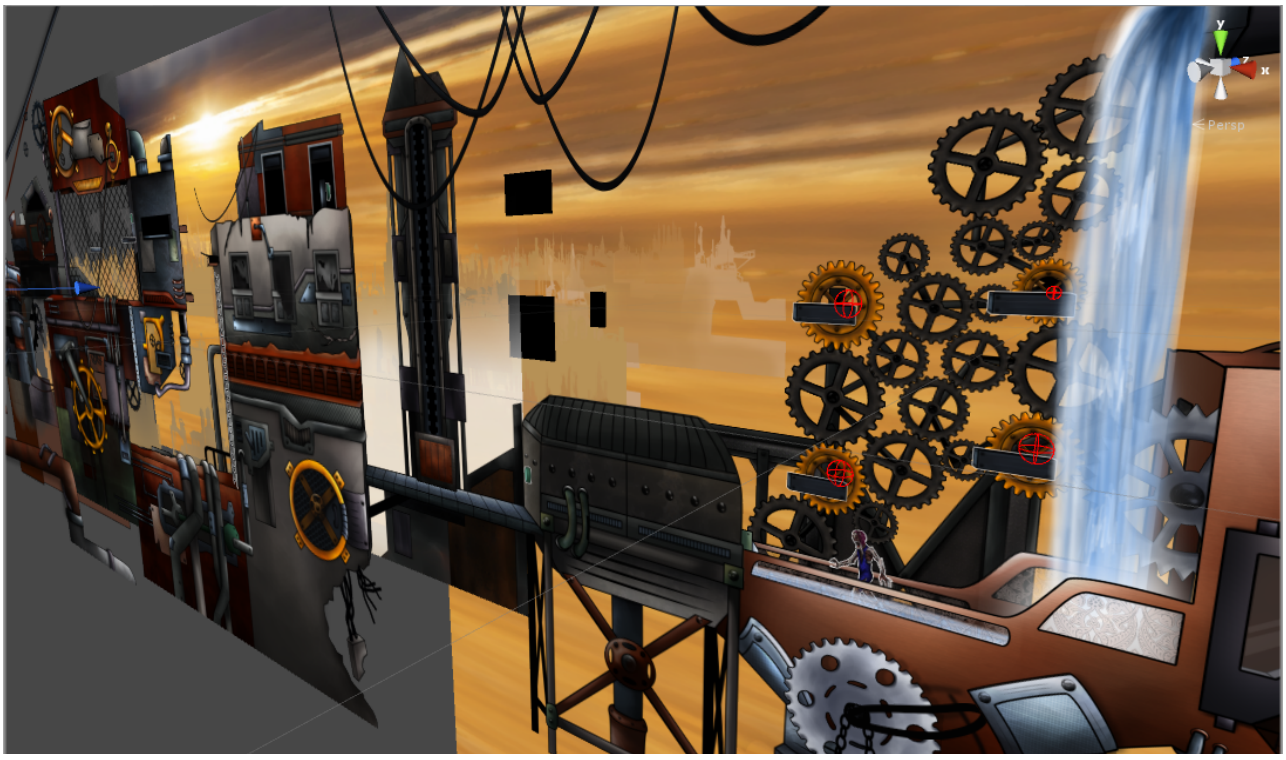
- IA alliée et ennemie. C'est une IA simple, puisqu'il s'agit d'un système de waypoint (l'agent essaie de rejoindre son waypoint uniquement sur l'axe x)
- Système de plateforme. Deux types de plateformes étaient demandés : une avec waypoints et une autre qui suivait un cercle.



- Le personnage principal a un pouvoir. Une fois activé, il permet de voir a travers certains murs. Pour cela, j'ai développé un shader spécifique en me basant sur le shader de sprite fourni par défaut. J'ai modifié le fragment shader pour cacher les pixels trop près du joueur (avec un dégradé).

- Un système de caisse qu'on peut pousser pour briser des planches ou tuer des ennemis.

- Un système de parallaxe intuitif. Une caméra s'occupe de rendre la scène principale en 2D (orthographique) et une autre, en perspective cette fois, s'occupe de rendre les différents éléments soumis à la parallaxe. Cela permet à Coralie de placer dans la scène les éléments en ajustant la profondeur et la parallaxe s'effectue automatiquement en fonction de la profondeur.



Conclusion

En conclusion, ce stage a été très intéressant car il m'a permis de refaire du bas niveau, ce qui peut être utile si je suis contraint à utiliser du hardware spécifique. J'ai aussi pu apprendre à utiliser des bases de données avec Unity via des web services. Enfin, j'ai pu découvrir le développement de jeu en VR avec le HTC Vive.

Annexes

Annexe 1 : La « documentation » du lecteur RFID

M302 mifare encoder communication protocol

Data stream format:

Data Strea Length (1BYTE) Command (1BYTE) Data D(1-20 BYTEs) CRC(1 BYTE)

Com port setting :

9600,8,n,1

Remark : in below description

-> means Host send data to module

<- means module send data to host

Function description and example

1) enquiry if module exist or not

->03 12 00 15 (Remark: 03 is length, 12 is command , 00 is data , 15 is CRC)

<-02 12 14

2) active buzzer

->02 13 15

<-02 13 15

3) Enquiry card

->03 02 00 05 Enquiry card by Idle mode (data=01, means all cards in RF area, data=00 means find Idle card)

<-03 02 01 06

4) Anticollision

->02 03 05

<-06 03 32 78 13 E8 AE (module return card serial number is：327813E8）

5) select card

->02 04 06

<-02 04 06

6) Stop card

->02 09 0B

<-02 09 0B

7) download keyA or keyB

->0A 0B 00 0F FF FF FF FF FF FF 1E download sector15 KEY

（Parameter: download mode + sector + key；mode=0, download KeyA; 01 means KeyB）

<-02 0B 0D

8) read data

->03 06 00 09 read block 00（parameter：block）

<-12 06 32 78 13 E8 B1 08 04 00 62 63 64 65 66 67 68 69 A6

9) write data

->13 07 04 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 2E

write data to block 04(block0 in sector1)（parameter：block+data）

<-02 07 09

10) verify key

->04 05 01 00 0A verify sector 01 password with default KEYA

（parameter: sector+verify mode; verify mode：00 -means default KEYA，01 means default KEYB，02 means directly KEYA，03 means directly by KEYB）

<-02 05 07

11) block operation

->09 08 01 04 04 00 00 00 01 1B block 04 value add 0x00000001

(parameter：operation mode+source block address+destination block+ value)

<-02 08 0A

Annexe 2 : La classe de base pour utiliser les Threads dans Unity

```
public class ThreadedJob {
	private bool m_IsDone = false;
	protected object m_Handle = new object();
	private System.Threading.Thread m_Thread = null;

	public bool IsDone {
		get {
			bool tmp;

			lock (m_Handle) {
				tmp = m_IsDone;
			}

			return tmp;
		}

		set {
			lock (m_Handle) {
				m_IsDone = value;
			}
		}
	}

	/// <summary>
	/// Create and run the job
	/// </summary>
	public virtual void Start() {
		m_Thread = new System.Threading.Thread(Run);
		m_Thread.Start();
	}

	/// <summary>
	/// Force stop the job
	/// </summary>
	public virtual void Abort() {
		m_Thread.Abort();
	}

	/// <summary>
	/// Job function. SHOUD NOT CONTAIN UNITY API CALL !
	/// </summary>
	protected virtual void ThreadFunction() { }
	/// <summary>
	/// Called when the job is finished. May contain Unity API.
	/// </summary>
	protected virtual void OnFinished() { }

	/// <summary>
	/// This should be called manually every Unity's update.
	/// </summary>
	public virtual bool Update() {
```

```
            if (IsDone) {
                  OnFinished();
                  return true;
            }

            return false;
      }

      /// <summary>
      /// Equivalent to the function Update.
      /// Allow to pause the coroutine
      /// until the job is done.
      /// </summary>
      /// <returns>The for.</returns>
      public IEnumerator WaitFor() {
            while(!Update()) {
                  yield return null;
            }
      }

      /// <summary>
      /// Launch the job.
      /// </summary>
      private void Run() {
            ThreadFunction();
            IsDone = true;
      }
}
```

Annexe 3 : La documentation du GT-511C3

Voir le document correspondant (Annexe3_GT-511C3_datasheet_V1 1_20130411[4]).