2013 - 2014

Cycle Ingénieur, 2ème année

Projet étudiant

LEVILAIN Rémi

ROUSSEL Fabien

Réalisation d'une application de réalité virtuelle pour l'évaluation des émotions

IstiA
École d'Ingénieurs de l'Université d'Angers

université angers

Réalisation d'une application de réalité virtuelle pour l'évaluation des émotions

LEVILAIN Rémi
ROUSSEL Fabien

Remerciements

Nous remercions l'ensemble des personnes qui ont contribué à l'avancement de ce projet. Nous tenons plus particulièrement à remercier M. Teddy PAYEN pour ces connaissances techniques à propos de la réalisation d'une application de réalité virtuelle.

Pour finir, je remercie Mme Deborah FOLOPPE notre encadrante ainsi que M. Paul RICHARD notre enseignant tuteur.

Rémi LEVILAIN & Fabien ROUSSEL

Réalisation d'une application de réalité virtuelle pour l'évaluation des émotions

LEVILAIN Rémi
ROUSSEL Fabien

Sommaire

RAPPORT DE PROJET :

Réalisation d'une application de réalité virtuelle pour l'évaluation des émotions

LEVILAIN Rémi
ROUSSEL Fabien

Réalisation d'une application de réalité virtuelle pour l'évaluation des émotions

LEVILAIN Rémi
ROUSSEL Fabien

I. INTRODUCTION

Dans un monde où la technologie progresse à grande vitesse, la réalité virtuelle se perfectionne et innove chaque jour. Que vous soyez passionné ou tout simplement curieux, ce rapport vous présentera les avancées technologiques et les limites présentes dans ce domaine.

1. Présentation de l'ISTIA

Composante de l'Université d'Angers, l'ISTIA (Institut des Sciences et Techniques de l'Ingénieurs d'Angers) est aujourd'hui une École d'Ingénieurs habilitée par la Commission des Titres d'Ingénieurs.

Figure 1 : L'ISTIA

L'ISTIA a pour objectif de former des ingénieurs opérationnels grâce à des enseignements théoriques et pratiques ainsi qu'au travers d'expériences acquises lors de projets et de stages. A l'issue de la troisième année, commune à tous les étudiants, l'étudiant s'oriente vers l'option de son choix : Automatique et Génie Informatique, Qualité et Sureté de Fonctionnement, ou Ingénierie de l'Innovation. Pour plus de détails, les chiffres clés de l'Université d'Angers sont disponibles à l'annexe A1.

II. LA REALITE VIRTUELLE

Il existe à l'ISTIA, lors de la 5ème année en option Automatique et Génie Informatique, une spécialité IHM/RV (Interface Homme-Machine/Réalité Virtuelle). Afin d'aborder la suite de ce rapport plus sereinement voici une brève explication de la réalité virtuelle. Certains mots issus du vocabulaire technique sont suivit d'un astérisque (*) signifiant que leur définition est écrite dans le glossaire à la fin de ce rapport.

1. Présentation

La réalité virtuelle est un procédé visant à immerger une personne dans un univers virtuel. Pour cela, une application de réalité virtuelle va tenter de stimuler plusieurs sens que ce soit le visuel, l'haptique ou le son. Les mondes virtuels peuvent aussi bien simuler le monde réel de façon à effectuer une mise en situation ou bien être un monde purement imaginaire.

Figure 2 : Exemple de CAVE*

Une interface de réalité virtuelle va donc utiliser un matériel spécifique. Pour favoriser l'immersion, la réalité virtuelle va en effet utiliser entres autres des affichages stéréoscopiques, des interfaces sonores 3D ou encore des interfaces haptiques.

2. Exemples d'applications

- **Réalité virtuelle appliquée aux serious games :**

On peut premièrement réaliser des serious games (jeu sérieux). Par exemple dans le domaine de la médecine pour apprendre aux futurs médecins les différents gestes. On comprend alors l'intérêt de l'immersion puisqu'il est nécessaire de rendre l'application la plus proche de ce que le futur médecin sera amené à faire. Ci-contre, le Jeu « Pulse ! » est destiné à l'apprentissage de gestes médicaux autant que du diagnostic. Le jeu présente notamment des patients virtuels très réalistes.

Figure 3 : Jeu de simulation médicale : Pulse

Figure 4 : Simulateur d'intervention de pompiers KIMM Fire

Dans le même ordre d'idée, on trouve certains serious game créés pour l'entrainement des pompiers. Ci-contre, KIMM Fire Simulator prépare les pompiers à une éventuelle intervention dans des milieux restreints.

- **Réalité virtuelle déstinée à l'amusement :**

On trouve également des applications de réalité virtuelle pour l'amusement tel que le Minecart (train de la mine). L'image stéréoscopique de l'Oculus Rift augmentant considérablement l'immersion, l'expérience en devient spectaculaire.

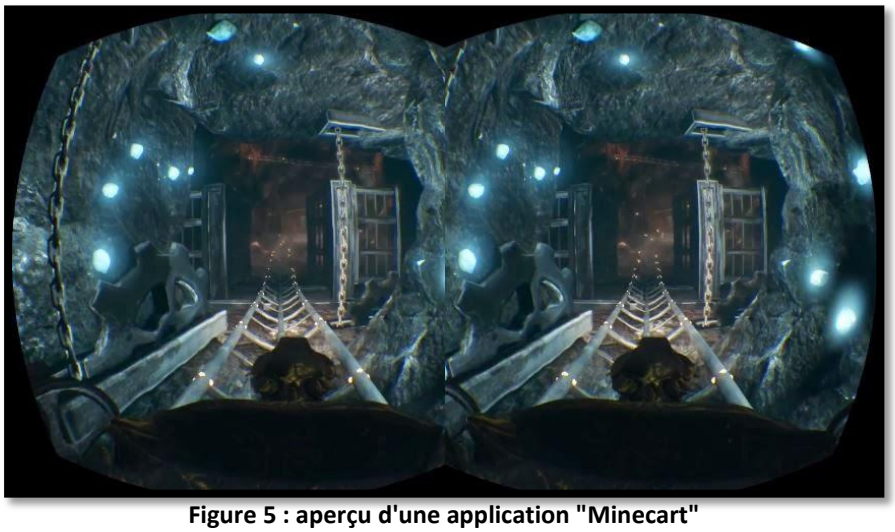


Figure 5 : aperçu d'une application "Minecart"

III. CONCEPTION DE L'APPLICATION

Dans le cadre de notre 4ème année au sein de l'ISTIA en option Automatique et Génie Informatique nous avons eu un projet étudiant à réaliser dans les domaines de notre formation. Etant deux étudiants souhaitant nous orientés vers la réalité virtuelle nous avons choisi un sujet de réalisation d'une application ayant pour but l'identification des émotions. L'usage de l'immersion virtuelle est donc le meilleur moyen de simuler un environnement dans lequel on pourra observer l'évolution ainsi que les réactions d'une personne.

1. Etude de l'existant

Au sein du laboratoire LARIS (Laboratoire Angevin de Recherche en Ingénierie des Systèmes), différents projets ont été réalisés utilisant les périphériques de réalité virtuelle que nous allions implémenter dans notre démarche. Nous pensions alors réutiliser les programmes précédemment établis afin de ne pas " réinventer la roue ".

Différents environnements libres de droit sont disponibles sur internet, afin de ne pas perdre de temps en modélisation nous avions prévu d'en récupérer plusieurs pour réaliser nos terrains.

Les pilotes des périphériques sont fournis par les constructeurs afin de faciliter leur utilisation et le plus souvent la communauté internet contient des projets avec des tutoriaux pour indiquer comment installer le périphérique. Nous avions donc des bases déjà existantes pour réussir l'implémentation des différents matériels nécessaires à notre application.

2. Définition de l'objectif

Les objectifs ont été définis lors des premières réunions du projet par M. RICHARD, notre enseignant tuteur et Mme FOLOPPE, notre encadrante et doctorante en psychologie. Après ces rencontres, différentes propositions ont été retenues et ensuite validées sur le plan technique par M. PAYEN.

- Réaliser 5 terrains inspirant respectivement une émotion : peur, tristesse, dégoût, joie, zen.
- Modéliser une zone de tutoriel
- Obtenir différentes méthodes de navigation dans l'environnement
- Intégrer les différents périphériques : l'Oculus Rift, le Razer Hydra et la veste haptique

Peu de temps après avoir défini nos différents objectifs M. Richard nous a annoncé que la modélisation des terrains sur Unity3D serait désormais réalisée par 2 étudiants de l'UFR Sciences d'Angers dans le cadre d'un de lors projet. Cela nous a donc permis de recentré nos objectifs sur les différents périphériques, la navigation ainsi que la zone de tutoriel.

3. Moyens à mettre en œuvre

Au vu des objectifs à atteindre il a été important de choisir les éléments nécessaires pour réaliser ces différentes tâches. Il a tout d'abord été logique de choisir le logiciel Unity3D pour créer et mettre en place les environnements virtuels.

Figure 6 : Logo d'Unity3D

Unity3D est un logiciel de création d'environnements virtuels qui possède un moteur physique basé sur le moteur PhysX de Nvidia. Il présente plusieurs avantages : sa librairie est tout d'abord assez fournie et complète à la fois pour les mathématiques, la physique ainsi que pour les effets de camera. Son système d'interface en glisser-déposer est également très intuitif. La version professionnelle dispose également d'un moteur d'ombre et d'un excellent rendu de l'eau.

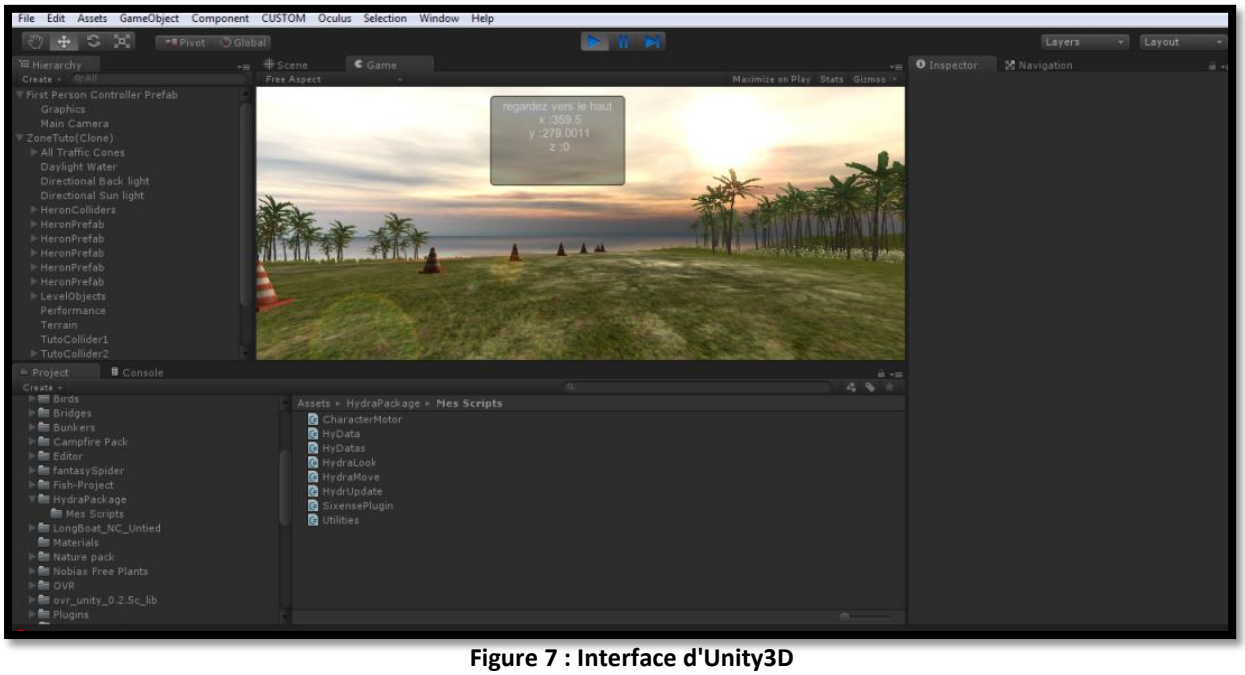


Figure 7 : Interface d'Unity3D

Pour former une application, il suffit d'assembler tous ces éléments dans l'éditeur grâce à des dépendances et des interactions. Certaines d'entre-elles sont déjà incluses dans Unity3D notamment les interactions physiques.

De plus, cet éditeur est fourni avec le logiciel de développement "MonoDevelop" afin de pouvoir réaliser les scripts. Au cours de ce projet, nous avons préféré utiliser "Microsoft Visual Studio 2012" avec lequel nous travaillons régulièrement au sein de l'ISTIA et dont les fonctionnalités nous sont plus familières.

Il a été nécessaire d'utiliser des ressources extérieures aux logiciels afin de réaliser notre application. Certains éléments des environnements ont été choisis sur des sites internet tels que le " Unity Assets Store ", "TF3DM " ou " TurboSquid ". Nous nous sommes également servis des forums officiels d'Unity3D ainsi que des différents tutoriels existants. Nous nous sommes également servis du site internet "freesfx" pour trouver des sons gratuits et de bonne qualité pour notre application. Enfin, le logiciel "Audacity" nous a permis de retoucher aux sons que nous avions téléchargés afin de les inclure proprement dans notre environnement.

Le Razer Hydra est un contrôleur de jeu à détection de mouvements conçue par l'entreprise Sixense. Ce périphérique présente plusieurs particularités intéressantes. Contrairement à une manette classique, l'Hydra est constituée de deux parties permettant ainsi aux deux mains d'être indépendantes. Il est également constitué d'une base possédant un capteur de mouvement électromagnétique d'une précision à l'échelle du millimètre.

Figure 8 : Le Razer Hydra

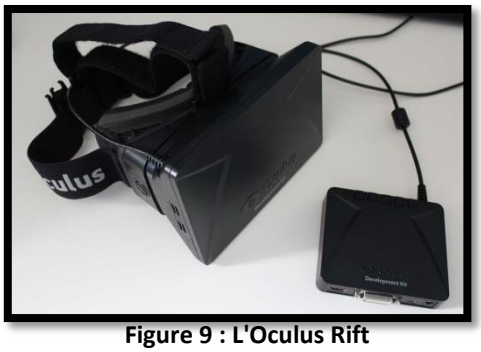


Figure 9 : L'Oculus Rift

L'Oculus Rift est un masque de réalité virtuelle. Il fonctionne grâce un détecteur de mouvement intégré au casque et connecté à l'ordinateur via USB. Ce capteur permet d'adapter la camera d'une application 3D en fonction de l'orientation de la tête du joueur. L'Oculus possède également un écran plat sur lequel est affichée une image stéréoscopique récupérée par deux caméras présentes dans l'environnement. Un système de lentilles placées devant les yeux du joueur permet à ce dernier d'obtenir une vision en relief de l'image. Ainsi, l'utilisation de l'Oculus Rift ajoute un vrai plus immersif dans une application de réalité virtuelle.

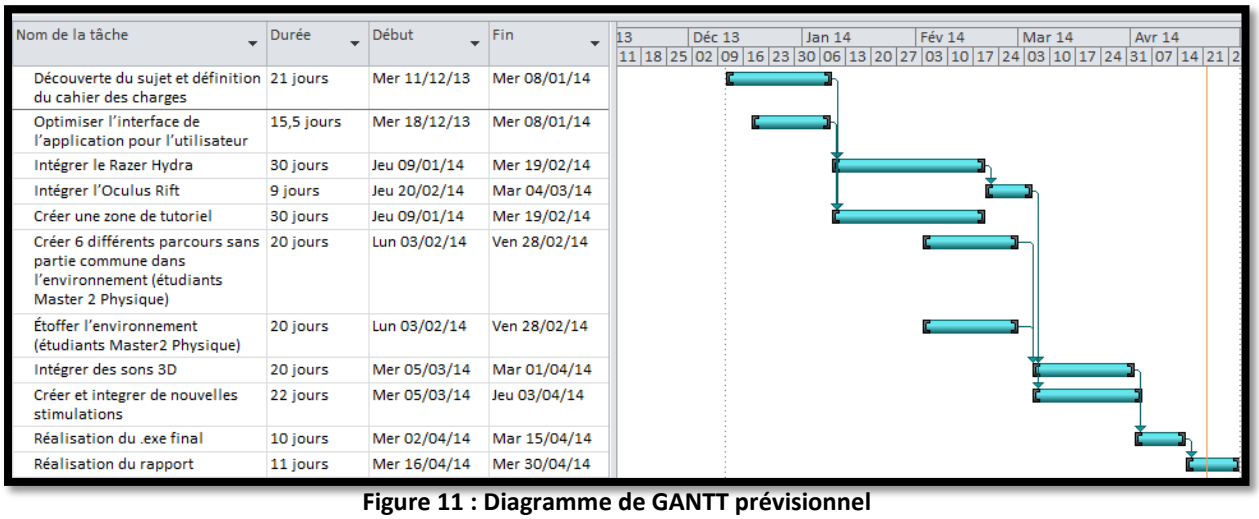
La veste haptique est un gilet sur lequel est cousue une dizaine de petit moteur électrique commandé par une carte Arduino Uno. Elle a été ensuite améliorée en rajoutant une autre carte permettant une liaison Wifi avec l'ordinateur évitant ainsi à l'utilisateur d'être dérangé par les fils. Il est nécessaire d'avoir une alimentation externe à l'aide d'une pile 9V car une alimentation simple par USB serait insuffisante pour l'ensemble des cartes et des moteurs, de plus cela rajouterai un fil supplémentaire.

Figure 10 : Aperçu de la veste haptique et ses moteurs

4. Définition et répartition des tâches à réaliser

Une fois l'ensemble des tâches et des moyens à notre disposition défini nous avons pu nous répartir le travail de manière à être le plus efficace possible. Fabien avait comme tâches la modélisation du déplacement dans l'environnement, l'ajout de l'Oculus Rift et la réalisation de la zone de tutoriel. Rémi travaillait sur l'implémentation du Razer Hydra, la réalisation des différentes méthodes de navigation ainsi que l'ajout de la veste haptique. Afin d'être plus clair nous avons ainsi réalisé un diagramme de GANTT prévisionnel présenté sur la page ci-contre

Ce diagramme montre quelles étaient les différentes tâches prévues au début du projet, c'est-à-dire début décembre 2013. Comme on peut le voir plusieurs tâches s'effectuent en parallèles étant donné la répartition effectuée entre Fabien et Rémi. Ce diagramme a évolué au cours du projet car différentes décision ont été prises en fonction de l'avancement des différentes étapes. Ces modifications seront abordées dans le chapitre associé à la gestion de projet.



Nom de la tâche	Durée	Début	Fin
Découverte du sujet et définition du cahier des charges	21 jours	Mer 11/12/13	Mer 08/01/14
Optimiser l'interface de l'application pour l'utilisateur	15,5 jours	Mer 18/12/13	Mer 08/01/14
Intégrer le Razer Hydra	30 jours	Jeu 09/01/14	Mer 19/02/14
Intégrer l'Oculus Rift	9 jours	Jeu 20/02/14	Mar 04/03/14
Créer une zone de tutoriel	30 jours	Jeu 09/01/14	Mer 19/02/14
Créer 6 différents parcours sans partie commune dans l'environnement (étudiants Master 2 Physique)	20 jours	Lun 03/02/14	Ven 28/02/14
Étoffer l'environnement (étudiants Master2 Physique)	20 jours	Lun 03/02/14	Ven 28/02/14
Intégrer des sons 3D	20 jours	Mer 05/03/14	Mar 01/04/14
Créer et integrer de nouvelles stimulations	22 jours	Mer 05/03/14	Jeu 03/04/14
Réalisation du .exe final	10 jours	Mer 02/04/14	Mar 15/04/14
Réalisation du rapport	11 jours	Mer 16/04/14	Mer 30/04/14

Figure 11 : Diagramme de GANTT prévisionnel

IV. REALISATION DE L'APPLICATION

Après nous être répartit les tâches nous avons donc commencé à travailler sur les différents éléments de l'application à proprement parlé. Nous décrirons tout d'abord de l'implémentation des périphériques, de la navigation puis de la zone de tutoriel.

1. Implémentation des périphériques

L'implémentation des périphériques consiste à les relier à l'application par le biais de différents programmes tels que les pilotes par exemple. Dans le cas du Razer Hydra, la réalisation de plusieurs scripts a été nécessaire contrairement à l'Oculus Rift où la majorité de l'implémentation est faite à l'aide des pilotes fournis par le fabricant.

- Le Razer Hydra :

Pour commencer l'implémentation du Razer Hydra nous avons utilisés les pilotes ainsi que les projets de démonstration de Sixense, le constructeur. Ceci nous a permis d'observer le fonctionnement de l'Hydra sous Unity3D et ainsi de nous inspirer de ce qui avait déjà été réalisé.

Après différentes observation sur le code fourni par Sixense, il a été décidé de récupérer le plug-in* reliant les données internes aux contrôleurs de l'Hydra à l'ordinateur et de refaire les couches de programmation supérieures. Pour ce faire, M. Payen nous a conseillé d'utiliser une architecture 3-tiers nous permettant de récupérer et d'exploiter proprement les données du périphérique. Ne connaissant pas encore cette architecture, M. Payen a pris le temps de nous expliquer son fonctionnement et son utilité afin que nous puissions commencer à la programmer plus rapidement.

Une architecture 3-tiers comporte différents scripts permettant la récupération des données brutes du périphérique, leur traitement et leur transmission à l'utilisateur. Respectivement ce sont les couches d'accès aux objets (DAO), couche métier et coucher client. Appliquer aux différentes scripts du Razer Hydra, nous obtenons le diagramme suivant :

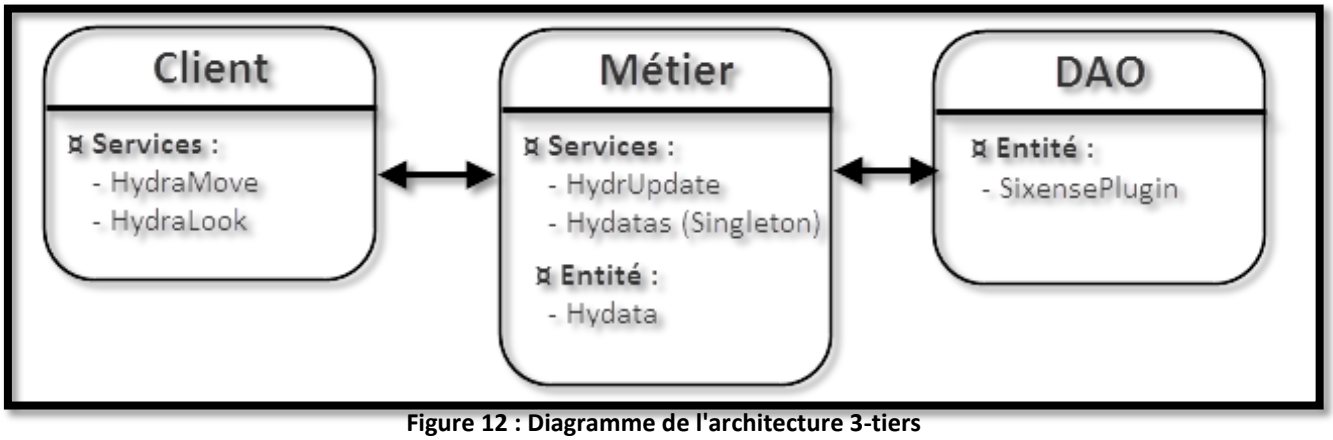


Figure 12 : Diagramme de l'architecture 3-tiers

Un diagramme UML plus détaillé est disponible en annexe A3 si vous souhaitez avoir une vue d'ensemble sur le contenu de ces classes.

Nous parlerons de la couche Client lorsque nous aborderons la réalisation de la navigation. Afin de comprendre comment fonctionne l'implémentation nous allons principalement décrire la couche Métier.

Un script "Hydata" contient les données d'une des manettes du Razer Hydra. Le script "Hydatas" contient 2 objets du type "Hydata", on peut donc l'associer à la représentation physique de l'Hydra. De plus c'est un "Singleton", cela signifie qu'un seul exemplaire de ce script peut être instancié, en effet l'application ne comporte qu'un seul Razer Hydra, la présence de plusieurs objets "Hydatas" nous poserait problème. Pour plus de précision la déclaration d'un singleton est présentée en annexe A2.

Le fonctionnement ensuite est simple, les 2 objets "Hydata" enregistrent les données fournies par la couche DAO. L'objet "Hydatas" regroupe les "Hydata" et permet la lecture des données qu'ils contiennent. Enfin le script "HydrUpdate" met à jour continuellement les valeurs des objets Hydata via le Hydatas. On peut alors faire le distinguo entre les services qui sont les scripts effectuant des actions alors que les entités ne contiennent que des informations utiles.

- L'Oculus Rift :

L'implémentation de l'Oculus est principalement réalisée à l'aide des plug-ins inclus par le constructeur. L'entreprise OculusVR fourni en effet de nombreux prefabs* qui sont des ensembles préconstruits facilitant l'ajout de nouveaux éléments complexes dans un projet. Pour nos besoins nous avons choisis le prefab* "OVRCameraController". Ce dernier permet d'obtenir un objet camera indépendant des mouvements du joueur.

Figure 13 : Le prefab OVRCameraController

Nous avons bien sûr pris soin de désactiver la camera principale sans la supprimer car elle contient des scripts importants, notamment le script des tâches à effectuer pour le tutoriel. Il ne reste plus qu'à dupliquer l'écran comme pour un vidéo projecteur et bien sûr mettre l'application en plein écran.

- La veste haptique :

L'implantation de la veste haptique est arrivée en fin de projet une fois l'Hydra et l'Oculus en place. Nous pensions que l'implantation se ferait sans trop de difficulté à l'aide des précédents projets réalisés mais il s'est avéré qu'aucun ne décrivait précisément la procédure d'installation et d'utilisation de la veste. De plus les alimentations externes à l'ai des piles n'était plus présent. Nous avons tout de même tenté de lancer certaines applications mais le protocole de communication filaire ou bien Wifi ne fonctionnait pas. La fin du projet arrivant nous n'avons pas eu le temps d'installer la veste haptique sur notre application. Nous espérons que l'idée sera reprise par de futurs étudiants.

2. Réalisation de la navigation

- Modification de la navigation basique d'Unity3D

Pour commencer, nous avons récupéré le prefab* de caméra à la 1ère personne proposé par Unity3D. Il n'est fourni qu'en JavaScript, nous l'avons convertis en C# afin d'uniformiser nos scripts en un seul et même langage. Nous avons ensuite ajouté différentes fonctionnalités pour rendre la navigation plus réaliste.

Nous avons dans un premier temps décidé d'ajouter des bruits de pas lorsque le joueur se déplace. Cet effet sonore renforce nettement l'impression de marche de l'utilisateur et améliore ainsi son immersion. Nous nous sommes premièrement renseignés sur des forums sur la meilleure façon de générer des sons de pas réalistes.

```
audio.volume = Random.Range(0.22F,0.5F);
audio.pitch = Random.Range(1F,1.4F);
audioDelay = delayShort;
audio.Play();
```

Dans le First Person Walker de Unity, nous avons implémenté plusieurs fonctions : une fonction par type de terrain sur lequel le joueur est susceptible de marcher, plus une fonction pour le saut. Lorsque qu'elles sont activées, chacune de ces fonctions émet un son de pas sélectionné aléatoirement parmi une liste d'objets audio nommés "Audioclip*". Ces "Audioclips*" sont alors lus à une vitesse aléatoire pour que les sons soit plus ou moins aigus.

Le délai précédant la lecture est lui aussi aléatoire. Tout ceci créé alors une marche réaliste car aucun bruit de pas successif ne se ressemble. Ci-dessous, l'ensemble des composants du script "First Person Walker" après avoir effectué nos modifications. Il y a 2 extraits de sons de pas pour chaque type de terrains. Nous avons bien sûr édité ces sons avec "Audacity" pour qu'ils soient les plus courts possibles et prennent ainsi moins d'espace.

Figure 14 : Composants du script "First Person Walker"

Pour la détection du type de terrain, nous nous sommes servis des tags* (chaîne de caractères associée à un objet) pour pouvoir déterminer le type de terrains sur lequel le joueur est en train de marcher. Par exemple, pour une cabane en bois, nous avons mis un tag* "WoodFloor" sur le sol de cette cabane (image ci-dessous). Il a ensuite fallu chercher ce tag* dans l'objet qui est en contact avec le personnage de façon à appeler la fonction appropriée en fonction de chaque situation dans lequel le joueur se trouve.

Figure 15 : Tag WoodFloor sur le sol

Sur la zone de tutoriel, il y a également certains endroits où le joueur déplace dans l'eau sans être totalement immergé. Nous avons donc implémenté une fonction pour générer la marche dans l'eau en effectuant une comparaison entre la taille du personnage et le niveau de l'eau. Nous avons enfin ajouté dans ce script un bruit de plongeon lorsque le joueur s'immerge. Enfin, il ne restait plus qu'à implémenter l'appel de ces fonctions sous condition que le joueur soit en train de marcher.

Nous avons également créé d'autres scripts pour améliorer la navigation. Premièrement nous en avons écrit un pour améliorer les effets de camera lorsque le joueur est sous l'eau en activant un flou bleuté et un "Audioclip*" d'ambiance sous-marine.

```
if (Mathf.Abs(horizontal) == 0 && Mathf.Abs(vertical) == 0)
{
    timer = 0.0f;
}
else
{
    waveslice = Mathf.Sin(timer);
    timer = timer + bobbingSpeed;
    if (timer > Mathf.PI * 2)
    {
        timer = timer - (Mathf.PI * 2);
    }
}
```

Ensuite, nous avons ajouté un script "Headbobber" sur la camera. Celui-ci permet de faire un mouvement de camera sinusoïdal sur l'axe vertical du personnage lorsque le joueur se déplace simulant ainsi le hochement de la tête. Pour cela, on effectue le sinus d'une variable que l'on incrémente à chaque frame*, autrement dit à chaque mise à jour de l'application. En effet, lorsqu'elle est exécutée l'ensemble des coordonnées sont recalculées en permanence et chaque boucle de calcul est appelée une frame*.

- Déplacement à l'aide du Razer Hydra

Une fois l'implémentation du Razer Hydra nous avons pu exploiter ces fonctionnalités en programmant 2 types de navigation différents.

Figure 16 : Stick et boutons d'un contrôleur de l'Hydra

Le 1er permet de se déplacer à l'aide des deux sticks présent sur chaque manette. Le stick gauche permet le contrôle du déplacement alors que le droit oriente votre caméra. D'un point de vue de la programmation il s'agit de deux variables par stick à récupérer associée aux axes vertical et horizontal. La valeur de ces données est comprise entre 0 et 1 en fonction de l'inclinaison donnée par l'utilisateur, la position neutre du stick étant l'associant des deux valeurs à 0.

Une fois ces informations récupérées elles sont traitées par les scripts de la couche Client : l'HydraMove et l'HydraLook associés respectivement aux déplacements et à la caméra. Ces programmes sont ainsi appelés continuellement par l'application afin d'obtenir les mouvements du personnage et de la caméra à chaque instant. Basés sur des scripts déjà existant d'Unity3D il a été nécessaire de les modifier afin de relier la valeur des variables de sticks aux directions ainsi qu'à l'intensité des déplacements à effectuer.

Dans la programmation spécifique d'Unity3D des entités ont été créées afin de simplifier le code des mouvements. La présence, par exemple, de Vector2 ou Vector3 permet de regrouper 2 ou 3 valeurs d'axes dans une seule et même variable. Dans notre cas, un Vector2 nous permet de contenir les informations d'un stick. Qui plus est cette variable est réutilisée dans les fonctions propres à Unity3D pour gérer les déplacements d'entités (personnage, caméra, etc…), il se révèle donc indispensable de les utiliser pour une meilleure compréhension et programmation.

La 2nde méthode de navigation utilise l'inclinaison donnée aux 2 contrôleurs du Razer Hydra et conserve le stick droit pour l'orientation de la caméra si l'Oculus n'est pas employé. Dans cette configuration on récupère les variables liés aux rotations gauche/droite et avant/arrière afin de piloter respectivement la rotation et la marche ou le recul du personnage. Ces variables sont comprises entre 0 et 360°, afin d'uniformiser l'intensité du mouvement elles sont ramenées entre 0 et 1. De plus une moyenne est faite entre les 2 contrôleurs afin que l'utilisateur oriente chacun d'entre eux pour se déplacer et non qu'un seul. Tout ce traitement s'effectue dans les scripts HydraMove et HydraLook en continu.

Pour finir des options sont accessibles avant de lancer l'application afin d'ajuster la sensibilité des mouvements qui reste parfois assez délicate à gérer. En effet l'écart d'amplitude de mouvement entre le déplacement aux sticks ou grâce à l'inclinaison des contrôleurs est assez important. Il faut donc prendre de paramétrer la sensibilité avant de lancer la tâche.

3. Réalisation du parcours de déplacement

Il fallait, pour répondre au cahier des charges, créer une suite de terrains correspondant aux émotions désirées. Sachant que l'ordre d'apparition d'environnements ainsi que le sens dans lequel on les parcourt est aléatoire.

Dans un premier temps, nous avons choisi de considérer les terrains comme les pièces d'un puzzle. Autrement dit, chaque environnement possédait une entrée et une sortie. Il suffisait ensuite de faire correspondre celles-ci aléatoirement.

Pour cela, nous avons créé un script dans l'objet du personnage comprenant la liste des environnements disponibles sous la forme de prefabs*. Nous avons également créé un script dans chacune de ces prefab* qui permettait de stocker la liste de leurs entrées/sorties disponibles de chaque terrain sous la forme de repère dans l'espace. La rotation de ces derniers doit être ajustée de manière à indiquer comment le terrain suivant doit être placé.

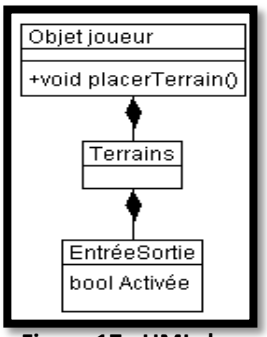


Figure 17 : UML de l'organisation des terrains

Ensuite, il a suffi de rassembler toutes ces informations à partir du script principal situé dans l'objet du personnage de façon à instancier correctement les terrains avec la bonne position et la bonne rotation.

Malgré cela nous nous sommes rapidement aperçus que la rotation des terrains ne fonctionnait pas. Après avoir pensé qu'il s'agissait d'une erreur de programmation, nous avons observés que les objets situés dans la même prefab* que le terrain se plaçaient correctement, montrant ainsi que le code était bon. Après une recherche sur internet, nous nous sommes aperçu que la technologie utilisée par Unity3D ne permettait pas d'effectuer une rotation sur un terrain. Il a fallu alors changer le cahier des charges.

Nous avons opté pour une téléportation du joueur de terrain en terrain à travers des portails. Pour cela nous avons repris l'architecture du programme précèdent avec toujours une liste de prefab* d'environnements situé dans un script principal avec dans chacun d'eux un code contenant la liste des entrée/sortie. Nous avons cependant ajouté un script "Teleport" sur chaque entrée/sortie ainsi qu'une BoxCollider déclenchant l'évènement "OnTriggerEnter*".

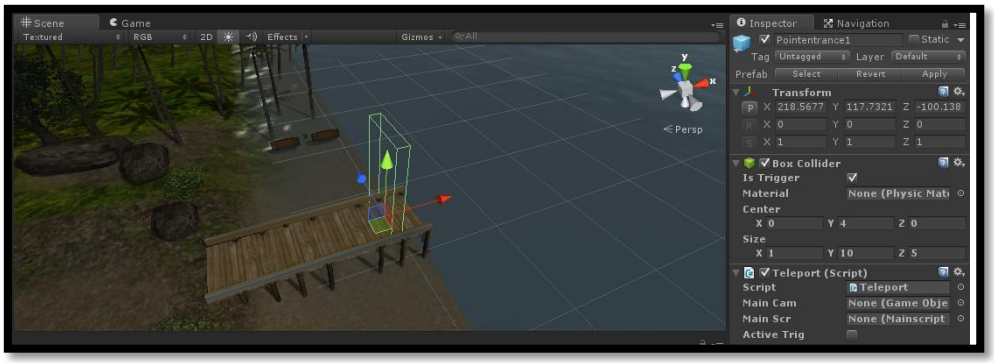


Figure 18 : BoxCollider d'une entrée/sortie

Lorsqu'un trigger est activé le script "Teleport" viens activer une fonction "placerTerrain" dans le programme principal situé dans l'objet du joueur. Un nouveau terrain est alors instancié et placé de façon à ce que l'utilisateur se situe au niveau de l'entrée du nouvel environnement.

Le script "Teleport" contenu dans les entrées/sorties de chaque terrain active la fonction "placerterrain" du script principal sur l'évènement "OnTriggerEnter*". Autrement dit, il permet, lorsque l'objet joueur se rapproche de la sortie du terrain, de demander au script principal de sélectionner un nouveau terrain et de l'instancier. La fonction "placerterrain" place alors l'environnement sous le personnage et détruit le terrain précédent. Elle compare pour cela les coordonnées du joueur, l'origine du futur terrain à instancier et la position de l'entrée de ce dernier (position du joueur + origine du terrain – entrée du terrain). Ensuite, elle instancie le terrain en prenant soin de garder ce terrain en mémoire grâce à la syntaxe suivante :

```
//Instantiation
NewTerrain = Instantiate(Terrains[idx], PositionTerrain, Quaternion.identity) as GameObject;
```

"NewTerrain" est donc devenu la référence vers le nouvel environnement instancié. L'erreur serait alors de travailler sur le prefab* et non sur ce nouvel objet. Dans le cas du script "placerterrain" nous avons, une fois le terrain instancié, retrouvés tous les objets et les scripts mais cette fois en passant par "NewTerrain" et non par la liste de prefab*.

Nous avons également créé la variable "ExTerrain" pour mettre en mémoire l'ancien terrain instancié. A la fin de notre fonction "placerterrain", nous avons donc les instructions suivantes :

```
if (ExTerrain != null) { Destroy(ExTerrain); }
ExTerrain = NewTerrain;
```

A la fin de notre fonction, l'ancien terrain est détruit puis c'est le terrain que nous venons d'instancier que l'on définit comme ancien terrain.

Une fois le terrain instancié, on tourne le joueur dans la bonne direction pour qu'il puisse suivre le chemin indiqué. Pour cela, on utilise l'orientation de l'entrée sortie que nous avons toujours placé vers l'intérieur du terrain.

Pendant le développement, nous avons choisi de rendre impossible le retour du joueur vers les terrains précédemment instanciés. Pour cela nous avons créés une communication du script principal vers ceux des entrées/sorties. Ce dernier n'a alors plus qu'à désactiver et activer les scripts adéquats pour choisir quel sera le parcours du joueur.

Nous n'avons pas oublié non plus de gérer le niveau de l'eau, la densité du brouillard et la skybox* en les stockant dans un script dans chaque prefab* de zone. Ceci alourdi le code, mais ces détails sont très importants pour l'ambiance et l'immersion (ci-dessous).

Figure 19 : Environnements avec et sans brouillard

On remarque alors que l'une des particularités du script principal est qu'il soit composé que d'une fonction publique qui ne sera appelée qu'à partir d'autres scripts. En d'autres termes, ce script ne fait rien à lui seul.

4. Réalisation de la zone de tutoriel et des menus

- Zone de tutoriel :

A la demande du cahier des charges, nous avons créé une zone de tutoriel pour permettre au joueur de prendre en main le système avant de commencer l'expérience. Pour cela, nous avons repris et aménagé à nos besoins une zone de la carte "Paradise Island" proposée dans l'Asset Store d'Unity3D.

Nous avons alors mis un script "M1Tutoriel" dans la camera principale. Nous avons voulu créer avec ce programme une suite de tâches que le joueur doit effectuer pour pouvoir passer au test. Nous avons alors utilisé une variable d'avancement du tutoriel ainsi qu'un switch case. En effet, Unity3D fonctionne grâce à un système de mises à jour successives. Il est donc impossible de procéder par un algorithme procédural classique. Ainsi, lorsque le joueur réussit une tâche on incrémente simplement une variable dans un script. Il ne reste plus qu'à donner au joueur des objectifs différents en fonction de la valeur de cette variable.

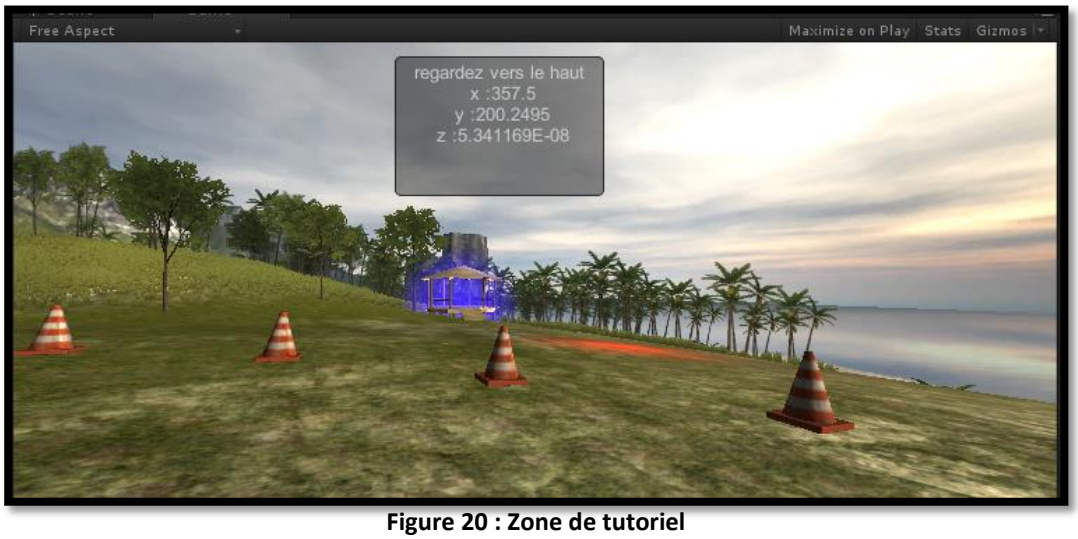


Figure 20 : Zone de tutoriel

Le tutoriel que nous avons créé consiste d'abord à faire quelques mouvements de caméra. Puis ensuite, nous guidons le joueur vers une cabane et nous lui demandons de pousser un certain nombre de plots dans une zone orange. Pour cela nous avons créé un script qui détruit le plot et incrémente une variable dans le programme principal lorsque qu'un plot se retrouve dans la zone orange.

Nous avons donc créé divers scripts de façon à gérer l'entrée du joueur et des plots dans les zones de déclenchements. Il a été nécessaire de gérer la force exercée par le personnage lorsqu'il pousse les plots. Enfin, nous avons écrit un code pour générer un bruit réaliste quand les plots s'entrechoquent.

Lorsque le tutoriel est terminé, nous demandons au joueur de retourner dans la cabane. Une fois arrivé, le script du tutoriel appelle la fonction "placerterrain" situé dans le script principal de l'application. Tous les scripts alors liés au tutoriel s'autodétruisent avant de commencer l'expérience.

- Menus contextuels :

Pour créer un menu dynamique, nous avons commencé par créer une nouvelle scène. Nous y avons ajouté des "gameObjects*" de type GUIText (texte pour l'interface utilisateur) ainsi qu'un cube faisant office de fond pour notre menu. Pour que cela fonctionne il ne faut pas oublier de mettre un composant de type GUILayer (lecteur d'interface utilisateur) sur la camera principale.

Figure 21 : Menu avec objets GUI

Nous avons alors créé plusieurs scripts permettant entre autres de changer la couleur et la taille de la police quand le joueur passe la souris sur le texte grâce aux évenements "OnMouseEnter" et "OnMouseExit". Nous également écrit un autre script dans lequel on effectue l'action appropriée en fonction d'un objet de type string si le joueur clique avec l'évenement "OnMouseUp" (Annexe A). Cet objet correspont soit au nom de la scène soit à la chaîne « Quitter ». Il faut en revanche que toutes les scènes soient ajoutées dans le menu « Build* Settings » de Unity3D pour que cela fonctionne.

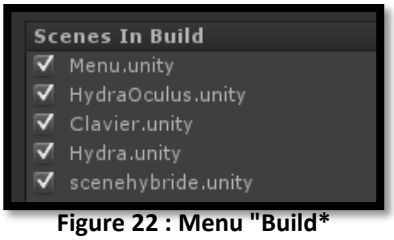


Figure 22 : Menu "Build* Settings" d'Unity3D

5. Difficultés rencontrées et solutions apportées

- **Difficulté technique :**

La première des difficultés est de bien nommer chaque composant de façon à faciliter la compréhension par d'autres personnes qui pourrait éventuellement reprendre le projet. Pour cela, il faut être le plus explicite possible dans les noms des gameobjects*, des variables et dans les commentaires. Il faut également adapter l'application afin que les changements de nom n'entraînent pas de bug. Par exemple dans les scripts, l'utilisation de tags* plutôt que les noms pour retrouver des gameobjects*. En effet, si une personne travaille sur le projet après nous si elle change les noms, le programme ne fonctionnera plus correctement.

Autre point important, il est préférable de bien organiser l'architecture de son projet dans le dossier Assets. De façon à pouvoir facilement retrouver des scripts et les réutiliser d'un projet à l'autre. Pour cela, nous avons nos scripts, nos prefabs* et nos gameobjects* par fonctionnalité.

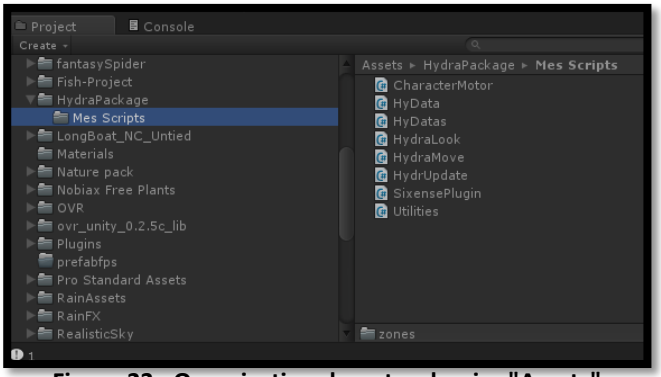


Figure 23 : Organisation de notre dossier "Assets"

Tout ce qui sert à la gestion des terrains dans un dossier appelé zones, tout ce qui concerne l'implémentation de l'Hydra dans un dossier appelé HydraPackage. Il faut en revanche éviter de reprendre l'architecture de base proposé par Unity3D qui consiste à mettre tous les scripts ensemble, toute les prefabs* ensemble, etc...

Au début du projet, il a été nécessaire d'effectuer différents essais afin apprendre le fonctionnement de la librairie d'Unity3D. Nous nous sommes également servis des forums d'aide officiels car certaines fonctions sont assez exigeantes en termes de syntaxe. D'autres appels de la librairie Unity3D possèdent de nombreuses surcharges fonctionnant plus ou moins bien selon les cas de figure notamment pour effectuer des interactions entre différents objets et scripts.

Ci-dessous, la meilleure méthode pour aller chercher un script "MainScr" contenu dans l'objet "MainCam" pour accéder à ses méthodes :

```
void Start () {
    //accès au script mainscript2
    MainCam = GameObject.FindGameObjectWithTag("Player") as GameObject;
    MainScr = (Mainscript2)MainCam.GetComponent(typeof(Mainscript2));
}
```

Comme on peut le voir, on utilise la fonction Unity3D "FindGameObjectWithTag" pour aller chercher les objets en fonction du tag* plutôt qu'en fonction de leur nom. Ceci de façon à ce qu'un changement de nom de l'objet en question n'influe pas sur le bon fonctionnement de l'application. Le projet étant susceptible d'être repris par d'autres personnes, il est préférable de prendre ses précautions. Pour aller chercher le script en lui-même, on utilise la fonction "GetComponent" avec la fonction "typeof" du C#. Cette dernière permet de renvoyer le type d'un objet. Grâce à la syntaxe montrée précédemment, la fonction "GetComponent" va alors rechercher tous les objets du type indiqué par "typeof". Cette méthode est de loin la plus fiable pour effectuer des communications entre scripts en C#.

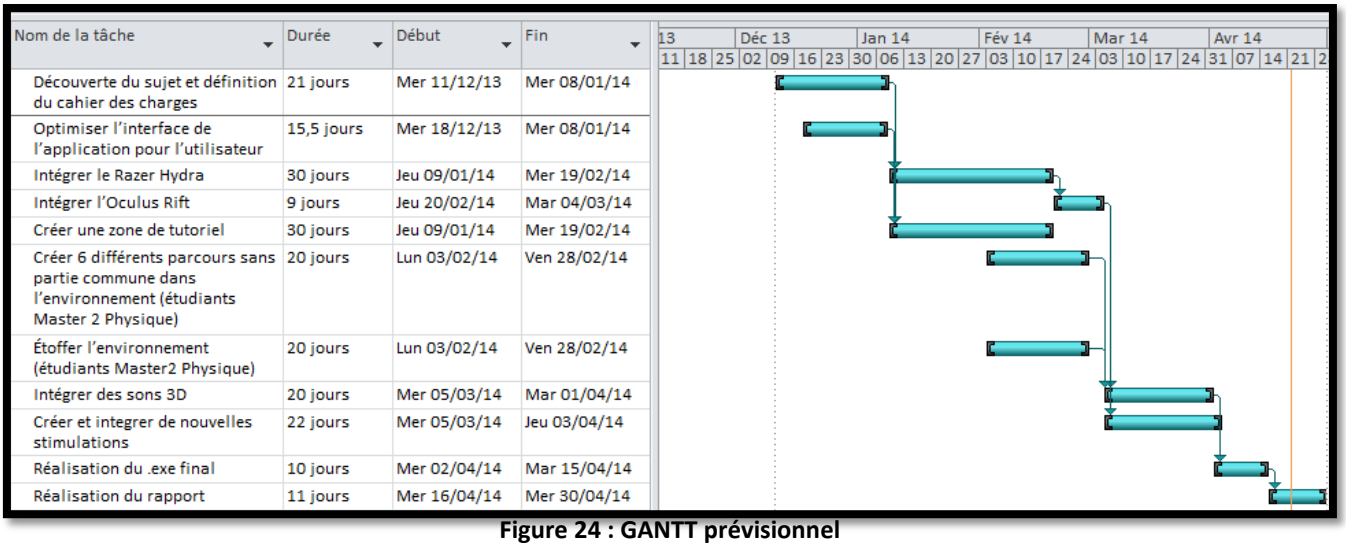
- **Difficulté humaine :**

Au cours de ce projet, nous avons remarqué l'importance de la communication. En effet, un autre groupe travaillait sur le même projet. Ce groupe s'occupait de la modélisation. Nous avons établi un cahier des charges des terrains qu'il devait créer lors d'une réunion avec eux. Néanmoins nous avons plus tard remarqué que nous n'avions pas été assez précis lors des réunions. Nous nous sommes retrouvés face à quelques problèmes : notamment des endroits du terrain où le joueur pouvait potentiellement rester bloqué. Par exemple, nous avons remarqué qu'il était impossible de ressortir de la rivière de la zone correspondant au bonheur en paramétrant un saut et une vitesse de déplacement réaliste. Autre exemple, sur le terrain associé à l'émotion de tristesse, si le joueur sort du chemin prévu, il lui est impossible d'y retourner. Ce problème est assez embêtant car il pourrait mettre un terme à l'expérience en cours.

En réaction à cette situation, nous avons utilisé le site internet "Doodle" pour pouvoir trouver des créneaux de réunions assurant la présence de toutes les personnes en lien avec le projet afin de corriger l'ensemble des problèmes évoqués précédemment.

V. GESTION DE PROJET

Comme nous l'avons présenté dans notre partie de répartition des tâches voici notre GANTT prévisionnel :

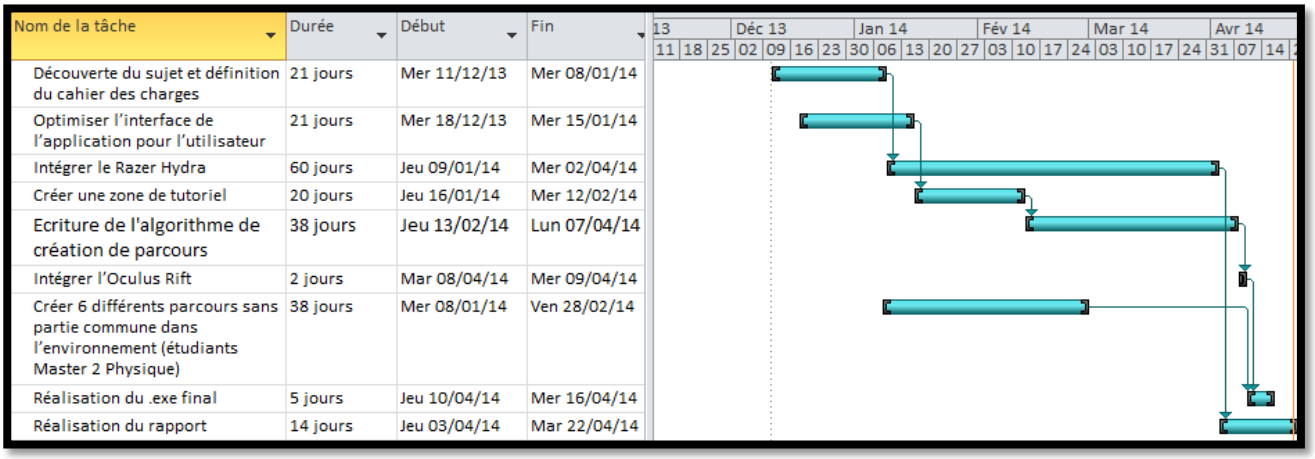


Nom de la tâche	Durée	Début	Fin
Découverte du sujet et définition du cahier des charges	21 jours	Mer 11/12/13	Mer 08/01/14
Optimiser l'interface de l'application pour l'utilisateur	15,5 jours	Mer 18/12/13	Mer 08/01/14
Intégrer le Razer Hydra	30 jours	Jeu 09/01/14	Mer 19/02/14
Intégrer l'Oculus Rift	9 jours	Jeu 20/02/14	Mar 04/03/14
Créer une zone de tutoriel	30 jours	Jeu 09/01/14	Mer 19/02/14
Créer 6 différents parcours sans partie commune dans l'environnement (étudiants Master 2 Physique)	20 jours	Lun 03/02/14	Ven 28/02/14
Étoffer l'environnement (étudiants Master2 Physique)	20 jours	Lun 03/02/14	Ven 28/02/14
Intégrer des sons 3D	20 jours	Mer 05/03/14	Mar 01/04/14
Créer et integrer de nouvelles stimulations	22 jours	Mer 05/03/14	Jeu 03/04/14
Réalisation du .exe final	10 jours	Mer 02/04/14	Mar 15/04/14
Réalisation du rapport	11 jours	Mer 16/04/14	Mer 30/04/14

Figure 24 : GANTT prévisionnel

Une fois notre projet terminé nous avons refait un GANTT afin de pouvoir faire la comparaison entre les deux. Voici donc notre diagramme final :

On observe ainsi que plusieurs tâches on été supprimées et que certaines ont durées plus longtemps que prévu initialement.

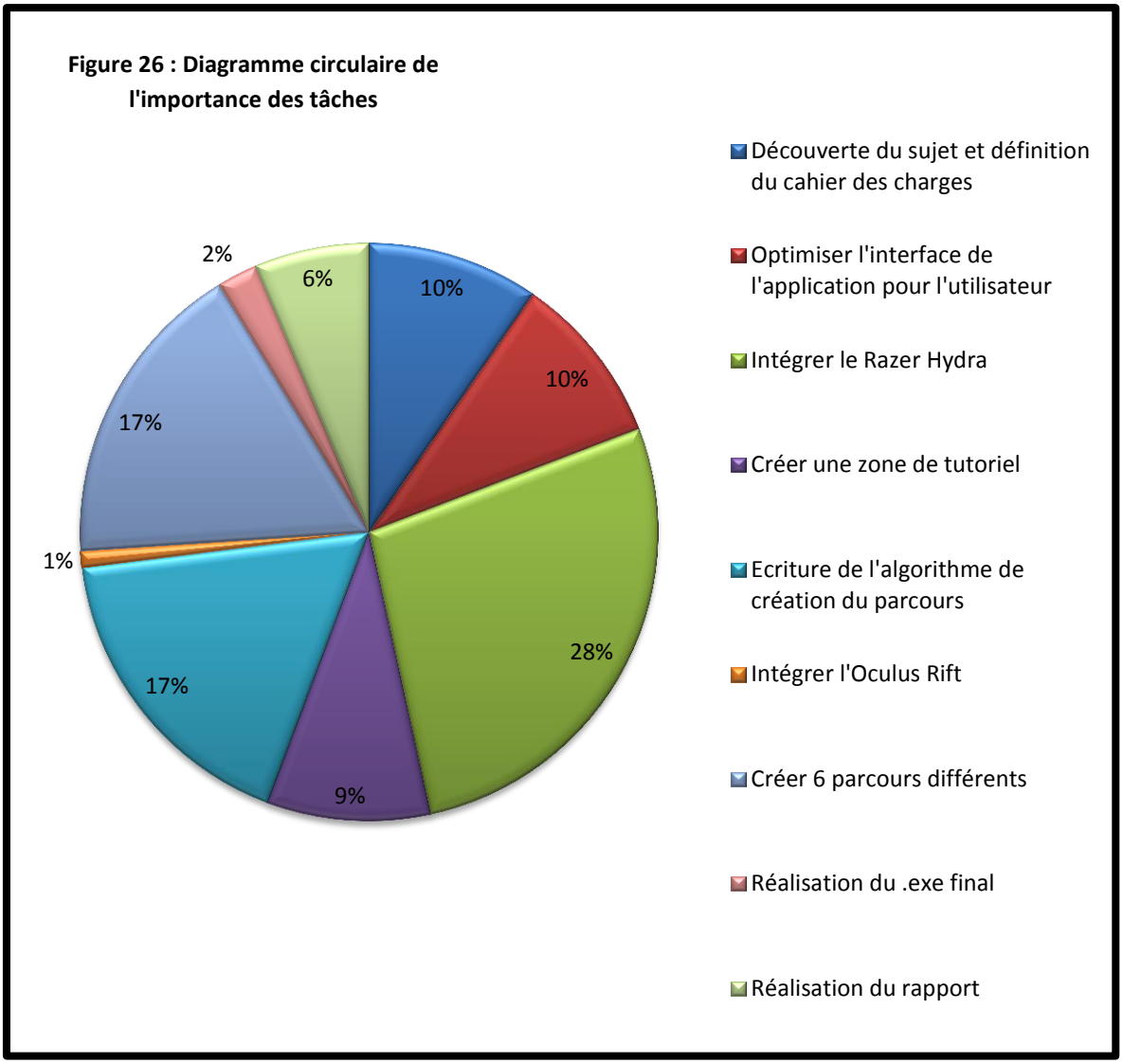


Nom de la tâche	Durée	Début	Fin
Découverte du sujet et définition du cahier des charges	21 jours	Mer 11/12/13	Mer 08/01/14
Optimiser l'interface de l'application pour l'utilisateur	21 jours	Mer 18/12/13	Mer 15/01/14
Intégrer le Razer Hydra	60 jours	Jeu 09/01/14	Mer 02/04/14
Créer une zone de tutoriel	20 jours	Jeu 16/01/14	Mer 12/02/14
Ecriture de l'algorithme de création de parcours	38 jours	Jeu 13/02/14	Lun 07/04/14
Intégrer l'Oculus Rift	2 jours	Mar 08/04/14	Mer 09/04/14
Créer 6 différents parcours sans partie commune dans l'environnement (étudiants Master 2 Physique)	38 jours	Mer 08/01/14	Ven 28/02/14
Réalisation du .exe final	5 jours	Jeu 10/04/14	Mer 16/04/14
Réalisation du rapport	14 jours	Jeu 03/04/14	Mar 22/04/14

Figure 25 : GANTT final

Nous avons aussi réalisé un diagramme circulaire représentant le pourcentage de chaque tâche par rapport à l'intégralité du projet :

Figure 26 : Diagramme circulaire de l'importance des tâches



VI. CONCLUSION

L'objectif de ce projet était de mettre en place une application immersive de réalité virtuelle dans le but de pouvoir évaluer les émotions émises par l'utilisateur lors de son évolution au sein des environnements développés.

Sur le plan technique, ce projet nous a apporté une expérience de travail de groupe mais aussi en équipe avec la réalisation des environnements par une autre équipe. La communication est bien entendu la clé du bon déroulement d'un projet que ce soit avec nos responsables de projets, avec une seconde équipe et entre nous.

Lors de ce projet, nous avons pu mettre en application les connaissances acquises à l'ISTIA dans le domaine de la programmation ainsi que la réalisation d'environnement sous Unity3D.

Sur le plan humain, cette expérience fut enrichissante. Il y a eu un réel échange avec M. PAYEN, consultant Unity3D sur ce projet, qui nous a fait part de son expérience, donné des conseils et accordé du temps si nécessaire. De plus Mme FOLOPPE a su nous indiquer ces exigences pour l'application d'un point de vue "psychologique" afin de déterminer clairement le cahier des charges.

Glossaire

Audioclip : Classe Unity3d pour les objets sonores. On régler différents paramètre comme par exemple sa vitesse de lecture (Audioclip.pitch), son volume (Audioclip.volume) ou bien le délai après la lecture de celui pour qu'un autre Audioclip puisse être lu (Audioclip.delay).

Build : En développement informatique, le build est le résultat de la compilation et de l'assemblage du programme que l'on a créé. Il s'agit donc de l'application telle qu'elle sera distribuée aux utilisateurs.

Event Handler : objet qui a déclenché un évènement sur lequel on peut éventuellement agir dans les scripts.

Frame : Unity3d fonctionne grâce à un système de mises à jour successives où l'ensemble de ce qui est affiché à l'écran est recalculé. Dans les script, cela se traduit par l'utilisation de fonction comme Update() ou FixedUpdate(). La différence entre les deux est que le temps entre deux exécutions de la fonction FixedUpdate() est toujours constant. La fonction Update(), quant à elle, est appelée à chaque frame. Le temps entre chaque frame pouvant varier, le temps entre deux appels de la fonction Update() peut varier également.

GameObject : objet se situant dans une scène 3d. Il peut contenir des objets filles dont les coordonnées dépendront alors de ses coordonnées.

OnControllerColliderHit() : fonction qui s'active lorsque l'objet mère dans lequel le script est situé entre en collision avec un autre objet qui fera office d'event handler de cette fonction.

OnTriggerEnter() : fonction qui s'active lorsque l'objet mère dans lequel le script est situé entre dans une zone de déclenchement.

Prefab : Déclencheurs (ou trigger) : zone activant notamment l'évènement OnTriggerEnter() dans les scripts de l'objet qui contient le déclencheur.

Skybox : C'est un component de type material qui constitue l'arrière-plan et le ciel du monde 3D.

Tag : Un tag est un objet de type string qui est associé à un objet. Il a la particularité de ne pas avoir de but particulier et le programmeur peut s'en servir comme bon lui semble.

Table des figures

Bibliographie

Site internet de l'ISTIA : http://www.istia.univ-angers.fr/fr/index.html

Forums officiels d'Unity3D : http://answers.unity3d.com/questions/index.html

Site Unity3D Francophone: http://www.unity3d-france.com/

Documentation officielle d'Unity3D : http://unity3d.com/learn/documentation

Wiki Unity3D : http://wiki.unity3d.com/

OpenClassRooms (ex- site du zero) : http://fr.openclassrooms.com/

Table des annexes

Annexes :

Les chiffres clés de l'Université d'Angers en 2012 :

FORMATION

- **19117** étudiants
- **987** enseignants
- **707** personnels administratifs
- **399** diplômes
- **16** licences
- **48** licences professionnelles
- **32** masters

53 spécialités professionnelles
16 spécialités de recherche
11 spécialités pro./recherche
10 spécialités enseignement

- **4** formations à distance
- **3647** nouveaux bacheliers

RECHERCHE

- **29** équipes de recherche
- **3** services communs de recherche
- **1** collège doctoral
- **8** écoles doctorales
- **6** créations d'entreprises issues des résultats des laboratoires
- **34** familles de brevets actives
- **447** doctorants
- **88** thèses soutenues en 2011

INTERNATIONAL

- **2327** étudiants étrangers

(12% du nombre total d'étudiants)

- **113** nationalités représentées
- **115** accords bilatéraux
- **187** universités partenaires dans le cadre du programme Erasmus

VIE À L'UNIVERSITÉ

- **2500** ordinateurs (soit en moyenne 1 pour 8 étudiants)
- **98** manifestations culturelles
- **45** activités physiques, sportives et artistiques

PATRIMOINE

168 556 m2 de surfaces bâties
dont **33 959 m2** dédiés à la recherche

Déclaration d'un Singleton dans la classe Hydatas :

```
//Singleton
private static volatile Hydatas _instance;
private static object _lock = new object();
static Hydatas() {} //Stops the lock being created ahead of time if it's not necessary

public static Hydatas Instance
{
    get
    {
        if (_instance == null)
        {
            lock(_lock)
            {
                if (_instance == null) _instance = new Hydatas();
            }
        }
        return _instance;
    }
}

private Hydatas() { }
```

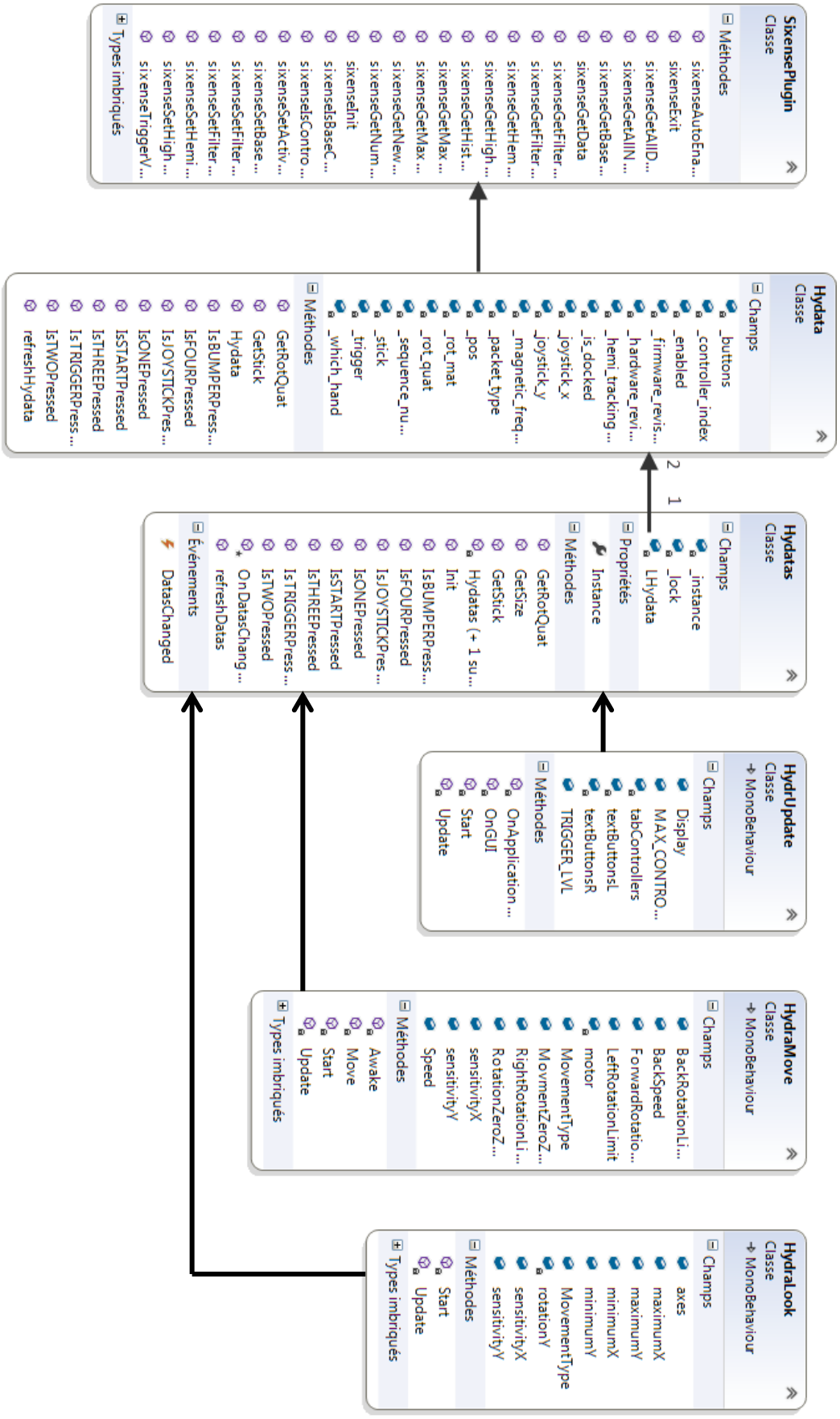
On observe une déclaration d'une instance de Hydatas. Ce sera cette instance l'unique exemplaire existant de Hydatas qui sera appelé par les autres scripts requérants les informations des contrôleurs du Razer Hydra.

La variable "lock" verrouille l'accès au constructeur de Hydatas si une instance est déjà présente, on ne peut donc pas avoir deux entités existantes simultanément.

Evènement "OnMouseUp":

```
void OnMouseUp()
{
    if (levelsuivant == "Quitter")
    {
        Application.Quit();
    }
    else
    {
        Application.LoadLevel(levelsuivant);
    }
}
```

Diagramme UML des classes du Razer Hydra :



2013 - 2014

2013 - 2014

Réalisation d'une application de réalité virtuelle pour l'évaluation des émotions

Projet réalisé par: LEVILAIN Rémi & ROUSSEL Fabien
Projet encadré par : M. RICHARD Paul

Résumé :

Ce projet a pour but de réaliser une application de réalité virtuelle comprenant différents terrains évoquant chacun une émotion différente : peur, tristesse, dégoût, joie, Zen. Ces terrains sont conçus pour faire réagir l'utilisateur qui sera en immersion dans l'application grâce à l'utilisation du Razer Hydra et de l'Oculus Rift. La finalité de cette application est son utilisation lors d'expérimentation psychologique pour évaluer les réactions de l'utilisateur. Ce rapport décrit la démarche employée ainsi que l'ensemble de la conception et de la réalisation de ce projet.

Mots-Clés : Réalité virtuelle, Oculus Rift, Razer Hydra, émotions

---

Summary:

The main objective of this project is to create a reality virtual application that contains different playground. Each of them is link to one feeling: fear, sadness, disgusting, happiness and Zen. The aim of theses environments is to make the user react by the virtual immersion created with the Razer Hydra and Oculus Rift. At the end, this application will be used during psychological experimentations in order to evaluate user's reactions. This report describe the method used, the conception and the realization of this project.

Keywords: Virtual reality, Oculus Rift, Razer Hydra, feelings