Doc Technique Hochets Interactifs

CHAIDRON Jules - LI Chenyu - OUVRARD Pierre
TYURIN Stepan - YUAN Mengxin

Encadrants : GAUGNE Ronan, chercheur à l'Université de Rennes
NICOLAS Théophane, archéologiste à l'INRAP

2022-2023
Version : 1 - version fin de projet Etudes Pratique 3A 22-23

Table des matières

Résumé

Ce document est la documentation technique de l'application Hochets Interactifs, projet d'Études Pratiques de 3e année, département Informatique à l'INSA de Rennes. Cette documentation est divisées en 2 parties :
- une partie couvrant l'architecture de l'application
- une partie contenant les remarques vis-à-vis de ce projet, ce qui peut-être amélioré, etc.

1 Pré-requis techniques

Pour utiliser et éditer cette application, les logiciels suivants doivent être installés sur le poste de travail que vous utiliserez :
- Unity 2021.3.X
- Application Oculus (dernière version)
- SteamVR

L'application a été développée et testée sur les casques Oculus Quest 2 et Oculus Rift S, mais peut potentiellement fonctionner avec un casque HTC Vive.

2 Rappel de fonctionnement

Cette application a pour but de permettre à un utilisateur de manipuler un objet 3D via réalité virtuelle, et de générer du son lorsque cet objet est secoué.

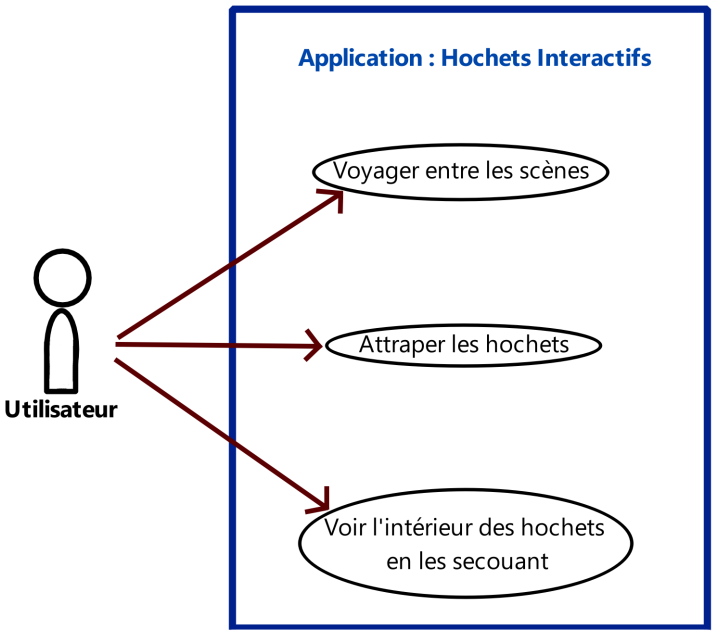
Les objets intégrés à cette application sont des hochets anciens, inaccessibles car entreposés dans des musées. Le principal objectif de cette application est de permettre à l'utilisateur d'entendre le son que ces hochets émettent.

Pour lancer l'application sans réalité virtuelle, il suffit d'ouvrir le dossier "hochet" comme projet dans Unity, ouvrir une des scènes (par exemple hochet/Assets/3INFO 22-23/Scene hub/hub.unity pour la scène Hub) en faisant un double-clic ou glisser-déposer le fichier de la scène, puis lancer la scène.

Pour ouvrir l'application en mode RV, il faut d'abord ouvrir l'application Oculus et connecter le casque (sans oublier Oculus Link pour les casques Quest). Ensuite il faut lancer SteamVR. Une fois que le casque est connecté et SteamVR est ouvert, il faut ouvrir le projet dans Unity et lancer une scène. Enfin, il suffit de mettre le casque sur la tête et l'application doit se lancer dans le casque.

Remarque : Il est possible de lancer l'application sans SteamVR avec juste Oculus mais le comportement de l'application dans ce cas n'est pas testé.

3 Architecture du projet



3.1 Résumé sur la structure du projet

Tous les fichiers concernant le projet se trouvent dans le dossier Assets. Vous y trouverez les dossiers suivants :

- *Sounds*, contient les divers sons utilisés dans l'application, notamment ceux des hochets.
- *Scripts*, avec les dossiers "3A 22-23" et "4A 09-22" qui contiennent respectivement les scripts de notre projet et du projet des 4A (version de septembre

2022) sur lequel notre projet se base.
- *Scenes*, qui contient les fichiers .unity des scènes, permettant de les ouvrir.
- *Prefabs*, qui contient les prefabs des hochets, des billes et de certains autres objets développés pour ce projet
- *"3INFO 22-23"*, qui contient les différents assets utilisés par les différentes scènes ainsi que les modèles sources des hochets (dans les dossiers des scènes correspondantes)
- *Divers*, qui contient d'autres fichiers qui sont utilisés dans l'application (certaines textures, fichiers de localisation, etc.)

3.2 Scènes

Il y a 6 scènes dans cette application, toutes se trouvent dans le dossier *Assets/Scenes/3A 2023 Scenes*.

La scène Hub est la première scène qui est chargée et dans laquelle l'utilisateur apparaît lors du lancement de l'application, elle sert à accéder à toutes les autres scènes. Il s'y trouve 4 modèles préfabriqués d'arbres, ainsi que 5 modèles 3D de hochets, en suspension au-dessus de 5 "nuages" colorés. Ces hochets et nuages sont animés à l'aide du script *Floater* (voir partie 3.6. Scripts).

Les 5 autres scènes ont été conçues pour s'approcher le plus possible d'une ambiance correspondant à la location et l'époque d'origine du hochet qui lui est associé. Toutes possèdent un asset de type Terrain, sur lequel ont été placés différents préfabriqués et modèles 3D. Dans chacune se trouve un hochet qui est lui même un modèle prefab (voir partie 3.3.Prefabs).

Chacune de ces scènes est accessible par téléportation depuis la scène hub en interagissant avec le hochet qui lui est associé dans la scène Hub. Et chaque scène a un point de retour à la scène Hub sous la forme d'un objet Porte, qui en téléporte l'utilisateur dans la scène Hub. Cette téléportation est assurée par le script *SceneChanger* (voir partie 3.6. Scripts).

3.3 Prefabs

La plupart des prefab de décor utilisés dans les scènes sont récupérés sur l'Unity Assets Store, et sont stockés dans le dossier *Assets/Prefabs du projet*.

Pour ce qui est des hochets, ce sont des prefabs créés pour cette application. Il s'agit de modèles 3D à qui nous avons fourni des box colliders, avons attaché un matériau spécifique, un son spécifique et un certain nombre de prefabs de type bille.

Les prefabs de type bille sont *BilleCeramique* et *hochet_bille_v2*. Ils se composent d'un modèle 3D auquel des colliders sont attachés, ainsi que du script *MakeSounds* (voir partie 3.6. Scripts).

Ces prefabs de hochet sont aussi enregistrés dans le dossier Assets/Prefabs du projet.

3.4 Modèles des hochets

Les modèles 3D des hochets ont été récupérés de différentes manières : présent comme asset dans le projet d'Etudes Pratique auquel se rattache notre application, trouvé sur SketchFab, ou encore trouvés sur les sites web des musées.

Le modèle 3D du hochet *HochetBois* à été retouché à l'aide de Blender car ce hochet ne fonctionne pas comme les autres : les autres reposants sur la présence d'au moins une bille à l'intérieur, le hochet en bois est une cloche, dont le battant

cylindrique vient taper les bords lorsque secoué. Le modèle 3D avec lequel nous étions parti avait le battant fusionné au reste du hochet, et ne pouvait pas se déplacer. Nous avons donc choisi de découper le battant pour le remplacer par un autre plus mobile que nous souhaitions attacher au reste du modèle afin de pouvoir rendre le modèle utilisable dans notre projet.

3.5 Sons

Les différents sons spécifiques pour les différents hochets ont étés soit enregistrés par nos soins (comme pour le hochet *HochetOsierHomeMade*) soit récupérés sur des projets existants (comme pour le hochet *HochetSpherique*, dont les sons proviennent du projet auquel cette application vient se greffer). Ils sont sous format .wav qu'Unity sait utiliser. Tous les sons sont enregistrés dans le dossier *Assets/Sounds* du projet.

3.6 Scripts

Notre application repose sur 4 scripts permettant de réaliser tout ce qui a été détaillé précédemment. Ces scripts sont :

Floater : ce script est utilisé dans la scène Hub pour générer le mouvement des hochets. Le script à trois paramètres : degrés par seconde, amplitude et fréquence. Le script crée d'abord deux variables, une pour stocker la position actuelle de l'objet et *posOffset* pour stocker la position de l'objet après chaque frame du mouvement. La méthode start() rend d'abord la variable *posOffset* égale à la position actuelle de l'objet. La méthode update() utilise la méthode *Rotate* d'Unity pour faire pivoter l'axe des y, et une fonction sinusoïdale pour faire flotter l'axe des y vers le haut et vers le bas. Enfin la méthode update() vient stocker les coordonnées calculées dans la variable *tempPos*. Le script se termine en modifiant la position actuelle avec celle stockée dans *tempPos*.

SceneChanger : Ce script se charge de transporter l'utilisateur d'une scène à l'autre. Ce script doit être attaché sur un objet interactif, et le nom de la scène de destination doit lui être rentré. Afin de mettre en œuvre la possibilité de changer de scène après avoir interagi avec un objet, nous devons d'abord savoir que le joueur interagit avec l'objet. Le script utilise la classe *XRSimpleInterable* pour savoir si un objet est en cours d'interaction. Dans la méthode start(), il détermine d'abord si l'objet est *interable* et il signale une erreur si ce n'est pas le cas. S'il l'est, il ajoute un Listener de type *OnselectEntered*, et, s'il reçoit un événement *OnselectEntered(SelectEnterEventArgs args)*, il appelle la méthode *LoadScene* de la classe *SceneManager* d'Unity pour charger la scène de destination.

MakeSounds : Ce script se charge de transformer les collisions entre les billes à l'intérieur des hochets en sons. Attaché aux modèles prefabs des différentes billes de l'application, ce script détecte toute collision rencontrée par la bille, en récupère la magnitude grâce à la méthode collision.relativeVelocity.magnitude fournie par Unity, et joue un son en modifiant le volume de ce son en fonction de cette magnitude (voir lignes 58 à 61). Pour définir quel son ce script doit jouer, il récupère dans la méthode Start() le son d'un des composants de l'objet dont il est enfant (ligne 33), et le stocke dans une variable de type *AudioSource* pour ne pas avoir à le recharger à chaque collision. Pour s'assurer

que la collision est assez importante pour générer du son de manière réaliste, nous avons ajouté une variable threshold, de type float, qui sert de seuil de magnitude au-delà duquel nous autorisons le son à être produit (voir test à la ligne 59). La méthode Update() (lignes 39 à 55) sert à vérifier si le son est autorisé à être joué à un niveau plus haut dans l'application, autrement dit si le son est déclaré comme coupé (Mute) ou non.

Transparence : Ce script rend transparent la texture extérieure du hochet. Il doit être attaché comme component au gameobject racine du hochet (celui qui contient le mesh et le material représentant l'extérieur du hochet). Il est possible de choisir la force nécessaire pour déclencher la transparence lorsque le hochet est secoué grâce au paramètre *shakeThreshold* (0.1 par défaut). Si la magnitude de rotation du hochet est supérieure à *shakeThreshold*, le script change le channel alpha du material du hochet par le taux de transparence (transparency, avec la valeur 0 étant une transparence totale et 1 aucune transparence, avec 0 par défaut), puis change le mode du rendu du material par Transparent ou Opaque. Si le hochet n'est plus secoué pendant 3 secondes, le hochet cesse d'être transparent.
Remarque : Ce script a un comportement correct uniquement dans l'éditeur Unity. Après un build, la transparence a des effets indésirables. La raison exacte est inconnue mais cela est probablement dû à la différence de gestion de shaders et de lumières entre l'éditeur et le build.

4 Asset EasyColliderEditor

Pour accéder à l'éditeur de colliders il faut choisir le gameobject du hochet qui contient le mesh, puis ouvrir *Window -> EasyColliderEditor*. Il permet de créer plus facilement des box colliders ainsi que de générer un ensemble de mesh colliders qui peuvent potentiellement remplacer les box colliders sans être convexes.

Pour plus de détails, voir :
`https://assetstore.unity.com/packages/tools/level-design/easy-collider-editor-67880`

Remarque : Pour faciliter le travail avec *EasyColliderEditer*, il est recommandé de temporairement remplacer le mesh du hochet par un mesh avec un nombre de polygones fortement réduit (<1000) pour faciliter la sélection des vertices et permettre une génération de mesh colliders relativement rapide. Il est possible de le faire en appliquant par exemple le modifier "decimate" sur le mesh dans le logiciel Blender.

5 Remarques générales

5.1 Améliorations possibles

Le script *MakeSounds* nous permet d'émettre les sons d'une manière "réaliste" en fonction de la magnitude de la collision entre les billes et les parois internes des hochets, mais c'est pour l'instant toujours le même son qui sera joué par un même hochet. Nous n'avons pas réussi avec le temps qui nous était donné de faire qu'il puisse jouer plusieurs sons différents pour un même hochet, pour cela, il faudrait réussir à attacher plusieurs sons à un même

prefab, et réussir à tirer au hasard parmis ces sons lequel sera joué lors d'une collision. Une piste que nous avons essayé d'approfondir pour cela fut de créer plusieurs GameObject attachés au prefab d'un hochet comme objet enfant, auxquels nous aurions attaché un son chacun, mais nous n'avons pas réussi à tirer au hasard parmis eux lequel fournirait le son au script *MakeSounds*.

Les colliders des différents hochets, bien que fonctionnels, peuvent toujours être améliorés pour éviter certains cas ou les billes s'y retrouvent coincées, ou encore pour se rapprocher davantage de la forme du hochet. Notamment les colliders générés pour le hochet en courge ne laissent pas suffisamment de liberté de déplacement à la bille à l'intérieur du hochet.

Les modèles 3D utilisés dans certaines scènes (notamment les bâtiments dans la scene du hochet polonais) peuvent être excessivement lourds en terme de mémoire, avec beaucoup de polygones à charger, sans apporter beaucoup de qualité. Ils peuvent être réduits pour diminuer le temps de chargement des scènes.

Dans d'autres cas les modèles sont à l'inverse de qualité trop basse et peuvent être remplacés par des modèles de plus haute qualité (notamment dans la scène courge)

5.2 Quelques conseils de développement

Le temps de chargement des scènes dépend directement de la surface de la scène / du paysage. Pour un temps de chargement proche de l'instantané, il est recommandé de ne pas dépasser 50x50.

En chargeant une scène en utilisant *SceneManager.LoadScene*, la lumière dans l'éditeur peut se comporter de façon étrange (absence de lumière ambiante par exemple). Le problème disparaît en lançant un .exe de l'application après un build, mais il est possible de remédier à ce problème. Pour cela il faut ouvrir Window -> Rendering -> Lighting et dans Baked Lightmaps choisir un lightmap autre que None. Il faut ensuite modifier quelque chose d'autre dans la scène (par exemple dans la même fenêtre dans l'onglet Scene, remplacer Lightning Settings Asset par un autre, puis remettre l'ancien) pour que Unity remarque des changements et permet d'enregistrer les changements de luminosité dans la scène.