

# Master of Science HES-SO in Engineering

Orientation : Technologies de l'information et de la  
communication (TIC)

Version Générique  
d'une Solution de Formation en AR

Fait par

**Quentin Forestier**

Sous la direction de  
Prof. Yassin Aziz, REKIK  
Dans le laboratoire/institut de/ à l'école HEPIA

Expert externe  
Stéphane Malandain

Lieu, HES-SO//Master, 2024

Accepté par la HES-SO//Master (Suisse, Lausanne) sur proposition de

Prof. Yassin Aziz REKIK, conseiller de travail de Master

Prof. Stéphane Malandain, Expert principal

Lausanne, le 9 février 2024

# TABLE DES MATIÈRES

<b>TABLE DES MATIÈRES .....</b>	<b>III</b>
<b>REMERCIEMENTS .....</b>	<b>V</b>
<b>NOMENCLATURE .....</b>	<b>VI</b>
<b>RÉSUMÉ .....</b>	<b>VII</b>
<b>INTRODUCTION .....</b>	<b>1</b>
<b>1. ANALYSE .....</b>	<b>5</b>
1.1. ANALYSE DE LA SOLUTION EXISTANTE .....	5
1.1.1. <i>Analyse de la structure et des communications entre les applications</i> .....	5
1.1.2. <i>Analyse étape par étape d'une simulation et des fonctionnalités</i> .....	6
1.1.3. <i>Analyse détaillée de la couche réseau</i> .....	10
1.1.4. <i>Analyse détaillée de la partie scénario et indices</i> .....	12
1.1.5. <i>Analyse détaillée de la partie monitoring et feedback</i> .....	13
1.2. SPÉCIFICATION DES BESOINS .....	14
<b>2. RECHERCHES ET CONCEPTIONS .....</b>	<b>17</b>
2.1. RECHERCHE ET CHOIX DU CASQUE DE RÉALITÉ AUGMENTÉE .....	17
2.2. RECHERCHE ET CONCEPTION DE LA COUCHE RÉSEAU .....	18
2.2.1. <i>Recherche sur les méthodes existantes</i> .....	19
2.2.2. <i>Conception de la structure du réseau</i> .....	20
2.3. RÉFLEXION ET CONCEPTION DE LA COUCHE SCÉNARIO .....	23
2.4. CONCEPTION DE LA COUCHE GESTION INTELLIGENTE DU JEU .....	29
2.4.1. <i>Conception des objets permettant le système d'impact à deux états</i> .....	30
2.5. CONCEPTION DE LA COUCHE INDICES ET PERTURBATEURS .....	31
2.6. CONCEPTION DES INTERACTIONS EN RÉALITÉ AUGMENTÉE .....	32
2.7. CONCEPTION DE LA COUCHE MONITORING .....	33
2.8. CONCEPTION DE LA COUCHE FEEDBACK .....	35
2.9. CONCEPTION DE L'ANCRAGE DU MONDE VIRTUEL .....	35
<b>3. IMPLÉMENTATION ET PROBLÈMES RENCONTRÉS .....</b>	<b>36</b>
3.1. IMPLÉMENTATION DE LA COUCHE RÉSEAU .....	36
3.2. IMPLEMENTATION DE LA COUCHE SCENARIO .....	41
3.3. IMPLEMENTATION DE LA COUCHE GESTION INTELLIGENTE DE SCENARIO .....	44
3.4. IMPLÉMENTATION DE LA COUCHE INDICES ET PERTURBATEURS .....	49
3.5. IMPLÉMENTATION DES INTERACTIONS EN RÉALITÉ AUGMENTÉE .....	53
3.6. IMPLEMENTATION DE LA COUCHE MONITORING .....	59
3.7. IMPLEMENTATION DE LA COUCHE FEEDBACK .....	63
3.8. IMPLEMENTATION DE L'ANCRAGE DU MONDE VIRTUEL .....	66
3.9. BILAN ET RESULTATS .....	68

<b>4. CONCLUSION .....</b>	<b>71</b>
<b>5. BIBLIOGRAPHIE .....</b>	<b>72</b>
<b>6. LISTE DES FIGURES .....</b>	<b>73</b>
<b>7. ANNEXES .....</b>	<b>76</b>
<i>Guide de développement d'un Scénario .....</i>	<i>77</i>

## **REMERCIEMENTS**

Je tiens à exprimer ma gratitude envers le professeur Yassin Aziz Rekik pour m'avoir offert l'opportunité de travailler sur ce projet et pour son soutien constant tout au long du processus.

Je souhaite également exprimer mes remerciements à ma copine pour avoir consacré du temps à tester la partie multijoueur du projet et pour son soutien inconditionnel tout au long de cette période.

## **NOMENCLATURE**

AR	Augmented Reality (Réalité augmentée)
HOST	Human and Organizational Skills Training
HUG	Hôpitaux universitaire genevois
JSON	JavaScript Object Notation
RPC	Remote Procedure Call, une méthode appelée sur un hôte distant
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VR	Virtual Reality (Réalité virtuelle)

## RÉSUMÉ

Les employés des HUG participent à des séances de renforcement d'équipe pour améliorer la communication, la collaboration et la gestion du stress. Certaines de ces sessions sont menées via le projet HOST, une plateforme de réalité augmentée proposant divers scénarios à accomplir. Cette solution plonge les participants dans un monde mêlant réalité et virtuel, les confrontant à des énigmes à résoudre. Un maître de jeu, grâce à une interface dédiée, suit leur progression et leur apporte son aide. Cependant, un problème majeur avec la solution existante est son manque de flexibilité pour l'ajout de nouveaux scénarios, énigmes et fonctionnalités. De plus, les utilisateurs ont fait part de leurs retours, notamment en ce qui concerne la charge de travail du maître de jeu qui était entravée par ses multiples responsabilités, notamment la prise de notes pendant la simulation.

Dans le cadre de ce travail, nous avons dû analyser ces retours et trouver des moyens de gérer au mieux l'intégration de nouveaux scénarios. Une solution a été développée dans le but de simplifier l'ajout de ces nouveaux éléments et de réduire la charge de travail du maître de jeu. De nouvelles fonctionnalités ont été intégrées, notamment la synchronisation des objets et un gestionnaire de scénario intelligent, afin d'améliorer l'immersion dans la simulation et d'assister le maître de jeu dans ses tâches.

## INTRODUCTION

Lorsqu'on est en milieu professionnel, il est fréquent de faire face à des problèmes de communication, souvent attribuables à des facteurs tels que le stress, la charge de travail excessive ou encore des différences culturelles. Un aspect crucial à considérer est la confiance et la collaboration, regroupés sous le terme générique de « Team Building ». L'objectif est d'améliorer la dynamique d'équipe en permettant à chacun de trouver sa place, de mieux comprendre les modes de fonctionnement des autres membres et d'atteindre un équilibre au sein de l'équipe.

Diverses approches peuvent être utilisées pour résoudre les problèmes de cohésion d'équipe, mais une solution qui revient fréquemment est l'utilisation des « Serious Games ». Ces jeux ont des objectifs informatifs, pédagogiques, idéologiques, tout en restant ludiques pour rendre la dimension sérieuse plus attrayante. Ils peuvent se présenter sous différentes formes, tels que les jeux de société, les jeux de rôle ou les jeux vidéo, tous définis par un contexte, un objectif et des règles spécifiques.

La clarté dans l'expression des objectifs et des règles d'un jeu est déterminante pour le succès d'un « Serious Game ». Ces éléments doivent être motivants tout en suscitant la curiosité et la réflexion.

Au quotidien, ces jeux sont utilisés dans divers contextes, notamment dans les hôpitaux, où des jeux sérieux sont conçus pour aider les enfants et même les adultes à effectuer leurs exercices de rééducation de manière plus engageante.



Figure 1 Jeu sérieux pour la rééducation (MindMaze, 2019)

Les « Escape Games » sont une autre forme de jeu qui a gagné en popularité ces dernières années. Pour rappel, un « Escape Game » implique de placer plusieurs participants dans une pièce close où ils doivent s'échapper en résolvant une série d'énigmes. De nombreuses entreprises ont adopté l'utilisation de ces salles pour renforcer la cohésion d'équipe, souvent dans le cadre d'activités de « Team Building ».



Figure 2 Exemple d'une salle d'Escape Game (Lade Tech, s.d.)

Depuis 2003, les équipes des Hôpitaux Universitaires Genevois participent à des Serious Games organisés par l'association "Ensemble", visant à améliorer la communication, la collaboration, la gestion du stress et la solidarité entre les collaborateurs en cas d'urgence médicale. L'un de ces Serious Games se déroule dans un scénario simulant un accouchement en avion. Les organisateurs recréent un environnement d'avion avec des tables, des chaises et des images de hublot pour favoriser l'immersion. Des acteurs, tels que la femme enceinte, son mari et le capitaine de bord, sont également présents. Les participants doivent résoudre des énigmes pour obtenir les outils nécessaires à la gestion de cette crise, mais le retour sur la prestation était difficile en raison de l'immersion limitée.

C'est dans ce contexte que la première version d'HOST a été développée. L'idée était de reprendre le scénario existant et d'y intégrer des décors et des éléments virtuels, en utilisant une application de réalité augmentée. Pendant la simulation, certains participants portent des casques de réalité augmentée, tandis que d'autres n'en portent pas, ajoutant une complexité nécessitant une communication efficace malgré la disparité des informations disponibles.

Un maître de jeu guide les participants à travers l'histoire du scénario. Après la réunion préparatoire, les joueurs se retrouvent plongés dans la salle, travaillant ensemble pour atteindre leur objectif. Ils doivent collaborer pour résoudre des énigmes, qu'elles soient réelles ou virtuelles, les informations nécessaires étant disponibles dans les deux mondes.



Figure 3 Avion virtuel dans lequel les participants évoluent

Le maître du jeu prend des notes sur la collaboration des joueurs et peut fournir des indices ou introduire des perturbateurs pour aider ou ralentir les participants. Il utilise une application Windows agissant en tant que serveur, faisant office d'entité principale du jeu et qui dirige les autres. Cela lui permet de visualiser les perspectives des joueurs, améliorant ainsi sa compréhension de leurs réflexions. L'interface comporte plusieurs sections, disposées de gauche à droite : envoi de messages aux participants, envoi de perturbateurs prédéfinis, envoi d'indices prédéfinis et prise de notes. Cependant, il a été observé que le maître de jeu rencontrait des difficultés à accomplir toutes ses actions, limitant ainsi sa capacité à prendre des notes et entravant la fourniture d'un feedback complet.

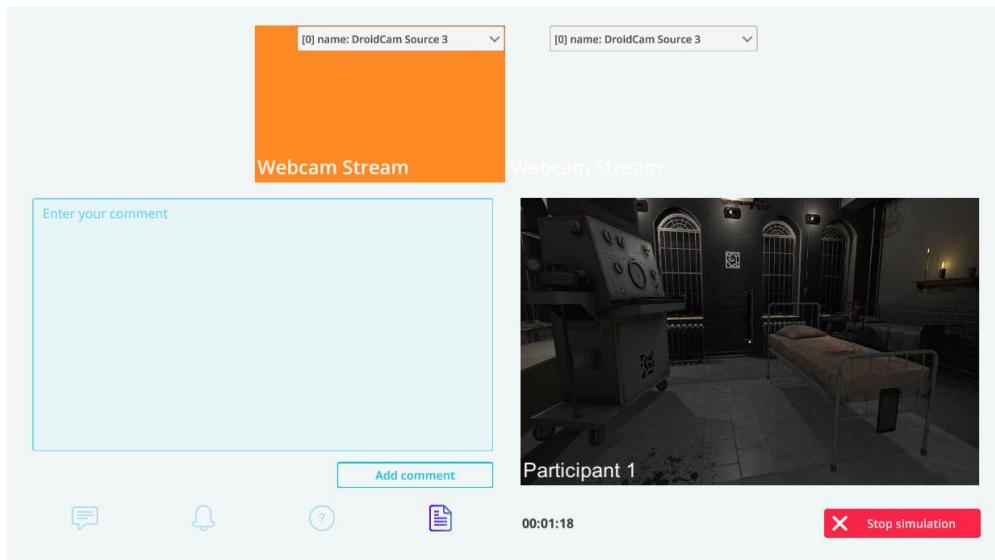


Figure 4 Application de monitoring

Comme visible dans l'image ci-dessus, les participants ne se trouvent pas à bord d'un avion. Le projet actuel comporte deux scénarios, le premier étant axé sur l'avion et développé en étroite collaboration avec les Hôpitaux Universitaires Genevois (HUG), tandis que le second se déroule dans un hôpital désaffecté, permettant ainsi de découvrir les possibilités et les limites du projet. Bien que ce deuxième scénario n'ait

pas encore été expérimenté, la complexité et le temps nécessaires pour son développement ont démontré la justification de ce travail de Master.

En résumé, la solution actuelle d'HOST comprend deux scénarios accessibles via une application de modération et une application dédiée aux HoloLens 2. Les énigmes sont conçues pour stimuler le travail d'équipe en combinant des indices numériques et réels. Cependant, cette solution présente des limitations en termes de perspectives d'améliorations, notamment pour la création de nouveaux scénarios et la réutilisation des énigmes, rendant le développement continu de cette application actuellement coûteux.

Suite aux retours de l'association « Ensemble », nous avons identifié les limites du projet et défini des axes d'amélioration cruciaux pour faire progresser le projet vers une nouvelle phase. En priorité, il est crucial d'établir une structure générique pour les scénarios, facilitant l'intégration de nouvelles idées de scénarios. La couche réseau doit également être réexaminée et adaptée à la nouvelle structure des scénarios.

Il a été relevé que le maître de jeu assume actuellement une charge de travail trop importante pour garantir une prise de notes correcte. Il est donc impératif de réduire cette charge de travail en mettant en place une gestion intelligente de certaines de ses responsabilités.

Avec le projet repartant de zéro, il est judicieux d'évaluer le matériel existant afin de déterminer s'il est pertinent de remplacer le casque de réalité augmentée par un modèle plus récent. Il est probable qu'un casque plus récent offre de nouvelles fonctionnalités bénéfiques pour le projet.

# 1. ANALYSE

Dans ce chapitre, nous allons analyser l'existant tout d'abord dans sa globalité, afin de mieux comprendre comment marche les applications, puis en la décomposant points par points afin d'en savoir plus sur chaque aspect. Nous pourrons également voir comment les choses ont été pensée, implémentée et leurs limites. Enfin, à partir des analyses faites, nous pourrons enfin spécifier les besoins et les attentes de ce projet de Master.

## 1.1. Analyse de la solution existante

### 1.1.1. Analyse de la structure et des communications entre les applications

Afin d'assurer une clarté maximale pour la suite des informations, il est crucial de comprendre la composition et l'interaction des deux applications. En ce qui concerne le réseau, les applications échangent des données via un réseau local. Il est évident qu'il peut y avoir plusieurs HoloLens 2 dans une session, mais il ne peut y avoir qu'un seul serveur.

Comme illustré dans l'image ci-dessous, chaque application possède sa propre base de données, stockée localement. Les données dans ces deux bases sont largement similaires, car elles incluent des informations relatives aux indices et perturbateurs.

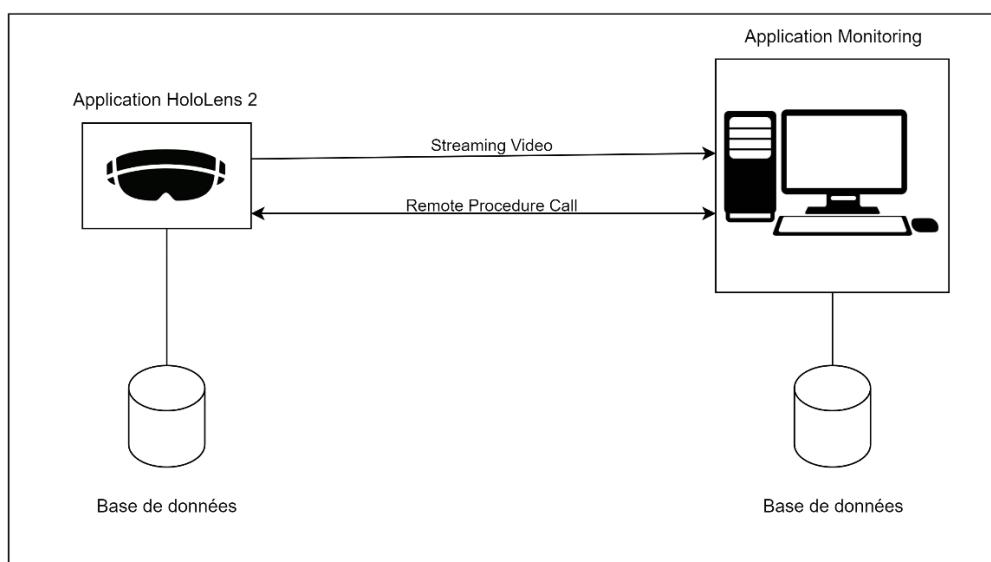


Figure 5 Structure de la communication entre les applications

La base de données de l'application de Monitoring contient également des informations sur les commentaires pris par le maître de jeu, ainsi que d'autres détails que je détaillerai ultérieurement. En fin de compte, ces bases de données sont peu utilisées, car les informations sur les indices et perturbateurs se retrouvent également dans les différents scripts. Il peut même arriver que l'intégrité des informations entre la base de données et le script ne soit pas assurée, ce qui ne pose pas de problème réel lors d'une simulation, mais plutôt dans la compréhension de quelle donnée est utilisée.

### 1.1.2. Analyse étape par étape d'une simulation et des fonctionnalités

Maintenant que la base des applications est établie, il est instructif de comprendre le déroulement complet d'une simulation. Tout d'abord, l'application de monitoring est lancée, initiant ainsi la création d'une simulation et la mise en place du serveur, ce qui permet aux participants équipés de HoloLens 2 de rejoindre la session.

L'application des HoloLens 2, nommée l'application de jeu, transporte les joueurs directement dans l'environnement virtuel. Cependant, cet environnement se déplace en suivant le regard de l'utilisateur, assurant ainsi un alignement avec le monde réel et permettant un ancrage synchronisé tant avec le monde réel qu'avec les autres joueurs.



Figure 6 Ancrage du monde virtuel avec le réel

Du côté de l'application de monitoring, il est possible de visualiser les casques connectés via leurs adresses IP, ainsi que de vérifier si les participants sont prêts en ayant « ancré » leur monde. Une fois que tous les participants sont prêts, la simulation peut être lancée.

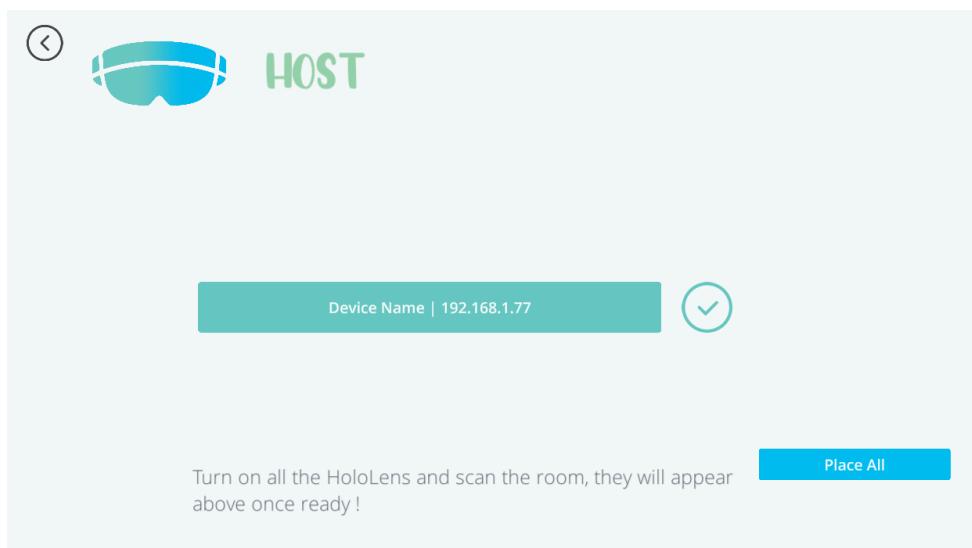


Figure 7 Lobby sur l'application de monitoring

Sur l'application de jeu, un petit tutoriel sur les différentes interactions apparaît. Une fois ce tutoriel terminé, les joueurs se retrouvent dans la salle virtuelle et le scénario peut débuter. Les joueurs peuvent interagir avec divers objets du décor pour résoudre des énigmes, mêlant ainsi le monde réel au virtuel. Par exemple, dans le scénario de l'avion, des indices sur un écran virtuel peuvent ouvrir un cadenas réel.



Figure 8 Tutoriel sur les interactions



Figure 9 Télévision avec des instructions pour ouvrir un cadenas

Du côté de l'application de modération, les flux vidéo des caméras des participants sont disponibles. Ces vues sont enregistrées tout au long de la simulation, permettant de fournir des retours aux participants. Le maître de jeu peut sélectionner la vue à agrandir, et cette vue principale est enregistrée. De plus, une webcam offre une vue globale de la pièce.

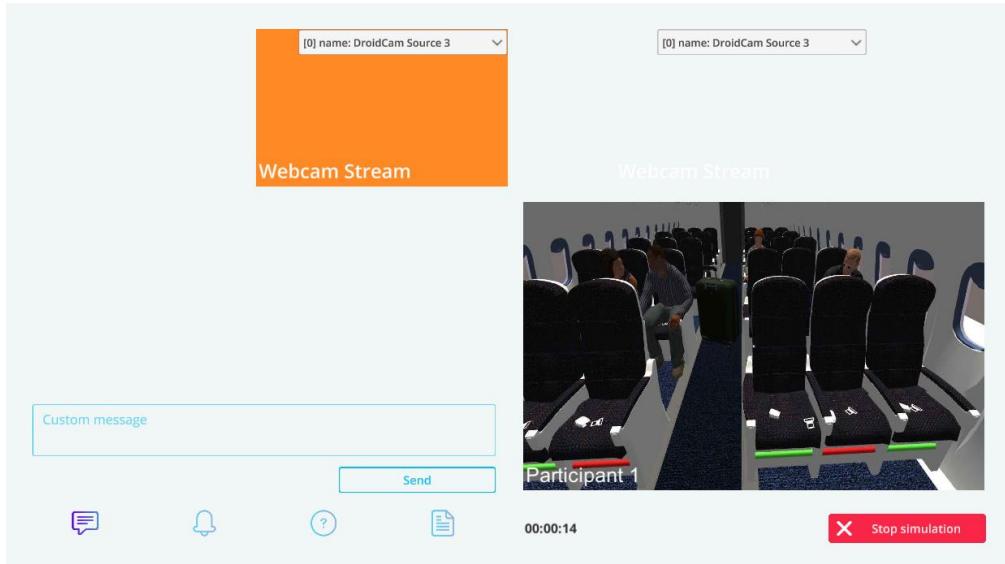


Figure 10 Interface utilisateur de l'application de monitoring

Le maître de jeu peut envoyer des messages aux participants via un champ textuel et déclencher des perturbateurs prédéfinis, tels que des sons ou des animations d'objets de jeu, en utilisant des boutons.

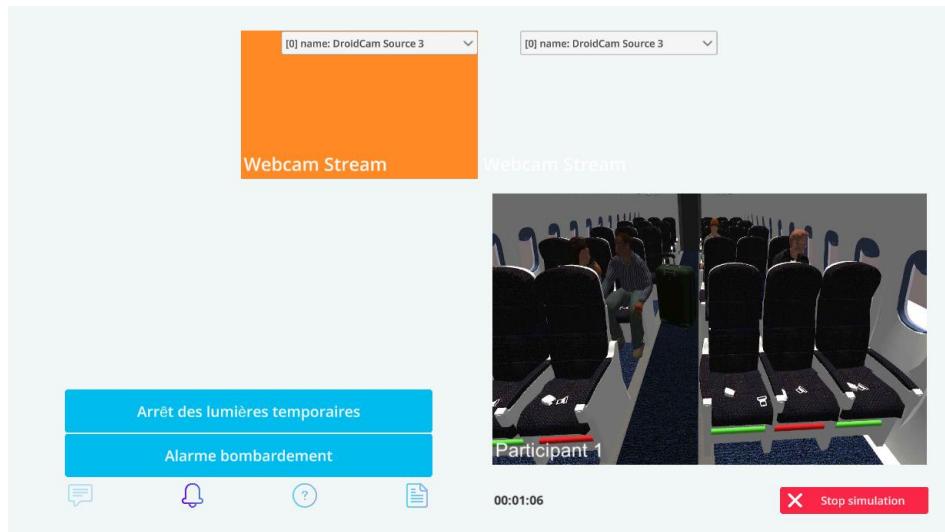


Figure 11 Interface utilisateur pour l'envoie des perturbateurs

Les indices peuvent prendre différentes formes, comme du texte, une image, un événement dans le monde ou un son. Deux listes déroulantes permettent de choisir l'énigme et l'indice à envoyer.

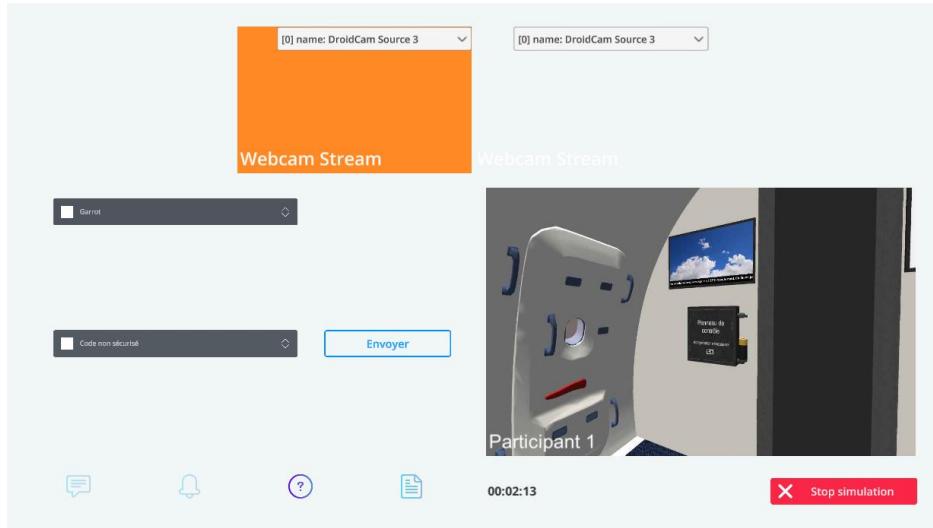


Figure 12 Interface utilisateur pour l'envoie d'indices

Enfin, le maître de jeu dispose d'un espace pour prendre des notes. Ces notes, ainsi que l'image et le temps en jeu, sont sauvegardés pour générer un rapport PDF de feedback ultérieurement. Les commentaires sont également utilisés comme sous-titres pour la vidéo enregistrée.

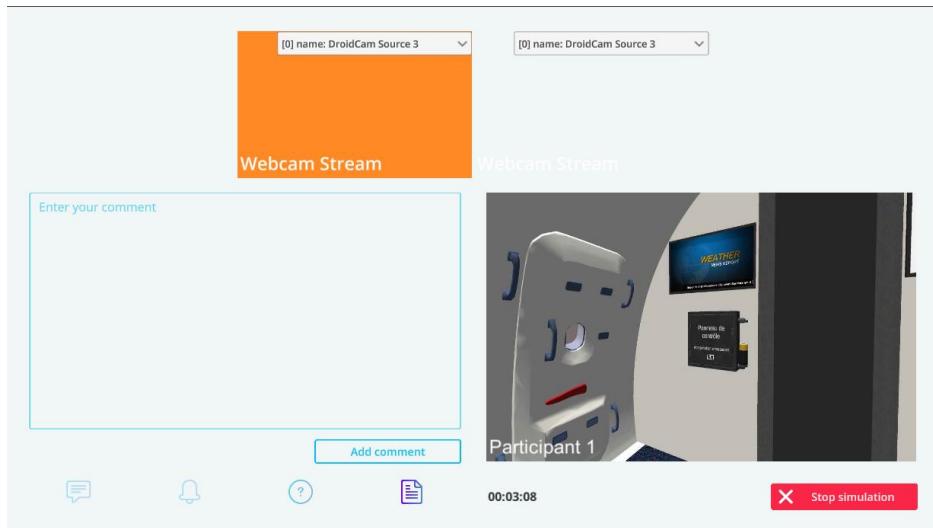


Figure 13 Interface utilisateur pour la prise de notes

La simulation est enregistrée, et à la fin, il est possible de générer un fichier mp4 où les sous -titres correspondant aux notes du maître de jeu sont directement incrustés sur l'image. Un rapport PDF comprenant des trios d'images, de commentaires et de temps est également généré, offrant un support pour les discussions en équipe à la fin de la simulation, permettant de se projeter grâce à l'image et au temps et de se remémorer la réflexion effectuée à ce moment précis.



Figure 14 Exemple de commentaires dans le PDF

### 1.1.3. Analyse détaillée de la couche réseau

La couche réseau du projet existant a été élaborée à l'aide d'un Asset Unity appelé « FMETP\_Stream », accompagné de l'utilisation de sockets TCP. Dans une première phase, un serveur et des clients sont créés grâce à cet asset, exploitant la fonction de découverte automatique disponible. Pour une clarté future, nous désignerons ce serveur comme le serveur « A » et ces clients comme les clients « A ». Cette paire de serveur/clients « A » facilite l'obtention de l'adresse IP du serveur sans avoir à la saisir manuellement.

Une fois l'adresse IP récupérée, un nouveau serveur, le serveur « B », ainsi que des clients « B » sont créés. Cette paire « B » fonctionne via une solution développée directement sur des sockets TCP, et elle est responsable de la majorité des échanges de l'application. Tous les RPC sont transmis via ces sockets, sous forme de JSON encodé en bytes, puis décodé à leur réception. Grâce à la réflexion, un système disponible en C# qui permet d'appeler une méthode d'un objet avec simplement une chaîne de caractères, il est possible d'appeler la méthode souhaitée pour son exécution.

Un RPC doit être enregistré dans le script répertoriant tous les RPC avec un identifiant. Étant donné que l'application de monitoring et l'application de jeu ne sont pas intégrées dans le même projet, il est nécessaire de reproduire les enregistrements dans les deux projets. Cela expose le projet à d'éventuels problèmes d'intégrité, pouvant être complexes à résoudre. De plus, étant un script utilisé pour tous les scénarios, toutes les inscriptions de tous les scénarios se trouvent dans ce script.

```

1 référence
private void InitRpc()
{
    HostNetworkManager.RegisterGameObject(HostNetworkId, this);
    HostNetworkManager.RegisterRPC(HostNetworkId, 0, "TriggerMessageEvent");
    HostNetworkManager.RegisterRPC(HostNetworkId, 1, "TriggerVirtualEvent");
    HostNetworkManager.RegisterRPC(HostNetworkId, 2, "TriggerHelpEvent");
    HostNetworkManager.RegisterRPC(HostNetworkId, 3, "StopSimulation");
    HostNetworkManager.RegisterRPC(HostNetworkId, 4, "StartStreaming");
    HostNetworkManager.RegisterRPC(HostNetworkId, 5, "SeatsFreeNotification");
    HostNetworkManager.RegisterRPC(HostNetworkId, 6, "SetSwitchState");
    HostNetworkManager.RegisterRPC(HostNetworkId, 7, "BreakerPanelOpen");
    HostNetworkManager.RegisterRPC(HostNetworkId, 8, "CryptedMessage");
    HostNetworkManager.RegisterRPC(HostNetworkId, 9, "MonitoringPressedButton");
    HostNetworkManager.RegisterRPC(HostNetworkId, 10, "MonitoringActiveButton");
    HostNetworkManager.RegisterRPC(HostNetworkId, 11, "MonitoringFeedback");
    HostNetworkManager.RegisterRPC(HostNetworkId, 12, "GarrotDone");
    HostNetworkManager.RegisterRPC(HostNetworkId, 13, "CryptedException");
    _rpcInitDone = true;
}

```

Figure 15 Script HelpRPC.cs où sont inscrits les RPC de l'application de monitoring

```

private void InitRpc()
{
    HostNetworkManager.RegisterGameObject(HostNetworkId, this);
    HostNetworkManager.RegisterRPC(HostNetworkId, 0, "TriggerMessageEvent");
    HostNetworkManager.RegisterRPC(HostNetworkId, 1, "TriggerVirtualEvent");
    HostNetworkManager.RegisterRPC(HostNetworkId, 2, "TriggerHelpEvent");
    HostNetworkManager.RegisterRPC(HostNetworkId, 3, "StopSimulation");
    HostNetworkManager.RegisterRPC(HostNetworkId, 4, "StartStreaming");
    HostNetworkManager.RegisterRPC(HostNetworkId, 5, "SeatsFreeNotification");
    HostNetworkManager.RegisterRPC(HostNetworkId, 6, "SetSwitchState");
    HostNetworkManager.RegisterRPC(HostNetworkId, 7, "BreakerPanelOpen");
    HostNetworkManager.RegisterRPC(HostNetworkId, 8, "CryptedException");
    HostNetworkManager.RegisterRPC(HostNetworkId, 9, "MonitoringPressedButton");
    HostNetworkManager.RegisterRPC(HostNetworkId, 10, "MonitoringActiveButton");
    HostNetworkManager.RegisterRPC(HostNetworkId, 11, "MonitoringFeedback");
    HostNetworkManager.RegisterRPC(HostNetworkId, 12, "GarrotDone");
    HostNetworkManager.RegisterRPC(HostNetworkId, 13, "CryptedException");
    _rpcInitDone = true;
}

```

Figure 16 Script HelpRPC.cs où sont inscrites les RPC de l'application de jeu

La paire de serveur/clients « A » sera ensuite réutilisée pendant la simulation pour transmettre les flux vidéo des participants au serveur. L'Asset « FMETP\_Stream », équipé de scripts permettant d'encoder et de décoder les vues des participants, assume la responsabilité totale du flux vidéo.

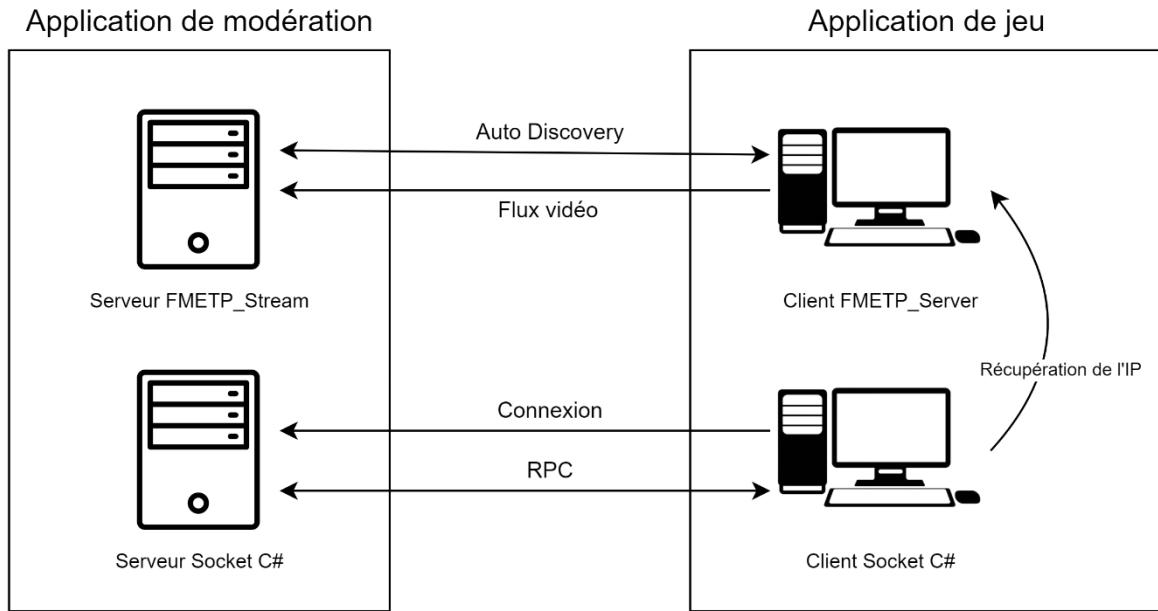


Figure 17 Schéma de l'utilisation des différentes paires serveur/client

#### 1.1.4. Analyse détaillée de la partie scénario et indices

L'analyse de la partie scénario s'est révélé complexe, car elle ne semble pas véritablement distincte des autres composantes. Bien que le scénario existe, il n'est pas clairement dissocié des autres éléments. On constate également qu'à l'origine, le projet a été conçu avec l'idée d'avoir plusieurs scénarios. En effet, l'inclusion d'informations sur les scénarios dans la base de données visait à rendre le code plus générique. L'idée sous-jacente était de permettre une fonctionnalité interchangeable en utilisant des données spécifiques à chaque scénario. Malheureusement, ce postulat de base n'a pas pu être maintenu.

Le premier scénario, celui de l'avion, est étroitement lié à la partie réseau. La logique de certaines énigmes est même intégrée directement dans le script qui gère le réseau. Cette interconnexion s'est accentuée avec l'ajout du deuxième scénario. Celui-ci n'est pas inclus dans la même application, mais dans une copie de la première où seules les informations spécifiques au scénario ont été modifiées. Cela inclut la scène Unity, les méthodes enregistrées en RPC, ainsi que les indices et les perturbateurs. Bien que l'ajout du scénario dans l'application, sans la copier, aurait pu être envisageable, les coûts auraient été prohibitifs. Une duplication considérable du code, et des parties du code chargées à chaque fois auraient été utilisées uniquement dans l'un ou l'autre des scénarios.

Il est également essentiel de noter que la notion de scénario existe de manière assez générale. Par exemple, on parle du scénario de l'avion pour décrire son environnement, mais il n'est pas impératif d'avoir une progression prédéfinie. Toutes les énigmes peuvent être résolues dans l'ordre souhaité, offrant une certaine liberté aux joueurs, mais pouvant également créer de la confusion quant à l'objectif. En revanche, le scénario de l'hôpital désaffecté a mis en place une séquence logique entre les énigmes, mais cette approche s'est avérée difficilement réutilisable et limitée à ce scénario spécifique.

Les indices et perturbateurs sont naturellement liés à un scénario. Tout comme les scénarios, ils ont été initialement conçus pour être introduits dans une base de données, facilitant ainsi leur interchangeabilité. Cependant, tout comme pour les scénarios, ils sont finalement codés en dur dans un script utilisé pour une grande partie de la gestion réseau. Bien qu'ils soient simples à utiliser et à mettre en place, il est nécessaire de veiller à attribuer le bon identifiant dans les deux projets, tout comme pour les RPC. À l'exception de l'affichage des indices sur l'interface de monitoring, où les informations sur les indices sont réécrites dans le code une seconde fois. Cette redondance peut prêter à confusion et entraîner davantage de problèmes d'intégrité.

```

Enigmes = new List<string>
{
    "Garrot",
    "Message crypté",
    "Monitoring",
    "Seringues",
    "Calcul mental - Tableau périodique élément"
};

Helps = new List<List<HelpCommand>>();

var garrot = new List<HelpCommand>
{
    new HelpCommand() { CommandName = "SendText", Text = "Code non sécurisé", TextParameter = "Le code n'est pas sécurisé." },
    new HelpCommand() { CommandName = "SendText", Text = "Numéro de la salle", TextParameter = "Le numéro de la salle a son importance" },
    new HelpCommand() { CommandName = "SendText", Text = "Garrot sur le patient", TextParameter = "Amener les bandages au patient !" }
};

Helps.Add(garrot);

var cryptedImage = new List<HelpCommand>();
cryptedImage.Add(new HelpCommand() { CommandName = "SendText", Text = "Toquer à la porte", TextParameter = "Quelqu'un a toqué à la porte." });
cryptedImage.Add(new HelpCommand() { CommandName = "SendImage", Text = "Image tablette reçue", ActionIndex = 54 });
cryptedImage.Add(new HelpCommand() { CommandName = "SendEvent", Text = "Flèche tablette", ActionIndex = 55 });
cryptedImage.Add(new HelpCommand() { CommandName = "SendImage", Text = "Exemple de déchiffrage", ActionIndex = 56 });
cryptedImage.Add(new HelpCommand() { CommandName = "SendEvent", Text = "Lumière sur toutes les clées", ActionIndex = 57 });

Helps.Add(cryptedImage);

```

Figure 18 Une partie du script HelpSelection.cs où sont écrits les indices

```

// Create the two original scenarios if they don't exist
if (GetScenarioByID(0) == null && GetScenarioByID(1) == null)
{
    Scenario ssel = new Scenario("Scenario Operation");
    ssel.AddVirtualEvent(new VirtualEvent(scenario: ssel.id, "1", recipient: "All", name: "Arrêt des lumières temporaires"));
    ssel.AddVirtualEvent(new VirtualEvent(scenario: ssel.id, "2", recipient: "All", name: "Alarme bombardement"));

    ssel.AddMessageEvent(new MessageEvent(scenario: ssel.id, type: "Alert", content: "Le code est en rapport avec le soleil, pensez à fouiller le sac", recipient: "All"));
    ssel.AddMessageEvent(scenario: ssel.id, type: "Alert", content: "Le modèle de l'avion est affiché quelque part", recipient: "All");
    ssel.AddMessageEvent(new MessageEvent(scenario: ssel.id, type: "Alert", content: "Ces formes sont présentes à différents endroits...", recipient: "All"));
    ssel.AddMessageEvent(new MessageEvent(scenario: ssel.id, type: "Alert", content: "Certains objets peuvent être manipulés à la main", recipient: "All"));
    ssel.AddMessageEvent(new MessageEvent(scenario: ssel.id, type: "Alert", content: "Attention ! Vous oubliez votre patiente", recipient: "All"));
    ssel.AddMessageEvent(new MessageEvent(scenario: ssel.id, type: "Alert", content: "Il vous reste 10min", recipient: "All"));

    ssel.AddHelpEvent(new HelpEvent(scenario: ssel.id, action_number: "1", recipient: "All", name: "Indice flèche sac"));

    PutScenario(ssel);
    Debug.Log("[DB] - Added two predefined scenarios");
}

```

Figure 19 Une partie du script DBManager.cs où sont réécrits les indices

### 1.1.5. Analyse détaillée de la partie monitoring et feedback

La supervision des joueurs revêt une importance cruciale dans ce projet. L'objectif de faciliter l'amélioration de la collaboration des participants nécessite la fourniture de retours constructifs. L'application met à disposition divers outils visant à assister le maître de jeu dans ses fonctions. Parmi ceux-ci, la possibilité d'accéder au flux vidéo des joueurs s'avère particulièrement utile. Cela permet au maître de jeu de se plonger davantage dans l'expérience du joueur, favorisant ainsi une meilleure compréhension de ses réflexions.

Cependant, cette fonctionnalité ne s'arrête pas là. En conjonction avec l'outil de prise de notes, il devient possible d'observer l'évolution des joueurs dans le jeu, de noter leurs performances, et en fin de partie, de générer une vidéo récapitulative ainsi qu'un rapport en PDF. La vidéo est enregistrée tout au long de la simulation à l'aide d'une adaptation sur Unity de FFmpeg, une collection de logiciels libres dédiés au traitement de flux vidéo et audio. Cette approche permet de spécifier une zone à enregistrer sur l'interface utilisateur et de capturer la vidéo. Il est important de souligner que l'utilisation de FFmpeg dépend du système d'exploitation, et dans le cas de l'application Windows, la version .exe est employée.

Enfin, la génération du rapport au format PDF est réalisée grâce à la librairie PDFSharpCore, une bibliothèque C# dédiée à la création de fichiers PDF. Cette approche offre la possibilité de créer un PDF en utilisant simplement une suite de fonctions en C#, configurables selon les besoins.

Cependant, un défi majeur se présente, car le maître de jeu, qui a également la responsabilité d'aider ou de perturber les joueurs pendant la simulation, ne peut se concentrer uniquement sur la prise de notes. Ce problème significatif a été identifié au cours des différentes itérations effectuées par les HUG. Alléger les responsabilités du maître de jeu pourrait donc conduire à des retours bien plus efficaces pour les participants et accroître considérablement l'impact d'une partie.

## 1.2. Spécification des besoins

Après avoir dressé un état des lieux de l'application actuelle, il est possible de dégager les principaux axes d'amélioration. Il est toutefois essentiel de définir clairement les utilisateurs destinés à utiliser l'application. Naturellement, cela inclut le maître de jeu, disposant de son application de monitoring dédiée, ainsi que les joueurs utilisant leur application pour la réalité augmentée. Il convient néanmoins de ne pas négliger l'aspect développeur, car le projet vise également à simplifier l'implémentation de futurs scénarios.

En résumé, les besoins seront spécifiés pour trois types d'utilisateurs, à savoir :

- Les joueurs ;
- Le maître de jeu ;
- Les développeurs.

### Spécification de la couche Réseau

En premier lieu, il serait pertinent d'opter pour l'utilisation d'un seul serveur et client pour l'intégralité de l'application. De plus, simplifier l'utilisation des RPC semble être un objectif louable, étant donné que c'est principalement la source du problème de regroupement de code à un seul endroit. Ainsi, concevoir et mettre en œuvre une toute nouvelle couche réseau semble être la meilleure approche.

En tant que développeur, mes attentes sont les suivantes :

- Rendre l'utilisation des RPC aussi simple que possible ;
- Éviter de concentrer l'ensemble des différents RPC dans un seul et unique script ;
- Faciliter l'ajout de scénarios ;
- Assurer que chaque script dédié à un scénario ne soit pas partagé avec un autre ;

- Simplifier la synchronisation des objets en n'impliquant que l'ajout de script(s) sur un objet ;
- Mettre en place une surcouche réseau pour simplifier tous les aspects liés au réseau ;

En tant que joueur, mes attentes sont les suivantes :

- Assurer la synchronisation des objets entre les différents joueurs ;
- Se connecter automatiquement au serveur sans que je n'aie à entrer l'adresse IP.

### **Spécification de la couche Scénario Générique**

Le point central concerne les scénarios et la manière de les rendre facilement implémentables. Il est crucial de définir une structure générique à laquelle tous les scénarios actuels et futurs pourront être rattachés. Ce cadre doit d'abord être conceptualisé en tenant compte de la couche réseau ainsi que des nouvelles attentes qui seront décrites ci-après.

En tant que développeur, mes attentes sont les suivantes :

- Isoler complètement le squelette du scénario du scénario en lui-même ;
- Définir clairement les étapes du scénario et permettre de les suivre ;
- Confier au squelette du scénario la gestion des synchronisations réseau principales ;
- Garantir que le squelette soit le moins restrictif possible ;
- Rendre les indices et perturbateurs également génériques et faciles à implémenter ;

En tant que joueur, mes attentes sont les suivantes :

- Obtenir une orientation légère du scénario pour la résolution d'une énigme ;
- Assurer que les résolutions d'énigmes soient synchronisées entre tous les participants ;
- Recevoir des indices lorsque je suis bloqué dans ma progression ;

En tant que maître de jeu, mes attentes sont les suivantes :

- Assurer que le scénario s'enchaîne sans nécessiter d'intervention constante.

### **Spécification de la couche Gestionnaire de Scénario Intelligent**

Compte tenu du fait que le maître de jeu assume actuellement trop de responsabilités, il est pertinent d'envisager la conception d'un outil de monitoring intelligent. Réduire l'intervention humaine permettrait de diminuer la charge de travail et simplifierait la prise de notes en vue du retour à la fin de la simulation.

En tant que développeur, mes attentes sont les suivantes :

- Établir des niveaux d'impact différents pour les indices et perturbateurs ;
- Permettre la définition de paramètres par défaut propres à chaque scénario ;
- Automatiser le calcul du niveau d'impact des indices et perturbateurs ;
- Calculer l'importance d'une énigme en fonction du temps estimé défini ;

En tant que joueur, mes attentes sont les suivantes :

- Assurer que les indices et perturbateurs ne soient pas trop prononcés lorsque je commence une nouvelle énigme, même en cas de retard ;
- S'assurer que les indices reçus portent sur des éléments qui n'ont pas encore été réalisés .

En tant que maître de jeu, mes attentes sont les suivantes :

- Assurer l'envoi automatique d'indices et de perturbateurs sans nécessiter une intervention manuelle ;
- Veiller à ce que les indices ne deviennent pas trop évidents trop rapidement ;
- Permettre au moins l'ajustement du paramètre de temps estimé pour chaque simulation ;
- Conserver les informations de temps sur chaque énigme afin de pouvoir effectuer un récapitulatif ;

### **Spécification de la couche monitoring**

Malgré le fait que l'envoie d'indices et de perturbateurs deviennent automatique, il faut néanmoins toujours avoir la possibilité de voir les différentes vues de participants, permettant de garder la méthode de feedback avec la vidéo et le rapport PDF.

En tant que maître de jeu, je souhaite que :

- Les vues de utilisateurs soient visibles sur l'application de monitoring ;
- Il soit possible de prendre des notes afin de pouvoir construire un feedback pour les joueurs ;
- Il soit possible d'envoyer des messages aux joueurs afin de leur communiquer des informations qu'ils auraient besoin pendant la simulation ;
- Une vidéo récapitulative avec la vue choisie ainsi que les commentaires soit générée ;
- Un rapport PDF ayant l'image ainsi que le commentaire pris à un temps T soit généré.

### **Spécification des besoins matériels**

Le projet repartant de zéro, il est pertinent d'examiner l'offre de casques de réalité augmentée afin de choisir celui qui correspond le mieux au nouveau projet. Un autre besoin, directement lié au casque, consiste à explorer des moyens de simplifier l'ancrage du monde virtuel pour les participants.

En tant que développeur, mes attentes sont les suivantes :

- Assurer que l'ancrage soit effectué au préalable d'un scénario, évitant ainsi la nécessité de répéter la manipulation pour chaque scénario ;
- S'assurer qu'au minimum, le casque permette les interactions de toucher et d'attraper.

En tant que joueur, mes attentes sont les suivantes :

- Garantir une immersion totale dans la simulation grâce au casque ;
- Permettre la vision de la réalité à tout moment ;
- Assurer que les interactions soient faciles à réaliser.

## 2. RECHERCHES ET CONCEPTIONS

Dans cette section, nous allons aborder la conceptualisation du projet, en examinant son architecture, son design, ainsi que les réflexions qui ont conduit à ce résultat. Le projet a été structuré en couches, avec une dépendance majoritairement ascendante, et c'est selon l'ordre de mise en œuvre que ce chapitre sera organisé.

### 2.1. Recherche et choix du casque de réalité augmentée

Le choix du matériel est clairement la première décision à prendre. C'est en fonction de cette décision que l'ensemble du projet prendra forme. Il est crucial de s'assurer que le casque sélectionné répond aux critères nécessaires, garantissant ainsi le bon déroulement du développement du projet. De plus, il est essentiel de penser à l'avenir. Quel casque permettra au projet de s'épanouir dans le futur ? Bien que répondre à cette question puisse être délicat, elle revêt une importance significative, car elle orientera l'élimination de certaines options.

#### Recherche sur les HoloLens 2

Avec le HoloLens 2, malheureusement, la réponse à la question posée dans le paragraphe précédent est claire. Microsoft a mis fin au développement du HoloLens 3, scellant ainsi le sort de cette technologie. Bien sûr, quelques mises à jour seront toujours apportées, mais il n'y a aucune raison de démarrer un « nouveau » projet sur une technologie en déclin.

Cependant, il n'est pas encore possible d'écartier complètement cette technologie. En effet, il s'agit de l'un des seuls casques autonomes réellement axés sur la réalité augmentée. De plus, il a été choisi pour le projet existant, ce qui simplifierait le développement du nouveau projet car il s'agit d'une technologie familière avec un exemple concret du projet.

Un avantage majeur est la capacité à voir la réalité comme à travers des lunettes, assurant une qualité visuelle du monde réel parfaite. Cependant, pour compenser ce point positif, il est important de noter que c'est du côté virtuel que certaines lacunes se manifestent. En fonction de l'exposition à la lumière, la couleur, ou la lumière virtuelle, il peut être compliqué de discerner les éléments virtuels.

Enfin, la question du prix se pose. L'application étant conçue pour être multijoueur, avec plusieurs personnes portant un casque, un coût élevé serait évidemment un inconvénient majeur. Or, les casques de réalité augmentée sont généralement coûteux et ne sont pas principalement destinés au grand public, mais plutôt aux entreprises. Malheureusement, les HoloLens 2 ne font pas exception à cette règle.

#### Recherche sur le Meta Quest Pro

Contrairement à Microsoft, Meta a fait le choix de se concentrer sur la réalité virtuelle. Après plusieurs succès avec leurs casques, ils ont décidé d'intégrer la réalité augmentée à leur offre. Le Meta Quest Pro permet aux joueurs de voir le monde réel à travers ses caméras et son écran, tout en superposant des éléments virtuels sur le flux vidéo du monde réel, offrant ainsi une autre approche de la réalité augmentée.

Les Meta Quest sont généralement commandés à l'aide de manettes, mais ils peuvent également suivre les mouvements des mains, ce qui en fait une option potentielle pour la technologie du projet. De plus, cette technologie est en plein essor, garantissant de nouvelles fonctionnalités dans les années à venir.

À la différence des HoloLens 2, le Meta Quest Pro affiche parfaitement les éléments virtuels, les rendant clairement discernables quelle que soit la lumière, la couleur ou l'environnement. Cependant, contrairement aux HoloLens, la qualité du flux vidéo de la réalité n'est pas optimale. Ce casque ne dispose pas d'une caméra couleur, mais plutôt d'une caméra en noir et blanc et de capteurs de couleurs. Bien qu'il soit possible de discerner les objets réels, jouer dans de telles conditions n'est pas particulièrement agréable.

### **Recherche sur le Meta Quest 3**

Au commencement du projet, ce casque est annoncé mais n'a pas encore de date de sortie confirmée. On prévoit sa disponibilité dans les deux mois suivant le début du projet, mais les détails restent flous. Cette incertitude peut s'avérer décourageante, car il est complexe de choisir un matériel sur lequel il n'est pas possible de travailler. Néanmoins, Meta a annoncé que, contrairement à ses casques précédents, celui-ci serait un casque de réalité mixte, combinant la réalité augmentée et la réalité virtuelle.

Ce casque est doté d'une caméra couleur, offrant une vision confortable de la réalité pendant le jeu, bien qu'il soit toujours difficile de regarder un écran à travers le casque. Il intègre également une détection de l'environnement, permettant de cartographier la pièce de jeu pour connaître l'emplacement du sol, des murs, du plafond et même des meubles.

Tout comme le Meta Quest Pro, il peut être contrôlé à l'aide de manettes et également par des mouvements des mains. Le suivi des mains a été amélioré par rapport à la version Pro, offrant une expérience plus naturelle et satisfaisante.

### **Choix du casque de réalité augmentée effectué**

Ce projet, se voulant expérimental et repartant de zéro, ne peut ignorer l'utilisation d'un casque de pointe sur le plan technologique. Le Meta Quest 3 apporte de nouvelles fonctionnalités et offre une garantie de pérennité technologique importante. Bien qu'il ne soit pas disponible dès le lancement, la base du kit de développement permettant de créer des applications pour le Meta Quest 3 est la même que celle du Pro.

Il semble envisageable de débuter le développement sur un Pro et de basculer ultérieurement vers le Quest 3. Cette approche est d'autant plus pertinente que certaines parties du projet, telles que le réseau, peuvent être complètement indépendantes du casque, n'étant pas directement liées à ses fonctionnalités.

Après des discussions avec le responsable de projet, M. Rekik, la décision commune a été prise d'adopter le Meta Quest 3 comme casque de réalité augmentée pour le projet.

## **2.2. Recherche et conception de la couche Réseau**

La couche réseau constitue la base fondamentale de l'application. Bien que l'objectif soit de la rendre aussi générique que possible en la dissociant autant que possible du reste, il est évident que les choix de conception des prochaines parties seront inévitablement guidés par la couche réseau.

Il est important de rappeler que, pour cette couche, les besoins principaux sont les suivants :

- - La possibilité d'effectuer des RPC ;
- - La possibilité de diffuser plusieurs flux vidéo ;
- - La possibilité de synchroniser les objets ;
- - La capacité de découvrir automatiquement le serveur.

### 2.2.1. Recherche sur les méthodes existantes

Plusieurs approches du problème sont envisageables. D'un côté, il est envisageable de concevoir sa propre couche à partir de sockets UDP, tandis que de l'autre, l'utilisation d'une bibliothèque existante est une option. L'avantage de la première solution réside dans sa capacité à répondre précisément à nos besoins, bien qu'elle exige un investissement significatif en termes de conception et d'implémentation.

Recourir à une bibliothèque existante aurait l'avantage de considérablement réduire la charge de travail. Cependant, trouver la bibliothèque appropriée, offrant les fonctionnalités nécessaires sans entraver l'évolutivité du projet, peut s'avérer complexe.

#### Netcode for GameObject

Netcode for GameObject est la solution propriétaire d'Unity. Cette solution dispose d'une documentation substantielle et est facile à mettre en place. Cependant, elle offre une multitude de fonctionnalités qui ne sont pas nécessaires pour le projet, du moins à court terme. Ainsi, bien que cette technologie soit facile à utiliser, elle peut être complexe à maîtriser.

L'architecture de cette solution est de type client-serveur, où le serveur a un rôle prépondérant et les clients se contentent de lui faire des demandes. Cela signifie que l'espace de jeu doit également être disponible sur le serveur, même si ce n'est pas nécessaire pour ce projet. L'avantage majeur de cette solution réside dans sa facilité d'utilisation pour les RPC. En effet, il suffit de décorer une méthode pour la rendre utilisable depuis une autre machine. De plus, la synchronisation des objets est simplifiée, ne nécessitant que l'ajout de scripts sur les objets à rendre partagés.

Cependant, cette solution ne propose pas de moyen prédéfini pour diffuser un flux vidéo. Bien que la transmission de ces flux ne devrait pas poser de problème, car ce ne sont que des octets à transférer, l'encodage et le décodage exigent un investissement de temps conséquent en développement. De plus, Unity ne propose pas de fonctionnalité de découverte automatique des serveurs, même si les versions précédentes disposaient de cette fonctionnalité.

#### Asset Unity « FMETP\_Stream »

Il existe une pléthore d'assets sur le magasin Unity. Examiner chacun d'eux serait laborieux, c'est pourquoi la discussion se concentrera exclusivement sur « FMETP\_Stream », l'asset utilisé dans la solution existante et bénéficiant d'une excellente évaluation sur le magasin Unity.

Cet asset permet d'établir des connexions TCP/UDP/Websocket entre plusieurs clients et un serveur. L'objectif principal de cet asset est la transmission de flux vidéo, que ce soit à partir d'un smartphone, d'un ordinateur, d'un HoloLens 2 ou même d'un Meta Quest. La version 3 de l'asset fournit également une

première approche de la synchronisation des objets, uniquement dans le sens du serveur vers le client. Cela signifie que le client n'a pas la capacité de déplacer un objet. De plus, l'asset prend en charge la découverte automatique des serveurs. Ainsi, une grande partie des besoins du projet est couverte par cet asset, qui présente également l'avantage d'être adaptable grâce à la possibilité de modifier le code source en C#.

Cependant, les RPC ne sont pas pris en charge par cette méthode. Il est donc nécessaire de développer une solution supplémentaire pour les rendre possibles.

### **Choix effectué**

Netcode for GameObject présente l'avantage d'offrir une gamme étendue de fonctionnalités, ce qui pourrait s'avérer utile dans l'évolution future de l'application. Cependant, son apprentissage requiert du temps, et son utilisation peut rapidement devenir complexe. De plus, Unity impose une structure forte qui pourrait ne pas correspondre aux besoins du projet. En outre, il ne couvre pas tous les besoins du projet.

C'est pourquoi la décision a été prise d'utiliser l'asset « FMETP\_Stream ». Il répond à tous les besoins du projet, à l'exception des RPC. Cependant, ceux-ci sont simples à mettre en place, comme en témoigne leur implémentation réussie dans l'application existante.

L'utilisation de cet asset garantira une prise en main plus complète de l'ensemble du processus. De plus, il sera plus aisés de concevoir des outils simplifiant le travail lors de l'ajout d'un scénario. En raison de sa simplicité, il sera également plus rapide de comprendre l'ensemble de l'asset, permettant ainsi d'être opérationnel plus rapidement.

#### **2.2.2. Conception de la structure du réseau**

À présent que la base a été sélectionnée, il est impératif de la développer davantage pour la rendre plus accessible. Divers outils doivent être mis en place pour répondre au mieux aux besoins du projet.

##### **Le script réseau Host Network Manager**

Le gestionnaire réseau principal, nommé "Host Network Manager" dans le script, agit comme une surcouche réseau pour le projet. C'est à cet endroit que tous les liens avec l'asset « FMETP\_Stream » sont établis et que les méthodes de communication réseau sont implémentées. Il joue un rôle central dans la gestion du réseau pour ce projet, étant le point de passage incontournable pour chaque interaction. Ce gestionnaire dirige les messages reçus vers les entités appropriées.

Au démarrage de l'objet contenant ce script, une recherche automatique des objets synchronisés est effectuée, et ceux-ci sont inscrits dans la liste d'objets synchronisés de l'asset. De plus, il est chargé de gérer les connexions et déconnexions des participants, assurant ainsi le bon fonctionnement de l'application. Il a également la responsabilité de gérer les RPC, qui seront détaillés dans la section suivante, en les encodant/décodant et en les envoyant à la cible appropriée. Du côté de la cible, il prend en charge la réception des RPC afin de les redistribuer à l'objet désiré, permettant ainsi l'exécution de la méthode associée.

## Le script réseau Host Network RPC

La méthode la plus simple pour effectuer des RPC consiste à coder/décoder des messages. Il suffit ensuite d'interpréter le message reçu pour déterminer quelle méthode appeler. En incluant des informations telles que l'ID de l'instance, le nom de la méthode et les paramètres, toutes les données nécessaires sont disponibles pour exécuter la méthode correctement.

Il est désormais crucial de concevoir un moyen de simplifier la création de nouveaux RPC, nécessitant la création d'un outil facilitant les développements futurs. En automatisant la découverte des instances dotées de RPC, il devient possible d'avoir plusieurs scripts comportant des RPC, permettant ainsi de répartir les responsabilités entre différents objets, palliant ainsi l'un des défauts du projet existant.

Il est évident que ces instances doivent avoir des ID différentes pour pouvoir les différencier. Cependant, il est préférable d'avoir un ID pour l'instance entière plutôt que pour chaque méthode, réduisant ainsi les erreurs en diminuant le nombre d'ID à gérer.

En créant le script « HostNetworkRPC » avec l'information de l'ID de l'instance, il suffit de faire hériter les scripts nécessitant des RPC de celui-ci pour que ces méthodes deviennent à leurs tours distantes. En maintenant une liste statique des instances dans le script « HostNetworkRPC », à laquelle tous les objets héritant du script seront ajoutés, il est possible de connaître en tout temps toutes les instances sur lesquelles appeler les RPC. Lorsque le gestionnaire reçoit un message avec l'ID de l'instance, il lui suffit de vérifier dans la liste quelle instance a l'ID correspondant à son message et de lui déléguer la responsabilité d'exécuter la bonne méthode.

Ainsi, le script a pour responsabilité d'être capable d'exécuter une méthode à partir du nom en chaîne de la méthode et de ses paramètres. Il est envisageable de redéfinir cette méthode pour chaque spécialisation, permettant d'ajouter des conditions en fonction du nom de la méthode, mais cela relève de l'implémentation et sera détaillé dans la section appropriée.

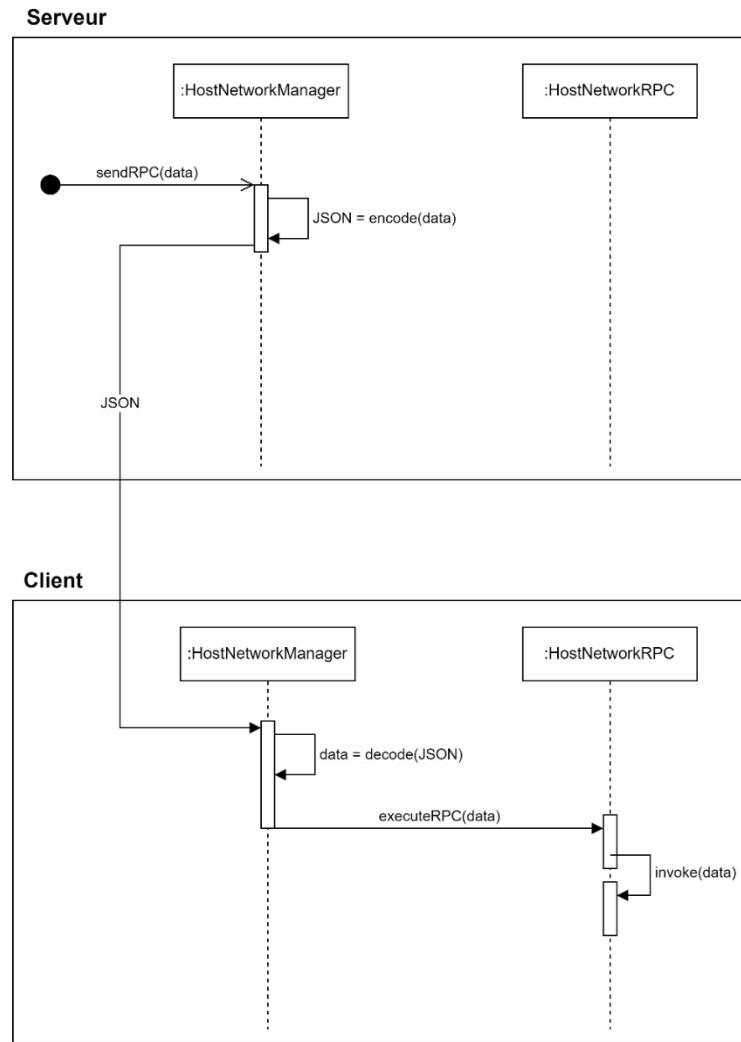


Figure 20 Diagramme de séquence simplifié des RPC

### Le script réseau Host Network Object

L'objectif de ce script est également de permettre la récupération automatique de tous les objets synchronisés dans une scène. L'asset choisi pour la synchronisation des objets du serveur vers le client requiert simplement de remplir la liste présente dans l'asset pour que cela fonctionne. Cependant, nous souhaitons également que les clients puissent déplacer les objets.

Ce script a été conçu pour résoudre ce problème spécifique. Il détectera lorsque l'objet a été déplacé, puis informera le serveur de sa nouvelle position. Le serveur pourra alors ajuster la position de l'objet dans son monde, assurant ainsi que ce déplacement soit synchronisé pour tous les utilisateurs.

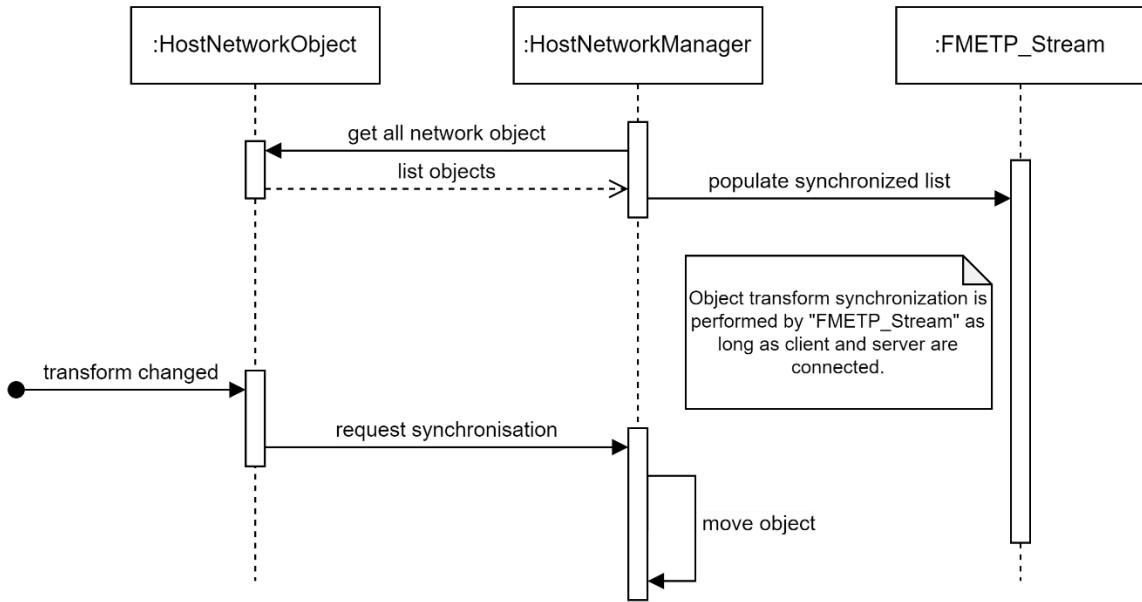


Figure 21 Diagramme de séquence simplifié des objets synchronisés

## 2.3. Réflexion et conception de la couche Scénario

### Réflexion sur la structure d'un scénario

La couche scénario est indéniablement la plus cruciale du projet. Une réflexion préliminaire sur la meilleure façon de réaliser la structure générique requise est essentielle. Pour ce faire, nous avons examiné ce qu'est un scénario :

- Un environnement ;
- Une histoire et un objectif ;
- Une série d'énigmes ;
- Des éléments à activer ou à trouver ;
- Un ordre potentiel entre chaque élément.

Ces points peuvent être illustrés avec des éléments présents dans les deux scénarios existants :

- L'environnement, qui est un avion ou un hôpital désaffecté ;
- L'histoire/objectif, qui consiste à accoucher une femme enceinte ou à opérer un patient ;
- Une série d'énigmes qui implique de trouver des informations sur le patient, des outils, et de modifier l'environnement pour atteindre l'objectif ;
- Des éléments multiples tels qu'une batterie à remettre en place, des cadenas à ouvrir, des informations visuelles ;
- L'ordre entre les énigmes n'existe pas pour le scénario de l'avion, mais il est présent dans le scénario de l'hôpital, où il faut d'abord découvrir l'allergie du patient avant de lui administrer un médicament, au risque de lui donner le mauvais.

Il devient évident qu'il nous faut différents objets pour classifier ces différents points. Un objet scénario, comprenant le but, l'environnement et la série d'éénigmes. Puis, les éénigmes en elles-mêmes, qui auraient chacune des éléments à activer ou à trouver.

Cependant, de cette manière, l'ordre facultatif entre les éénigmes est manquant. On pourrait l'imposer en fonction de l'ordre des éénigmes dans la liste du scénario, mais cela rendrait impossible la réalisation asynchrone de différentes étapes. Un niveau supplémentaire, représentant les éléments importants d'une éénigme, entre en jeu.

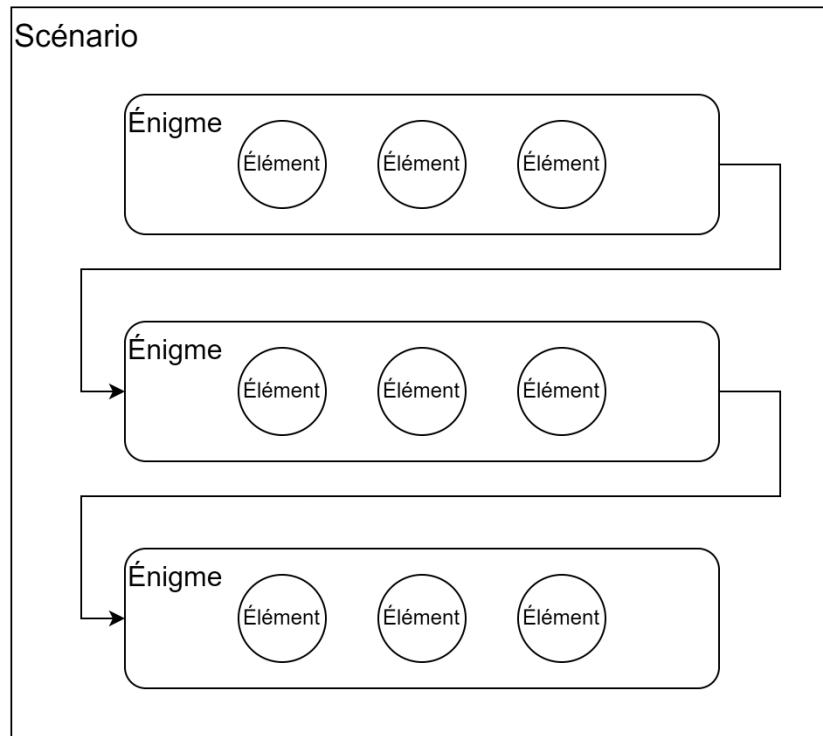


Figure 22 Schéma de la structure d'un scénario

En intégrant la notion d'éléments déclencheurs, tout en les rendant non ordonnés, nous sommes désormais capables de développer des scénarios ordonnés, non ordonnés, ou une combinaison des deux.

Dans le cas d'un scénario ordonné ou mixte, imaginons plusieurs éénigmes telles que l'éénigme A et l'éénigme B. L'éénigme A possède deux éléments déclencheurs, à savoir la résolution d'un cadenas et le déplacement d'un objet. Ces deux éléments peuvent être accomplis dans l'ordre souhaité, mais une fois les deux terminés, l'éénigme est considérée comme achevée, et on peut passer à la suivante.

Pour un scénario non-ordonné, il n'y aurait qu'une seule éénigme regroupant tous les éléments à accomplir pour l'ensemble du scénario. Ainsi, il serait possible de résoudre le scénario de manière non-ordonnée.

C'est sur cette base que toute la structure a été conçue. Il est maintenant nécessaire d'extraire les événements importants afin de les rendre paramétrables par le développeur. En suivant le schéma ci-dessus, quelques moments clés peuvent être identifiés.

Tout d'abord, nous avons le début et la fin d'un scénario. De manière similaire, chaque énigme doit également posséder un début et une fin. Quant aux éléments déclencheurs, ils ne présentent pas de début ou de fin réels, étant donné qu'ils ne suivent pas d'ordre spécifique. Cependant, il est utile de connaître le moment où ils ont été complétés.

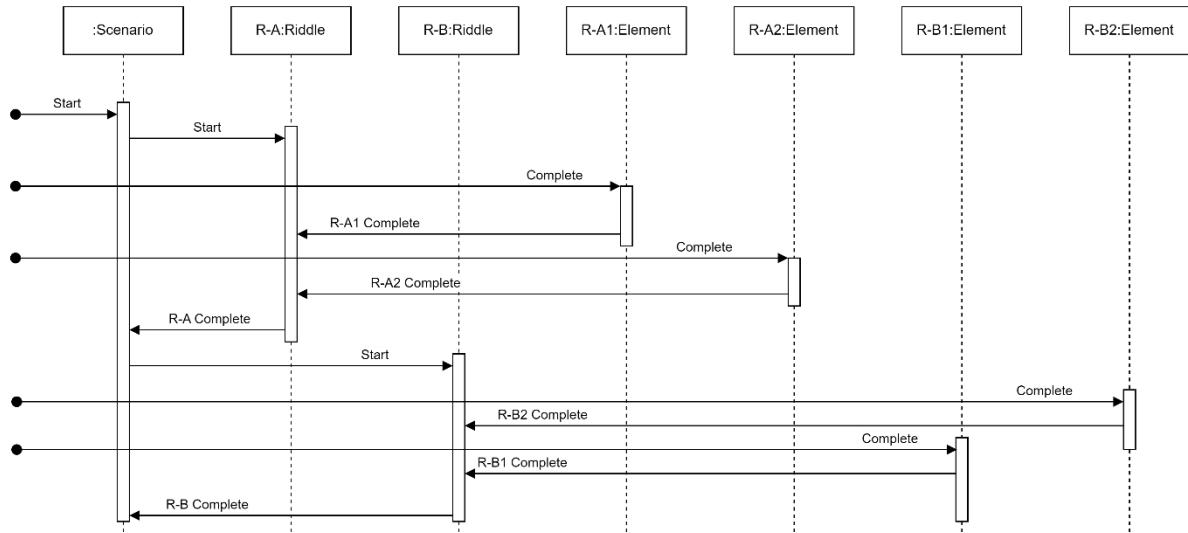


Figure 23 Diagramme de séquence simplifié et non complet de la structure du scénario

Dans le diagramme ci-dessus, qui ne présente pas de logique réelle mais expose uniquement les interactions entre les objets, on observe qu'il est possible de terminer les éléments dans n'importe quel ordre, mais pas les énigmes.

On comprend rapidement que la logique pour déterminer si une énigme est terminée consiste à vérifier que tous ses éléments sont complets. De même, pour le scénario, une fois que toutes les énigmes sont achevées, on peut conclure que le scénario est terminé.

Cependant, une situation particulière demeure en suspens. Que se passe-t-il si un élément est complété avant le démarrage de son énigme associée ? D'un point de vue du joueur, il serait inacceptable que l'interaction ne soit pas prise en compte. Il n'est pas concevable qu'en effectuant deux fois la même action, le résultat puisse différer. La seule solution viable est de considérer que l'élément a été complété et de vérifier au démarrage d'une énigme si celle-ci est déjà terminée ou non.

### Conception de la structure d'un scénario

Pour élaborer un squelette générique intégrant les moments clés définis dans la section précédente, il est nécessaire de les distinguer du reste et de les rendre modifiables. Après mûre réflexion, deux possibilités se sont présentées. La première consiste à recourir à l'héritage et au polymorphisme.

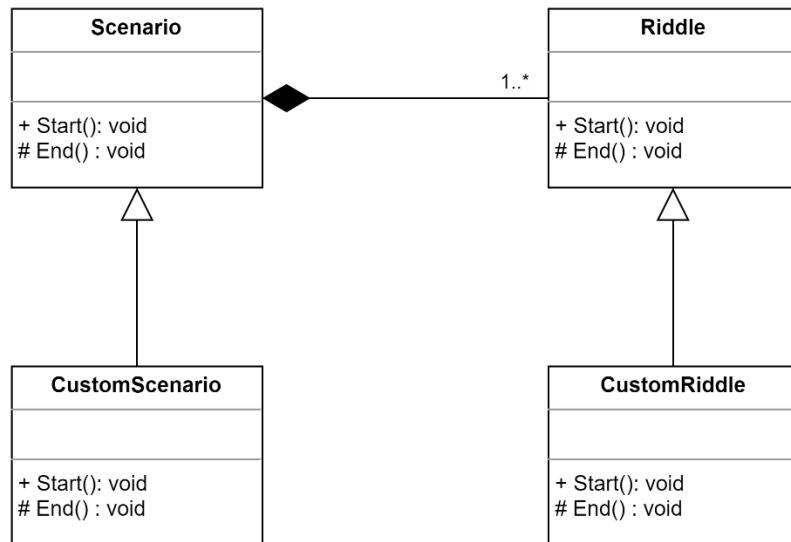


Figure 24 Diagramme de classe simplifié démontrant la structure avec héritage

Comme illustré sur le diagramme, il est envisageable de créer des sous-classes personnalisées afin d'ajouter la logique spécifique à un scénario. Le corps de la méthode pourrait être codé dans la classe parente pour implémenter la logique réseau. Cette première méthode est fonctionnelle, mais elle impliquerait un nombre important de scripts différents dans le projet, ce qui pourrait entraîner une confusion.

La deuxième méthode consiste à utiliser des événements. Unity met à disposition ce mécanisme, qui permet de déclencher des fonctions à des moments opportuns. Cela facilite la gestion des interactions et des réactions du jeu. Pour donner un exemple, lors de l'appui sur un bouton, cela produit un événement de clic, auquel on peut réagir de la manière souhaitée.

L'idée est de mettre en place ce même mécanisme. On créerait un seul squelette, ayant les moments clés discutés précédemment, et lorsqu'un moment clé est atteint, il suffit de déclencher l'événement.

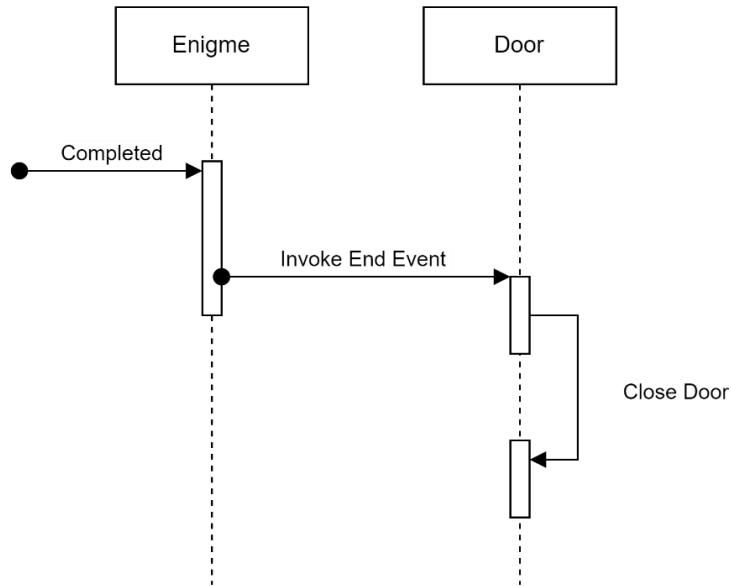


Figure 25 Exemple d'utilisation d'événements

Par exemple, dans la figure ci-dessus, un objet "Door" s'est enregistré à l'événement "End" de l'énigme. Lorsque l'énigme se termine, l'événement est déclenché, ce qui va à son tour activer la fonction souhaitée sur l'objet "Door", qui est de fermer la porte.

Cette deuxième solution a été choisie en raison de sa simplicité d'utilisation, même si elle implique une légère perte de performances lors de l'exécution. Elle permet d'éviter la répétition de code, voire d'écrire du code, car sur Unity, il est possible d'associer une fonction à un événement à la manière d'un glisser-déposer.

#### Lié la structure du scénario à la couche réseau

Un dernier point crucial à souligner est que la progression du scénario doit être synchronisée chez tous les clients. Pour cela, les trois scripts, à savoir Scenario, Riddle, et Element, doivent hériter de HostNetworkRPC, comme évoqué dans la section [Host Network RPC](#).

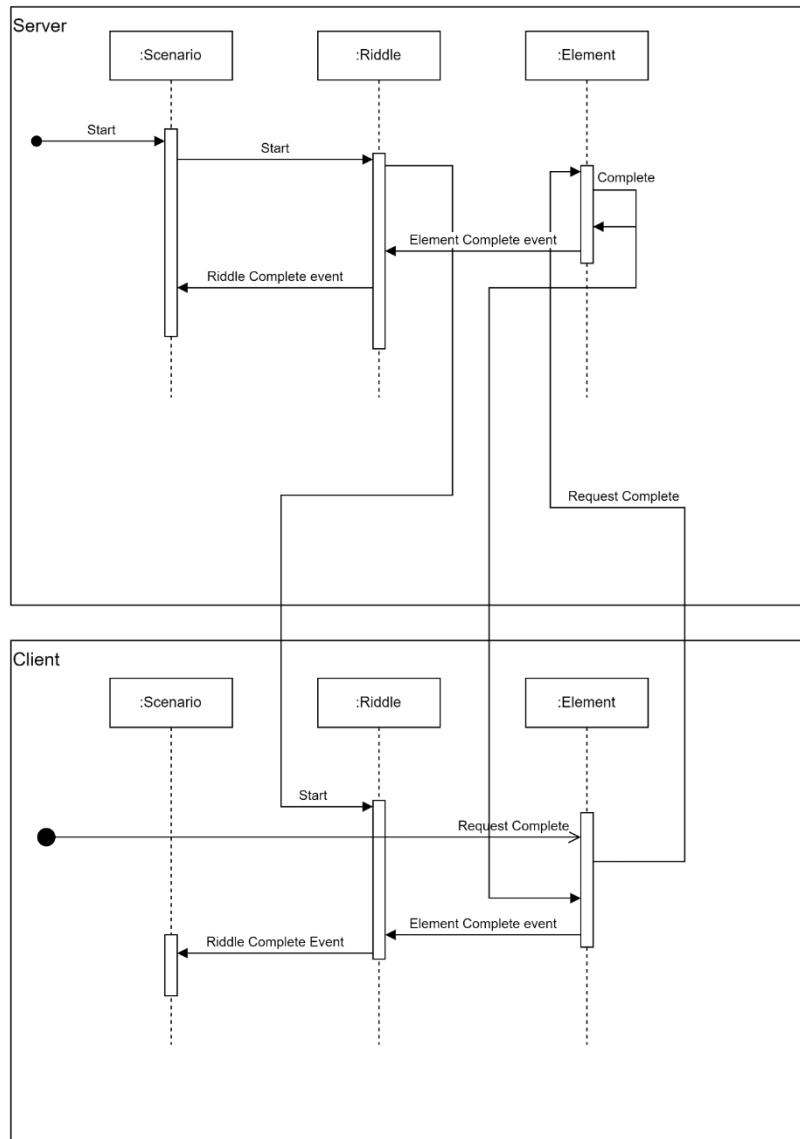


Figure 26 Diagramme de séquence simplifié montrant la structure du scénario avec les communications réseau

La figure ci-dessus peut sembler complexe, mais elle est en réalité assez simple à comprendre. L'objet scénario exécute exclusivement sa logique sur le serveur, garantissant ainsi que les énigmes commencent simultanément chez tous les participants. On observe également que lorsqu'une énigme démarre, elle le fait sur le serveur, qui commande ensuite par RPC aux clients de démarrer la même énigme.

Enfin, lorsqu'un élément est complété, on informe le serveur via RPC afin qu'il puisse réellement compléter l'élément ciblé chez tous les utilisateurs. Bien que la figure illustre le cas avec une seule énigme et un seul élément, l'augmentation du nombre de l'un ou l'autre n'altère en rien ce schéma, il sera simplement répété autant de fois que nécessaire.

## 2.4. Conception de la couche Gestion intelligente du jeu

La gestion intelligente du jeu requiert une dose importante d'imagination pour la rendre agréable. Du point de vue du développeur, il est essentiel de définir clairement comment l'avancement du jeu est évalué. Ainsi, des déclencheurs numériques sont indispensables pour confirmer le bon déroulement du scénario. Par exemple, si les joueurs ouvrent des cadenas physiques, il est impossible de savoir s'ils ont accompli l'étape correspondante. Par conséquent, trouver une méthode pour valider cette étape est impératif.

Cela nécessite une réflexion approfondie qui ne peut être négligée. Une possibilité serait d'utiliser la reconnaissance d'images, en intégrant des QR codes dans les zones restreintes, ce qui permettrait de détecter quand ces zones sont ouvertes. Cependant, cette solution n'est pas envisageable car Meta ne permet pas l'utilisation du flux vidéo.

Du point de vue des joueurs, il est crucial que les indications ne génèrent pas de frustration. Distribuer un indice immédiatement après la résolution d'une énigme pourrait décevoir les joueurs. Un système basé sur deux états, calculant la différence entre le temps prévu et le temps réel pour le scénario et l'éénigme en cours, pourrait atténuer cette frustration. En accordant plus d'importance à l'état de l'éénigme en cours, on s'assure que tout écart par rapport au timing prévu pour le scénario est modéré, évitant ainsi le risque d'envoyer des indices trop tôt lors du début d'une énigme.

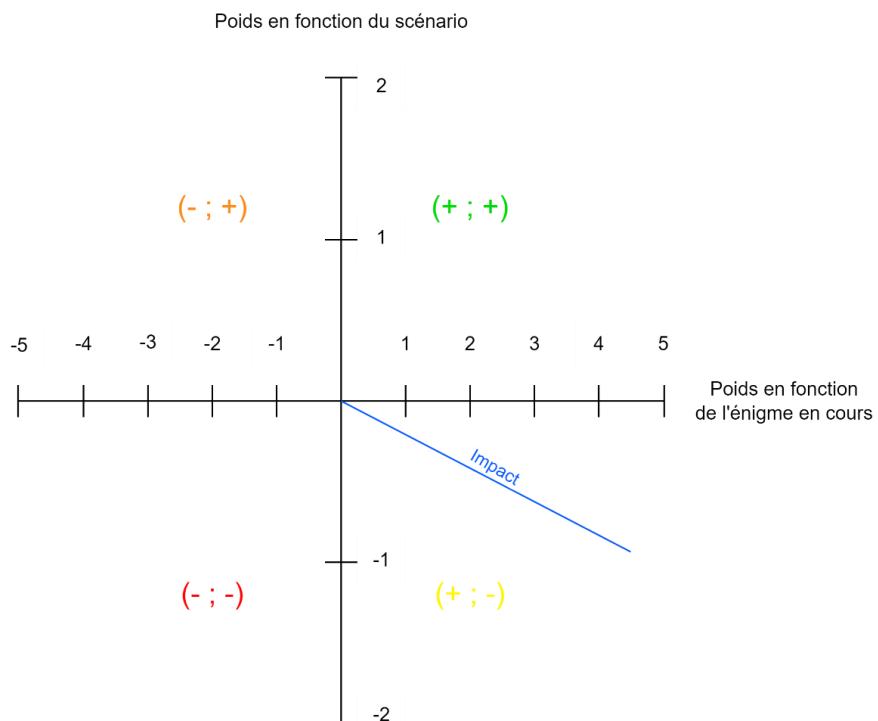


Figure 27 : Réflexion sur la manière d'influencer les joueurs

Ce système repose sur les principes suivants :

- Le scénario peut être en retard, dans les temps ou en avance.
- L'éénigme en cours peut être en retard, dans les temps ou en avance.

- Le différentiel de temps de l'énigme en cours a un impact plus significatif que celui du scénario.

Le retard ou l'avance sur le scénario doit simplement ralentir ou guider les participants. Ainsi, sa valeur varie de -2 à 2. Qu'il soit dans un extrême ou dans l'autre, il reste largement minoritaire par rapport au différentiel de l'énigme en cours.

D'un autre côté, l'impact du différentiel sur l'énigme en cours varie entre -5 et 5. L'objectif est d'aider ou de perturber le joueur. Une avance considérable sur une énigme ne doit pas entraîner la fourniture d'indices, même si, au niveau de l'ensemble du scénario, les joueurs sont en retard.

Cependant, il est crucial qu'en fonction de l'avance ou du retard sur une énigme seulement, toutes les influences puissent être exercées. Ainsi, des influences potentielles entre -7 et 7 sont envisageables, mais l'influence réelle se situe entre -5 et 5.

Cinq scénarios émergent de ce système, mais seuls trois seront décrits, les deux restants étant simplement une symétrie. Si le scénario et l'énigme sont dans les temps, aucune influence perturbatrice ou indice ne doit être envoyé. Définir précisément ce que signifie "dans les temps" sera une démarche empirique, basée sur des tests et des retours d'expérience.

Si le scénario est en avance mais que l'énigme est en retard, il est nécessaire de faire progresser l'énigme pour ne pas frustrer les joueurs. Cependant, il faut également leur laisser le temps de chercher, car ils disposent encore d'un délai. En utilisant le système présenté ci-dessus, des valeurs maximales peuvent varier entre -4 et 1, ce qui indique que cela atténue simplement légèrement l'impact des indices.

Le dernier scénario survient lorsque les deux variables sont en retard ou en avance. Dans ce cas, des indices ou des perturbateurs percutants doivent être envoyés. La variation d'impact se situe entre -1 et -7, mais avec une valeur réelle maximale de -5. Cela signifie qu'en combinant ces deux valeurs d'impact, on atteindra plus rapidement le maximum qui est de -5.

#### **2.4.1. Conception des objets permettant le système d'impact à deux états**

Il est crucial de noter que ce système fonctionne en fonction du temps. Par conséquent, il sera nécessaire de développer un gestionnaire qui centralisera toutes les informations nécessaires au bon fonctionnement de ce dispositif automatique. Ce gestionnaire sera doté des chronomètres et des paramètres de la simulation, et il aura la responsabilité de déterminer si les joueurs doivent être influencés.

En ce qui concerne les paramètres, il est important de souligner qu'ils doivent être simples à ajuster afin que le maître de jeu puisse les modifier avant le début d'une simulation. Ainsi, au lieu d'utiliser un temps défini pour chaque étape, il est préférable d'opter pour un poids. On pourra alors calculer le temps estimé pour chaque étape en fonction de son poids par rapport à celui de l'ensemble du scénario.

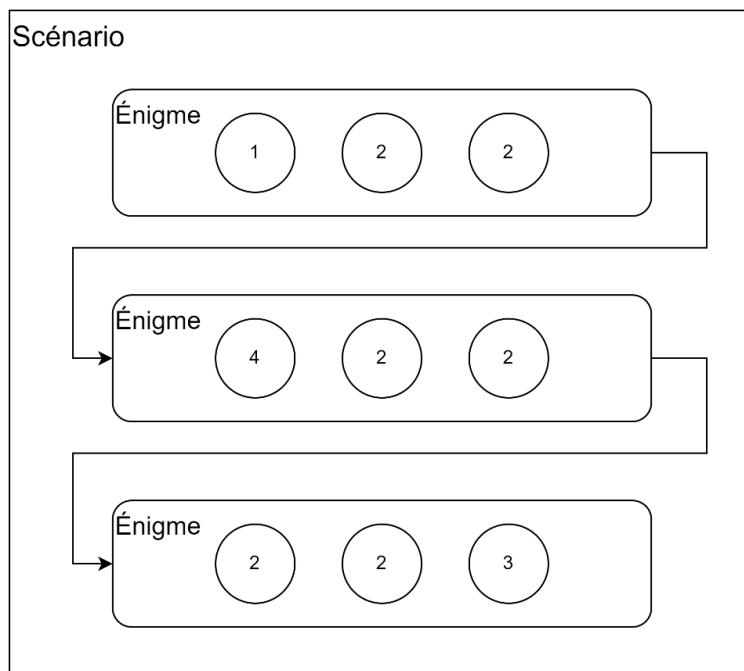


Figure 28 Système de poids pour paramétrer le scénario

En examinant le schéma de la structure du scénario, on remarque que des poids ont été attribués à chacun des éléments. On peut calculer que le poids total du scénario est de 20. Si l'on fixe une durée de simulation à 20 minutes, soit 1200 secondes, on peut déterminer combien de temps doit durer chaque unité de poids. Dans ce cas, cela correspond à une minute.

Par conséquent, on peut déduire que l'énigme 1 doit durer 5 minutes, la deuxième doit durer 8 minutes et que la troisième doit durer 7 minutes. Grâce à cela, on peut déterminer si le groupe est en retard ou en avance. Cette approche est également simple à configurer, car il suffit de modifier une valeur numérique « abstraite » pour chacun des éléments.

Il est également important de considérer l'implémentation d'états pour l'avancement du scénario et de l'énigme en cours. Ces états permettront de calculer le taux d'impact, basé sur le système discuté. Il est évident qu'il faudra tout d'abord attribuer des valeurs arbitraires aux niveaux d'impacts à retourner, et cela se fera par le biais de tests en situation réelle en recueillant le ressenti des participants.

Un avantage d'utiliser des états est leur interchangeabilité. Si l'on souhaite pousser la configuration encore plus loin que de simples poids, il est tout à fait possible de créer de nouveaux états, assignables par le maître de jeu en début de simulation, qui auront des courbes et des conditions différentes pour le retour d'impact.

## 2.5. Conception de la couche Indices et Perturbateurs

Cette couche est directement liée aux deux précédentes, à savoir le scénario et le gestionnaire intelligent du jeu. On peut regrouper les indices et les perturbateurs sous le terme générique « Influenceurs », car ils ne diffèrent que par leur impact positif ou négatif.

Pour rester en cohérence avec ce qui a été développé précédemment, il est essentiel que ces influenceurs aient un niveau d'impact allant de -5 à -1 pour les perturbateurs et de 1 à 5 pour les indices. Il est crucial de les rendre sémantiques, c'est-à-dire qu'ils doivent être utilisés uniquement lorsque l'élément associé n'a pas encore été complété.

Fondamentalement, les influenceurs ont uniquement besoin d'un moyen de déclenchement. Qu'ils soient textuels, sonores ou sous la forme d'une animation d'objet, ils doivent pouvoir être activés facilement. Pour simplifier cette implémentation, les événements se révèlent être la manière la plus efficace d'y parvenir.

En créant des « récepteurs » d'influence dédiés à chacun des moyens interactifs, il est possible de simplifier énormément la mise en place des indices. Il suffit d'inscrire l'événement sur le bon récepteur en lui passant le paramètre désiré, par exemple le texte ou l'image, pour que cela soit fonctionnel.

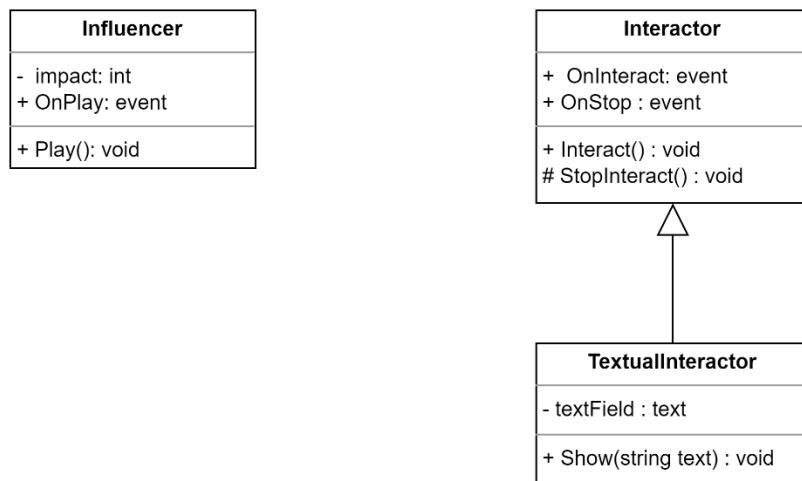


Figure 29 Diagramme de classes simple des influenceurs et des interacteurs

Pour illustrer le concept, prenons l'exemple où la méthode « Show » de l'objet « TextualInteractor » est enregistrée pour être utilisée avec le paramètre « 0000 est le code du cadenas ». Lorsque l'événement est déclenché, l'interacteur va directement exécuter sa fonction avec le paramètre que nous avons prédefini, faisant ainsi afficher l'indice souhaité.

## 2.6. Conception des Interactions en réalité augmentée

Les fonctionnalités d'interaction sont déjà intégrées dans le SDK de Meta. Cependant, il est possible d'ajouter une surcouche pour faciliter leur utilisation dans un projet spécifique. C'est précisément ce qui a été conçu à travers des objets génériques pouvant être réutilisés à chaque itération définie du projet.

Pour ce projet, diverses interactions sont nécessaires :

- Observer un objet ;
- Toucher (ou « Poke ») un objet ;
- Déplacer un objet ;
- Attraper un objet.

De manière plus détaillée, observer un objet implique de déclencher un événement lorsque celui-ci entre dans le champ de vision et est regardé pendant un certain temps. Toucher correspond à l'appui sur un bouton, réalisé avec un index. Bien que Meta propose déjà ce type d'interaction, aucun événement de "clic" sur le bouton n'est directement disponible, et son inclusion serait pertinente pour faciliter la conception d'un scénario.

Déplacer et attraper un objet peuvent sembler similaires, mais ce sont des actions distinctes. L'une consiste à fermer réellement la main sur l'objet et à le déplacer à volonté, tandis que l'autre implique une action physique de « pousser ». L'interaction pour attraper des objets est déjà présente dans le SDK de Meta, mais une fois de plus, aucun événement n'indique si l'objet a été attrapé, s'il est toujours dans les mains de l'utilisateur ou s'il a été relâché.

En résumé, l'objectif est de concevoir des interfaces simplifiées pour le SDK de Meta, offrant des méthodes prêtes à l'emploi pour accéder aux différents événements liés à l'interaction avec un objet..

## 2.7. Conception de la couche Monitoring

La couche de monitoring du projet existant satisfaisait déjà aux exigences spécifiées pour ce projet. Afin de gagner du temps, il a été décidé de reprendre en grande partie ce qui avait été réalisé et de l'adapter au nouveau projet.

Il est toutefois intéressant de discuter de certains points clés utilisés, notamment l'asset FMETP\_Stream qui propose un "Game View Encoder" et un "Game View Decoder". Ces scripts, comme leurs noms l'indiquent, ont pour fonction d'encoder et de décoder une vue. Il suffit de leur spécifier quelle caméra encoder et d'envoyer le flux de bytes à la destination souhaitée. Ces objets fonctionnent par paire, au moyen d'un ID, permettant ainsi d'avoir plusieurs encodeurs et décodeurs pour gérer plusieurs flux simultanément.

Le monitoring permet toujours d'envoyer des messages et de prendre des notes. Cependant, la partie perturbateurs et indices a été supprimée, remplacée par la gestion intelligente du scénario. En conséquence, l'interface a été revue pour être aussi naturelle que possible à utiliser.

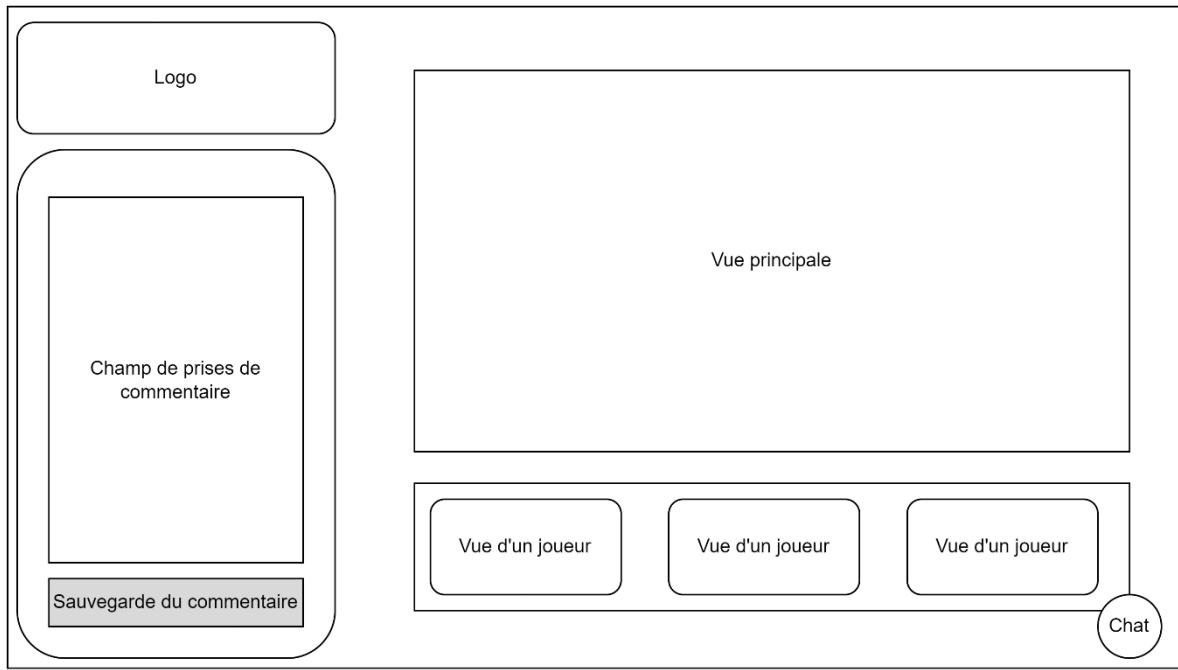


Figure 30 Mock up de l'interface en simulation

Le bouton situé en bas à droite du schéma déclenche l'ouverture d'une fenêtre modale équipée d'un champ texte et d'un bouton, permettant d'envoyer un message aux participants.

La principale nouveauté en matière de surveillance réside dans la possibilité de paramétrier un scénario, désormais accessible grâce au gestionnaire intelligent. Il a donc été nécessaire de concevoir une interface permettant au maître de jeu d'ajuster librement les paramètres selon ses préférences.

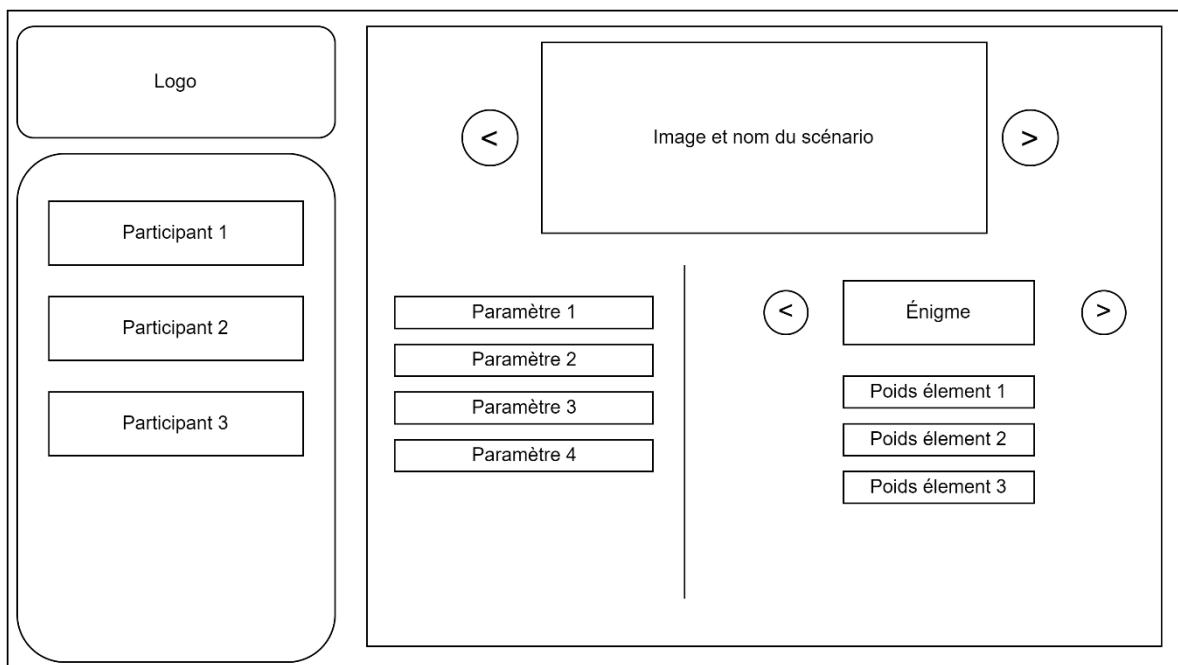


Figure 31 Interface permettant le paramétrage de la simulation

On observe que l'interface est divisée en deux parties distinctes : l'une dédiée à la visibilité des joueurs connectés et l'autre réservée au choix et à la configuration du scénario ainsi que de ses énigmes.

## 2.8. Conception de la couche Feedback

Après avoir pleinement compris les mécanismes de génération de feedback dans le projet existant, il apparaît que des simplifications sont envisageables. Tout d'abord, en évitant l'utilisation d'une base de données, car les commentaires stockés n'ont aucune utilité au-delà de la simulation en cours. Par conséquent, il semble plus judicieux de les stocker simplement, ainsi que l'image et le timestamp, dans une liste, offrant ainsi une solution plus simple à mettre en œuvre. Cependant, cela implique que la modification du rapport PDF après la simulation n'est plus envisageable.

Il a donc été décidé que le rapport serait automatiquement généré à la fin de chaque simulation, tout comme la vidéo récapitulative. Concernant la génération du PDF, le code existant dans le projet actuel a été repris pour gagner du temps, étant déjà fonctionnel. Il a simplement été adapté pour ne plus dépendre de la base de données, mais plutôt de la liste nouvellement créée. La capture vidéo, quant à elle, est réalisée à l'aide d'un codec externe FFMPEG spécialement adapté à Unity. Étant donné que ce système fonctionne comme souhaité dans le projet existant, il a été intégré de la même manière.

En ce qui concerne les sous-titres, l'utilisation des commentaires et des timestamps permet de créer un fichier .srt contenant les informations nécessaires, à savoir le moment d'affichage et leur contenu.

## 2.9. Conception de l'ancrage du monde virtuel

Ancrer le monde virtuel dans le monde réel peut sembler simple, mais cela ne l'est pas. Dans le projet existant, le monde suivait le regard jusqu'à ce que le joueur appuie sur un bouton pour le fixer, ce qui pouvait entraîner des problèmes, car le monde virtuel pouvait être incliné par rapport au monde réel.

Heureusement, le Meta Quest 3 offre la possibilité de se repérer dans l'espace et de scanner la pièce pour placer des objets virtuels au-dessus d'éléments réels. Un concept découle de cette fonctionnalité. Si le casque est capable de reconnaître le sol et le centre de la pièce, il devient possible de positionner le monde virtuel correctement, ainsi qu'à la bonne hauteur.

Cependant, subsiste le problème de la rotation. Il est nécessaire de s'assurer que le monde a la même orientation pour tous les joueurs. Pour résoudre cela, on peut demander aux joueurs de pointer une flèche dans une direction spécifique, qui est un point fixe du monde réel. En obtenant la rotation de cette flèche, il devient possible de déterminer la rotation nécessaire pour que le monde soit aligné de la même manière pour tous les joueurs.

### 3. IMPLÉMENTATION ET PROBLÈMES RENCONTRÉS

Ce chapitre expose les différentes mises en œuvre réalisées à partir des concepts discutés dans la section précédente. Il sera subdivisé en sous-sections correspondant à chaque couche du projet. Nous aborderons également les défis rencontrés lors de l'implémentation de ces différentes couches, ainsi que les choix qui ont été nécessaires pour les surmonter.

#### 3.1. Implémentation de la couche réseau

Les trois scripts abordés lors de la conception ont été implémentés avec succès. Ils sont fonctionnels et faciles à utiliser, comme prévu. Le script HostNetworkManager a été converti en singleton, ce qui permet un accès facile sans créer de dépendances fortes entre tous les objets de l'application. Étant donné qu'il n'y a aucune raison d'avoir plusieurs managers, cette modification n'a posé aucun problème lors du développement et a même simplifié la structure.

Le script HostNetworkManager est responsable de la découverte de tous les objets synchronisés. Cependant, il a été décidé que ces objets seraient enregistrés au moment du démarrage du scénario. Cette décision a été motivée par le fait qu'un client pouvait charger la scène avant le serveur, ce qui entraînait des problèmes de synchronisation des objets. En effet, par défaut, les objets synchronisés étaient placés en (0,0,0) avec une rotation identité et une échelle de (1,1,1). Il était alors possible que ces objets soient déplacés, redimensionnés ou tournés incorrectement. Pour éviter cela, les objets sont désormais enregistrés une fois que le scénario est lancé, c'est-à-dire lorsque tous les participants, y compris le serveur, sont prêts. Cette approche a résolu le problème de manière efficace.

```
private void RegisterNetworkObjects()
{
    HostNetworkObject[] networkObjects = FindObjectsOfType<HostNetworkObject>(FindObjectsSortMode.InstanceID);
    FMNetworkManager.instance.NetworkObjects = new GameObject[networkObjects.Length];
    int count = 0;
    foreach (HostNetworkObject networkObject in networkObjects)
    {
        FMNetworkManager.instance.NetworkObjects[count++] = networkObject.gameObject;
    }

    FMNetworkManager.instance.UpdateNumberOfSyncObjects();
}
```

Figure 32 Code permettant d'enregistrer les objets afin qu'ils deviennent synchronisés

Il est important de souligner qu'une méthode supplémentaire, nommée « UpdateNumberOfSyncObjects », a dû être ajoutée au code fourni par l'asset. Cela a été nécessaire pour pouvoir modifier la liste des objets synchronisés après le démarrage de l'application. En raison de cette modification nécessaire de l'asset, une autre adaptation a été apportée : l'utilisation des valeurs réelles du transform d'un objet plutôt que des valeurs par défaut.

```

public void UpdateNumberOfSyncObjects()
{
    NetworkTransform = new FMNetworkTransform[NetworkObjects.Length];
    for (int i = 0; i < NetworkTransform.Length; i++)
    {
        NetworkTransform[i] = new FMNetworkTransform();
        NetworkTransform[i].position = NetworkObjects[i].transform.localPosition;
        NetworkTransform[i].rotation = NetworkObjects[i].transform.localRotation;
        NetworkTransform[i].localScale = NetworkObjects[i].transform.localScale;
    }
}

```

Figure 33 Code permettant de mettre à jour la liste des objets synchronisés

En ce qui concerne la demande des clients de mettre à jour un objet, le gestionnaire décode le message et recherche dans la liste des objets connectés l'objet approprié pour le mettre à jour. Cependant, il arrivait parfois que, en raison de la récursivité des objets Unity, cela dépasse la profondeur maximale lors de l'encodage ou du décodage en JSON. Étrangement, cela ne se produisait pas à chaque itération, et comme ce n'était pas un problème majeur, il a été décidé de l'ignorer. Étant donné que les objets sont synchronisés soixante fois par seconde, il n'est pas grave si l'une de ces synchronisations n'est pas effectuée. Il serait cependant possible de créer notre propre extension de la librairie Newtonsoft pour sérialiser et déserialiser les objets en évitant cette récursion.

```

public void HandleObjectSyncMessage(HostNetworkMessage message)
{
    if (!IsServer()) return; // Client can't update network objects

    try
    {
        HostNetworkObjectTransform objectTransform = HostNetworkTools.DeserializeObjectTransform(message.Data);

        FMNetworkManager.instance.NetworkObjects.FirstOrDefault(x => x.GetComponent<HostNetworkObject>().Id == objectTransform.Id)
            .GetComponent<HostNetworkObject>()
            .SetTransform(objectTransform);
    }
    catch (Exception e)
    {
        // Sometimes sync message bug but if it miss one, it will be resync later
        Debug.LogError(e.Message);
    }
}

```

Figure 34 Code permettant au serveur de satisfaire la demande de synchronisation d'un client

Le manager est également chargé d'envoyer les RPC à la cible désirée. Lorsque la cible n'est pas spécifiée, le serveur envoie le message à tous les clients, tandis que les clients l'envoient uniquement au serveur. On remarque également que cette fonction encode un objet HostNetworkRPCMessage en JSON, contenant toutes les données nécessaires au fonctionnement du RPC.

```

public void SendRPC(HostNetworkRPCMessage rpcMessage, string targetIP = null)
{
    string data = HostNetworkTools.SerializeRPCMessage(rpcMessage);
    HostNetworkMessage message = new HostNetworkMessage()
    {
        MessageType = HostNetworkMessageType.RPC,
        NetworkTarget = HostNetworkTarget.Server,
        Data = data,
    };

    if (targetIP != null)
    {
        FMNetworkManager.instance.SendToTarget(HostNetworkTools.SerializeMessage(message), targetIP);
        return;
    }

    if (IsServer())
    {
        FMNetworkManager.instance.SendToOthers(HostNetworkTools.SerializeMessage(message));
    }
    else
    {
        FMNetworkManager.instance.SendToServer(HostNetworkTools.SerializeMessage(message));
    }
}

```

Figure 35 Code de la fonction permettant d'envoyer des RPC

```

public class HostNetworkRPCMessage
{
    22 références
    public int InstanceId { get; set; }
    22 références
    public string MethodName { get; set; }
    33 références
    public object[] Parameters { get; set; }

    1 référence
    public HostNetworkRPCMessage ConvertInt64ToInt32Payload()...
}

```

Figure 36 Classe HostNetworkRPC et ses différents champs

On remarque qu'il existe une méthode permettant de convertir les int64 en int32. La bibliothèque utilisée pour encoder et décoder les données en JSON convertit par défaut tous les nombres en int64. Cependant, la plupart des programmes utilisent par défaut des nombres de type int32. Par conséquent, il a été nécessaire de les convertir pour qu'ils puissent être utilisés correctement.

En ce qui concerne la réception des RPC, le manager a la responsabilité de décoder le message et de le transmettre à l'objet ciblé, afin que celui-ci puisse exécuter la méthode correspondante.

```

public void HandleRPCMessage(HostNetworkMessage message)
{
    HostNetworkRPCMessage rpcMessage = HostNetworkTools.DeserializeRPCMessage(message.Data);
    HostNetworkRPC.GetRPCInstance(rpcMessage.InstanceId).HandleRPC(rpcMessage);
}

```

Figure 37 Code pour décoder et transmettre les RPC

La transition est toute trouvée pour aborder le script HostNetworkRPC. Comme le montre la figure 37 ci-dessus, il est possible d'accéder de manière statique à la liste des entités ayant des RPC. Cette liste est populée à chaque fois qu'un objet héritant de ce script est démarré. Évidemment, lorsqu'un de ces objets est détruit, par exemple lors d'un changement de scène, il est d'abord retiré de la liste pour éviter d'avoir un objet null.

En outre, le script contient également la fonction permettant d'exécuter la méthode reçue sur le réseau.

```
public void HandleRPC(HostNetworkRPCMessage message)
{
    MethodInfo method = this.GetType().GetMethod(message.MethodName, BindingFlags.NonPublic | BindingFlags.Public | BindingFlags.Instance);

    if (method.GetParameters().Length > message.Parameters.Length)
    {
        message.Parameters = message.Parameters.Append(HostNetworkManager.instance.IsServer()).ToArray();
    }

    method.Invoke(this, message.Parameters);
}
```

Figure 38 Code pour exécuter les RPC

Dans cette image, la première ligne revêt une importance particulière. En effet, en C#, il est possible d'utiliser la réflexion pour obtenir une méthode à partir de son nom sous forme de string. C'est précisément ce qui est accompli ici, où la méthode GetType() récupère le type de l'objet actuel, et donc la classe qui hérite de HostNetworkRPC, tandis que GetMethod() permet de récupérer les informations de la méthode. Ensuite, une fois les informations de la méthode récupérées, il suffit de l'appeler avec les paramètres reçus dans le message.

Plusieurs paramètres sont disponibles, notamment les BindingFlags, qui filtrent les méthodes à rechercher. Dans notre cas, nous recherchons les méthodes publiques et non publiques, tandis que le drapeau instance permet de vérifier les méthodes dans toutes les sous-classes.

Comme mentionné précédemment, pour utiliser cet outil, il suffit de faire hériter la classe désirée de HostNetworkRPC. Ainsi, toutes les méthodes de la classe deviennent utilisables à distance. Pour effectuer un RPC, il suffit d'appeler la méthode présente dans le Manager, en s'assurant que tous les paramètres sont correctement remplis.

```
HostNetworkManager.instance.SendRPC(new HostNetworkRPCMessage()
{
    InstanceId = this.InstanceId,
    MethodName = "StartRiddle",
    Parameters = new object[] { }
});
```

Figure 39 Code pour envoyer un RPC

Maintenant que nous avons abordé les RPC, concentrons-nous sur les objets synchronisés. Le script HostNetworkObject n'est pas complexe sur le plan logique, mais il s'avère extrêmement utile.

```

private void Update()
{
    if (!HostNetworkManager.instance.IsServer() && this.transform.hasChanged && IsSynced)
    {
        HostNetworkManager.instance.RequestObjectSync(this.GetTransform());
    }
}

```

Figure 40 Méthode Update des objets synchronisés

Unity offre une fonctionnalité appelée « hasChanged » sur les transform. Cette fonction permet de déterminer si un objet a été déplacé ou tourné. Ainsi, il suffit de surveiller si un objet a été déplacé et, dans ce cas, d'envoyer une demande de synchronisation au serveur. Il est important de noter que lorsque le serveur modifie les positions des objets synchronisés, la valeur de « hasChanged » devient également true. Pour remédier à cela, la fonction de l'asset est modifiée pour que cette propriété reste false lors d'une synchronisation.

Pour rendre un objet synchronisé, il suffit d'ajouter le script à un GameObject. Celui-ci sera automatiquement synchronisé avec les autres participants.

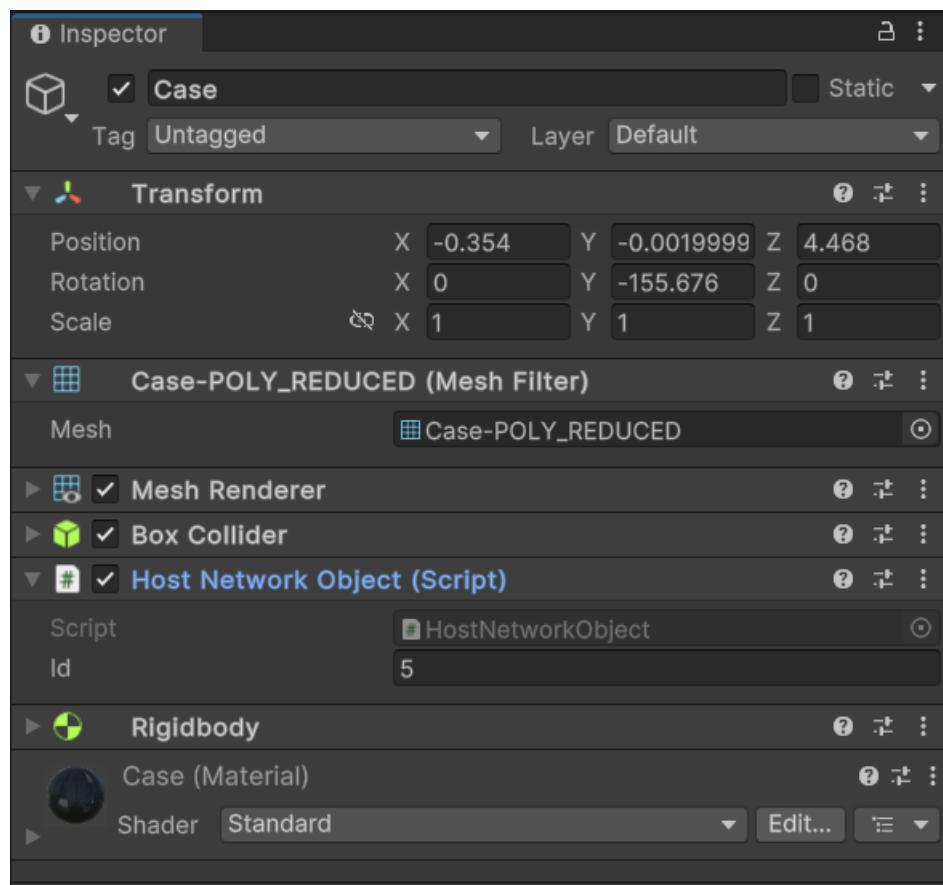


Figure 41 La vue de l'inspecteur pour la valise dans le scénario de l'avion

Il convient cependant de noter que le déplacement des objets chez les autres participants peut être légèrement saccadé. Pour le joueur qui déplace un objet, cela ne pose aucun problème, mais les autres

participants peuvent rencontrer une légère latence. Cet aspect n'est pas particulièrement problématique, car la simulation ne nécessite pas une réactivité élevée et cela n'affecte pas l'immersion dans l'expérience.

### 3.2. Implémentation de la couche Scénario

Grâce aux outils développés pour la couche réseau et à la conception élaborée de la structure générique, l'implémentation du squelette générique s'est déroulée sans difficulté notable. Les trois éléments principaux - le scénario, les énigmes et les éléments déclencheurs - ont été intégrés conformément à la conception préalable.

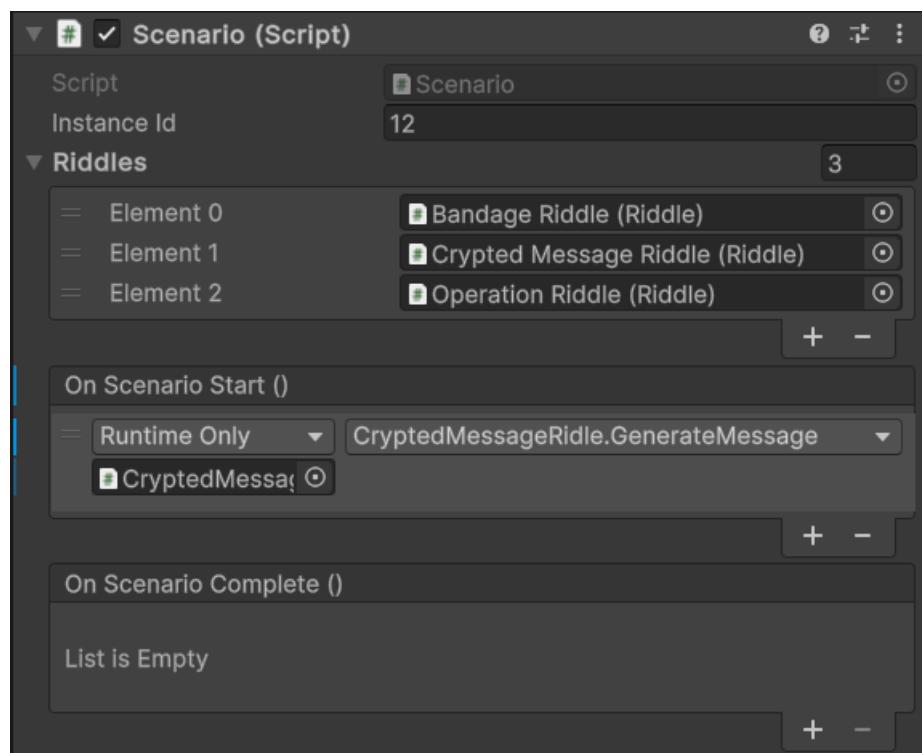


Figure 42 Inspecteur Unity représentant le script Scenario

Dans cette illustration, le scénario présent affiche trois énigmes distinctes. Une fonction est enregistrée pour être exécutée au démarrage du scénario. Il est également notable que le scénario possède un identifiant d'instance. Cette caractéristique découle du fait que le script Scenario.cs hérite de HostNetworkRPC, comme précédemment mentionné.

Tout ajout, suppression ou modification de l'ordre des énigmes se fait exclusivement depuis l'inspecteur. Cette approche démontre la simplicité de création d'un nouveau scénario. L'objet Scenario s'enregistre automatiquement aux événements des énigmes, ce qui permet de suivre leur progression et de savoir quand elles sont terminées.

```

public void StartScenario()
{
    if (!HostNetworkManager.instance.IsServer()) return;
    foreach (Riddle riddle in riddles)
    {
        riddle.onRiddleComplete.AddListener(OnRiddleComplete);
    }
    currentRiddleIndex = 0;
    StartRiddle();
    onScenarioStart.Invoke();
}

```

Figure 43 Code montrant que le scénario s'enregistre auprès des événements des énigmes

En commençant par le haut de l'image, on observe tout d'abord que le scénario s'enregistre à l'événement « OnRiddleComplete » de chacune de ses énigmes, puis démarre la première énigme et invoque l'événement « OnScenarioStart »), tel que représenté sur la vue de l'inspecteur dans la figure 42.

```

public void OnRiddleComplete(Riddle r)
{
    if (!HostNetworkManager.instance.IsServer()) return;
    if (r != riddles[currentRiddleIndex]) return;
    if (currentRiddleIndex == riddles.Count - 1)
    {
        onScenarioComplete.Invoke();
    }
    else
    {
        currentRiddleIndex++;
        StartRiddle();
    }
}

```

Figure 44 Code de la méthode OnRiddleComplete

Il est pertinent de se pencher à nouveau sur la méthode « OnRiddleComplete », qui est déclenchée lorsque les énigmes se terminent. La seconde condition revêt une importance particulière, car elle permet d'ignorer les énigmes qui se terminent alors qu'elles ne sont pas en cours.

Il n'y a pas lieu de s'inquiéter à cet égard, car la responsabilité de cette vérification est directement dévolue à l'énigme elle-même. Lorsqu'une énigme démarre, elle procède d'abord à une vérification pour s'assurer qu'elle n'est pas déjà terminée.

```

public void StartRiddle()
{
    if (HostNetworkManager.instance.IsServer())
    {
        HostNetworkManager.instance.SendRPC(new HostNetworkRPCMessage()
        {
            InstanceId = this.InstanceId,
            MethodName = "StartRiddle",
            Parameters = new object[] { }
        });
    }
    onRiddleStart.Invoke(this);
    if (IsCompleted())
    {
        onRiddleComplete.Invoke(this);
    }
}

```

Figure 45 Code de la méthode StartRiddle

Il est également à noter que le démarrage de l'énigme s'effectue sur le serveur, qui ensuite envoie un RPC à tous les clients pour démarrer la même énigme.

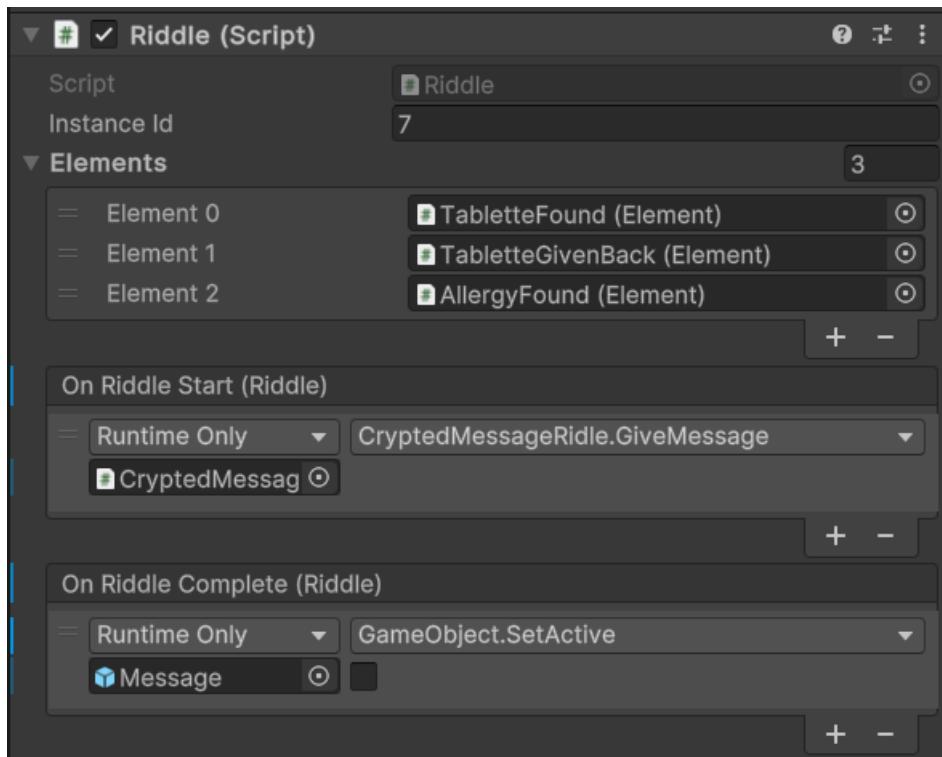


Figure 46 Inspecteur Unity représentant le script Riddle

La structure d'une énigme est essentiellement similaire à celle du scénario. Elle comprend une liste d'éléments déclencheurs ainsi que des événements de début et de fin d'éénigme. Dans cet exemple, lors du début de l'éénigme, la fonction « GiveMessage » est appelée. À la fin de l'éénigme, il est défini que le GameObject « Message » sera désactivé.

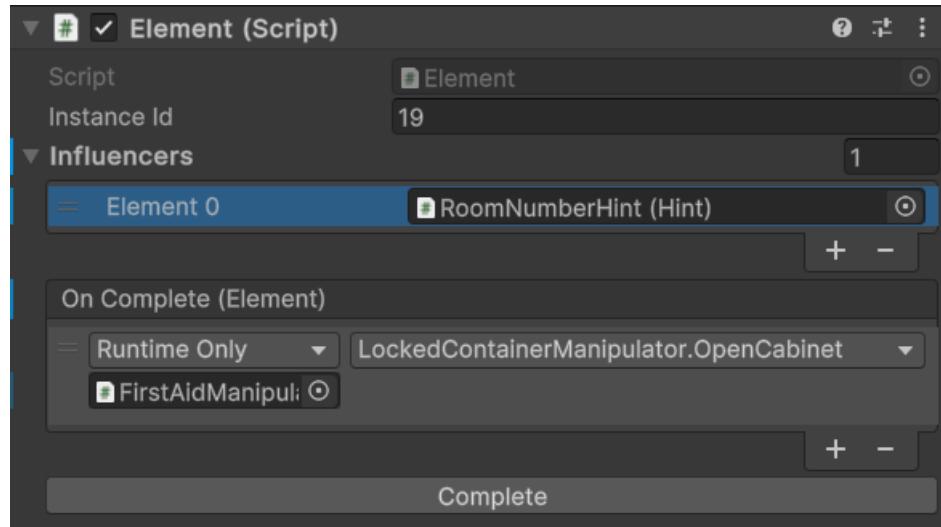


Figure 47 Inspecteur Unity représentant le script Element

Enfin, examinons les éléments paramétrables du script Element.cs. Cette fois-ci, on remarque qu'il n'y a pas d'événement de début ou de fin, mais seulement un événement pour la compléition. Lorsque cet élément est complété, la fonction OpenCabinet est exécutée. Cela permet d'effectuer une action chez tous les joueurs sans avoir besoin d'envoyer spécifiquement un RPC uniquement pour cette fonction. Ce mécanisme en réseau fonctionne pour tous les événements des trois scripts, simplifiant ainsi la synchronisation entre les participants.

On peut également remarquer la présence d'une liste d'influenceurs. Les influenceurs sont directement liés aux éléments déclencheurs, ce qui permet de pouvoir envoyer des indices uniquement pour des éléments qui ne sont pas encore terminés. L'implémentation des influenceurs sera expliquée plus en détail dans une section ultérieure.

Toute l'implémentation de cette couche n'a pas posé de problème particulier, à l'exception de la nécessité de veiller à ce que tous les éléments déclencheurs soient complétables. Sans cela, une énigme serait alors bloquée, et le scénario également.

Pour faciliter le développement, un bouton "Compléter" a été mis en place. Étant donné que les scénarios sont désormais ordonnés, il peut parfois être fastidieux de passer par toutes les étapes précédant celle que l'on souhaite déboguer. Ce bouton permet de compléter les éléments, facilitant ainsi le passage aux étapes suivantes. Il n'est bien sûr disponible que dans Unity, lorsque le scénario est en développement.

### 3.3. Implémentation de la couche Gestion intelligente de Scénario

Comme discuté lors de la phase de conception, pour faire fonctionner le gestionnaire intelligent de scénario, il est essentiel que les scénarios aient des paramètres. Trois paramètres principaux ont été identifiés, ainsi que deux autres secondaires. Tout d'abord, il y a la durée souhaitée de la simulation, le pourcentage de différence entre le temps estimé et le temps réel considérant que les joueurs sont toujours dans les temps, ainsi que le poids de chaque élément d'une énigme.

Comme expliqué dans la conception, ce système de poids simplifie la répartition du temps pour chacune des étapes de la simulation. Ainsi, il est possible de calculer les temps estimés pour chaque étape, ce qui permet de déterminer où les joueurs devraient être à un moment donné.

En connaissant l'avance souhaitée, on peut supposer que si l'avance réelle est dans un certain pourcentage par rapport à l'avance souhaitée, les joueurs sont dans les temps. Ce pourcentage est donc un paramètre important car il influence fortement l'envoi d'indices et de perturbateurs.

Les deux paramètres secondaires sont le temps minimal entre chaque indice et chaque perturbateur. En définissant une durée souhaitée basse, les indices pourraient être envoyés rapidement, et pour pallier à ce problème, l'ajout de ces deux paramètres permettrait d'instaurer une limite fixe.

Initialement, ces paramètres étaient définis directement dans le script Scenario.cs ainsi que Riddle.cs pour les poids. Cependant, lors du développement de la couche de surveillance et du choix de scénario, il est apparu que cette solution n'était pas adaptée. En effet, il devenait impossible de les modifier par le maître de jeu juste avant une simulation, et seules les valeurs par défaut définies par le développeur pouvaient être utilisées.

Pour remédier à ce problème, il a été nécessaire de séparer les données des paramètres du scénario lui-même. Ainsi, il suffisait de passer les paramètres entre la scène de choix et la scène de jeu pour que cela fonctionne. De nouveaux scripts ont donc été créés pour résoudre cette problématique.

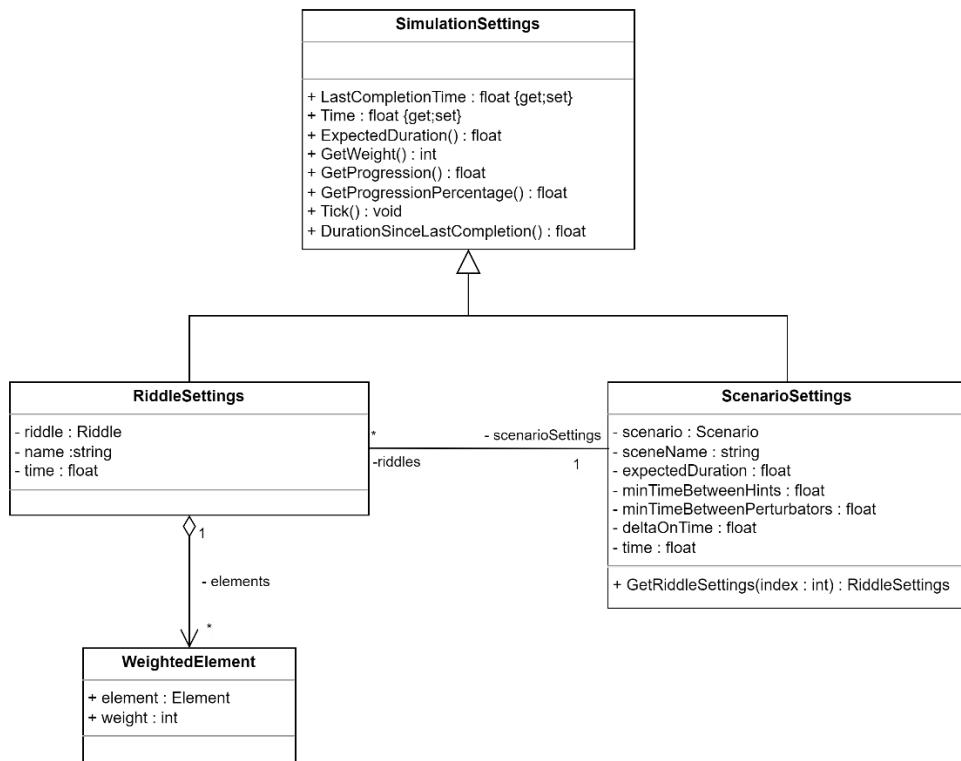


Figure 48 Diagramme de classe des scripts contenant les paramètres

Ces paramètres ont été définis dans des objets, ScenarioSettings et RiddleSettings. Ces objets sont capables de calculer leur temps estimé alloué, de suivre leur progression et de connaître le temps écoulé

depuis le début du démarrage de leur objet associé. Ces deux scripts héritent de `SimulationSettings`, qui possède les méthodes de base utilisées pour le calcul du niveau d'influence. Chacun des deux scripts spécifiques possède également une référence au scénario ou à l'énigme auquel il est lié, permettant ainsi de générer automatiquement les éléments de chacune des listes dans l'inspecteur ainsi que dans l'interface de surveillance, comme nous le verrons dans une section ultérieure.

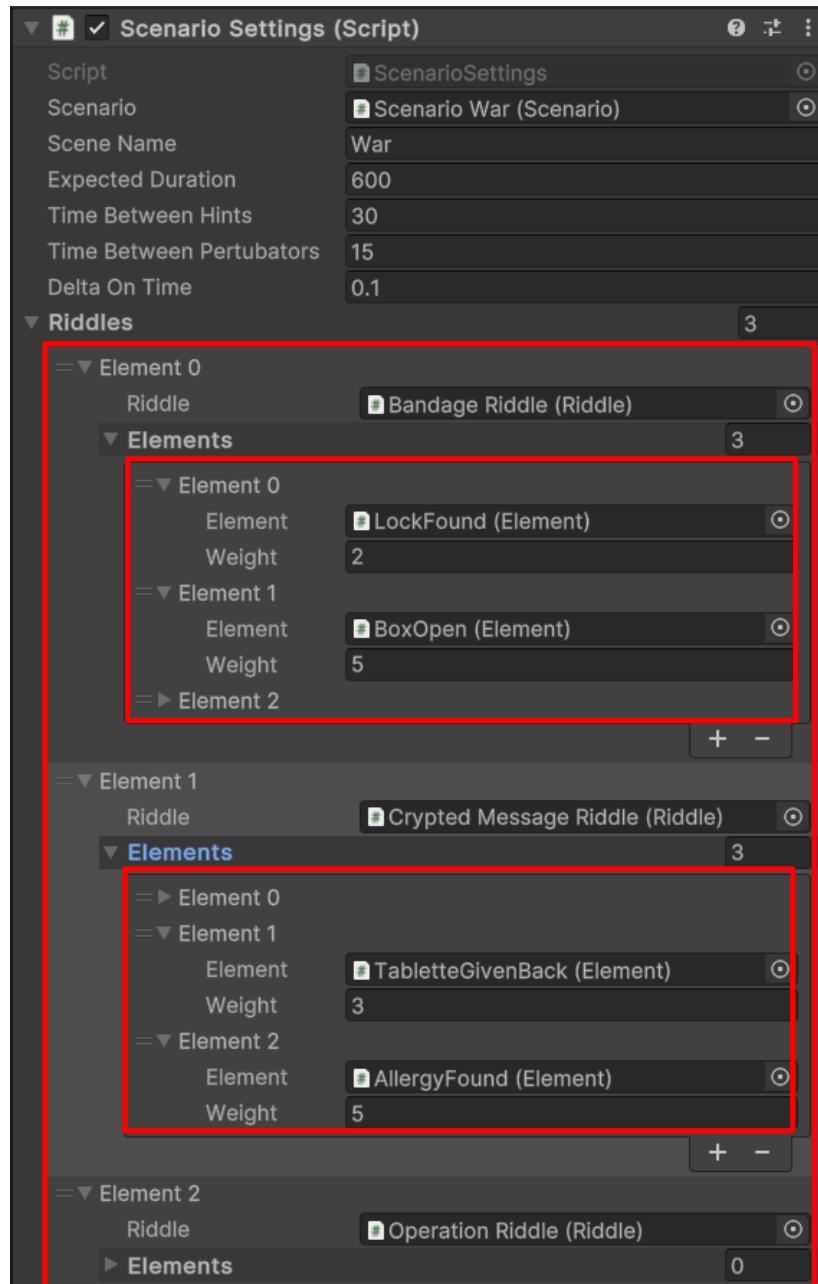


Figure 49 Vue des paramètres d'un scénario sur l'inspecteur Unity

Chacun des éléments encadrés en rouge a été généré automatiquement, simplifiant ainsi le processus. Il ne reste plus qu'à ajuster les poids selon les besoins spécifiques. Dans l'inspecteur, les valeurs par défaut de chaque paramètre sont définies conformément à la vision du concepteur du scénario.

Nous aborderons dans la partie [Implémentation de la couche Monitoring](#) comment le maître de jeu peut modifier ces paramètres selon ses préférences.

Un autre aspect important de la gestion intelligente à discuter concerne les états, qui permettent de calculer le niveau d'influence à envoyer. Ces états utilisent les informations fournies par les paramètres pour évaluer la valeur de l'impact.

```
int influenceLevel = currentScenarioState.ComputeInfluenceLevel(settings) +
                     currentRiddleState.ComputeInfluenceLevel(settings.GetRiddleSettings(currentRiddleIndex));

if (influenceLevel != 0)
{
    if (influenceLevel > 0 && settings.Time - lastHintTime >= settings.TimeBetweenHints)
    {
        settings.Scenario.PlayInfluence(influenceLevel);
        lastHintTime = settings.Time;
    }
    else if (influenceLevel < 0 && settings.Time - lastPertubatorTime >= settings.TimeBetweenPertubators)
    {
        settings.Scenario.PlayInfluence(influenceLevel);
        lastPertubatorTime = settings.Time;
    }
}
```

Figure 50 Calcul du niveau d'influence dans le ScenarioManager

La première ligne effectue une addition des deux niveaux d'influence calculés par les états, puis, en fonction des paramètres de fréquence, détermine si un événement doit être envoyé aux joueurs.

Tous les états héritent du script ProgressionState, qui contient une méthode abstraite, la fonction qui retourne le niveau d'influence, ainsi que deux méthodes utilitaires.

```
public abstract class ProgressionState
{
    7 références
    public abstract int ComputeInfluenceLevel(SimulationSettings settings);

    1 référence
    public float ExpectedProgression(SimulationSettings settings)
    {
        return settings.Time / settings.ExpectedDuration();
    }

    4 références
    public float DeltaProgression(SimulationSettings settings)
    {
        return settings.GetProgression() - ExpectedProgression(settings);
    }
}
```

Figure 51 Script de la classe ProgressionState.cs

```

public class OnTimeState : ProgressionState
{
    3 références
    public override int ComputeInfluenceLevel(SimulationSettings manager)
    {
        return 0;
    }
}

```

Figure 52 Script de la classe OnTimeState

```

public class RiddleLateState : ProgressionState
{
    3 références
    public override int ComputeInfluenceLevel(SimulationSettings settings)
    {

        float deltaRiddleProgression = DeltaProgression(settings);

        if (deltaRiddleProgression < -0.8f)
        {
            return 5;
        }
        else if (deltaRiddleProgression < -2f / 3f)
        {
            return 4;
        }
        else if (deltaRiddleProgression < -0.5f)
        {
            return 3;
        }
        else if (deltaRiddleProgression < -1f / 3f)
        {
            return 2;
        }
        else if (deltaRiddleProgression < -0.25f)
        {
            return 1;
        }

        return 0;
    }
}

```

Figure 53 Script de la classe RiddleLateState.cs

La différence de progression entre celle actuelle et celle souhaitée est calculée, et en fonction de paliers, un indice d'influence différent est retourné. Cette approche est probablement la plus simple, mais il serait également envisageable d'écrire de nouveaux états utilisant une fonction mathématique, probablement d'ordre impair, pour calculer la valeur de l'impact. Dans ce cas, les valeurs ont été testées de manière empirique et définies en fonction du ressenti de ces différents tests.

```

private ProgressionState currentScenarioState;
private ProgressionState currentRiddleState;

[SerializeField]
private ProgressionState scenarioEarlyState = new ScenarioEarlyState();
[SerializeField]
private ProgressionState scenarioLateState = new ScenarioLateState();
[SerializeField]
private ProgressionState scenarioOnTimeState = new OnTimeState();

[SerializeField]
private ProgressionState riddleEarlyState = new RiddleEarlyState();
[SerializeField]
private ProgressionState riddleLateState = new RiddleLateState();
[SerializeField]
private ProgressionState riddleOnTimeState = new OnTimeState();

```

Figure 54 États disponibles dans le script ScenarioManager.cs

Dans l'image ci-dessus, on remarque que deux états sont utilisés pour suivre l'état actuel de la simulation, tandis que les autres servent à paramétriser les trois options pour chaque état. Par conséquent, il serait également envisageable de rendre ces états paramétrables par le maître du jeu avant le démarrage de la simulation.

### 3.4. Implémentation de la couche indices et perturbateurs

La conceptualisation de cette partie peut être un peu complexe à saisir, c'est pourquoi des exemples d'utilisation seront bénéfiques pour une meilleure compréhension de sa mise en œuvre. Tout d'abord, il est important de rappeler que les influenceurs, à savoir les indices et les perturbateurs, disposent d'un événement « Play » et d'un niveau d'influence.

```

public abstract class Influencer : HostNetworkRPC
{
    [SerializeField]
    private UnityEvent play;

    5 références
    public abstract int GetInfluenceLevel();
    1 référence
    public void Play()
    {
        play.Invoke();
    }
}

```

Figure 55 Script Influencer.cs

Ainsi, les influenceurs ne comportent pas directement l'indice ou la perturbation eux-mêmes, mais seulement leur niveau d'impact. L'indice ou le perturbateur est simplement un paramètre de l'enregistrement d'une méthode d'un interacteur.

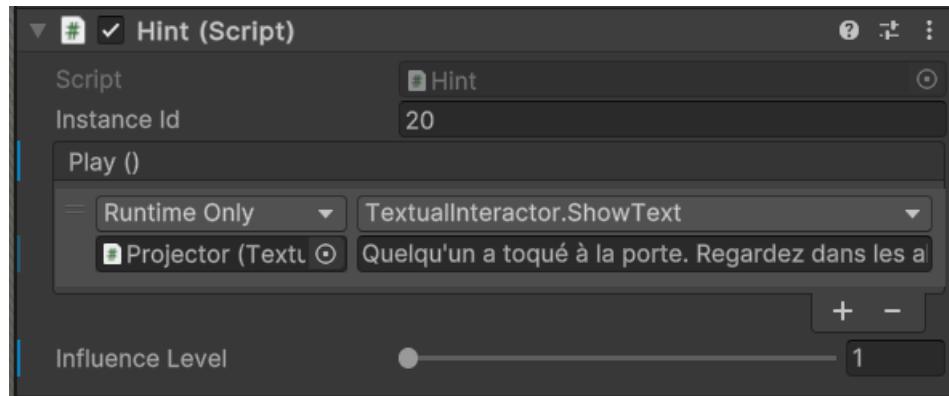


Figure 56 Inspecteur Unity permettant de visualiser le script Hint

Le script Hint.cs, présenté ci-dessus, hérite du script Influencer.cs. Il autorise des niveaux d'influence compris entre 1 et 5, assurant ainsi que le gestionnaire de scénario le considère comme un indice. L'aspect intéressant de ces influenceurs réside dans l'événement "Play". Comme mentionné précédemment, c'est en enregistrant la méthode à cet événement "Play" que l'on associe réellement l'indice.

Dans cet exemple, il s'agit d'un interacteur textuel, indiquant que l'indice sera de type texte. On peut également voir le texte qui sera affiché, à savoir : « Quelqu'un a toqué à la porte. Regardez dans les alentours ».

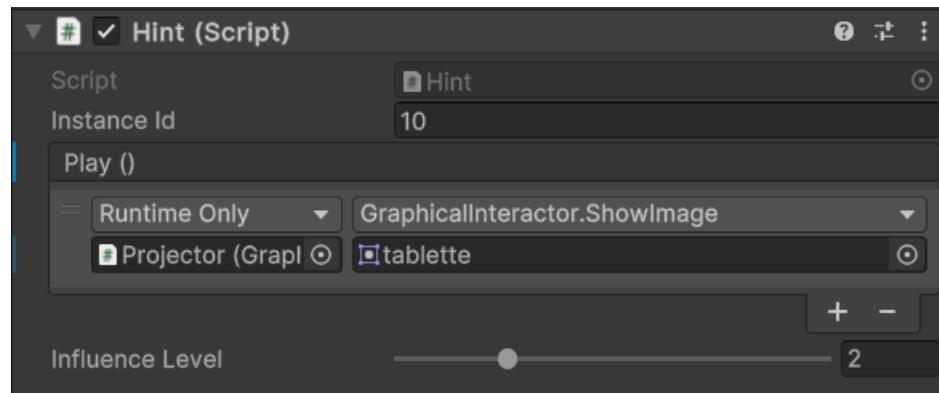


Figure 57 Exemple d'indice affichant une image

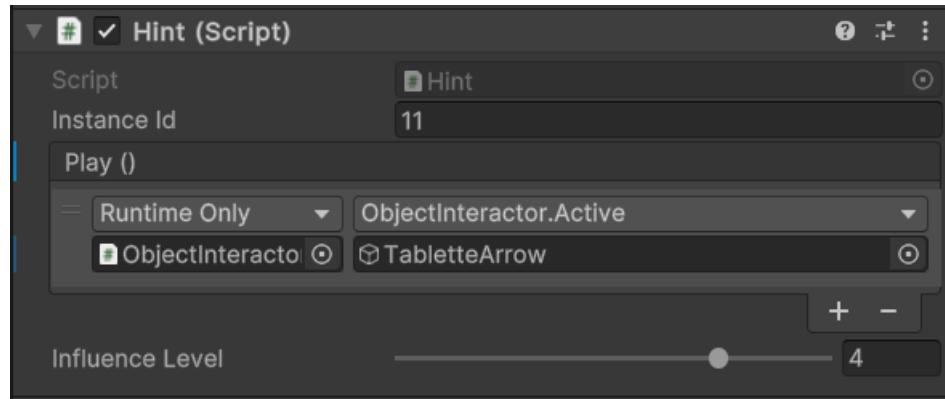


Figure 58 Exemple d'indice affichant un GameObject

Dans les exemples présentés ci-dessus, on remarque que peu importe le type d'indices ou de perturbateurs que l'on souhaite afficher, le mécanisme reste le même : passer en paramètre d'une méthode d'un interacteur le bon type pour créer l'indice. Sur la figure 50, une image est transmise en paramètre à l'interacteur graphique. Sur la figure 51, un GameObject est passé en paramètre à l'interacteur objet.

Créer un indice ou une perturbation n'a jamais été aussi simple, à condition d'avoir un interacteur. Heureusement, les interacteurs sont également très simples à mettre en place.

```
public class TextualInteractor : InfluenceInteractor
{
    [SerializeField]
    private TMP_Text textMesh;

    1 référence
    public void ShowText(string text)
    {
        Interact();
        textMesh.text = text;
    }
    6 références
    protected override void StopInteraction()
    {
        base.StopInteraction();
        textMesh.text = string.Empty;
    }
}
```

Figure 59 Script TextualInteractor.cs

Prenons l'exemple de l'interacteur textuel, où l'on peut observer un champ texte destiné à afficher l'information. Ce dernier possède également une méthode appelée « ShowText » qui permet d'afficher l'indice. Il hérite également du script InfluenceInteractor, qui gère la mise en place des événements de début et de fin.

```

public UnityEvent OnInteract;
public UnityEvent OnStopInteraction;

5 références
protected virtual void Interact(float duration=25)
{
    OnInteract.Invoke();
    CancelInvoke("StopInteraction");
    Invoke("StopInteraction", duration);
}

10 références
protected virtual void StopInteraction()
{
    OnStopInteraction.Invoke();
}

```

Figure 60 Script Influencelnteractor.cs

Ce script offre deux événements permettant au développeur de scénario de simplifier l'affichage de l'indice. Dans un exemple concret, dans le scénario de l'hôpital désaffecté, lorsqu'un vieux projecteur démarre, le son du projecteur doit être activé et les rouages doivent tourner.

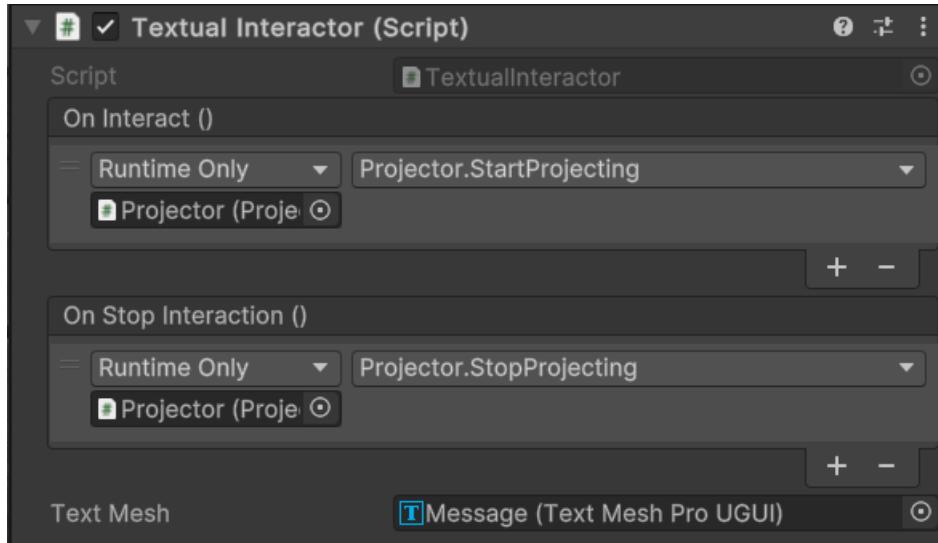


Figure 61 Inspecteur Unity affichant un interacteur textuel

Quand l'interaction démarre, le projecteur est activé pour émettre du bruit et tourner. À la fin de l'interaction, le bruit et la rotation s'arrêtent. Cette approche permet à tous les indices envoyés à cet interacteur d'allumer automatiquement le projecteur, sans nécessiter de code supplémentaire.

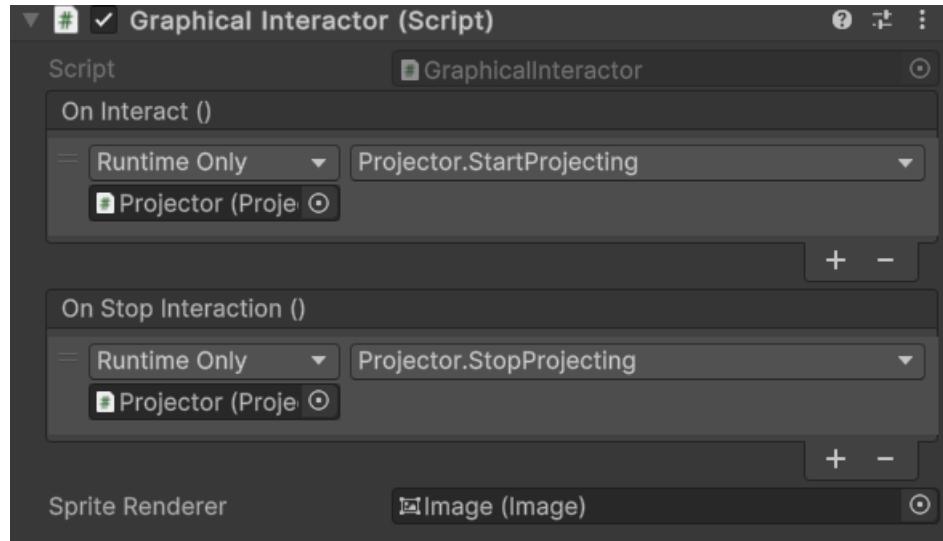


Figure 62 Inspecteur Unity affichant un interacteur graphique

Un inconvénient de cette approche est la nécessité de dupliquer les enregistrements d'événements si un même objet peut avoir plusieurs types d'interactions, comme c'est le cas pour le projecteur avec les interactions textuelles et graphiques. Malgré cela, grâce à ce système, la création d'un nouvel indice prend seulement quelques minutes, voire quelques secondes, à condition que toutes les ressources nécessaires soient disponibles (par exemple, avoir l'image déjà prête).

### 3.5. Implémentation des Interactions en réalité augmentée

Pour activer les interactions avec le Meta Quest 3, la première étape consiste à configurer l'objet représentant l'utilisateur. Ce GameObject, fourni par le SDK de Meta, intègre les caméras qui suivent les mouvements du joueur, ainsi que le suivi du corps et de la position dans l'espace.

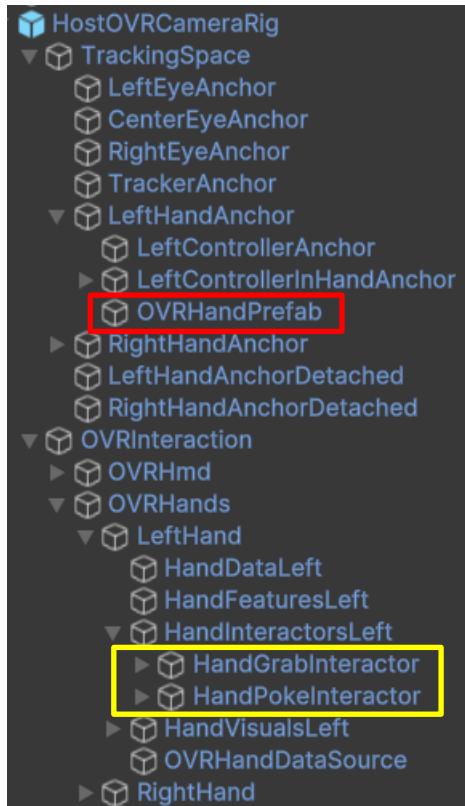


Figure 63 Hiérarchie de l'objet fourni par Meta

Le GameObject encadré en rouge a été ajouté au projet pour représenter les mains de l'utilisateur. Étant donné que les manettes peuvent également être utilisées, Meta n'a pas inclus les mains ni les manettes dans l'objet par défaut, donc il est nécessaire de les ajouter manuellement. Il est à noter que les deux méthodes, avec les mains et avec les manettes, peuvent être utilisées simultanément, bien que cela n'était pas pertinent pour ce projet et n'a donc pas été exploré.

De plus, il est nécessaire d'ajouter les possibilités d'interactions pour chaque main, comme encadré en jaune. Dans notre cas, nous avons ajouté l'interaction d'attraper et de toucher. Une fois ces interactions configurées, nous pouvons commencer à créer les objets nécessaires au projet.

Les interactions disponibles avec le Meta Quest 3 ont été simplifiées grâce aux prefabs fournis par Unity. Ce sont des objets réutilisables que l'on peut simplement glisser dans la scène. Chaque prefab correspond à une interaction, et tous les scripts nécessaires pour celle-ci sont déjà inclus.

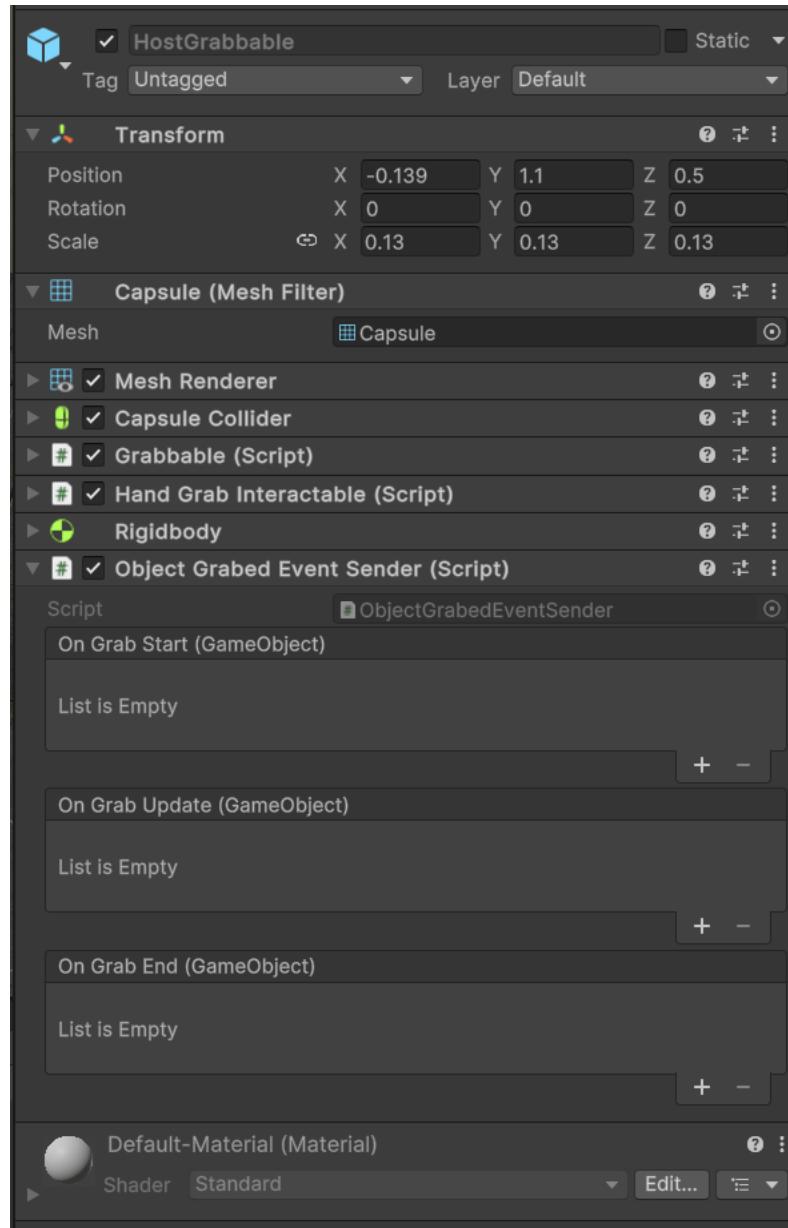


Figure 64 Prefab de l'objet attrapable

Une fois cet objet ajouté à la scène, il devient saisissable par les utilisateurs. Cela est rendu possible grâce aux scripts « Grabbable » et « Hand Grab Interactable » fournis par Meta. Nous avons également ajouté le script "Object Grabbed Event Sender" pour accéder aux différents événements que l'objet peut envoyer. Ainsi, la configuration du reste du scénario devient plus simple.

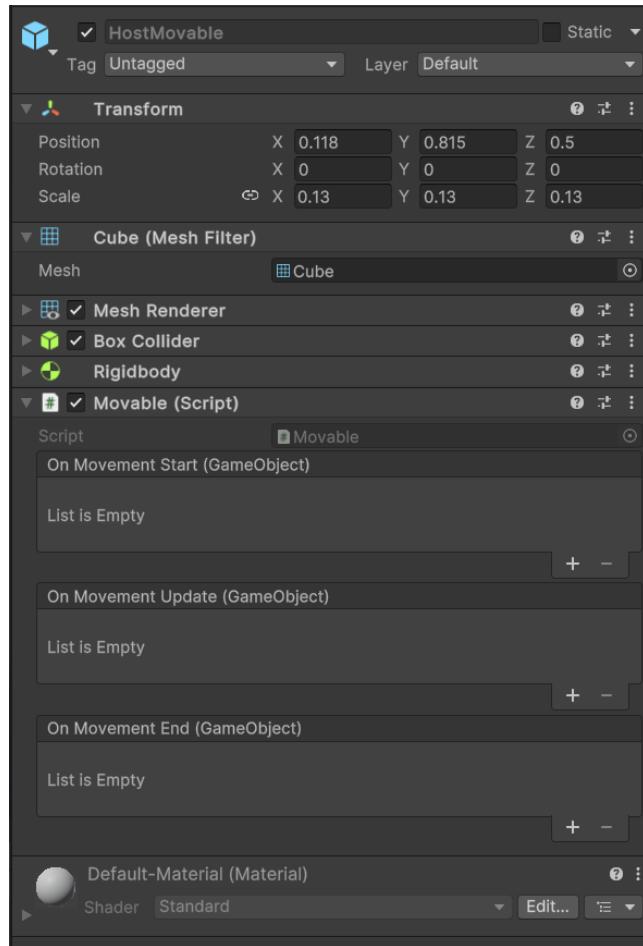


Figure 65 Prefab de l'objet déplaçable

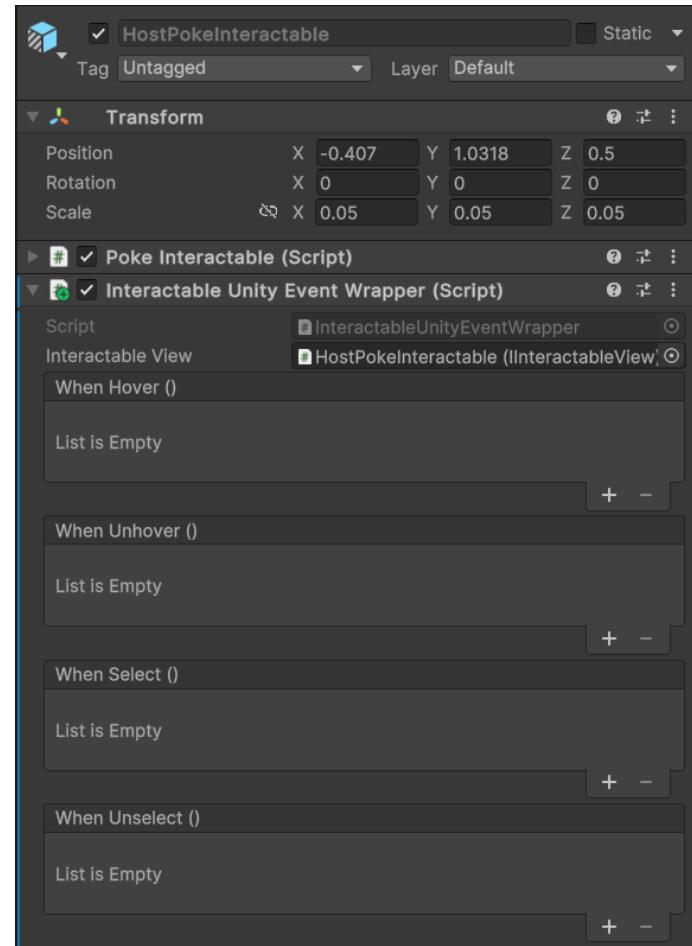


Figure 66 Prefab de l'objet touchable

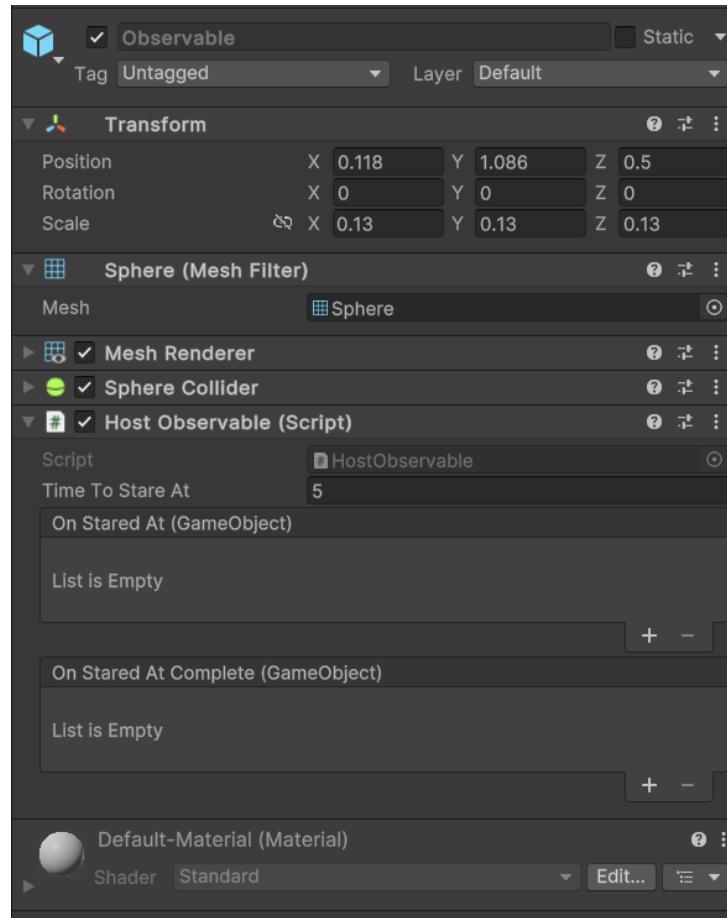


Figure 67 Prefab de l'objet regardable

Cette méthode demeure simple mais présente néanmoins un problème. Par défaut, des formes telles que des capsules, des sphères ou des cubes sont utilisées, ainsi que leur collider associé. Il est nécessaire de modifier chaque Mesh Renderer et chaque collider pour les adapter à l'objet que nous voulons afficher. Une autre possibilité consiste à placer le modèle de l'objet à l'intérieur de celui-ci, ce qui permet de déléguer l'affichage correct de l'objet au modèle ajouté en tant qu'enfant.



Figure 68 Hiérarchie du GameObject MissingBattery

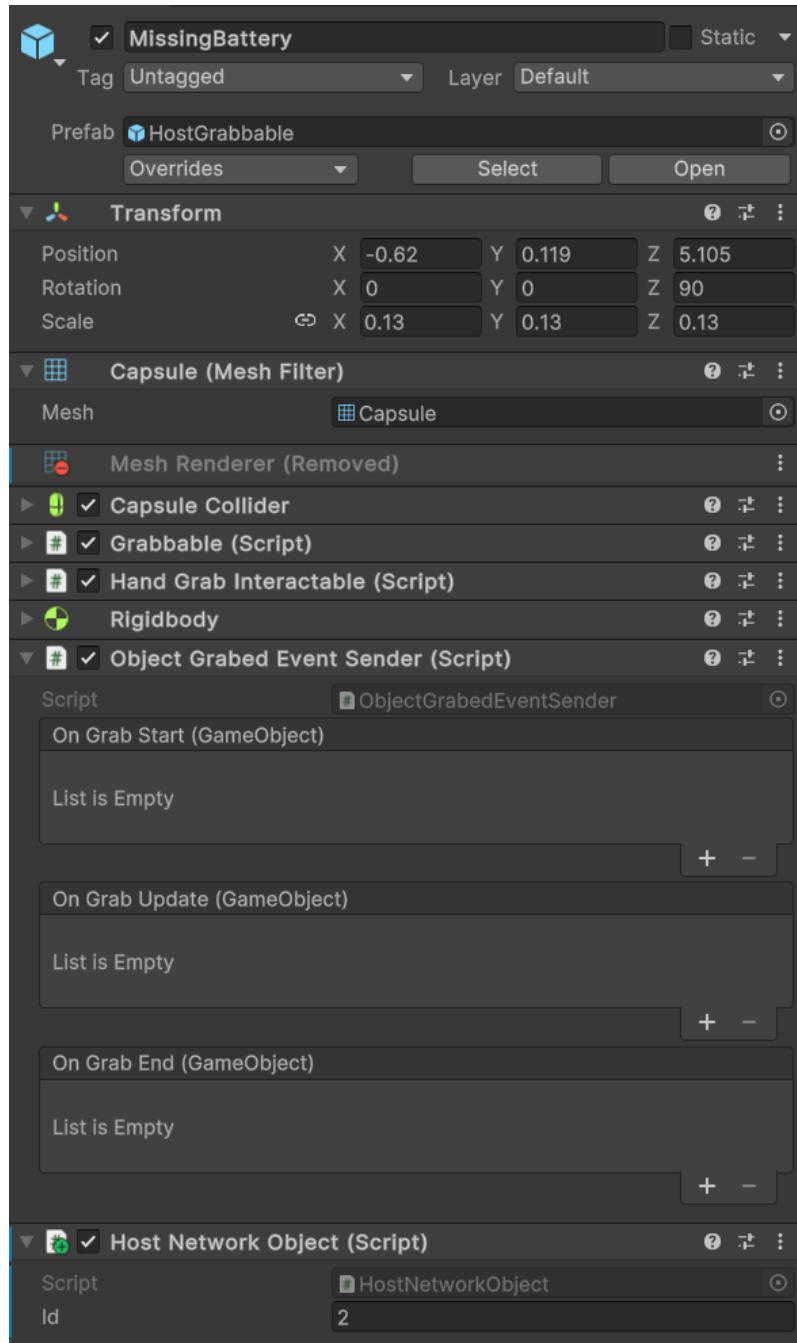


Figure 69 Inspecteur Unity représentant le GameObject MissingBattery

Voici un exemple d'objet attrapable où le modèle affichant l'objet est un enfant de l'objet attrapable. Le collider a tout de même dû être modifié pour être adapté à la taille et à la forme du modèle. On remarque également l'ajout du script "HostNetworkObject" pour rendre l'objet synchronisé.

### 3.6. Implémentation de la couche Monitoring

#### Implémentation de la partie Monitoring dans la scène du lobby

L'intégration de la fonctionnalité de choix de scénario s'est avérée plus complexe que prévu. En effet, la configuration des paramètres et leur association aux bons objets a posé des défis imprévus. Chaque scénario a dû être converti en prefab Unity pour centraliser les informations des scénarios en dehors d'une unique scène. Cette approche a permis d'accéder aux détails de chaque scénario, tels que le nombre d'énigmes et d'éléments, directement dans la scène où le gestionnaire de simulation devait être paramétré.

Il était crucial de veiller à ce que les identifiants dans les prefabs soient uniques. Au cours du développement, des conflits d'ID d'instance RPC ont surgi entre un élément de la scène et l'un des scripts du scénario (Scenario, Riddle, Element), entraînant des problèmes et des conflits. Malheureusement, aucune solution définitive n'a été trouvée pour résoudre ce problème. Cependant, en étant conscient de cette situation, il est possible d'effectuer une vérification minutieuse au niveau des prefabs pour anticiper d'éventuels problèmes.

L'utilisation de ScriptableObjects aurait pu potentiellement éviter ce problème. Cela aurait impliqué que l'objet ScenarioManager soit présent dans chaque scène, et non transférer entre les scènes, sans possibilité de le configurer préalablement. Par conséquent, les paramètres modifiés par le maître de jeu auraient dû être transmis via un ScriptableObject, et ces valeurs auraient été mises à jour au démarrage de la scène. En outre, cela aurait rendu plus complexe l'accès à la structure réelle du scénario, incluant toutes les énigmes et éléments, ce qui aurait compliqué la création de l'interface utilisateur pour le suivi.

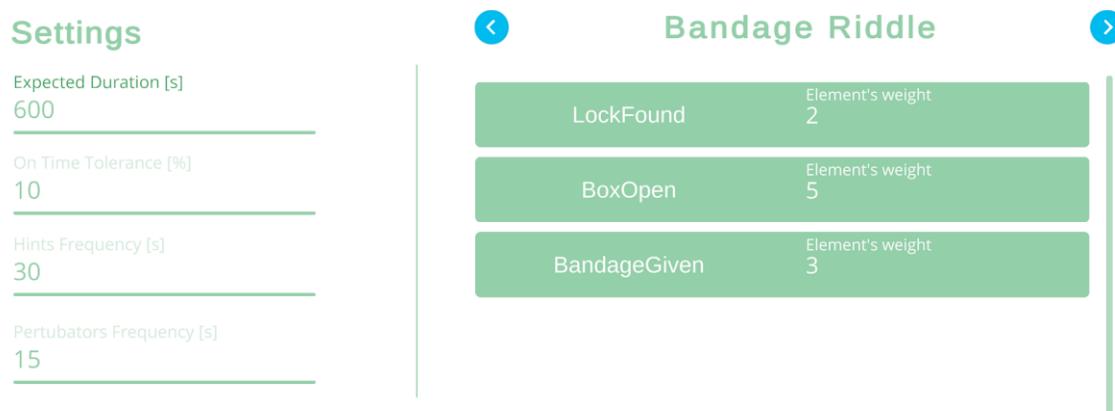


Figure 70 Interface des paramètres dans l'application de Monitoring

En pratique, l'ajout d'un nouveau scénario est hautement automatisé. Il vous suffit d'inclure la prefab dans la liste déroulante du carrousel, car les paramètres sont générés automatiquement.



Figure 71 Interface du choix de scénario dans l'application de Monitoring

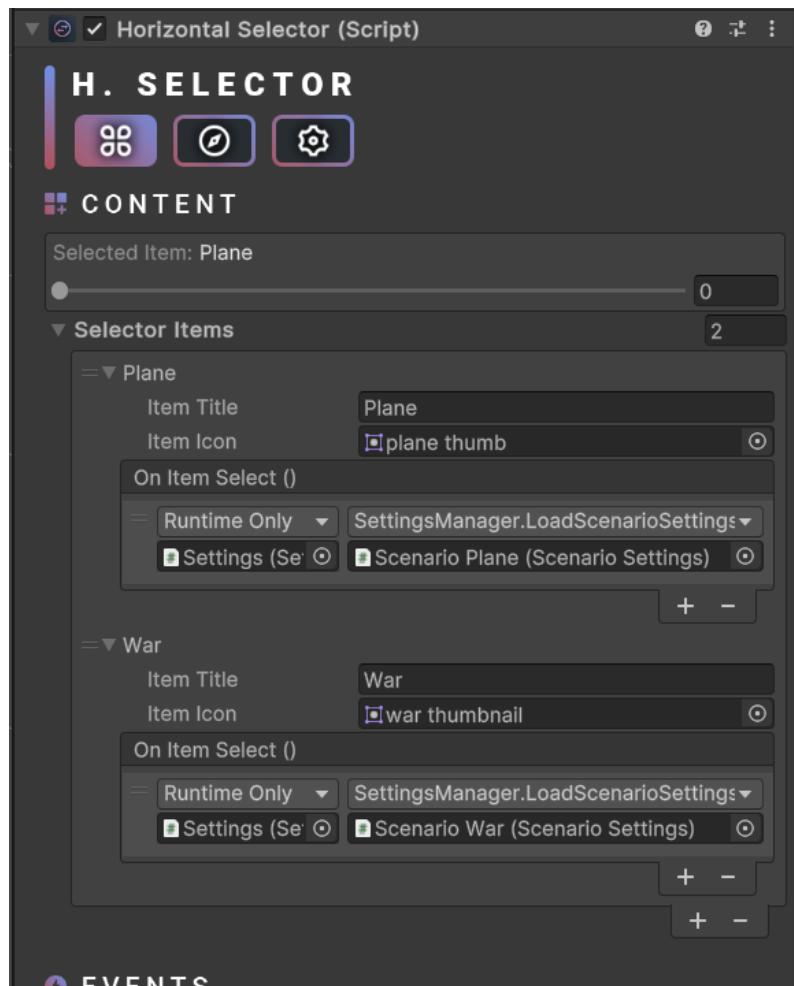


Figure 72 Inspecteur Unity de l'objet ScenarioSelector

Nous observons qu'une fonction est invoquée lorsqu'un élément du sélecteur est modifié. C'est cette fonction qui établit le lien entre l'interface et les données, permettant ainsi de générer l'interface et de sauvegarder les modifications.

```

public void LoadScenarioSettings(ScenarioSettings settings)
{
    scenarioManager.SetScenarioSettings(settings);
    expectedDurationInput.onEndEdit.RemoveAllListeners();
    expectedDurationInput.text = settings.ExpectedDuration().ToString();
    expectedDurationInput.onEndEdit.AddListener((string value) => settings.SetExpectedDuration(float.Parse(value)));

    onTimeToleranceInput.onEndEdit.RemoveAllListeners();
    onTimeToleranceInput.text = (settings.DeltaOnTime * 100).ToString();
    onTimeToleranceInput.onEndEdit.AddListener((string value) => settings.DeltaOnTime = float.Parse(value) / 100);

    hintsFrequencyInput.onEndEdit.RemoveAllListeners();
    hintsFrequencyInput.text = settings.TimeBetweenHints.ToString();
    hintsFrequencyInput.onEndEdit.AddListener((string value) => settings.TimeBetweenHints = float.Parse(value));

    pertubatorsFrequencyInput.onEndEdit.RemoveAllListeners();
    pertubatorsFrequencyInput.text = settings.TimeBetweenPertubators.ToString();
    pertubatorsFrequencyInput.onEndEdit.AddListener((string value) => settings.TimeBetweenPertubators = float.Parse(value));

    PopulateRiddleSelector(settings);
}

```

Figure 73 Code du script SettingsManager.cs permettant de lier l'interface aux données

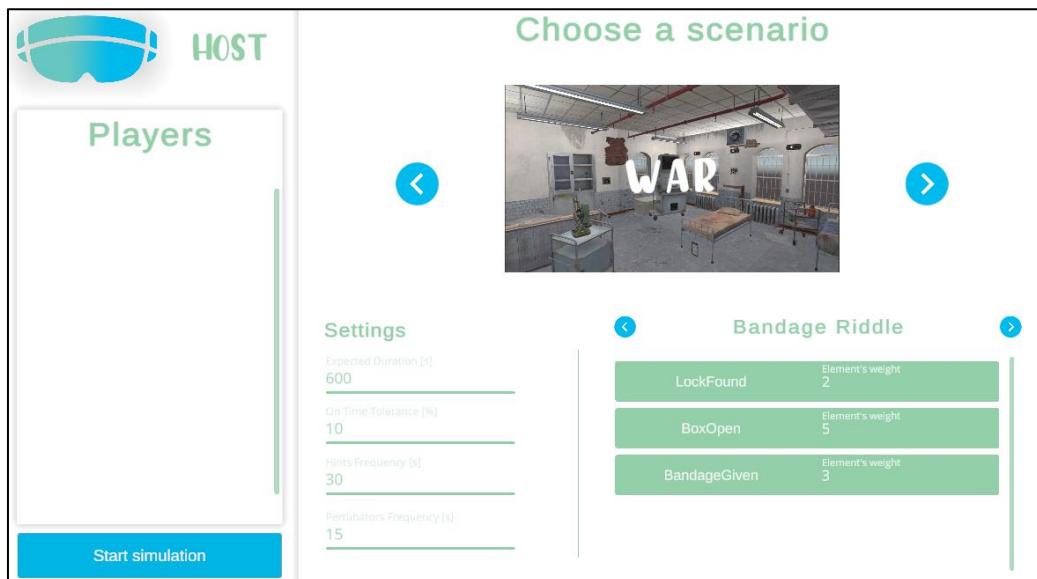


Figure 74 Interface complet de la scène du lobby dans l'application de Monitoring

### Implémentation de la partie Monitoring dans la scène de jeu

Cette section concerne principalement la gestion des flux vidéo des participants. En exploitant les fonctionnalités fournies par l'outil « FMETP\_Stream », notamment les encodeurs et décodeurs de vue, il était simplement nécessaire d'assurer l'identifiant identique tant chez le client que chez le serveur.

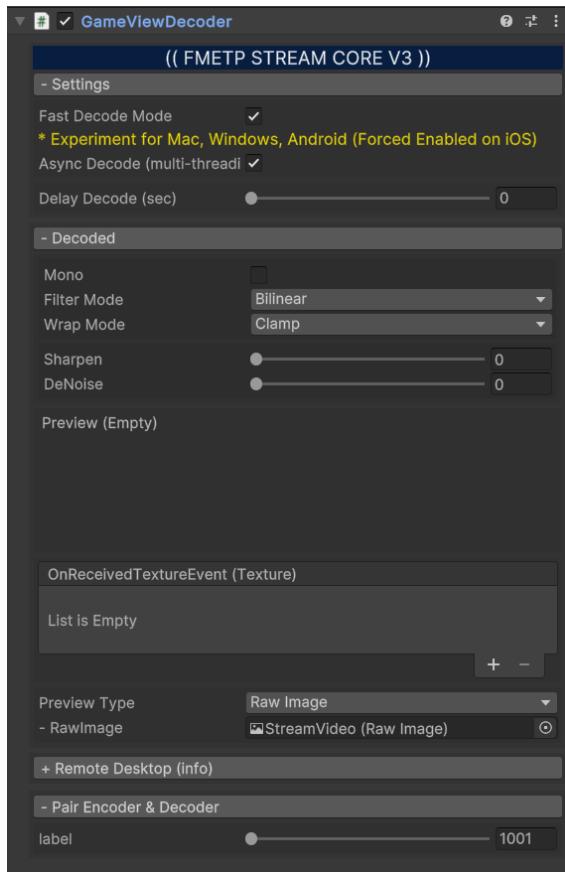


Figure 76 Inspecteur du script GameViewDecoder

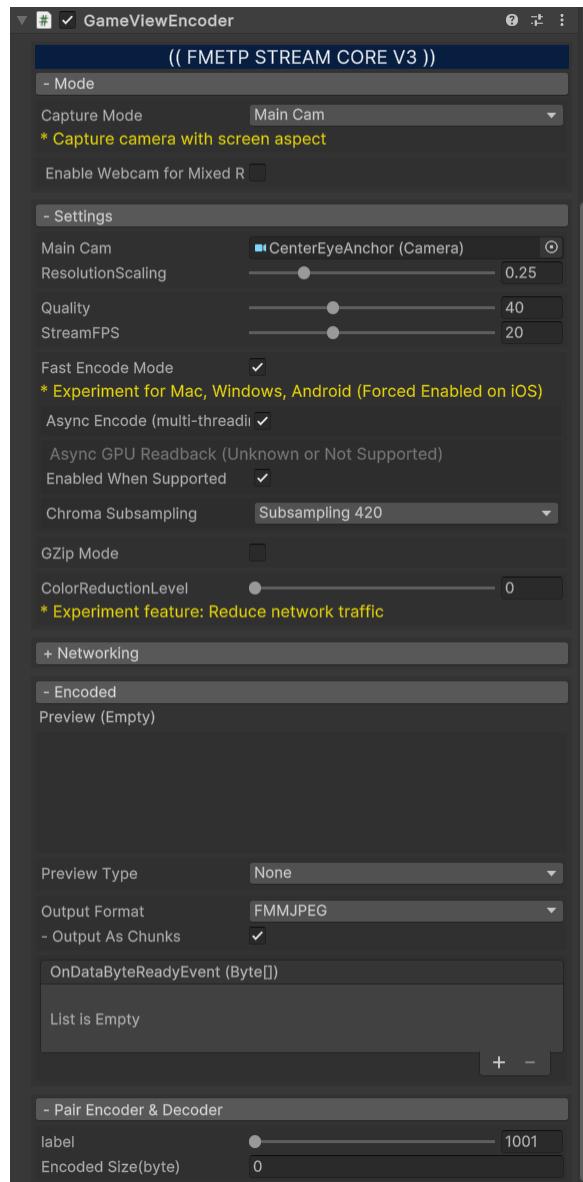


Figure 75 Inspecteur du script GameViewEncoder

Sur les deux objets présentés, un identifiant sous forme de label est clairement visible, servant d'ID. En établissant des paires d'encodeurs et de décodeurs, il devient alors possible d'accéder aux flux vidéo des participants via l'application de surveillance.

Pour le décodeur, il suffit de spécifier sur quelle image afficher le flux reçu, tandis que pour l'encodeur, il faut simplement préciser quelle caméra encoder. Une fois cette configuration effectuée, l'encodeur doit transmettre le flux de données en utilisant l'événement « `OnDataByteReadyEvent` ». Le décodeur dispose quant à lui d'une méthode « `Action_ProcessImageData` » à laquelle le serveur doit transmettre les données reçues de l'encodeur.

Le maître du jeu a également la possibilité d'envoyer des messages personnalisés aux joueurs. Cela se fait en utilisant les RPC créés dans la couche réseau. Un objet `MonitorManager` est présent à la fois sur l'application de surveillance et sur l'application de jeu. Pour afficher ces messages, le même mécanisme que celui utilisé pour les indices est employé, c'est-à-dire en utilisant les interacteurs.

```

public void SendMessageToPlayers(string text)
{
    if (HostNetworkManager.instance.IsServer())
    {
        HostNetworkManager.instance.SendRPC(new HostNetworkRPCMessage()
        {
            InstanceId = this.InstanceId,
            MethodName = "SendMessageToPlayers",
            Parameters = new object[] { text }
        });
    }
    else
    {
        textualInteractor.ShowText(text);
    }
}

```

Figure 77 Code de la fonction SendMessageToPlayers

### 3.7. Implémentation de la couche Feedback

La couche de feedback, étroitement liée à l'implémentation du monitoring, exploite une partie de l'interface utilisateur pour offrir la fonctionnalité de prise de notes.

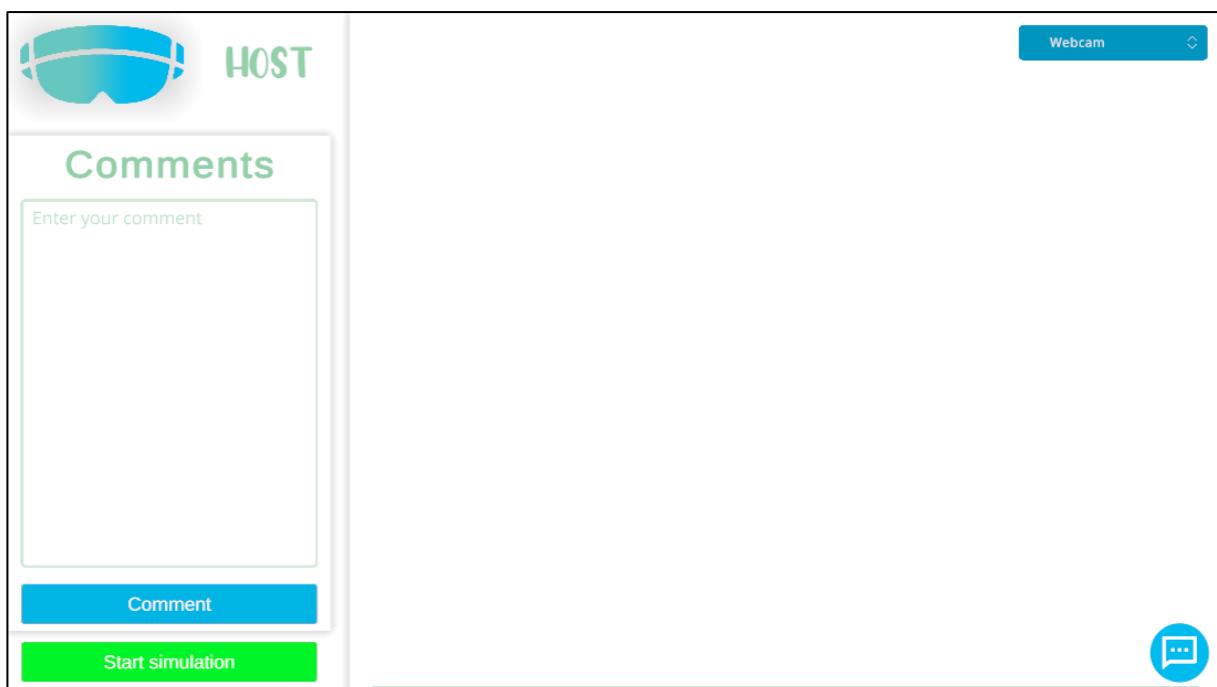


Figure 78 Interface lors d'une simulation

L'interface est conçue de manière à permettre au maître du jeu de se concentrer pleinement sur la prise de notes. Dans cette optique, la partie droite de l'interface affiche uniquement les flux vidéo, tandis que la partie gauche est réservée à la prise de notes. Lorsqu'un commentaire est enregistré à l'aide du bouton "Commentaire", il est stocké dans un objet appelé "DebriefingManager", chargé de sauvegarder les commentaires et de générer le PDF ainsi que la vidéo.

```

Comment comment = new Comment()
{
    text = commentInput.text,
    time = TimeSpan.FromSeconds(SceneManager.instance.GetTime()),
    image = texture.EncodeToPNG(),
};

```

Figure 79 Code pour créer un objet Comment

En tenant compte de ces données, la génération de feedback est simple. Le fichier de sous-titres est créé en parcourant la liste des commentaires et en formatant les informations qu'il contient au format srt.

```

foreach (Comment c in comments)
{
    srtWriter.WriteLine(commentIndex.ToString());
    srtWriter.WriteLine(GetSurroundingTimeStamp(c.time, -5f, 5f));
    srtWriter.WriteLine(c.text);
    srtWriter.WriteLine();
    commentIndex++;
}

```

Figure 80 Écriture du fichier de sous-titres

La fonction « GetSurroundingTimeStamp » détermine la durée pendant laquelle les sous-titres seront affichés. Dans cet exemple, ils sont affichés pendant 10 secondes, soit 5 secondes avant le TimeStamp du commentaire et 5 secondes après.

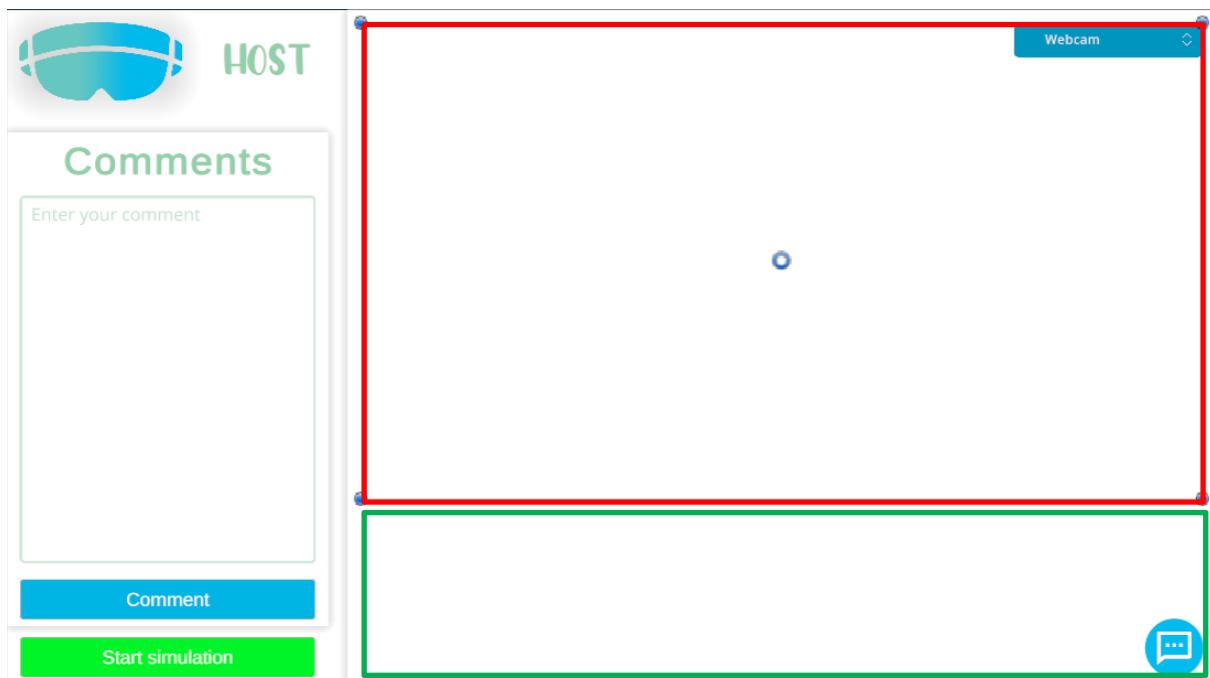


Figure 81 Interface de monitoring

Le cadre rouge représente l'espace principal dédié à la visualisation des flux vidéo. Les flux disponibles dans le cadre vert peuvent être échangés avec le flux principal pour l'afficher en plus grand. De plus, cette

zone rouge est enregistrée tout au long de la simulation pour créer une vidéo récapitulative. Les sous-titres mentionnés précédemment sont ajoutés à cette vidéo. L'enregistrement de la vidéo est effectué à l'aide de FFmpeg, une solution open source adaptée pour Unity, fournissant des scripts pour gérer les différentes commandes de FFmpeg, notamment la capture vidéo.

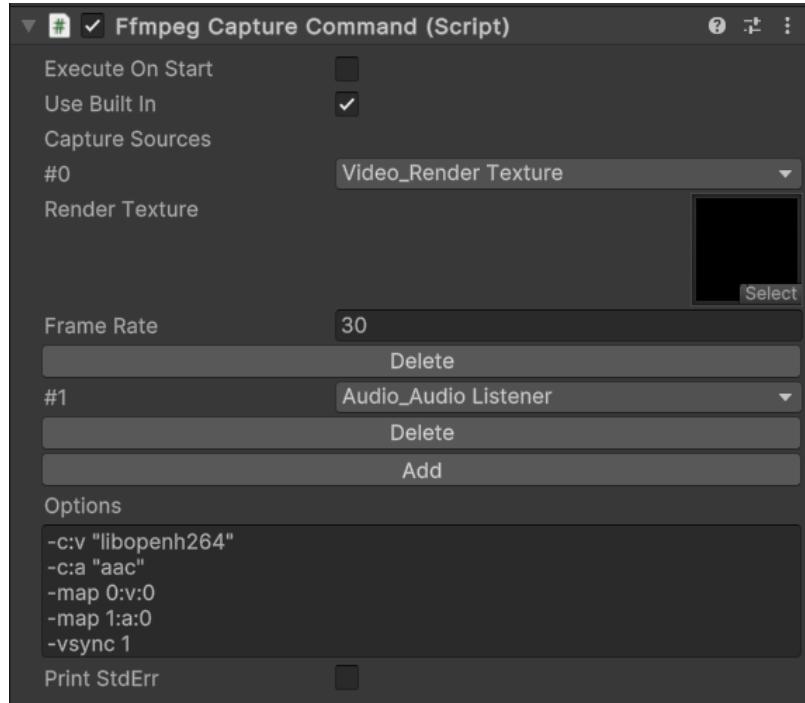


Figure 82 Script FFmpeg permettant la capture vidéo

Cet outil offre également la possibilité de « graver » les sous-titres directement sur la vidéo, éliminant ainsi le besoin d'ajouter manuellement le fichier de sous-titres à la vidéo. Cependant, cette fonctionnalité n'a malheureusement pas fonctionné comme prévu. En raison de la complexité et de la richesse des fonctionnalités de FFmpeg, il peut parfois être difficile de maîtriser toutes les subtilités de son utilisation. Dans ce cas précis, il est nécessaire d'attendre que la vidéo soit bien enregistrée, que le fichier de sous-titres le soit également, pour enfin exécuter la commande avec ces deux éléments pour créer une nouvelle vidéo ayant les commentaires brûlés sur l'image.

Quant à la génération du rapport PDF, elle est assurée par un autre composant, le PDFReportGenerator. Ce composant a pour seule responsabilité de créer le rapport à partir d'un nom de fichier et de la liste des commentaires. Il utilise la librairie PDFSharpCore, qui propose une gamme étendue de méthodes pour la création de documents PDF.

```
XImage image = XImage.FromStream(() => new MemoryStream(_defaultImageData));
gfx.DrawImage(image, new XRect(position.X, position.Y, width, image.PixelHeight * width / image.PixelWidth));
```

Figure 83 Exemple de code permettant d'afficher une image sur le PDF

### 3.8. Implémentation de l'ancrage du monde virtuel

Pendant la phase de lobby, l'ancrage du monde virtuel par rapport au monde réel est réalisé. Ainsi, la création de nouveaux scénarios ne nécessite pas de résoudre ce problème, car il est déjà pris en charge. Pour ce faire, le SDK de Meta a été à nouveau utilisé. Meta offre une fonctionnalité de découverte de la pièce, permettant de détecter les murs, le plafond, le sol et les meubles. Sur chaque surface détectée, il est possible de faire apparaître un objet virtuel ayant la même taille que son homologue réel.

En découvrant cette fonctionnalité, une question s'est posée : est-il possible de générer le monde virtuel en fonction du monde réel, c'est-à-dire de faire apparaître des éléments virtuels à différents endroits de la pièce plutôt que de projeter une réplique virtuelle de la pièce entière ? Après discussion avec M. Rekik, nous avons conclu que cette approche nécessiterait un travail de conception de scénario plus important et que cela limiterait la possibilité de créer des scénarios dans des environnements variés. Par conséquent, cette proposition n'a pas été retenue.

Cependant, l'implémentation utilise tout de même ce concept, car en faisant apparaître un objet sur le sol grâce à la fonctionnalité de découverte de la pièce, cet objet apparaît au centre exact de la pièce.

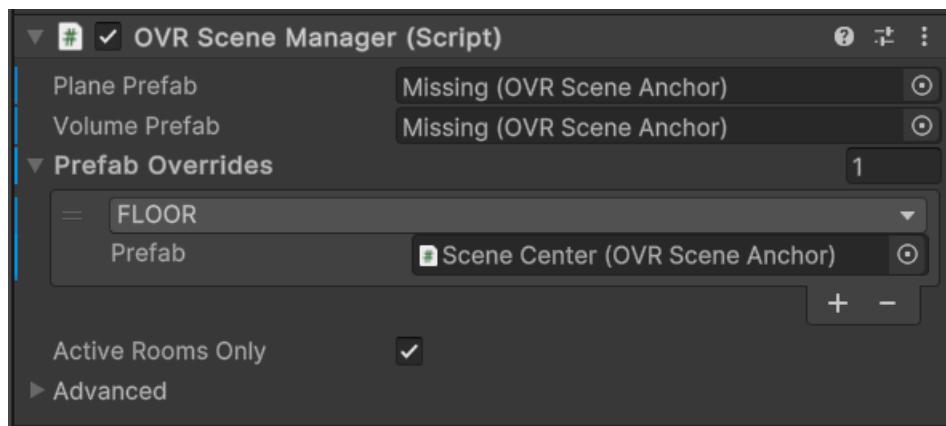


Figure 84 OVR Scene Manager fourni par Meta

Une prefab a donc été créée, apparaissant sur le sol. Cet objet permet de déterminer la position centrale de la pièce, identique pour tous les participants. De plus, cela permet également d'ajuster à la bonne hauteur, évitant ainsi les problèmes d'objets en dessous du sol.

Cependant, un problème subsiste : celui de la rotation. Pour résoudre ce problème, les joueurs doivent ajuster la rotation du monde eux-mêmes. Pendant la phase de lobby, une flèche pointe dans leur direction. Ils doivent ensuite orienter la flèche vers une marque placée sur le sol réel.

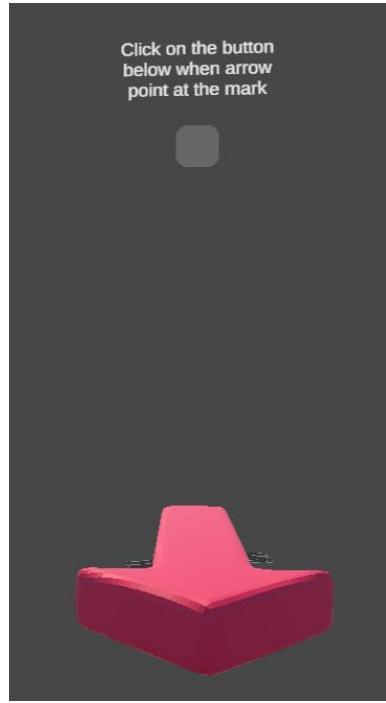


Figure 85 Flèche permettant la bonne rotation du monde virtuel

Une fois que la flèche pointe au bon endroit, les participants n'ont plus qu'à appuyer sur le bouton pour verrouiller sa rotation. Ensuite, pour centrer les mondes des scénarios, cette flèche est transférée dans la scène du scénario et le monde est orienté en fonction de la flèche. Enfin, la flèche est désactivée.

```

AutoCenter anchor = FindAnyObjectByType<AutoCenter>();
if (anchor != null && !isPlaced)
{
    Debug.Log("Placing scene");
    Debug.Log(gameObject.name);

    transform.position = anchor.transform.position;
    transform.rotation = anchor.transform.rotation;
    isPlaced = true;

    if (hideSceneAnchor)
    {
        foreach(Transform child in anchor.transform)
            child.gameObject.SetActive(false);
    }
}

```

Figure 86 Code permettant de positionner le monde par rapport à flèche

Cependant, un dernier problème subsiste et pourrait perturber considérablement la simulation. Si un joueur quitte la pièce ou retire son casque, il existe un risque que le monde virtuel change de position et d'orientation. Cela est dû au fait que lorsque l'application entre en veille, c'est-à-dire lorsque le casque est retiré ou que l'utilisateur sort des limites, le point central des coordonnées du casque change. En conséquence, le monde virtuel se recentre sur la nouvelle origine, ce qui peut entraîner des décalages indésirables. Une solution serait de réeffectuer la phase d'ancrage du casque, en affichant à nouveau la

fameuse flèche au milieu de la pièce et en la refaisant tourner pour que le monde soit à nouveau ancré correctement.

### 3.9. Bilan et résultats

En conclusion, l'application a été développée avec Unity 2023.1.15f1, une version qui a bénéficié des nombreuses fonctionnalités nouvelles disponibles avec le Meta Quest 3. Le code source est accessible sur GitHub via le lien suivant : <https://github.com/Host-Project/HostProject2.0>. En téléchargeant ces sources et en utilisant la bonne version d'Unity, aucune autre manipulation n'est nécessaire pour accéder au projet.

Les deux scénarios existants dans la version précédente ont été intégrés avec succès dans ce projet, confirmant ainsi son bon fonctionnement. Des ajustements ont toutefois été nécessaires pour adapter ces scénarios au nouveau système d'indices et de perturbateurs. Ces modifications n'ont toutefois aucun impact sur le déroulement de la simulation. Un guide de développement est disponible en annexe, indiquant la marche à suivre pour créer de nouveaux scénarios. Ce guide démontre également qu'il n'est pas obligatoire de coder pour créer un scénario, ce qui démontre de la simplicité de l'utilisation de ce nouveau squelette générique.



Figure 87 Monde virtuel du scénario Avion



Figure 88 Monde virtuel du scénario Guerre

Les applications de monitoring et pour Meta Quest 3 sont intégrées dans un même projet. Pour compiler chacune d'elles séparément, il est d'abord nécessaire d'activer l'objet Serveur et de désactiver l'objet Client dans la scène "01\_Lobby", ou vice versa.

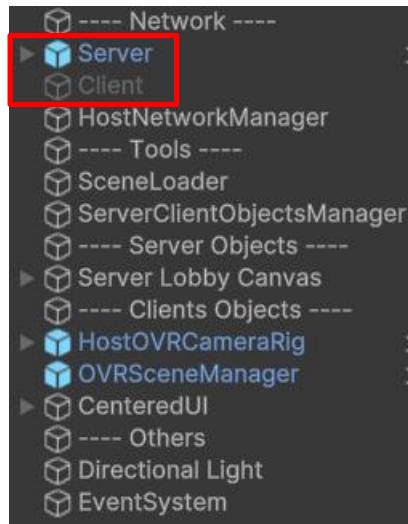


Figure 89 Hiérarchie de la scène 01\_Lobby

Si l'objectif est de compiler le serveur, il est impératif de vérifier que la plateforme cible de la compilation est définie comme étant « Windows, Mac, Linux ». Dans le cas de Windows, un fichier .exe sera généré, permettant ainsi d'utiliser l'application de monitoring.

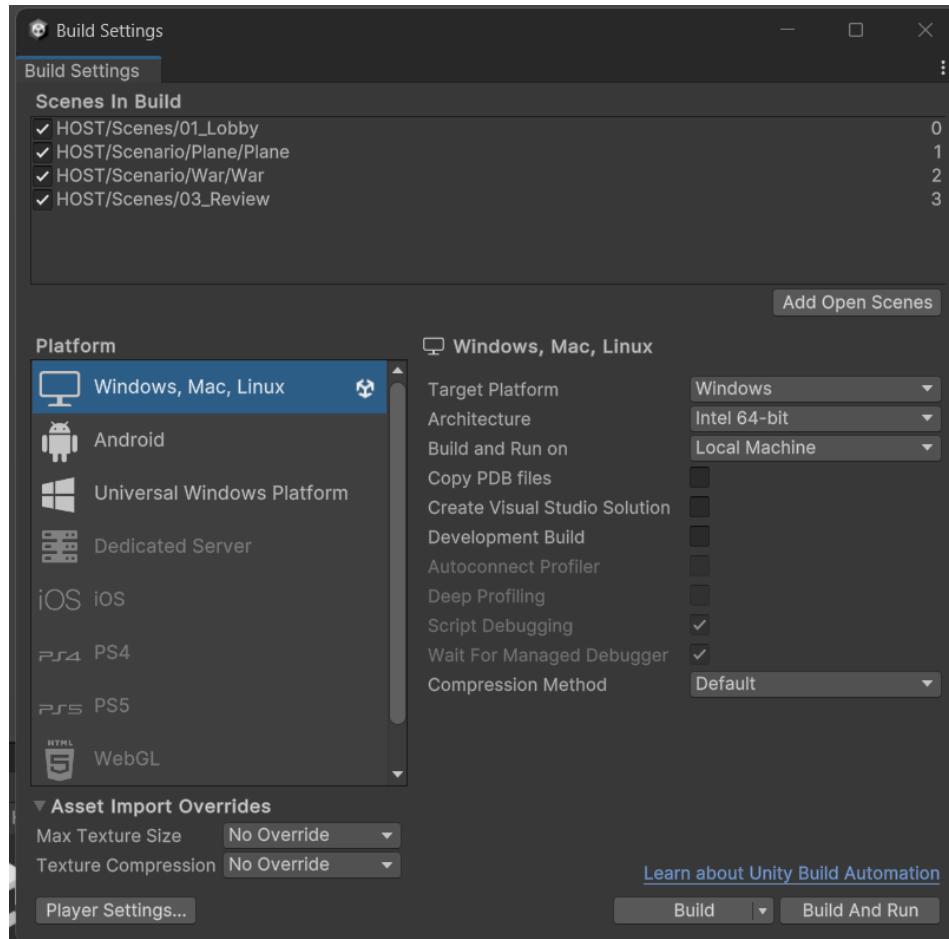


Figure 90 Fenêtre de configuration de compilation Unity

Si l'objectif est de compiler pour le Meta Quest 3, il est nécessaire de suivre tout d'abord le tutoriel disponible sur Meta pour se familiariser avec le développement Meta Quest et Unity : <https://developer.oculus.com/documentation/unity/unity-tutorial-hello-vr/>. Ensuite, il faut compiler l'application en choisissant la plateforme "Android". Si Unity détecte le casque, comme illustré dans le tutoriel, il est possible de cliquer sur « Build And Run », ce qui installera automatiquement l'application sur le casque. Dans le cas contraire, il est nécessaire de passer par l'application « Meta Quest Developer Hub », qui permet d'installer des fichiers .apk sur le casque, générés par Unity lors de la compilation.

Sur le plan du développement, il est tout à fait possible d'utiliser l'application de monitoring en mode « Play Mode » sur Unity. Cependant, il reste indispensable de compiler l'application pour le Meta Quest 3 afin de tester les modifications. À ce jour, il n'existe pas d'outils simplifiant ce processus.

Tester une application de jeu peut être complexe, mais des tests manuels ont été effectués pour chacune des couches abordées dans ce rapport. Ainsi, il est assuré que la structure du scénario fonctionne correctement et est bien synchronisée entre tous les participants. La partie multijoueur a également été testée à plusieurs reprises avec deux personnes, confirmant que l'application fonctionne ainsi que les interactions possibles entre les participants.

## 4. CONCLUSION

Dans ce travail de Master, une analyse de la solution existante ainsi que des retours reçus des utilisations l'ayant utilisé pour en tirer plusieurs axes d'améliorations. Il était recherché d'avoir une meilleure immersion, une réduction des responsabilités du maître de jeu, ainsi que l'aspect pour les développeurs qui est de faciliter l'implémentation de nouveaux scénarios. Ceci englobait des outils pour les communications réseaux, pour le scénario en lui-même ainsi que pour les indices et perturbateurs.

Les résultats obtenus sont encourageants, les besoins spécifiés ont été adressés, certains de manière forte tel que l'aspect nouveau scénario. Cette facilité de mise en place des scénarios peut varier d'un individu à l'autre, mais le temps nécessaire pour implémenter un nouveau scénario a considérablement diminué par rapport à la version précédente du projet. Le maître de jeu peut désormais se concentrer davantage sur la prise de notes, grâce à un gestionnaire de jeu intelligent qui a allégé une grande partie de sa charge de travail. Bien que le système mis en place soit satisfaisant pour l'instant, il est probable qu'il nécessite quelques ajustements pour être encore plus efficace.

Les joueurs auront désormais la possibilité de transmettre les objets qu'ils détiennent à leurs compagnons, ce qui constitue une amélioration significative de l'immersion. La synchronisation des objets renforce le sentiment de faire partie du monde virtuel, malgré les légers délais éventuellement ressentis.

La majorité des concepts ont été anticipés en amont, ce qui a permis de mener le projet à bien avec relativement peu d'incidents, bien qu'il y en ait eu quelques-uns. Les problèmes rencontrés étaient principalement liés à un manque de familiarité avec les ressources utilisées, notamment l'asset utilisé, le SDK de Meta et la structure des projets Unity.

Il est évident que ce projet offre de nombreuses possibilités d'amélioration. À mon avis, la meilleure fonctionnalité à implémenter serait de créer des environnements virtuels en fonction du monde réel, en exploitant notamment les fonctionnalités plus en profondeur utilisées dans la partie ancrage du projet. Meta développe également activement des fonctionnalités pour le Meta Quest 3, notamment au niveau de l'occlusion, qui pourraient être intégrées au projet. Si Meta venait à permettre l'accès au flux de la caméra à l'avenir, l'ajout d'une analyse vidéo serait très bénéfique, notamment pour valider les étapes du jeu en cachant des QR codes dans les objets réels.

Je suis ravi d'avoir eu l'opportunité de réaliser ce projet de Master, et je remercie une fois de plus M. Rekik pour cela. Cela m'a permis de plonger véritablement dans le monde du développement de jeux, offrant une nouvelle perspective à ma carrière naissante. Je suis convaincu que je continuerai à travailler sur des projets de réalité virtuelle ou augmentée à l'avenir, que ce soit à titre personnel ou professionnel.

Je suis également satisfait du travail que j'ai accompli et de mon organisation tout au long de ce projet. J'ai beaucoup appris sur la gestion de projet et l'importance de prévoir du temps supplémentaire pour les imprévus. J'ai également eu l'occasion de mettre en pratique mes compétences analytiques, qui avaient été peu sollicitées dans mon cursus scolaire jusqu'à présent.

## 5. BIBLIOGRAPHIE

- [1] *After Quest 2 HMD goes to sleep while running my app, scene is offset when HMD awakes.* (2023, Juillet 22). Récupéré sur Meta Forum: <https://communityforums.atmeta.com/t5/Unity-VR-Development/After-Quest-2-HMD-goes-to-sleep-while-running-my-app-scene-is/td-p/974665>
- [2] Barman, C. (2020). *Formation médicale en réalité augmentée.*
- [3] *Check if an object is grabbed.* (2022, Avril 25). Récupéré sur Meta Forum: <https://communityforums.atmeta.com/t5/Unity-VR-Development/Check-if-an-object-is-grabbed/m-p/961601#M20629>
- [4] FrozenMist. (s.d.). *FMETP Stream.* Récupéré sur FrozenMist: <https://www.frozenmist.com/>
- [5] *How to disable Quest recentering in Unity XR 2019.3?* (2020, Février 21). Récupéré sur Meta Forum: <https://communityforums.atmeta.com/t5/Quest-Development/How-to-disable-Quest-recentering-in-Unity-XR-2019-3/m-p/870469#M3242>
- [6] Hutinet, M. (2020). *HOST (HUMAN AND ORGANIZATIONAL SKILLS TRAINING).*
- [7] Ivashchenko, D. (2023, Août 7). *Unity Realtime Multiplayer, Part 2: TCP, UDP, WebSocket Protocols.* Récupéré sur Hackernoon: <https://hackernoon.com/unity-realtime-multiplayer-part-2-tcp-udp-websocket-protocols>
- [8] Lade Tech. (s.d.). *une salle magique.* Récupéré sur Lade Tech: <https://lade.tech/portfolio/salle-magique/>
- [9] Meta. (s.d.). *Interactions.* Récupéré sur Meta Quest: <https://developer.oculus.com/resources/hands-design-interactions/>
- [10] Meta. (s.d.). *Passthrough API Overview.* Récupéré sur Meta Quest: <https://developer.oculus.com/documentation/unity/unity-passthrough/>
- [11] Meta. (s.d.). *Use OVRCameraManager.* Récupéré sur Meta Quest: <https://developer.oculus.com/documentation/unity/unity-scene-use-scene-anchors/>
- [12] MindMaze. (2019, Juillet 12). *Jeu vidéo et santé : unis pour le meilleur et pour guérir?* Récupéré sur Mindmaze: <https://mindmaze.com/jeu-video-et-sante-unis-pour-le-meilleur-et-pour-guerir/>
- [13] Unity. (s.d.). *Get Started with NGUI.* Récupéré sur Unity Multiplayer Networking: <https://docs-multiplayer.unity3d.com/netcode/current/tutorials/get-started-nogui/>
- [14] Unity. (s.d.). *Simple client and server.* Récupéré sur Unity Manual: <https://docs.unity3d.com/Packages/com.unity.transport@2.0/manual/client-server-simple.html>
- [15] Valecillos, D. (2023, Avril 25). *How to Use Scene Understanding For Complex VR Passthrough Experiences!* Récupéré sur Youtube: <https://www.youtube.com/watch?v=UdXwZgRcf7U>
- [16] Valecillos, D. (2023, Novembre 4). *Quest 3: Powerful Mixed Reality Features with The NEW Meta Depth API.* Récupéré sur Youtube: <https://www.youtube.com/watch?v=mk6UYMaHZOo>

## 6. LISTE DES FIGURES

Figure 1 Jeu sérieux pour la rééducation (MindMaze, 2019) .....	1
Figure 2 Exemple d'une salle d'Escape Game (Lade Tech, s.d.) .....	2
Figure 3 Avion virtuel dans lequel les participants évoluent .....	3
Figure 4 Application de monitoring .....	3
Figure 5 Structure de la communication entre les applications .....	5
Figure 6 Ancre du monde virtuel avec le réel .....	6
Figure 7 Lobby sur l'application de monitoring .....	6
Figure 8 Tutoriel sur les interactions .....	7
Figure 9 Télévision avec des instructions pour ouvrir un cadenas .....	7
Figure 10 Interface utilisateur de l'application de monitoring .....	8
Figure 11 Interface utilisateur pour l'envoie des perturbateurs .....	8
Figure 12 Interface utilisateur pour l'envoie d'indices .....	9
Figure 13 Interface utilisateur pour la prise de notes .....	9
Figure 14 Exemple de commentaires dans le PDF .....	10
Figure 15 Script HelpRPC.cs où sont inscrits les RPC de l'application de monitoring .....	11
Figure 16 Script HelpRPC.cs où sont inscrites les RPC de l'application de jeu .....	11
Figure 17 Schéma de l'utilisation des différentes paires serveur/client .....	12
Figure 18 Une partie du script HelpSelection.cs où sont écrits les indices .....	13
Figure 19 Une partie du script DBManager.cs où sont réécrits les indices .....	13
Figure 20 Diagramme de séquence simplifié des RPC .....	22
Figure 21 Diagramme de séquence simplifié des objets synchronisés .....	23
Figure 22 Schéma de la structure d'un scénario .....	24
Figure 23 Diagramme de séquence simplifié et non complet de la structure du scénario .....	25
Figure 24 Diagramme de classe simplifié démontrant la structure avec héritage .....	26
Figure 25 Exemple d'utilisation d'événements .....	27
Figure 26 Diagramme de séquence simplifié montrant la structure du scénario avec les communications réseau .....	28
Figure 27 : Réflexion sur la manière d'influencer les joueurs .....	29
Figure 28 Système de poids pour paramétriser le scénario .....	31
Figure 29 Diagramme de classes simple des influenceurs et des interacteurs .....	32
Figure 30 Mock up de l'interface en simulation .....	34
Figure 31 Interface permettant le paramétrage de la simulation .....	34
Figure 32 Code permettant d'enregistrer les objets afin qu'ils deviennent synchronisés .....	36
Figure 33 Code permettant de mettre à jour la liste des objets synchronisés .....	37
Figure 34 Code permettant au serveur de satisfaire la demande de synchronisation d'un client .....	37
Figure 35 Code de la fonction permettant d'envoyer des RPC .....	38
Figure 36 Classe HostNetworkRPC et ses différents champs .....	38
Figure 37 Code pour décoder et transmettre les RPC .....	38
Figure 38 Code pour exécuter les RPC .....	39
Figure 39 Code pour envoyer un RPC .....	39

Figure 40 Méthode Update des objets synchronisés.....	40
Figure 41 La vue de l'inspecteur pour la valise dans le scénario de l'avion .....	40
Figure 42 Inspecteur Unity représentant le script Scenario .....	41
Figure 43 Code montrant que le scénario s'enregistre auprès des événements des énigmes .....	42
Figure 44 Code de la méthode OnRiddleComplete .....	42
Figure 45 Code de la méthode StartRiddle.....	43
Figure 46 Inspecteur Unity représentant le script Riddle .....	43
Figure 47 Inspecteur Unity représentant le script Element .....	44
Figure 48 Diagramme de classe des scripts contenant les paramètres .....	45
Figure 49 Vue des paramètres d'un scénario sur l'inspecteur Unity .....	46
Figure 50 Calcul du niveau d'influence dans le ScenarioManager .....	47
Figure 51 Script de la classe ProgessionState.cs .....	47
Figure 52 Script de la classe OnTimeState .....	48
Figure 53 Script de la classe RiddleLateState.cs .....	48
Figure 54 États disponibles dans le script ScenarioManager.cs.....	49
Figure 55 Script Influencer.cs .....	49
Figure 56 Inspecteur Unity permettant de visualiser le script Hint .....	50
Figure 57 Exemple d'indice affichant une image.....	50
Figure 58 Exemple d'indice affichant un GameObject.....	51
Figure 59 Script TextualInteractor.cs .....	51
Figure 60 Script InfluenceInteractor.cs.....	52
Figure 61 Inspecteur Unity affichant un interacteur textuel .....	52
Figure 62 Inspecteur Unity affichant un interacteur graphique .....	53
Figure 63 Hiérarchie de l'objet fourni par Meta .....	54
Figure 64 Prefab de l'objet attrapable .....	55
Figure 65 Prefab de l'objet déplaçable .....	56
Figure 66 Prefab de l'objet touchable.....	56
Figure 67 Prefab de l'objet regardable .....	57
Figure 68 Hiérarchie du GameObject MissingBattery .....	57
Figure 69 Inspecteur Unity représentant le GameObject MissingBattery .....	58
Figure 70 Interface des paramètres dans l'application de Monitoring .....	59
Figure 71 Interface du choix de scénario dans l'application de Monitoring .....	60
Figure 72 Inspecteur Unity de l'objet ScenarioSelector .....	60
Figure 73 Code du script SettingsManager.cs permettant de lier l'interface aux données .....	61
Figure 74 Interface complète de la scène du lobby dans l'application de Monitoring .....	61
Figure 75 Inspecteur du script GameViewEncoder .....	62
Figure 76 Inspecteur du script GameViewDecoder .....	62
Figure 77 Code de la fonction SendMessageToPlayers.....	63
Figure 78 Interface lors d'une simulation.....	63
Figure 79 Code pour créer un objet Comment.....	64
Figure 80 Ecriture du fichier de sous-titres.....	64
Figure 81 Interface de monitoring .....	64
Figure 82 Script FFmpeg permettant la capture vidéo .....	65
Figure 83 Exemple de code permettant d'afficher une image sur le PDF .....	65

Figure 84 OVR Scene Manager fourni par Meta .....	66
Figure 85 Flèche permettant la bonne rotation du monde virtuel .....	67
Figure 86 Code permettant de positionner le monde par rapport à flèche .....	67
Figure 87 Monde virtuel du scénario Avion.....	68
Figure 88 Monde virtuel du scénario Guerre .....	68
Figure 89 Hiérarchie de la scène 01_Lobby .....	69
Figure 90 Fenêtre de configuration de compilation Unity .....	70

## **7. ANNEXES**

### Liste des annexes

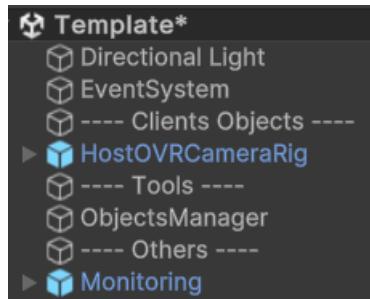
- Guide de développement d'un Scénario

## Guide de développement d'un Scénario

Ce guide couvrira tous les éléments essentiels pour créer un nouveau scénario, en suivant un format étape par étape afin de garantir l'inclusion de tous les points importants dans l'ordre recommandé.

### Étape 1 : Création de la scène

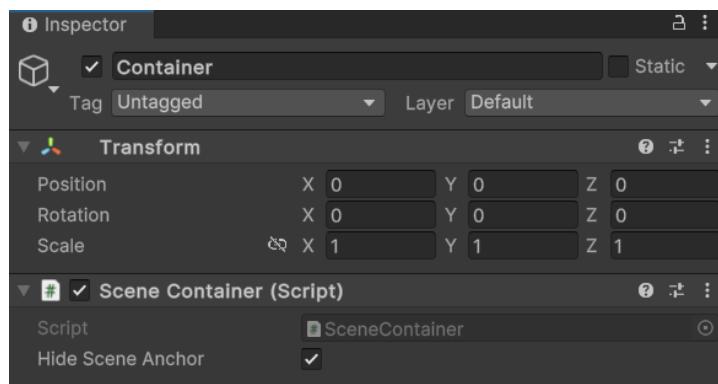
Il existe une scène nommée « Template » qui comprend tous les éléments communs et nécessaires à chaque scénario. Cette scène est située dans le répertoire : 'Assets/Host/Scenario/Template.unity'.



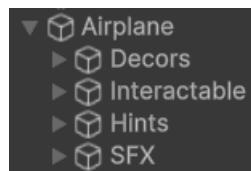
Il suffit de la copier afin de créer la base du nouveau scénario.

### Étape 2 : Conception du monde virtuel

Dans la nouvelle scène créée, il est nécessaire d'incorporer un objet qui agira comme le parent pour l'environnement virtuel. Cet objet devra être équipé du script « Scene Container », qui assure l'alignement automatique du monde au centre de la pièce réelle et dans la bonne orientation.

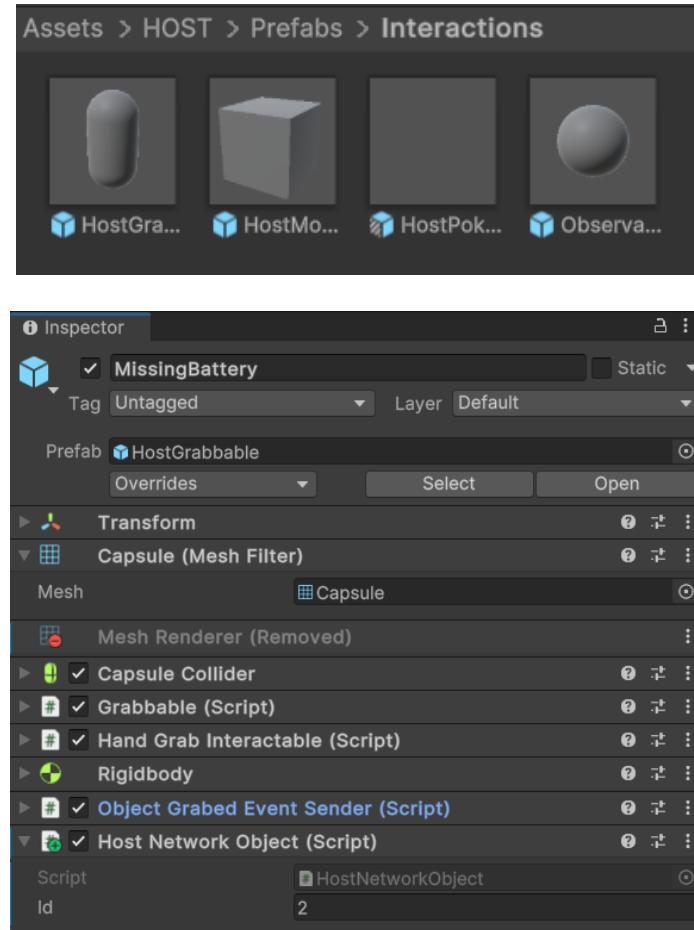


Il faudra ensuite créer le monde virtuel comme l'entend le scénario, et le tout à l'intérieur de cet objet. Par exemple, pour le scénario de l'avion cela est fait de la manière suivante :



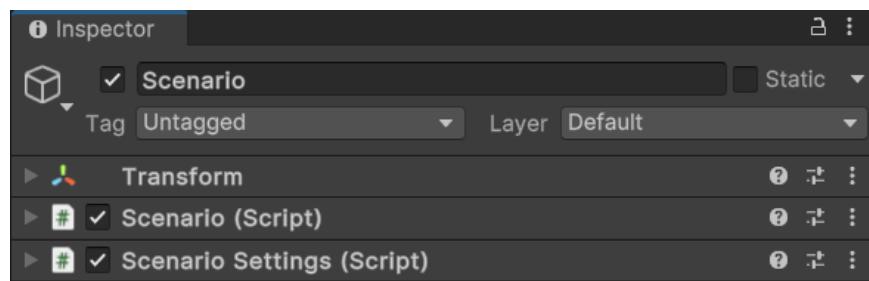
La structure à l'intérieur de l'objet n'a aucune contrainte, il est laissé de libre de classer les différents éléments comme le créateur du nouveau scénario le souhaite.

Il est possible d'utiliser les prefabs mises à disposition pour le projet, ainsi que le script permettant de rendre un objet synchronisé.

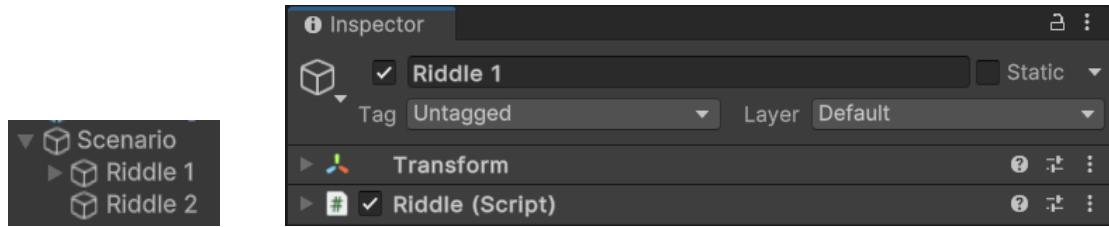


### Étape 3 : Création de la structure du scénario

Il faut ensuite créer un GameObject Scenario, qui aura les scripts « Scenario.cs » ainsi que « ScenarioSettings.cs ».

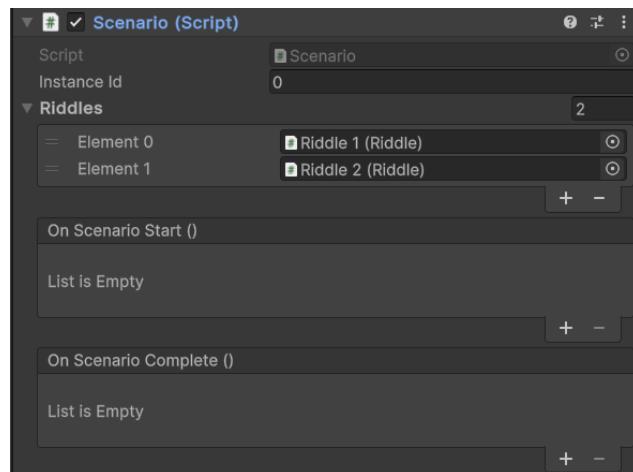


Il faudra ensuite créer les GameObject représentant les énigmes, autant que le demande le nouveau scénario. Ces GameObject doivent avoir le script « Riddle.cs ».



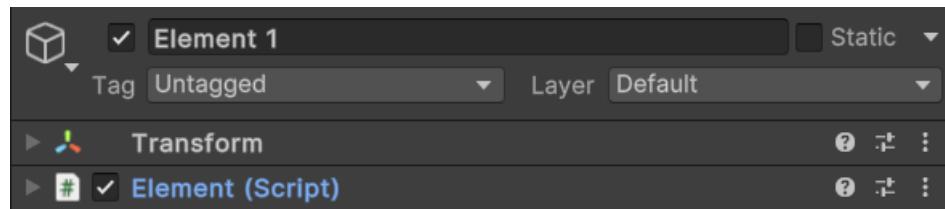
*Il n'est pas nécessaire que les énigmes soient des enfants du scénario dans la hiérarchie.*

Une fois cela fait, il faut ajouter les énigmes au scénario, via l'inspecteur Unity.

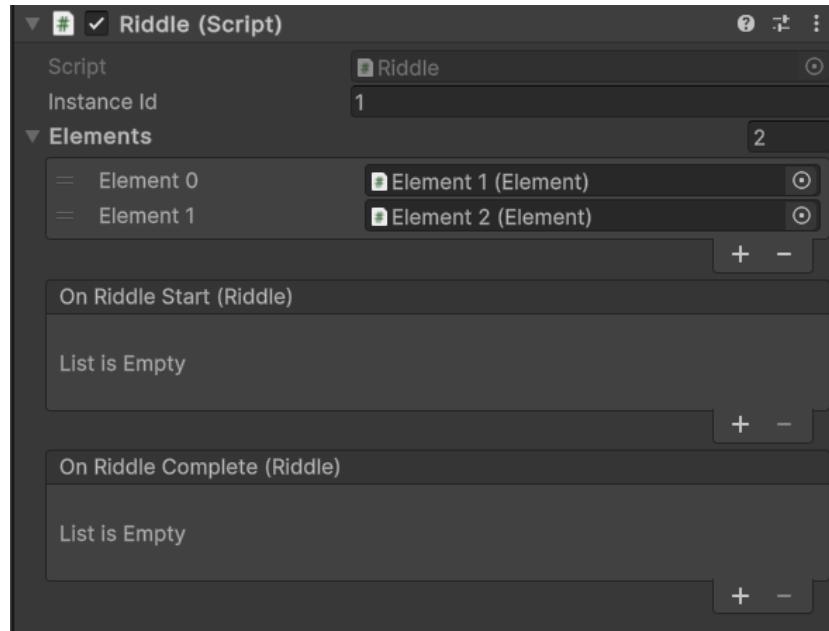


*Les énigmes étant ordonnées, s'il est nécessaire que toutes les étapes soient réalisable en même temps, n'utiliser que seule énigme.*

Il faut maintenant créer les éléments déclencheurs, en créant à nouveau des GameObject avec le script « Element.cs ».

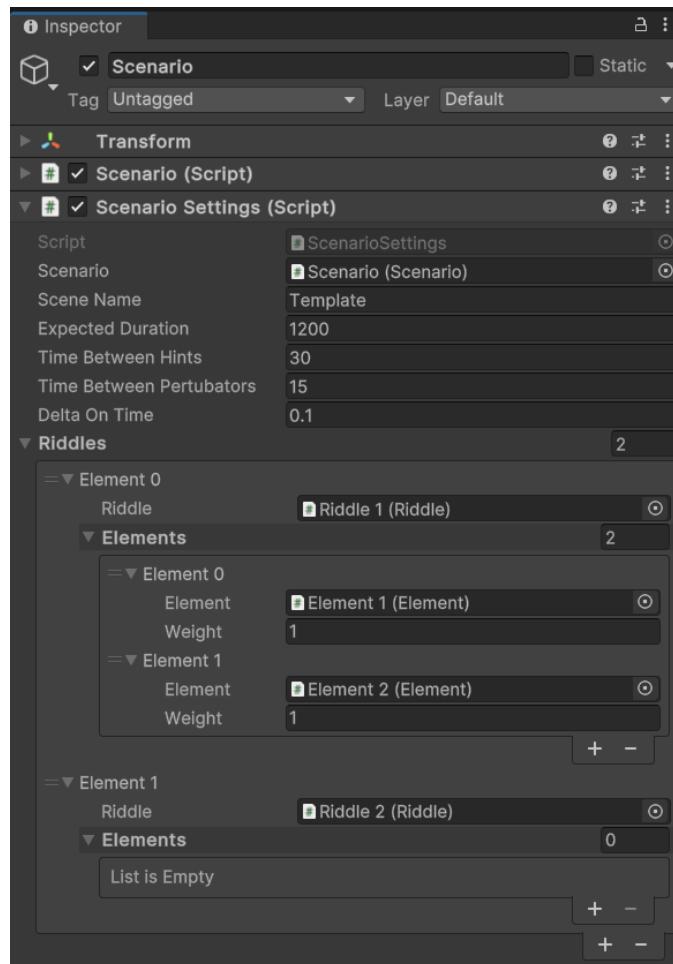


De la même manière qu'il était nécessaire d'assigner les énigmes au scénario, il faut assigner les éléments à leur énigme correspondante.



Effectuez ceci pour chacune des énigmes du scénario.

Une fois la structure en place, il faut paramétriser le scénario en utilisant l'inspecteur Unity du script « ScenarioSettings.cs ».



*Il est important que le paramètre « Scene Name » soit le même que le nom de la scène dans lequel le scénario est créé.*

On peut ici définir les paramètres par défaut qui seront utilisé lors de la simulation, dans le cas où le maître de jeu ne les modifie pas.

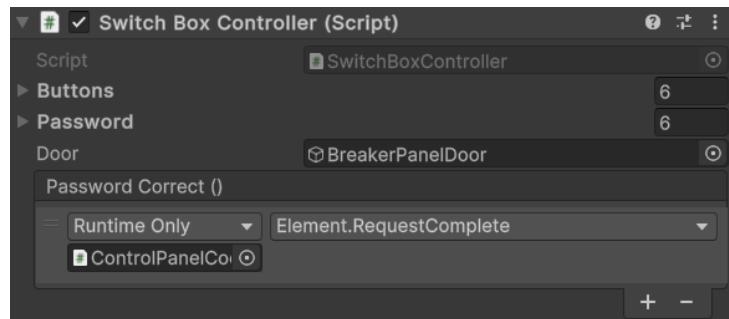
Les paramètres « Expected Duration », « Time Between Hints » ainsi que « Time Between Perturbators » sont en secondes. Le paramètre « Delta On Time » est en pourcentage, indiquant ainsi la différence de temps qu'il faut afin de considérer que les joueurs sont encore dans les temps.

#### Étape 4 : Mise en place des événements principaux du scénario

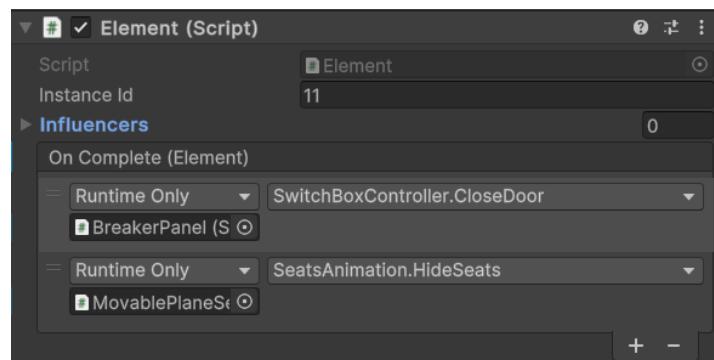
Cette section est directement liée à un scénario. Il sera nécessaire de coder les interactions spécifiques souhaités, en s'aidant des événements principaux proposés ici. Seul le concept pour être décrit dans ce guide.

En premier lieu, il est nécessaire que chacun des éléments déclencheurs aient leur méthode « RequestComplete » utilisée. Sans quoi, le scénario sera bloqué.

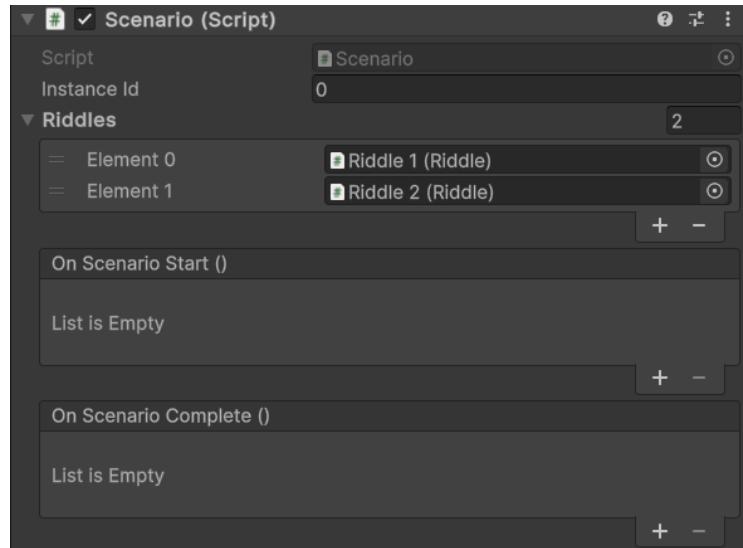
Voici un exemple dans le scénario de l'avion.



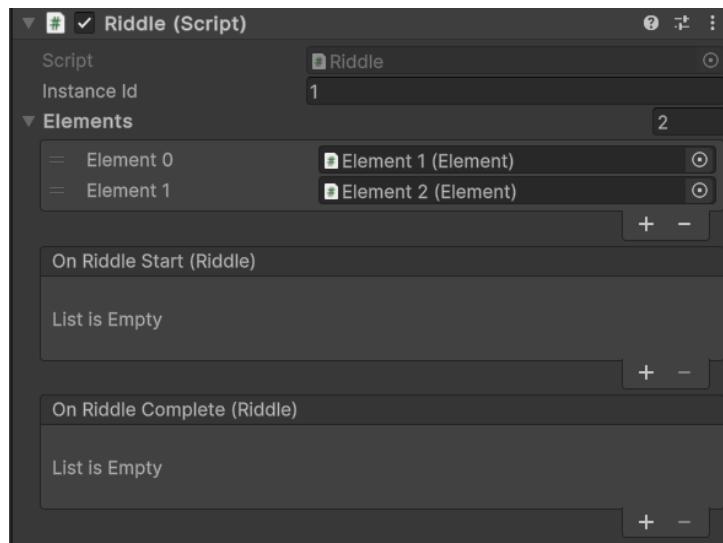
Lorsque le code est trouvé, on demande la compléction de l'élément. Pour effectuer une action qui sera répercuté chez tous les joueurs, il est nécessaire d'utiliser l'événements « OnComplete ».



D'autres événements sont également disponibles, tel que :



Le début et la fin du scénario, où il est possible d'effectuer une fonction pour, par exemple, mettre en place des éléments dans le décor.



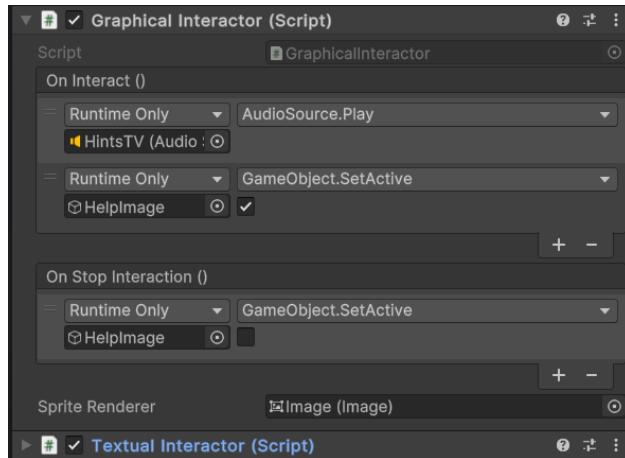
Le début et la fin de chaque énigme, qui pourrait être utilisé pour effectuer une animation en début d'énigme, comme par un personnage qui donne un objet aux joueurs.

*Tous ces événements seront effectués chez tous les joueurs, et sur le serveur.*

### Étape 5 : Mise en place des indices et perturbateurs

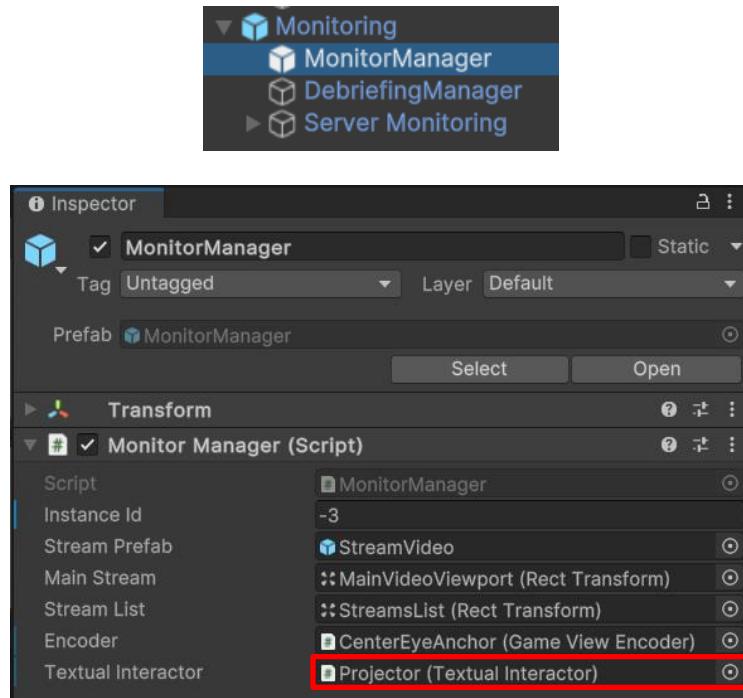
Le mécanisme pour créer des indices et des perturbateurs nécessitent tout d'abord de mettre en place un objet, ou plusieurs affichants les indices.

Plusieurs scripts sont à disposition, mettant à disposition plusieurs types d'indices différents.



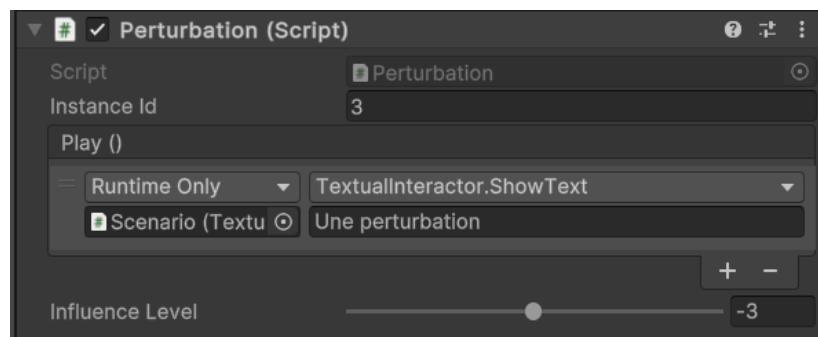
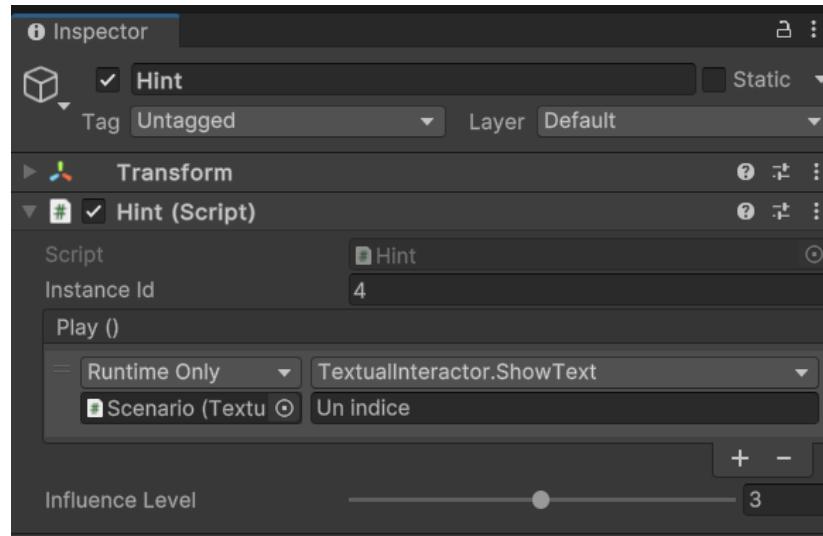
Ce script permet également de paramétriser comment l'objet devra réagir à la réception d'indice. Par exemple, dans le cas d'une télévision, celle-ci doit s'allumer et potentiellement faire un bruit.

Il faut également définir quel interacteur utilisé pour les messages envoyés par le maître de jeu. Cela se fait via l'objet suivant :



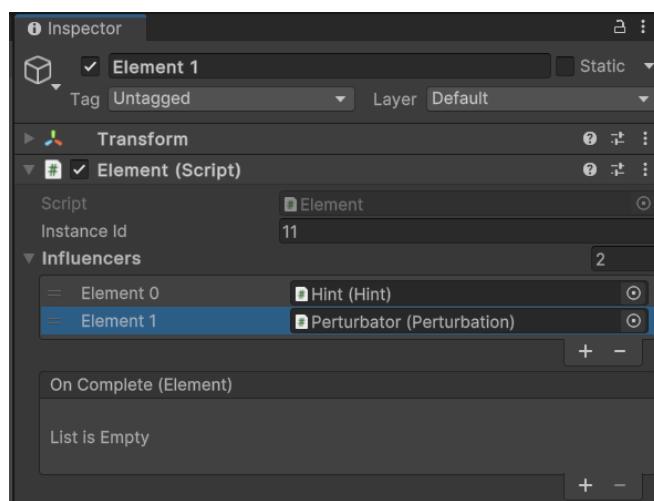
Il faut glisser déposer l'interacteur dans l'encadré rouge.

Une fois cela en place, il faut créer les indices et perturbateurs. Pour cela il faut créer des GameObject ayant les scripts « Hint.cs » ou « Perturbation.cs ».



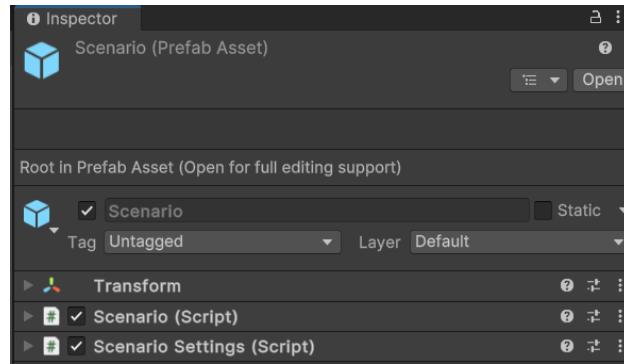
Il est possible de spécifier le niveau d'influence de l'indice ou de la perturbation à l'aide du slider. Cela permettra au gestionnaire intelligent d'envoyer un indice adapté à la situation.

Enfin, les indices et perturbateurs doivent être ajouté à un certain élément.

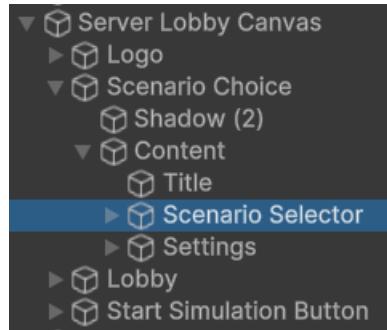


#### Étape 6 : Ajout du scénario dans le carrousel des choix disponible

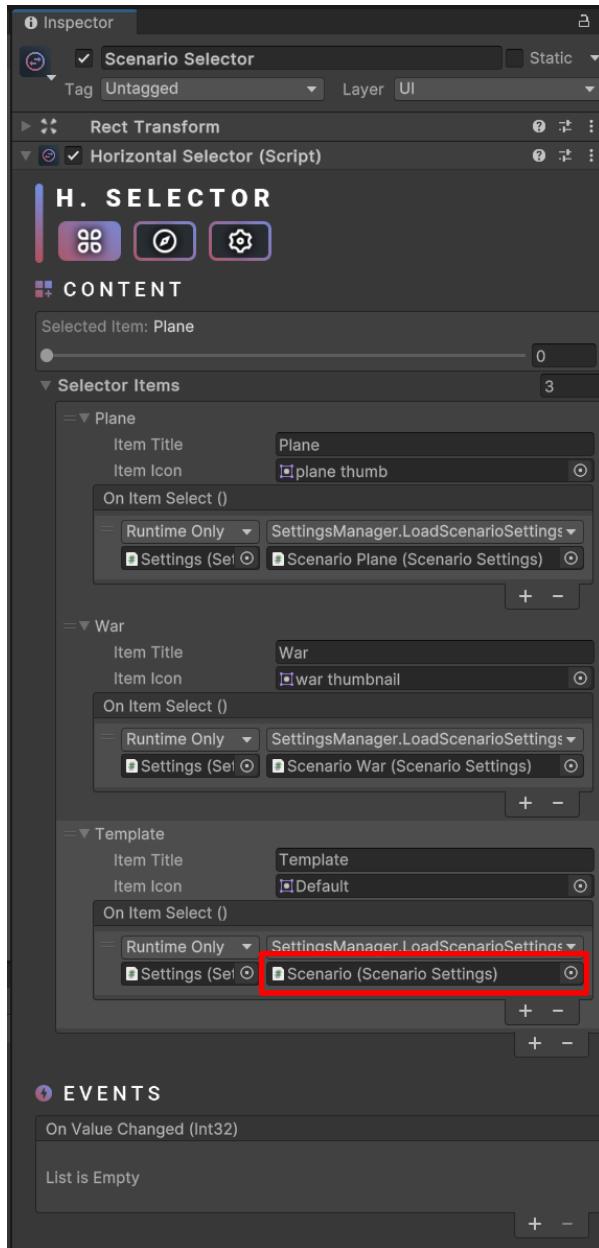
Avant tout de chose, il est nécessaire de transformer le GameObject ayant le script « Scenario.cs » en prefab.



Dans la scène « 01\_Lobby », disponible au chemin : ‘Assets/Host/Scenes/01\_Lobby.unity’, il faut atteindre l’objet suivant :



Dans l’inspecteur du Scenario Selector, il faut ajouter un item dans la liste, spécifié son nom ainsi qu’une image, et créer l’événement comme pour les scénarios pré existant.



Dans l'encadré rouge, il est nécessaire de glisser déposer la prefab créée au préalable.

#### **Étape facultative : Mise en place de logique réseau spécifique à un scénario**

Si, pour les besoins du scénario, il est nécessaire de faire des actions nécessitant une synchronisation plus fréquente qu'avec les événements mis en place dans la structure du scénario, il est possible de programmer la fonctionnalité à l'aide des RPC.

Par exemple, dans le scénario existant de l'hôpital désaffecté, une phase de monitoring est à faire par les joueurs. Ils doivent appuyer sur le bon bouton, qui est décidé aléatoirement par le serveur, plusieurs fois de suite. Pour ce faire, un script a été créé héritant de « HostNetworkRPC.cs ».

```

    ↗ Script Unity (1 référence de ressource) | 0 références
public class MonitoringRidle : HostNetworkRPC
{
}

```

Puis, il faut implémenter la logique tout en envoyant les informations aux autres clients sur le réseau.

```

public void SendPushedButton(int id)
{
    if (FMNetworkManager.instance.NetworkType == FMNetworkType.Server)
    {
        ButtonClicked(id);
    }
    else
    {
        HostNetworkManager.instance.SendRPC(new HostNetworkRPCMessage()
        {
            InstanceId = this.InstanceId,
            MethodName = "ButtonClicked",
            Parameters = new object[] { id}
        });
    }
}

```

Ici on voit le code qui permet au client d'envoyer quel bouton a été appuyé au serveur.

```

public void SendMonitoring()
{
    if(FMNetworkManager.instance.NetworkType == FMNetworkType.Server)
    {
        if (!hasBeenPressed)
        {
            Feedback(false);
        }
        currentButton = Random.Range(1, 6);
        HostNetworkManager.instance.SendRPC(new HostNetworkRPCMessage()
        {
            InstanceId = this.InstanceId,
            MethodName = "ActiveButton",
            Parameters = new object[] { currentButton}
        });

        ActiveButton(currentButton);
    }
    Invoke("SendMonitoring", 10);
}

```

La fonction s'exécutant toutes les 10 secondes et qui envoie un bouton aléatoire à cliquer à tous les joueurs.

```

private void Feedback(bool success)
{
    if (FMNetworkManager.instance.NetworkType == FMNetworkType.Server)
    {
        HostNetworkManager.instance.SendRPC(new HostNetworkRPCMessage()
        {
            InstanceId = this.InstanceId,
            MethodName = "IsSuccess",
            Parameters = new object[] { success }
        });

        IsSuccess(success);
    }
}

```

La fonction qui permet d'afficher le feedback permettant de communiquer aux joueurs s'ils ont appuyé sur le bon bouton ou non.