

# Documentation TER

Abel Henry-Lapassat

## -----SOMMAIRE

### Introduction -

- 1) Présentation Projet
- 2) Evolution Projet
- 3) Etat Actuel

### Outils Utilisés -

- 1) Unity
- 2) Photon
- 3) HTC-Vive & SteamVR

### Fonctionnalités Principale -

- 1) Salle (Unity + Photon)
- 2) Players (VR + 3D)
- 3) Operator (3D)
- 4) Mode Expe
- 5) Mode Demo
- 6) Fichier d'instructions

### Scripts Correspondants -

- 1) Rendering.cs / NetworkManager.cs / PlayerSpawner.cs
- 2) NetworkPlayer.cs / Teleporter.cs / DragDrop.cs
- 3) NetworkOperator.cs / Rendering.cs / OperatingMenu.cs
- 4) Rendering.cs / Expe.cs / Trial.cs / logs...
- 5) Rendering.cs
- 6) Assets → Resources → CreatorFiles.py ...

### Script d'utilisation -

- 1) Récupération du code
- 2) Version 2 Player
- 3) Version Operator + 2 Player

### Guide & Tips -

- 1) Unity
- 2) Photon
- 3) Potentielles Extensions

### Présentation Projet -

Ce Projet a tout d'abord été initié lors du stage de Master 1 de Camille Dupre sous la tutelle de Olivier Chapuis dans le Laboratoire International des Sciences du Numérique, au sein de l'équipe ILDA. C'est un projet effectué sous Unity, ayant pour but d'effectuer une expérience de manipulation de donnée en réalité Virtuelle. Ce projet a ensuite été modifié par Julien Berry lors de son stage en L3 Magistère, encore supervisé par Olivier Chapuis, utilisant les mêmes outils pour un objectif différent. Et enfin il a dernièrement été repris par moi-même (Abel Henry-Lapassat) lors de mon stage en L3 Magistère, toujours sous la tutelle de monsieur Chapuis, pour un objectif encore une fois différent, en gardant encore les mêmes outils de travail.

### Evolution Projet -

Lors du stage de Camille, l'objectif du projet était de proposer une expérience de manipulation d'images (dixit cards) comme il pourrait être fait sur un Wall Display, mais cette fois-ci en réalité virtuelle. Cette expérience impliquait deux joueurs, devant chacun placer certaines des cartes proposées dans un certain ordre afin de créer sa propre histoire. Les deux joueurs participaient à cette expérience en étant Co-Localisés et immobile, avec chacun leur casque VR ainsi que les controllers associés. Et ils évoluaient dans une salle constituée de 3 murs seulement afin de pouvoir s'orienter correctement tout au long de l'expérience, avec une récolte des logs de leurs déplacements & actions.

(PHOTO DU RENDU VISUEL DU STAGE DE CAMILLE)

L'objectif du stage de Julien était différent, cette fois-ci le but était d'implémenter différents modes de déplacements pour les joueurs, tels que la Téléportation, le déplacement par JoyStick, ou encore le Drag n Drop. Tout cela dans le même environnement que pour l'expérience de Camille. Mais en implémentant cette fois-ci une expérience différente, qui constitue en une succession de recherche d'image, en changeant le rôle des joueurs (celui qui cherche et celui qui voit l'image) ainsi que le mode de déplacement utilisé, afin de pouvoir comparer les différents résultats et définir lequel des déplacements était potentiellement le meilleur.

(PHOTO DU RENDU VISUEL DU STAGE DE JULIEN)

Enfin les objectifs de mon stage étaient certes moins conséquent mais plus nombreux, en effet la première partie de mon stage n'a pas été autant orientée autour d'une expérience mais plutôt autour de l'ajout de nouvelles fonctionnalités utiles à l'usage de ce projet, comme l'implémentation d'un troisième participant, ayant non pas le rôle de Player, mais d'Opérateur. Avec pour spécificité de pouvoir tourner sous un OS tel que Linux (ne supportant pas la réalité virtuelle avec Unity). En gardant le fonctionnement initial du projet laissé par Julien, il était donc possible d'avoir un participant Opérateur (observant & contrôlant l'expérience), qui était le seul à pouvoir diriger l'expérience. Et dans un second temps, le but était d'implémenter un mode Demo dans lequel des participants pourraient essayer toutes les fonctionnalités implémentées librement.

(PHOTO DU RENDU VISUEL DE MON STAGE)

## Etat Actuel -

Le projet possède donc actuellement deux fonctionnements différents ; Le premier est similaire au projet laissé par Julien, un fonctionnement n'impliquant que deux joueurs, dans lequel il peuvent néanmoins expérimenter librement les modes de déplacements et la manipulation de carte, en gardant la possibilité de lancer une expérience.

Le deuxième fonctionnement est spécifique à mes ajouts au projet laissé par Julien, il s'agit du fonctionnement impliquant 3 utilisateurs, dont l'un d'eux étant possiblement sur une machine Linux (l'opérateur), et étant le seul à pouvoir diriger l'expérience, de son lancement à son arrêt. Et laissant les joueurs en simple démonstration tant qu'il le souhaite.

## -----OUTILS UTILISES

### Unity

Ce projet prend place exclusivement sous l'IDE Unity, et se sert de beaucoup des fonctionnalités propres à ce dernier. Telles que la notion de GameObjects, de MonoBehaviour, ainsi que de composants (propres à un GameObject) dont font partis les Script.

Il faut donc faire très attention car tout ne se situe pas dans les Scripts (C-Sharp), une grosse partie des comportements et des features du projet sont décrites, accessibles et utilisables seulement depuis l'interface Unity, d'autres sont accessibles par les deux biais, tandis qu'encore d'autres ne sont eux qu'accessibles uniquement via les Scripts, via une utilisation de programmation orientée objet.

Si les concepts d'Unity ne sont pas évidents, je met à disposition un petit guide (rapide et spécifique à ce projet) dans la toute dernière partie de cette documentation.

### Photon

Afin de gérer la partie "multi-user" de ce projet, est utilisée la bibliothèque Photon, dont les fonctionnalités empruntées et utilisées pour ce projet sont les suivantes :

- Création d'une Photon Room faisant office de "serveur local" (plus dans la théorie que la pratique...)
- Attribution du titre de "MasterClient" à la première machine qui lance le programme.
- Instantiation de différents Photon "Players", deux types différents étant chacun régi par un script particulier : Operateur et Joueurs.
- Création et Destruction (selon les besoins) de plusieurs Photon "Room Objects", représentant toutes les cartes (dixit\_images) nécessaire à l'expe.
- Principe de "PhotonView" définissant globalement ce qui est propre à un objet photon spécifique.
- Principe d'Ownership afin de réguler correctement la manipulation des cartes (ainsi que leurs création et destruction).
- Spécificité d'un "MonoBehaviorPunCallbacks" pour certaines classes, visant à ajouter certaines fonctionnalités de photon aux classes bénéficiant déjà des fonctionnalités de Unity données par "MonoBehavior".
- Utilisation de fonctions "[PunRPC]" permettant la communication et l'utilisation de données photon particulières (PhotonView, Ids, Buffer ...).

La partie qui s'occupe de gérer le networking général du projet à heureusement déjà été implémenté par Camille, de ce côté là il n'y a donc pas grand chose à faire à part comprendre le fonctionnement de manière globale. Le point plus délicat est celui de l'utilisation des fonctionnalités de communications (MonoBehaviorPunCallbacks et PunRPC) de photon, si besoin

j'ai mis à disposition un petit guide (spécifiques aux fonctionnalités de Photon utilisées dans ce projet) sur la librairie Photon et ses fonctionnalités.

## **HTC-Vive & SteamVR**

L'une des parties de ce projet les plus importantes de ce projet est l'utilisation de la réalité virtuelle pour les deux participants "Joueurs", pour se faire nous utilisons des équipements HTC-Vive (headset + hands controllers), chacun pluggé à un des deux ordis de la salle (Ilda et West).

Les deux équipements sont déjà configurés, ils ne nécessitent que de vérifier leur branchements, alimentations, et ne pas toucher aux réglages.

Afin de gérer la réalité virtuelle sous Unity, on utilise le logiciel proposé par Steam, SteamVR. Pour se faire il est d'ors et déjà inclu dans le projet ainsi que sur les deux machines Ilda et West, pour l'utilisation d'une 3e machine (Opérateur potentiel) il n'y a rien à faire, simplement s'assurer que si elle ne supporte pas la réalité virtuelle, alors on ne lui demande pas de devoir gérer un des joueurs (qui nécessiterait de toute façon d'y plugger un équipement).

Attention néanmoins aux erreurs fréquentes de SteamVR, malheureusement SteamVR n'est pas d'une fiabilité sans faille, et il faut être prêts à en subir les conséquences... En soit rien de bien méchant, simplement vérifier en cas de bugs que tout est bien lancé, branché, allumé etc. Et ne pas hésiter en cas de soucis à redémarrer les équipements et machines si jamais les erreurs persistent.

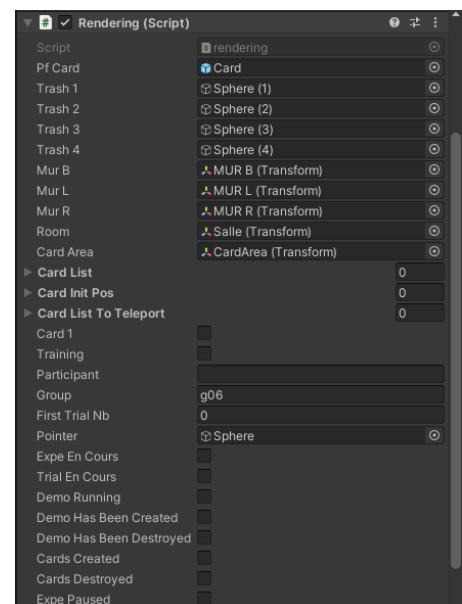
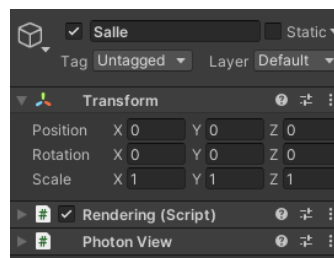
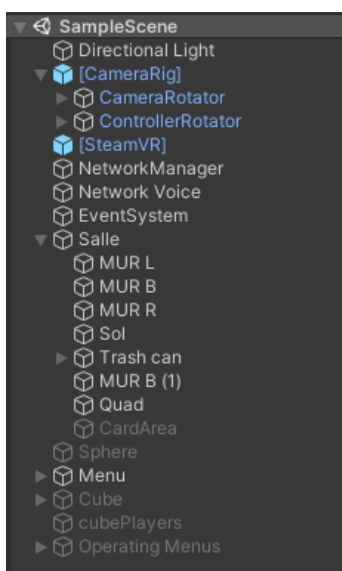
## -----FONCTIONNALITES PRINCIPALES

### Salle (Unity + Photon)

Ce projet a pour fonctionnalités centrale l'expérience implémentée. Cette dernière se déroule dans une salle virtuelle, au sein de laquelle le joueur pourra se déplacer de différentes manières (virtuelles uniquement), et manipuler les cartes mises à disposition. Cette salle a deux "représentations" différentes.

La première est la représentation visuelle (partie virtuelle) au sein de laquelle les joueurs ainsi que l'opérateur pourront évoluer et observer toutes les composantes de l'expérience. Cette dernière est composée de 3 murs uniquement, afin de permettre aux joueurs de ne pas perdre la perception de leur orientation à l'intérieur de cette salle.

Cette représentation est visible et implémentée principalement dans l'interface de Unity, sous la forme d'un GameObject "Salle" ayant 4 GameObjects enfants, avec chacun des attributs physiques tels que des MeshCollider, ou encore des BoxCollider, afin de pouvoir interagir avec eux d'un point de vue physique.



La seconde représentation est celle de Photon, en effet comme énoncé dans la partie "Photon" des "OUTILS UTILISES", on instancie une Photon Room afin de servir de serveur local (en quelques sortes) pour la salle en question. Ce qui permet tout simplement de gérer le nombre maximum de participants, et d'utiliser la fonctionnalités des "PhotonView" de manière exclusive à cette salle. Cette dernière n'est instanciée que lorsque ce projet précisément est exécuter, et n'est accessible – si déjà créée – si et seulement si on tente de s'y connecter en utilisant le code de ce projet.

```
//override methods
0 references
public override void OnConnectedToMaster(){
    base.OnConnectedToMaster();

    //setting the room's parameters
    RoomOptions room_options = new RoomOptions {
        MaxPlayers = 3,
        IsVisible = true,
        IsOpen = true
    };

    //creating the room
    PhotonNetwork.JoinOrCreateRoom("Room", room_options, TypedLobby.Default);
```

## Players (VR + 3D)

Dans ce projet, les joueurs participants à l'expérience en utilisant la Réalité Virtuelle, chacun des participants ayant le rôle de joueur se doit donc d'utiliser un équipement de VR (headset + controllers) grâce auquel il va pouvoir prendre part à l'expérience.

Le comportement et les interactions possibles du joueur avec l'expérience et ses composantes sont toutes décrites dans les Scripts (avec certaines des spécificités présentes dans l'IDE Unity). Voici donc la liste des fonctionnalités liées aux joueurs :

- Différents modes de déplacements (Teleportation, Drag, Joystick, Synchronisation)
- Manipulation de cartes (Tagging, déplacement, destruction)
- Visibilité de l'avatar du collaborateur (pas du sien)
- Choix du mode de déplacement
- Choix de la couleur du tag
- Direction de l'expe (si et seulement si aucun opérateur dans le programme)

Les possibilités d'interactions du joueur avec le programme sont dictées (activité ou non, dictée ou libre, etc) selon le script suivi (demos, expes, avec/sans operateur ...), mais sont toutes implémentées et fonctionnelles normalement sans erreurs.

## Operator (3D)

L'opérateur quant à lui a un mode de fonctionnement qui diffère de celui des joueurs, en effet pour commencer il n'a accès à la scène et à l'expe que d'un point de vue extérieure, en 3D uniquement. Ses interactions sont quant à elles plus générales et orientées sur la direction de l'expérience et de ce qu'il se passe dans le programme :

- Lancement d'une demo (affichage d'images sans contraintes envers le comportement des joueurs).
- Arrêt d'une demo (destruction des images toujours sans contraintes sur les joueurs).
- Lancement d'une experience (à n'importe quel moment, affichage d'images en contraignant les joueurs à suivre l'expe qui s'en suit).
- Arrêt d'une expe (destruction des images et annulation des contraintes envers les joueurs).
- Arrêt complet du programme.
- Téléportation des joueurs au centre de la pièce.
- Déplacement libre dans toute la scène en vue 3D
- Lock de sa caméra activable/désactivable.

### (PHOTO DE LA VUE DE L'OPERATEUR SUR UNE EXPE A 2 JOUEURS)

L'opérateur lorsqu'il est présent se doit de prendre les décisions, et est le seul à pouvoir les prendre, il est omniscient, voyant et dirigeant tout ce qui se déroule sous ses yeux, selon un seul script d'agissement, quant aux possibilités de lancement et d'arrêt de demo/expe, mais aucune contrainte majeure ne lui est appliquée.

## Mode Expe

Le mode expe est le mode permettant aux joueurs de participer à une expérience scriptée, au cours de laquelle le but sera de récolter tout un tas d'informations sur leur agissements, le but de l'expe est de tester les différentes réactions et aisance des participants en utilisant différents modes de déplacement VR. Cette dernière est découpée en plusieurs Trials, chacun décrits dans un fichier d'instructions utilisé pour instancier une expérience.

Lors d'un trial, chaque joueur se voit attribué un rôle et un mode de déplacement spécifique. L'un des deux joueurs sera "Observateur" et l'autre "Chercheur", le joueur observateur verra l'une des cartes affichées mise en évidence, et devra guider sans pouvoir bouger ni utiliser son contrôleur, indiquer au joueur "Chercheur" la carte en question. Le joueur "Chercheur" quant à lui

est donc libre de se déplacer dans la scène selon le mode de déplacement qui lui est attribué dans ce Trial, et doit se diriger vers la carte vue par l' "Observateur" afin de cliquer dessus et de terminer le Trial en question.

(PHOTO DE S ECRANS DES JOUEURS EN MODE EXPE AU DEBUT DU TRIAL)

Chaque Trial n'est autre qu'une inversion des rôles "Chercheur" a "Observateur", avec des modes de déplacements attitrés pour un certains nombre de Trial à la suite.

## Mode Demo

Le mode Demo pour sa part est encore plus simple, les joueurs ont accès à leur palette de contrôle et peuvent librement expérimenter les différents déplacements implémentés ainsi que la manipulation des cartes mises à leur disposition.

Aucune information n'est récupérée de leur agissement pour le moment, c'est un mode prévu simplement au test des fonctionnalités implémentées pour les utilisateurs qui désireraient se servir du programme sans avoir à essayer toute l'expe ni à farfouiller dans le code.

(PHOTO D'UNE MANIPULATION DE CARTE ET DE LA PALETTE D'UN JOUEUR)

## Fichier d'instructions

Les fichiers d'instructions sont des fichiers générés par des programmes python (très simples), qui contiennent les informations d'une expérience, ils sont de la forme suivant :

Voici le rendu du ficheir d'instruction obtenu grâce au script python (Assets/Resources/Experiment/\*.py) :

```
Group;Participant;CollabEnvironememn;trialNb;training;MoveMode;wall;CardToTag;  
g01;ope;C;0;1;drag;none;B;22;  
g01;p01;C;0;1;drag;move;B;22;  
g01;p02;C;0;1;drag;search;B;22;
```

Ce fichier est celui utilisé pour les expériences à 3 participants (ope, p01, p02), et chaque ligne décrit les contraintes & informations du participant concerné, comme indiqué en 1re ligne.

Ces fichiers permettent donc de dicter les scripts d'une experience de manière pseudo-aléatoire et de voir à l'avance qu'est ce qu'il s'y passe avec précision.

## -----SCRIPTS CORRESPONDANTS

Les scripts montrés et expliqués ne sont évidemment pas les seuls faisant appels aux fonctionnalités correspondantes, mais sont ceux permettant de décrire les principaux fonctionnements de ces dernières, et sont brièvement expliqués afin de permettre une compréhension plus efficace et rapide de ce projet...

Pour beaucoup des fonctionnalités énoncées, une certaine partie de leur description et instantiation est présente dans l'interface Unity, ce que je vais essayer de décrire tant bien que mal ci-dessous, en parallèle de la description dans les scripts C#.

Tous ces scripts se trouvent dans le dossier Assets/Script/ du projet.

### Salle

Toutes les caractéristiques principales des deux représentations de la salle se retrouvent dans les scripts suivants :

- NetworkManager.cs contient dans un premier temps toutes les informations primaires liées au réseau de la Photon Room, et s'occupe d'instantier la salle et ses caractéristiques en tant que "serveur local", et redirige les machines qui se connectent au programme vers les autres fonctions s'occupant d'instantier les objets Photons nécessaires.

```
public class NetworkManager : MonoBehaviourPunCallbacks {

    //Unity Awake method, called before any other ones
    0 references
    void Awake(){
        //Syncing the scenes
        PhotonNetwork.AutomaticallySyncScene = true;
    }

    //Unity Start method, used here to connect the machine to the server & room
    0 references
    void Start(){
        Connect();
    }

    //override methods
    0 references
    public override void OnConnectedToMaster(){
        base.OnConnectedToMaster();

        //setting the room's parameters
        RoomOptions room_options = new RoomOptions {
            MaxPlayers = 3,
            IsVisible = true,
            IsOpen = true
        };

        //creating the room
        PhotonNetwork.JoinOrCreateRoom("Room", room_options, TypedLobby.Default);
    }

    0 references
    public override void OnJoinedRoom(){
        base.OnJoinedRoom();
    }

    0 references
    public override void OnPlayerEnteredRoom(Player new_player){
        base.OnPlayerEnteredRoom(new_player);
    }

    //connect method
    1 reference
    private void Connect(){
        PhotonNetwork.ConnectUsingSettings();
    }
}
```



- PlayerSpawner.cs, qui fait suite à NetworkManager.cs, et s'occupe d'instantier les objets Photon correspondants aux joueurs & à l'opérateur, selon la configuration voulue, et le nombre de machines déjà connectées au programme.

```
public class Spawner : MonoBehaviourPunCallbacks {
    //entities prefabs
    3 references
    private GameObject spawned_player_prefab;
    2 references
    private GameObject spawned_operator_prefab;

    //operator predicate
    1 reference
    private bool with_ope = true;

    //override methods
    0 references
    public override void OnJoinedRoom(){
        base.OnJoinedRoom();
        if(PhotonNetwork.LocalPlayer.IsMasterClient){
            if(with_ope){
                spawned_operator_prefab = PhotonNetwork.Instantiate("Operator", new Vector3(0,0,0), transform.rotation);
            } else {
                spawned_player_prefab = PhotonNetwork.Instantiate("Player", new Vector3(0,0,0), transform.rotation);
            }
        } else {
            spawned_player_prefab = PhotonNetwork.Instantiate("Player", new Vector3(0,0,0), transform.rotation);
        }
    }

    0 references
    public override void OnLeftRoom(){
        base.OnLeftRoom();
        PhotonNetwork.Destroy(spawned_player_prefab);
        PhotonNetwork.Destroy(spawned_operator_prefab);
    }
}
```

- Rendering.cs contient quant à lui toutes les informations relatives aux comportements des composantes du projet, notamment la classe MyCard correspondant aux dixit\_cards qu'on affiche tout au long de l'expe, ainsi que les fonctions de création & destruction de ces dernières.

```
public class MyCard : MonoBehaviour {
    //specific card attributes
    0 references
    private GameObject go_card = null;
    1 reference
    private string pos_tag = "";
    1 reference
    private PhotonView pv;
    1 reference
    private Transform parent;

    1 reference
    public MyCard(Texture2D tex, Transform wall){
        GameObject go_card = PhotonNetwork.InstantiateRoomObject("Card", wall.position, wall.position, 0, null);
        go_card.GetComponent<Renderer>().material.SetTexture("_MainTex", tex);
        parent = wall;
        pv = go_card.GetComponent<PhotonView>();
        pos_tag = "onWall";
    }
}
```

```
[PunRPC]
0 references
private void LoadCards(int card_pv_id, int wall_pv_id, int pos, int card_index){
    //loading textures
    if(textures==null){
        textures = Resources.LoadAll("dixit_all/", typeof(Texture2D));
    }
    //defining spacing attribute
    card_spacing = 0.7f / (card_per_line - ((int)(rendering.card_per_wall%2)));

    //getting the concerned wall & card
    Transform wall = PhotonView.Find(wall_pv_id).transform;
    GameObject card = PhotonView.Find(card_pv_id).gameObject;

    //setting the texture
    Texture2D tex = (Texture2D)textures[card_index];
    card.GetComponent<Renderer>().material.SetTexture("_MainTex", tex);

    //setting the card's attributes
    card.transform.parent = wall;
    card.transform.rotation = wall.rotation;
    card.name = "Card " + card_index;
    card.transform.localScale = new Vector3(width, height, 1.0f);

    //setting card's position
    if(pos < card_per_line){
        card.transform.localPosition = new Vector3(min_card_x + width + card_spacing * pos, -height, -0.01f);
    } else {
        pos = pos - card_per_line;
        card.transform.localPosition = new Vector3(min_card_x + width + card_spacing * pos, height, -0.01f);
    }
}
```

Mais aussi toutes les fonctions permettant de gérer les éléments contextuels du programme, comme la demo ou l'expe, en faisant appel aux fonctions s'occupant de la création, du lancement, et de l'arrêt de ces éléments.

```
[PunRPC]
0 references
public void StartExperiment(string grp, int nb){
    participant = SetParticipantName();
    experiment = new Experiment(participant, grp, nb, card_list, with_ope);

    if(experiment.current_trial.collab_env == "C"){
        GameObject sound = GameObject.Find("Network Voice");
        sound.SetActive(false);
    }
}
```

Et enfin sont aussi présentes dans ce script, les fonctions appelée par l'opérateur lorsqu'il effectue un KeyStroke sur les touches données (E,T,D,Space) interagissant avec la demo & l'expe, ces fonctions seront montrées dans la partie Operator.

## Players

Pour ce qui est des caractéristiques des joueurs, elles sont découpées en 3 catégories, chacune gérée par un script différent :

- Network\_Player.cs qui contient l'initialisation d'un Player, en récupérant et instantiant tous les GameObjects nécessaire au fonctionnement des features d'un Player (Palette, VR, Déplacements ...), et contient les fonctions principales don't un joueur aura besoin pour accéder à ces features.

```
void Start(){
    if(GameObject.Find("/Network Operator(Clone)")!=null){
        //if we get an operator then we are instantiating the referred operator as the existing one
        ope = GameObject.Find("/Network Operator(Clone)");
        ope_script = ope.GetComponent<NetworkOperator>();
        is_operator = false;
    } else {
        //and if we don't get any operator then this player is directly referring to the room
        room = GameObject.Find("/Room");
        room_render = room.GetComponent<Rendering>();
        //if the instantiated player is the first one to join & this is well its view then he become the operator
        is_operator = (photonView.IsMine && PhotonNetwork.LocalPlayer.ActorNumber==1);
    }

    //getting the VR attributes
    cam_rig = GameObject.Find("/[CameraRig]");
    headset = GameObject.Find("Camera (eye)");
    ctrl_right = GameObject.Find("/[CameraRig]/ControllerRotator/Controller (right)");
    ctrl_left = GameObject.Find("/[CameraRig]/ControllerRotator/Controller (left)");

    //getting the right controller's pose
    pose = ctrl_right?.GetComponent<SteamVR_Behaviour_Pose>();

    left_hand.gameObject.SetActive(true);
    palette.gameObject.SetActive(true);

    //unshowing my avatar for myself
    bool pv_is_mine = photonView.IsMine;
    left_hand.gameObject.SetActive(!pv_is_mine);
    right_hand.gameObject.SetActive(!pv_is_mine);
    head.gameObject.SetActive(!pv_is_mine);
    chest.gameObject.SetActive(!pv_is_mine);

    ray_name = ray_cast.GetComponent<Renderer>().material.name.ToString();
}
```

Ce Script s'occupe aussi de constamment vérifier (via la fonction Update() de Unity) l'état du joueur et lui permettre d'effectuer les actions correspondantes à son état, en faisant appel à des fonctions spécifiques selon son état.

```
void Update(){
    if(is_operator){
        OperatorActions();
    }

    if(experiment==null && room_render.expe!=null){
        //whilst we are inside an experiment then the ability to switch the move mode is disabled
        expe = room_render.expe;
        palette.gameObject.SetActive(false);
    }

    //checking if the other either I or the other player is synced
    bool? sync_tag_test = GameObject.Find("[CameraRig]/ControllerRotator/Controller (right)")?.GetComponent<Teleporter>().tag_sync;
    bool? is_other_synced_test = GameObject.Find("[CameraRig]/ControllerRotator/Controller (right)")?.GetComponent<Teleporter>().is_other_synced;

    sync_tag = sync_tag_test == null ? false : (bool)sync_tag_test;
    is_other_synced = is_other_synced_test == null ? false : (bool)is_other_synced_test;

    if(photonView.IsMine){
        if(pun_view){
            pun_view = false;
        }
        MapPosition();
    }

    //simulating new ray
    Ray ray = new Ray(ctrl_right.transform.position, ctrl_right.transform.forward);

    //and now checking its conditions & situations
    if(Physics.Raycast(ray, out hit)){
        //checking to change either color of raycast or move_mode
        if(trigger.GetStateDown(pose.inputSource)){
            if(hit.transform.tag=="Color tag" && (sync_tag || photonView.IsMine)){
                //then we change the color of the ray
                UpdateRayColor();
            } else if(photonView.IsMine){
                //then we change the move mode of THIS player
                UpdatePalette(hit.transform.tag);
                photonView.RPC("ChangeMoveMode", Photon.Pun.RpcTarget.all, move_mode);
            }
        }
    }
}
```

- Teleporter.cs contient toutes les informations nécessaire au déplacement du joueur, selon le mode de mouvement, ce Script s'occupe d'appliquer un comportement particulier selon les inputs du joueur grâce à son Controller droit.

### (SCHEMA DES INPUTS POSSIBLES SELON LES MOVEMODE ?)

- DragDrop.cs est le Script permettant au joueur de manipuler les cartes mises à disposition dans le programme, pour se faire le Script regarde la position du pointeur généré par le contrôleur droit du joueur et si elle est sur une carte, donne la possibilité au joueur de la déplacer en cliquant et gardant le trigger enfoncé tout en bougeant le pointeur. Ou de le tagger en cliquant simplement dessus.

## Operator

Les possibilités d'interactions de l'opérateur sont contenus dans les deux Scripts suivants :

- NetworkOperator.cs contient le script de l'update de l'operator, qui définit les différentes actions que ce dernier peut effectuer, et les modifications qu'il peut apporter au programme tout au long de son fonctionnement.

- Rendering.cs contient une parties des fonctions opératrices, en effet étant directement relié à la salle ce script contient des fonctions auxquelles seul l'opérateur peut faire appel lors de certaines actions, notamment les actions de Demo :

```

public void DPressed(){
    if(!demo_created){
        InstantiateCards();
        CreateCards();
        demo_created = true;
        demo_running = true;
    } else if(demo_running && !demo_destroyed){
        photonView.RPC("ResetCards", Photon.Pun.RpcTarget.All);
        demo_destroyed = true;
        demo_running = false;
    } else if(!demo_running && demo_created){
        photonView.RPC("ActivateCards", Photon.Pun.RpcTarget.All);
        demo_running = true;
    } else if(demo_running && demo_destroyed){
        photonView.RPC("ResetCards", Photon.Pun.RpcTarget.All);
        demo_running = false;
    }
}
}

```

Donc lorsqu'il presse la touche D, selon l'état de la demo (lancée, en cours, terminée ...)

alors cela a des effets précis :

- Si la demo n'a jamais été créée alors on instancie les cartes puis on les crée, en s'assurant d'actualiser les prédicats.

- Si la demo est en cours et n'a encore jamais été stoppée alors on appelle une méthode qui va reset les cartes, et actualiser les prédicats de la demo en conséquence.

- Si la demo n'est pas en cours mais a déjà été créée (ie a été stoppée) alors on appelle une méthode qui va simplement activer les cartes (et non pas les recréer), puis on actualise les prédicats.

- Enfin si la demo est en cours et a déjà été stoppée, alors on reset les cartes, la différence avec la 2e possibilité est que cette fois on actualise différemment les prédicats.

Et ce script contient aussi les actions d'Expe :

```

public void SpacePressed(){
    if(!expe_running){
        if(demo_running || demo_created){
            demo_running = false;
            photonView.RPC("ResetCards", Photon.Pun.RpcTarget.All);
            photonView.RPC("ActivateCards", Photon.Pun.RpcTarget.All);
            photonView.RPC("StartExperiment", Photon.Pun.RpcTarget.All);
            expe_running = true;
        } else {
            InstantiateCards();
            CreateCards();
            photonView.RPC("StartExperiment", Photon.Pun.RpcTarget.All);
            expe_running = true;
        }
    } else if(expe_running && !trial_running){
        photonView.RPC("NextTrial", Photon.Pun.RpcTarget.AllBuffered);
    }
}
}

```

Donc lorsqu'il presse la touche ESPACE, selon l'état de l'Expe (mêmes états que pour la demo), alors cela a des effets précis :

- Si l'expe n'est pas en cours, on regarde si une demo a déjà été lancée et est actuellement en cours, si c'est le cas alors l'expe vient 'Override' la demo et prendre sa place. Sinon on instancie les cartes. Dans les deux cas on appelle la fonction "StartExperiment" qui lance l'expérience.



→ Si l'expe est en cours mais qu'aucun trial n'est en cours alors on passe au trial suivant.

L'opérateur possède aussi la possibilité de se mouvoir grâce aux flèches directionnelles et la souris de son ordinateur, ainsi que de lock/unlock sa caméra, le code concernant ces possibilités se trouve dans la fonction Update de la classe "NetworkOperator" :

```
void Update(){
    if(experiment==null && room_render.experiment!=null){
        experiment = room_render.experiment;
    }

    if(!cam_locked){
        transform.Translate(Vector3.forward * 0.01f * Input.GetAxis("Vertical"));
        transform.Translate(Vector3.right * 0.01f * Input.GetAxis("Horizontal"));
        if(!MouseOffScreen()){
            rotation_x -= Input.GetAxis("Mouse Y")*sensitivity;
            rotation_y += Input.GetAxis("Mouse X") * sensitivity;
            rotation_x = Mathf.Clamp(rotationX, -90,90);
            transform.rotation = Quaternion.Euler(rotation_x,rotation_y,0);
        }
    }

    //all operator's inputs for specifics actions
    if(Input.GetKeyDown(KeyCode.L)){
        //locking the camera
        cam_locked != cam_locked;
    }
}
```

Et pour agir avec la room, il peut utiliser les touches détectées par ce code (espace, T, E, D):

```
if(Input.GetKeyDown(KeyCode.Space)){
    //triggering the room's experiment
    room_render.SpacePressed();
}
if(Input.GetKeyDown(KeyCode.E)){
    //ending the room's experiment
    room_render.EPressed();
}
if(Input.GetKeyDown(KeyCode.T)){
    //teleporting both players to each other (+ to center of the room)
    room_render.TPressed();
}
if(Input.GetKeyDown(KeyCode.D)){
    //triggering the room's demo
    room_render.DPressed();
}
```

## Expe

### (Rendering.cs / Expe.cs / Trial.cs / logs...)

Pour ce qui est de l'expérience, sa création et son actualisation se font dans la classe "Rendering.cs" via les actions qu'effectuera l'opérateur, et est donc dictée par les fonctions appelées par les méthodes précédemment évoquées "StartExperiment" et "NextTrial" :

```
[PunRPC]
0 references
public void StartExperiment(string grp, int nb){
    participant = SetParticipantName();
    experiment = new Experiment(participant, grp, nb, card_list, with_ope);

    if(experiment.current_trial.collab_env == "C"){
        GameObject sound = GameObject.Find("Network Voice");
        sound.SetActive(false);
    }
}
```

Avant de rentrer dans des explications de code, voici la principale différence entre l'instantiation d'une démo et celle d'une expérience :

La démo autorise l'accès aux cartes et au changement de mode de navigation, tandis que l'expérience l'interdit. Pour se faire, lorsqu'on instancie une expérience, la palette de contrôle des joueurs est désactivée de la manière suivante dans la fonction "Update" de "NetworkPlayer" :

```
if(experiment==null && room_render.expe!=null){
    //whilst we are inside an experiment then the ability to switch the move mode is disabled
    expe = room_render.expe;
    palette.gameObject.SetActive(false);
}
```

Et en ce qui concerne la possibilité de manipuler les cartes, c'est dans la fonction "Update" de la classe "DragDrop" qu'on la désactive, en la définissant directement comme non-activée :

```
if(experiment==null && room_render.experiment){
    experiment = GameObject.Find("/Room").GetComponent<Rendering>().experiment;
    enabled = false;
}
```

*Revenons maintenant à des explications de code plus spécifique à l'expérience.*

Donc la fonction "StartExperiment" appelle directement le constructeur de l'expérience et en instancie une nouvelle de la façon suivante :

```
public Experiment(string prt, string grp, int start_nb, List<GameObject> c_list, bool with_ope) {

    //variables instantiation
    expe_running = true;
    group = grp;
    participant = prt;
    trial_nb = start_nb;
    card_list = c_list;

    //getting the player's attributes
    player = GameObject.Find("Network Player(Clone)");
    player_script = player.GetComponent<Network Player>();
    controller = GameObject.Find("/[CameraRig]/ControllerRotator/Controller (right)");
    ctrl_tp = controller.GetComponent<Teleporter>();

    //getting the operator's room attribute depending on who's the operator (master client)
    int master_id = PhotonNetwork.MasterClient.ActorNumber;
    GameObject master = PhotonNetwork.Find(master_id).gameObject;
    room_render = SetRoomRender(with_ope);

    //initialization of log files for this expe
    string date = System.DateTime.Now.ToString("yyyy-MM-dd-HH-mm-ss");
    string path = "Assets/Resources/logs/class-" + group + "-" + participant + "-" + date + ".csv";
    writer = new StreamWriter(path, false);
    writer.WriteLine(
        "Group;Participant;CollabEnvironment;trialNb;training;MoveMode;Task;Wall;CardToTag;" +
        "nbMove;nbMoveWall;nbDragWallFloor;distTotal;nbRotate;rotateTotal;" +
        "trialTime;moveTime"
    );
    writer.Flush();

    path = "Assets/Resources/logs/class-" + participant + "-" + date + ".txt";
    kine_writer = new StreamWriter(path, false);
    expe_description_file = SetDescriptionFile(with_ope);

    TextAsset data_txt = (TextAsset) Resources.Load(expe_description_file);
    string txt = data_txt.text;
    List<string> lines = new List<string>(txt.Split('\n'));
    trials = new List<Trial>();

    foreach(string line in lines){
        List<string> values = new List<string>(line.Split(';'));
        if(values[0]=="pause" && trials.Count!=0 && trials[trials.Count-1].group==group){
            trials.Add(new Trial(this, values[0], "", "", "", "", "", "", "", ""));
        } else if(values[0]==group && values[1]==participant){
            trials.Add(new Trial(this, values[0], values[1], values[2], values[3],
                values[4], values[5], values[6], values[7], values[8]));
            trials[trials.Count-1].path_log = path;
            trials[trials.Count-1].kine_writer = kine_writer;
        }
    }
    current_trial = trials[trial_nb];
    kine_writer.WriteLine(current_trial.group + " " + current_trial.participant + " kine action");
    kine_writer.Flush();
}
```

Pour resumer ce bloc de code, le constructeur récupère les informations nécessaires (différents participants, informations de trial, cartes ...) puis instancie un fichier de logs ainsi qu'un fichier d'instructions grâce à des StreamWriter, qui vont écrire dans un fichier le descriptif des trials afin de dicter toute l'expérience, en faisant appel à un script python. Et suivants ces informations on ajoute les trials correspondant à la liste des trials de l'expérience.

Ensuite lorsque l'on appelle la méthode "NextTrial", cette dernière ne fait qu'appel à la fonction "NextTrial" de l'expérience :

```
[PunRPC]
0 references
public void NextTrial(){
    experiment.NextTrial();
}
```

Voici un visuel du code de cette fonction (peu lisible en effet), suivi d'une explication sur son fonctionnement :

```
public void NextTrial(){
    if(!trial_running){
        trial_running = true;
        SetInfoLocation();
        ctrl_tp.menu.SetActive(true);

        if(trials[trial_nb].task=="search"){
            if(trials[trial_nb].training=="1"){
                ctrl_tp.menu.transform.Find("moveModeText").GetComponent<TextMesh>().text = "Search \n Training";
            } else {
                ctrl_tp.menu.transform.Find("moveModeText").GetComponent<TextMesh>().text = "Search";
            }
            ctrl_tp.menu.transform.Find("textInfo").GetComponent<TextMesh>().text = "You are the one synchronized \n click to start the trial \n spot the card and tell the other";
        } else {
            if(trials[trial_nb].training=="1"){
                ctrl_tp.menu.transform.Find("moveModeText").GetComponent<TextMesh>().text = "Navigate " + theTrials[trialNb].moveMode + " \n Training";
            } else {
                ctrl_tp.menu.transform.Find("moveModeText").GetComponent<TextMesh>().text = "Navigate " + theTrials[trialNb].moveMode;
            }
            ctrl_tp.menu.transform.Find("textInfo").GetComponent<TextMesh>().text = "You are the one moving \n wait for the other to start \n let the other tell you where to go";
        }

        trials[trial_nb].StartTrial();
        current_trial = trials[trial_nb];
        trial_running = true;
    } else if(trial_nb==trials.Count-1){
        Write();
        trials[trial_nb-1].card.transform.GetChild(1).gameObject.SetActive(false);
        trials[trial_nb].card.transform.GetChild(1).gameObject.SetActive(false);
        Finish();
    } else {
        Write();
        IncrementTrialNb();

        if(trials[trial_nb].group=="#pause"){
            trial_running = false;
            ctrl_tp.photonView.RPC("ResetPosition", Photon.Pun.RpcTarget.AllBuffered);
            ctrl_tp.menu.transform.Find("textInfo").GetComponent<TextMesh>().text = "Pause";
            writer.WriteLine("#pause;");
            writer.Flush();
            IncrementTrialNb();
            ctrl_tp.menu.transform.Find("moveModeText").GetComponent<TextMesh>().text = "Next move " + theTrials[trialNb].moveMode;
            SetInfoLocation();
            ctrl_tp.menu.SetActive(true);
        } else {
            SetInfoLocation();

            if(trials[trial_nb].task=="search"){
                if(trials[trial_nb].training=="1"){
                    ctrl_tp.menu.transform.Find("moveModeText").GetComponent<TextMesh>().text = "Search \n Training";
                } else {
                    ctrl_tp.menu.transform.Find("moveModeText").GetComponent<TextMesh>().text = "Search";
                }
                ctrl_tp.menu.transform.Find("textInfo").GetComponent<TextMesh>().text = "You are the one synchronized \n click to start the trial \n spot the card and tell the other";
            } else {
                if(trials[trial_nb].training=="1"){
                    ctrl_tp.menu.transform.Find("moveModeText").GetComponent<TextMesh>().text = "Navigate " + theTrials[trialNb].moveMode + " \n Training";
                } else {
                    ctrl_tp.menu.transform.Find("moveModeText").GetComponent<TextMesh>().text = "Navigate " + theTrials[trialNb].moveMode;
                }
                ctrl_tp.menu.transform.Find("textInfo").GetComponent<TextMesh>().text = "You are the one moving \n wait for the other to start \n let the other tell you where to go";
            }
            ctrl_tp.menu.SetActive(true);
            trials[trial_nb].startTrial();
            current_trial = trials[trial_nb];
        }
    }
}
```

→ Si aucun trial n'est en cours, alors on initie les informations pour les joueurs (localisation + menus textuels), et on informe le joueur selon les informations récupérées dans le fichier d'instructions (plus précisément le mode de déplacement). Puis on lance le trial concerné (contenu dans la liste des trials de l'expérience).



→ Si le trial actuel est le dernier alors on termine l'expérience par l'appel à la méthode "Finish" de cette classe Experiment, qui actualise simplement les informations textuels ainsi que les prédicats de statut de l'expérience.

```
public void Finish(){
    Write();
    ctrl_tp.menu.transform.Find("textInfo").gameObject.SetActive(true);
    ctrl_tp.menu.transform.Find("textInfo").GetComponent<TextMesh>().text = "Fin de l'expérience";
    trial_running = false;
    expe_running = false;
    writer.Close();
    kineWriter.Close();
}
```

→ Sinon on actualise les informations pour les joueurs comme lors du 1er cas, puis on lance le trial qui suit l'actuel. En actualisant les informations nécessaires à l'expérience.

## Demo (Rendering.cs)

En ce qui concerne la demo, tout se trouve dans 'Rendering.cs' et la seule différence avec l'expérience, c'est qu'ici on n'empêche en aucun cas la manipulation des cartes ainsi que la possibilité d'interactions avec la palette pour les joueurs, tout se trouve donc dans la fonction "Dpressed", avec initialisation, désactivation, réactivation des cartes, grâce aux fonctions suivantes :

```
public void InstantiateCards(){
    if(training){
        textures = Resources.LoadAll("dixit_training/", typeof(Texture2D));
    } else {
        textures = Resources.LoadAll("dixit_all/", typeof(Texture2D));
    }
}
```

```
public void CreateCards(){
    Transform wall_object;
    int on_wall_pos;
    string wall_label;

    for(int i=0; i< card_per_wall*3; i++){
        if(i< textures.Length){
            if(i< card_per_wall){
                wall_object = LeftWall;
                wall_label = "L";
                on_wall_pos = i;
            } else if(i< 2*card_per_wall){
                wall_object = BackWall;
                wall_label = "B";
                on_wall_pos = i - card_per_wall;
            } else {
                wall_object = RightWall;
                wall_label = "R";
                on_wall_pos = i - 2*card_per_wall;
            }

            Texture2D texture = (Texture2D)textures[i];
            MyCard card = new MyCard(texture, wall_object, i);
            photonView.RPC("AddCardToList", Photon.Pun.RpcTarget.AllBuffered);
            card.pv.RPC("LoadCard", Photon.Pun.RpcTarget.AllBuffered, card.pv.ViewID, wall_object.GetComponent<PhotonView>().ViewID, wall_pos, i);
            photonView.RPC("AddPosToList", Photon.Pun.RpcTarget.AllBuffered);

            GameObject card_object = PhotonView.Find(card.pv.ViewID).gameObject;
            Vector3 card_scale = card_object.transform.localScale;
        } else {
            break;
        }
        cards_created = true;
    }
}
```

→ Script d'instantiation et de création des cartes, ici on récupère les textures des dixit\_cards, puis on les répartie sur les 3 murs visibles par les participants, et enfin on crée la carte en question avant de l'ajouter à la liste des cartes. En parallèle de cela on ajoute les positions



initiales de chacune des cartes dans la salle à une liste qui servira à les repositionner dès que nécessaire.

Je ne montre pas les script des fonctions “LoadCard” ni du constructeur d’une carte car ils ne sont pas primordiaux à la compréhension du fonctionnement de la démo.

```
[PunRPC]
0 references
public void ResetCards(){
    Transform wall;
    GameObject card;
    //PhotonView card_pv;
    //int pv;
    for(int i=0; i<60; i++){
        card = card_list[i];

        if(i < card_per_wall){
            wall = LeftWall;
        } else if(i < 2* card_per_wall){
            wall = BackWall;
        } else {
            wall = RightWall;
        }

        card.transform.parent = wall;
        card.transform.rotation = wall.rotation;
        card.transform.localScale = new Vector3(0.033f, 0.239f, 1.0f);

        if(card_init_pos[i]!=null){
            card.transform.localPosition = card_init_pos[i];
        }

        card.SetActive(false);
    }
}
```

→ Script de réinitialisation des cartes, dans lequel on itère sur chacune des cartes de la liste, afin d’individuellement les remettre à leur position initiale en répétant le même procédé que lors de leur instantiation, et on récupère la position initiale dans la room dans la liste les contenant. Enfin on désactive la carte visuellement en attendant sa réactivation.

```
[PunRPC]
0 references
public void ActivateCards(){
    //GameObject card_pv;
    //int pv;
    foreach(GameObject card in card_list){
        card.SetActive(true);
    }
}
```

→ Script d’activation des cartes, dans lequel on itère simplement sur toutes les cartes de la liste afin de les réactiver visuellement, sachant qu’elles ont déjà toutes été repositionnée par la fonction “ResetCards”.

## Fichiers d’instructions (Assets → Resources → CreatorFiles.py ...)

Le code de ces programme n’est pas essentiel à la compréhension du projet mais peut servir lors de l’implémentation d’autres expérience, afin de définir de nouveaux fichiers d’instructions par exemple, mais c’est du python assez basique pour écrire dans un nouveau fichier ligne par ligne des informations générée pseudo-aléatoirement.

Voici donc leur localisation afin de pouvoir aller les consulter si besoin :

*Assets/Resources/CreatorFiles.py* (sans operateur)

*Assets/Resources/textDataCreator.py* (avec operateur)

## -----SCRIPT D'UTILISATION

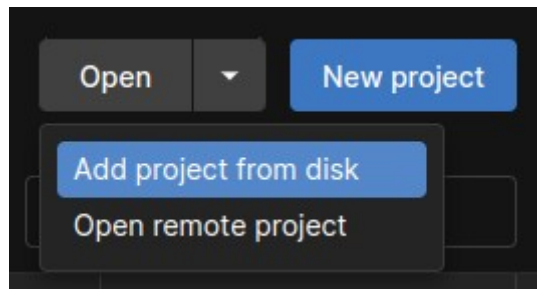
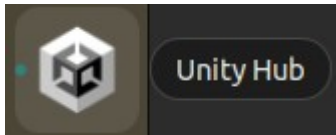
### Script de récupération du code

- 1) Récupérer le code du projet au lien suivant fourni grâce à la commande git ci-dessous :

```
~$ git clone --branch Viable https://github.com/Jvaljer/TERL3
```

- 2) Importer le projet sur UnityHub

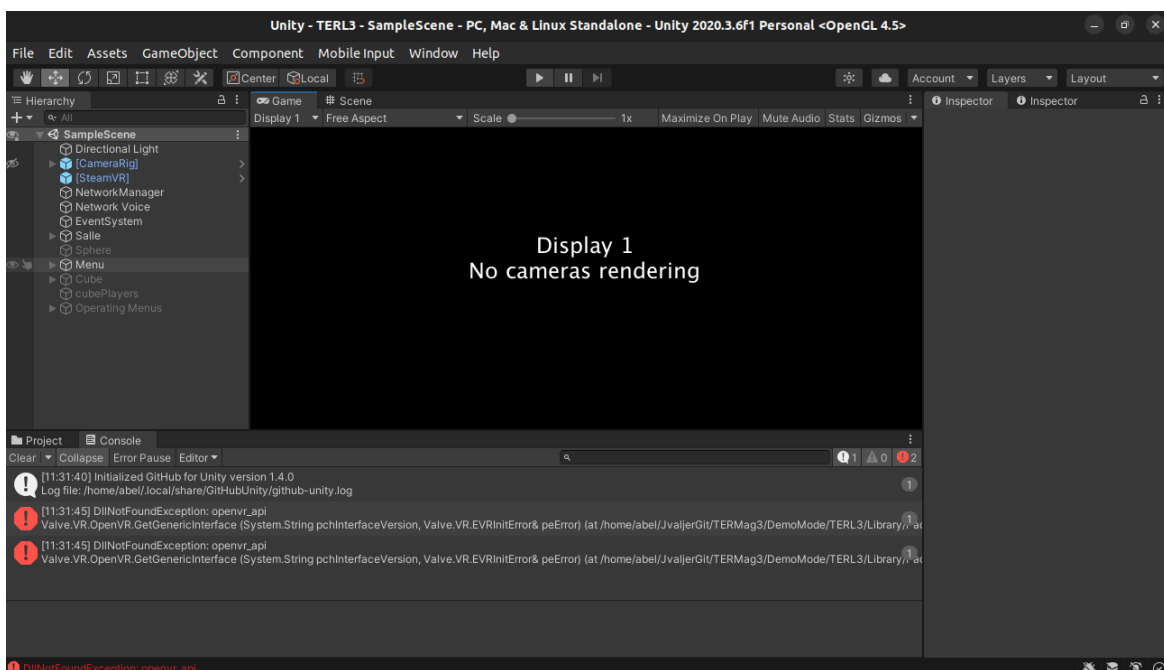
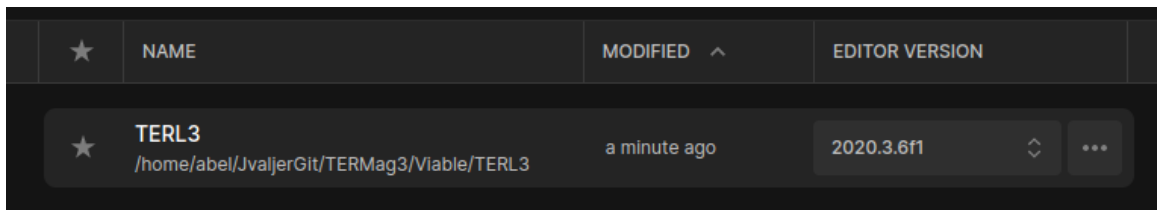
UnityHub → Open → Add Project from disk → TERL3



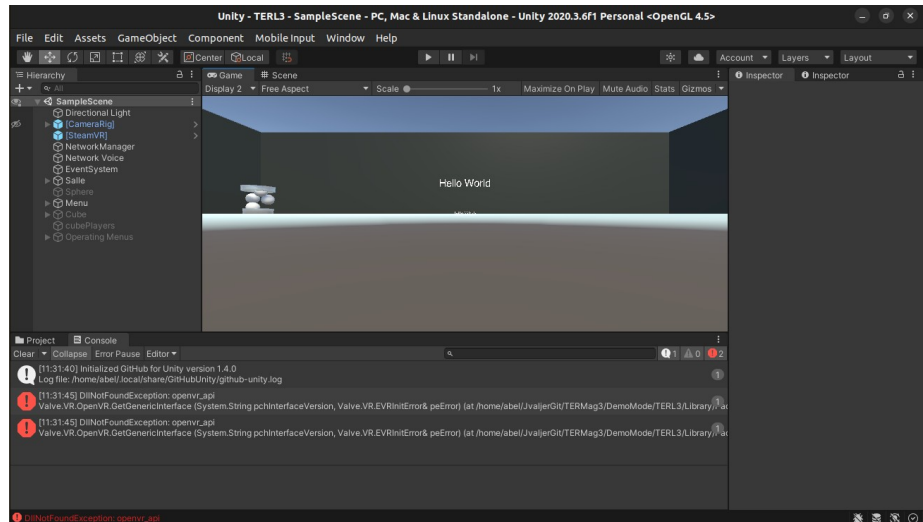
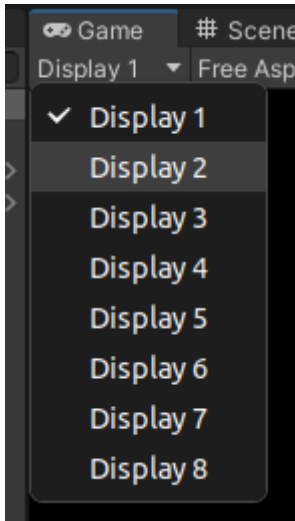
- 3) Répéter les étapes 1) & 2) sur toutes les machines voulues.

### Script Pour 2 Joueurs

Preliminaire → Lancer le projet sur toutes les machines

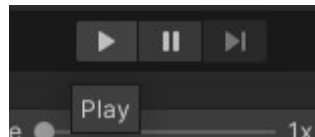


- 1) S'assurer de bien être sur le 'Display 2' sur les deux machines



2) Vérifier que SteamVR fonctionne correctement (bien lancé etc)

3) 'Play' le projet sur la machine que l'on souhaite définir comme opératrice de (même si le participant est joueur)



4) 'Play' le projet sur la deuxième machine → *les deux joueurs peuvent se voir si tout va bien*

5) Suivre les instructions d'utilisation suivantes avec la machine opératrice :

- touche D → lance/stoppe une démo (manipulation de carte activée + navigation libre)
- touche Espace → lance une expe/ passe au trial suivant
- touche E → arrête l'expe en cours
- touche T → téléporte les deux joueurs l'un à l'autre (centre de la pièce)

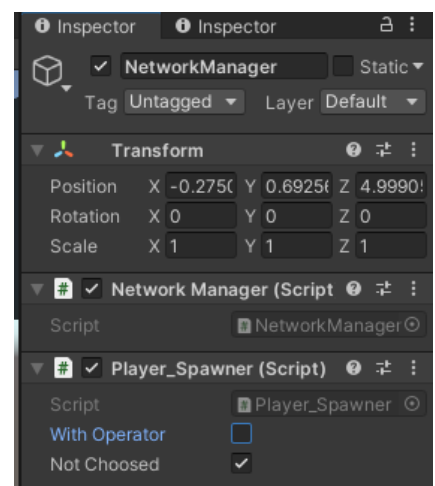
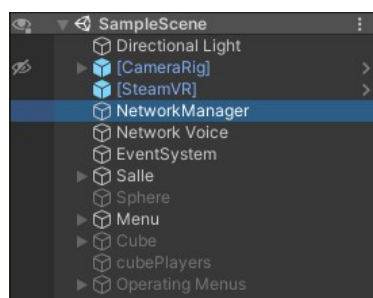
6) Le reste se passe dans le casque

## Script Pour Operateur + 2 Joueurs

Suivre le même script que pour l'utilisation à 2 joueurs, seulement cette fois on lance la machine opératrice avant les deux autres, et on laisse cette dernière sur 'Display 1', et c'est elle qui aura accès aux touches opératrices.

Pour choisir d'être avec ou sans opérateur, il faut (via l'interface Unity) activé le prédicat de présence d'opérateur :

*NetworkManager* → *Inspector* →  
*Player\_Spawner* → "With Operator"



**Unity**

**Photon**

**Potentielles Extensions**