

# RAPPORT DE PROJET



EITM

Gianluca Minoprio, Rehan Khan,  
Lina Diringer, Hector De Turckheim

[http ://www.mydesk.byethost8.com /index.html](http://www.mydesk.byethost8.com/index.html)

## Table des matières

<b>1</b>	<b>Introduciton</b>	<b>4</b>
<b>2</b>	<b>Reprise du cahier des charges</b>	<b>5</b>
2.1	Le groupe . . . . .	5
2.2	Explication du concept . . . . .	5
2.3	Attribution des tâches . . . . .	6
<b>3</b>	<b>Présentation du projet</b>	<b>8</b>
3.1	Environnement . . . . .	8
3.1.1	La salle de classe . . . . .	8
3.1.2	Les décors . . . . .	8
3.2	Visualisation des écrans . . . . .	10
3.3	Souris et clavier virtuel . . . . .	11
3.3.1	Clavier . . . . .	11
3.3.2	Implémentation du pointeur . . . . .	11
3.3.3	Gestion des fenêtres . . . . .	15
3.3.4	Gestion de la souris . . . . .	22
3.4	Création du site web . . . . .	25
3.4.1	Forme du site web . . . . .	25
3.4.2	Hébergement du site web . . . . .	27
3.5	La barre de tâches . . . . .	28
3.5.1	Implémentation . . . . .	28
3.5.2	Personnalisation . . . . .	29
3.6	Le menu associé à la barre des taches . . . . .	34
3.6.1	Bouton « Quit » . . . . .	34
3.6.2	Réglage du volume . . . . .	34
3.6.3	Bouton changement de couleurs . . . . .	35
3.7	Déplacement des fenêtres . . . . .	36
3.8	Création d'un environnement personnalisable . . . . .	37
3.9	Réalisation d'un tutoriel . . . . .	38
3.9.1	Détection du premier lancement . . . . .	38
3.9.2	Conception du décor . . . . .	39
3.9.3	Cinématique d'introduction . . . . .	39
3.9.4	Le tutoriel . . . . .	42
3.10	Le lecteur audio . . . . .	45
3.10.1	Le lecteur et ses boutons . . . . .	45

3.10.2	La bibliothèque musicale . . . . .	50
<b>4</b>	<b>Récit de réalisation</b>	<b>54</b>
4.1	Quelques difficultés... . . . .	54
4.2	... mais aussi beaucoup de positif! . . . . .	54
<b>5</b>	<b>En conclusion</b>	<b>55</b>
5.1	Ressenti individuel . . . . .	55
5.1.1	Hector . . . . .	55
5.1.2	Réhan . . . . .	55
5.1.3	Lina . . . . .	56
5.1.4	Gianluca . . . . .	56
5.2	Conclusion générale . . . . .	58

# 1 Introduction

Vous êtes sur le point de lire notre rapport final de notre projet de SUP. Nous commencerons par faire un retour rapide sur le cahier de charges : notre sujet, nos objectifs, qui fait quoi, etc...

Nous vous présenterons ensuite notre projet de façon chronologique, afin de suivre son évolution au fil de ces mois de travail. Nous aborderons ensuite les quelques difficultés rencontrés, avant d'aborder les points plus positifs de ce travail de groupe.

Pour finir, chaque membre expliquera son ressenti, pour enfin clore ce rapport par une conclusion générale.

## 2 Reprise du cahier des charges

### 2.1 Le groupe

Ce projet est réalisé par une équipe de quatre élèves d'EPITA Strasbourg :

- Gianluca Minoprio
- Réhan Khan
- Lina Diringer
- Hector De Turckheim

### 2.2 Explication du concept

Dès janvier, nous avons décidé de recréer un bureau virtuel sous Unity. L'idée d'un projet utilisant la réalité virtuelle nous plaisait, ainsi la création d'un Virtual Desktop semblait être un bon challenge à relever.

L'objectif principal de notre projet est d'offrir à l'utilisateur un environnement plus agréable pour réaliser ses différentes tâches sur son ordinateur. Grâce à *My Desk*, il est possible d'élargir son espace de travail tout en restant fidèle à un ordinateur classique. Un clavier, une barre de tâches, une souris, la visualisation des différentes fenêtres ouvertes... Tous ces éléments se retrouvent dans la réalité virtuelle dans un espace 100% personnalisable.

En plus d'améliorer le confort de l'utilisateur, *My Desk* présente aussi un avantage financier non négligeable : inutile d'investir dans plusieurs écrans énormes qui prennent toute la place sur le bureau, puisqu'il est possible d'en avoir autant qu'on veut grâce à la réalité virtuelle.

## 2.3 Attribution des tâches

Nous rappelons ici nos différents objectifs fixés à chacun pour ce projet :

### Pour la première Soutenance

Parties	Groupe
Environnement : - Mesure d'une table de l'école qui servira de référence pour le modèle 3D du bureau virtuel - Mesure d'une salle du campus qui servira de référence pour modéliser l'environnement de travail - Modélisation de la salle qui sera l'environnement de travail	Lina, Hector
Visualisation des écrans : - Récupération du flux vidéo du bureau - Affichage et déplacement des différentes fenêtres dans l'environnement 3D créé	Gianluca, Rehan

### Pour la deuxième Soutenance

Parties	Groupe
Souris et clavier virtuel : - Affichage d'un curseur de souris sur les écrans virtuels à partir du pointage de la manette - Affichage d'un clavier virtuel utilisable pour écrire	Gianluca, Lina
Site internet - Création du site internet présentant le projet	Hector, Rehan

### Pour la dernière Soutenance

Parties	Groupe
Environnement : -Création d'environnements à thèmes (la mer, la forêt, etc...) basé sur les points de référence déjà mesurés -Création d'un tutoriel d'utilisation de l'application qui apparaîtra à la première utilisation	Gianluca, Hector
Implémentation de la barre de tâche : -Création d'une barre de tâche ergonomique en 3D -Récupération des informations de la barre de tâche du bureau Windows -Affichage des icônes correspondantes sur la barre de tâche en 3D	Rehan, Lina

## 3 Présentation du projet

Nous avons choisi de vous présenter notre projet de façon chronologique. Étant un travail de groupe, il sera plus intéressant de voir son évolution pas à pas.

### 3.1 Environnement

*Réalisé par Hector et Lina*

L'environnement constitue un élément important de notre bureau virtuel, et ce fut la première étape de la réalisation de notre projet.

#### 3.1.1 La salle de classe

Dans un premier temps, nous avons choisi de reconstituer à l'identique notre salle de classe. Le défi était de rendre cet environnement le plus fidèle possible à la réalité. Un travail de mesure précise a alors été réalisé. Nous avons pris en note chaque dimension des murs de la salle, ses fenêtres, jusqu'aux mesures de nos tables et de nos chaises.

Nous passons ensuite à l'étape de la modélisation sous Unity. Nous avons commencé par reproduire la salle et ses murs, en prenant soin de modéliser chaque fenêtre et chaque porte à l'identique. (Cf. annexes)

Une fois la structure de la salle réalisée, nous avons modélisé la table et la chaise de notre salle de classe. Là encore, il a fallu faire preuve de précision. Ces deux objets, et en particulier la table, nécessitait un travail sur les formes plus complexes comme les courbures des pieds ou du dossier. Pour ce faire, nous avons utilisé l'outil ProBuilder pour aboutir à un résultat que vous trouverez en annexes.

#### 3.1.2 Les décors

Sachant que la salle modélisé avait des fenêtres, un néant était maintenant visible depuis celle-ci. Nous avons donc choisi de rendre la vue plus agréable en créant deux types de décors : une forêt et un désert.

Ceux-ci furent réalisés à l'aide d'assets proposant des éléments de décors en lowpoly, dont le graphisme s'adapte bien au minimalisme de la réalité virtuelle. Une alternance entre ces décors a pu être créée en appuyant sur un



bouton de la manette.

En plus des décors, nous avons pu implémenté un cycle jour / nuit, pour ajouter du réalisme à la scène.

Vous trouverez le design de ces décors dans les pages annexes.

## 3.2 Visualisation des écrans

*Réalisé par Gianluca et Réhan*

Pour visualiser les écrans nous avons utilisé un asset réalisé par un développeur du pseudonyme *Hecomi*. Son asset utilise une API qui permet de récupérer le flux vidéo de chaque fenêtre du bureau de l'ordinateur.

Dans notre expérience, le script récupère la position x et y des fenêtres dans le bureau et reproduit depuis un point de référence chaque fenêtre dans notre environnement virtuel par rapport à leurs positions x et y. Cela nous permet donc d'avoir chaque fenêtre en plein écran et de naviguer comme si l'on était sur un ordinateur classique. Les fenêtres sont finalement des textures qui s'affichent et s'actualisent sur des objets 3D. A chaque image, les fenêtres de l'environnement s'actualisent et s'adaptent par rapport aux interactions faites sur le bureau. Donc quand la fenêtre d'une application s'agrandit, l'objet 3D va se former par rapport à la même échelle que la texture. Voici ci-dessous un schéma qui montre la connection entre le .dll et le script Unity.

Vous trouverez un schéma explicatif de l'API dans les pages annexes.

### 3.3 Souris et clavier virtuel

#### 3.3.1 Clavier

*Réalisé par Lina*

Nous avons ensuite implémenté un clavier pour notre Virtual Desktop. Celui-ci est formé d'un Canvas regroupant 2 GameObjects se superposant : **minuscule** et **majuscule**. Ces deux objets forment un ensemble de bouton portant chacun une lettre, chiffre ou symbole.

Chaque bouton applique la fonction `alphabetFunction()` qui permet d'ajouter le caractère correspondant à la chaîne de caractère courante.

```
string word = null;
string alpha;
public Text myName = null; ◀ ...
public GameObject minuscule, capitale; ◀ capitale

◀ Event handler ◀ 90 asset usages
public void alphabetFunction (string alphabet)
{
    word = word + alphabet;
    myName.text = word;
}
```

*Code de `alphabetfunction`*

Enfin, nous avons créé un bouton permettant d'alterner entre les deux claviers **minuscule** et **majuscule**. Celui-ci masque l'un et affiche l'autre, selon le clavier courant.

#### 3.3.2 Implémentation du pointeur

*Réalisé par Gianluca*

Dans la création de notre pointeur, ce sont en fait 2 pointeurs qui sont calculés dans le processus. Il y a 2 types de pointeurs, chacun réagissant à de types de GameObjects différents :

- les Canvas / UI (User Interface) : utilisés pour les événements du clavier qui réagissent aux clics.
- les objets physiques : des objets dotés d'un collider.

Vous trouvez ci-dessous l'implémentations de ces deux pointeurs.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.EventSystems;
5  using UnityEngine.InputSystem;
6  using UnityEngine.UI;
7
8  public class pointer : MonoBehaviour
9  {
10
11     public float DefaultLength = 3.0f;
12     public EventSystem eventSystem = null;
13     public StandaloneInputModule inputModule = null;
14     private LineRenderer lineRenderer = null;
15
16     references
17     private void Awake()
18     {
19         lineRenderer = GetComponent<LineRenderer>();
20     }
21
22     references
23     private void Update()
24     {
25         UpdateLength();
26     }
27
28     references
29     private void UpdateLength()
30     {
31         RaycastHit hit = CreateForwardRaycast();
32         if (hit.distance < GetCanvasDistance())
33         {
34             lineRenderer.SetPosition(0, transform.position); //actualise la longueur du pointeur
35             lineRenderer.SetPosition(1, CalculateEnd());
36         }
37         else
38         {
39             lineRenderer.SetPosition(0, transform.position); //actualise la longueur du pointeur
40             lineRenderer.SetPosition(1, GetEnd());
41         }
42     }
43
44     // Méthodes pour la création du pointeur canvas
45     references
46     private Vector3 GetEnd()
47     {
48         float distance = GetCanvasDistance();
49         Vector3 endPosition = DefaultEnd(DefaultLength);
50         if (distance != 0.0f)
51             endPosition = DefaultEnd(distance);
52         return endPosition;
53     }
54
55     references
56     private RaycastHit FindFirstRaycast(List<RaycastHit> results)
57     {
58         foreach (RaycastHit result in results)
59         {
60             if (result.collider.CompareTag("Pointer"))
61                 return result;
62         }
63         return null;
64     }
65
66     private float GetCanvasDistance()
67     {
68         if (eventSystem != null)
69             return eventSystem.hierarchyRoot.GetComponent<Canvas>().worldCullingDistance;
70         return 0.0f;
71     }
72
73     private Vector3 DefaultEnd(float length)
74     {
75         return transform.position + transform.forward * length;
76     }
77
78     private RaycastHit CreateForwardRaycast()
79     {
80         Ray ray = new Ray(transform.position, transform.forward);
81         RaycastHit hit;
82         if (Physics.Raycast(ray, out hit))
83             return hit;
84         return null;
85     }
86 }

```

```

pointer.cs 9 X UwcWindowTexture.cs UwcWindow.cs UwcAltTabWindowTextureManager.cs UwcDesktopLayouter.cs vinputs.cs MouseOperation.cs UwcWindowTexture
Assembly-CSharp
private RaycastHit[] FindRaycast(Collider[] colliders, Ray ray)
{
    foreach (RaycastHit result in results)
    {
        if (!result.gameObject)
            continue;
        return result;
    }
    return new RaycastHit();
}

// Méthode pour la création du raycast physiques (pointeur pour objet physique)
// Méthode public pour utiliser dans le script UwcDesktopLayouter
public Vector3 CalculateEnd()
{
    RaycastHit hit = CreateForwardRaycast();
    Vector3 endPosition = DefaultEnd(DefaultLength); //longueur max du pointeur
    if (hit.collider)
        endPosition = hit.point; // modifie la longueur du pointeur si il rencontre un objet
    return endPosition;
}

// Méthode public pour utiliser dans le script UwcDesktopLayouter
public RaycastHit CreateForwardRaycast()
{
    RaycastHit hit;
    Ray ray = new Ray(transform.TransformPoint(0, 0, 0.1f), transform.forward); //création du rayon au niveau de la main qui va se projeter devant lui
    // mettre le début du rayon à z: 0.1 pour éviter que le rayon rencontre la main
    Physics.Raycast(ray, out hit, DefaultLength); //return vrai si le ray rencontre un collider sinon c faux dans ce cas on l'utilise pour générer le ray que l'on a créer
    return hit;
}

// Méthode public pour utiliser dans le script UwcDesktopLayouter
private Vector3 DefaultEnd(float length)
{
    return transform.position + (transform.forward * length);
}

```

## Son fonctionnement :

Nous avons alors eu recours au type Raycast. Le Raycasting est une technique qui correspond à envoyer un rayon invisible depuis une caméra. Elle permet donc de connaître la distance entre deux GameObjects. Pour générer un Raycast, il a fallu implémenter une caméra sur le point initial du pointeur, qui correspond à la main de l'utilisateur.

Nous avons donc créer un GameObject appelé **pointeur**. Il a pour composant :

- un script **PhysicsRaycaster** : celui-ci calcule la distance entre la main et un object composé d'un collider
- une caméra : on lui rattache le **PhysicsRaycaster**
- un Line Renderer : il s'actualise à chaque Update lorsqu'elle rencontre un objet

Sachant que l'on génère deux pointeurs, on doit en afficher uniquement un pour avoir un rendu utilisable. Alors, à chaque Frame on calcule la distance entre un canvas et un objet physique. On choisit la distance la plus courte des deux et on génère le trait le plus petit. Cela est fait par `UpdateLenght()` qui s'actualise à chaque Update.

Deux technoques sont utilisées pour calculer la distance entre la main et l'objet :

- `CreateForwardRaycast()` pour les objets physiques :  
Cette méthode crée un Raycast qui se projette et retourne le rayon qui a rencontré un objet. Dans notre cas, lorsque l'on a créé le rayon, on l'a initialisé à une position de 0.1f plus en avant pour éviter que le rayon rencontre le collider de la main. Au début, en créant le pointer, le rayon s'initialisait à la paume de la main. Le rayon rencontrait les doigts et donc le pointeur se générerait uniquement entre la paume et les doigts. Cette méthode est ensuite utilisée pour `CalculateEnd()`, qui retourne un `Vecteur3`, qui est le point de rencontre entre le pointeur et l'objet.
- Utilisation de `EventSystem` de Unity et plus particulièrement `RaycastEvent` :  
Ils vont permettre de calculer la distance du rayon entre la main et un Canvas. La fonction `GetCanvasDistance()` est la méthode qui calcule la distance. Elle utilise `FindFirstRaycast()` qui permet de récupérer le Raycast avec le résultat le plus exact. `GetCanvasDistance()` est ensuite utilisée dans `GetEnd()` et `UpdateLength()`. `GetEnd()` est la méthode qui va retourner le point de contact avec le pointeur et le canvas.

Nous avons ensuite rajouté une variable publique de type float qui est la taille maximale du pointeur. Elle permet un rendu plus agréable pour l'utilisateur. Cette variable est utilisée dans `DefaultEnd()`, `CreateForwardRaycast()`, `CalculateEnd()`, `GetEnd()` et `GetCanvasDistance()`. Elle permet au rayon de s'arrêter obligatoirement à la distance maximale. Cela optimise le calcul du rayon, donne un rendu plus visuel au pointeur et évite que celui-ci traverse toute la salle. Puisque cette variable est facilement réglable depuis l'inspecteur Unity, on pourrait imaginer un menu futur où l'on pourrait choisir cette distance maximale du pointeur.

Ce pointeur une fois réalisé, nous avons pu nous pencher sur la gestion des fenêtres, l'interaction avec le clavier et la souris.

### 3.3.3 Gestion des fenêtres

On rappelle que l'on avait le choix entre deux types d'assets d'enregistrement de nos écrans fait par Hecomi :

- Desktop Capture : enregistrement du bureau
- Windows Capture : enregistrement des fenêtres

Ces deux assets font la jonction entre une API fait par Windows qui permet de récupérer et modifier beaucoup de paramètres du bureau : la position de la souris, celle d'une fenêtre ou encore l'image d'une fenêtre.

Voici un lien de L'API qui explique en profondeur les possibilités : <https://docs.microsoft.com/en-us/dotnet/api/system.runtime.interopservices?view=netframework-4.8>

Pour notre projet, l'assets Windows Capture est le plus adéquat. En effet, cela nous permet en réalité virtuelle de voir plusieurs fenêtres en même temps, de pouvoir les disposer comme on le veut dans l'espace 3D et d'adapter leurs tailles à notre guise.

A l'opposé, un enregistrement avec Desktop Capture ne changerait pas l'expérience du bureau puisque l'enregistrement du bureau est identique à celui d'un écran lambda. Le seul avantage qu'avait Desktop Capture est son optimisation, car elle demande moins de ressources de calcul puisque c'est une seule texture qu'il faut afficher, alors que Windows Capture affiche et actualise plusieurs textures en même temps.

Pour la gestion des fenêtres, on s'est basé sur un prefab que proposait Hecomi avec son asset. Le but de ce prefab est de disposer les fenêtres par rapport à leurs positions x, y dans le bureau et leur position z par rapport à AltTab (la profondeur de la fenêtre dans le bureau). Quand on reste appuyé sur AltTab, une liste de fenêtre s'affiche. L'API enregistre un index pour chaque fenêtre, cet index dépendant de cette liste AltTab. Donc la position des fenêtres dans Unity s'agence dans la même disposition que les fenêtres du bureau.

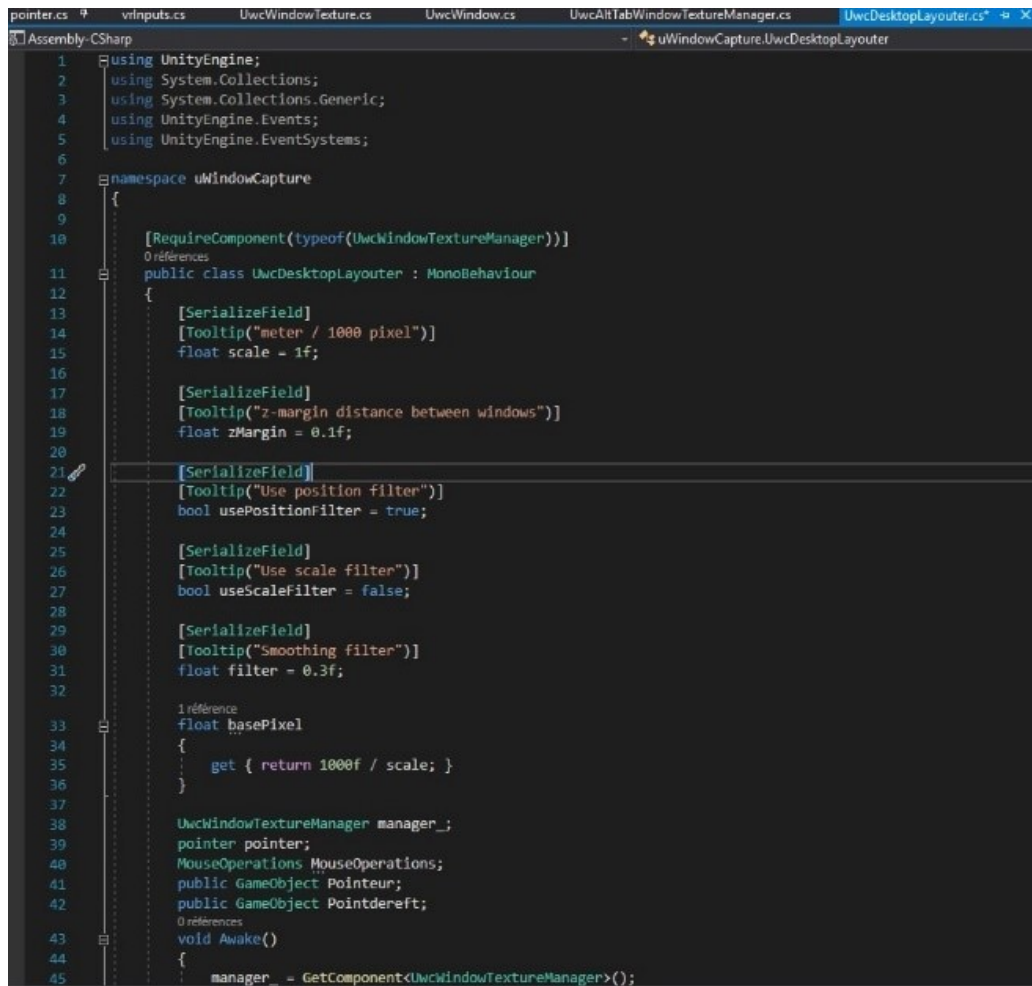
Cette classe **UwcAltTabWindowTextureManager** est utile uniquement à l'initialisation des fenêtres. Dans la classe **OnWindowAdded(UwcWinow window)**, nous avons ajouter une condition. Le créateur de l'asset a créé une classe **UwcWindow**, qui a pour variables un *Titre*. Ce *Titre* est récupéré directement du nom de l'exécutable. Nous avons donc utilisé **UwcManager** qui a une méthode **Find(string title)** qui renvoie le UwcWindow qui a le titre ou une partie du titre identique. Nous avons donc enlevé l'affichage de la fenêtre de notre application, car celle-ci créait un miroir infini : elle montrait ce que l'utilisateur voyait mais lui-même voyait la fenêtre qui elle montrait encore la vue en VR...

Avec cette condition, on a donc enlevé un bug qui était très gênant pour l'utilisateur en VR. Dans **OnWindowAdded(UwcWinow window)**, la méthode n'affiche pas la fenêtre enfant. Les fenêtres enfants sont les fenêtres qui se créent quand on fait un clic gauche, ou ce sont encore les pop-ups qui apparaissent sur certaines applications. La méthode n'affiche pas non plus les fenêtres non visibles, c'est-à-dire les fenêtres en fond de tâche et il n'affiche pas les fenêtres qui ne sont pas dans la liste AltTab. Toutes ces conditions permettent un affichage bien trié des fenêtres et évite une abondance de fenêtres dans tous les sens, ce qui serait perturbant pour l'utilisateur.

Il nous fallait ensuite un script qui permet l'actualisation de chaque fenêtre : leur position, leur texture, leur taille ou de son interaction avec le pointeur.

Voici le code qui permet d'actualiser les fenêtres à chaque frame :



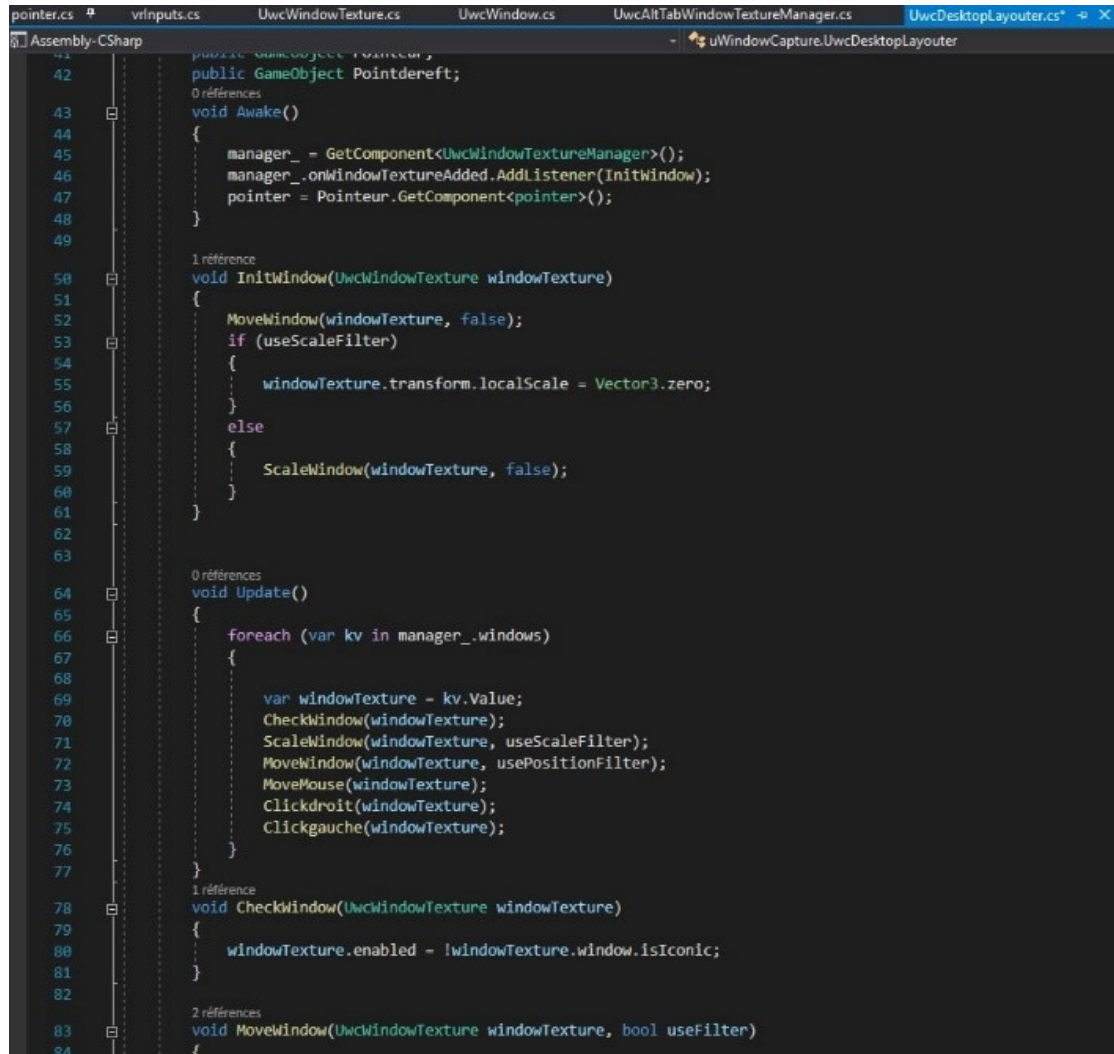


```

1  using UnityEngine;
2  using System.Collections;
3  using System.Collections.Generic;
4  using UnityEngine.Events;
5  using UnityEngine.EventSystems;
6
7  namespace uWindowCapture
8  {
9
10     [RequireComponent(typeof(UwcWindowTextureManager))]
11     public class UwcDesktopLayouter : MonoBehaviour
12     {
13         [SerializeField]
14         [Tooltip("meter / 1000 pixel")]
15         float scale = 1f;
16
17         [SerializeField]
18         [Tooltip("z-margin distance between windows")]
19         float zMargin = 0.1f;
20
21         [SerializeField]
22         [Tooltip("Use position filter")]
23         bool usePositionFilter = true;
24
25         [SerializeField]
26         [Tooltip("Use scale filter")]
27         bool useScaleFilter = false;
28
29         [SerializeField]
30         [Tooltip("Smoothing filter")]
31         float filter = 0.3f;
32
33         1 référence
34         float basePixel
35         {
36             get { return 1000f / scale; }
37         }
38
39         UwcWindowTextureManager manager_;
40         pointer pointer;
41         MouseOperations mouseOperations;
42         public GameObject Pointeur;
43         public GameObject Pointdereft;
44
45         0 références
46         void Awake()
47         {
48             manager_ = GetComponent<UwcWindowTextureManager>();
49         }
50     }

```

FIGURE 1



```

41: public GameObject Pointderef;
42: public GameObject Pointderef;
43: void Awake()
44: {
45:     manager_ = GetComponent<UwcWindowTextureManager>();
46:     manager_.onWindowTextureAdded.AddListener(InitWindow);
47:     pointer = Pointeur.GetComponent<pointer>();
48: }
49:
50: 1 référence
51: void InitWindow(UwcWindowTexture windowTexture)
52: {
53:     MoveWindow(windowTexture, false);
54:     if (useScaleFilter)
55:     {
56:         windowTexture.transform.localScale = Vector3.zero;
57:     }
58:     else
59:     {
60:         ScaleWindow(windowTexture, false);
61:     }
62: }
63:
64: 0 références
65: void Update()
66: {
67:     foreach (var kv in manager_.windows)
68:     {
69:         var windowTexture = kv.Value;
70:         CheckWindow(windowTexture);
71:         ScaleWindow(windowTexture, useScaleFilter);
72:         MoveWindow(windowTexture, usePositionFilter);
73:         MoveMouse(windowTexture);
74:         Clickdroit(windowTexture);
75:         Clickgauche(windowTexture);
76:     }
77: }
78: 1 référence
79: void CheckWindow(UwcWindowTexture windowTexture)
80: {
81:     windowTexture.enabled = !windowTexture.window.isIconic;
82: }
83: 2 références
84: void MoveWindow(UwcWindowTexture windowTexture, bool useFilter)

```

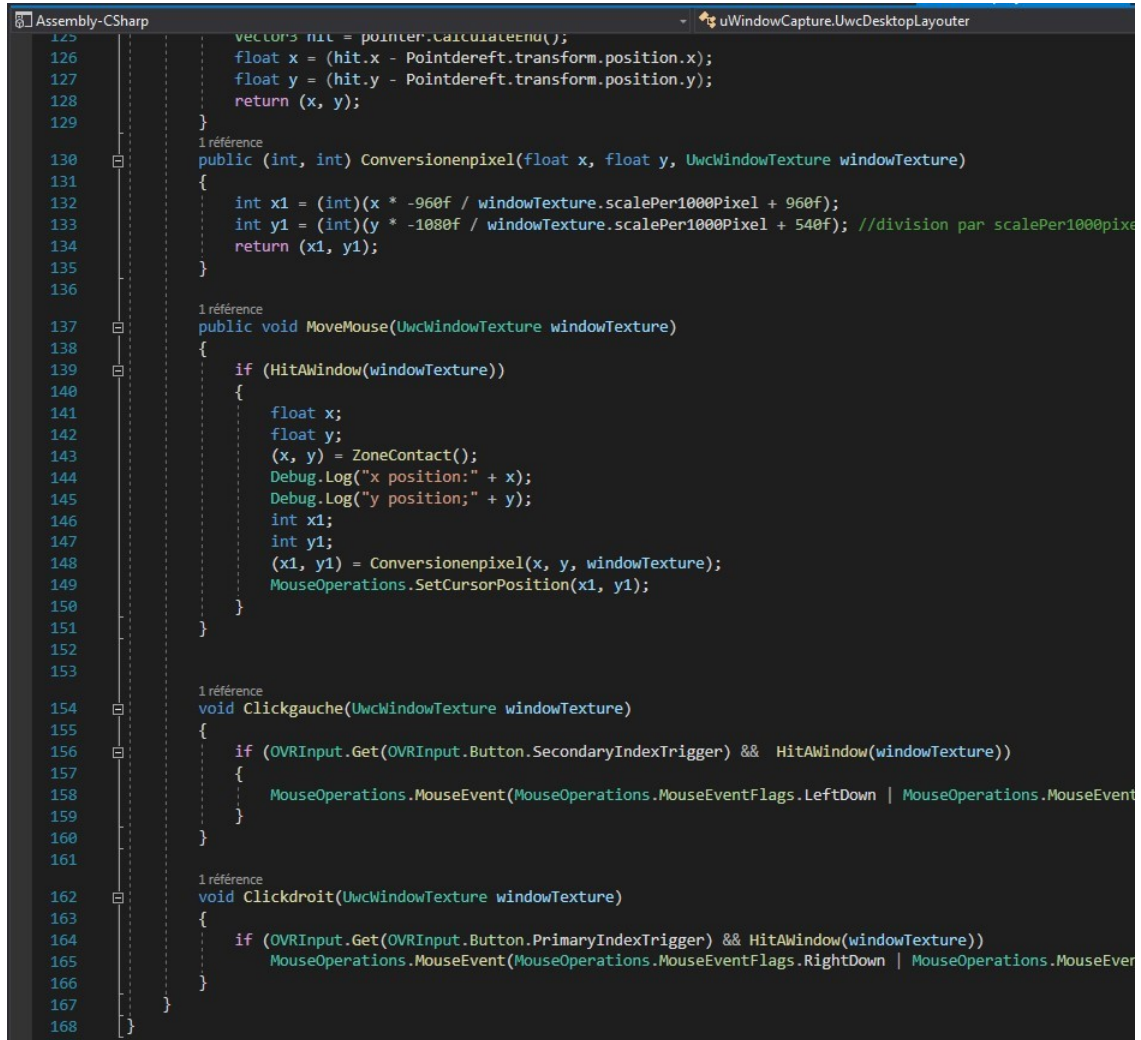
FIGURE 2

```

pointer.cs # vinputs.cs UwcWindowTexture.cs UwcWindow.cs UwcAltTabWindowTextureManager.cs UwcDesktopLayout.cs
Assembly-CSharp
83 void MoveWindow(UwcWindowTexture windowTexture, bool useFilter)
84 {
85     var window = windowTexture.window;
86     var a = windowTexture.transform.position;
87     var pos = UwcWindowUtil.ConvertDesktopCoordToUnityPosition(window, basePixel);
88     pos.z = window.zOrder * zMargin;
89     var targetPos = transform.localToWorldMatrix.MultiplyPoint3x4(pos);
90     windowTexture.transform.position = (useFilter ?
91         Vector3.Slerp(windowTexture.transform.position, targetPos, filter) :
92         targetPos);
93 }
94
95 5 références
96 public bool HitWindow(UwcWindowTexture window)
97 {
98     RaycastHit hit = pointer.CreateForwardRaycast();
99     if (hit.collider == window.collider_)
100     {
101         return true;
102     }
103     return false;
104 }
105
106 2 références
107 public void ScaleWindow(UwcWindowTexture windowTexture, bool useFilter)
108 {
109     //augmente la taille de la fenetre de 2% si le joystick est enfoncé vers l'avant
110     if (OVRInput.Get(OVRInput.Button.PrimaryThumbstickUp) && HitWindow(windowTexture))
111     {
112         windowTexture.scalePer1000Pixel *= 1.02f;
113     }
114     //diminue la taille de la fenetre de 2% si le joystick est enfoncé vers l'arrière
115     if (OVRInput.Get(OVRInput.Button.PrimaryThumbstickDown) && HitWindow(windowTexture))
116     {
117         windowTexture.scalePer1000Pixel *= 0.98f;
118     }
119     windowTexture.scaleControllType = WindowTextureScaleControllType.BaseScale;
120 }
121
122 1 référence
123 public (float, float) ZoneContact()
124 {
125     Vector3 hit = pointer.CalculateEnd();
126     float x = (hit.x - Pointdereft.transform.position.x);
127     float y = (hit.y - Pointdereft.transform.position.y);

```

FIGURE 3



```

125         vectors.nit = pointer.calculateEnd();
126         float x = (hit.x - Pointdereft.transform.position.x);
127         float y = (hit.y - Pointdereft.transform.position.y);
128         return (x, y);
129     }
130     1 référence
131     public (int, int) Conversionenpixel(float x, float y, UwcWindowTexture windowTexture)
132     {
133         int x1 = (int)(x * -960f / windowTexture.scalePer1000Pixel + 960f);
134         int y1 = (int)(y * -1080f / windowTexture.scalePer1000Pixel + 540f); //division par scalePer1000p
135         return (x1, y1);
136     }
137     1 référence
138     public void MoveMouse(UwcWindowTexture windowTexture)
139     {
140         if (HitAWindow(windowTexture))
141         {
142             float x;
143             float y;
144             (x, y) = ZoneContact();
145             Debug.Log("x position:" + x);
146             Debug.Log("y position:" + y);
147             int x1;
148             int y1;
149             (x1, y1) = Conversionenpixel(x, y, windowTexture);
150             MouseOperations.SetCursorPosition(x1, y1);
151         }
152     }
153     1 référence
154     void Clickgauche(UwcWindowTexture windowTexture)
155     {
156         if (OVRInput.Get(OVRInput.Button.SecondaryIndexTrigger) && HitAWindow(windowTexture))
157         {
158             MouseOperations.MouseEvent(MouseOperations.MouseEventFlags.LeftDown | MouseOperations.MouseEvent
159         }
160     }
161     1 référence
162     void Clickdroit(UwcWindowTexture windowTexture)
163     {
164         if (OVRInput.Get(OVRInput.Button.PrimaryIndexTrigger) && HitAWindow(windowTexture))
165             MouseOperations.MouseEvent(MouseOperations.MouseEventFlags.RightDown | MouseOperations.MouseEvent
166     }
167 }
168

```

FIGURE 4

## Code la classe **UwcDesktopLayouter**

Voici les champs que nous avons ajouté :

- **Pointer pointer** qui vient du script du pointeur sera utile pour des futures fonctions pour vérifier le contact entre pointeur et fenêtres.

- **MouseOperations MouseOperation** : ce champ vient de la classe **MouseOperations** qui utilise aussi l'API *System.Runtime.InteropServices*, mais dans ce cas pour tout ce qui est simulation de la souris. Nous reviendront en détails sur cette classe quand nous expliquerons comment nous avons simulé la souris avec le mouvement du pointeur.
- **GameObject pointeur** est un **GameObject** modifiable depuis l'inspecteur Unity. Il correspond au pointeur et sera aussi utile pour la partie simulation de la souris.
- **GameObject Poindereft** est un **GameObject** modifiable depuis l'inspecteur Unity. Il correspond au **GameObject Windows** qui fait spawner les fenêtres du bureau dans la même disposition x,y depuis un point de référence. Ce champ sera aussi très utile pour la simulation de la souris.

La méthode `InitWindow(UwcWindowtexture windowtexture)` est uniquement utilisée pour initialiser les fenêtres dans la méthode `Awake()`. Dans cette méthode, la liste de fenêtre **manager\_** est initialisée, elle permettra de faire le parcours de cette liste **manager\_** et d'appliquer pour chaque fenêtre leur classe **UwcWindowTexture** sur chaque méthode et à chaque frame.

La méthode `MoveWindow` permet de bouger la texture de la fenêtre par rapport à sa position dans le bureau.

La méthode `HitAWindow` n'est pas d'origine dans la classe. Cette méthode permet de vérifier si la fenêtre est en contact avec le pointeur. Pour faire fonctionner cette méthode, il a fallu changer l'accès du collider de **UwcWindowTexture** (il était privé d'origine).

La méthode `ScaleWindow` a été radicalement changée. Maintenant, la fenêtre augmente de taille si la fenêtre est sélectionnée par le pointeur et si le joystick du controller est enfoncé vers le haut. De même, la fenêtre diminue de taille si le joystick est appuyé vers le bas. La fenêtre peut aussi changer de forme si elle change de forme dans le bureau. Dans ce cas, elle change de forme tout en gardant la grandeur en plus que on lui a donnée ou retirée.

Ces trois méthodes sont celles qui permettent la gestion des fenêtres. Main-

tenant, nous allons expliquer la simulation de la souris par le pointeur qui a été entièrement réalisée par nos soins.

### 3.3.4 Gestion de la souris

Sur Windows, la souris se déplace sur un axe  $x, y$ . Le point d'origine 0, 0 est en haut à gauche et les valeurs  $x, y$  représentent un pixel, ce sont donc des entiers. Le point le plus éloigné de l'origine est en bas à droite avec  $(x, y) = (1920, 1080)$

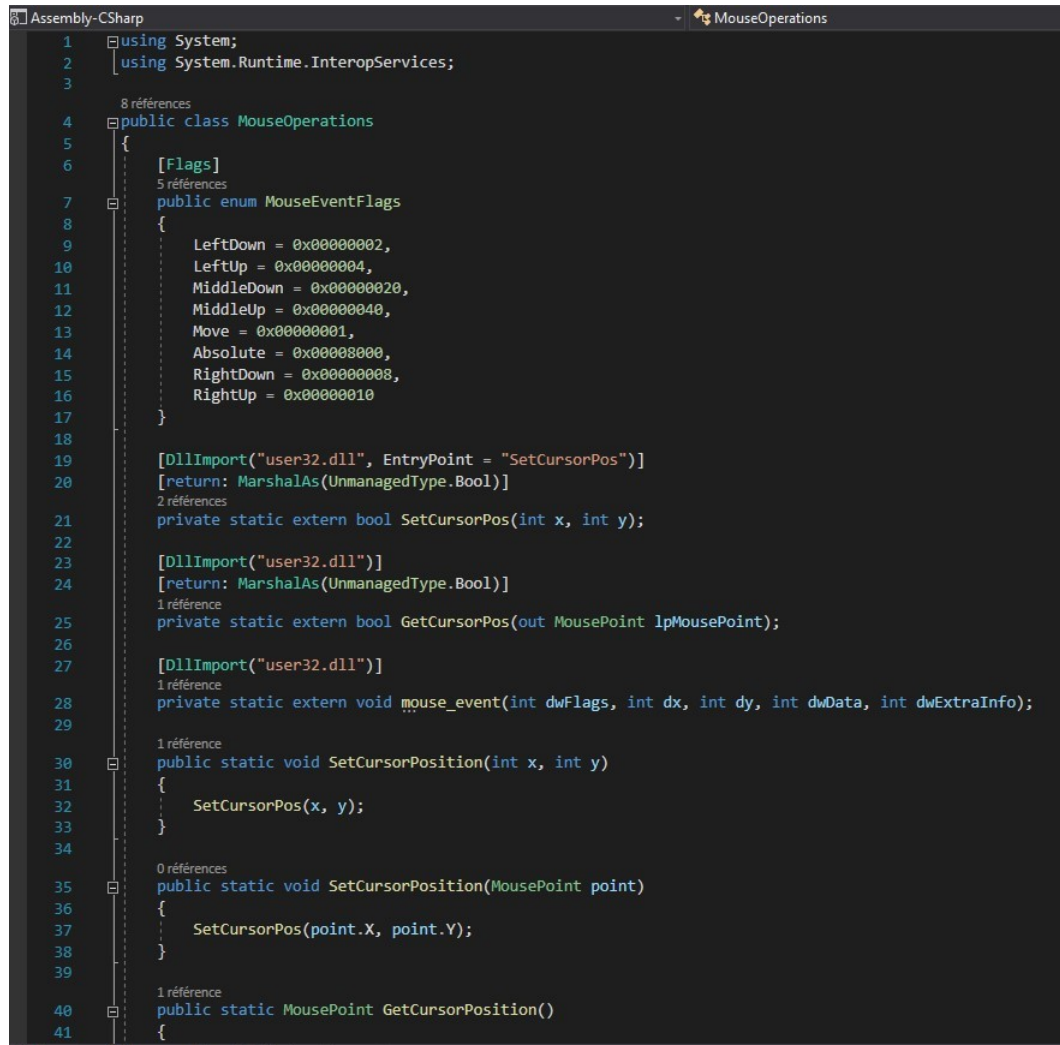
Pour le **gameObject Window** qui affiche toutes les fenêtres comme un bureau, son point de référence est au centre du gameObject. Il faut donc traduire la position  $x, y$  du pointeur par rapport au centre du gameObject **Window** et convertir la distance Unity en pixel.

La méthode **ZoneContact()** retourne un tuple de float qui correspond à la nouvelle coordonnée du pointeur par rapport à celle du point de référence.

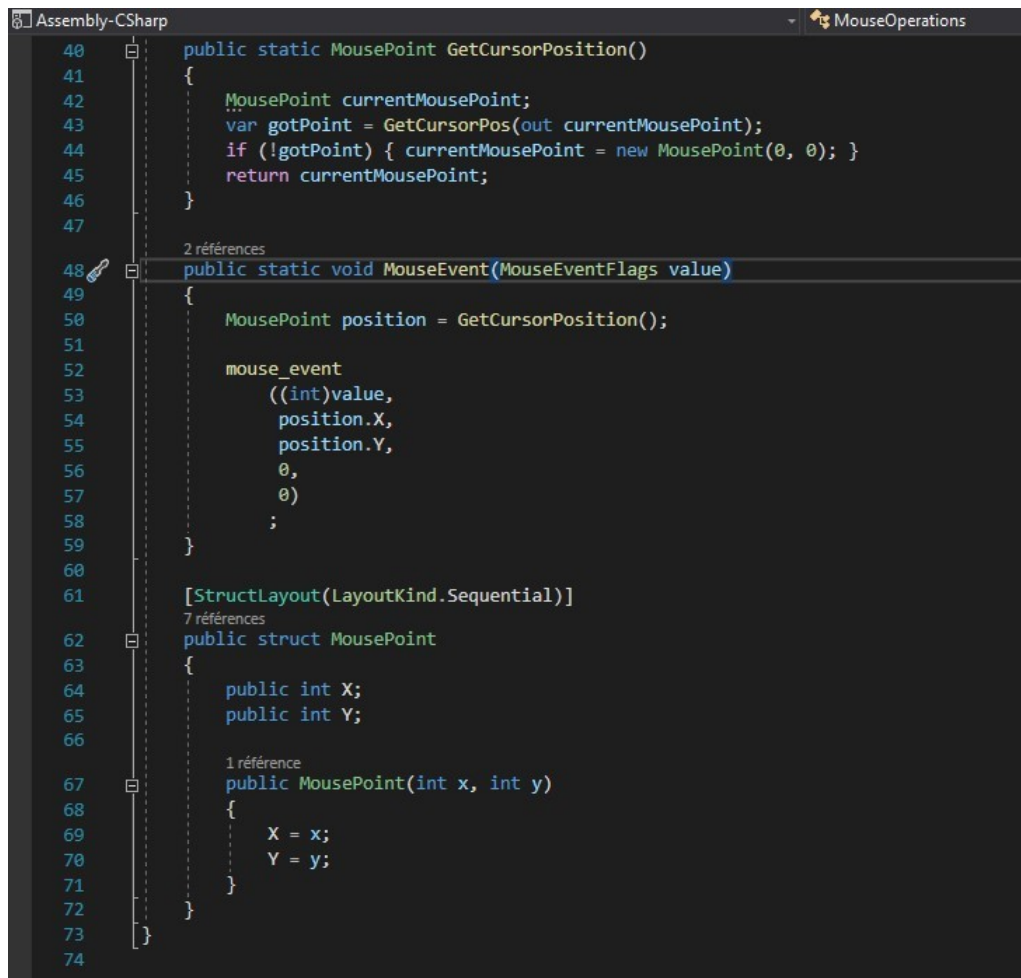
La méthode **Conversionenpixel()** prend en argument deux flottants et un **UwcWindowTexture**. Pour créer cette méthode, ne trouvant pas de conversion mesure Unity en pixel, on a récupéré 3 valeurs  $x$  et  $y$  grâce à **ZoneContact()**. On a récupéré les valeurs pour les coordonnées aux coins haut gauche, centre et le coin bas droit du gameObject **Window**. On en a déduit une fonction affine pour la conversion de la position  $x$  en pixel et la position  $y$  en pixel. Dans la méthode, on a rajouté l'argument **UwcWindowTexture** car dans la fonction elle divise  $x$  et  $y$ . Cette division est nécessaire pour compenser le changement de taille lorsque les fenêtres s'agrandissent.

Pour finir, on applique la méthode **MoveMouse()** qui utilise les deux méthodes précédentes pour l'appliquer sur **SetCursorPosition(x, y)**, une méthode venant du script **MouseOperations** qui fait la liaison entre le .dll et le C#.

Ci-dessous le script de la classe **MouseOperations** :



```
1  using System;
2  using System.Runtime.InteropServices;
3
4  public class MouseOperations
5  {
6      [Flags]
7      public enum MouseEventFlags
8      {
9          LeftDown = 0x00000002,
10         LeftUp = 0x00000004,
11         MiddleDown = 0x00000020,
12         MiddleUp = 0x00000040,
13         Move = 0x00000001,
14         Absolute = 0x00000000,
15         RightDown = 0x00000008,
16         RightUp = 0x00000010
17     }
18
19     [DllImport("user32.dll", EntryPoint = "SetCursorPos")]
20     [return: MarshalAs(UnmanagedType.Bool)]
21     private static extern bool SetCursorPos(int x, int y);
22
23     [DllImport("user32.dll")]
24     [return: MarshalAs(UnmanagedType.Bool)]
25     private static extern bool GetCursorPos(out MousePoint lpMousePoint);
26
27     [DllImport("user32.dll")]
28     private static extern void mouse_event(int dwFlags, int dx, int dy, int dwData, int dwExtraInfo);
29
30     public static void SetCursorPosition(int x, int y)
31     {
32         SetCursorPos(x, y);
33     }
34
35     public static void SetCursorPosition(MousePoint point)
36     {
37         SetCursorPos(point.X, point.Y);
38     }
39
40     public static MousePoint GetCursorPosition()
41     {
42     }
```



```
Assembly-CSharp MouseOperations
40 public static MousePoint GetCursorPosition()
41 {
42     MousePoint currentMousePoint;
43     var gotPoint = GetCursorPos(out currentMousePoint);
44     if (!gotPoint) { currentMousePoint = new MousePoint(0, 0); }
45     return currentMousePoint;
46 }
47
48 2 références
49 public static void MouseEvent(MouseEventFlags value)
50 {
51     MousePoint position = GetCursorPosition();
52
53     mouse_event
54     ((int)value,
55     position.X,
56     position.Y,
57     0,
58     0)
59     ;
60 }
61 [StructLayout(LayoutKind.Sequential)]
62 7 références
63 public struct MousePoint
64 {
65     public int X;
66     public int Y;
67
68     1 référence
69     public MousePoint(int x, int y)
70     {
71         X = x;
72         Y = y;
73     }
74 }
```

Finalement les méthodes `clickdroit` et `clickgauche` permettent de faire des clics de souris avec les boutons du controller oculus.



## 3.4 Création du site web

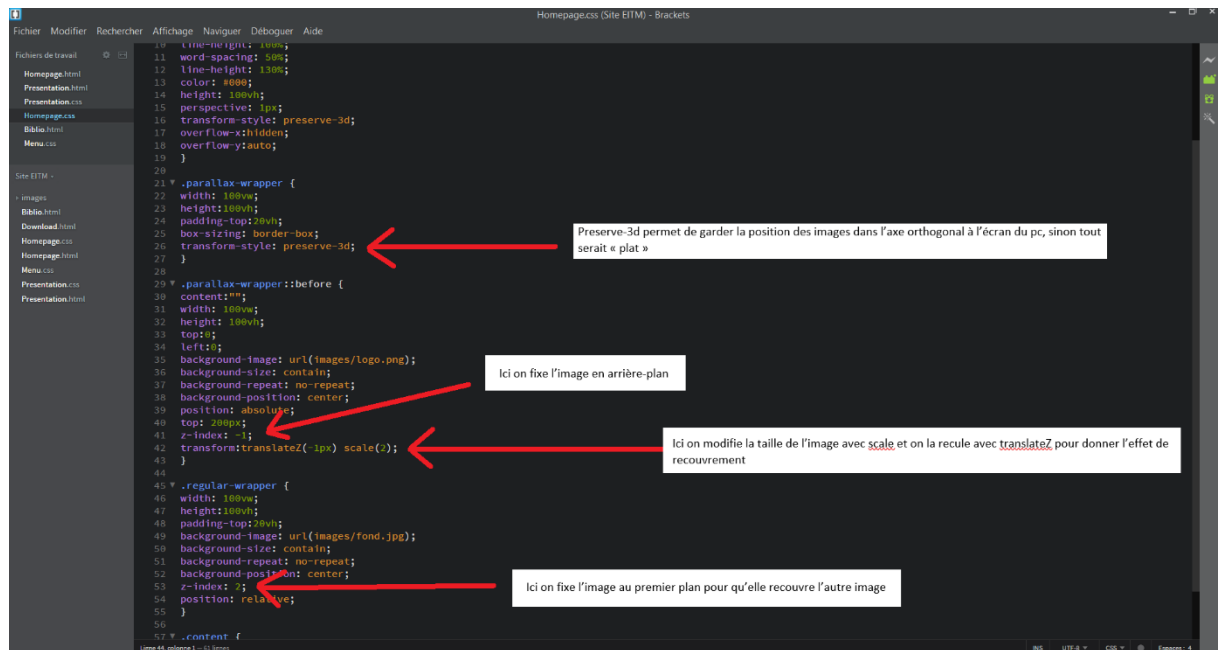
*Réalisé par Hector et Réhan*

### 3.4.1 Forme du site web

Pour le site web, il a tout d'abord fallu découvrir les différents langages web communs (html, css, php). En apprenant ces langages (dont on a réalisé la différence de syntaxe avec le C#, OCaml ou python qu'on a connu), on a voulu expérimenter l'effet de « parallaxe » sur la page d'accueil, qui a été fait en css.

#### L'effet de parallaxe

L'effet de parallaxe consiste à donner l'impression qu'il y a différentes couches qui composent le site web et que chaque couche se déplace à une vitesse différente. On voit sur les captures d'écrans jointes dans les annexes que l'image du bas recouvre l'image du haut quand on descend dans la page. Bien que cet effet soit répandu sur les sites web, l'appliquer à notre site a été bien plus difficile que ce à quoi on s'attendait.



Code de l'effet de parallaxe

Vous pouvez retrouver quelques captures d'écrans du site dans les annexes.

## Le menu réactif

On a ensuite créé un menu réactif, qui s'adapte à la taille de la fenêtre, et change de forme si la fenêtre devient trop petite. Vous trouverez en annexes les deux dispositions.

Cette disposition dépend donc de la taille de la fenêtre. Si celle-ci passe en dessous de 1080 px (pixels css), les liens vers les autres pages disparaissent et le titre « MyDesk » est centré. Si on passe la souris sur le logo, un menu déroulant apparaît avec les liens vers les autres pages du site, ce qui est plus optimisé pour une petite fenêtre. Pour ce menu, on a également dû en apprendre plus sur le css.

### 3.4.2 Hébergement du site web

Une fois le site web créé, il a fallu l'envoyer sur le web. Ce dernier n'étant pas destiné à être vu par des centaines de milliers de personnes par jour pendant plusieurs années, nous avons donc choisi un hébergeur gratuit : *byethost*. Nous avons ensuite utilisé un client FTP bien connu nommé FileZila pour envoyer nos fichiers .html et .css, ainsi que nos fonds et notre logo sur internet.

La principale difficulté que nous avons dû surmonter a été de mettre en ligne le fichier .zip contenant notre projet. A la base, nous pensions mettre le fichier à disposition via un lien de téléchargement directe. Cependant, ce dernier ayant une taille de 500 Mo environ, il a été impossible de l'uploader sur notre site web via notre client FTP (surtout avec une connexion des années 90).

Nous avons ensuite envisagé d'utiliser un répertoire GitHub pour héberger le fichier et simplement mettre un lien vers celui-ci sur notre site web. Malheureusement, ce fut un nouvel échec encore une fois à cause de la taille trop importante du fichier. Nous avons donc finalement choisi d'utiliser Dropbox pour mettre en ligne notre projet et avons mis un lien sur notre site web pour pouvoir y accéder.

## 3.5 La barre de tâches

*Réalisé par Lina*

Afin de rendre l’affichage des écrans plus facile, nous avons décidé d’intégrer une barre de tâches à notre virtual desktop. Celle-ci s’inspire des barres traditionnelles, où l’on retrouve les icônes des fenêtres ouvertes, et qui permettent de les rendre visible ou non. Une fois encore, nous avons eu recours à l’asset utilisé pour l’affichage des écrans. En effet, nous nous sommes inspirés de la structure d’affichage sous forme de « Window List ». Ce format était intéressant puisqu’il récupérait l’information liée aux noms des fenêtres et des icônes des différentes applications.

### 3.5.1 Implémentation

Nous avons dans un premier temps changé l’orientation de la visualisation de la liste, car une barre de tâches est généralement orientée horizontalement. Il a donc fallu recréer le prefab correspondant à chaque élément de la liste. Nous ne souhaitons que garder l’icône de l’application, donc nous avons dimensionné le prefab de façon à ce qu’il n’affiche plus de texte ou autres éléments peu pertinents pour une barre de tâches ergonomique.

Nous avons ensuite recréé l’objet principal qui donne la forme de la liste. Comme dit précédemment, nous voulons notre barre horizontale, ainsi nous avons créé un objet rectangulaire prêt à afficher nos différentes icônes. On lui a rattaché le script **WindowList**, qui à chaque frame récupère l’information du bureau pour afficher ou non les icônes des fenêtres utilisées. Ainsi, pour chacune de ces fenêtres, ce script instancie le nouveau prefab auquel on rattache le script **WindowListItem**. Celui-ci s’occupe de l’affichage de l’icône sur le prefab. Chaque prefab a un composant button, qui permet de rendre les différents objets de type Window visibles ou non.

Pour plus d’optimisation, nous avons choisi de garder fixe la taille de notre rectangle, pour éviter d’avoir un rectangle à rallonge si beaucoup de fenêtres sont ouvertes. On a alors ajouté une barre de défilement sous le rectangle qui permet de naviguer dans la liste d’icônes.

La structure principale de notre barre était alors créée et fonctionnelle. Cependant, comme nous voulions notre virtual desktop le plus personnalisable que possible, nous voulions donner à l'utilisateur la possibilité de choisir la couleur de sa barre de tâches.

### 3.5.2 Personnalisation

Nous avons ensuite créé un panneau permettant à l'utilisateur de choisir la couleur et l'opacité de sa barre.

#### Des couleurs prédéfinies

Nous avons commencé par créer plusieurs boutons ayant chacun une couleur fixe. C'est donc la couleur du bouton qui, lorsqu'on appuie sur celui-ci, allait être attribué au composant Image de notre barre de tâche. Chaque bouton a donc le même script **changeColor** comportant la méthode **SetColor()**. Cette classe a pour attribut un GameObject **barre**, qui est notre objet barre de tâche.

```
public class changeColor : MonoBehaviour
{
    [SerializeField] public GameObject barre; ◀ ...

    ◀ Event handler ◀ 12 asset usages
    public void SetColor()
    {
        Color color = GetComponent<Image>().color;
        color.a = barre.GetComponent<Image>().color.a;
        barre.GetComponent<Image>().color = color;
    }
}
```

*Code de la classe changeColor*

On remarque que l'opacité de la barre est conservée : le canal alpha `color.a` prend la valeur de celui initialement défini sur l'objet **barre**.

Ainsi, nous avons créé 12 boutons de couleurs différentes appelant chacun la méthode `SetColor()`.

## Opacité modulable

La seconde étape dans la personnalisation de notre barre de tâches concernait le choix de son opacité. Nous avons créé un second script contenant la classe **opacity**. Contrairement aux boutons, ce script est directement rattaché à l'objet barre de tâche. Elle n'a en fait qu'un attribut : une zone de texte affichant le pourcentage de l'opacité de la barre.

Elle possède aussi 2 méthodes : `ChangeOpacity(float value)` et `AdjustText(float value)` où *value* est la valeur prise par le slider permettant de changer l'opacité.

```
public class opacity : MonoBehaviour
{
    public Text percentText; ❸ ...
    ❹ Event handler ❹ No asset usages
    public void ChangeOpacity(float value)
    {
        Color currentColor = GetComponent<Image>().color;
        currentColor.a = value;
        GetComponent<Image>().color = currentColor;
    }

    ❹ Event handler ❹ No asset usages
    public void AdjustText(float value)
    {
        percentText.text = Convert.ToInt32(value*100) + " %";
    }
}
```

*Code de la classe opacity*

La dernière fantaisie concernant l'opacité est la couleur du slider qui change

selon la couleur de la barre. Le changement des différents composants du slider s'effectue grâce à la classe **OpacitySliderColor**, basée sur le même principe du changement de couleur via les boutons.

### Couleur entièrement modifiable

Le dernier ajout dans le panneau permettant le changement de couleur de la barre de tâches fut une option permettant de choisir soi-même la couleur, grâce aux valeurs des canaux R, G et B des pixels.

Nous avons donc choisi d'implémenter 3 objets de types barres coulissantes correspondants à chacun des canaux. L'utilisateur peut alors augmenter ou diminuer à sa guise chaque canal, ce qui lui permet d'avoir accès à toutes les couleurs possibles.

Ainsi, nous avons créé une nouvelle classe : **customizedColor**. Celle-ci est aussi rattaché à l'objet barre de tâches et possède 6 attributs :

- 3 sliders : **RedSlider**, **GreenSlider**, **BlueSlider**, correspondant respectivement aux sliders des canaux rouge, vert et bleu (R, G, B)
- 3 objets Text : **RedValue**, **GreenValue**, **BlueValue**, correspondant respectivement aux textes affichant les valeurs des canaux rouge, vert et bleu (R, G, B). Ces valeurs vont de 0 à 255.

Cette classe possède aussi 3 méthodes appliquées au sliders :

- `changeRedChannel(float value)`
- `changeGreenChannel(float value)`
- `changeBlueChannel(float value)`

où *value* est la valeur prise par le slider.

```
public void changeRedChannel1(float value)
{
    Color newColor = GetComponent<Image>().color;
    newColor.r = value;
    GetComponent<Image>().color = newColor;
}

Event handler No asset usages
public void changeGreenChannel1(float value)
{
    Color newColor = GetComponent<Image>().color;
    newColor.g = value;
    GetComponent<Image>().color = newColor;
}

Event handler No asset usages
public void changeBlueChannel1(float value)
{
    Color newColor = GetComponent<Image>().color;
    newColor.b = value;
    GetComponent<Image>().color = newColor;
}
```

*Code des méthodes des sliders*

Enfin, la fonction `Update()` de cette classe permet d'assurer la synchronisation des sliders et des valeurs contenues dans les zones de textes. En effet, si l'utilisateur change la couleur de sa barre via les boutons aux couleurs prédéfinies, la couleur du sprite affichant la couleur courante ainsi que les 3 sliders doivent être en accord avec les nouvelles valeurs :



```
public void Update()
{
    // synchronize Les sliders selon les valeurs des channels
    RedSlider.GetComponent<Slider>().value = GetComponent<Image>().color.r;
    GreenSlider.GetComponent<Slider>().value = GetComponent<Image>().color.g;
    BlueSlider.GetComponent<Slider>().value = GetComponent<Image>().color.b;

    // synchronize Les valeurs des zones de textes
    RedValue.text = Convert.ToInt32(GetComponent<Image>().color.r * 255).ToString();
    GreenValue.text = Convert.ToInt32(GetComponent<Image>().color.g * 255).ToString();
    BlueValue.text = Convert.ToInt32(GetComponent<Image>().color.b * 255).ToString();
}
```

*Code de la fonction `Update()` de la classe `CustomizedColor`*

Remarquons que les canaux R, G et B prennent une valeur flottante entre 0 et 1, d'où la multiplication par 255 pour afficher la valeur correspondante dans les zones de textes.

## 3.6 Le menu associé à la barre des taches

*Réalisé par Réhan*

On a implémenté un bouton pour afficher le menu, qui permet de personnaliser l'application pour améliorer l'expérience utilisateur. Pour que le menu soit agréable à utiliser, une bonne solution est de faire deux boutons, un pour afficher le menu et un pour effacer le menu. Lorsque le premier est cliqué, il affiche tous les GameObjects des boutons du menu, s'efface lui-même et affiche le bouton pour effacer le menu. Le bouton pour effacer le menu a le fonctionnement inverse.

Vous trouverez les images des boutons du menu ajouté à la barre de tâches dans les annexes.

### 3.6.1 Bouton « Quit »

On a implémenté un bouton Quit pour fermer l'application, en associant un script au bouton. Le script exécute la fonction `quit()` qui utilise la fonction `UnityEngine.Application.Quit()`.

On aurait pu se contenter de la fermeture depuis le casque, qui permet de quitter n'importe quelle application à n'importe quel moment, mais cette manière donne plus de risques de bug, il était donc préférable d'implémenter un bouton pour fermer l'application depuis elle-même sans passer par le casque.

### 3.6.2 Réglage du volume

Pour modifier le volume de l'application indépendamment du volume du casque VR, on a ajouté un slider qui modifie le volume global de l'application. Cela permet de personnaliser un peu plus l'application et de ne pas devoir changer le son du casque à chaque utilisation. Le script associé modifie l'attribut **AudioListener.volume** en fonction de la valeur du slider, ce qui permet de modifier le volume général de l'application et pas seulement d'une seule AudioSource.

### **3.6.3 Bouton changement de couleurs**

Ce bouton permet d'afficher le panneau de couleur expliqué dans la section 3.5.2 de ce rapport. Il permet la personnalisation de la barre de tâches.

### 3.7 Déplacement des fenêtres

*Réalisé par Réhan*

Pour que notre environnement soit réellement agréable à utiliser, on s'est inspirés du bureau de Tony Stark (Iron Man) qui bouge toutes ses fenêtres avec ses mains. L'idée d'un tel bureau fait rêver, on a donc mis en place un tel système dans notre application.

Pour pouvoir bouger les fenêtres, il a fallu utiliser les classes **OVR Grabber** et **OVR Grabbable** fournies par Oculus. En ajoutant le composant OVR Grabbable à un GameObject, cet objet devient attrapable, cela permet d'attraper des objets avec les manettes Oculus. La classe **OVR Grabber** rend l'objet attrapeur, cette classe est généralement appliquée sur les manettes pour pouvoir attraper un objet avec sa main. Cependant, pour améliorer encore l'expérience utilisateur, on a également appliqué cette classe sur le pointeur ce qui permet « d'attraper » une fenêtre à distance pour la déplacer, comme on le ferait avec notre souris sur un pc.

Ainsi, on peut bouger toutes les fenêtres comme Iron Man avec nos mains et également comme une souris de pc avec le pointeur du contrôleur.

### 3.8 Création d'un environnement personnalisable

*Réalisé par Gianluca*

Dans cette partie, nous avons voulu permettre à l'utilisateur une personnalisation de son environnement bureautique. Pour pouvoir faire cela nous avons eu l'idée d'utiliser une API qui liera notre application à la librairie d'objet 3D du nom de *Poly*. Cette librairie créée par Google permet de répertorier une multitude d'objet 3D qui sont exportés par des utilisateurs. Nous avons donc créé un menu compatible avec cette API qui permet de rechercher et importer les objets que l'on veut pour personnaliser notre environnement en temps réel.

Le menu se situe à droite de la barre de tâche. Une fois ouvert, une barre de recherche est proposée, lorsque l'on cherche ce que l'on veut. La demande basée sur notre recherche est envoyée à l'API. Cette demande peut être modifiée par plusieurs paramètres, par exemple on peut demander le nombre d'asset que l'on veut (dans notre cas on en demande que 9 car notre menu a que 9 boutons). On peut demander aussi le filtre qu'il applique à la recherche : si on veut les plus beaux, les plus récents, les plus likés. Dans notre cas on a mis le paramètre qui tri les assets par ceux de meilleure qualité en premier. L'API nous renvoie le résultat de nos recherches qui seront affichés dans le menu.

Une fois la recherche faite, on récupère une liste d'asset. Chaque asset est trié par de nombreuses étiquettes : son nom, le nom de l'auteur, son ID. Dans notre cas, pour pouvoir afficher leurs miniatures, on a utilisé leur ID, qui, avec une méthode de cette API, nous permet d'importer la même miniature que l'on peut retrouver sur le site. Ensuite, on remplace l'image de nos boutons par celles des miniatures et on rajoute en-dessous leurs titres et le nom de l'auteur, pour avoir le droit d'importer les objets 3D. Une fois que l'utilisateur fait le choix d'importer l'asset de son choix, il clique sur la miniature, ce qui lance la procédure d'importation de l'API. Une fois l'API importé, on fait spawn l'objet devant l'utilisateur.

Une fois l'objet spawner, l'utilisateur peut le placer comme il le veut dans son environnement en changeant sa taille et en le déplaçant où il veut sur la scène.

Vous trouverez une capture de cette nouvelle fonctionnalité dans les annexes.

### 3.9 Réalisation d'un tutoriel

*Réalisé par Hector*

Afin que l'utilisateur puisse se familiariser avec notre application et son fonctionnement, nous avons décidé de concevoir un tutoriel. Voici comment nous nous y sommes pris :

#### 3.9.1 Détection du premier lancement

Tout d'abord, il fallait détecter si l'utilisateur avait déjà ouvert l'application précédemment. Pour cela, nous avons utilisé les PlayerPrefs de Unity pour enregistrer l'information. Expliquons son principe :

On initialise une valeur « isFirst » à 0. Si la valeur est à 0 alors on charge la scène du tutoriel et on change la valeur de « isFirst » à 1, sinon, on charge la scène "régulière".

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class FirstTime : MonoBehaviour
6  {
7      public int firstTime;
8
9      void Start()
10     {
11         firstTime = PlayerPrefs.GetInt("isFirst");
12         if (firstTime == 0)
13         {
14             SceneManager.LoadScene("Tutoriel");
15             PlayerPrefs.SetInt("isFirst", 1);
16         }
17         else
18         {
19             SceneManager.LoadScene("Poly Menu");
20         }
21     }
22 }
23
```

*Code de la classe FirstTime*

Mais malheureusement, ce code ne suffisait pas en lui-même. En effet, à chaque fois que nous lançons l'application, le script chargeait une des deux scènes puis rechargeait le script (car il était inclus aux scènes), ce qui enclenchait le chargement d'une des deux scènes, etc... et cela à l'infini. Pour résoudre ce problème, nous avons créé une nouvelle scène vide "Loader" à qui nous avons appliqué le script **First Time**. Ainsi, quand le programme se lance, "Loader" est chargé en premier avant de charger une des deux scènes en fonction de la valeur du *playerpref*.

### 3.9.2 Conception du décor

La seconde étape a été de réaliser un décor pour le tutoriel. La première idée que nous avons eu a été de construire une petite chambre avec une table, un écran, une lampe, une porte et une fenêtre. Cette première version aurait très bien pu faire l'affaire mais nous avons néanmoins choisis de changer de plan en construisant un second décor plus spacieux et plus lumineux, donc plus agréable.

Ce dernier est constitué d'une île (où se situe l'environnement de travail avec le bureau, l'écran et la chaise) au milieu d'un lac lui-même entouré d'une ville, le tout sous un ciel étoilé. (Nous vous invitons à jeter un œil à la scène dans les annexes.)

Nous avons utilisé la bibliothèque d'objets 3D Poly de Google pour la plupart des éléments de décors. Cette dernière étant assez riche, nous avons pu trouver tout ce dont nous avons besoin assez facilement pour finalement avoir un rendu plutôt satisfaisant.

### 3.9.3 Le tutoriel

Une fois tous ces aspects esthétiques mis en place, il était temps de s'attaquer au cœur même du tutoriel, à savoir la formation de l'utilisateur à notre application.

Nous avons rajouté un écran plat au décor afin d'afficher les instructions ainsi qu'un bouton "continuer" pour permettre à l'utilisateur d'avancer dans le tutoriel de *MyDesk*. Voici le code :

```
1 using System;
2 using System.Collections;
3 using System.Collections.Generic;
4 using System.Globalization;
5 using UnityEngine;
6 using UnityEngine.SceneManagement;
7 using UnityEngine.UI;
8
9 public class tutorial : MonoBehaviour
10 {
11     public Text text;
12     public Button button;
13     private int etape = 1;
14
15     void clickAction(Button buttonClicked)
16     {
17         etape += 1;
18     }
19
20     void Start()
21     {
22         text.gameObject.SetActive(false);
23         button.gameObject.SetActive(false);
24     }
25
26     void Update()
27     {
28         if (Input.anyKeyDown)
29         {
30             text.gameObject.SetActive(true);
31             button.gameObject.SetActive(true);
32         }
33
34         if (etape == 1)
35         {
36             text.text = "Bienvenue dans le tutorial de MyDesk";
37         }
38         if (etape == 2)
39         {
40             text.text = "Utilisez la barre de tâche pour accéder aux différents onglets";
41         }
42         if (etape == 3)
43         {
44             text.text = "Utilisez la gâchette arrière pour déplacer les écrans";
45         }
46         if (etape == 4)
47         {
48             text.text = "A présent, allez dans le menu Poly en pointant le logo";
49         }
50         if (etape == 5)
51         {
52             text.text = "Recherchez un élément de décor et utilisez le bouton A pour le faire apparaître";
53         }
54     }
55 }
```

*Code de la classe tutorial*

Quand le bouton "continuer" est pressé, la variable *etape* augmente de 1. Le texte affiché sur l'écran géant étant dépendant de la valeur de cette variable, l'utilisateur avance donc dans le tutorial. Finalement, une fois que *etape* atteint la valeur finale de 11, le programme quitte le tutorial et charge la scène principale.



```
76  if (etape == 11)
77  {
78      SceneManager.LoadScene("Menu");
79  }
```

*Dernière étape*

### 3.10 Le lecteur audio

*Réalisé par Lina*

Nous avons ensuite décidé d'ajouter un feature supplémentaire à notre projet : un lecteur audio. Celui-ci permet à l'utilisateur de jouer le morceau de son choix parmi une playlist de musique (nous avons uniquement téléchargé des morceaux gratuits et libres de droits).

#### 3.10.1 Le lecteur et ses boutons

La première étape fut de choisir nos différents morceaux d'ambiance. Grâce au site *www.bensound.com*, nous avons pu télécharger 8 fichiers .mp3 contenant des pistes audios instrumentales, propices à une ambiance sereine et de travail (libre à l'utilisateur d'importer ses propres fichiers s'il est plus productif dans une ambiance de hard rock ou de métal!).

Nous avons donc créer 8 gameobjects auquel on rattache les pistes audios. Ces 8 objets ont pour objet parent « Playlist ». Afin d'identifier ces objets comme étant des chansons contenues dans une playlist, nous avons dû créer les classes Song et Playlist.

#### L'objet Song

L'objet Song a 4 attributs :

- Un **audioClip** : notre piste audio
- Un **nom** : le nom de la playlist, ici « Playlist » (très original en effet)
- Un **nombre de chanson** : le nombre d'objet dont il est enfant

La fonction **Start** permet d'initialiser ces attributs et ajouter les différents objets Song à la tracklist.

```
public class Song : MonoBehaviour
{
    public AudioClip audioClip;  ⚡ Serializable
    [SerializeField]
    public string Name;  ⚡ Unchanged
    [SerializeField]
    public Playlist Playlist;  ⚡ Unchanged
    public int trackNumber;  ⚡ Unchanged

    // Start is called before the first frame update
    ⚡ Event function
    void Start()
    {
        Name = audioClip.name;
        Playlist = transform.parent.GetComponent<Playlist>();
        trackNumber = transform.GetSiblingIndex() + 1;
    }
}
```

*Code de la classe Song*

A noter que nous ajoutons +1 au TrackNumber car une liste de chanson ne commence généralement pas à l'index 0.

## L'objet Playlist

L'objet Playlist a 4 attributs :

- Une **liste de chanson** : sa tracklist qui sont les objets Song dont il est parent
- Un **nom** : le nom de la playlist, ici « Playlist » (très original en effet)
- Un **nombre de chanson** : le nombre d'objet dont il est enfant

La fonction **Start** permet d'initialiser ces attributs et ajouter les différents objets Song à la tracklist.

```
public class Playlist : MonoBehaviour
{
    [SerializeField] public List<Song> songList;  ◀ Serializable
    public string Name;  ◀ Unchanged
    public int NbOfSongs;  ◀ Unchanged

    // Start is called before the first frame update
    ◀ Event function
    void Start()
    {
        Name = this.name;
        NbOfSongs = transform.childCount;
        foreach (Transform child in transform)
        {
            songList.Add( item: child.GetComponent<Song>());
        }
    }
}
```

*Code de la classe Playlist*

## L'objet MusicManager

Nous avons ensuite créer l'objet MusicManager, qui sera l'objet comprenant la source audio. Ses attributs principaux sont une **playlist**, une **chanson** Song et un **entier** CurrentTrack.

A l'appel de la fonction **Start()**, on initialise sa chanson, qui sera l'objet Song contenue à la position 0 de la playlist, ainsi que son volume et les textes correspondant au temps de lecture. Ceux-ci sont répartis autour du slider **ProgressBar**, dont la valeur est le temps joué par la piste audio courante.

```
void Start()
{
    CurrentTrack = 0;

    // Chanson
    song = playlist.songList[CurrentTrack];

    // Source audio
    audioSource = GetComponent();
    audioSource.clip = playlist.songList[CurrentTrack].audioClip;

    // Setting the volume
    audioSource.volume = 0.5f;
    VolumeBar.value = 0.5f;

    // Affichage du temps de lecture
    timePlayed.text = "0:00";
    int lengthInSeconds = Convert.ToInt32(audioSource.clip.length);
    timeLeft.text = lengthInSeconds / 60 + ":" + AddZero( sec: lengthInSeconds % 60) + lengthInSeconds % 60;
}
```

*Code de la fonction **Start()** de la classe **MusicManager***

La fonction `Update()` permet d'assurer que le nom de la chanson affiché correspond bien à celui de son objet `Song`, ou encore que le texte affichant les temps en minutes correspondent aux bonnes valeurs.

On change également les icônes du volume si le booléen `audioSource.mute` vaut vrai. Le `MusicManager` possède également quelques méthodes permettant d'arrêter ou jouer la piste audio, de la mettre en mute, de la recommencer, de passer à la chanson suivante ou précédente. (Cf. annexe figure ...)

```

void Update()
{
    song = playlist.songList[CurrentTrack];
    audioSource = GetComponent();
    audioSource.clip = playlist.songList[CurrentTrack].audioClip;

    // Affichage du nom de la chanson et playlist
    SongName.text = song.Name;

    // Progress Bar
    ProgressBar.value = audioSource.time / song.audioClip.length;

    // Volume Icon update
    if (audioSource.mute)
    {
        VolumeIcon.GetComponent<Image>().sprite = MuteSprite;
    }
    else
    {
        VolumeIcon.GetComponent<Image>().sprite = UnmuteSprite;
    }

    // Affichage du temps de lecture
    // Time played
    int playedInSecond = Convert.ToInt32(audioSource.time);
    timePlayed.text = playedInSecond / 60 + ":" + AddZero(sec: playedInSecond % 60) + playedInSecond % 60;

    // Time left to play
    int lengthInSecond = Convert.ToInt32(audioSource.clip.length);
    int leftInSecond = lengthInSecond - playedInSecond;
    timeLeft.text = "-" + leftInSecond / 60 + ":" + AddZero(sec: leftInSecond % 60) + leftInSecond % 60;
}

```

*Code de la fonction `Update()` de la classe `MusicManager`*

## L'objet `ButtonManager`

Comme son nom l'indique, le `ButtonManager` regroupe l'ensemble des Gameobjects formant les boutons. On en compte 6 :

- Le bouton *Play* : joue le morceau courant
- Le bouton *Pause* : met en pause le morceau courant
- Le bouton *Mute* : change la valeur du booléen `audioSource.mute` du morceau courant
- Le bouton *Restart* : initialise la valeur de l'entier `audioSource.time` à 0
- Le bouton *Next* : augmente l'entier `CurrentTrack` de 1, s'il est égal à la longueur de la playlist - 1, il prend la valeur 0
- Le bouton *Previous* : diminue l'entier `CurrentTrack` de 1, s'il est égal à 0, il prend la valeur de la longueur de la playlist - 1

Le `ButtonManager` est associé à chacun des boutons et fait directement appel aux méthodes du `MusicManager`.

### 3.10.2 La bibliothèque musicale

Afin de rendre le choix des musiques plus facile, nous avons implémenté une petite bibliothèque musicale qui se présente sous forme de liste de chansons. Elle n'apparaît pas directement à l'ouverture du lecteur audio, mais lorsqu'on appuie sur le bouton en haut à gauche du lecteur.

#### Création d'un prefab

Dans notre liste, nous voulions représenter chaque chanson de la même manière : sous forme de case avec le titre, la longueur et le numéro de la chanson. De plus, si la chanson de la case est celle jouée par le lecteur, l'opacité de la case est augmentée et l'icône play apparaît à côté du numéro de chanson. (Cf. annexe)

Notre prefab **SongCase** comprend donc 4 objets enfants : trois zones de textes et un `Sprite` (une image) pour l'icône play.

#### Création de la classe `SongListElement`

Une fois l'aspect graphique défini grâce au prefab, nous avons créé la classe `SongListElement`, dont on rattache le script au **SongCase**. Celle-ci possède 2 attributs : un objet **MusicManager** et un objet **Song**.

Ensuite, nous avons écrit 3 méthodes appelées dans `Start()` :

- `DisplaySongName()` : le texte `SongName` du prefab affiche le nom de l'objet `Song`.
- `DisplayTrackNumber()` : le texte `TrackNumber` du prefab affiche l'entier `TrackNumber` de l'objet `Song`.
- `DisplaySongLength()` : le texte `SongLength` du prefab affiche la longueur de la piste audio de l'objet `Song`.

```

void DisplayTrackNumber()
{
    // Displays track number
    Transform TrackNumber = transform.Find("TrackNumber");
    Text TNumberText = TrackNumber.GetComponent<Text>();
    TNumberText.text = song.trackNumber.ToString();
}

void DisplaySongName()
{
    // Displays song name
    Transform SongName = transform.Find("SongName");
    Text SNameText = SongName.GetComponent<Text>();
    SNameText.text = song.Name;
}

void DisplaySongLength()
{
    string AddZero(int sec)
    {
        if (sec < 10)
            return "0";
        else return "";
    }

    Transform SongLength = transform.Find("SongLength");
    Text SNameText = SongLength.GetComponent<Text>();
    int lengthInSeconds = Convert.ToInt32(song.audioClip.length);
    SNameText.text = lengthInSeconds / 60 + ":" + AddZero( sec: lengthInSeconds % 60 ) + lengthInSeconds % 60;
}

```

*Code des 3 méthodes appelées par **Start()** de la classe **SongListElement***

Comme précisé plus haut, nous voulons que notre case change d'aspect graphique si la chanson qu'elle représente est celle qui est jouée par le lecteur. On a donc créé 2 méthodes qui changent ces paramètres graphiques : **PlayedSongSettings()** et **NormalSettings()**. La première définit l'opacité du prefab à 80% et affiche l'icône du triangle et la deuxième méthode définit l'opacité à 30% et masque l'icône.



```

void PlayedSongSettings()
{
    // augmente l'opacité du fond
    Color playedColor = GetComponent<Image>().color;
    playedColor.a = 0.8f;
    GetComponent<Image>().color = playedColor;

    // affiche le petit triangle play sur la gauche
    transform.Find("IsPlaying").GetComponent<Image>().enabled = true;
}

Frequently called
void NormalSettings()
{
    // masque l'icone play
    transform.Find("IsPlaying").GetComponent<Image>().enabled = false;

    // met l'opacité normale
    Color playedColor = GetComponent<Image>().color;
    playedColor.a = 0.3f;
    GetComponent<Image>().color = playedColor;
}

```

*Code des 2 méthodes appelées par `Update()` de la classe `SongListElement`*

Ces deux fonctions sont appelées par `Update()` qui teste si l'objet `Song` est le même que celui du `MusicManager` (= la chanson jouée).

## Création de la classe `SongList`

La classe `SongList` représente notre bibliothèque musicale, et a pour but de créer les différentes cases pour les chansons de notre playlist. Elle a donc 3 attributs : une **Playlist**, un **MusicManager** et un `GameObject` **songCase** (qui ici est notre prefab).

Elle possède une méthode `CreateCases()` qui crée un objet **songCase** pour chaque chanson de la playlist.

```
void CreateCases()
{
    foreach (Song song in playlist.songList)
    {
        songCase.GetComponent<SongListElement>().song = song;
        songCase.GetComponent<SongListElement>().musicManager = musicManager;
        GameObject myCase = Instantiate(songCase, transform);
    }

    instancied = true;
}
```

*Code de la méthode `CreateCases()` de la classe `SongListElement`*

## 4 Récit de réalisation

### 4.1 Quelques difficultés...

Comme dans toute réalisation de projet, nous étions amenés à faire face à quelques obstacles.

La difficulté principale et évidente selon nous a été la distance. En effet, un travail de groupe est évidemment plus difficile lorsqu'il est impossible de se réunir physiquement. Néanmoins, cette situation a été la même pour tout le monde à l'échelle mondiale et pourrait peut-être se reproduire (même si on ne l'espère pas), il a donc fallu s'adapter et tirer des leçons de cette période si particulière. Les outils de communication nous ont jamais été aussi utiles, et ont permis de maintenir un contact et une coordination au sein de notre groupe.

### 4.2 ... mais aussi beaucoup de positif !

Malgré la distance, nous avons su relever le défi du travail en groupe. Même si nous ne nous connaissions pas vraiment entre nous, la réalisation de ce projet nous a permis de faire connaissance et découvrir les talents et ambitions de chacun.

La bonne entente a toujours régné, et nous avons toujours su donner priorité à la communication pour atteindre nos objectifs. En plus d'un enrichissement humain, chacun a pris plaisir à contribuer à ce projet en mettant en avant ses qualités. Ce projet a donc été une belle expérience pour chaque membre.

## 5 En conclusion

### 5.1 Ressenti individuel

#### 5.1.1 Hector

De mon point de vue, cette soutenance s'est mieux passé que prévu. Je pensais plus subir la pression du dernier rendu, mais j'ai finalement bien réussi à gérer ma quantité de travail ce qui m'a permis de ne pas faire de nuits blanches pour résoudre mes problèmes en vitesse. De plus, je commençais à bien maîtriser l'outil Unity ce qui me permet d'être plus rapide et de perdre moins de temps à faire des recherches qu'avant.

Pour ce qui est du travail collectif, je suis assez fier de la solidarité et de la communication qui règne dans notre groupe. Nous avons réussi à finir le projet dans une ambiance positive, sans se haïr les uns les autres, ce qui peux pour certains relever de l'exploit. Enfin je tiens à souligner que personne n'a fui ses responsabilités ainsi que sa charge de travail, ce que j'ai personnellement apprécié.

#### 5.1.2 Réhan

Avoir été confronté à un projet dès la première année a été pour moi une expérience très enrichissante. J'ai commis énormément d'erreurs qui me permettront d'en faire beaucoup moins la prochaine fois. Ce projet m'a permis d'avoir un avant-gout de la réalité du travail et m'a permis de voir les difficultés à travailler en équipe, à se coordonner. De plus, avec la situation de confinement, et les cours à distance, sans pouvoir voir ses camarades, la motivation à baissée et la tentation de ne pas travailler s'est fait ressentir. Ceci m'a encore plus montré à quel point la gestion du temps est importante pour mener à bien un projet.

Notre groupe avait une très bonne entente, ce qui a grandement facilité la coordination et la communication. De plus, l'outil Unity collab a permis de travailler facilement à plusieurs sur une scène sans passer par des échanges de code compliqués. Ce projet m'a permis d'apprendre des choses qu'on ne voit pas en cours, de mieux comprendre l'utilisation d'assets, de préfabs, etc... même si le principal enseignement que je tire de cette expérience réside dans l'organisation du travail de groupe.

Je suis fier de l'application qu'on a réussi à créer, je ne savais pas du tout à quoi m'attendre, surtout qu'on se lançait dans la VR, ce qui était nouveau pour moi. Ce domaine nous a forcé à faire plus de recherches, car il y a beaucoup moins de ressources disponibles que dans d'autres domaines, mais ce fut bénéfique dans notre formation.

### **5.1.3 Lina**

Je suis très satisfaite de l'application que notre groupe a créée. D'un point de vue plus personnel, c'est à l'issue de cette troisième soutenance que j'ai le plus appris. Je me suis lancé de réels défis, notamment avec la création du lecteur audio. J'ai encore une fois appris à chercher, réfléchir à mes propres solutions, non sans me prendre la tête de temps en temps... Je suis fière de mon investissement et également du résultat.

Côté groupe, je suis contente d'avoir contribué à ce projet avec Gianluca, Hector et Réhan. Nous avons su coopérer pour aboutir à la création d'une application qui nous plaît, et qui a été très intéressante à réaliser. Enfin, ce projet m'a fait découvrir la réalité virtuelle, un domaine que je ne connaissais pas avant et que je trouve fascinant.

### **5.1.4 Gianluca**

Après un semestre entier à avoir développé un projet innovant avec une équipe sympathique. La première chose que je retiendrais c'est de ne pas faire un projet aussi ambitieux avec aussi peu de moyen. Quand je parle de moyen, c'est de moyen tout autant matériel qu'intellectuel.

. Pour rappel notre projet était de faire une alternative à notre utilisation de notre ordinateur en utilisant la VR, pour être plus efficace, productif et être dans une ambiance dépayssante. Dans le fond tous nos objectifs ont été partiellement réussis.

Ce que je veux dire par là c'est que toutes les fonctionnalités annoncées au cahier des charges ont été réalisées mais certains ont un résultat correct mais n'arrivant pas à l'échelle d'excellence que l'on s'est fait. Un des facteurs majeurs à cela, selon moi, aura été le matériel.

Etant le seul à avoir un casque de VR, j'ai dû être au centre de toutes modifications pour les testés. J'ai donc dû suivre toutes les implémentations, de

mon côté d'un point de vue de mon apprentissage cela a été excellent car j'ai eu un gain en connaissance c# et unity fulgurant. Je suis passé d'être apeurer par les codes à faire et parfois perdu par les quantités de codes. A être à l'aise face au projet, créer des solutions aux problèmes à bases de scripts et être indépendant des aides sur internet, qui avant m'était essentiel et maintenant n'est qu'un soutien pratique à mon travail.

Le meilleur exemple d'expérience qui ma forger dans le domaine a été la création du pointeur VR de A à Z grâce à un tutoriel. Et surtout le fait de l'agencer avec les fenêtres du bureau VR. Car d'un, vu que cela est un problème très spécifique il n'y avait aucune solution voir même d'aides à lesquelles je pouvais m'accrocher. J'ai donc dû comme « un vrai ingénieur » posé le problème, voir comment faire pour connecter un pointeur qui se basé de coordonnées x,y,z sur Unity à faire bouger une souris sur un bureau qui n'a pas les mêmes unités de longueurs (longueurs en pixel pour le bureau et valeurs flottant sur Unity) et qui a des dimensions x, y.

Le fait de résoudre ce problème seul grâce à des outils mathématiques et à des nombreux essais pour trouver la solution exacte au problème j'ai découvert la satisfaction qu'un ingénieur peu avoir une fois son problème résolu. Malgré tous ces côtés positifs de la grande quantité de travail que j'ai eu, il m'est arrivé souvent d'imaginer pendant mes heures de travaux comment la répartition de travail aurait été si un ou plusieurs de mes camarades avaient aussi un casque pour effectuer les tests.

Car à chaque fin de soutenance il me fallait implémenter des fonctionnalités qui fonctionne parfaitement sur le bureau de mes collègues à une appli VR. Il m'a fallu parfois repenser voire refaire tout une partie pour l'agencer dans le projet. Au-delà de l'épuisement physique je suis content d'avoir travailler avec l'équipe que j'ai eue. Je les aie découvert d'une manière que je n'aurais pas connu sans se projet. Mon seul regret vis-à-vis de ce projet aura été le fait qu'il m'ai fallu passer plusieurs heures de plus pour implémenter les résultats fonctionnel de mes camarades dans le projet à cause du matériel. J'aurais préféré des tâches un peu plus réparties, non pas par paresse mais plutôt pour éviter un grand stress et de grandes fatigues. De plus je tiens à rappeler que cette répartition n'a pas était faite non pas parce que mes camarades était fainéant mais par un problème matériel. De plus avec le confinement, le fait de ne pas pouvoir se voir et partager mon casque ma isoler dans les tests à faire et la finalisation du projet. Mais heureusement grâce à notre communication efficace et la persévérance de mon équipe face à ce challenge on réussit à prouver qu'une équipe ignorant tout de unity, c# et de la VR était

capable de finaliser un projet de cette envergure qui s'inspire d'application créée par des grandes firmes comme Oculus. Dorénavant lors de la création et la gestion d'un projet, qui sont des choses que j'aime et que j'ai appris à aimer, j'utiliserais l'expérience de ce projet pour prendre plus de recul sur le temps et le matériel à prendre en compte avant de me lancer. Je tiens à remercier mon équipe, Lionel Brosius et Paul Hervot. Mon équipe qui sans leurs aides je n'aurais fait que 1/10 de mes idées et aussi grâce à leurs esprits novateur qui ont proposer et implémenter des fonctionnalités que je n'aurais jamais penser.

De l'autre Lionel Brosius et Paul Hervot, pour avoir pris leurs temps à nous guider et nous corriger, surtout Lionel Brosius qui pendant cette période de confinement s'est occupé de tous nos appels et visionnage de vidéo. Ce projet aura été ma première expérience dans la création et la gestion de projet. Elle n'a pas été la meilleure, mais en prenant du recul elle sera celle qui me rendra meilleur pour éviter les mêmes erreurs dans la réalisation des prochains projets que j'ai en tête.

## 5.2 Conclusion générale

La réalisation de ce projet s'est donc faite sereinement et dans la bonne entente. Chaque membre du groupe a su s'investir pour acquérir de nouvelles compétences, à la fois humaines et techniques. Nous garderons tous un très bon souvenir de la création de notre première application, qui marque la fin de notre première année à l'Epita.