Ryan PRAK (Chef de Projet) Chiyoung
LEE

Aymeric HESNARD

Rapport de la première soutenance

Le groupe Les Radiants a pour objectif ultime de finaliser le projet INVASION et d'en faire un vrai FPS équilibré. INVASION est un jeu FPS qui s'inspire des jeux FPS actuels dont les plus connus actuellement : Valorant, Call of Duty, etc.

Nous avons défini dans le cahier des charges que le but de notre projet est de maîtriser divers langages informatiques et logiciels qui allient à la fois intelligence artificielle et système multijoueur. Pour ce faire, nous voulons réaliser un projet concret : un jeu First Person Shooter (FPS). Nous avons choisi ce genre de jeu vidéo car nous pensons qu'il nous amènerait à accomplir notre objectif, par exemple, l'utilisation du logiciel Unity avec le langage C#. De plus, nous avons aussi cité des intérêts collectifs et importants pour notre potentiel carrière d'ingénieur : le travail d'équipe (communication) et l'organisation.

Pour ce rapport de première soutenance, nous souhaitons vous présenter une chronologie de groupe comportant des tâches individuelles.

Dans un premier temps, le journal de bord dans lequel nous rédigé et répertorié le statut de nos tâches (étant donné que nous n'avons pas pensé à prendre immédiatement la chronologie de nos tâches, certaines dates peuvent être approximatives et peuvent englober des périodes plus ou moins allongées):

10/01/2023 :

Nous finalisons l'écriture du CDC, il nous manque en effet le planning des soutenances.

Nous prévoyons pour :

1. La première soutenance
    - Site web (début de la structure puisque des vidéos de démonstration et des images du jeu seront nécessaires pour illustrer/apporter une touche esthétique au site web)
    - Interface et/ou launcher (début de programmation : image du launcher, pas forcément fonctionnel)
    - POO (déplacement et début de mise en place d'un système multijoueur)
2. La soutenance intermédiaire
    - Le jeu est fonctionnel (ajout d'un système de tir…)
    - Interface et/ou launcher fonctionnel (on a un lanceur qui permet à l'utilisateur de démarrer le jeu et d'accéder au site officiel de INVASION)
    - Le site Web est illustré d'images, d'animations du jeu vidéo tel qu'il est actuellement
    - Début de Game Design et Environnement (avec effets visuels et sonores)
    - Début de l'élaboration et de l'utilisation de l'Intelligence Artificielle sur des cibles non vivantes pour effectuer des tests
    - Recherche Multijoueur
3. La soutenance finale
    - Finalisation Game Design et Environnement
    - Finalisation de l'IA
    - Multijoueur, intégration du joueur et de l'IA dans un réseau

Nous prévoyons pour la prochaine séance une mise en commun de liens et de ressources pour accomplir les objectifs de la première soutenance, en particulier les tutoriels Unity, ainsi que l'installation des outils nécessaires. Il est possible que nous prévoyons d'utiliser un tableur Excel pour les coûts du projet. Nous commencerons probablement un diaporama pour l'oral et des esquisses ou croquis pour le site web ainsi que l'interface et le launcher.

Launcher/Interface - (Ryan)

01/03/2023 :

Partage de vidéos youtube afin de programmer le launcher en C# ainsi que des exemples de launcher de différents jeux vidéo FPS et non FPS **(Fig.5 et 6)**.

https://www.youtube.com/watch?v=WedZMTpJxwc

https://www.youtube.com/watch?v=sVkTayi_NdU

03/03/2023 :

Installation de Visual Studio 2019 et des outils nécessaires pour le lanceur. J'ai regardé la première vidéo et j'ai rencontré quelques problèmes, à chaque fois que je rajoute un élément, ça me supprime celui que j'ai précédemment ajouté. J'ai appris plus tard qu'il fallait glisser les éléments à partir du Bureau vers la solution dans Visual Studio grâce à un forum du site Stack Overflow. https://stackoverflow.com/questions/12833294/provide-value-on-system-windows-baml2006-typeconvertermarkupextension-threw-a

Premier launcher **(Fig.7)**.

04/03/2023 :

En sélectionnant l'application WPF sur VIsual Studio, le logiciel m'affiche une fenêtre blanche et vide. Grâce à la boîte à outils à laquelle je peux accéder avec le raccourci Ctrl+Alt+X, j'ai pu ajouter 2 éléments interactifs : les boutons "DÉMARRER" et "LES RADIANTS". En suivant la vidéo, j'ai défini un bouton Démarrer pour lancer le jeu. Pour cela, on double-clique sur le bouton la fenêtre (lanceur) qu'on veut définir. Visual Studio nous ouvre ensuite un onglet vers le fichier .cs qui nous permet de coder le bouton en C#. Selon l'arborescence de fichiers, on utilise la méthode Process.Start qui lance une ressource (ici le fichier exécutable du jeu) et l'associe à un composant Process (c'est-à-dire une instance de la classe Process). Cette association permet de fournir l'accès à des processus locaux et distants, de démarrer et d'arrêter des processus système locaux. Pour utiliser cette méthode, il a fallu utiliser l'espace de nom System.Diagnostics, permettant ainsi l'interaction avec des processus système (gestionnaire de tâches)

```
private void Button_Click_2(object sender, RoutedEventArgs e)
{
    Process.Start("path/Invasion.exe");
}
```

J'ai effectué le même processus pour le bouton "LES RADIANTS". J'ai ajouté une touche esthétique au lanceur en ajoutant notre logo et un fond d'écran **(Fig.9)**. Pour cela j'ai dû coder en XAML, un langage de balisage déclaratif (similaire à HTML) qui permet de simplifier la création d'une interface utilisateur pour une application. J'ai commencé par définir un fond d'écran en utilisant la propriété Background (définit un pinceau qui remplit la zone de contenu du panneau) de classe Grid (définit une grille flexible composée de colonnes et lignes).

```
<Grid.Background>
    <ImageBrush Stretch="None" ImageSource="/launcherwall.png" AlignmentY="Top" AlignmentX="Center"/>
</Grid.Background>
```

Petite précision : L'ajout d'éléments ou d'outils modifie le code XAML de la fenêtre.

Pour ajouter le logo et une phrase d'accroche, j'ai simplement utilisé la boîte à outils en ajoutant un élément image dans la fenêtre à l'emplacement souhaité et défini la source vers le fichier .png ajouté au préalable dans la solution.

```
<Image HorizontalAlignment="Left" Height="226" Margin="304,0,0,0" VerticalAlignment="Center" Width="197" Source="/logo_invasion.PNG"/>
<Label Content="  TAKE AIM AND FIRE !" HorizontalAlignment="Center" Height="43" Margin="0,324,0,0" VerticalAlignment="Top" Width="192" F
```

Pour accéder au site web, nous avons faire des recherches sur Stack Overflow, ce qui nous a amené à ce code là :

```
private void Website_Click(object sender, RoutedEventArgs e)
{
    string htmlFilePath = "C:/Users/mcblo/Desktop/PROJET_S2/Launcher/Launcher_Invasion/WebSite/index.html";
    string fullPath = System.IO.Path.GetFullPath(htmlFilePath);
    ProcessStartInfo psi = new ProcessStartInfo
    {
        FileName = "cmd.exe",
        Arguments = $"/c start {fullPath}",
        WindowStyle = ProcessWindowStyle.Normal
    };
    Process.Start(psi);
}
```

Dans un premier temps on cherche à obtenir un chemin absolu, puis à définir un jeu de valeurs pour rendre le fichier .html exécutable et donc le lancer sur un navigateur. Enfin nous exécutons ce jeu de valeurs pour exécuter le fichier HTML.

Remarque : J'ai d'ailleurs commis une erreur dont j'avais une connaissance, celle de séparer les mots d'un nom de dossier (solution dans ce cas) par un espace plutôt que par un underscore ou un trait d'union, ce qui m'a obligé à créer une autre solution. En effet, lorsque je cherche à accéder à mon fichier .html, ce qui est après l'espace n'est pas pris en compte (chemin d'accès).

Site Web (HTML/CSS) - (Aymeric)

02/03/2023 :

Comme la plupart des membres du groupe n'avait jamais utilisé le langage html ou très peu, nous avons passé un temps assez conséquent à nous documenter et à apprendre les bases et les applications possibles du langage html combiné avec le langage css.

Pour cela nous avons visionné des vidéos explicatives et appris du code sur des sites spéciaux.

Notamment vu l'importance de l'esthétique des sites web aujourd'hui nous avons passé encore plus de temps à nous entraîner sur les différentes utilisations du css.

Pour éditer les fichiers html et css, nous avons utilisé différents éditeurs : Notepad++ et SublimeText.

Ensuite nous avons commencé la création du site web avec ce squelette :

-une page d'accueil avec le logo du projet qui permette d'accéder aux autres page demandé par la consigne qui sont la page de présentation du projet et du groupe; la page faisant le compte des ressources que nous avons utilisé comme les logiciels, les sites utiles que nous avons utilisé ou non et les images; et enfin celle pour télécharger le projet.

Cet accès se fait via une barre en haut de la page :

- la page de ressource qui recense l'ensemble des ressources utilisé sous la forme de deux listes distinctes les ressources utilisées et les liens utiles.
- La page de présentation qui permet de se renseigner sur le projet et ses participants. Étant donné qu'elle est composée principalement de texte brut nous avons décidé d'utiliser le système d'ancre afin de se déplacer plus facilement dans la page.

Nous avons consulté ces sites pour nous repérer parmi les différentes balises :

https://developer.mozilla.org/en-US/

https://www.w3schools.com/

Nous avons cherché un logo en utilisant un générateur de logos en ligne **(Fig.1)**.

https://www.logogenie.fr/

Nous avons recherché une police pour l'ensemble du site web, d'un style jeu vidéo fps : zorque, en ayant consulté ce site : https://graphiste.com/blog/typographies-gratuites-jeux-video-gaming/

03/03/2023 :

Proposition d'un squelette pour la page d'accueil, présentation et ressources **(Fig.2)**.

04/03/2023 :

Modification dans la page ressource du site: ajout de la même barre que dans la page d'accueil afin d'un retour en arrière possible.

Retravail de la page d'accueil du site: amélioration de l'esthétique **(Fig.3 et 4).**

En ce qui concerne le code :

Nous all

index.html > ...

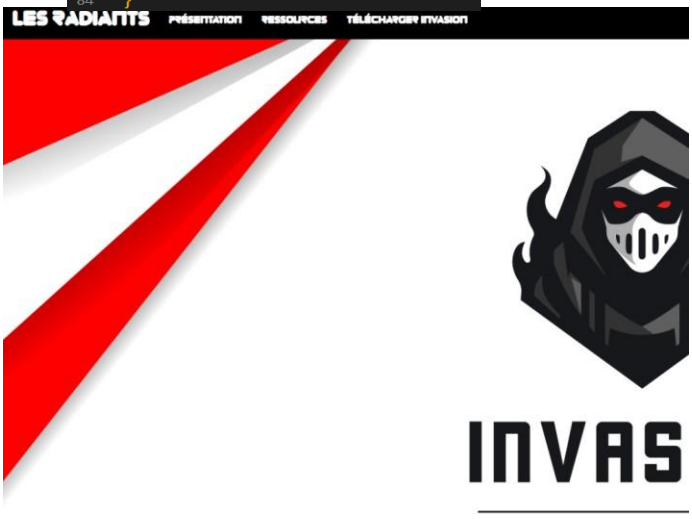
```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8"/>
        <link rel="stylesheet" href="styles/style.css"/>
        <title>Invasion</title>
        <style>
            @font-face {
            font-family: "zorque";
            src: url("fonts/zorque.otf") format("opentype");
        }
        </style>
    </head>
    <body background="img/invasionwall2.jpg">
        <header>
            <ul>
                <li><a href="index.html" style="text-decoration: none;"><h1>Les Radiants</h1></a></li>
                <li><a href="presentation.html" class="navlink">Présentation</a></li>
                <li><a href="ressources.html" class="navlink">Ressources</a></li>
                <li><a href="download.html" class="navlink">Télécharger Invasion</a></li>
            </ul>
        </header>

        <div class="container" align = "center">
            <br><br>
            <h3><img class="logo" src="img/logo_invasion.png" alt="logo of invasion"></h3>
            <br><br><br><br><br>
        </div>

        <footer>
            <h3>Site créé par Chiyoung LEE, Aymeric HESNARD et Ryan PRAK</h3>
        </footer>
    </body>
</html>
```

Le code ci-dessus est celui de la page d'accueil de notre site web. Nous avons décidé afin d'avoir davantage de liberté d'utiliser à la fois un fichier css et de faire appelle à du css dans nos code, ce qui peut être vu dans la partie style (ligne 7 à 11) où nous définissons la police d'écriture que la page devra utiliser.

```
50  header {
51      padding: 5px 10px;
52      background-color: black;
53      margin-top: -8px;
54      margin-left: -8px;
55      margin-right: -8px;
56  }
57
58  header ul {
59      list-style-type: none;
60      margin: 0px;
61      padding: 0;
62      overflow: hidden;
63  }
64
65  header ul li h1 {
66      color: white;
67  }
68
69  header ul li:first-child {
70      margin-top: 5px;
71      margin-right: 15px;
72      margin-left: 5px;
73  }
74
75  header ul li a.navlink:hover {
76      background-color: rgb(255, 0, 0);
77      color: #ffffff;
78      border-radius: 5px;
79      transition: background-color .3s;
80  }
81
82  header ul li {
83      float: left;
84  }
```

mais nous avons également modifié l'aspect de toutes nos en-tête à travers le fichier css.

Par exemple dans le code ci-contre, à la ligne 75 nous définissons que si la souris touche un élément de l'en-tête alors celui-ci changera de couleur et de taille tant que la souris est dessus



Avant que la souris ne soit sur le bouton télécharger **Pendant**

```
download.html
download.html > html > body > div.container > div.items-container > div.row > div.fp-items
21          </ul>
22      </header>
23
24      <div class="container" align = "center">
25          <div class="items-container">
26              <h3><img class="logo" src="img/mini_logo_invasion.png" alt="logo of invasion"></h3>
27              <div class="row">
28                  <div class="fp-items">
29                      <div class="item item-animation">
30                          <h4 align = "center" >Version complète de Invasion</h4>
31                          <button type="download">
32                              <a href="DL_Invasion" download><img src="img/dl_30.png" alt="buttonpng"/></a>
33                          </button>
34                      </div>
35
36
37                  </div>
38                  <div class="fp-items">
39                      <div class="item item-animation">
40                          <h4 align = "center">Version lite de Invasion</h4>
41                          <button type="download">
42                              <a href="DL_Invasion_Lite" download><img src="img/dl_30.png" alt="buttonpng"/></a>
43                          </button>
44                      </div>
45
46                  </div>
47                  <div class="fp-items">
48                      <div class="item item-animation">
49                          <h4 align = "center">Rapport du projet Invasion</h4>
50                          <button type="download">
51                              <a href="Documents/test_doc.pdf" download><img src="img/dl_30.png" alt="buttonpng"/></a>
52                          </button>
53                      </div>
54                  </div>
55              </div>
```

Une partie qui nous causa davantage de problème fut la page de téléchargement de notre site **(Fig. 4)**.

Nous avons dû à plusieurs reprises nous pencher activement sur le code afin de savoir comment faire en sorte que cela soit centré au niveau du centre de la page et de la bonne taille.

Nous avons alors à la fois centrer via le code html mais aussi créer plusieurs classes en css afin que ce que nous voulions faire soit en partie réalisé.

```
146   .item {
147       padding: 30px;
148       transition: all .3s;
149       margin-bottom: 30px;
150       box-shadow: 0 0 5px 0 rgba(68, 43, 148, .2);
151       background: black;
152       color: white;
153       border-radius: 6px;
154       height: 200px;
155
```

Par exemple dans le code ci-contre nous définissons la couleur des "box"(l.151 et l.152), leur taille(l.154) mais aussi l'espace qu'il y aura en dessous des boites(l.149).

Unity - (Chiyoung)

21/01/2023 :

Partage d'une ressource potentielle pour la programmation du jeu (voir Annexe)

26/01/2023 :

Installation de Unity, consultation du site Unity (voir Annexe) et création du repo git en suivant la suggestion des ACDC :

(master) -> (developer) -> (branch1/feature1)

Recherche des différentes features pour créer les différentes branches.

01/03/2023 :

Répartition des tâches sur l'outil de gestion de projet en ligne Trello
1er push sur la branche "develop" concerne :

- un début de projet Unity (création d'une plateforme et des obstacles, et du joueur avec les scripts (PlayerMotor.cs et PlayerController.cs) pour le déplacement du joueur et la rotation de sa caméra)

- création des dossiers "Interface" et "Site"

- création du .gitignore et du README

2e push sur "develop" concerne :

- création des fichiers README dans chaque dossier "Interface" et "Site" afin de push les dossiers

Annexes Unity (liens)
https://www.youtube.com/watch?v=65i-egqXY0o&list=PLUWxWDlz8PYI2OdmIhOnxg92kl0DgS1EN

https://unity.com/download

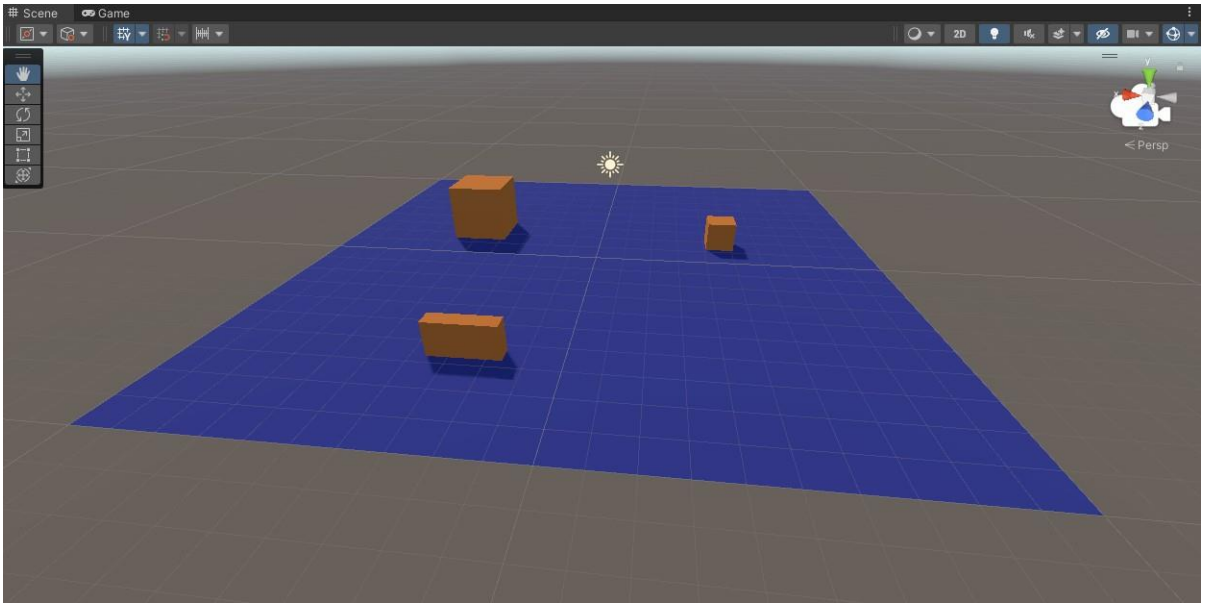
Ce qui suit après n'a pas précisément été daté, on estime fin février et début mars.

1) Progression du projet

- Mise en place de la scène, création du joueur, configuration des mouvements

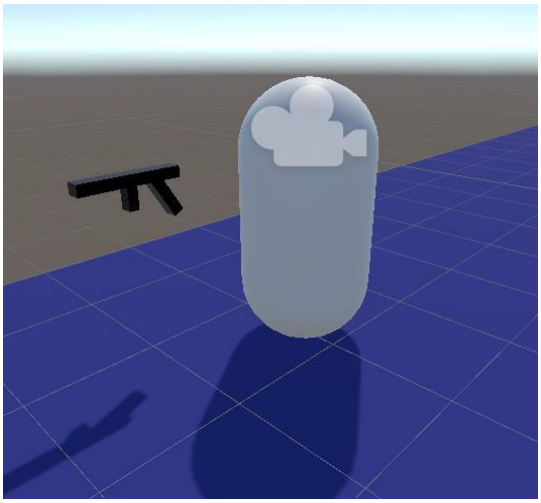
Pour le début de notre projet, j'ai commencé par mettre en place une **plateforme** avec des **cubes** pour se repérer sur la scène ainsi qu'un objet **Light** qui fait office du soleil (source de lumière).
On peut facilement créer de nouveaux objets sur Unity en faisant un clic gauche et en cliquant sur « 3D Object -> objet que l'on veut créer ».

On obtient ainsi un premier rendu de notre scène.



Ensuite, j'ai procédé à la création de notre joueur, donc notre **Player**.
Comme ce n'est encore qu'un prototype, j'ai utilisé l'objet **Capsule** pour simuler le corps de notre joueur. Une fois le corps créé, il lui fallait ajouter un objet **Camera** afin d'obtenir son point de vue (POV) à la **première personne**, comme notre jeu consiste un **FPS** (*First Person Shooter).*
Et enfin, j'ai créé un autre objet qui *ressemble* à une arme à feu à l'aide de 3 objets enfants cubes (encore une fois, ce n'est qu'un prototype) pour avoir une image d'un jeu de FPS.

Voici une première image de notre **Player**, de sa **hiérarchie** et de son **POV**.



La raison pour laquelle j'ai mis l'objet **Gun** comme objet enfant de **Camera** est en effet pour avoir toujours le même **POV** du **Player**, c'est-à-dire que lorsque le joueur tourne sa tête (notamment vers le haut car le corps ne bouge seulement horizontalement), son arme va suivre le mouvement de **Camera** et par conséquent, on peut facilement repérer la direction de son orientation.

Par la suite, il s'agit de permettre au **Player** de pouvoir se déplacer.
Pour cela, j'ai choisi l'un des **components** de Unity, **Character Controller.** En particulier, ce **component** sera non seulement utilisé pour les mouvements du **Player** mais également pour sa rotation de **Camera** ainsi que pour l'action de sauter (**Jump**).

Pour commencer, on crée un nouveau **script** qui va justement contrôler le **Character Controller**, et je le nomme « **FPSController** ».
On utlise évidemment le module **UnityEngine** pour accéder à toutes les fonctions de Unity.
Commençons par rajouter **[RequireComponent(typeof(CharacterController))]** en dehors de notre classe afin de préciser qu'il faut le component **Character Controller** pour pouvoir utiliser ce script.

Ensuite, on déclare toutes les variables nécessaires pour calculer les mouvements de notre joueur : la puissance du saut, la gravité, la caméra, la vitesse de marche, la vitesse de course, les angles limites de rotation autour de l'axe X, une variable qui stock l'angle de rotation autour de l'axe X, un vecteur où l'on va stocker le mouvement calculé, un booléeen qui précise si le joueur peut se déplacer et enfin un Character Controller pour stocker notre component utilisé.

On obtient ainsi un début de code comme ci-dessous :

```
using UnityEngine;

namespace Player2
{
    [RequireComponent(typeof(CharacterController))]
    1 asset usage  Chiyoung *
    public class FPSController : MonoBehaviour
    {
        [SerializeField]
        public float jumpPower = 3f;  "7"

        [SerializeField]
        public float gravity = 10f;  "25"

        public Camera playerCamera;  Changed in 1 asset

        public float walkSpeed = 3f;  Unchanged

        public float runSpeed = 5f;  Unchanged

        public float lookXLimit = 90f;  Unchanged

        public float lookSpeed = 2f;  Unchanged

        private Vector3 _moveDirection = Vector3.zero;

        private float _rotationX;

        public bool canMove = true;  Unchanged

        private CharacterController _characterController;
```

Sur Unity, il existe des fonctions qui s'exécutent automatiquement en respectant certaines conditions (les **Event function**), et les 2 plus classiques sont les fonctions **Start()** et **Update()**. Comme l'indique leur nom, la fonction **Start()** est exécutée lorsque le script se met en route et la fonction **Update()** est appelée à chaque frame.

Dans notre fonction **Start()**, on initialise simplement notre **_characterController**, verrouille le curseur et le rend invisible.

```
Event function  Chiyoung *
void Start()
{
    //Get our component
    _characterController = GetComponent<CharacterController>();

    //Locking the cursor and make it invisible
    Cursor.lockState = CursorLockMode.Locked;
    Cursor.visible = false;
}
```

Dans notre fonction **Update()**, on doit calculer les mouvements effectués à l'instant du frame actuel.
Commençons par le **déplacement du joueur** sur l'**axe X** et **Z**, c'est-à-dire le déplacement sur les côtés, avancer et reculer. D'abord, il faut récupérer les valeurs de chaque axe sous forme de **vecteurs**. On va également utiliser un **booléen** qui vérifie si la touche **Left Shift** est appuyée par la méthode **Input.GetKey**. En effet, cette touche va permettre au joueur de marcher (sinon on considère que le joueur court constamment lorsqu'il se déplace). Ensuite on calcule les déplacement en **multipliant la vitesse de déplacement** (vitesse de marche ou de course en fonction du booléen précédemment évoqué) **par la valeur de la position de l'axe** en question. Enfin, on met à jour notre variable qu'on a défini au début, **_moveDirection** en multipliant la valeur actuelle de chaque axe par les nouvelles qu'on vient de calculer.

```
void Update()
{
    #region  Handles Movement

    //Get values of the transform's actual position in each axis X (right) and Z (forward)
    Vector3 forward = transform.forward;
    Vector3 right = transform.right;

    //Left Shit to walk
    bool isWalking = Input.GetKey(KeyCode.LeftShift);

    //Calculate movement speed in each axis
    float curSpeedZ = canMove ? (isWalking ? walkSpeed : runSpeed) * Input.GetAxis("Vertical") : 0;
    float curSpeedX = canMove ? (isWalking ? walkSpeed : runSpeed) * Input.GetAxis("Horizontal") : 0;

    //Get the position in Y axis of the calculated movement vector (initially = 0)
    float movementDirectionY = _moveDirection.y;

    //update the movement vector in axis X
    _moveDirection = (forward * curSpeedZ) + (right * curSpeedX);

    #endregion
```

Ensuite, il s'agit de coder **l'action de sauter**. En principe, on vérifie si le bouton pour sauter (par défaut la touche **Space**) est appuyé, si le joueur peut se déplacer et s'il est en contact avec le sol. Si ces 3 conditions sont vérifiées, on attribut à la valeur de l'**axe Y** notre variable **jumpPower**, sinon on lui attribut **movementDirectionY**, qui est la valeur actuelle en **Y** du joueur qu'on récupère juste avant. Ensuite, on fait une deuxième vérification si le joueur est sur le sol. Si ce n'est pas le cas, on attribut à **_moveDirection.y** la **gravité multipliée par deltaTime**, ce qui signifie que l'on va appliquer le calcul une fois par seconde.
Pour clarifier cela, voici un exemple. Au 1er tour de la fonction **Update()**, le **_moveDirection** est un vecteur nul, et comme le joueur est sur le sol au départ, ce dernier ne changera point. Cependant, si le joueur effectue un saut, **_moveDirection.y** prend la valeur de **jumpPower** et ainsi le joueur est propulsé vers le haut. Au prochain tour de **Update()**, comme le joueur est dans l'air, on va

rentrer dans la 2e condition **if**, donc **_moveDirection.y** va prendre une valeur négative avec le calcul effectué. De ce fait, au prochain tour de la fonction **Update()**, **movementDirectionY** va prendre cette valeur négative, donc en rentrant dans le 1er **else**, **_moveDirection.y** prend cette valeur et donc le joueur va descendre.
Il faut comprendre qu'encore au prochain tour, cette variable va toujours rester dans le négatif, sauf si le joueur rappuie sur le bouton **Space** : dans ce cas, **_moveDirection.y** reprend la valeur de **jumpPower**, et ainsi de suite. Voici l'implémentation de ce système de saut :

```
#region Handles Jumping

//Checks if the button for jumping is pressed, if the player can move, and if he is on the ground
if (Input.GetButton("Jump") && canMove && _characterController.isGrounded)
{
    // player can jump
    _moveDirection.y = jumpPower;
}
else
{
    //the player isn't jumping or already not on the ground
    _moveDirection.y = movementDirectionY;
}

// if player is not on the ground, make him go down with gravity
if (!_characterController.isGrounded)
{
    _moveDirection.y -= gravity * Time.deltaTime;
}
```

A ce stade, on peut appliquer le mouvement effectué par le joueur :

```
//apply calculated movement vector above
_characterController.Move( motion: _moveDirection * Time.deltaTime);
```

Il nous reste plus qu'à gérer le système de **rotation**.
Il est important de préciser que la rotation va s'effectuer dans 2 différentes parties : le **corps** du **Player** horizontalement et la **Camera** du **Player** verticalement (en effet on ne veut pas que le corps du joueur se penche lorsqu'on regarde vers le haut ou vers le bas).
Pour cela, on utilise la variable **_rotationX** pour la rotation de la **Camera**, et pour la rotation du **corps**, on utilisera directement la méthode **.rotation** sur le **transform** (component qui contient la position, la rotation et la taille de l'objet) du **Player**. A titre indicatif, on nomme _rotation**X** car la rotation s'effectue **AUTOUR** de l'**axe X** (donc rotation verticale).
On soustrait d'abord à **_rotationX** la valeur du **mouvement de la souris** dans l'**axe Y** (récupérée encore une fois avec le **Input.GetAxis**) multipliée par **lookSpeed**, qu'on a déclaré au début, c'est en effet la sensibilité de la souris.

Une fois cette valeur obtenue, on doit la **bloquer** dans un certain **intervalle d'angles** car on ne veut pas que le joueur puisse tourner sa tête de 360 degré verticalement.
Pour cela, on utilise la méthode **Mathf.Clamp**, qui prend 3 arguments, en 1er lieu la valeur à bloquer, les 2 derniers correspondent a chaque borne de l'intervalle.
Il nous reste plus qu'à **appliquer** la **rotation** calculée sur la **Camera** du joueur.
On utilise le setter de **.localRotation** et non **.rotation** car sinon la **Camera n'effectue pas de rotation indépendamment de son objet parent** qui est ici notre joueur.
Cependant, on peut directement faire **.rotation** sur notre joueur étant l'objet parent. C'est d'ailleurs la raison pour laquelle on n'a pas besoin de configurer la **rotation horizontale** de la **Camera**, car si le **Player** tourne, la **Camera** tourne également.

```
#region Handles Rotation

if (canMove)
{
    //Calculate rotation around the X axis of the camera and limiting Vertical Rotation
    _rotationX += -Input.GetAxis("Mouse Y") * lookSpeed;
    _rotationX = Mathf.Clamp( value: _rotationX,  min: -lookXLimit,  max: lookXLimit);

    //Apply the calculated rotation around the X axis to the camera. We only want the camera to move vertically. Not the entire player
    playerCamera.transform.localRotation = Quaternion.Euler(_rotationX,  y: 0, z: 0);

    //Apply directly the calculated rotation around the Y axis
    //Player is only moving horizontally
    transform.rotation *= Quaternion.Euler( x: 0,  y: Input.GetAxis("Mouse X") * lookSpeed,  z: 0);
}

#endregion
```

*Précision : on utilise **Quaternion.Euler** qui est une méthode qui convertit un vecteur en une rotation.*

- Début du système de multijoueur

Cette partie concerne le **multijoueur**, mais pour l'instant sans parler de **réseau public** et de **serveur public**.
Je ne fais qu'utiliser un **Asset** (une extension) que l'on peut utiliser sur **Unity** pour mettre en place le multijoueur, et ce dernier s'appelle **Mirror**.
Pour commencer, il faut ajouter le component **Network Identity** de **Mirror** à notre **Player**.
Ensuite, il faut mettre en place le **Network Manager** sur notre scène qui va permettre la configuration de notre système de multijoueur via **Mirror**.
Dans ce **Network Manager**, on rajoute les components **Kcp Transport** et **Network Manager HUD**.

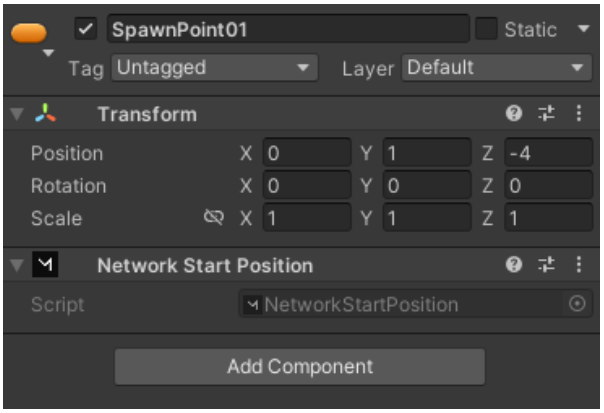
Kcp Transport, comme l'indique son nom, va nous permettre de transmettre les informations nécessaires au sein de notre réseau local.

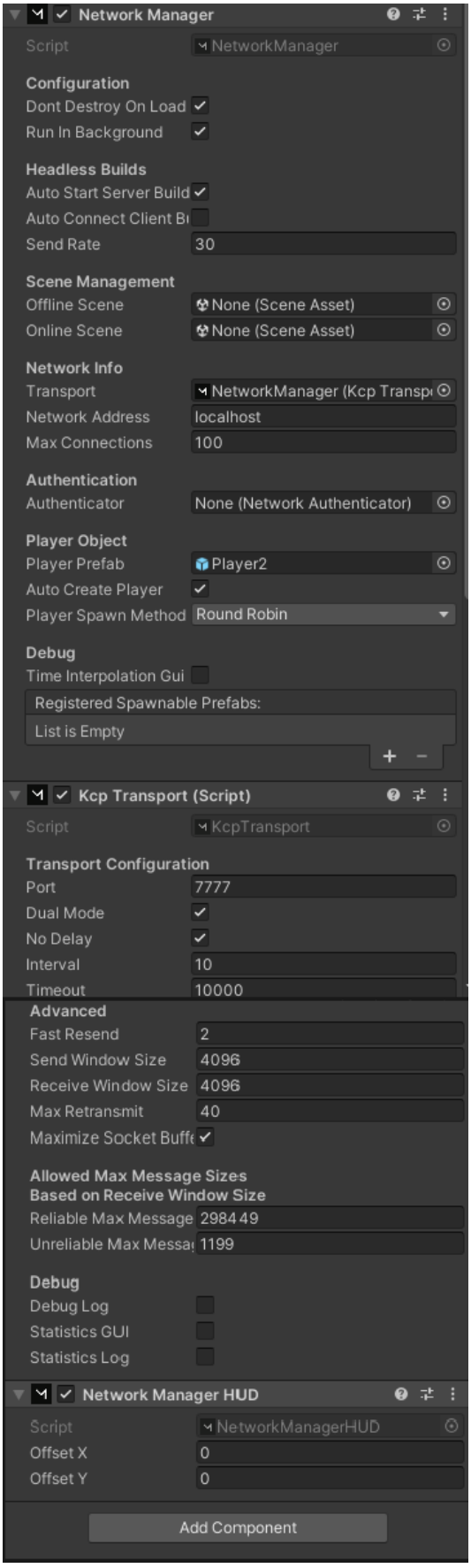
Network Manager HUD va nous permettre d'afficher l'interface suivant en haut à gauche de notre écran :

Grâce au bouton **Host**, on peut lancer un **localhost** et accueillir les autres **instances locales** (pour l'instant).

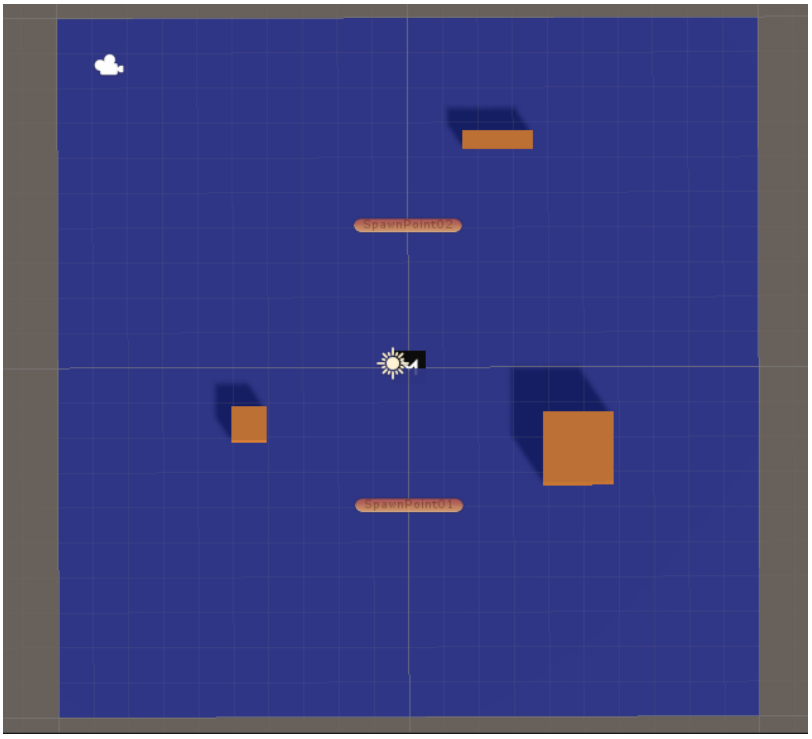
Il est imporant de préciser qu'il faut bien mettre dans l'option **Player Prefab** le **prefab** de notre **Player.**
(Un prefab est un objet que l'on créé et enregistr en tant qu'un nouvel objet que l'on peut réutiliser à tout moment.
A ce stade, j'ai déjà créé un prefab de **Player**.)

Ensuite, j'ai placé 2 points d'apparition des joueurs, ce qui se fait facilement par l'ajout du component **Network Start Position** a chaque objet qui fait office de **SpawnPoint** :





D'un point de vue du haut, notre scène ressemble à l'image suivant :



A ce stade, si on fait un **Build**, on peut rejoindre la partie locale lancée dans Unity depuis l'instance du Build. Cependant, **deux problèmes** s'imposent : le 1er est le fait que le **hôte a le contrôle de tous les joueurs présents** sur scène, le 2e est le fait que les **actions ne sont pas synchronisées sur toutes les instances**, c'est-à-dire que chacun joue un jeu différent de celui des autres.

Pour régler le 1er problème, il nous faut un nouveau script qui devra désactiver tous les components qui permettent le contrôle des autres joueurs.
On crée le script **Player Setup**. On utilise un array **Behaviour[]** qui va contenir tous les components. Ensuite, je crée une méthode **DisableComponents()** qui désactive tous les components de cet array. On l'appellera dans une condition vérifiant que le joueur en question est bien un autre joueur que le nôtre. On met également en place une méthode **AssignRemotePlayer()** qui va directement assigner un layer (une catégorie) du nom « RemotePlayer », également appelé en meme temps que le **DisableComponents().**

On configure également la **caméra de la scène** lorsque aucun joueur n'est sur scène directement sur Unity (quand aucune partie n'est lancée). Et on pourra la désactiver lorsque notre joueur local se connecte à une partie.

Voici l'implémentation du début de notre script :

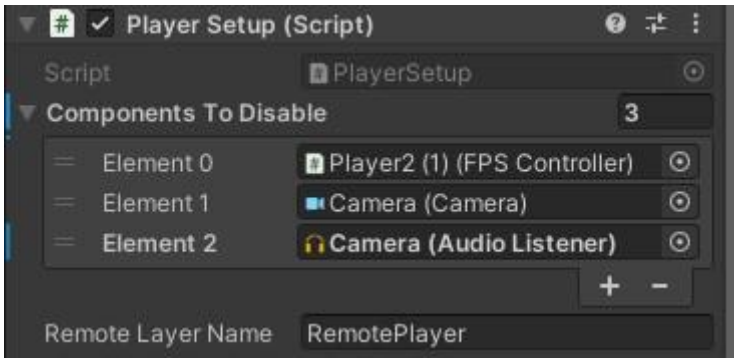
```
using UnityEngine;
using Mirror;

2 asset usages  Chiyoung *
public class PlayerSetup : NetworkBehaviour
{
    [SerializeField]
    private Behaviour[] componentsToDisable;  Serializable

    [SerializeField] private string remoteLayerName = "RemotePlayer";  Unchanged

    private Camera _sceneCamera;
    Event function  Chiyoung
    private void Start()
    {
        // Disable components if it is not my player so we don't have the control of other players.
        if (!isLocalPlayer)
        {
            //if is not my player, disable all components of other player and assign him the layer "RemotePlayer"
            //so it can send information when only another player is shot
            DisableComponents();
            AssignRemotePlayer();
        }
        else
        {
            _sceneCamera = Camera.main;

            if (_sceneCamera != null)
            {
                _sceneCamera.gameObject.SetActive(false);
            }
        }
```

*Indication : le **[SerializeField]** permet d'afficher la variable sur Unity, afin de pouvoir modifier sa valeur (si c'est une variable, peut modifier la taille, le contenu si c'est un array) directement sur l'éditeur Unity.*

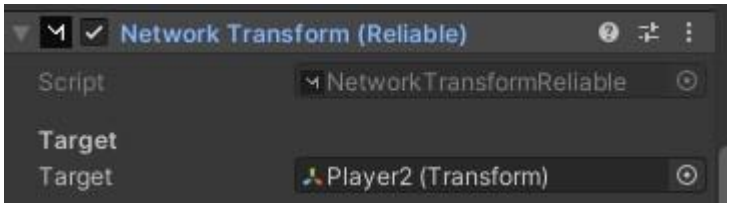
```
1 usage  Chiyoung
private void AssignRemotePlayer()
{
    //Assign layer "RemotePlayer"
    gameObject.layer = LayerMask.NameToLayer(remoteLayerName);
}


1 usage  Chiyoung *
private void DisableComponents()
{
    foreach (var t Behaviour in componentsToDisable)
    {
        t.enabled = false;
    }
}
```

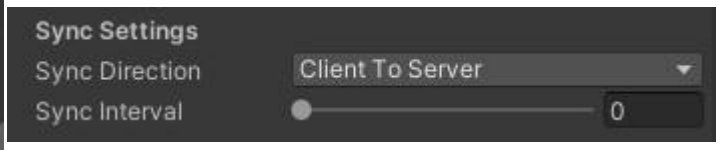
Une fois sur Unity, on ajoute notre script **Player Setup** à notre Prefab de **Player** et on remplit bien l'array **componentsToDisable** avec le script **FPSController** qui permet le contrôle du mouvement des joueurs, la **caméra** du joueur, et l'**audio listener** qui est attaché à la caméra, car

sinon on entendra ce qu'entendent les autres joueurs.



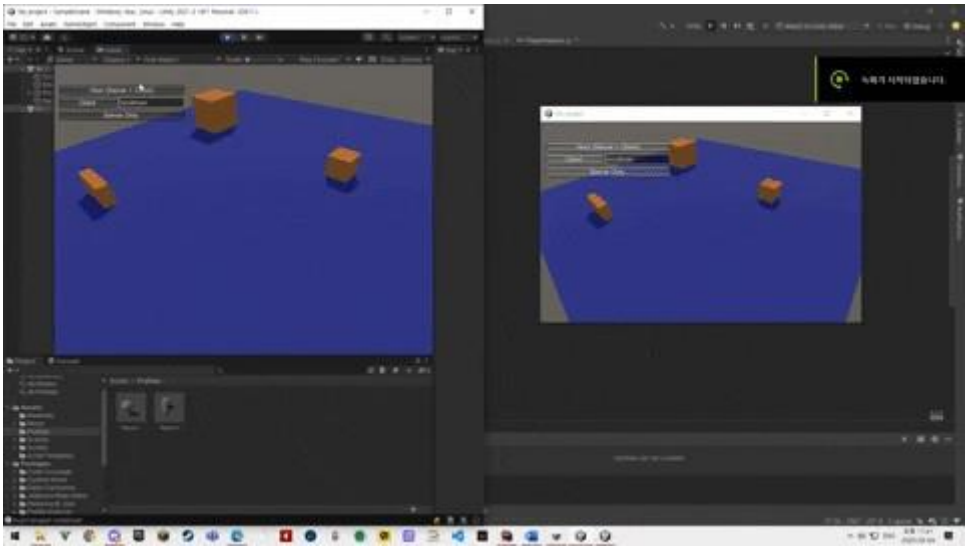
A ce stade, le 1er problème est résolu, mais le 2e persiste.
Pour la synchronisation des instances, il est plutôt simple puisque **Mirror** nous fournit les components dédiés à cela.
Il s'agit d'ajouter au **Player** deux components **Network Transform**, un pour la synchronisation du **Player** lui-même et un autre pour sa **Camera** (oui, encore une fois ils sont partiellement indépendants).





Il faut bien penser pour le 1er NT (Network Transform) à mettre dans **Target** le **Player**, et pour le 2e NT, la **Camera.** Il faut également penser à préciser dans **Sync Direction** que la synchronisation se fait du **Client vers le Server**.

Ainsi, lorsqu'on refait un **Build Run**, on peut bien rejoindre notre hôte sur Unity depuis une autre instance, et d'observer les mouvements effectués sur une instance depuis une autre tout en étant synchronisés. On observe également que chaque instance a uniquement le contrôle de son propre joueur :



Récits de réalisation :

Chiyoung

Je suis plutôt satisfait de mon travail et de celui de mes collègues pour l'instant, et j'ai hâte de poursuivre la suite du projet afin d'en faire un vrai FPS fonctionnel.

Cependant, il m'a fallu beaucoup de temps et de recherches pour comprendre toutes les nouvelles fonctionnalités, notamment tout ce qui est en rapport avec Unity.

Je pense que c'est dommage qu'on n'ait pas pu mieux s'organiser et surtout le fait qu'on n'a pas eu le temps pour régler nos problèmes d'images sur Overleaf… On n'arrivait pas à placer les images où on voulait et on considère cela comme un point crucial pour une meilleure lisibilité et compréhension de notre travail. Pour le prochain rapport, nous rédigerons sans exception nos rapports en Latex.

Ryan

En tant que chef de projet, il était difficile d'organiser au début des séances productives avec les TPs, les examens et les MiMos. Par conséquent, la majorité de ces séances était concentrée pendant les vacances, période où certains des membres du groupe n'étaient pas disponibles de manière permanente. Nous avons donc pris un léger retard. Néanmoins, après la mise en place d'un journal de bord, le partage de liens vidéos pour nous initier, j'ai constaté que nous étions plus motivés. En ce qui concerne Unity, j'ai eu du mal à comprendre certaines approches montrées dans les vidéos. Pour le site web, il était plutôt difficile de réaliser un site web en partant de zéro avec les innombrables balises html et les différentes propriétés css. Pour l'interface/lanceur, j'étais un peu déçu que l'utilisation de C# se résumait pour l'instant à lancer des applications. En revanche, il était plutôt aisé de se familiariser avec Visual Studio pour illustrer le lanceur en effectuant des recherches. Je suis en partie satisfait du travail fourni car il répond à nos objectifs de première soutenance et nous avons pris une légère avance par rapport à nos prévisions. J'en ressors tout de même frustré en tant que chef de projet de ne pas avoir pu proposer aux membres du groupe une meilleure organisation en termes de disponibilité. Pour la prochaine soutenance, je compte mettre en place un meilleur système, en utilisant notamment le logiciel Trello afin de satisfaire tout le groupe.

Aymeric

Étant donné que nous avions des TPs ou des évaluations, nous avons mis un petit peu de temps avant de commencer réellement à avancer sur le projet. Néanmoins je trouve que l'avancé actuel du projet est plus que correct. Nous avons un site web fonctionnel, un repo git assez bien organisé et plein d'idées à exploiter. Cependant, étant rentré en campagne avec ma famille pendant la semaine de vacances, j'ai rencontré d'assez grands problèmes, à la fois de connexion et d'organisation, d'autant plus qu'au lycée je n'avais pas des spécialités orientées vers l'informatique donc je dois apprendre presque tout de zéro. Heureusement je me rends compte que beaucoup de personnes sont prêtes à partager leur connaissance pour le bien de tous ce qui facilite mon apprentissage.

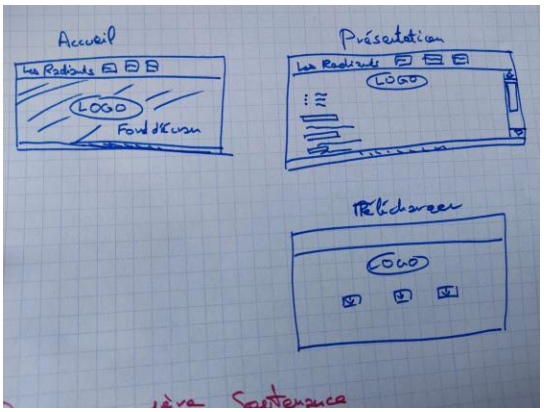
Cela ne m'empêche pas de prendre beaucoup de plaisir à travailler en groupe autour d'un projet commun dont j'espère que nous le mènerons à bien.

Conclusion :

Le bilan d'avancement du projet est plutôt satisfaisant, nous avons eu de légers retards mais nous nous sommes rattrapés. Étant des débutants en la matière, il était difficile d'obtenir des résultats satisfaisants au commencement du projet, principalement caractérisé par un apprentissage des bases des langages et la maîtrise des logiciels/éditeurs. On espère alors réaliser plus tard de meilleurs travaux qui reflètent réellement notre créativité. Pour la prochaine soutenance (intermédiaire), il est prévu que le jeu ait un système de tir, un environnement, que le lanceur soit fonctionnel (démarrage du jeu avec des animations, ...), que le site Web soit achevé avec une illustration d'images, de démonstrations, un début de mise en place d'une intelligence artificielle.

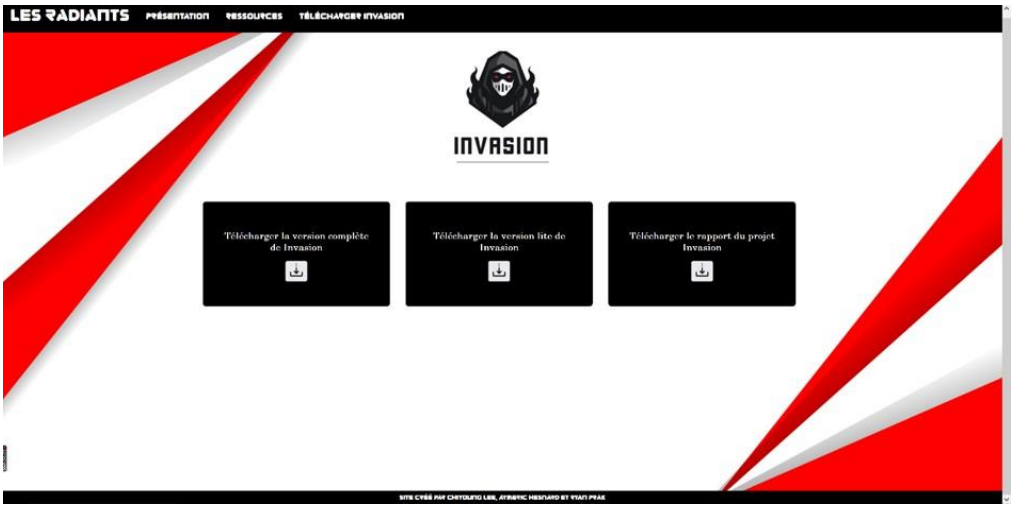
Annexe :

(Fig.1)

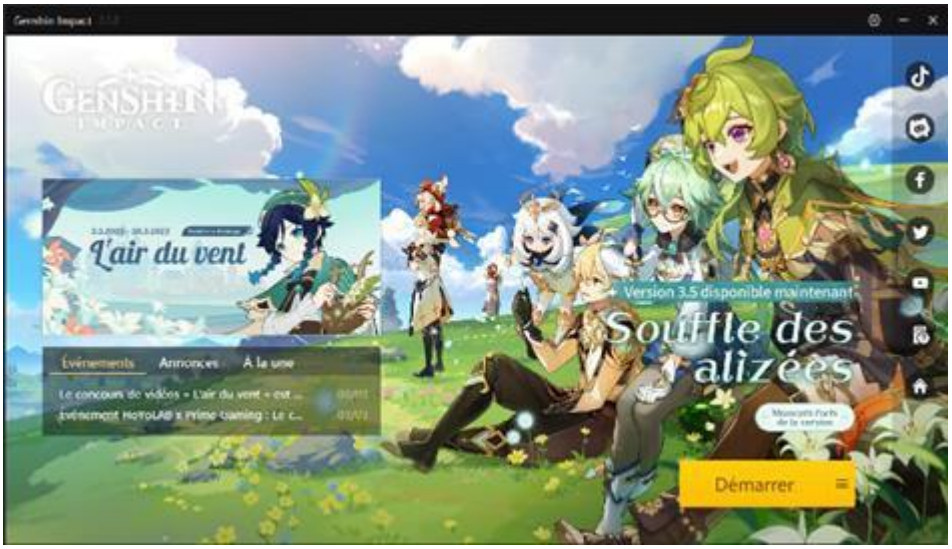


(Fig.2)

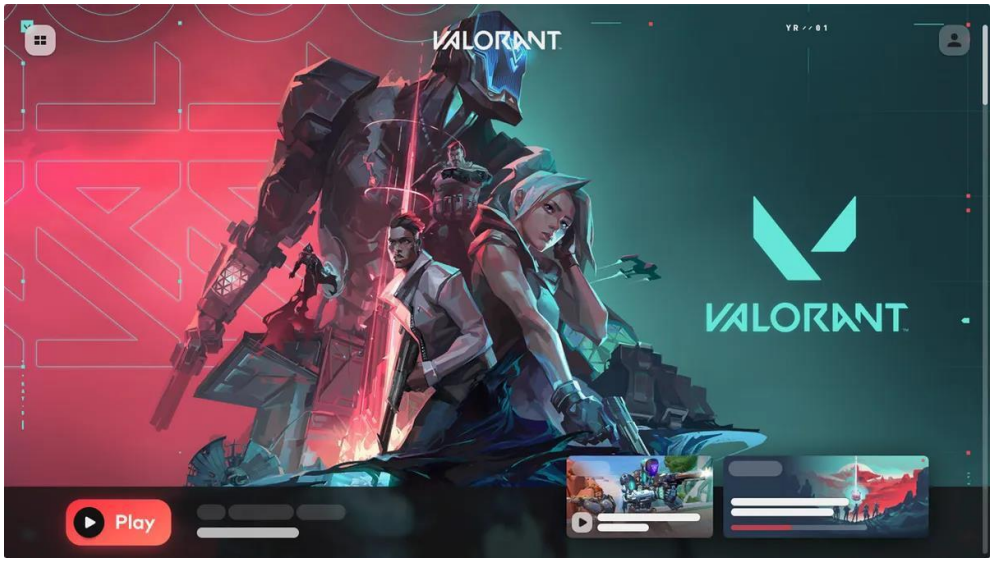
(Fig.3)



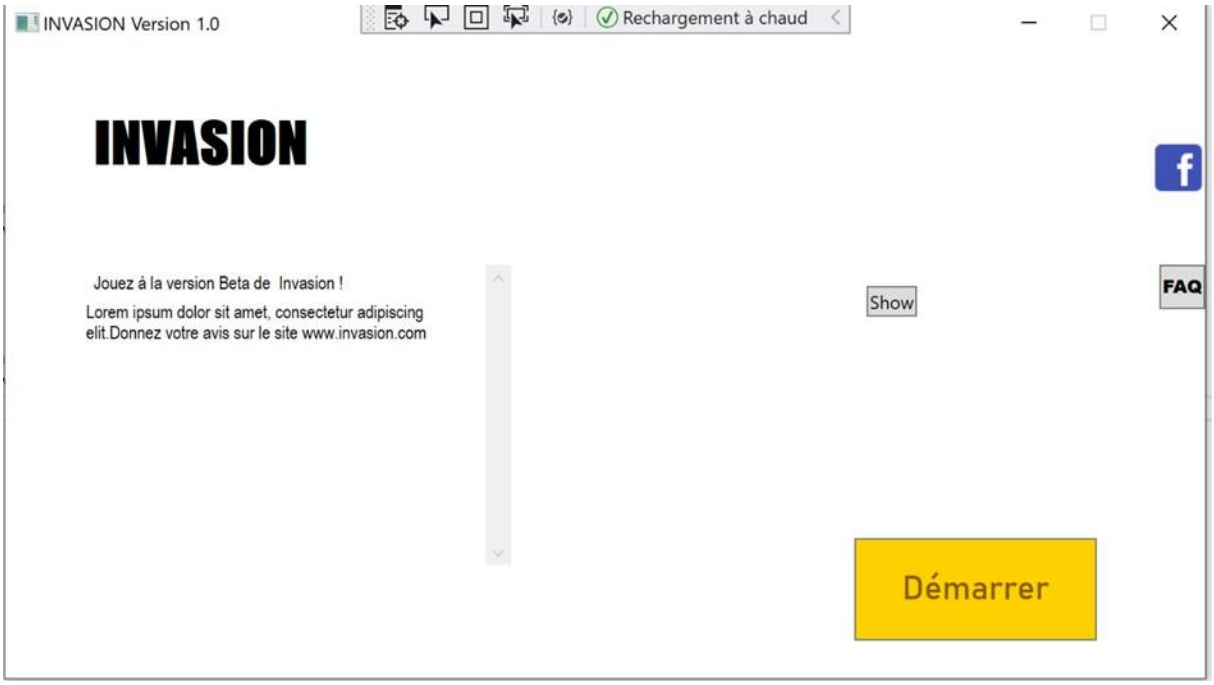
(Fig.4)



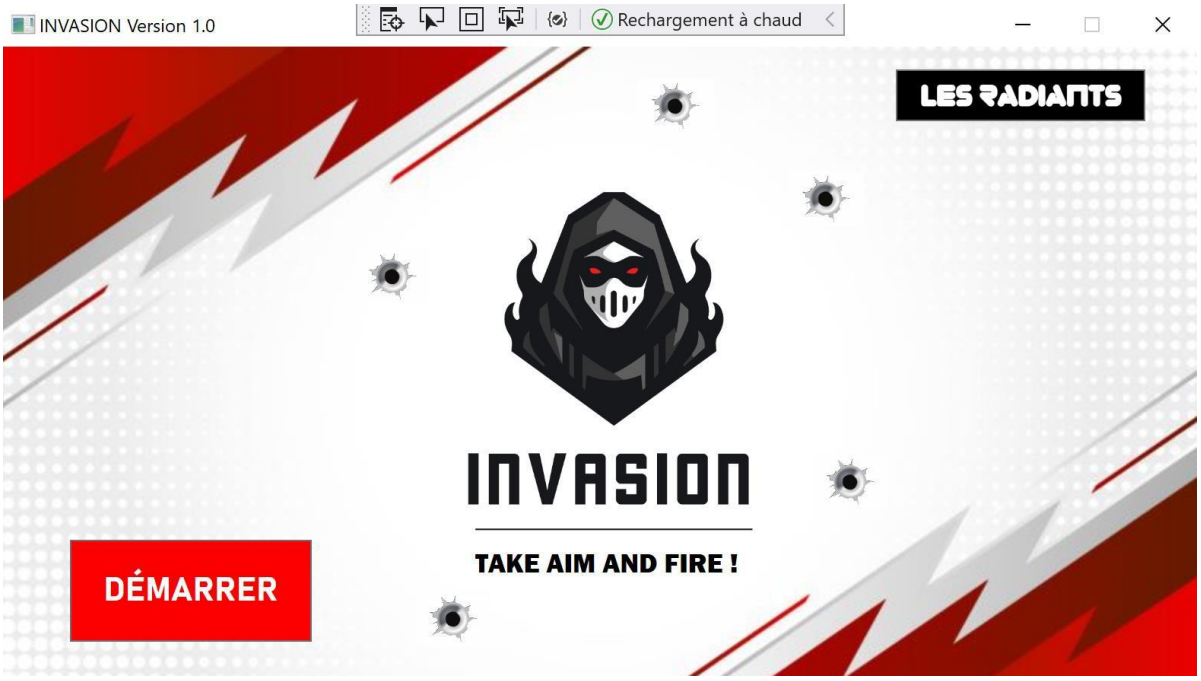
(Fig.5)



(Fig.6)


INVASION Version 1.0
Rechargement à chaud
INVASION
Jouez à la version Beta de Invasion !
Lorem ipsum dolor sit amet, consectetur adipiscing elit.Donnez votre avis sur le site www.invasion.com
Show
FAQ
Démarrer

(Fig.7)


INVASION Version 1.0
Rechargement à chaud
LES RADIANTS
INVASION
TAKE AIM AND FIRE !
DÉMARRER

(Fig.8)