Brain Fighter

(Documentation)

GRAILLOT Youé

Sommaire

Description du projet

Brain Fighter est une version améliorée d'un prototype crée antérieurement par HD43D Studios et Neurotechx MTL dont voici un aperçu :

Illustration 1: Ancien Brain Fighter

Celui comporte une scène de combat classique avec un opposant de chaque côté de l'écran. Chaque joueur est représenté par un cerveau et dispose de deux actions possibles : lever le bouclier ou attaquer. Il dispose pour cela d'un casque Muse[1] qui va lire son activité cérébrale. Cette information est ensuite traduite en deux valeurs booléennes représentant la concentration et la relaxation :

- 1 et 0 pour concentré
- 0 et 1 pour relaxé
- 1 et 1 pour concentré et relaxé

[1] Cf. **Error! Reference source not found.**

Cette tâche est réalisée par un programme Python pour que l'information soit ensuite transmise à Unity qui va finalement indiquer aux avatars d'attaquer ou de se défendre (lever le bouclier). L'Illustration 2 : Pipeline de l'ancien Brain Fighter décrit plus en détail le pipeline mis en place :

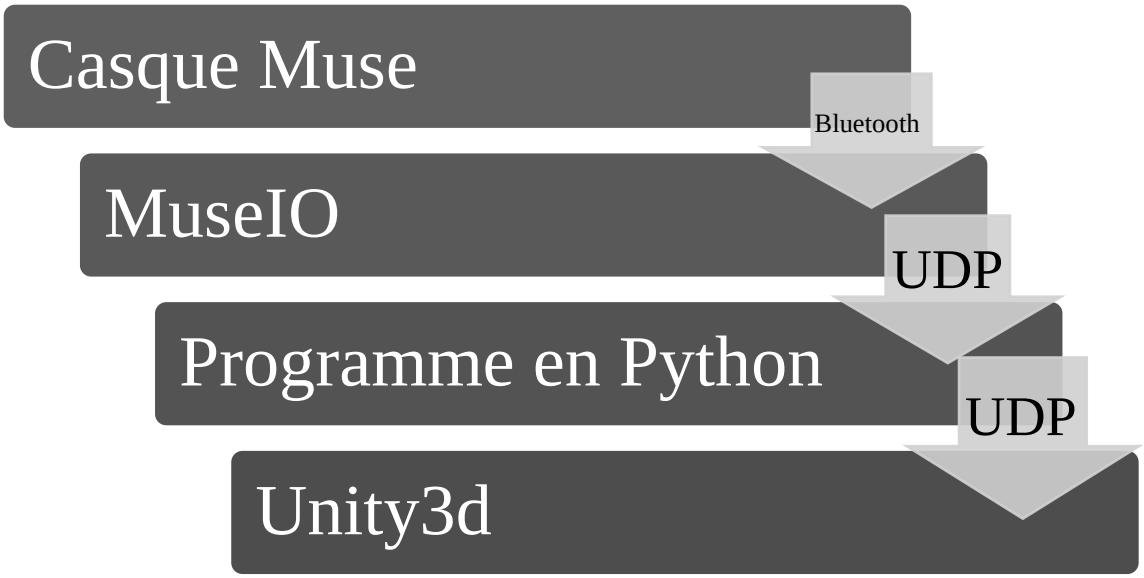


Illustration 2 : Pipeline de l'ancien Brain Fighter

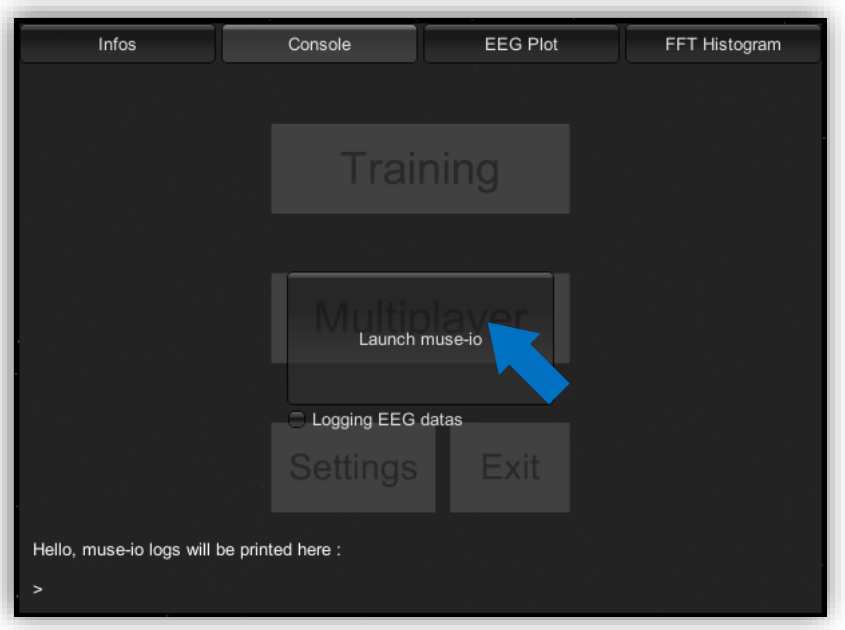
Ce pipeline a été revu dans la version finale en intégrant les algorithmes python directement dans Unity, ce qui permet de communiquer directement entre MuseIO et Unity.

L'apport de la réalité virtuelle avec le casque Oculus permet de changer le gameplay et d'ajouter une autre prise de décision. Il faut désormais viser une cible du regard si l'on veut espérer lui infliger des dégâts. Le bouclier peut quant à lui parer les attaques provenant de n'importe quelle direction.

Connecter le Muse

1. Associer le casque Muse par Bluetooth, pour cela exercer une longue pression sur le bouton power.

2. Dans le jeu, afficher la console avec la touche [Tab] et dans la section « Console » démarrer MuseIO :



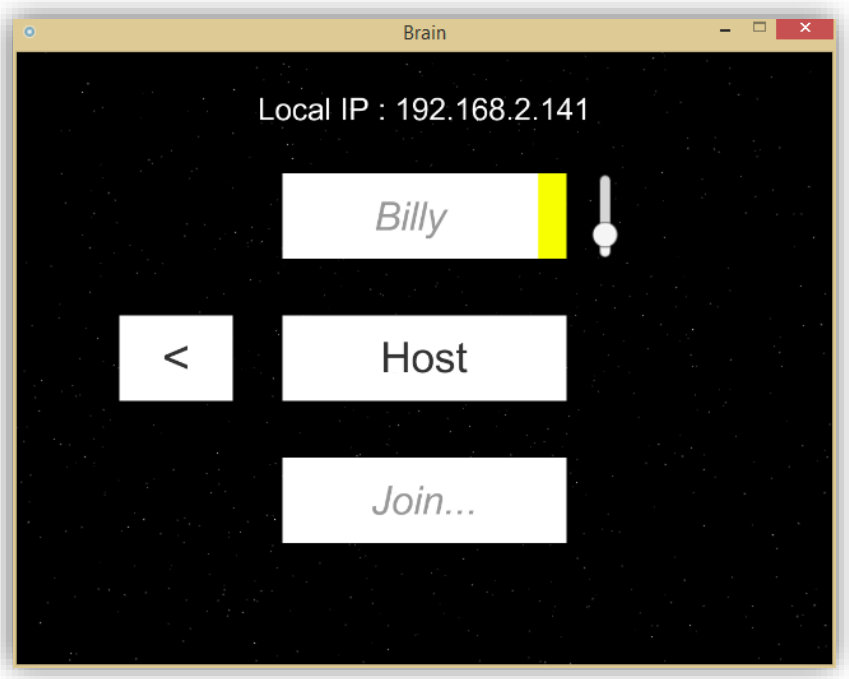
   Patienter, MuseIO va tenter de se connecter à un casque Muse associé avec l'appareil.
   Une fois connecté vous pouvez voir l'état des capteurs qui doivent tous être bien connectés (en vert) dans la section « Infos ».

3. Appuyer de nouveau sur [Tab] pour fermer la console.
   Il est possible de rouvrir la console à tout moment pour consulter l'état des capteurs, le résultat des algorithmes et les graphiques EEG/FFT.

Lancer une partie multijoueur

1. Dans le menu principal du jeu, sélectionner « Multiplayer ».

2. Vous avez le choix entre devenir l'hôte d'une partie en choisissant « HOST ». Pour rejoindre une partie en tant que client, rentrer l'IP de l'hôte dans le champ « Join… ». L'adresse IP des ordinateurs est affichée tout en haut de l'écran :



Le premier champ permet de définir un nom et sa couleur qui seront affichés au-dessus de votre avatar et visible par les autres joueurs.

3. Si aucun casque Muse n'est connecté, il est possible de jouer à la manette à l'aide des deux gâchettes arrières ou au clicque souri.
La touche [Echap] (ou [Esc]) permet de mettre fin à la session et de revenir à l'écran d'accueil.

Comment fonctionne :

Le projet Unity

Le projet est divisé en 4 scènes :

- 0 - Init :
  - Contient uniquement l'objet {Settings} contenant plusieurs enfants. Cet objet ne se détruit jamais (avec la méthode DontDestroyOnLoad()) et on accède à 0 - Init qu'une seule fois au démarrage pour ne pas dupliquer {Settings}. On quitte la scène une fois chargé.
- 1 - Menu :
  - Scène vide servant uniquement à afficher le menu.
- 2 - Training :
  - Scène de l'entrainement contenant le décor et {Offline Player}.
- 3 - Multi :
  - Scène de l'entrainement contenant le décor. Les objets joueurs sont instanciés à la position des {Spawn Points}.

{Settings} : Contient le {Network Manager} qui permet la mise en réseau du jeu mais c'est par l'appel des méthodes Join() et Host() du {Level Manager} que l'on se connecte à une partie.

{Brain} : Regroupe les mécaniques de gameplay. C'est cette classe dérivée de {NetworkBehaviour} qui permet de notifier les autres instances du jeu des actions utilisateur avec les « Command », méthode spéciale qui permet de communiquer avec les clients. Elle gère à la fois les attributs des joueurs (points de vie, état du bouclier, charge de l'attaque) et ses actions (tirer, se défendre, subir des dégâts).

{PlayerController} : Sa fonction principale est de permettre le contrôle à la manette/souris lorsqu'aucun casque de VR n'est détecté.

{IA} : Contrôleur des Bots de l'entrainement. L'EditorScript {IA_Controller} permet l'affichage de deux boutons « Attack » et « Defend » dans l'inspecteur pour forcer ces actions manuellement.

Le prefab OnlinePlayer contient divers objets enfant dont la Caméra, des scripts de contrôles et des interfaces utilisateur qui sont détruits à l'instanciation si la machine actuelle n'a pas l'autorité sur l'objet (si OnlinePlayer est une instance d'un autre client).

La connexion avec le Muse

Un objet de la scène nommé « MUSE_INTERFACE » regroupe 5 scripts C# :

- MUSE_IO_READER :
  - Exécute MuseIO et écoute son flux sortant pour mettre à jour l'affichage de la console
- MUSE_DATAS :
  - Toolbox servant à stocker les informations du Muse, les informations sont donc accessibles publiquement avec l'instruction MUSE_DATAS.Instance.X
- MUSE_LOGGING :
  - Sauvegarde les données EEG dans un fichier CSV
- MUSE_DEBUG :
  - Permet d'afficher l'interface de debug
  - Il va ajouter un script EEGVisualizer et trois FFTVisualizer qui servent à dessiner les graphiques EEG/FFT
- Algorithms :
  - Contient l'algorithme de l'ancien BrainFighter

La console permet d'appeler la méthode Launch() de MUSE_IO_READER qui démarre MuseIO. Le code suivant configure l'exécution de MuseIO pour cacher la console et afficher les informations de la console directement dans Unity :

```
void Start()
{
    // Init muse-io

    MUSE_CMD = new Process();

    MUSE_CMD.StartInfo.FileName = "muse-io";
    MUSE_CMD.StartInfo.Arguments = "--osc osc.udp://127.0.0.1:" + Port + " --osc-timestamp";
    MUSE_CMD.StartInfo.RedirectStandardOutput = true;
    MUSE_CMD.StartInfo.UseShellExecute = false;
    MUSE_CMD.StartInfo.CreateNoWindow = true;

    // Inits

    DebugConsole = GetComponent<MUSE_DEBUG>();
}
```

Une fois MuseIO lancé, la méthode CmdOutputHandler() li le flux sortant et détecte la connexion du casque par reconnaissance de la chaine « Connected. » pour lancer l'écoute du port UDP associé et recevoir les données du casque :

```
void CmdOutputHandler(object sendingProcess, DataReceivedEventArgs outLine)
{
    if (!string.IsNullOrEmpty(outLine.Data))
    {
        if(outLine.Data.Contains("Connected."))
            listener = new UDPListener(Port, Receive);

        DebugConsole.WriteLine(outLine.Data);
    }
}
```

La méthode va ainsi traiter les paquets UDP et les interpréter en paquet OSC grâce à la librairie SharpOSC :

```
void Receive(OscPacket packet)
{
    OscMessage messageReceived = (OscMessage)packet;
    MUSE_DATAS.Instance.DataAttribution(messageReceived.Address, messageReceived.Arguments.ToArray());
}
```

Finalement, les données sont stockées dans MUSE_DATAS qui possède 3 attributs de type UnityEvent :

- Raw_fft_Callback
    - Appel Algorithms. Muse_fft_alpha_all_median() à la réception d'un message dont l'adresse est : « /muse/elements/raw_fftX »
- ConcentrationCallback
    - Appel une méthode à la réception d'un message dont l'adresse est : « /muse/elements/experimental/concentration »
- MellowCallback
    - Appel une méthode à la réception d'un message dont l'adresse est : « /muse/elements/experimental/mellow »

Ces UnityEvent permettent ainsi de communiquer avec le reste de l'application.