Combat de robot immersif par utilisation de VR et motion plateform

OKEILI Adam*

Institut Supérieur Industriel de Bruxelles
53981@etu.he2b.be

I. Mots-clefs

— Motion Plateform : est un simulateur de mouvement permettant de créer l'illusion que l'utilisateur est dans un environnement en réel mouvement.
— VR : Technologie permettant à l'aide d'un casque VR de plonger en immersion l'utilisateur dans un univers virtuelle.
— VR sickness : Trouble générer par le faussé entre ce que l'utilisateur voit dans à travers son casque VR et ses autres sens, causant des pertes d'équilibres et potentiellement un état nauséeux chez l'utilisateur.
— Collider : Est un élément en 2D ou 3D fourni par Unity, permettant de générer une zone de détection pouvant enclencher divers mécanismes.
— Input : Ensemble des interactions (ex : bouger les sticks, appuyer sur un bouton, etc ...) entre l'utilisateur et le contrôleur (Joystick).

Résumé

Ce programme a été réalisé dans le cadre du Bureau d'études du Master 1 de la section informatique. Le projet a pour objectif de permettre à l'utilisateur de plonger d'un jeu de combat de robot à la première personne. Le projet utilise différentes technologies telles que le casque VR, la motion plateform et des joysticks, afin de renforcer l'immersion de l'utilisateur au sein de l'univers.

II. Introduction

Le projet peut être divisé en 4 couches :

1. Couche VR : Couche qui s'occupe de gérer la perception de l'environnement virtuel par l'utilisateur.
2. Couche Mouvement : Couche qui raccorde les informations de l'univers 3D à la motion plateform, permettant à celle-ci de se mouvoir en fonction des informations reçus.
3. Couche Input : Couche qui sert de contrôleur, permettant à l'utilisateur d'interagir avec le milieu virtuel à l'aide des Joysticks.
4. Couche Logiciel : Partie s'occupant de toute la logique du système (système de combat, modification de l'interface graphique, etc.. . .) et coordonne les tâches des différentes couches.

III. Matériel & Méthodes

I. Outils utilisés

En ce qui concerne les outils utilisés pour la réalisation de ce projet, ceux-ci sont au nombre de 6. À savoir l'environnement de développement Visual Studio, le Framework Unity, les

*Rapport réalisé dans le cadre du PBE

Joysticks, le casque VR, la motion platform et l'un des programmes démo de la motion plateform proposé dans les ressources.

Le choix d'utiliser l'environnement de développement Visual Studio est lié au fait qu'il est celui proposé par défaut par Unity.

Unity est l'un des Framework de jeux vidéo le plus utilisé. Son utilisation au sein de ce projet à été imposé.

L'utilisation des joysticks a été choisi, afin de renforcer l'idée de pilote de robot.

Le casque VR permet de donnée une vue libre de l'intérieur du cockpit, ajoutant encore une part de réalisme.

La motion platform permet tout comme le casque VR, d'accroître l'immersion de l'utilisateur dans le jeu, mais également de réduire les risques de "VR sickness". La VR sickness est un trouble temporaire engendré par l'utilisation de casque VR, celle-ci est lié au fait que certains environnements VR sont mal agencés, créant un gap entre ce que l'utilisateur voit et ce qu'il ressent, déstabilisant ainsi son cerveau, générant après utilisation l'apparition d'une perte d'équilibre et/ou de nausée chez l'utilisateur. Par conséquent, L'utilisation de la motion platform peut permettre de réduire ce gap entre la réalité et l'univers fictif vu par l'utilisateur.

II. Techniques utilisées

Les différentes couches du projet présentent quelque technique qui ont contribué au résultat.

Donc tout d'abord, pour la couche gérant la motion plateforme, seulement 3 paramètres seront utilisés pour contrôler la plateforme. Ces 3 paramètres permettent d'élever légèrement le siège, l'incliner en avant et en arrière, et de déplacer le siège vers l'avant et vers l'arrière.

```
// Current platform's heave in game
private float m_heave = 0;

// Current platform's pitch in game
private float m_pitch = 0;

// Current platform's roll in game
private float m_roll = 0;
```

Figure 1 – *Paramètre utilisé pour contrôler la chaise*

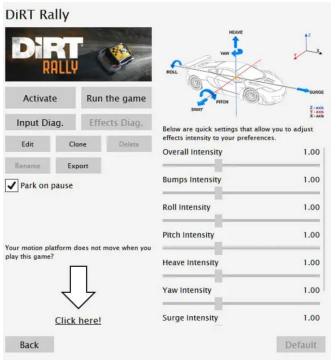

Figure 2 – *Représentation des paramètres de la plateform*

Avant de commencer à envoyer les informations à la plateform, il faut tout d'abord créer une session qui indiquera à la chaise que des informations lui seront envoyé, permettant de faire bouger la plateform.

```
void Start()
{
    instance = this;
    // Load ForceSeatMI library from ForceSeatPM installation directory
    m_fsmi = new ForceSeatMI();

    if (m_fsmi.IsLoaded())
    {

        // Prepare data structure by clearing it and setting correct size
        m_platformPosition.mask = 0;
        m_platformPosition.structSize = (byte)Marshal.SizeOf(m_platformPosition);

        m_platformPosition.state = FSMI_State.NO_PAUSE;

        // Set fields that can be changed by demo application
        m_platformPosition.mask = FSMI_POS_BIT.STATE | FSMI_POS_BIT.POSITION;

        m_fsmi.BeginMotionControl();

        SendDataToPlatform();
    }
    else
    {
        Debug.LogError("ForceSeatMI library has not been found!Please install ForceSeatPM.");
    }
}
```

Figure 3 – *Instanciation d'une session de motion plateform*

Les informations seront envoyées par la méthode suivante :

```
void SendDataToPlatform()
{
    // Convert parameters to logical units
    m_platformPosition.state = FSMI_State.NO_PAUSE;
    m_platformPosition.roll = Mathf.Deg2Rad * m_roll;
    m_platformPosition.pitch = -Mathf.Deg2Rad * m_pitch;
    m_platformPosition.heave = m_heave * 100;

    Debug.Log("roll: " + m_platformPosition.roll + ", pitch: " + m_platformPosition.pitch + "heav : " + m_platformPosition.heave);

    // Send data to platform
    m_fsmi.SendTopTablePosPhy(ref m_platformPosition);
}
```

Figure 4 – *Méthode permettant d'envoyer les informations à la chaise*

La classe SensorManager qui est la classe permettant de gérer la couche mouvement du projet, a été mis en place en en utilisant le design pattern singleton, permettant de centraliser et détacher la gestion des mouvements

au travers du projet. Cette classe dispose de plusieurs méthodes accessibles qui modifie les paramètres de la chaise, pour simuler une situation (par ex : l'utilisation d'une secousse verticale pour simuler le pas du robot). Les autres classes appellent donc ces méthodes au moment adéquat pour s'accorder à ce que l'utilisateur voit.

Par exemple lorsque l'utilisateur appuie sur la gâchette pour tirer, la méthode va après avoir générer le projectile) appeler la méthode permettant de simuler la secousse de recul (SendShootSensation).

Pour la couche contrôle qui consiste à lier les Joysticks à des actions dans le jeu, celle-ci à été réalisé par l'utilisation du "Input System" d'Unity. L'input system est un package contenant entre autres les outils permettant de créer un système de contrôle. Le système de contrôle va permettre de lier des actions au différentes entrées (action, touche, etc.. . .) de l'utilisateur et lier ces actions à du bout de code. Par exemple, j'ai pu lier la gâchette du joystick à une nouvelle action "Fire" que j'ai créé pour permettre à l'utilisateur dans le jeu de tirer. Ou encore l'action "Forward" lié à l'inclinaison du Joystick permettant à l'utilisateur de se déplacer.

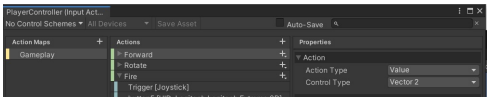

Figure 5 – *Input System*

La partie VR utilise les packages SteamVR, XRInputManagement, OpenVR. SteamVR permet en plus de fournir les outils gérant l'environnement VR, et également de simuler l'utilisation d'un casque VR, ce qui m'a permis de continuer de travailler sur mon projet sans casque VR. XRInputManagement est le package VR d'Unity, celui-ci gère le fonctionnement de la VR dans Unity et permet par exemple de définir les packages que nous allons utiliser pour notre environnement de développement. OpenVR est une API permettant l'utilisation de certain casque VR, il existe d'autre API du genre tel qu'Oculus. Le choix d'utiliser OpenVR dans le projet est uniquement lié à sa compatibilité avec le casque utilisé.

La partie logique du projet n'utilise pas de librairies particulières. Néanmoins, je pense qu'il est pertinent de préciser que le projet utilise un "NavmeshAgent" qui est un composant d'Unity permettant à l'élément le portant de se mouvoir sur un terrain statique. L'utilisation du NavmehAgent implique l'utilisation de la librairie "AI" d'Unity. En outre, pour des raisons de simplicités, le comportement du robot ennemi est géré par le "StateMachineBehavior" fourni par avec le système d'animation d'Unity. Le StateMachineBehavior est une classe qui comme son nom l'indique permet d'implémenter de manière simple le design pattern "StateMachine », qui consiste à faire en sorte qu'un élément se trouve à chaque instant dans état fini. Cet état dispose de ces propres comportement (ex : avancer) et peut se déplacer vers certain état spécifique (ex : état déplacement -> état attaque).

III. Développement

En ce qui concerne la réalisation de ce projet, celle-ci s'est réalisé de la manière suivante :

Dans un 1er temps, j'ai d'abord utilisé l'ordinateur se trouvant dans la salle où se trouve la chaise. Mon objectif était de trouver le programme permettant de faire lien entre l'ordinateur et la chaise, et des programmes servant d'exemples pour mieux appréhender la manière dont il faut envoyer les informations à la chaise pour la faire mouvoir.

Après avoir saisi grossièrement le fonctionnement de la chaise et la manière de lui faire parvenir les informations, j'ai décidé de commencer à travailler sur la logique du programme à savoir le jeu en lui-même. Pour élaborer le jeu de combat de robot, je me suis inspiré de nombreux tutoriel expliquant la manière de développer un FPS (First Person Shooter) et un jeu 3D de manière plus globale. Avant d'utiliser des robots comme modèle, j'ai d'abord utilisé des capsules faisant office de personnage. Le système de tire utilise la trajectoire du pistolet comme cible et non le centre

de la caméra. Dans un jeux vidéo de tire à la première personne, il est possible de déplacer la caméra ou la cible pour viser. Cependant, étant donné que le jeu est un jeu VR, j'ai préféré éviter d'ajouter ce mécanisme au jeu afin d'éviter de potentielle risque de "motion sickness".

À la base, lors de la réalisation des mouvements de l'IA, j'ai d'abord tenté de déplacer l'adversaire uniquement via des scripts. Malheureusement le résultat n'était pas grandiose, en effet, la transition entre les mouvements était étrange, l'adversaire était bloqué dans ces assauts répétés et tournait parfois indéfiniment autour de l'adversaire. Pour pallier à ce problème, j'ai tenté une approche un peu différente, j'ai incorporé à la place d'un script de mouvement, un composant NavMeshAgent au robot ennemi. Grâce au NavMesh, le mécanisme de poursuite s'était grandement amélioré. Néanmoins les transitions entre mouvements étaient encore maladroites. C'est alors qu'après recherche j'ai trouvé qu'Unity permettait d'incorporer des comportements au sein des différentes animations ainsi qu'un système de "StateMachine". Grâce à cela, j'ai configuré plus aisément les différents états de l'adversaire et y incorporer le code que j'avais utilisé jusqu'à maintenant pour les actions de l'adversaire.

Dans un second temps, j'ai commencé à travailler sur la partie VR du projet. Pour cette partie, je me suis attelé en premier lieu à l'emplacement de la caméra au niveau de la tête du robot pour renforcer l'idée de piloter un robot. Ensuite, j'ai mis en place l'interface fournissant les informations concernant les statiques du joueur à savoir le bouclier(santé) et le nombre de munition. La conception d'une interface dans un univers 3D, est bien différente de la conception d'une interface normal. En effet, lors de la conception d'une interface, l'écran dispose d'un canvas(conteneur) principale contenant chaque élément de l'interface, alors que dans une interface 3D, il y'a un canvas pour chaque élément et chaque canvas dispose d'une position propre dans l'univers du jeu.

Les 2 problèmes rencontrées avec la VR sont l'utilisation du système de simulation sans casque VR qui fonctionne 1 fois sur 2, ainsi que la présence d'une barre de l'armature du cockpit obstruant la vue du joueur. Pour le premier problème, la cause n'a pas été identifié mais semble lié à SteamVR, concernant le seconde problème, l'idée de cacher la barre m'a été donné par mon camarade M.Baumgartner. Avec cette idée, j'ai cherché si cela était possible et en effet, en utilisant une forme 3D disposant d'un matériel avec un Shader paramétré spécifiquement, et en assignant un script à l'élément que l'on désire dissimuler, il est possible d'effacer des parties 3D comme un outil de gomme (à la différence que la partie effacer n'est pas lié à un trait mais à une forme 3D).

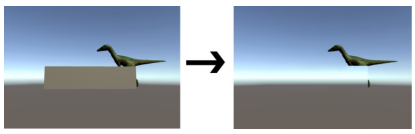

Figure 6 – *Effacer une partie 3D avec une forme 3D*

En ce qui concerne la partie Motion Plateforme, celle-ci est réalisé par une série d'essai-erreur. La première sensation à être mis en place est celle du déplacement. Pour essayer d'envoyer l'impulsion (incliner légèrement vers l'arrière le joueur, faire monter le siège a des hauteurs variables et le faire redescendre aussi tôt) au bon timing, je me suis d'abord calqué sur les pas du robot en plaçant des détecteurs sur les pieds du robot, mais le rendu n'était pas très réaliste, alors M.Grobet m'a conseillé d'envoyer l'impulsion a un rythme constant. Pour réaliser cela j'ai donc utilisé un Timer que je désactivais ou non en fonction que le joueur se déplaçait ou non. Pour la sensation de recul lorsque l'utilisateur tire, celle-ci fut très simple à implémenté, car elle consiste juste à envoyer le signal lorsque l'utilisateur tire et faire reculer le joueur d'un cran vers l'arrière avant de le ramener à sa position normale. Chaque sensation dispose d'une méthode qui modifie les valeurs devant être traité par la chaise et une méthode permettant de réinitialiser ces valeurs à 0. Une autre sensation que je désirais implémenter était le recul lorsque le joueur subit des dégâts,

mais je ne trouvais pas comment modéliser cette sensation sachant que celle-ci pourrait se confondre avec celle du tire ou risque de bloquer la réception des autres sensations, donc j'ai décidé de ne pas l'implémenter. Pour obtenir un meilleur résultat lors de la gestion des sensations, j'ai réalisé (tardivement) qu'il est nécessaire d'utiliser le design pattern "StateMachine" pour le joueur également, qui à chaque état modifierait un attribut état dans la classe gérant les mouvements. A chaque frame une méthode pourrait modifier les attributs traités par la chaise en fonction de l'état du joueur.

Finalement, j'ai travaillé sur la partie contrôleur qui consistait à lier les actions du Joystick en action dans le jeu. Pour cela j'ai utilisé le "Inputs Action" d'Unity qui permet de mapper les différentes touches et action d'un contrôleur en événement pouvant être lié à du code. Grâce à cela j'ai pu ajouter la possibilité de rotation du robot, de le faire avancer et de tirer. Il est possible de stocker les informations (ici le l'orientation du Jostick) [Figure 7] du contrôleur pour ensuite les traiter (ici pour calculer le mouvement). Il est aussi important de préciser que chaque action est désactivée et qu'il faut nécessairement les activées. Malheureusement, l'un des objectifs qui était d'utiliser les Joysticks pour contrôler à échouer. En effet, les contrôleurs sont considérés comme identique, de ce fait chaque action liée à l'un des contrôleurs est utilisable par l'autre, perdant donc l'utilité d'avoir deux contrôleurs.

```
controls = new PlayerController();
controls.Gameplay.Rotate.performed += ctx => playerRot = ctx.ReadValue<Vector2>();
controls.Gameplay.Forward.performed += ctx => playerMove = ctx.ReadValue<Vector2>();
controls.Gameplay.Forward.Enable();
controls.Gameplay.Rotate.Enable();
controls.Gameplay.Forward.canceled += ctx => playerMove = Vector2.zero;
controls.Gameplay.Rotate.canceled += ctx => playerRot = Vector2.zero;
```

Figure 7 – *Récupère les valeurs des actions Forward et Rotate et active ces actions*

IV. Résultats

Figure 8 – *Résultat à la fin du projet*

Il est possible de constater que le projet est fonctionnel et répond à la demande qui était, le développement d'un jeu de combat de robot, utilisant la VR et la Motion Plateforme pour renforcer l'immersion du joueur au sein de l'environnement 3D.

V. Conclusion

En conclusion, le projet a pu être réalisé dans les temps tout en respectant l'énoncé et les outils imposés. De plus le développement de ce projet m'a permis de mieux me familiariser avec le développement de jeu utilisant la VR, ainsi qu'une première prise en main avec la Motion Plateforme.

En ce qui concerne les perspectives d'amélioration du projet, l'utilisation d'un état fini pour la gestion du robot du joueur et de la classe gérant les sensations, permettrait d'obtenir un résultat bien meilleur. En effet, l'utilisation d'une telle pratique rendrait la gestion des secousses plus souple et simplifié, tout en permettant d'éviter que des sensations ne s'empile. Un autre point d'amélioration serait également au niveau du comportement de l'IA lors du combat, en complexifiant ses actions et lui créer des patterns et attaques différents (possible de réaliser cela avec des assets payant sur l'Assetstore d'Unity). Finalement, les dernières améliorations a souligné sont l'utilisation d'un second contrôleur pour permettre de séparer la rotation du joueur et ces mouvements sur 2 contrôlleurs différents, et une meilleure utilisation du système graphique d'Unity.