

Hello World

Rapport de soutenance n°1



16 juin 2024

Table des matières

1	Introduction	2
1.1	Présentation du projet	2
1.2	Histoire et gameplay	2
1.3	Rappel du planning / Répartition des tâches	3
2	Aspect Technique	4
2.1	Multijoueur	4
2.1.1	Introduction	4
2.1.2	Implémentation	4
2.1.3	Problèmes rencontrés	7
2.2	Graphisme & Musique (Sacha)	8
2.2.1	Effets Sonores	8
2.2.2	Musique	8
2.2.3	Logo (Sacha & Damien)	9
2.2.4	Croquis ancien logo	9
2.2.5	Personnage	10
2.2.6	Design des personnages (Sacha & Damien)	10
2.2.7	Animations des personnages	10
2.2.8	Cinématique	11
2.3	Interfaces	12
2.3.1	Login (Sacha)	12
2.3.2	Settings	16
2.3.3	Inventaire (Damien & Sacha)	18
2.3.4	Shop	19
2.4	Mécaniques de jeu	20
2.4.1	Déplacement	20
2.4.2	Saut et chute	20
2.4.3	Modes de jeu	21
2.5	Perspectives d'Amélioration	22
3	Website	24
4	Retards, Avances et Difficultés	25
4.0.1	Adaptation au groupe	25
4.0.2	Avances	25
5	Conclusion Globale	25
5.1	Prévisions pour la prochaine soutenance	25

1 Introduction

1.1 Présentation du projet

BattleUnitVerse est un jeu de plateforme 2D créé par l'entreprise Hello-World, dont les membres sont Damien Placé, Sacha Sitbon, Raphaël Nawawi, Barthélémy Marco Mora—Kaba et Yanis Dali. Ce jeu utilise le moteur Unity et est disponible pour toutes les plateformes d'ordinateurs tels que Mac, Windows et Linux. Le but de ce jeu est de mettre K.O. l'adversaire en le sortant de l'arène de combat, incluant également divers modes de jeu et possibilités.

1.2 Histoire et gameplay

Le concept de BattleUnitVerse a été inspiré par de grands jeux de combats tels que Brawlhalla et Super Smash Bros. L'univers de ce jeu est basé entre autres sur l'existence de plusieurs univers spatio-temporelles appelés « verses » et de voyageurs explorant ces différents univers appelés « voyageurs du verse » se déplaçant à l'aide de machines capables d'explorer les mondes parallèles appelés Battleunits. Ces personnages ont pour mission d'assurer le contrôle de ces univers et de combattre leurs adversaires non-issus de leur univers. Les joueurs incarnent ces derniers afin de s'affronter dans un jeu de plateforme où le but ici est de mettre K.O. l'adversaire en le sortant de l'arène de combat.

1.3 Rappel du planning / Répartition des tâches

<Hello World/>

BattleUnitVerse

REPARTITION DES TÂCHES

Membres \ Tâches	DAMIEN	SACHA	YANIS	RAPHAEL	BARTHELEMY
<i>DESIGN DU JEU</i>		Réponsable		Suppléant	
<i>GRAPHISME ET MUSIQUE</i>	Suppléant	Responsable		Suppléant	
<i>INTERFACE</i>	Responsable	Suppléant			Suppléant
<i>MULTIJOUEUR</i>	Responsable				Suppléant
<i>COLLISION PLATFORM</i>			Suppléant		Responsable
<i>IA</i>		Responsable	Suppléant		
<i>MAP/SKIN</i>	Responsable	Suppléant			
<i>COMPÉTENCE SPÉCIAL</i>	Responsable			Suppléant	
<i>SPAWN OBJET RT JOUEURS</i>			Responsable		Suppléant
<i>DÉPLACEMENT/CALCUL</i>			Responsable		Suppléant
<i>PUB/TRAILER</i>	Suppléant	Responsable			
<i>SITE WEB</i>		Suppléant		Responsable	

La planification est la partie la plus fondamentale du projet. Elle doit être rigoureusement respectée pour que le projet puisse être mené à bien. En travaillant sur notre projet, nous nous sommes rendu compte que notre planification était peut-être trop ambitieuse. En effet, les difficultés d'organisation, d'apprentissage ainsi que d'autres facteurs individuels, nous ont amenés à revoir nos échéances afin d'avoir une planification plus réaliste et donc plus favorable au projet et à l'ensemble de l'équipe.

(voir en pièce jointe le diagramme de Gantt)

Tableau d'avancement des tâches

Site Web	100 %
Mécanique de déplacement	100 %
Design et interface	100 %
Collision	100 %
Spawn objet	En cours... (0 %)
Compétences spéciales	En cours... (0 %)
Intelligence artificielle	En cours... (0 %)
Pub et trailer	En cours... (0 %)
Multijoueur	En cours d'achèvement (70 %)

2 Aspect Technique

2.1 Multijoueur

2.1.1 Introduction

Dans cette section, nous abordons la mise en œuvre du mode multijoueur dans notre projet. L'intégration d'une fonctionnalité multijoueur était essentielle pour offrir une expérience agréable et sociale aux utilisateurs. Pour ce faire, nous avons choisi d'utiliser Photon, une solution bien établie et largement utilisée pour le développement de jeux multijoueurs avec Unity.

Photon offre une architecture robuste basée sur le cloud, permettant une communication en temps réel entre les joueurs, ainsi qu'une gestion efficace des sessions et des connexions. Cette section se concentre sur la description du fonctionnement de Photon dans le contexte de notre projet.

2.1.2 Implémentation

L'implémentation du mode multijoueur a été réalisée en plusieurs étapes. Dans un premier temps, nous avons ajouté le package Photon Pun 2 disponible sur l'Asset Store de Unity. En effet Pun 2 est une version largement documentée et qui est énormément utilisée dans la création de jeu multijoueur pour approfondir notre compréhension et notre expertise dans l'utilisation de ce package. Barthélémy et moi-même (Damien) avons suivi une formation sur Udemy, une plateforme de formation en ligne réputée.

Système de connection : Lorsque les joueurs lancent l'application, ils sont dirigés vers un écran de login. Dans ce dernier, ils peuvent entrer leur pseudonyme. Une fois qu'un joueur a saisi son pseudo, il peut cliquer sur le bouton de connexion. À ce stade, Photon Network est configuré pour utiliser les paramètres de connexion spécifiés. Si la connexion est réussie, les joueurs sont redirigés vers le lobby principal du jeu.

```
1 public void OnLoginButtonClicked()
2 {
3     string playerName = PlayerPseudo.text;
4
5     PhotonNetwork.NickName = playerName;
6     PhotonNetwork.ConnectUsingSettings();
7 }
```

Création et Gestion des Rooms : Une fois dans le lobby principal, les joueurs ont plusieurs options, dont la création d'une nouvelle room, la sélection d'une room existante, ou alors rejoindre une room aléatoirement.

Création d'une Room : Les joueurs peuvent créer leur propre room en spécifiant un nom et le nombre maximum de joueurs. Une fois que la room est créée, les autres joueurs peuvent la rejoindre. Si un joueur est le premier à rejoindre la room, il devient automatiquement l'hôte de la room.

```
1 public void OnCreateRoomButtonClicked()
2 {
3     //on recupere le nom de la room
4     string roomName = RoomNameInputField.text;
5     //si le nom de la room est vide => nom par defaut
6     roomName = (roomName.Equals(string.Empty)) ? "Room " +
7         Random.Range(1000, 10000) : roomName;
8
9     //maxPlayers est le nombre de joueurs max
10    byte maxPlayers;
11    //si le nombre de joueurs est invalide => valeur par defaut
12    byte.TryParse(MaxPlayersInputField.text, out maxPlayers);
13    // 2 et 4 valeurs min et max de joueur
14    maxPlayers = (byte) Mathf.Clamp(maxPlayers, 2, 4);
15
16    RoomOptions options = new RoomOptions {MaxPlayers =
17        ↪ maxPlayers};
        PhotonNetwork.CreateRoom(roomName, options, null);
}
```

Rejoindre une Room Aléatoire : Les joueurs peuvent également choisir de rejoindre une room aléatoire. Dans ce cas, Photon Network sélectionne automatiquement une room disponible et rejoint le joueur à celle-ci. Pour gérer cette fonctionnalité, nous avons mis en place des méthodes spécifiques dans notre script de gestion du réseau.

Lorsqu'un joueur tente de rejoindre une room aléatoire et que la tentative échoue, la méthode `OnJoinRandomFailed` est déclenchée. Dans cette méthode, un nom de room aléatoire est généré à l'aide d'un numéro aléatoire. Ensuite, une instance de `RoomOptions` est créée pour spécifier le nombre maximum de joueurs pouvant rejoindre la room. Enfin, la méthode `CreateRoom` de Photon Network est appelée pour créer une nouvelle room avec le nom généré et les options spécifiées.

Sélectionner une Room Existante : Pour rejoindre une room existante dans notre jeu, nous utilisons les dictionnaires `cachedRoomList` et `roomListEntries` pour afficher une liste des rooms disponibles dans l'interface utilisateur. Cette liste est généralement mise à jour périodiquement pour refléter les changements dans les rooms disponibles.

Lorsque le joueur sélectionne une room dans la liste, nous récupérons les informations de cette room à partir du dictionnaire `cachedRoomList` et utilisons les méthodes fournies par Photon Network pour rejoindre cette room spécifique. Une fois que le joueur a rejoint avec succès la room, les informations sur les joueurs présents dans la room sont affichées dans l'interface utilisateur en utilisant le dictionnaire `playerListEntries`.

```
1  private Dictionary<string, RoomInfo> cachedRoomList;
2  private Dictionary<string, GameObject> roomListEntries;
3  private Dictionary<int, GameObject> playerListEntries;
```

Intancier les joueurs : Chaque joueur dans notre jeu est représenté par un "BattleUnit", une entité contrôlable qui peut interagir avec d'autres éléments du jeu. L'instanciation de ces BattleUnits sur la carte est gérée par Photon. Pour ce faire, nous avons utilisé la fonction `PhotonNetwork.Instantiate()`.

Lorsqu'un joueur se connecte au jeu, `PhotonNetwork.Instantiate()` est appelé avec le préfabriqué de la BattleUnit correspondant au joueur. Cela déclenche la création d'une instance de cette BattleUnit sur la carte de jeu, permettant ainsi au joueur de commencer à jouer.

Un aspect crucial de cette implémentation est l'utilisation de `PhotonView` pour synchroniser les mouvements et les actions des BattleUnits entre tous les clients connectés. Chaque BattleUnit est associée à un `PhotonView`, ce qui permet à Photon de synchroniser automatiquement les données entre les clients. Cela garantit que tous les joueurs voient les mêmes actions se produire sur leur écran, créant ainsi une expérience de jeu fluide et cohérente.

2.1.3 Problèmes rencontrés

Nous avons rencontré un problème lié à l'affichage des animations des personnages lors des parties multijoueurs. Actuellement, les animations ne sont pas visibles pour tous les joueurs connectés. Après une analyse approfondie, nous avons identifié que ce problème est dû à la manière dont nous avons configuré les scripts de mouvement dans le PhotonView.

Par conséquent, la création d'un tout nouveau script pour gérer les animations sera nécessaire pour résoudre ce problème.

de plus certains joueurs peuvent parfois rencontrer des lags pendant les parties. Pour résoudre ce problème, nous prévoyons d'ajouter un script de lag compensation. Ce script permettra de corriger les éventuels décalages causés par la latence du réseau, assurant ainsi une expérience de jeu plus fluide et cohérente pour tous les joueurs.

2.2 Graphisme & Musique (Sacha)

2.2.1 Effets Sonores

Actuellement, bien que je n'aie pas encore intégré d'effets sonores dans le jeu, je prévois de créer une variété d'effets sonores pour enrichir l'expérience des joueurs. Ces effets sonores incluront notamment des sons pour les actions telles que la pose de murs, la création de sols, ainsi que les coups portés par les personnages.

2.2.2 Musique

La bande sonore de BattleUnitVerse, que j'ai composée dans FL Studio, accompagne les différentes phases du jeu, offrant une atmosphère agréable aux joueurs. Actuellement, j'ai intégré de la musique d'ambiance pour créer une atmosphère immersive, ainsi qu'une musique de victoire pour célébrer les succès des joueurs. Bien que je n'aie pas encore ajouté de pistes musicales spécifiques pour les moments de combat, cela est prévu dans nos futures itérations pour enrichir davantage l'expérience sonore globale.

J'ai cherché à perfectionner mes compétences en musique pour jeux vidéo. J'ai donc décidé de regarder des tutoriels en ligne sur la création de musiques captivantes spécifiquement conçues pour les jeux vidéo. Ces tutoriels m'ont donné de précieux conseils et techniques que j'ai ensuite appliqués dans FL Studio pour concevoir la musique finale du jeu.

Grâce à ces tutoriels, j'ai pu approfondir ma compréhension des éléments clés nécessaires pour créer une ambiance immersive dans un jeu vidéo. J'ai appris l'importance des basses dans la création d'une atmosphère dynamique, ainsi que des techniques pour créer des transitions entre les différentes sections de la musique.

Bien que j'aie déjà eu une bonne base dans l'utilisation de FL Studio, ces tutoriels m'ont permis d'explorer de nouvelles techniques et de perfectionner mes compétences pour créer une bande sonore.

```
1 public void SetVolume()
2 {
3     if (audioSource != null)
4     {
5         audioSource.volume = volumeSlider.value;
6     }
7     else
8     {
9         Debug.LogWarning(" AudioSource not assigned in the
10            ↪ inspector.");
11    }
12 }
```

2.2.3 Logo (Sacha & Damien)

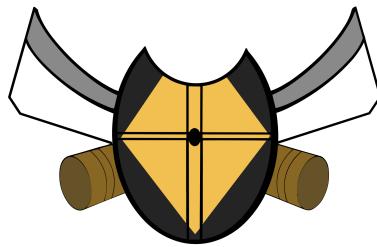


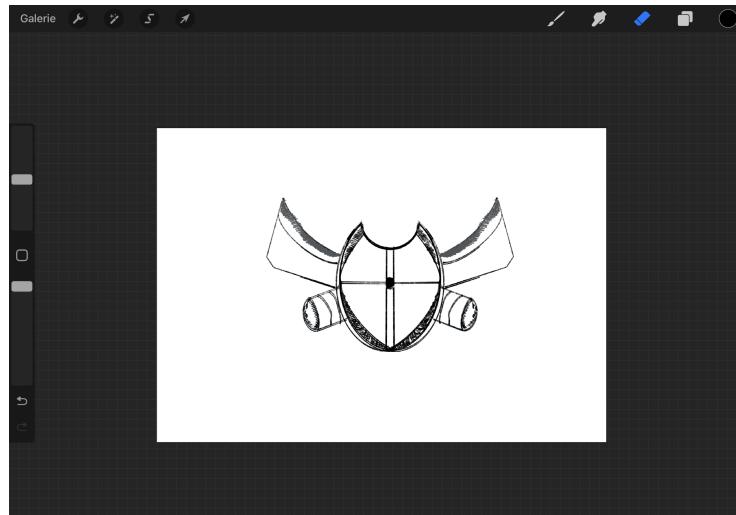
FIGURE 1 – Ancien logo



FIGURE 2 – Nouveau logo

Nous avons récemment entrepris une refonte du logo original de BattleUnitVerse pour lui donner une apparence plus réaliste tout en conservant ses éléments caractéristiques. Utilisant l'application Procreate, nous avons repensé chaque détail pour créer une version modernisée du logo. Le bouclier, représentant la lettre "B" pour Battle, a été redessiné avec des lignes nettes et des reflets pour lui donner une texture tridimensionnelle. La forme du bouclier, évoquant le "U" de Unit, a été préservée tout en étant modernisée pour s'aligner avec les designs. Pour représenter le "V" de Verse, nous avons introduit des éléments d'épées croisées, symbolisant l'action et l'aventure au cœur du jeu.

2.2.4 Croquis ancien logo



Croquis ancien logo sur procreate

2.2.5 Personnage

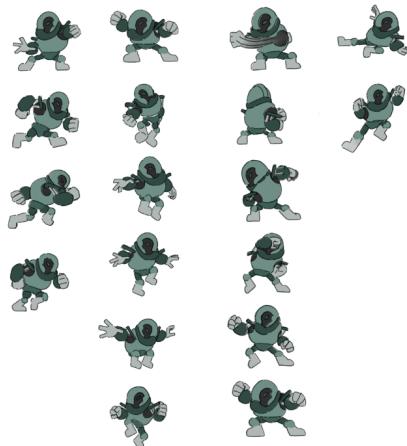
Dans BattleUnitVerse, les personnages "BattleUnit" incarnent les machines que les joueurs contrôlent, chacune ayant un objectif clair : gagner contre son adversaire ou le ramener dans son univers d'origine. Ces machines, imprégnées de technologie avancée et de puissance, sont les représentations physiques des joueurs dans l'univers du jeu.

2.2.6 Design des personnages (Sacha & Damien)

En collaboration avec Damien, nous avons travaillé étroitement pour concevoir le design des robots emblématiques du jeu, appelés les "BattleUnits". Ces robots sont l'essence même de BattleUnitVerse, et leur design a été soigneusement élaboré pour capturer l'essence de l'univers du jeu. À travers de nombreuses itérations et des dessins réalisés sur Procreate, nous avons exploré diverses formes, structures et détails pour trouver le design parfait représentant le BattleUnit.

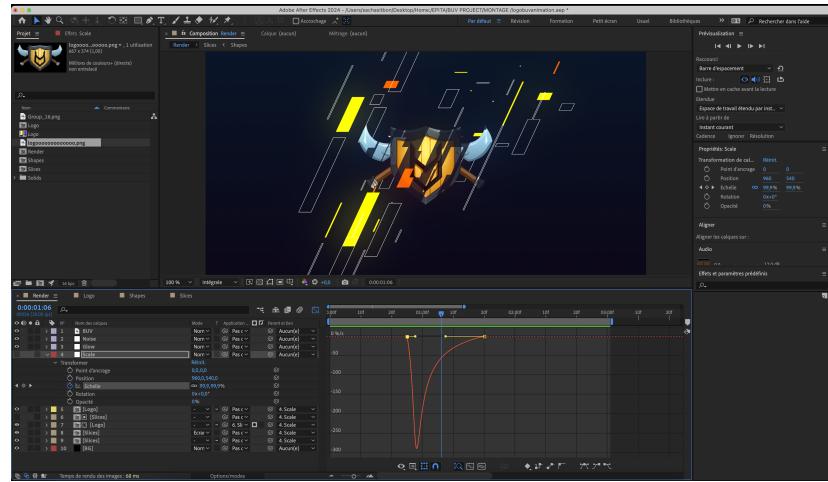
2.2.7 Animations des personnages

Après avoir finalisé les designs des BattleUnits, j'ai entrepris de leur donner vie à travers des animations réalistes. J'ai personnellement réalisé les animations de coups, de sauts, de glissades et de mouvements.



BattleUnit

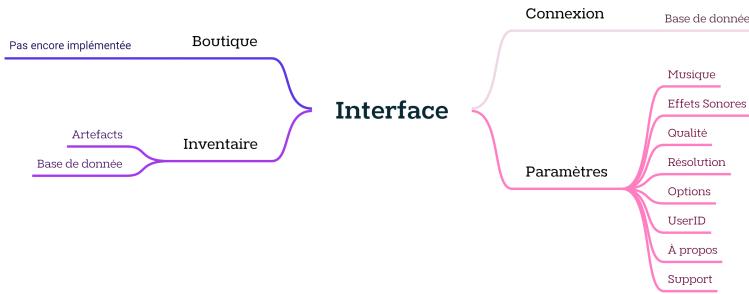
2.2.8 Cinématique



Dans les cinématiques de BattleUnitVerse, j'ai personnellement intégré un effet de glitch/faille qui représente l'univers complexe et mystérieux du jeu, influencé par la convergence temporelle et les différents univers. Cet effet visuel, créé sur After Effects, ajoute une dimension supplémentaire à l'esthétique de l'univers, capturant l'idée de distorsion temporelle et de réalités parallèles. J'ai également implémenté ces cinématiques sur Unity, les intégrant au moteur de jeu. De plus, j'ai inclus la musique de fond directement dans le code, synchronisant les visuels avec l'ambiance sonore pour une expérience cinématique cohérente. J'ai intégré la cinématique de BattleUnitVerse sur Unity, la plaçant au lancement du jeu. De plus, j'ai ajouté un bouton "skip" qui permet aux joueurs de passer directement à la scène de connexion.

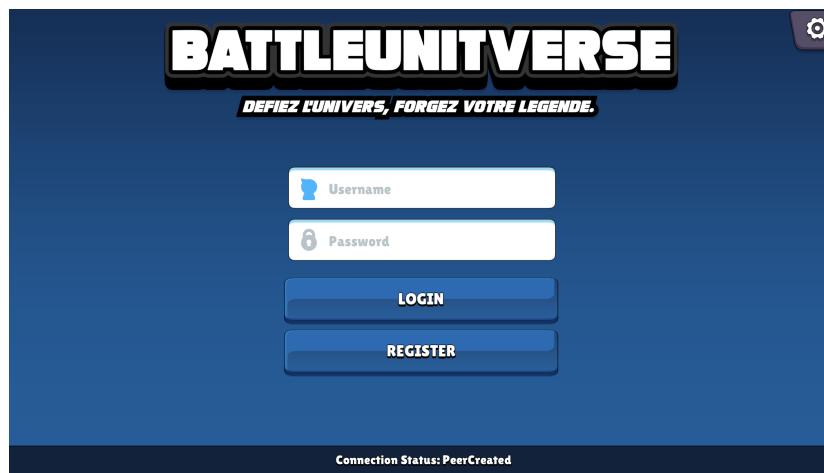
2.3 Interfaces

2.3.1 Login (Sacha)



Dans cette section, nous abordons les points suivants concernant la gestion des comptes joueurs :

- **Établissement de la connexion initiale** avec la base de données et implémentation des fonctionnalités essentielles, telles que l'inscription des utilisateurs et la gestion des sessions.
- **Sécurité des données utilisateurs** : adoption de mesures préventives contre les vulnérabilités pour protéger les informations des joueurs.
- **Amélioration de compétences en gestion de bases de données** pour une administration efficace et optimisée.
- **Perspectives d'évolution**, notamment la migration de notre API vers un serveur dédié une fois son développement achevé et sa stabilité assurée.



```
1 <?php
2
3     DEFINE("HOST", "localhost");
4     DEFINE("DBNAME", "BUV");
5     DEFINE("USERNAME", "root");
6     DEFINE("PASS", "");
7
8     class Helper {
9         function ConnectBDD(){
10             return new PDO('mysql:host=' . HOST . ';' . dbname' . DBNAME,
11                             ↪ USERNAME, PASS);
12
13         }
14
15         function GetUsers($name = null){
16             $bdd = $this ->ConnectBDD();
17
18             $where = ($name != null) ? "AND player=" . $name : "";
19
20             $sql = "SELECT * FROM user WHERE id>0 " . $where;
21
22             $result = $bdd->prepare($sql);
23             $result->execute();
24
25             $d = $result->fetchALL(PDO::FETCH_ASSOC);
26             return $this->ParseUserJson($d);
27
28         }
29
30     }
31
32 ?>
```

Explication du code : Ce document présente une description du code PHP donné, qui implémente une connexion à une base de données MySQL et récupère des informations sur les utilisateurs.

Définition des Constantes

Le script commence par définir quatre constantes pour la connexion à la base de données MySQL :

- **HOST** : Définit l'adresse de l'hôte de la base de données. La valeur assignée est "localhost", indiquant que la base de données est située sur le même serveur que le script.
- **DBNAME** : Définit le nom de la base de données à utiliser. La valeur assignée est "BUV".
- **USERNAME** : Définit le nom d'utilisateur pour la connexion à la base de données. La valeur assignée est "root".
- **PASS** : Définit le mot de passe pour la connexion à la base de données. La valeur assignée est une chaîne vide, indiquant qu'aucun mot de passe n'est requis.

Classe Helper

La classe **Helper** contient deux méthodes principales pour interagir avec la base de données.

Méthode ConnectBDD()

Cette méthode établit une connexion à la base de données MySQL et retourne un objet PDO. Elle utilise les constantes définies précédemment pour construire la chaîne de connexion.

Méthode GetUsers(\$name = null)

Cette méthode est utilisée pour récupérer des informations sur les utilisateurs de la base de données. Elle prend un paramètre optionnel \$name qui, s'il est fourni, filtre les utilisateurs par leur nom.

1. Une connexion à la base de données est établie en appelant **ConnectBDD()**.
2. La méthode construit une requête SQL pour sélectionner tous les utilisateurs qui correspondent aux critères fournis. Si un nom est spécifié, il est utilisé pour ajouter une condition de filtrage.
3. La requête est exécutée, et les résultats sont récupérés et retournés sous forme d'un tableau associatif.
4. Les données récupérées sont ensuite traitées par une autre méthode (supposée existante), **ParseUserJson**, pour une éventuelle transformation ou mise en forme.

Introduction au Développement de l'API

Dans le cadre du développement de notre jeu, l'attention a été portée sur la création d'une API permettant la communication efficace avec la base de données à travers le langage PHP. Le code présenté s'inscrit dans cette démarche comme un exemple préliminaire de la structure que nous ambitionnons pour notre API. Bien que des progrès significatifs aient été réalisés, notamment dans la récupération des informations des utilisateurs basée sur certains critères tels que le nom, il est reconnu que l'intégralité de l'API est loin d'être complète.

Développement Actuel et Perspectives

Actuellement, l'API en développement offre la possibilité de récupérer les données des utilisateurs. Cependant, la vision à long terme englobe l'intégration de fonctionnalités complémentaires essentielles, notamment :

- L'inscription de nouveaux utilisateurs,
- La gestion des sessions,
- La mise à jour des informations des comptes utilisateurs.

La conscience de la nécessité d'améliorer la sécurité et la robustesse de l'API guide également nos efforts, avec l'objectif de garantir la protection des données utilisateur et la prévention contre les vulnérabilités.

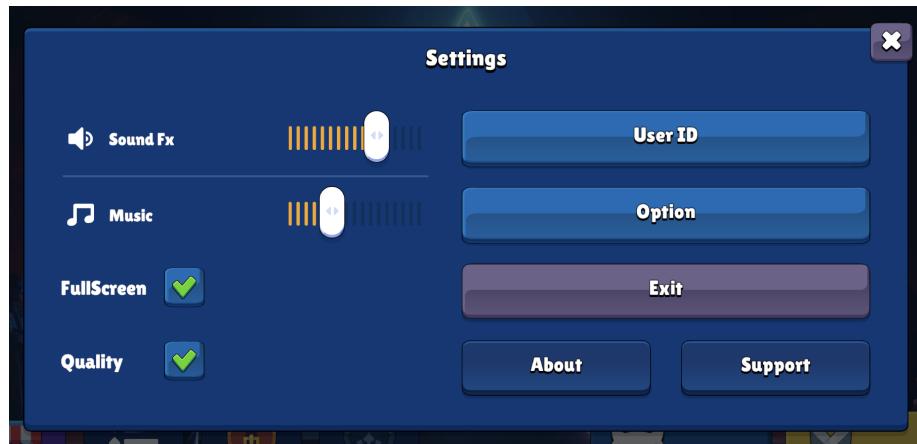
Formation et Outils de Développement

Pour soutenir le développement de l'API, un engagement personnel a été pris pour améliorer les compétences en gestion de bases de données par le biais d'une formation spécialisée sur Udemy. Cette initiative a permis d'élargir ma compréhension des meilleures pratiques en matière de conception et manipulation des bases de données.

Du point de vue des outils de développement, l'approche a été structurée autour de :

- L'utilisation de **PHPMyAdmin** pour la gestion de la base de données MySQL, offrant une interface utilisateur intuitive pour la création, manipulation, et visualisation des données.
- L'utilisation de **WampServer** pour simuler un environnement de serveur web local, facilitant ainsi le processus de développement et de test. Bien que l'API soit actuellement hébergée localement, l'ambition est de la transférer sur un serveur dédié postérieurement, assurant ainsi sa stabilité et sa disponibilité.

2.3.2 Settings



Settings

Dans la section "Settings" j'ai mis en place plusieurs fonctionnalités visant à offrir aux joueurs un contrôle personnalisé sur divers aspects de l'expérience de jeu :

Contrôle du son : J'ai implémenté une barre de réglage permettant aux joueurs d'ajuster le volume de la musique de fond selon leurs préférences. De plus, j'ai également prévu une fonction similaire pour contrôler les effets sonores du jeu, bien que ces derniers ne soient pas encore intégrés.

Option plein écran : Pour offrir une expérience de jeu plus immersive, j'ai inclus une option permettant aux joueurs de basculer entre le mode plein écran et le mode fenêtré. Ils peuvent choisir de cocher une case pour activer le mode plein écran, ou laisser la case décochée pour jouer dans une fenêtre redimensionnable, garantissant ainsi une expérience de jeu responsive.

Bouton Exit : J'ai ajouté un bouton "Exit" qui permet aux joueurs de quitter le jeu facilement. Cette fonctionnalité offre une solution pratique pour les joueurs qui souhaitent quitter le jeu rapidement et en toute simplicité. En outre, bien que ces fonctionnalités ne soient pas encore pleinement implémentées, j'ai également intégré des éléments visuels pour les sections suivantes :

UserID : Cette section permet aux joueurs de récupérer leur identifiant de joueur. Bien que cette fonctionnalité ne soit pas encore disponible, l'interface graphique correspondante est déjà présente pour faciliter son intégration future.

Options : Dans cette section, les joueurs auront la possibilité de personnaliser divers paramètres du jeu, tels que les options graphiques, les paramètres de contrôle et d'autres préférences personnalisables. Bien que cette fonctionnalité ne soit pas encore opérationnelle, son emplacement est déjà prévu dans l'interface utilisateur.

Options : Cette page fournit des informations sur l'équipe de développement du jeu et son contexte. Bien qu'elle soit actuellement non fonctionnelle, elle est présente visuellement pour une intégration future.

Support : Cette section fournira aux joueurs un moyen de contacter l'équipe de développement en cas de besoin d'assistance ou de rapport de bugs. Bien qu'elle ne soit pas encore active, son inclusion visuelle est déjà réalisée pour une activation ultérieure.

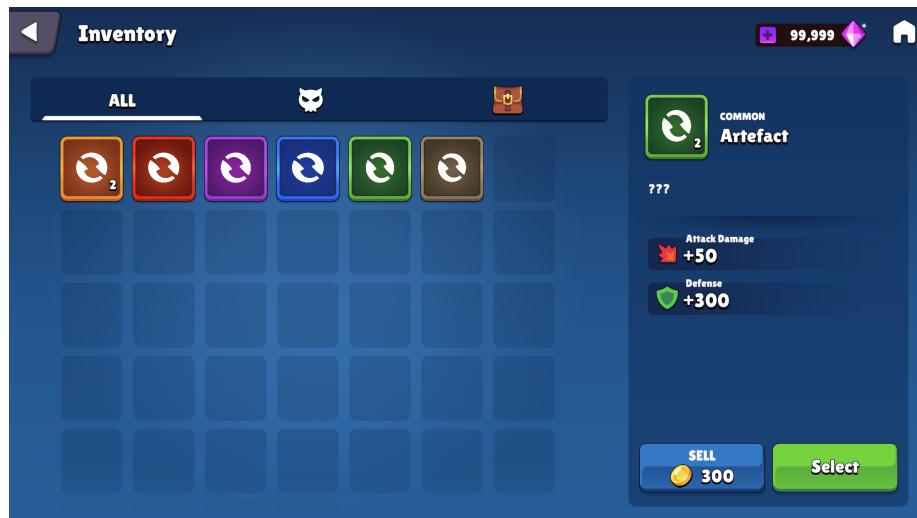
```
1  public void SetQuality(bool isHighQuality)
2  {
3      int qualityLevel = isHighQuality ? 5 : 0;
4      QualitySettings.SetQualityLevel(qualityLevel);
5  }
6
7  public void SetFullscreen(bool isFullscreen)
8  {
9      Screen.fullScreen = isFullscreen;
10 }
11
12 public void SetVolume()
13 {
14     if (audioSource != null)
15     {
16         audioSource.volume = volumeSlider.value;
17     }
18     else
19     {
20         Debug.LogWarning(" AudioSource not assigned in the
21                         ↪ inspector.");
22     }
23 }
```

SetQuality(bool isHighQuality) : Cette fonction permet de régler la qualité graphique du jeu en fonction d'un booléen "isHighQuality". Si "isHighQuality" est vrai, le niveau de qualité est défini sur 5 (le niveau le plus élevé), sinon, il est défini sur 0 (le niveau le plus bas). Elle utilise la fonction QualitySettings.SetQualityLevel() pour effectuer ce réglage.

SetFullscreen(bool isFullscreen) : Cette fonction permet de basculer entre le mode plein écran et le mode fenêtré du jeu en fonction d'un booléen "isFullscreen". Si "isFullscreen" est vrai, le jeu passe en mode plein écran ; sinon, il passe en mode fenêtré. Elle utilise la propriété Screen.fullScreen pour effectuer cette transition.

SetVolume() : Cette fonction permet de régler le volume du son dans le jeu en fonction de la valeur du curseur de volume. Elle vérifie d'abord si la variable "audioSource" est affectée (assignée) dans l'inspecteur. Si c'est le cas, elle ajuste le volume de l'audio source en fonction de la valeur du curseur de volume. Sinon, elle génère un avertissement dans la console indiquant que l'audio source n'est pas affectée.

2.3.3 Inventaire (Damien & Sacha)



Inventaire

Gestion des artefacts : L'inventaire permettra aux joueurs de stocker les artefacts qu'ils obtiennent à la fin de chaque partie. Chaque artefact peut appartenir à différentes catégories, allant des artefacts peu communs aux très rares. Chaque catégorie d'artefact confère des avantages spécifiques au joueur lors de ses prochaines parties.

Variété d'artefacts : Les artefacts peuvent être de nature diverse, offrant des avantages variés et uniques aux joueurs. Certains peuvent renforcer les compétences du joueur, augmenter les gains de ressources ou offrir des bonus spéciaux pendant le jeu.

Stockage dans la base de données : Dans le futur, Sacha prévoit d'intégrer la gestion de l'inventaire directement dans la base de données. Chaque joueur aura son propre inventaire stocké dans la base de données, avec des entrées pour chaque artefact qu'ils possèdent. Cette initiative permettra aux joueurs de conserver leurs artefacts même en cas de déconnexion ou de changement d'appareil, assurant ainsi une expérience de jeu fluide et cohérente.

2.3.4 Shop

Concernant la section boutique de notre jeu, nous avons décidé de la développer en dernière étape du projet. Bien que cette fonctionnalité ne soit pas prioritaire pour le moment, elle offrira à terme aux joueurs la possibilité d'acquérir des objets ou des coffres qui enrichiront leur expérience de jeu. Nous sommes conscients que ces éléments peuvent augmenter l'engagement et la personnalisation, mais nous souhaitons d'abord nous concentrer sur le cœur de l'expérience ludique.

2.4 Mécaniques de jeu

2.4.1 Déplacement

Nous avons créé des scripts dans notre jeu. Ces scripts permettent de définir un comportement pour chaque actions de l'utilisateur afin de pouvoir se déplacer, sauter etc. Les commandes pour se déplacer dans le jeu s'adaptent en fonction du clavier que le joueur utilise. Ainsi, il n'est pas nécessaire de configurer les commandes dans des réglages.

Le joueur peut se déplacer dans l'espace dans le sens horizontal. Lorsque une commande pour se déplacer est activée, il faut bien définir une vitesse afin que le personnage se déplace. Par ailleurs, lorsque le joueur change de direction, le corps du personnage se retourne également pour faire face au nouveau sens de déplacement. Cela ajoute un certain réalisme au jeu et favorise l'immersion dans le jeu.

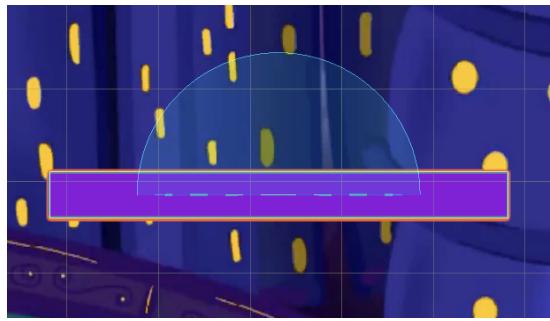
```
1     float dirX = Input.GetAxis("Horizontal"); // -1 0 1
2     rb.velocity = new Vector2(dirX * moveSpeed,
3                               ↪ rb.velocity.y);
```

2.4.2 Saut et chute

Le joueur peut sauter deux fois avant d'atterrir de nouveau sur le sol. Il ne pourra pas sauter plus de deux fois. Il faut donc faire en sorte que le personnage puisse sauter deux fois à chaque fois qu'il pose pied sur une plateforme. Il faut donc s'assurer de stocker le nombre de sauts restants afin que la limite ne soit pas dépassée par l'utilisateur.

```
1     if (IsGrounded())
2     {
3         Jumps = 2;
4     }
5
6     if (Input.GetButtonDown("Jump") && Jumps>1)
7     {
8         rb.velocity = new Vector2(rb.velocity.x, jumpForce);
9         Jumps--;
10    }
```

Lorsqu'il saute, le joueur retombe vers le bas. Pour qu'il puisse s'appuyer sur des plateformes, un collider est mis en place. Il agira comme une force de réaction normale. Cela permettra au joueur de ne pas tomber dans le vide lorsqu'il se trouve sur une surface.



Une des parties les plus fondamentales du jeu est sûrement celle du spawn. C'est l'apparition du joueur sur le terrain de jeu. Les joueurs apparaissent à deux ou à quatre.

2.4.3 Modes de jeu

Le mode joueur contre joueur peut affronter 2 à 4 joueurs (1 VS 1 et 2 VS 2 respectivement). Ceux qui sont éliminés cessent de jouer et passent en mode spectateur. Une musique se lance sur le jeu de celui qui gagne : il remporte la partie.

Il y a deux modes de jeu. Un mode wave c'est à dire un mode de jeu où des ennemis à abattre vont arriver par vagues. Le but serait donc de faire spawn une multitude d'ennemis sur le terrain. Ceux-ci ce défendraient grâce à une intelligence artificielle. Le joueur gagne si il parvient à supporter les vagues et est éliminé si il meurt.

On peut vaincre les ennemis en effectuant une suite de combos c'est à dire d'attaques. La structure du terrain de jeu avec les plateformes permettent au joueur de se déplacer dynamiquement à travers le terrain. Les doubles sauts et les doubles sauts permettront également d'esquiver les adversaires facilement et participent donc au dynamisme du jeu.



2.5 Perspectives d'Amélioration

nous envisageons d'implémenter une intelligence artificielle (IA) pour le mode "wave" de notre jeu. Ce mode se concentre sur la défense de portails contre les assauts de monstres venant d'autres mondes. Les IA, incarnant les monstres, utiliseront une technique de "Pathfollowing", ce qui signifie qu'elles poursuivront les joueurs. Cette approche vise à rendre les défis plus dynamiques et engageants, en augmentant l'immersion et la stratégie requise pour protéger les portails efficacement.

Listing 1 – Exemple d'implementation auquel Sacha et Yanis ont réfléchi

```

1 public class EnemyAI : MonoBehaviour
2 {
3     // Cible que l'ennemi doit suivre
4     public Transform playerTarget;
5     // L'agent de navigation pour le pathfinding
6     private NavMeshAgent agent;
7     public float detectionRadius = 10.0f;
8     private Rigidbody2D rb;
9     public float health = 100f;
10    public float attackRange = 1.0f;
11    public float attackDelay = 2.0f;
12    private float lastAttackTime = 0;
13
14    void Start()
15    {
16        agent = GetComponent<NavMeshAgent>();
17        rb = GetComponent<Rigidbody2D>();
18    }
19
20    void Update()
21    {
22        if (rb.velocity.x > 0.05f)
23        {
24            transform.localScale = new Vector3(1f, 1f, 1f);
25        }
26        else if (rb.velocity.x < -0.05f)
27        {
28            transform.localScale = new Vector3(-1f, 1f, 1f);
29        }
30
31        if (PlayerInDetectionRadius())
32        {
33            agent.SetDestination(playerTarget.position);
34            if (Vector3.Distance(playerTarget.position,
35                transform.position) <= attackRange && Time.time

```

```
35             ↪ > lastAttackTime + attackDelay)
36         {
37             Attack();
38             lastAttackTime = Time.time;
39         }
40     }
41
42     private bool PlayerInDetectionRadius()
43     {
44         return Vector3.Distance(transform.position,
45             ↪ playerTarget.position) < detectionRadius;
46     }
47
48     private void Attack()
49     {
50         Debug.Log("L'ennemi attaque le joueur !");
51     }
52
53     public void TakeDamage(float damage)
54     {
55         health -= damage;
56         if (health <= 0)
57         {
58             Die();
59         }
60     }
61
62     private void Die()
63     {
64         Debug.Log("L'ennemi est mort.");
65         playerTarget.GetComponent<PlayerHealth>().LoseLife();
66         Destroy(gameObject);
67     }
```

Explication du code

Le code fourni illustre l'intelligence artificielle (IA) de monstres dans BattleUnitVerse utilisant Unity. Les monstres sont dotés de la capacité de suivre un joueur ciblé, de détecter sa présence dans un rayon spécifique et de l'attaquer lorsqu'il est à portée. Ce comportement est réalisé à l'aide d'un agent de navigation pour le déplacement et d'un Rigidbody pour la physique. Le monstre attaque à intervalles réguliers et peut subir des dégâts, menant éventuellement à sa destruction et à une pénalité pour le joueur.

3 Website

Nous avons communément décidé de créer un site à l'image de notre groupe et de notre créativité, c'est-à-dire qui respectent nos couleurs soit le jaune-noir et qui reflète de notre imagination. Pour ce faire, les chargés de la conception du site sont Sacha et Raphaël. Le site web appelé "BattleUnitWeb" a été codé en html sur "Visual Studio", le développement du site s'étend sur 2 mois à compter du mois de janvier. Pour le moment le seul élément qui n'est pas encore présent est le lien pour télécharger le jeu, qui ne sera mis en place que pour la troisième soutenance avec la version finale. Dans BattleUnitWeb nous pouvons y retrouver d'amples informations tel que l'histoire du jeu, la création du multijoueur qui est en soi très important. Les descriptions de différents objets sont aussi exposées pour avoir une meilleure compréhension du jeu, car BattleUnitVerse n'est pas seulement un jeu mais derrière celui-ci se cache une véritable histoire, en lien avec notre nouveau monde et les avancées technologiques.

4 Retards, Avances et Difficultés

4.0.1 Adaptation au groupe

Notre premier planning, probablement trop ambitieux, ne prévoyait pas certains problèmes comme l'adaptation à l'environnement de travail de chacun (systèmes d'exploitation, horaires...), les difficultés d'apprentissage ou l'application de nouvelles connaissances. Notre principale difficulté, qui fût donc de respecter des échéances trop courtes est entrain de se résoudre avec la mise en place de nouvelles échéances favorisant l'apprentissage et l'exercice plus équilibré de notre savoir faire, propice à l'épanouissement de chacun dans le développement du projet. Nous pensons donc rattraper notre retard très rapidement.

4.0.2 Avances

Nous avons eu néanmoins des facilités. En effet, nous avons pris beaucoup d'avance pour ce qui est du design et de l'interface du jeu. Nous pensons que c'est une bonne nouvelle, nous enjoignant à poursuivre nos efforts malgré les difficultés. L'interface et le design est la partie la plus importante pour l'utilisateur, c'est avec elle qu'il interagit du début à la fin de la séance de jeu. Elle permet de personnaliser son expérience, de faire des achats etc... Nous sommes donc satisfait d'avoir pris de l'avance dans ce domaine et pensons même en profiter pour le perfectionner davantage. Nous avons également pris beaucoup d'avance en ce qui concerne le multijoueur. Étant en cours d'achèvement, nous sommes heureux de constater notre anticipation quant à ce domaine. Il s'agit, là aussi, d'un des piliers du projet : En effet, BattleUnitVerse est un jeu basé sur le jeu en ligne. C'est également une des parties les plus difficiles à concevoir du jeu, nous sommes donc satisfait des progrès conséquents en la matière. C'est bien la partie du projet dont nous sommes les plus fiers.

5 Conclusion Globale

5.1 Prévisions pour la prochaine soutenance

Nos prévisions sont optimistes. En effet, le multijoueur est déjà en cours d'achèvement. Une fois terminé, il ne restera plus que trois tâches avant de terminer le jeu et enfin clôturer le projet avec la création de publicités et d'un trailer. Malgré nos difficultés passées, nous pensons avoir gagné en expérience et savons dès lors et déjà que nous rattrapons notre retard. Nous pensons terminer totalement le site web ainsi que le multijoueur dans peu de temps. Nous pensons également avoir terminé de développer notre intelligence artificielle avant la prochaine soutenance. En effet, ces deux étapes sont théoriquement préparées et ne manquent plus qu'à être développées.