

RAPPORT DE LA Deuxième SOUTENANCE

**RELATIF A LA CONCEPTION D'UN JEU DE TIR EN 3D
IMPLÉMENTANT UN MODE MULTIJOUEUR EN
RÉSEAU**

“ PATH MAKER”

Paul CREDOZ	E2
Charles-Antoine LEGER	E2
Clément LABBÉ	E2
Pierre-Louis POZZO DI BORGO	E2

SOMMAIRE

I - Introduction	3
II - Réalisations effectuées	4
II.1 - Gameplay	
II.1.1 - <i>Le Player</i>	4
II.1.2 - <i>Déplacement du joueur</i>	6
II.1.3 - <i>Implémentation du tir</i>	6
II.1.4 - <i>Respawn des joueurs</i>	7
II.1.5 - <i>Capture de drapeau</i>	7
II.1.6 - <i>Gestion des points</i>	8
II.2 - Audio	9
II.3 - Mode multijoueur	
II.3.1 - <i>Gestion des lobbies</i>	12
II.3.2 - <i>Gestion des équipes</i>	14
II.4 - Gestion des comptes des joueurs	
II.4.1 - <i>Création ou connection au compte</i>	15
II.4.2 - <i>Hébergement du backend</i>	16
II.5 - Texture	17
II.6 - Modélisation 3D	
II.6.1 - <i>Modélisation des décors</i>	19
II.6.2 - <i>Modélisation des personnages</i>	21
II.7 - Animations	23
II.8 - Site Internet	24
III - Conclusion	24

INTRODUCTION

L'objet relatif à ce rapport s'inscrit dans le projet du second semestre de la première année d'étude à l'EPITA (Ecole Pour L'informatique et les Techniques Avancées). Ce projet a ainsi pour objectif premier de nous faire découvrir les coulisses de la création d'un logiciel moderne et en particulier celle d'un jeu vidéo en trois dimensions implémentant un mode multijoueur en réseau.

Depuis le rendu du cahier des charges, notre détermination n'a pas baissé et les spécifications de notre projet n'ont pas changé.

Ce rapport est celui de la seconde des trois soutenance prévu dans ce projet. Dans ce rapport, on détaille les fonctionnalités qui ont été implémentées depuis le rendu du cahier des charges.

GAMEPLAY

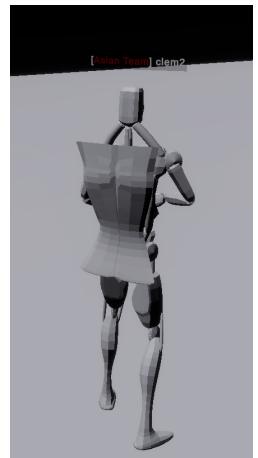
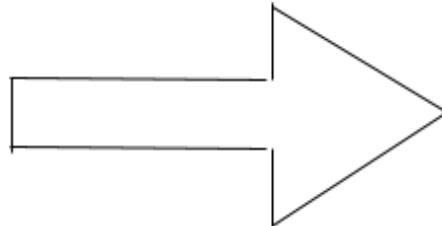
Dans cette partie, on présente les fonctionnalités liées au gameplay qui ont été implémentées.

Le Player

Pour ce qui est de la partie gameplay, Clément s'en est chargé avec l'assistance de Pierre-Louis.

Il a d'abord établi le corps du personnage que nous appellerons durant toutes les explications « Player ».

Lors de la première soutenance, notre player était une capsule avec 2 petits cylindres représentant à la fois les yeux et permettant d'observer la rotation lors des phases de tests.

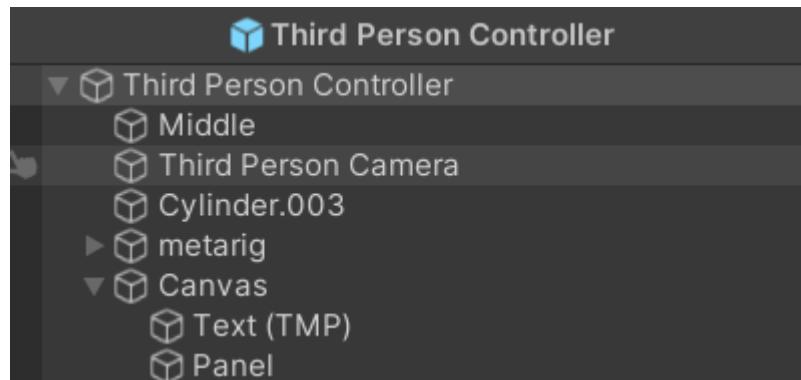


Le corps de notre Player est en réalité bien plus complexe qu'il puisse paraître

Il faut garder à l'esprit que la hitbox de notre player est toujours celle d'une capsule, c'est à dire que le Player a une forme visuelle légèrement différente de celle dans le jeu.

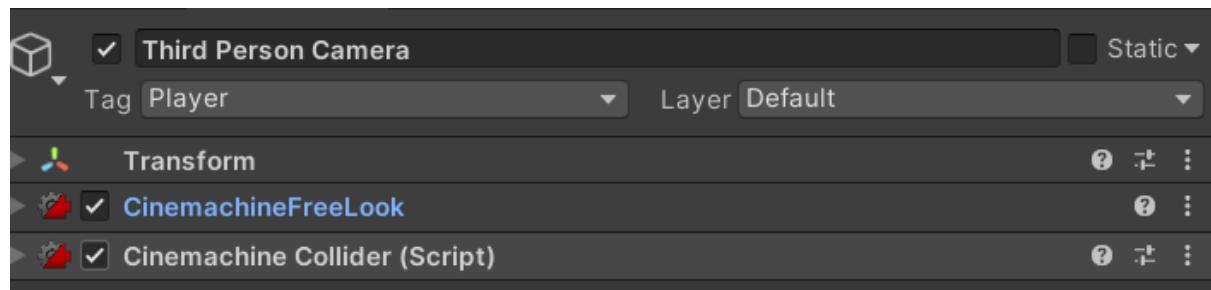
La forme visuelle n'est qu'une apparence, et permet d'ajouter des Animations.

En voici sa composition :



Comme vous pouvez le voir ci-dessus, le Player est composé d'un « empty object » appelé Third person Controller, qui contient tous les éléments de notre Player.

Premièrement, le Player possède une caméra appelée Third Person Camera. Elle a deux composants :



Ces deux composants permettent de contrôler la caméra à l'aide de sa souris et d'éviter tout obstacle entre la caméra et le Player.

Les composants Cylinder.003 et metarig sont présents pour toute la partie visuelle et animation.

Enfin le canvas va permettre d'afficher les pseudonymes des joueurs durant la partie.

Déplacement du joueur

Comme dans tout jeux vidéo le Player doit évoluer dans le jeu donc se déplacer et interagir avec son environnement.

On a donc ajouté un script permettant de se déplacer dans toutes les directions sur le plan (Oxz) de unity, en d'autres termes sur le plan horizontal. Nous n'avons pas encore réalisé de déplacement sur l'axe Y (sur l'axe Vertical) comme par exemple sauter ou s'accroupir.

Le joueur (personnage contrôlant le Player) déplacera donc le Player avec les touches classiques de Unity en QWERTY soit W,Q,S et D. ce déplacement respectivement en avant, à gauche, en arrière et à droite.

Le Player peut aussi courir lors de l'appui sur la touche shift, et sa vitesse va donc augmentée.

Implémentation du tir

Dans un TPS le joueur doit être capable de tirer sur les autres joueurs, nous avons donc implémenté ce système mais à notre sauce.

```
if (Input.GetButtonDown("Fire1"))
{
    if (IsAiming())
    {
        delta = 10f;
        Vector2 screenCenterPoint = new Vector2(Screen.width / 2f, Screen.height / 2f);
        Ray ray = Camera.main.ScreenPointToRay(screenCenterPoint);
        ray.origin = ray.GetPoint(delta);
        if (IsLocalPlayer)
        {
            ShootServerRpc(ray);
        }
    }
}
```

```
[ServerRpc]
void ShootServerRpc(Ray ray)
{
    if (Physics.Raycast(ray, out RaycastHit raycastHit, 999f, aimColliderLayerMask))
    {
        //Raycast hit something
        var enemyHealth = raycastHit.transform.GetComponent<PlayerHealth>();
        if (enemyHealth != null)
        {
            enemyHealth.TakeDamage(50);
        }
    }
    ShootClientRpc(ray);
}
```

Respawn des joueurs

En effet, dans un jeu les joueurs se tirent dessus et par conséquent meurent.

Nous devons donc implémenter à la fois un système de répartition. C'est-à-dire lorsque un joueur meurt il réapparaît à un endroit donné en fonction de son équipe.

Par ailleurs nous devons aussi implémenté un système d'apparition car les joueurs lors de leur connexion ne sont jamais morts il faut donc gérer ce cas.

```
public class Spawner : NetworkBehaviour
{
    [SerializeField] private GameObject hostPrefab;
    [SerializeField] private GameObject clientPrefab;
    public override void OnNetworkSpawn()
    {
        if (IsHost)
        {
            var host = Instantiate(hostPrefab, Vector3.zero, Quaternion.identity);
            host.GetComponent<NetworkObject>().SpawnAsPlayerObject(NetworkManager.Singleton.LocalClientId);
        }
        else
        {
            SpawnServerRpc(NetworkManager.Singleton.LocalClientId);
        }
    }

    [ServerRpc]
    public void SpawnServerRpc(ulong clientId)
    {
        var client = Instantiate(clientPrefab, Vector3.zero, Quaternion.identity);
        client.GetComponent<NetworkObject>().SpawnAsPlayerObject(clientId);
    }
}
```

Capture de drapeau

Notre jeu sera basé sur la capture de drapeau pour gagner la partie. Pour cela le Player doit aller chercher un drapeau si situant dans la base des adversaires. Une fois récupéré, le Player doit ramener le drapeau dans sa propre base. Une fois cette mission accomplie, l'équipe ayant ramené le drapeau dans sa base gagne des points.

```

void OnCollisionEnter(Collision collision)
{
    var teamStateObj = gameObject.GetComponent<TeamLogic>();
    m_teamState = teamStateObj.playerNetworkTeam.Value;
    // Debug.Log($"player is in {m_teamState}");
    if (collision.gameObject.tag == "AsianFlag" && (m_teamState == TeamState.GreekTeam) && !HasFlag)
    {
        HasFlag = true;
        collision.gameObject.SetActive(false);
    }
    else if (collision.gameObject.tag == "GreekFlag" && (m_teamState == TeamState.AsianTeam) && !HasFlag)
    {
        HasFlag = true;
        collision.gameObject.SetActive(false);
    }
}

```

Gestion des points

La gestion des points est importante pour finir la partie et désigner une équipe vainqueure.

Nous pouvons observer dans le HUD (c'est l'ensemble des informations présentes sur l'écran)

Le Player ici présent vient de récupérer un drapeau son équipe gagne donc 3 points et gagne la manche

On peut voir le score personnel du joueur en bas à gauche

Au milieu en haut se situe le temps restant de la manche.



Audio

Un jeu est naturellement composé d'une ambiance sonore, qui lui est propre. Par exemple dans un jeu de style FPS lorsque les joueurs se déplace ils émettent un son propre au bruit de pas, pareillement pour le saut, ou encore sur Rainbow six siege il existe un bruit lorsque le joueur s'allonge et s'accroupit, ce bruit est similaire à un frottement entre des habits.

Pour l'instant notre jeu n'est composé que d'un son de saut, de déplacement, de tir (lorsque le Player fait apparaître la balle devant lui) et un son d'accélération (lorsque le joueur appuis sur le bouton droit de la souris, la balle s'accélère et cela émet un son).

```
public AudioSource jumpSound;  
public AudioSource walkSound;  
public AudioSource fireSound;  
public AudioSource accelerationSound;
```

voici les quatre sons implémentés pour l'instant

- jumpSound pour le son émis par le saut
- walkSound pour le son émis par le déplacement horizontal du Player
- fireSound pour l'apparition de la balle
- accelerationSound pour l'accélération de la balle

voici une partie du script permettant de sauté et émettre le son

```
//application de la gravité  
ySpeed += Physics.gravity.y * Time.deltaTime;  
  
if (characterController.isGrounded)  
{  
    ySpeed = 0f;  
    if (Input.GetKeyDown(KeyCode.Space))  
    {  
        Debug.Log("jump");  
        walkSound.Stop();  
        jumpSound.Play();  
  
        ySpeed = jumpSpeed;  
    }  
}
```

voici la partie du script permettant d'émettre le son lors du déplacement du Player

```
if (Input.GetKeyDown(KeyCode.Z) && characterController.velocity.magnitude > 0)
{
    walkSound.Play();
    Debug.Log("walkSound");
}
if (characterController.velocity.magnitude == 0)
{
    walkSound.Stop();
}
```

voici la partie du script permettant d'émettre le son lors du clic gauche et droit

```
if (Input.GetMouseButton(0))
{
    fireSound.Play();
    Debug.Log("Fire !");
}
if (Input.GetMouseButton(1))
{
    accelerationSound.Play();
    Debug.Log("pffffiooo");
}
```

Les Debug.Log permettent de tester l'entrée dans les conditions et comprendre certains bugs.

Mode multijoueur

Depuis la dernière soutenance, nous nous sommes penchés tout spécialement sur la question du multijoueur dans notre jeu. Nous avons fait des erreurs que nous corrigé au fur et à mesure ce qui nous permet aujourd’hui de disposer d’un jeu offrant la possibilité aux joueurs de créer puis rejoindre des salons de connections (appelés lobbies dans la suite de ce rapport), de connecter ces joueurs via internet et de pouvoir, évidemment les faire évoluer dans un environnement synchronisé complexe, c'est à dire disposant de fonctionnalités comme la réapparition des joueurs, la synchronisation de variables entre tous les joueurs, la synchronisation de la position des joueurs ainsi que de la synchronisation des animations liées aux joueurs ainsi qu'à des objets de l'environnement.

Nous voulons aussi préciser que nous avons implémenté seul c'est à dire sans l'aide de plugin (comprenant des composants prêt à l'emploi) comme Photon Pun, UNet ou bien Mirror mais bien avec Netcode for GameObject (anciennement MLAPI), qui est la librairie officielle bas niveau de Unity.

Pour la partie lobby, nous nous appuyons sur le service officiel Lobby d'Unity mettant à notre disposition des api pour créer, rejoindre et détruire des lobbies. Cette librairie étant de bas niveau, il a été possible de customiser de façon approfondie les fonctionnalités de nos lobbies, nous avons en effet implémenté un système de choix des équipes dans les lobbies permettant aux joueurs de s'accorder sur les équipes avant le début de la partie.

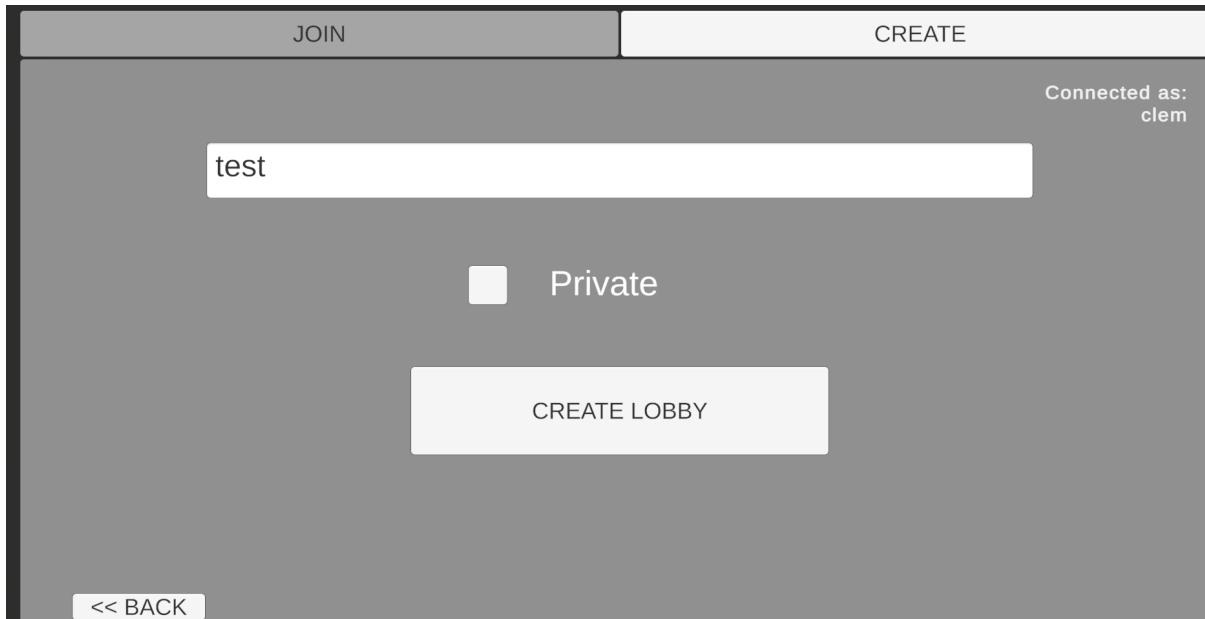
Path Maker permet aux joueurs de jouer via internet (et non pas seulement sur le même réseau). Cela est possible grâce au service Relay d'Unity qui se comporte comme un intermédiaire entre le joueur hébergeant la partie en cours et les autres joueurs se connectant à elle et se trouvant sur un autre réseau. Cette partie du développement du mode multijoueur a été la source de beaucoup de problèmes divers. Après plusieurs semaines de débogage et de documentation, cette partie fonctionne désormais.

Enfin, la dernière partie du développement du mode multijoueur de Path Maker se trouve être celle de la synchronisation états des clients entre eux. C'est dans cette partie que la librairie Netcode for GameObject est utilisée, elle permet en effet de pouvoir synchroniser les états de tous les joueurs entre eux. La procédure est la suivante: lorsque les clients veulent effectuer une action dans le jeu: déplacement, animation ou bien tir, un RPC (remote procedure call) est alors envoyé au serveur qui se charge non seulement d'effectuer l'action demandée mais aussi qui envoie le nouvel état de la partie à tous les autres clients. Nous avons pris du temps pour comprendre concrètement comment ce schéma peut être implémenté en programmation, encore une fois après de longues heures sur la documentation de la librairie nous avons pu aboutir à une version basique mais stable et surtout fonctionnelle implémentant les fonctionnalités détaillées dans le cahier des charges.

Gestion des lobbies

Lors du lancement du jeu pour pouvoir jouer avec autrui les joueurs ont la possibilité de créer un Lobby ou d'en rejoindre.

Certains lobby peuvent être privé dans ce cas pour rejoindre le lobby les joueurs doivent inscrire le code du lobby.



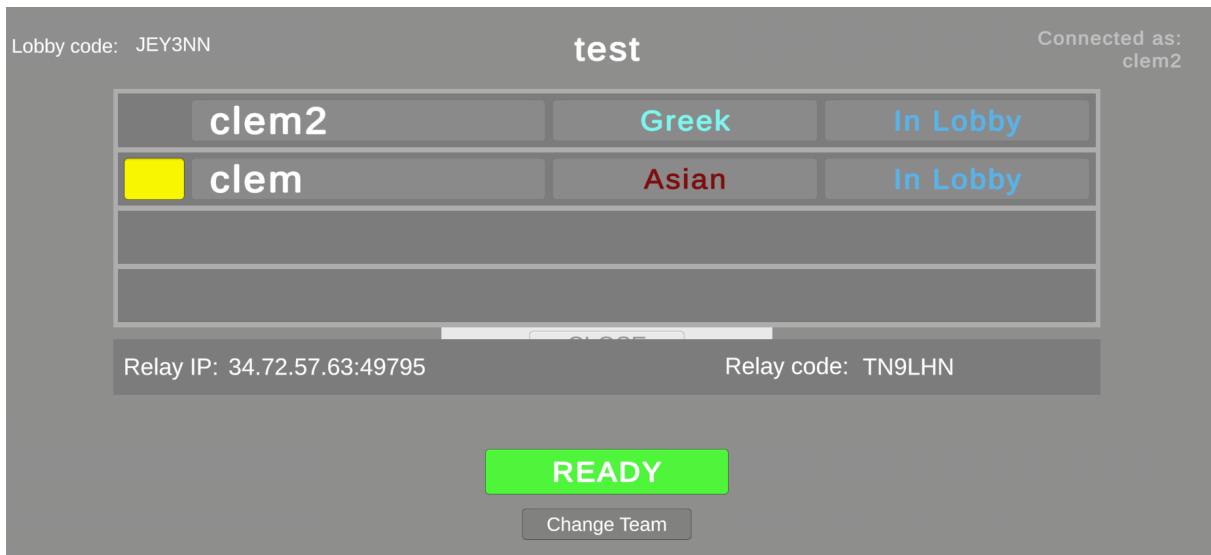
En effet pour un peu plus de concret voici la présentation de la création de lobbies.

Pour créer un lobby rien de plus simple, il suffit d'écrire le nom que vous voulez donner à votre lobby, ici nous avons décidé de donner comme nom “test”.

Ensuite vous devez choisir si le lobby que vous êtes en train de créer va être privé ou public. Pour cela il faut sélectionner la case private si votre lobby doit être privé, dans le cas contraire il faut décocher cette case, comme dans l'exemple ci dessus.

Pour finir, on cliquera sur “Create Lobby” pour créer le lobby.

Une fois le lobby créé nous arrivons sur cette page :



On peut observer plusieurs éléments très importants qui ont été implémentés :

Le nom du lobby est noté en haut au centre, ici "test". Par ailleurs, nous pouvons voir les différentes personnes connectées.

Ici on peut voir l'host avec le carré jaune soit "clem" et un second joueur clem2. Dans la case à droite "In Lobby" est affiché le statut qui peut être "Connecting", lorsque le joueur est en train de se connecter, "In Lobby" lorsque le joueur attend les autres joueurs et ensuite "Ready" lorsque le joueur appuie sur le bouton vert prévu à cet effet "READY".

Quand tous les joueurs du Lobby sont prêts, la partie se lance.

Gestion des équipe

Les parties de notre jeu se disputent en équipe de 2,d'où les 4 places dans le lobby, nous avons donc créé deux équipes. La première s'appelle les grecques ils sont de la couleur bleu pour rappeler les toitures bleu en grèce. La seconde équipe est les asiatiques en rouge, rappelant la couleur de l'architecture japonaise et des portes rouges.

Comme vous avez pu le voir précédemment le changement des équipes dans le Lobby est assez simple, il suffit de cliquer sur le bouton “change team” et l'équipe à droite du pseudo du joueur se changera automatiquement.



Gestion des comptes des joueurs

Lors du lancement du jeu, nous arrivons sur cette page, nous permettant de nous connecter avec “LOGIN” soit de quitter le jeu avec le bouton “EXIT”.



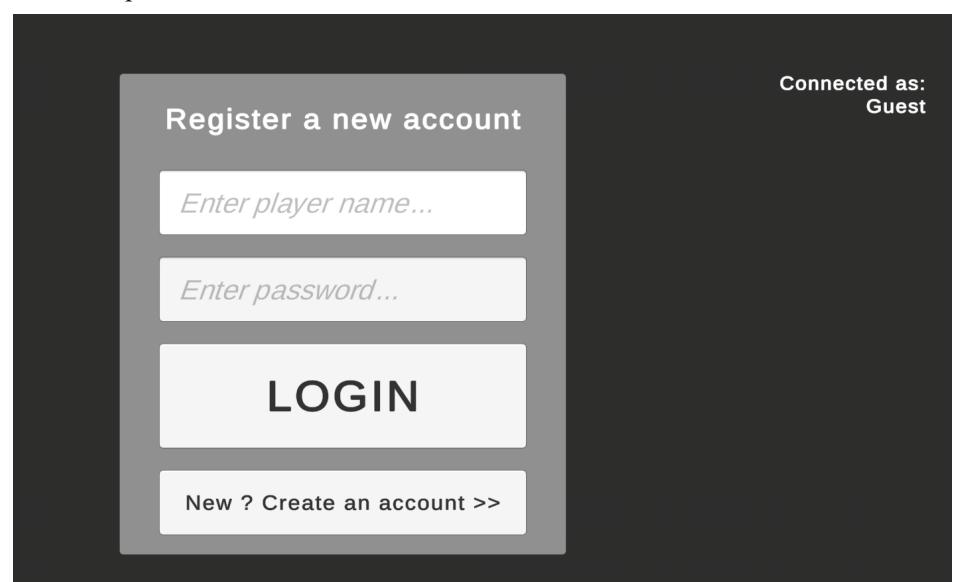
Création ou connection au compte

Par ailleurs, on peut observer en haut à droite sur quel compte vous êtes connectés, ici nous ne sommes pas connectés car on voit le pseudo “Guest” par ailleurs le bouton “LOGOUT” n'est pas présent ce qui signifie que nous ne sommes pas connectés.

A présent, connectons nous !

Nous venons de cliquer sur le bouton “LOGIN”. Nous avons 2 possibilité :

- se connecter
- s'enregistrer (dans le cas où nous n'avons pas de compte)



Une fois connecté, nous sommes redirigés vers une page où nous pouvons trouver une partie



Cette page est assez similaire à la page du lancement du jeu, excepté le fait que nous n'avons plus de bouton “LOGIN” mais plutôt “LOGOUT” et que le bouton “PLAY” s'est affiché.

Une fois le bouton “PLAY” pressé, le jeu nous redirigera vers la sélection ou création des lobbies joignables.

Hébergement du serveur backend

Dans le dernier rapport de soutenance ainsi que dans le cahier des charges relatifs à ce projet, il était question de l'hébergeur du serveur backend s'occupant de l'authentification des joueurs ainsi que de la sauvegarde de leurs informations personnelles (noms, scores etc). Nous avions décidé à l'époque de choisir Vercel, un hébergeur gratuit et simple d'utilisation. Cependant, suite à une récente mise à jour du service que nous comptions utiliser dans notre jeu, nous avons dû changer d'hébergeur et en choisir un qui satisfaisait nos demandes (coûts, simplicité d'utilisation). Après plusieurs recherches, nous avons décidé de choisir Heroku: une service gratuit et très facile d'accès, offrant la possibilité d'héberger des serveurs Node.js comme le backend de Path Maker et de les lier facilement à des bases de données comme la base MongoDB, que nous utilisons pour sauvegarder les informations de nos joueurs.

Textures

Une très grande partie des textures ont été réalisées, elles suivent parfaitement le style que nous avons décidé d'utiliser le “Low poly”. Ce genre de modélisation simpliste mais réaliste s'accompagne de textures très coloré et unies (voir exemple dans : Modélisation 3D). Toutes les textures ont été réalisées avec le logiciel blender, ce logiciel est très pratique puisqu'on peut modéliser les différents éléments du jeu puis directement leur rajouter une texture. Le seul inconvénient c'est que certaine texture ne sont pas compatible avec l'export en *.fbx* que nous utilisons pour les rajouter dans notre projet unity. Cela n'a pas posé de réel problème, mais juste une grosse déception lorsque nous avons découvert que les textures de type "émission" qui imitent très bien les néons avec une légère lumière qui est émise n'était pas compatible avec unity. Cela aurait rajouté un gros plus à nos design futuriste pour les personnages et les armes.

Mis à part cette déception, la création de texture s'est extrêmement bien passée, nous avons pu avoir des textures qui correspondent bien au “Low poly”. Nous n'avons également pas joué sur la réflexion de la lumière pour augmenter le réalisme de certains éléments.



Sur l'image précédente on peut voir que la texture grise reflète la lumière afin de simuler une texture métallique. Alors que la texture marron ne reflète rien, ce qui ressemble plus à du bois. Avec seulement deux textures très basique on peut rendre ce revolver assez réaliste.

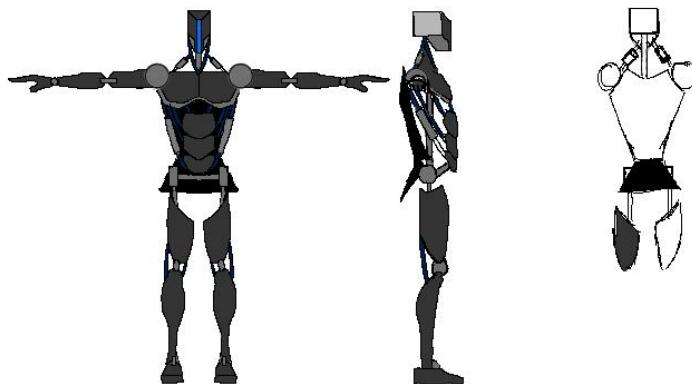
Modélisation 3D

Les différents designs des objets, des personnages et de l'interfaces sont imaginés par la totalité des membres de l'équipe, dessinés par Pierre-Louis pour avoir une bonne idée de ce à quoi ils ressembleront.

Les dessins sont réalisés soit sur papier soit avec une tablette graphique sur le logiciel Clip Studio Paint.

Ensuite, ces différents dessins sont modélisés en 3D grâce à l'outil Blender par Paul et Pierre-Louis, nous nous sommes formés grâce à des tutoriels sur Youtube et en découvrant nous même les différents outils de modélisation en créant les modèles 3D du jeu.

Dessins préparation à l'élaboration d'un des personnages.



Voici un aperçu du terrain avec les différents objets et bâtiments réunis:

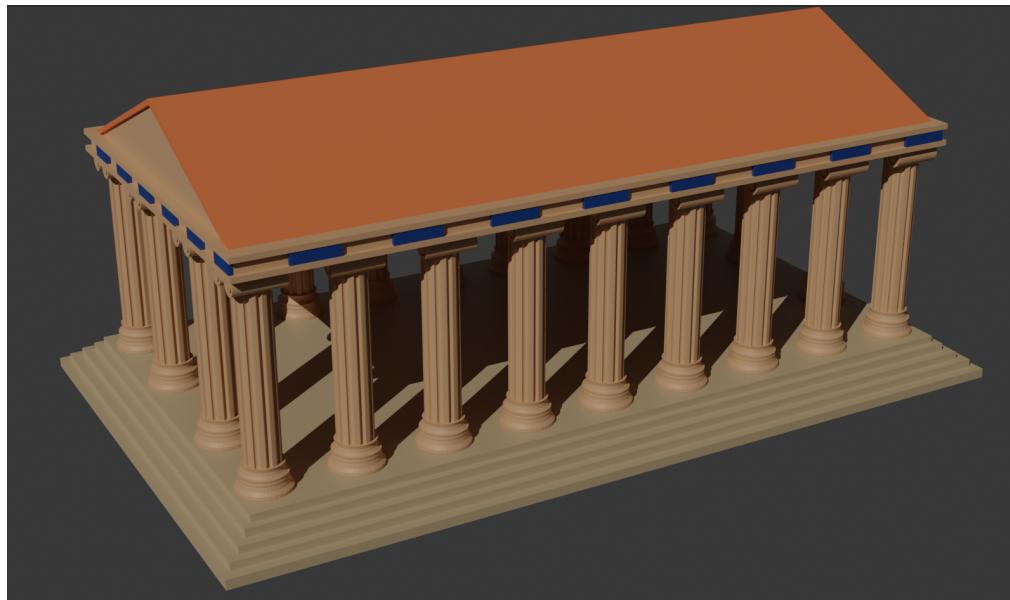


Modélisation des décors

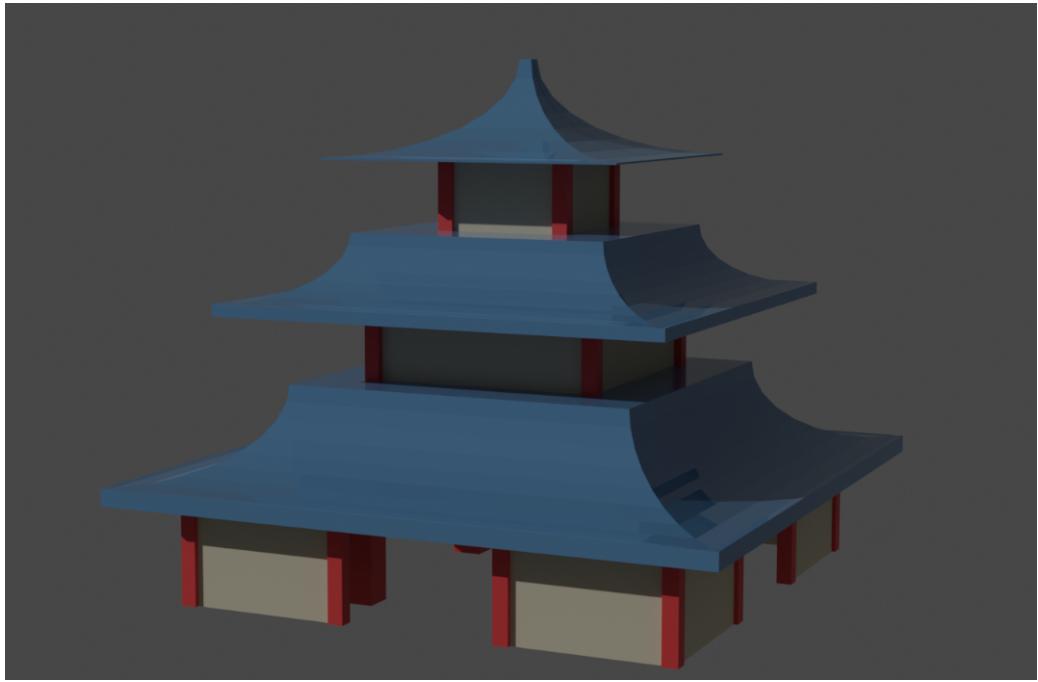
Les décors sont inspirés de différents styles architecturaux tels que d'un côté l'Asie médiévale et de l'autre la Rome et la Grèce antique.

Il y a deux sortes de décors, les décors qui ont une utilité pour le gameplay (les bâtiments, manivelles et ponts) et les décors qui ont un but uniquement esthétique (arbres, rochers, lampes et poutres).

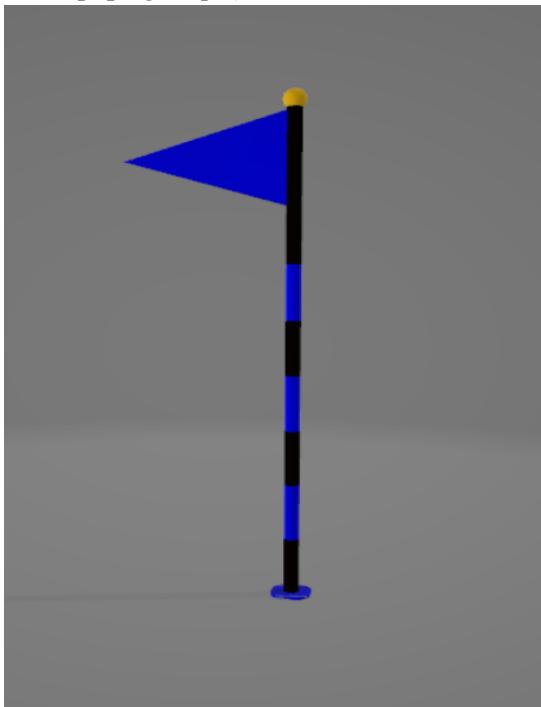
Temple de style grecque:



Temple de style Asiatique:



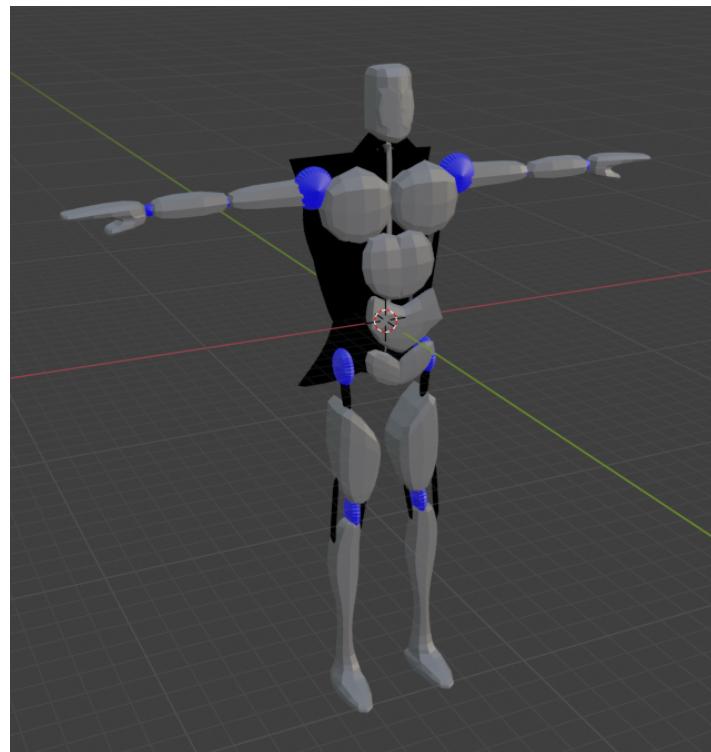
Ces temples contiennent les drapeaux des deux équipes.(drapeau rouge pour l'équipe asiatique et bleu pour l'équipe grecque).



Modélisation des personnages et armes:

Il y a deux personnages, le premier a été créé d'après le dessin présenté en introduction de la partie modélisation.

Voici le premier personnage:



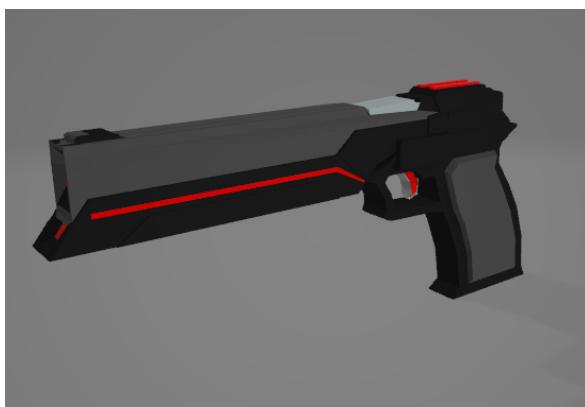
Et voici le deuxième:

Notre jeu est un jeu de tir, il y a donc des armes, que nous avons modélisé nous même.

Les armes ne sont évidemment pas inspirées des styles grecque et asiatique car dans l'antiquité les armes à feu n'existaient pas.



Voici les trois armes:

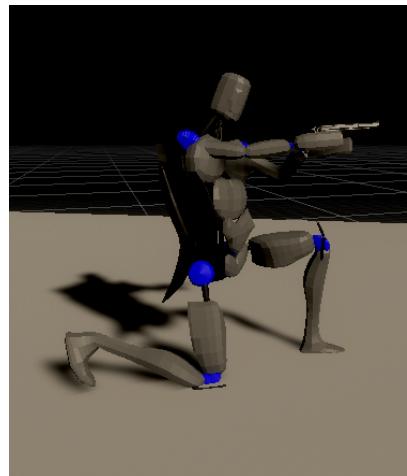
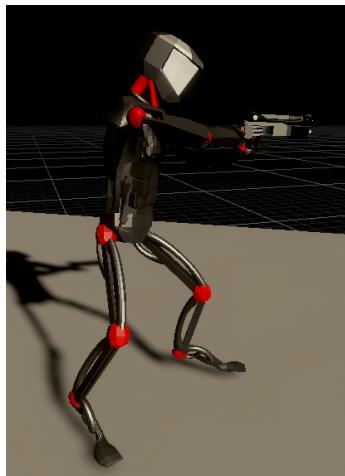


Animations

Les animations ont légèrement évolué depuis la première soutenance, au départ nous avions choisis des animations pour que le personnage porte une arme à deux mains. Nous avons fait ce choix au début parce que ce type d'animation était beaucoup présente sur le site internet mixamo.com ce qui nous facilitait donc la tâche, les animations étant assez complexes et longues à réaliser.

Mais finalement nous avons modélisé des armes à une seule main, ce qui allait mieux avec l'univers de notre jeu.

A la suite de ce choix, nous n'avons pas pu disposer le l'entièreté des animations avec un personnage qui tient son arme à une main. Nous avons donc dû en modifier certaines afin que sa posture soit plus cohérente. Mais le résultat n'est pas encore parfait, nous allons donc devoir retravailler certaines animations afin de ne plus avoir de problème.



Nous avons également modifier le script et l' animator sur unity qui permettent de gérer les animations des personnages. Nous les avons juste adaptés au multijoueur, mais nous avons la structure principale.

Site Internet

Le site internet a été réalisé par Paul en HTML, CSS et JS à l'aide à l'aide du logiciel Bootstrap. Il est actuellement hébergé sur Netlify, voici le lien : <https://pathmaker-ygreg.netlify.app/>

Le site étant hébergé sur une plateforme gratuite, nous avons décidé d'acheter un nom de domaine, un autre site internet est donc disponible : <http://path-maker.fr/>

La page d'accueil possède une rapide présentation du jeu ainsi que la présentation de l'équipe. Chaque membre possède sa propre page avec sa présentation.

On retrouve également un lien vers la page de présentation du projet, avec sa nature et ses origines.

Dans la dernière partie il y a les liens vers les rapports de la première et seconde soutenances et le cahier des charges.

Conclusion

En conclusion de ce rapport, on rappelle ce qui doit être fait pour la prochaine soutenance. Notre groupe étant dans les temps, ce planning n'a pas changé depuis celui rendu dans le cahier des charges.

	Soutenance 1	Soutenance 2	Soutenance 3
Gameplay	33%	66%	100%
User Interface	0%	50%	100%
Modélisation	33%	66%	100%
Texture	33%	66%	100%
Audio	0%	50%	100%
IA	0%	50%	100%
Mode multijoueur	60%	90%	100%
Gestion de compte	40%	80%	100%
Site Internet	42%	84%	100%
Marketing	0%	0%	100%

