E.S.U.

Emergency Service Unit

Rapport de soutenance 1

Louis Place
Maxence Oden
Nathan Rabet
Vincent Libeskind

Contents

1 Introduction

Cette première période avant la première soutenance a permis de développer les bases de notre jeu vidéo et de prendre en main les outils que nous utilisons dans le cadre du projet : à savoir Unity et Git, avec Github.

Ce rapport de soutenance présentera les fonctionnalités décrites dans le cahier des charges et notre avancement sur ces dernières, avec le rôle de chaque membre du groupe dans leurs développements. Le travail qui a été fait jusqu'alors a pour principal objectif de donner une base concrète du jeu vidéo que nous imaginons, sur laquelle nous serons en mesure d'ajouter plus facilement de nouvelles fonctionnalités et améliorer celles existantes.

2 Mise au point du cahier des charges

Lors de la mise en place des outils de gestion du projets, nous nous sommes rendu compte qu'il était peu intéréssant d'utiliser "asana.com" étant donné que nous avons à notre disposition un outil totalement intégré avec GitHub, de ce fait, nous avons simplement décidé de migrer de "asana.com" vers l'outil de gestion de projet intégré à Github.

3 Répartition des tâches

Tâches	Louis	Maxence	Nathan	Vincent
Multijoueur		X		
Personnages			X	
Animation				X
Site web	X			
Menu		X		
Post-processing				X
Cartes	X			X

4 Avancements du projet

4.1 Développement du multijoueur

Pour le multi-joueurs, nous avons choisi d'utiliser la bibliothèque Photon afin de faciliter sa conception. Un des grands avantages de Photon est qu'il possède ses propres serveurs ce qui permet de simplifier la gestion des salles de joueurs.

4.1.1 Les salles

Nous avons utilisé Photon PUN (Photon Unity Networking) pour avoir accès aux serveurs de Photon. Nous avons pris la version gratuite qui nous donne la possibilité d'avoir 20 clients connectés sur le serveur maître en simultané. Au-delà de ce nombre de joueurs, les autres clients ne pourront pas se connecter aux serveurs. Voici notre code de connexion des clients:

```
public override void OnConnectedToMaster() //Si connecté au serveur maitre de Photon
{
    PhotonNetwork.JoinLobby (TypedLobby.Default); //Rejoindre le lobby
    ClientState.text = PhotonNetwork.NetworkClientState.ToString(); //Affichage
}


public override void OnJoinedLobby() //Si on rentre dans un lobby
{
        RoomOptions myRoomOption = new RoomOptions(); //Création de la var de room
        myRoomOption.MaxPlayers = 20; //20 joueurs max

        PhotonNetwork.JoinOrCreateRoom ("OfficialRoom", myRoomOption, TypedLobby.Default); //Essai de rejoindre la partie en crée une sinon
        ClientState.text = PhotonNetwork.NetworkClientState.ToString(); //Affichage
}

public override void OnJoinedRoom () //Si partie rejoind ou crée
{
    ClientState.text = PhotonNetwork.NetworkClientState.ToString(); //Affichage
    gameManager.GetComponent<GameManagerScript>().IsConnected(); //Appel de la fonction IsConnected de GameManager
}
```

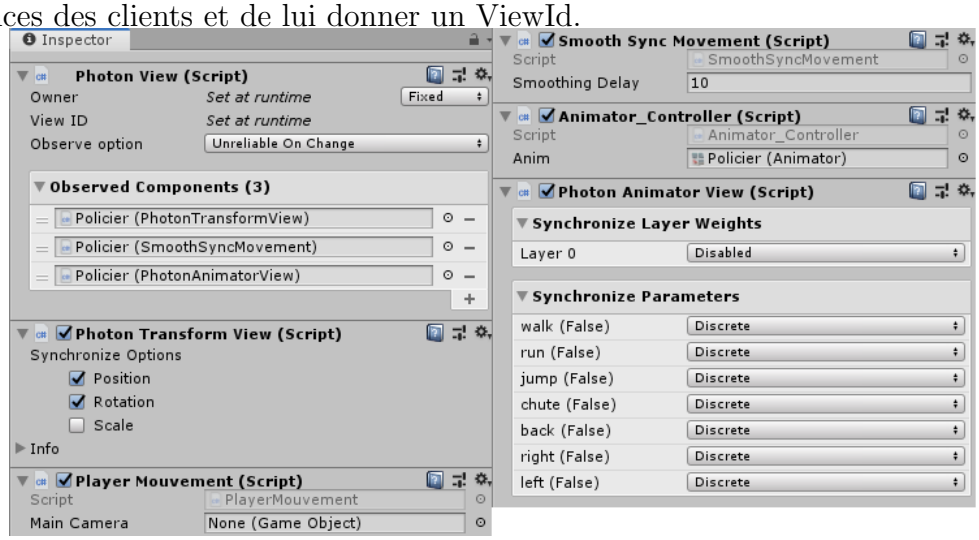
Comme on peut le voir sur le code au-dessus, nous avons choisi de réunir tous les joueurs dans une unique salle. Il y n'aura donc qu'une seule partie à la fois constituer de tous les joueurs actuel de notre jeu. Donc, dès que l'utilisateur va demander de jouer en multijoueurs, le script va essayer de rejoindre la salle "OfficialRoom". Si d'autres personnes jouent en multijoueurs, il va les rejoindres. sinon il va créer la salle "OfficialRoom" et en devenir le maître.

4.1.2 Les joueurs

Une fois dans la partie, on a le choix entre les deux équipes. Pour l'instant, quand on choisit son équipe cela nous fait apparaître dans la carte en tant que policier grâce à cette fonction:

```
public void SpawnPlayer() //Function de création de joueur
{
    if (PhotonNetwork.NetworkClientState.ToString() == "Joined") //Si on est en partie
    {
        //
        //  Ajout d'un switch en fonction de la classe choisit (Rajout en paramètre de cette Fonction)
        //
        GameObject MyPlayer = PhotonNetwork.Instantiate(PrefabPlayer.name,SpawnPoint.position,Quaternion.identity,0) as GameObject; //Instantier le prefab
        MainCamera.GetComponent<TPSCamera>().lookAt = MyPlayer.transform.Find("posCam").transform; //Set de la var lookAt de la cam
    }
}
```

Elle permet, lors de son appel d'instancier le joueur dans la partie. Instancier permet de faire apparaître le prefab du joueur dans toutes les instances des clients et de lui donner un ViewId.



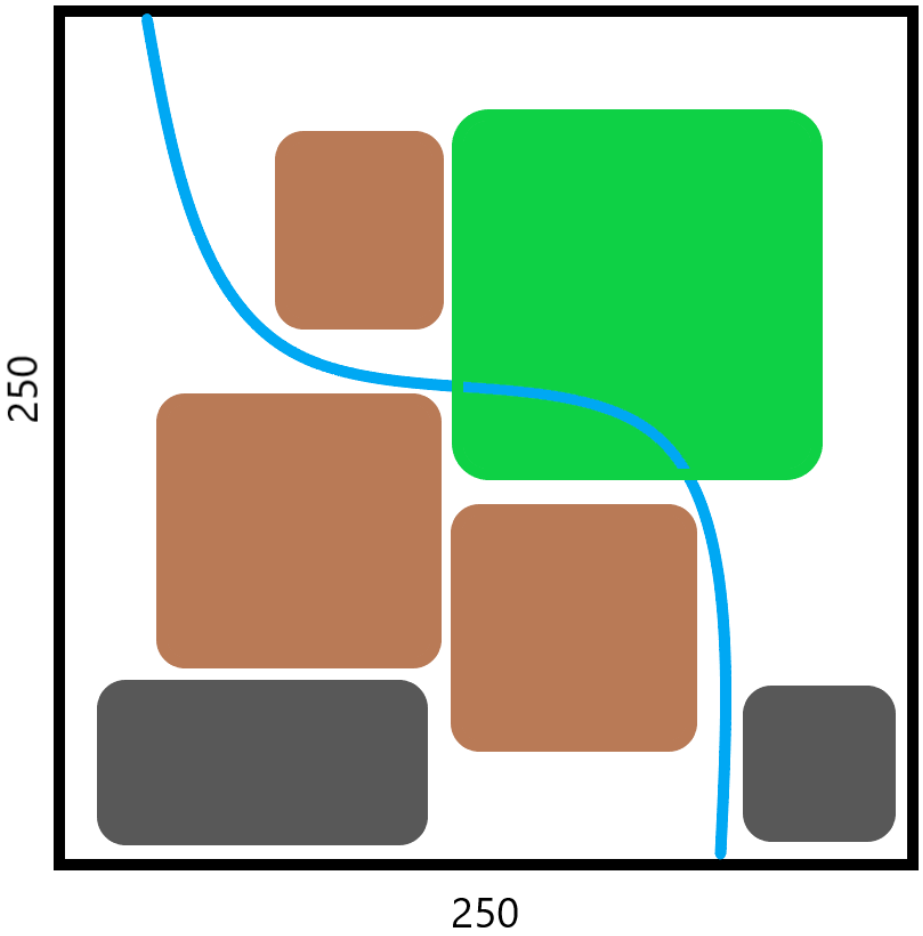
Ce ViewId permet de reconnaître les différents prefabs de chaque client. Par exemple, quand l'utilisateur appuie sur la touche pour avancer, on teste si le prefab est le nôtre avant de le déplacer avec la fonction view.IsMine. Cela permet d'éviter de bouger tous les personnages du serveur. Le déplacement des personnages est le même pour toutes les classes. On a la possibilité d'avancer, reculer, décale, sauter et courir. Chaque possibilité de mouvement a une animation qui est synchronisée à chaque changement d'état. La position et la rotation du joueur sont synchronisées à chaque déplacement.

4.2 Design des cartes

4.2.1 Introduction

Avant toute création de cartes, nous avons réfléchi à la taille de nos cartes : l'objectif est de réaliser des cartes ni trop grandes, ni trop petites. Notre jeu est basé sur des cartes fermées qui doivent être suffisamment petites pour ne pas déclencher un manque d'action et empêcher le bon déroulement de la partie. De plus, elles doivent être suffisamment grandes pour proposer une bonne expérience de jeu, avec différents secteurs à explorer. On ne souhaite pas réaliser des cartes juste pour pouvoir y jouer dessus, mais également pour créer des mondes à parts, proposant des ambiances et des paysages différents et uniques.

4.2.2 Carte 1



Avant de débuter la conception de la carte dans Unity, il fallait faire un schéma afin de décider le type de la carte, les emplacements des différentes zones et son échelle. La carte est donc composée de trois zones principales :

- Zone verte : le parc qui possède un lac, des arbres ainsi que de la verdure.
- Zone grise : les espaces possédant du relief, ce sont des emplacements éloignés du centre de la carte qui sont des endroits stratégiques.
- Zone marron : l'espace urbain, qui possède les zones d'habitation et les

usines.

La carte vue de haut ressemble donc à cela:

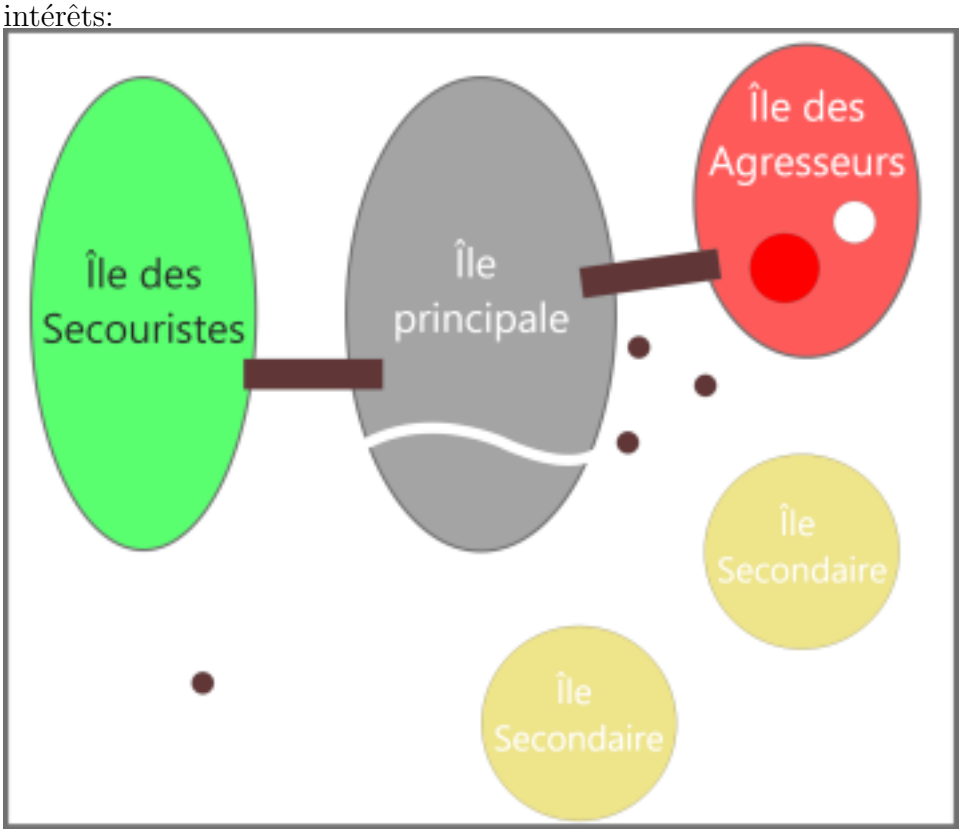
La présence d'une rivière, coupant la carte en deux, est un aspect important du gameplay : elle oblige les joueurs à utiliser des passages stratégiques comme les deux ponts situés aux entrés nord et est de la ville. Le lac, en plus de l'aspect esthétique, pourra devenir un élément important de la partie : c'est un lieu potentiel où les pompiers pourront venir se réapprovisionner en eau. De plus, il oblige les joueurs à faire un détour pour pouvoir accéder à l'autre partie de la carte.

Une des particularités de cette carte est son design simple : elle n'est pas surchargée, est la zone parc possède quelques arbres sans trop de décors.

4.2.3 Carte 2

Pour la seconde carte du jeu nous avons souhaité implémenter des nouveautés en terme de gameplay.

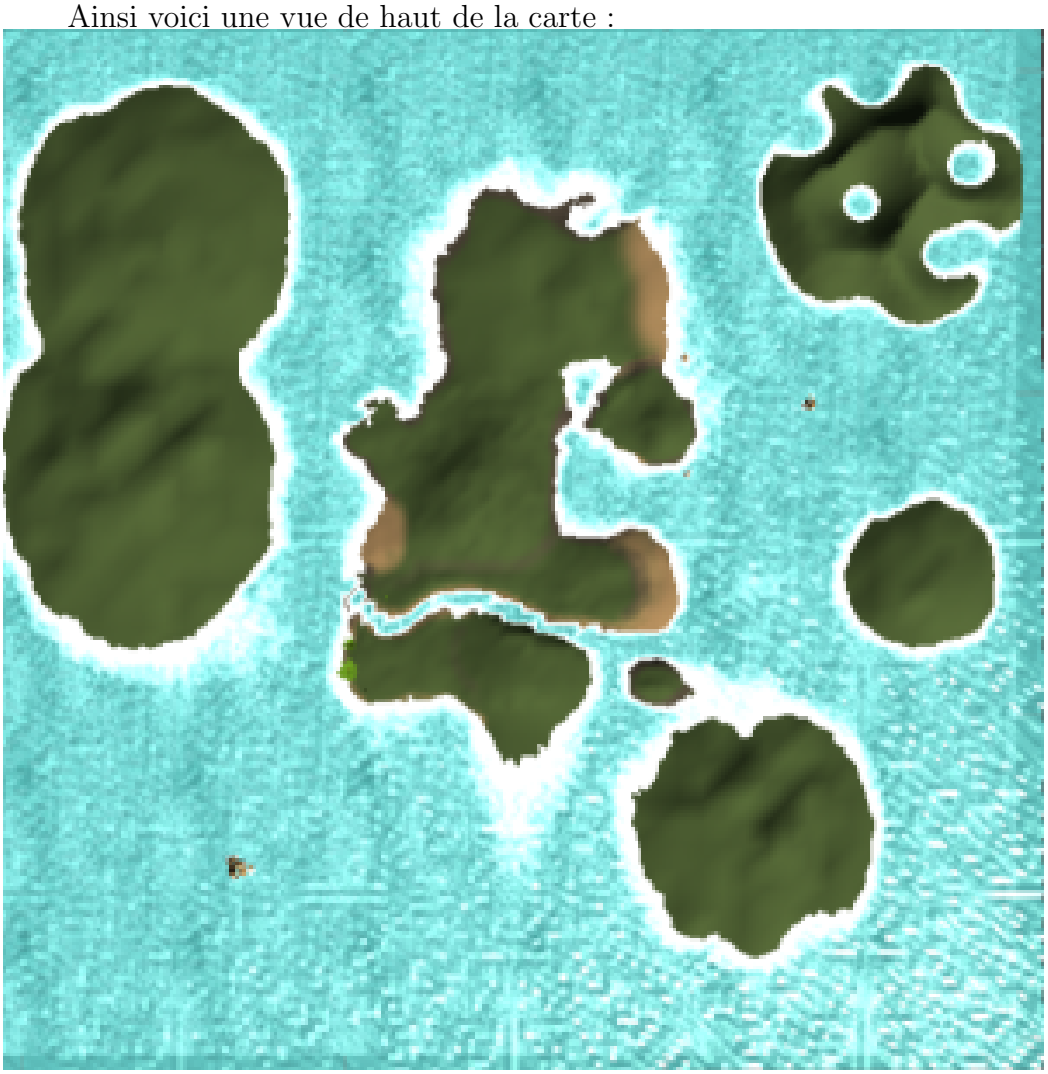
En effet la carte se décompose en plusieurs îles ayant chacune leurs intérêts:



La carte est constituée de plusieurs îles avec pour chacune une utilité propre. A terme la carte sera ainsi décomposée :

- Île du Volcan qui sera la base des criminels
- Île des secouristes à l'extrême opposée de l'île des criminels
- Île centrale et aussi l'île principale où on retrouvera la ville qui sera le véritable cœur d'action de cette carte avec des bâtiments à détruire et des victimes à sauver.
- Les îles adjacentes seront destinées à abriter des bonus et power-up pour les joueurs.

Ainsi voici une vue de haut de la carte :



Par ailleurs le volcan sera pour la classe des pyromanes, un point de réapprovisionnement, par analogie à la rivière pour les pompiers, dans la première carte.

Donc à terme la carte concentrera plusieurs zones stratégiques pour les deux équipes qui s'affrontent. Celles-ci seront localisées en majorité sur l'île principale mais également sur les îles adjacentes où se trouveront des pouvoirs spéciaux qui pourraient permettre de renverser le cours de la partie. On notera également que lorsqu'un joueur voudra passer d'une île à une autres il devra traverser l'océan et sera ainsi à découvert. A terme seul deux ponts

seront présents sur la carte, ils relieront l'île principale à l'île des secouristes et à l'île des criminels et représenteront là encore des points de contrôle stratégiques.

4.3 Réalisation des cartes

Unity intègre un composant terrain, qui permet de nombreuses choses :

- Ajout de détails (herbes, fleurs, arbustes)
- Ajout d'arbres
- Ajout de textures
- Modélisation en temps réel de la carte

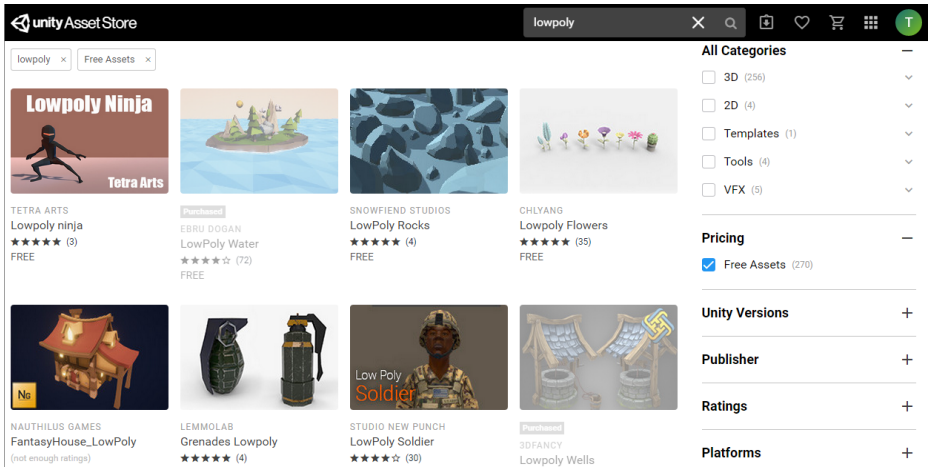
Cela permet d'éditer, de modifier simplement la carte à l'aide d'une brosse.

4.3.1 PolyWorld

Il s'agit d'un asset trouvé sur l'asset store et qui nous a permis de faciliter la conception des cartes en lowpoly. En effet on y retrouve plusieurs fonctionnalités intéressantes. Tout d'abord on a la possibilité de convertir n'importe quel asset normal en asset de style lowpoly. On retrouve également cette fonctionnalité sur les terrains et les skyboxes. Par ailleurs, le logiciel utilise une méthode de conversion qui est de créer des centaines de polygones à la surface de l'objet converti. Cela nous a donc grandement aider pour la réalisation de la première carte et de la deuxième carte, qui ont d'abord été réalisés comme des cartes normales, puis convertit pour adopter le style lowpoly.

4.4 Librairie Graphique

L'Unity Asset Store est une bibliothèque regroupant des milliers de modèles 3D.

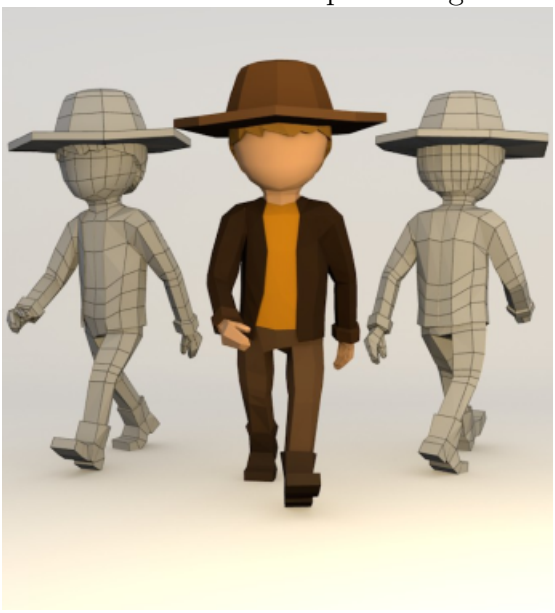


Il propose des modèles gratuits de qualité, et nous l'avons donc en partie utilisé afin de récupérer des modèles de bâtiments, d'arbres, d'herbes, de routes ainsi que des textures pour les utiliser sur les cartes. Nous avons récupéré un pack contenant des dizaines de bâtiments afin d'avoir de la diversité sur nos cartes. Cela a l'avantage d'avoir le même style graphique pour chacun de nos bâtiments.

4.5 Création des personnages

4.5.1 Création de la base

Afin d'avoir un personnage sur lequel baser les autres, nous avons décidé d'utiliser comme modèle un personnage cartoon en low poly.



Ce personnage a été créé grâce à une suite de tutoriels divers de *Sebastian Lague* : vidéaste intéressé dans tout ce qui est développement numérique, dont la création de modèle sous Blender.

4.5.2 Structuration globale des personnages

Afin de donner à nos personnages des aspects permettant de les distinguer, nous avons choisi de placer sur les personnages "Pompier" et "Policier" des casques en rapport avec leurs fonction. De ce fait, nous avons respectivement attribués pour le Pompier et le Policier un casque F1 de série S et un casque pare-balle de la Brigade Anti-Criminalité (BAC). Afin d'implémenter ces couvre-chefs sans trop de difficulté, nous avons utilisé la bibliothèque **3D WAREHOUSE**.

L'avantage de cette bibliothèque est que tous les éléments la composant étaient gratuits dans leur téléchargement et dans leur utilisation, ce qui nous a permis de gagner un temps précieux.

4.5.3 Création du modèle "Pompier"

Ce personnage était le premier à être créé.

Thème de couleur

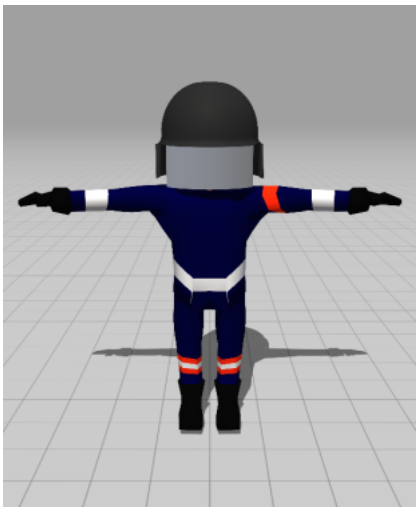
Après avoir testé plusieurs thèmes de couleurs, nous avons finalement choisi le rouge pour la veste de feu, ainsi que pour le surpantalon de feu. En effet, nous voulions au départ choisir du bleu foncé comme pour les pompiers civils, mais nous nous sommes rendu compte que cette couleur rendant le Pompier et le Policier peu distinguables. De ce fait, nous avons préféré la couleur de la Brigade des Sapeurs-Pompiers de Paris (BSPP) : le rouge vif.
Matériaux composants la tenue

1. Bandes réfléchissantes
   Afin de rendre la tenue du pompier plus réaliste, nous avons choisis de rendre les bandes jaunes réfléchissantes. De ce fait, la tenue renvoie

plus faiblement la lumière par rapport à ces bandes.

2. Casque de pompier
   Souhaitant avoir une physique de la lumière au rendez-vous, nous avons aussi mis des moyens sur la réflexion du casque. En effet, afin d'avoir les angles les plus précis sur la lumière de ce type de casque, nous avons utilisés le casque de service du chef de projet (Nathan RABET) afin de comprendre et de réadapter au mieux la réflexion de la lumière du casque de notre modèle. Cependant, étant donné que certains aspects physiques ne rendaient pas un aspect adaptable au Low Poly, nous avons fait le choix de retirer la réflexion totale sur le haut du casque.

4.5.4 Création du modèle “Policier”



Ce modèle à été créé conjointement avec celui du Pompier.

Thème de couleur

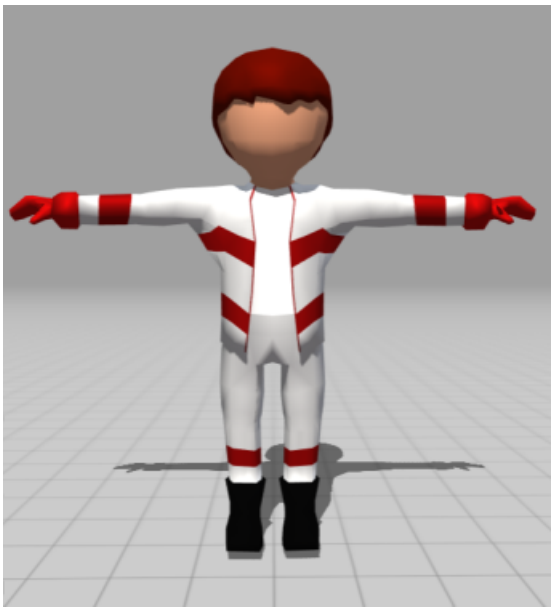
Lors de la différenciation entre le thème du Pompier et celui du Policier, nous avons choisis le thème rouge pour le pompier pour éviter de le confondre avec Policier. C’est donc en toute logique que nous avons attribué le thème bleu foncé au modèle du Policier.

Matériaux composants la tenue

1. Bandes réfléchissantes
   Tout comme le Pompier, nous avons décidé d’accentuer la réflexion des bandes réfléchissantes de la tenue du Policier.
   De plus, afin d’augmenter la capacité de reconnaissance du personnage, nous avons rajouté une banderole orange sur son bras gauche, signe qu’il fait bien parti de la police.

2. Casque de la BAC
   Notre personnage étant censé tirer sur une horde d'ennemis lors de son entrée en jeu, nous avons donc choisi de lui implémenter un casque pare-balle pour plus de cohérence. Ce casque à été pris sur **3D** WAREHOUSE (sauf pour la visière qui a intégralement été faite sur Blender), cependant, un travail de réflexion des matériaux à aussi été fait sur ce couvre-chef, car voulant y mettre autant d'optique que pour le pompier, nous avons choisi de mettre sa visière totalement réfléchissante.

4.5.5 Création du modèle “Médecin”



Ce personnage a été conçu en dernier, juste après le Pompier et le Policier.

Thème de couleur

Souhaitant rendre ce personnage visible et reconnaissable, nous avons choisi de lui attribuer des couleurs en mélange avec celui du médecin du **SAMU** et celui du **MEDIC** dans Team Fortress 2.


Nous n'avons pas mis de couvre-chef à ce personnage, estimant qu'aucun ne lui était fondamentalement nécessaire.

4.6 Implémentation des animations

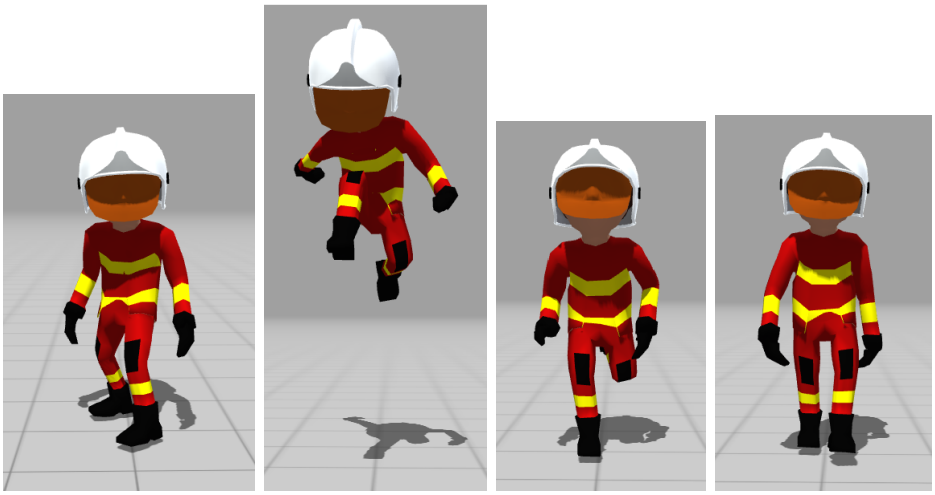
4.6.1 Les différents types d'animations :

Chaque personnage possède huit animations qui sont essentielles pour le déplacement du joueur :

- Marche avant
- Marche arrière
- Marche à droite
- Marche à gauche
- Saut
- Course avant
- Réception au sol
- Position de repos

Toutes ces animations proviennent de la bibliothèque d'animation mixamo.com. Ce site internet permet de synchroniser les animations à un modèle 3D fourni par l'utilisateur.

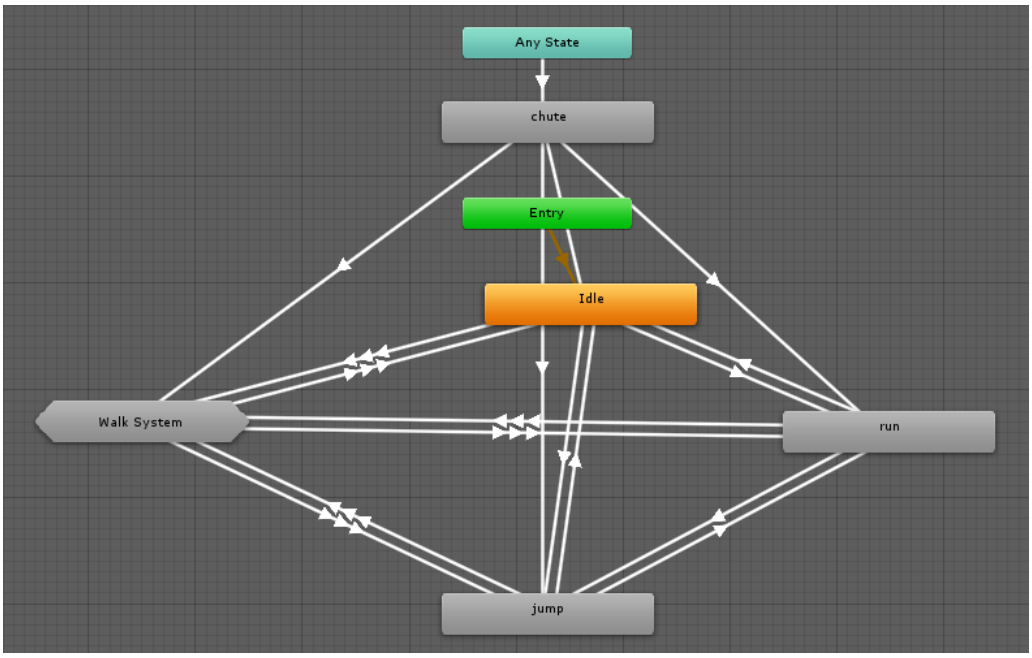
4.6.2 Exemples d'animations :



Au repos Saut Course Marche

4.6.3 Le composants *"animator"*

L'un des outils principaux d'unity dans la gestion des animations est le composant *"animator"*. Il permet de créer un algorithme sous forme de diagramme. L'un de ses avantages est sa simplicité d'utilisation : il suffit de créer différents états d'animation et de les relier entre eux à l'aide de liens.

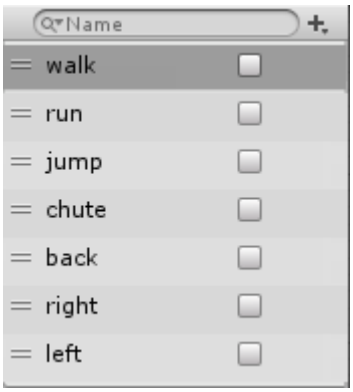


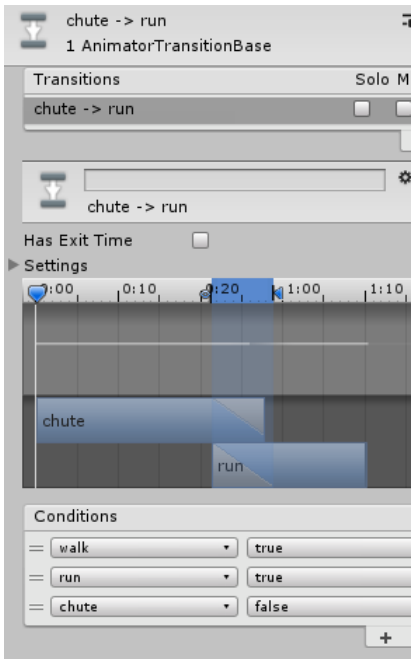
La case *"Entry"* symbolise l'entrée de l'algorithme. Ici l'algorithme arrive donc sur l'animation *"Idle"* qui est l'animation du personnage au repos. Cela signifie que lorsque aucune autre animation n'est lancée, l'algorithme revient sur la position *Idle*.

4.6.4 Les booléens

Les conditions présentent dans les liens pour changer d'animation à une autre sont des booléens.
Il y a sept booléens implémentés :

- walk
- run
- right
- left
- back
- jump
- chute





Sur les liens de transition entre les animations, on retrouve les conditions des états des booléens pour pouvoir passer d'une animation à une autre.

Par exemple dans le cas de la transition entre la chute et la course, le booléens *"walk"* et *"run"* doivent être à **True** et le booléen *"chute"* doit être à **False**.

Changement d'état des booléens :

L'extrait de script suivant montre la fonction générale du changement d'état des booléens. Ce script nommé *"Animator_Controller"* est attaché au prefab du joueur.

```
26          if (Input.GetKey(KeyCode.W))
27          {
28              anim.SetBool( name: "walk",  value: true);
29          }
30          else
31          {
32              anim.SetBool( name: "walk",  value: false);
33          }
34
35          if (Input.GetKey(KeyCode.W) && Input.GetKey(KeyCode.LeftShift))
36          {
37              anim.SetBool( name: "run",  value: true);
38          }
39          else
40          {
41              anim.SetBool( name: "run",  value: false);
42          }
43
44          if (Input.GetKey(KeyCode.S))
45          {
46              anim.SetBool( name: "back",  value: true);
47          }
48          else
49          {
50              anim.SetBool( name: "back",  value: false);
51          }
```

Son fonctionnement est le suivant dans le cas de l'animation marche avant par exemple : on teste si le joueur appuie sur la touche *"**W**"*. Si c'est le cas, le booléen *"walk"* de l'animator liée au script passe à **True** sinon sur **False**.
Ce principe est le même pour toutes les autres animations.

4.7 Menu / HUD

Nous avons commencé l'implémentation des menus ainsi que des HUD (affichage tête haute) qui sont des éléments graphiques donnant des informations aux joueurs.

4.7.1 Barre de vie

Les éléments constituants la barre de vie ont été réalisées à l'aide du logiciel

"Paint".

On retrouve un élément constituant le cadre de la barre et un élément décoratif. L'implémentation dans Unity se fait à l'aide du composant UI, qui permet de créer un système de gestion de HUD. Un slider (élément défilant) est ajouter au cadre de la barre de vie et permet de rendre visible graphiquement au joueur la quantité de vie restante. Un script est ajouté à la barre de vie et permet de régler la valeur du slider en fonction de la vie. Si la valeur du slider est au maximum, la barre de vie est pleine. La valeur du slider est modifié dans le script *"Player_Manager"* rattaché au joueur. Elle est égale à la valeur de la vie, gérée par ce script.

5 Conclusion

Avant cette première soutenance, nous avons donc mis en place de solides bases de jeu et de gameplay qui vont d'ici les prochains mois devenir plus intéressants à exploiter pour les joueurs.