Soutenance

PastaOverflow

14 mars 2019

1 Introduction

Aux vues de notre avancement dans notre projet, nous avons modifié notre cahier des charges afin de le faire correspondre à notre progression actuelle. Ainsi, nous détaillerons de une première partie les changements que nous avons apporté à notre cahier des charges, dans une deuxième partie, nous détaillerons la réalisation selon le point de vue de chacun des membres de l'équipe. C'est dans une troisième partie que nous montrerons notre planning indiquant notre progression dans nos parties respectives. Enfin, dans une dernière partie, nous conclurons le rapport de soutenance et nous détaillerons les annexes.

2 A propos du Cahier Des Charges

Lors de la réalisation du jeu, nous nous sommes rendus compte que notre cahier des charges nous imposait des tâches inappropriées à notre progression. Par exemple, il est impossible de permettre aux ennemis de tirer à vue sur le joueur si le système de tir n'a pas été implémenté. Nous avons donc reporté cette fonctionnalité à la deuxième soutenance. Nous avons donc créé un tableau permettant de visualiser toutes les modifications que nous avons apporté au planning de chaque soutenance.

3 Avancement par Parties

3.1 Intelligence artificielle

3.1.1 Pathfinding

Alexandre :

Il m'a semblé important et pertinent de permettre à l'ennemi de se déplacer vers le joueur. J'ai donc ajouté des instructions dans la méthode update() :

```
void Update()
    {
        /*if (target!=null)
        {
            // this.gameObject.transform.position =
               target.transform.position; Teleportation
```

```
            de l'ennemi sur le joueur s'il le detecte

        }*/
        if(target !=null && playerHasMoved())
        {
            updateLerpingParams();
        }
        if (isCloseCombat)
        {
            isLerping = false;
        }
        else
        {
            if (isLerping)
            {
                currentLerpTime += Time.deltaTime;

                if (currentLerpTime > lerpTime)
                {
                    currentLerpTime = lerpTime;
                }
                //lerp!
                float perc = currentLerpTime / lerpTime;
                transform.position =
                   Vector3.Lerp(startpos, endpos, perc);
                //Rotation des ennemis

                Vector3 direction =
                   (target.transform.position -
                   transform.position).normalized;
                Quaternion lookRotation =
                   Quaternion.LookRotation(new
                   Vector3(direction.x, direction.y,
                   direction.z));
                transform.rotation =
                   Quaternion.Slerp(transform.rotation,
                   lookRotation, Time.deltaTime * 5f);
            }
        }


    }
```

Le booléen isLerping est tout d'abord initialisé à false. S'il est vrai, on utilise la commande transform.position = Vector3.Lerp(startpos, endpos, perc) ; Celle-ci permet actualiser, avec un lerp la position de l'ennemi. perc correspond à sa vitesse. startpos et endpos sont les positions de l'ennemi et du joueur. Cependant, si nous laissons ces opérations, l'ennemi se déplace vers la position du joueur au moment où il est détecté. Cependant, si le joueur se déplace, sa position change, et donc, la nouvelle position sera différente de endpos.

Nous avons donc besoin d'une méthode indiquant si le joueur s'est déplacé, d'où la méthode PlayerHasMoved.

```
private bool playerHasMoved() //Sans cette methode,
   l'ennemi fonce sur la position du joueur sans se
   demander s'il s'est deplace entre temps
    {
        bool playerMoved = false;

        if(endpos!=target.transform.position)
        {
            playerMoved = true;
        }
        else
        {
            playerMoved = false;
        }
        return playerMoved;
    }
```

Ainsi, si l'on analyse la méthode update(), nous constations que si le joueur s'est déplacé, on actualise les positions startpos et endpos, via la méthode updateLerpingParams().

```
private void updateLerpingParams() //met a jour les
   positions de Lerping
    {
        endpos = target.transform.position;
        startpos = transform.position;
        currentLerpTime = 0f;
        isLerping = true;
    }
```

De plus, le booléen is_lerping est actualisé à true : on confirme le lerp. Ainsi, nous avons maintenant un déplacement de l'ennemi vers la position du joueur, même s'il se déplace entre temps.

Un dernier problème apparait : lorsque l'ennemi atteint le joueur (target), il continue à se déplacer sur sa position, il le pousse donc. Pour palier ce problème, j'ai implémenté une fonctionnalité nommmée CloseCombat.

```
void OnCollisionEnter(Collision collision) //On entre
    en close combat
    {
        if (collision.gameObject.tag=="Player")
        {
            isCloseCombat = true;
        }
    }
```

On constate que si l'ennemi touche le joueur, le booléen isCloseCombat est set à true. Evidemment, la méthode OnCollisionExit permet de set isCloseCombat à false.

```
void OnCollisionExit(Collision collision)//On sort du
    close combat
    {
        if (collision.gameObject.tag=="Player")
        {
            isCloseCombat = false;
        }
    }
```

Ainsi, avec ces méthodes, la fonction update() utilise le booléen isCloseCombat pour set isLerping à false si isCloseCombat vaut true. Ainsi, l'ennemi suit le joueur et dès qu'il rentre en collision avec le joueur, l'ennemi s'arrête et reprend sa course si le joueur décide de fuir.

Une dernière fonctionnalité importante pour moi est le fait que l'ennemi ne doit pas suivre le joueur jusqu'à l'infini ! J'ai décidé d'implémenter la méthode OnTriggerExit

```
private void OnTriggerExit(Collider other)
    {
        if (other.gameObject.tag=="Player")
        {
            target = null;
            isLerping = false;
        }
    }
```

qui permet de set lerping à false si le joueur sort de la zone de détection de l'ennemi. Ainsi, j'ai implémenté une intelligence artificielle qui permet à

l'ennemi de détecter le joueur, de le suivre et de se tourner en même temps, de ne pas le pousser et de ne pas le suivre si celui-ci sort de sa zone de détection.

3.1.2 Détection du Joueur a Distance

Alexandre :

Dans notre cahier des charges, nous avons annoncé que les ennemis repéreraient le joueur, le regarderaient mais resteraient immobiles. A l'aide d'un script C#, j'ai d'abord implémenté une méthode OntriggerEnter qui permettait initialement de détecter le joueur.

```
void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.tag=="Player")
        {

            target = other.gameObject;

            updateLerpingParams();
        }

        float Xdetected = other.transform.position.x;
        float Ydetected = other.transform.position.y;
        float Zdetected = other.transform.position.z;
    }
```

Si l'objet qui entre dans la box de collider possède le tag "Player", on assigne son gameObject à la variable target. La méthode updateLerpingParams() sera détaillée plus tard.
Grâce à cette méthode, l'intelligence artificielle détecte le joueur. Il faut maintenant que l'ennemi puisse se tourner vers le joueur lorsque celui-ci est détecté. Pour y parvenir, on implémente ces deux lignes dans la méthode update(), qui est actualisée à chaque frame du jeux.

```
if (isLerping)
{
        currentLerpTime += Time.deltaTime;

        if (currentLerpTime > lerpTime)
        {
                currentLerpTime = lerpTime;
        }
        //lerp!
```

```
            float perc = currentLerpTime / lerpTime;
            transform.position = Vector3.Lerp(startpos,
               endpos, perc);
            //Rotation des ennemis

            Vector3 direction = (target.transform.position -
               transform.position).normalized;
            Quaternion lookRotation =
               Quaternion.LookRotation(new
               Vector3(direction.x, direction.y,
               direction.z));
            transform.rotation =
               Quaternion.Slerp(transform.rotation,
               lookRotation, Time.deltaTime * 5f);
}
```

On constate que l'ennemi effectue un Slerp (un déplacement) de type rotation, dont les coordonnées cibles sont direction.x,direction.y,direction.z.

3.2 Niveaux

3.2.1 Architecture

Jean-Philippe :

Pour la première partie de la map nous avons décidé de se baser sur la structure d'un hangar dont le but est d'atteindre l'autre bout du hangar pour terminer cette partie du niveau. Des armes sont disposées le long du le chemin afin d'affronter les ennemis rencontrer si cela s'avère nécessaire. Des ennemis sont présents sur la map empêchant de traverser la map sans difficulté. Des conteneurs, des caisse et des barils sont disposé permettant d'avoir une couverture mais aussi d'empêcher le joueur d'aller à la fin du niveau avec une simple ligne droite. Une passerelle est disposé faisant le tour de la carte est de contourner un obstacle. La disposition des conteneurs, des caisses et des barils est faite de manière à permettre au joueur de ne pas avoir un décor trop redondant. Alexandre

La carte à été développée en deux parties (cf image 1) : le joueur a le choix entre un passage impliquant une certaine discrétion de sa part et un passage forçant le joueur à évoluer de façon plus nerveuse. En effet, cette voie possède un passage comprenant deux ennemis sur les côtés qui détectent obligatoirement le joueur, s'il décide de l'emprunter. Ce niveau illustre le principe du jeu : un mode infiltration et un mode plus nerveux et intense. Si le joueur décide de tourner à gauche, il arrivera dans un couloir dans lequel un premier

ennemi est placé en faction. Après l'avoir éliminé, celui-ci peut continuer son chemin, faire face à un denier ennemi et atteint la sortie. (cf image 2).
Si le joueur tourne à droite, il arrive dans un premier couloir avec des ennemis cachés, qui l'attendent. Ainsi, si le joueur décide d'utiliser ce chemin, il devra obligatoirement faire face à ces deux ennemis.(cf image 3)

3.2.2 Détails

Jean-Philippe :

Pour les détails J'ai ajouter diverse taille de caisses ainsi que des barils. Des conteneurs sont parfois ouvert avec un différent nombre de caisses à l'intérieur des portes de conteneurs ouverte différemment. Les différents objet sont positionner différemment pour varier le décor. Alexandre

Dans le cahier des charges, il a été précisé que nous n'ajouterions pas de détails pour nos niveaux. Cependant, nous avons pris de l'avance en ajoutant des conteneurs, des caisses et des lumières.(cf image 2).

3.3 Gameplay

Le joueur peut se déplacer librement sur une carte à l'aide de touches directionnel et les collision joueur-carte-objet. Le joueur peut aussi utiliser sa souris afin d'explorer son environement en trois dimension. De plus, le joueur a la possiblilité de neutraliser des ennemis à l'aide d'armes à feu ramassés dans les niveaux. Une arme ramassée est aujouté dans l'inventaire invisible du joueur. S'il possède déjà l'arme qu'il ramasse, les munitions seront ajoutés dans une des trois catégories d'arme disponibles (Légére, Moyenne ou Lourde).
Chaque arme possède des dégâts, une cadence de tir et un nombre de munition qui leur sont propre.
Afin de déterminer si le joueur touche un ennemi lors du tir, les armes utilisent le principe de 'Raycast' ou le lancer de rayon en Français. Dès que le joueur tire, on vérifie si le 'Rayon' envoyé ne touche pas un mur ou tout autre objet qui n'est pas un ennemi. Cette vérification est faite en comparant l'étiquette de l'objet en question avec l'étiquette 'Enemy'. Si l'objet touché est un ennemi, nous appelons dans cet enemy une fonction permettant de décrémenter la vie de cet objet en fonction des dégâts de l'arme utilisée :

```
if (hit.transform.tag == "Enemy")
{
hit.transform.gameObject.GetComponent<Enemy>().Hit(playerWeapons[currentW
}
```

3.4 Modélisation 3D

3.4.1 Armes

3.4.2 Ennemis

3.5 Aspect Sonore

3.5.1 Effets sonores

Benoit :

J'ai enregistré, à l'aide d'un micro studio et d'un ordinateur, des sons d'une arme à feu dans un centre d'armes à feu à Strasbourg. J'ai tiré, en étant encadré par un professionnel de tir, dans un stand de 25 mètres. M'étant aperçu du faible nombre de sons, j'ai pris d'enregistrer des sons d'une arme à air comprimé.

3.5.2 Musique

Amaury :

De mon coté, j'ai commencé à chercher des musiques libre de droits qui pourrait convenir à l'ambiance générale du jeu. J'ai remarqué que peu de musique de qualité existe et pourrait s'apparenter à l'ambiance du jeu. Ainsi, il a été difficile pour moi de trouver des musiques adapté mais voici une liste de trois des meilleurs musiques qui conviendront pour notre projet :

— Brutal Cycle (TeknoAXE)
— Waypoint J (TeknoAXE)
— Mission A (TeknoAXE)

Cet auteur nous permet d'utiliser ses œuvres gratuitement et librement. Il autorise une utilisation commerciale comme personnelle de ses œuvres.

3.6 Autre

3.6.1 Multijoueur

Jean-Philippe :

Le multijoueur est encore en cours de développement et n'est donc pas encore implémenté dans le jeu. Certains objets capitaux n'ont pas pu être adaptés pour le multijoueur puisque ces derniers étaient encore en développement. Ces objets furent développés en priorité pour le mode solo du jeu. Le développement du multijoueur en est au stade de devoir reconnaitre les objets propre à chaque client.

3.6.2 Menu

Amaury :

Le Menu se divise en deux partie. La première partie est le panel du menu avec l'image du logo en noir et blanc le fond et les boutons jouer (Play), options (Options) et quitter (Quit). Ainsi j'ai fait plusieurs scripts pour chaque boutons.

Le bouton Play lance la fonction StartGame() qui charge la scène du jeu principale appelée startSceneName.

```
public void StartGame()
{
        SceneManager.LoadScene(startSceneName);
}
```

Le bouton Options lance la fonction Options() qui active le panel des options (devant le menu principale).

```
public void Options()
{
        OptionsPanel.SetActive(true);
}
```

Le bouton Quit lance la fonction Quit() qui ferme le jeu.

```
public void Quit()
{
        Application.Quit();
}
```

Puis j'ai fait le menu des options sans les options (car nous n'avons pas encore implémenté tout ce qui est nécessaire pour le faire). Le menu des options se compose de deux images de pistolets qui rappelle le coté action du jeu, de trois boutons pour le menu et d'un bouton de fermeture du menu des options.

Le bouton de fermeture est implémenté par un script avec deux fonctions. La première est le Start() qui initialise la valeur du panel lui même.

```
private void Start()
{
        OptionsP =
           OptionsP.GetComponent<MainMenu>().OptionsPanel;
}
```

La fonction QuitOptionsMenu() est appelée quand on clique sur le bouton de fermeture du menu d'options qui rend le panel des options inactif.

```
public void QuitOptionMenu()
{
        OptionsP.SetActive(false);
}
```

Vous pouvez voir le rendu de ces deux menus en annexes de ce rapport.

3.6.3 Site Internet

Amaury :

Le site internet à été implanté avec hugo (un interpréteur de markdown) et voici le résultat du post que j'ai créé pour la première soutenance :

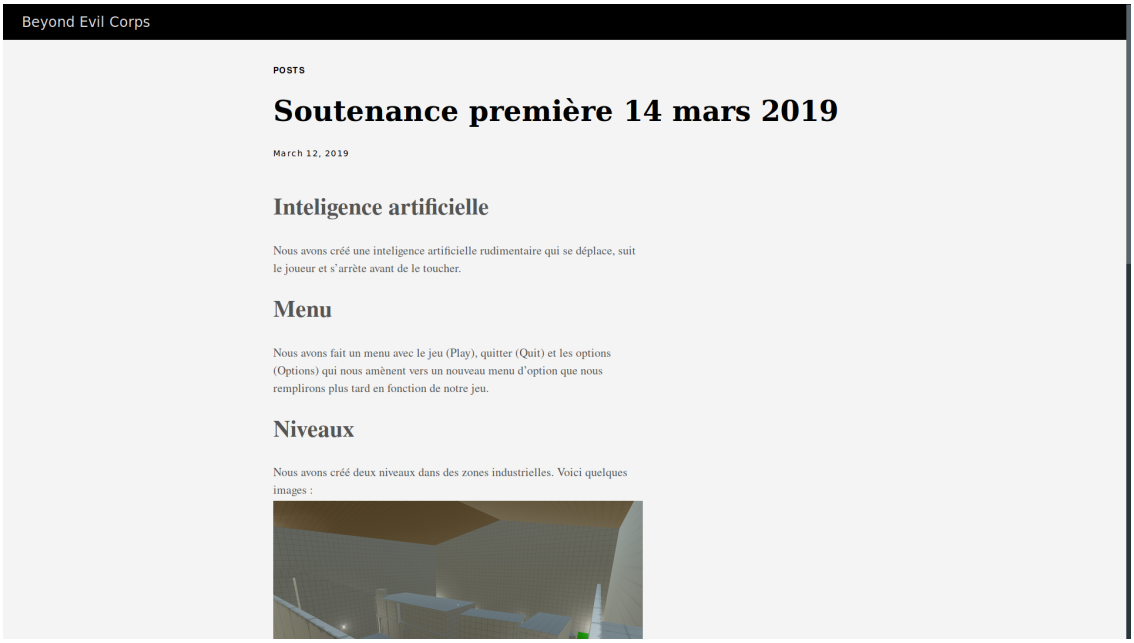

FIGURE 1 – Le Post sur la Première Soutenance

4 Avances et Retards

4.1 Intelligence artificielle

4.1.1 Avances

Alexandre :

— Détection du joueur simplifiée : il est possible d'adapter la zone de détection des ennemis
— Pathfinding :
  — les ennemis se déplacent vers le joueur, de face (donc se capables de se tourner) en s'arrêtant devant lui
  — Si le joueur se déplace hors de la zone de détection des ennemis, ils arrêtent de le poursuivre

4.1.2 Retard

Alexandre :

Aucun

4.2 Niveaux

4.2.1 Avances

Alexandre et Jean-Philippe

Ajout de détails :
— conteneurs
— caisses
— lumières colorées

4.2.2 Retard

Alexandre et Jean-Philippe

Aucun

4.3 Gameplay

4.3.1 Avances

Benoit

J'ai implémenté deux armes au lieu d'une.

4.3.2 Retard

Benoit

Le bouton de course à mal été implémenté

4.4 Modélisation 3D

4.4.1 Avances

Benoit
Aucunes

4.4.2 Retard

Benoit
Tout est modélisé de façon rudimentaire. Faute de connaissance de Blender. La tache à été sous-estimé.

4.5 Aspect Sonore

4.5.1 Avances

Amaury et benoit
Aucunes

4.5.2 Retard

Amaury et benoit
Aucun

4.6 Autre

4.6.1 Avances

Amaury
Nous sommes en avance sur le menu car nous voulions faire 2 boutons mais nous en avons 3 et un menu d'options en plus.

4.6.2 Retard

Jean-Philippe
Retard sur le multi-joueurs car impossibilité de l'implémenter à ce stade.

5 Avancements Prévus à Venir

5.1 Intelligence artificielle

5.1.1 Pathfinding

Alexandre :

Nous voudrions également faire en sorte que les ennemis fassent des rondes dans les couloirs. Pour y parvenir, j'implémenterai une fonction, basée sur un booléen (qui déterminerait si le joueur est détecté ou non par un ennemi) qui bloquerait les déplacements sur des points précis (pour les allers/retours). Cependant, si le joueur est détecté par un ennemi, la méthode ne sera plus appelée et les ennemis se déplaceront vers le joueur (en tirant, dont le script aura d'abord été implémenté).

5.1.2 Détection du Joueur a Distance

Alexandre :

Pour la deuxième soutenance, nous voudrions que les ennemis possèdent l'action de tirer : ils pourront techniquement tirer, il ne manquera plus que le script permettant que les ennemis tirent, ce qui est situé dans la section "gestion des armes". Mon objectif est d'augmenter l'interaction avec le joueur : pour cette première soutenance, ils se contentent de fixer le joueur. Ainsi, les ennemis auront un objet associé qui sera l'arme (dont le script appellera celui implémenté par le responsable de la gestion des armes). Pour que l'ennemi puisse tirer, il faudra que le joueur se situe dans sa zone de détection, qu'il n'y ait pas de mur (ou d'ennemis) entre lui et le joueur. Une méthode faisant appel à au moins deux booléens assurant ces conditions est nécessaire. Si et seulement si ces conditions sont respectées, je ferai appel (dans une autre méthode) au script utilisé dans la perte des points de vie. La cadence de tire sera déterminée par le responsable de la section gestion des armes. Aussi, à l'aide des tags des objets du jeu, je ferai en sorte que les ennemis ne se tirent pas dessus.

5.2 Niveaux

Alexandre et Jean-Philippe

Lors de la deuxième soutenance, nous aurons ajouté 3 cartes : Une dans l'univers de l'usine et les deux autres dans le thème des bureaux. La première carte se déroulera dans une usine, elle sera très détaillée et non texturée. Les deux autres cartes se dérouleront dans des bureaux, elles ne seront pas

détaillées et non texturées. Aussi, nous aurons détaillé les premières cartes et commencé à ajouter des textures.

5.3 Gameplay

5.3.1 Déplacement du Joueur

Benoit

Pour la prochaine soutenace, je prévois d'ajouter une animation de marche et de course pour le joueur en modifiant la hauteur de la caméra. La majorité des armes sont déjà implémentées. Il reste à ajouter le fusil d'assaut. Aussi, il faut que j'ajoute un mode permettant aux ennemis de tirer afin que le responsable intelligence artificielle puisse utiliser le script.

5.4 Modélisation 3D

Benoit

Les ennemis doivent encore être modélisés, leur skin sera celui indiqué dans le cahier des charges. Il existera trois types d'ennemis : le léger (en costume), le normal (avec un gilet par-balles) et le gros (en kevlar). Deplus, il me faut encore modéliser le joueur afin de lui donner une apparence concrête et définitive (évidemment en lien avec le thème de notre jeu). Les armes seront modélisées, elles auront plus réaliste et possèderont des animations (tirs et rechargement).

5.5 Aspect Sonore

5.5.1 Effets sonores

Benoit :

Les coups de feu des armes sont enregistrés mais restent brut. En effet, la majeure partie nécéssite un traitement audio afin de les rendre plus percutants et plus agréable à écouter. C'est à dire, faire paraître l'arme puissante sans provoquer un mal de tête.

5.5.2 Musique

Amaury :

Composition d'une ou plusieurs musique en fonction de mon inspiration et implémentation des musiques libre de droit.

5.6 Autre

5.6.1 Multijoueur

Jean-Philippe :

Pour la seconde soutenance le multijoueur sera implémenté de manière à ce que 2 joueurs puissent se trouver en même temps sur une map conçu spécialement pour ce mode de jeu. Le choix pour le mode multijoueur sera donc aussi ajouté. La gestion des joueurs, des armes et des ennemies sera ajoutée. Nous comptons faire en sorte que notre mode multi-joueurs puisse s'intégrer aussi bien dans la vision que nous avons de notre projet informatique que dans sa version la plus concrète et finale.

5.6.2 Menu

Amaury :

Faire un sous menu entre multi-joueur et le mode solo. ajout d'un menu interne au jeu avec la touche échap.

6 Expérience personnelle

6.1 Benoît

Pour notre première soutenance, j'ai réalisé le déplacement du joueur sur la carte, la gestion des armes et de la modélisation 3D. Unity est un moteur qui, malgré sa réputation de moteur de jeu facile à apprhender, est pourtant difficile à prendre en main. Par exemple, une option qui se doit d'être visible se retrouvent souvent dans une myriade de sous menus en tout genre. L'interface de Unity peut être comparé à un cockpit d'avion, dont chacun de ces boutons pouvant mener l'avion à sa perte.

6.2 Alexandre

Mes missions dans notre projet concernent l'intelligence artificielle et les niveaux. Mon avancée dans ces domaines s'est révélée bien plus complexe que prévue initialement. En effet, la mise en place d'une intelligence artificielle capable de détecter un joueur m'a semblé complexe au début du projet. Cependant, en me renseignant auprès des outils mis à disposition par Unity

en terme de documentation, j'ai pu mieux comprendre et diviser les tâches qu'implique un tel projet.

En effet, lorsque que mon intelligence artificielle était à son point le plus simple, elle se dirigeait vers la position du joueur, sans prendre en compte les déplacements éventuels du joueur. si l'ennemi entre en collision avec le joueur, il s'arrête. Les ennemis sont désormais capables de détecter le joueur si celui si passe dans une zone de détection. Cette zone est modifiable. Il est donc possible de placer des caméras qui seraient composées d'une zone de détection commune à tous les ennemis par exemple.Ces exemples montrent comment la division permet de régler les problèmes les plus complexes, en les réduisant à plusieurs sous problèmes, plus faciles à résoudre. Aussi, en plus de l'intelligence artificielle, j'ai également réalisé un niveau. La conception et la réalisation d'un niveau sont très différentes de celles d'une intelligence artificielle. En effet, il s'agit de deux opposés : le développement d'une intelligence artificielle consiste à programmer, à implémenter des fonctions, des méthodes, contrairement au développement d'un niveau qui ne met en œuvre que des compétences de game design. J'ai personnellement préféré la programmation d'une intelligence artificielle au développement d'un niveau.

La création d'un niveau est cependant très intéressante dans la mesure où elle met en pratique l'imagination du concepteur ainsi que son anticipation du joueur : il faut orienter le joueur vers la sortie, le faire passer au plus près des ennemis sans pour autant que celui-ci ait l'impression d'être emprisonné dans un circuit. Mon avancée bonus consiste en l'ajout de détails non prévus initialement. Par exemple, la présence de détails tels que des conteneurs permet de relier le niveau à son thème, à savoir l'usine pour cette première soutenance. Ainsi, à travers les différentes difficultés qu'il m'a été donné de rencontrer, j'ai pu améliorer ma compréhension de l'ensemble des mécanismes liés à l'intelligence artificielle. Aussi, pouvoir voir mon intelligence artificielle "prendre vie" dans le jeu est une source de satisfaction nouvelle et très enrichissante pour moi.

6.3 Amaury

Dans cette soutenance, j'ai été responsable du menu de départ (lorsque l'utilisateur lance le jeu). J'ai commencé par le Menu et ses butons qui affiche le nom du jeu en bas de l'écran. J'ai écrit les scripts pour l'activation des boutons vers les bonnes scènes, activation de panel etc... J'ai fait le menu des options en mettant le nom du jeu eu haut avec une couleur différente et des nouveaux boutons d'options. Ce qui me mets à 70 au lieu de 30 d'avancement

dans la section menu.

Avec le menu, j'ai réalisé le site web qui est essentiel pour la communication de notre projet. J'ai crée le site web qui est fonctionnel pour, d'une part présenter l'avancée de notre projet et d'autre part le faire découvrir aux nouveaux utilisateurs. En parallèle du site web, j'ai choisi de changer le logo car notre équipe n'était pas satisfaite de celui-ci. Ainsi, en utilisant Gimp et une image (libre de droit) de la tour Rockefeller (Empire State Building), j'ai créé le nouveau logo de notre projet que vous pouvez voir (cf annexes).

Avec ces trois choses j'ai aussi donné des idées de création dans les différents niveaux. J'ai donné quelques idées à Alexandre et Jean-Philippe pour les niveaux sachant que c'était une tache annexe pour moi. Mon départ c'est avéré davantage difficile que ce que j'avais prévu. J'ai commencé par vouloir installer Rider sur Windows pour pouvoir faire un script sur le menu. Je ne pouvais pas l'installer à cause de Windows qui avait une version trop ancienne. Windows n'avais pas été mis à jour à default de place sur mon disque dur. Ainsi donc, j'ai perdu une journée à essayer d'installer une mise a jour qui ne voulait pas s'effectuer puis je suis passé sur Linux qui à fonctionné.

Faire le menu ma beaucoup apporté car c'était la première fois que je fessais quelque chose avec Unity. C'était aussi la première fois que j'écrivais un script sur Rider et j'ai donc appris comment fonctionnent les classes de Unity en C#. Le site web est écrit en Markdown j'ai donc appris ce langage pour l'écrire.

La création du logo m'a apporté beaucoup de connaissance de logiciel de traitement d'image qui me serviront surement par la suite de ce projet. Ce début de projet ma montré que Linux est beaucoup plus pratique pour gérer ce type de projet. Il m'a aussi appris à faire des scripts ainsi que des sprites sur Unity. Enfin ce début de projet me donne envie de continuer à apprendre et à m'investir dans le jeu et l'équipe.

6.4 Jean-Philippe

Ma partie consiste en la création de la première partie de la carte du premier niveau. Cette carte se trouve dans un univers industriel, elle sera alors un hangar de stockage entièrement fermé. La carte s'avère difficile à créer avec les outils de base de unity mais s'en est vus facilitée lorsque l'on commence à maitriser l'outil probuilder. Cet outil permet ainsi d'ajouter facilement des objets plus complexes que les objets basiques fournit par unity. J'ai du apprendre à utiliser probuilder en essayant de faire différents objets basiques tels que des murs ou des caisses de différentes tailles, me permettant

ainsi de faire des structures plus complexescomme des structures trouées et des ensembles d'objet permettant de faire un conteneur. Il permet aussi de modifier n'importe quel objet ayant déjà été placé sur la carte. L'idée du hangar m'inspira pour faire une sorte de chemin défini par les caisses et les conteneurs rangés de manière organisé vers le fond du hangar, devenant de plus en plus désorganisés vers les portes d'arrivées des marchandises. Le point de départ ayant été défini comme une porte de sortie de secours qui est alors placée plus vers l'avant du bâtiment et le point d'arrivée une autre porte de sortie se trouvant à l'opposé du hangar.

L'idée de base du niveau est de pouvoir se cacher à l'intérieur de certains conteneurs laissés ouverts. La création de ce niveau à été difficile dû à la corruption des données de la carte dès la première création de la carte empêchant ainsi de sauvegarder cette dernière faisant ainsi perdre une grande quantité de travail. La dernière difficulté vis à vis de la carte fut l'ajout des lumière qui avaient un défaut de génération, laissant ainsi de temps à autre des lampes disparaître lors de changement d'angle de la caméra du joueur.

La création de carte n'étant pas mon unique rôle, je me suis lancé dans la gestion du multijoueur. L'implémentation de ce dernier est encore en développement puisque pour faire cela il est nécessaire de modifier les scripts du "player" ainsi que celui des armes et des ennemis qu'il faut adapter au multijoueur car ces scripts sont penser au départ pour le jeu solo mais étant corriger régulièrement la modification de ces derniers nécessite d'être sûr de ne plus modifier le script après l'adaptation. Je me suis alors renseigner sur la gestion du mode multijoueur permettant de l'ajouter dès que l'adaptation sera possible.

Ces recherches m'ont permis de comprendre les bases d'un système de room se trouvant dans un lobby et la gestion des clients de chaque room avec, comme finalité, de mettre en place, d'installer un mode multi-joueurs efficace, intéressant, optimisé et en lien avec notre version propre et unique de notre projet, notre jeu, en groupe en informatique.

7 Conclusion

De notre idée, Behind Evil Corp, est née un intérêt grandissant pour le développement d'un projet informatique en groupe. La confrontation permanente de nos idées à la réalité est parfois difficile et plus longue que prévu. Notamment des problèmes inopinés. Ces problèmes étant parfois très frustrant, il est néanmoins très satisfaisant de les résoudre. Certain au travers de la programmation on put avoir une premier approche de l'intelligence arti-

ficielle. Nous avons dans l'ensemble appris à utiliser Unity, même si parfois son utilisation c'est avéré hasardeuse !

8 Annexes

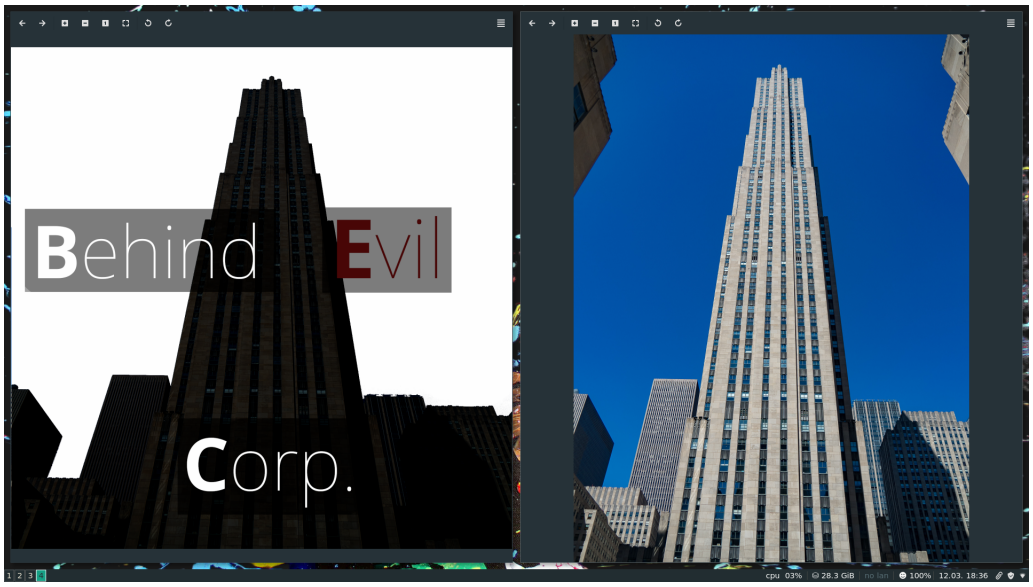

FIGURE 2 – Notre logo Avant-Après

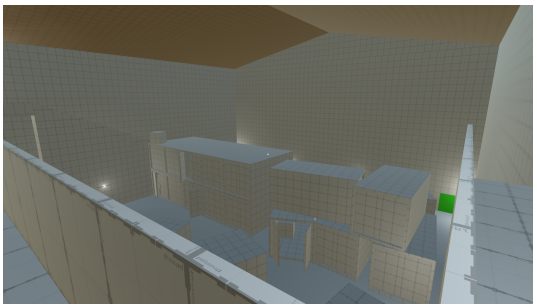

FIGURE 3 – Partie 1 du Niveau 1 in Game a partir de la passerelle

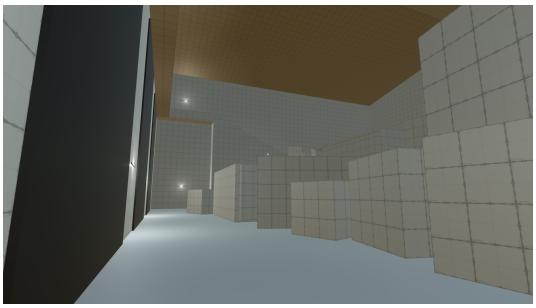

FIGURE 4 – Partie 1 du Niveau 1 in game Départ

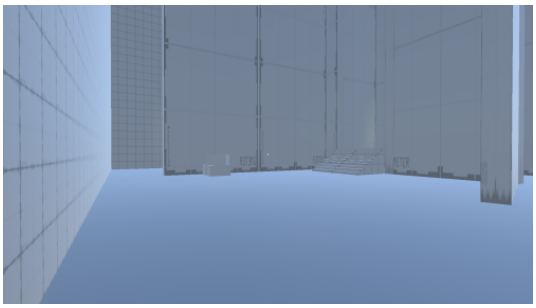

FIGURE 5 – Level 2 image 1

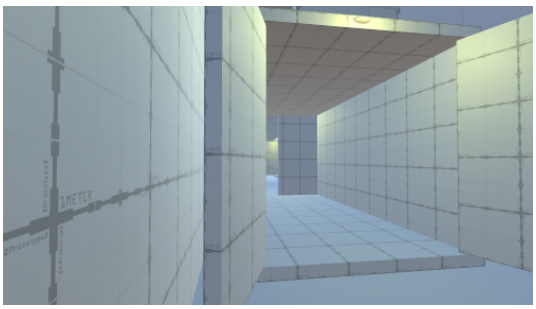

FIGURE 6 – Level 2 image 2

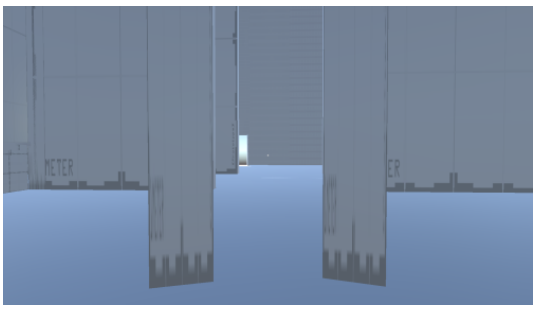

FIGURE 7 – level 2 image 3

FIGURE 8 – Menu Principale

FIGURE 9 – Menu des Options