Panic At Tortuga

Rapport de soutenance n°2

Avril 2021

LIFE INVADERS

Production

Table des matières

1 Introduction

Cette seconde période de développement de `Panic At Tortuga` a été très intéressante et a permis d'améliorer de nombreuses parties du jeu final.

1.1 Intelligence Artificielle

1.1.1 NPC Zone & Système de déplacement

Nous avons essayé plusieurs types de déplacement pour les IA. Nous avons créé des NPC zones qui permettent de créer des quartiers où les NPC peuvent se balader librement dans la zone. Si un NPC est tué dans la zone, un nouveau apparaît avec un effet visuel approprié.

Pour le système de déplacement de l'IA, nous avons essayé plusieurs algorithmes différents :

— Le premier algorithme que nous avions montré lors de la première soutenance était un déplacement vers un élément alétoire d'une liste de coordonées prédéfinie. Le tracé était vu et revu, et de ce fait, trop prévisible. De plus, si un personnage allait d'un point A vers un point B, et un autre de B vers A, alors ils allaient prendre le même chemin et donc se croiser (face à face) et finir coincé.

— Nous avons donc décidé de rajouter plus d'alétoire dans leurs déplacements. Les IA cherchent une destination "accessible" dans un rayon proche. Le problème était que la librairie Unity AI intégrée cherche un chemin complet ou partiel. Le rendu final montrait des personnages qui finissaient toujours par être attiré par les bordures de la carte.

— Le dernier système intégré changait un point. Au lieu de chercher une destination proche de lui, il cherchait une destination dans le rayon d'action de la NPC zone.

1.1.2 Evenements

Afin de rendre les IA moins scriptées, nous avons ajouté des événements alétoire.

Il y a des zones de discussions où les NPC peuvent aller. Les joueurs peuvent interagir avec et se fondre dans la discussion, pour se cacher.

1.2 Réseau

1.2.1 Transition du Lobby vers le jeu

Un système a été mis en place permettant le passage de la scène de lobby vers la scène de jeu. Ce système fonctionne a l'aide d'un timer, qui est synchronisé entre tout les joueurs, y compris ceux rejoignant le lobby après son lancement. Ce timer ne se lance uniquement après que le serveur soit à moitié rempli et dure 3 minutes. Si le nombre de joueurs descend en dessous de ce seuil, le timer s'arrête. De plus, qaund le serveur est complet, le temps d'attente est réduit à 30 secondes. A la fin du timer, le Master Client charge la map de jeu, qui est synchronisé avec tout les joueurs.

1.2.2 Synchronisation des joueurs

Tout comme sur le lobby, les mouvements des joueurs ainsi que leurs animations sont synchronisés. Cependant, un deuxième élément s'ajoute à celà : l'apparition (spawn) des joueurs. Pour celà, des points d'apparitions (spawpoints) sont répartis sur la map. Le Master Client distribue ces points aux joueurs qui apparaîtront à l'endroit reçu. Celà permet de faire apparaîte chaque joueur à une position unique sur la carte : un point = un joueur, pas plus. Une fois celà fait, il faut également synhcroniser la réapparition des joueurs, pour celà on applique le même système, en ne prenant en compte que les joueurs morts.

1.2.3 Synchronisation de l'IA

La grande difficulté niveau multijoueur, c'est la synchronisation des PNJSCelà est une tâche important dû à la nature du jeu, mais également difficile dû au grand nombre d'IA présentes sur la map.

Tout d'abord, il faut synchroniser l'apparition des PNJs. Encore une fois, c'est le Master Client qui instancie les personnages grâce à Photon. Ils sont donc au départ placé de la même façon pour tout les joueurs.

Mais les problèmes commencent au moment de synchroniser le mouvement des IA. Le système qui a été créé par Dov permet à l'IA de se déplacer sur la map, il faut maintenant que ce mouvement soit propagé de façon quasi-identique à tout les joueurs. Pour celà, plusieurs méthodes ont été envisagés :

-L'utilisation de Photon Transform View, comme pour les joueurs. Ce système a vite montré ses limites, car inadapté à la synchronisation d'un grand nombre d'objets, fonctionant sur la base d'un envoi pseudo-continu d'informations. Ainsi de nombreux problèmes apparaissaient, et la synchronisation des mouvement en a souffert.

-Calcul de chemin client-side à partir du même point. L'idée est la suivante : le master client calcule un point, qui est la destination de l'IA et la partage aux autres joueurs. Puis chaque joueur calcule le chemin pris par l'IA pour y arriver. Celà réduit considérablement la quantité d'information échangée, mais un autre problème se pose : le calcul de chemin pour les NavMesh Agents n'est pas déterministe. Ainsi le chemin calculé par chaque client à partir du même point n'est pas le même, ce qui entraîne également une désynchronisation de la position.

-Enfin, le choix retenu est le calcul d'un chemin entier par le Master Client, qui envoie ensuite l'intégralité de ce chemin aux autres joueurs. Ainsi le chemin est le même pour tout le monde, mais l'envoi des points se fait de façon discrète : on envoit uniquement l'array de positions généré par le Master Client. Ce système permet d'avoir une synchronisation satisfaisante des déplacement et une utilisation minime de la bande passante.

1.2.4 Synchronisation des assassinats et des morts

Une dernière partie de la synchronisation inclue celle des évènement de mort, pour les joueurs ainsi que les IA. Pour celà, il a été décidé d'utiliser le sytème d'Event Photon. Auparavant, la synchronisation de méthodes passait par l'utilisation de Photon View et de RPC. Mais le système de kill/death demande une propagation plus importante de l'évènement, car plusieurs systèmes différents doivent y réagir. C'est la que Photon rentre en jeu.

En effet, ce dernier permet de synchroniser le déclenchement d'évènement. Pour celà, on utilise la méthode RaiseEvent ainsi qu'un code représentant notre évènement. Cette action appelle la méthode callback OnEvent, dans laquel il suffit d'associer le code de l'évènement à une méthode. Ainsi, à la mort d'un joueur, le tueur déclenche l'évènement mort en indiquant l'identité du joueur tué. Chaque joueur reçoit cette évènement et peut déterminer, par exemple, si le joueur mort est lui-même, ou bien si un autre joueur a tué sa cible... En bref, chaque client prend une décision en fonction de l'information reçu Les détails du système de morts sont dans la partie gameplay.

1.3 Gameplay

1.3.1 Système de verrouillage

Pour tuer un personnage (joueur ou NPC), nous avons ajouté un système de verrouillage. Celui-ci est assez pratique, lorsque l'on passe dans ce mode, l'écran change, un effet réalisé grâce au post processing de Unity permet de donner un effet sépia/vieux films [1]. Un contour blanc autour des personnages visés au centre de l'écran permettent voir quel personnage va être sélectionné.

Une fois sélectionné et lorsque le joueur est proche de sa cible, il peut alors l'éliminer.

L'effet a demandé de créer plusieurs overlay de caméra, afin d'avoir un effet graphique appliqué uniquement sur certains layers, et de les surperposer les unes sur les autres.

Nous avons fourni un vrai travail sur les différentes animations de morts des personnages. Lors de sa réalisation, un problème s'est posé : Il fallait que les animations de deux GameObject (ici le

1. voir Effets Graphiques

joueur qui tue et le NPC/joueur tué) soient synchronisées et très précises spatialement. Après avoir regardé plusieurs types de solutions, nous nous sommes tournés vers un outil préintégré nommé Timeline. Ce dernier permet de réaliser des clips vidéos.

Voici donc comment nous avons intégré les timelines :
Des faux personnages jouant les animations sont ajoutés sur la cartes à la position et rotation du tueur. On masque le tueur et le tué de la carte. On change le mesh et le matériau de chaque personnage de la timeline pour qu'il corresponde au tueur et au personnage tué.

Une fois l'animation terminé, un signal est envoyé à script qui réaffiche alors le joueur qui était masqué.

Table des figures

Liste des tableaux