

Panic At Tortuga

Rapport de soutenance n°2

Avril 2021



Table des matières

1	Introduction	2
2	Développement	2
2.1	Intelligence Artificielle	2
2.1.1	NPC Zone & Système de déplacement	2
2.1.2	Evenements	2
2.2	Carte du jeu	2
2.3	Site Internet	2
2.3.1	réalisation	2
2.3.2	Hébergement	3
2.3.3	Améliorations futures	3
2.4	Réseau	4
2.4.1	Transition du Lobby vers le jeu	4
2.4.2	Synchronisation du lancement de partie	4
2.4.3	Synchronisation des joueurs	4
2.4.4	Synchronisation de l'IA	5
2.4.5	Synchronisation des assassinats et des morts	5
2.4.6	Synchronisation du déroulé de partie	6
2.5	Gameplay	6
2.5.1	Système de verrouillage	6
2.5.2	Finishers	6
2.5.3	Système de manche	8
2.5.4	Système de mort	8
2.5.5	Système de point	8
2.6	Graphismes	9
2.6.1	Shader Eau	9
2.6.2	Mode Jour/Nuit	10
2.6.3	Shader de disparition/apparition	10
2.6.4	Post Processing	13
3	Conclusion	14

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.

2 Développement

2.1 Intelligence Artificielle

2.1.1 NPC Zone & Système de déplacement

Nous avons essayé plusieurs types de déplacement pour les IA. Nous avons fait 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 (shader d'apparition).

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 aléatoire d'une liste de coordonées prédefinie. Le tracé était de ce fait trop prévisible et ne permettait pas de se déplacer librement tout en restant discret. De plus, si un personnage allait d'un point A vers un point B, et un autre de B vers A, alors en prenant le chemin le plus court ils finissaient par se croiser (face à face) et finir coincés.
- Nous avons donc décidé de rajouter plus d'aléatoire 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 attirés par les bordures de la carte.
- Le dernier système intégré cherche une destination dans une certaine zone. Au lieu de chercher un point proche de lui, il cherche un chemin vers une destination aléatoire, et recommence lorsque ce point est atteint ou si aucun chemin complet n'a été trouvé.

2.1.2 Evenements

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

En outre, des zones de discussion ont été implémentées : ce sont des zones où les joueurs et le NPC peuvent interagir et se fondre dans la discussion pour se cacher. Les IA qui les traversent s'y arrêtent et repartent au bout d'une durée de temps aléatoire.

2.2 Carte du jeu

La carte est désormais finie, et de nouveaux éléments ont été ajoutés, comme un shader animé (créé avec l'outil Shader Graph) pour l'eau, faite par Dov, ou encore de nouvelles lumières dynamiques. Elle est aussi dotée de nombreuses échelles, qui permettent de fuir ses poursuivants de façon discrète, ainsi que de petites rues reliant les avenues. En outre, les nombreux NPC ainsi que les marchandises exposées au milieu des rues font aussi de bonnes diversions.

Mais la principale nouveauté est le mode nuit : en effet, il est désormais possible de passer du jour à la nuit grâce à de simples boutons-radios. Ce mode nuit, comme l'indique son nom, plonge l'environnement dans l'obscurité, et permet de faire ressortir la beauté de la ville endormie, tout en ajoutant un côté angoissant aux parties, qui deviennent *de facto* beaucoup plus animées.

2.3 Site Internet

2.3.1 réalisation

La réalisation d'un site internet pour le projet était un objectif programmé pour la deuxième soutenance, ce qui est maintenant chose faite. Pour ce dernier, nous avons retenu Bootstrap, qui est une collection d'outils HTML, CSS et Javascript apportant des éléments de site esthétiques

et simples à utiliser. Le thème Freelancer, élégant et sobre, nous a tapé dans l'oeil ; cependant, comme c'est un thème très communément utilisé, nous changerons à terme certains éléments afin d'y apposer notre signature. Bootstrap Studio nous a aidés à réaliser un site dynamique, mais ne s'est pas montré à la hauteur de nos espérances en matière de personnalisation. En effet, on ne peut pas modifier le code HTML (seulement ajouter / supprimer des blocs ou éditer les attributs), et les styles CSS ne sont modifiables qu'en créant une copie du fichier CSS d'origine.

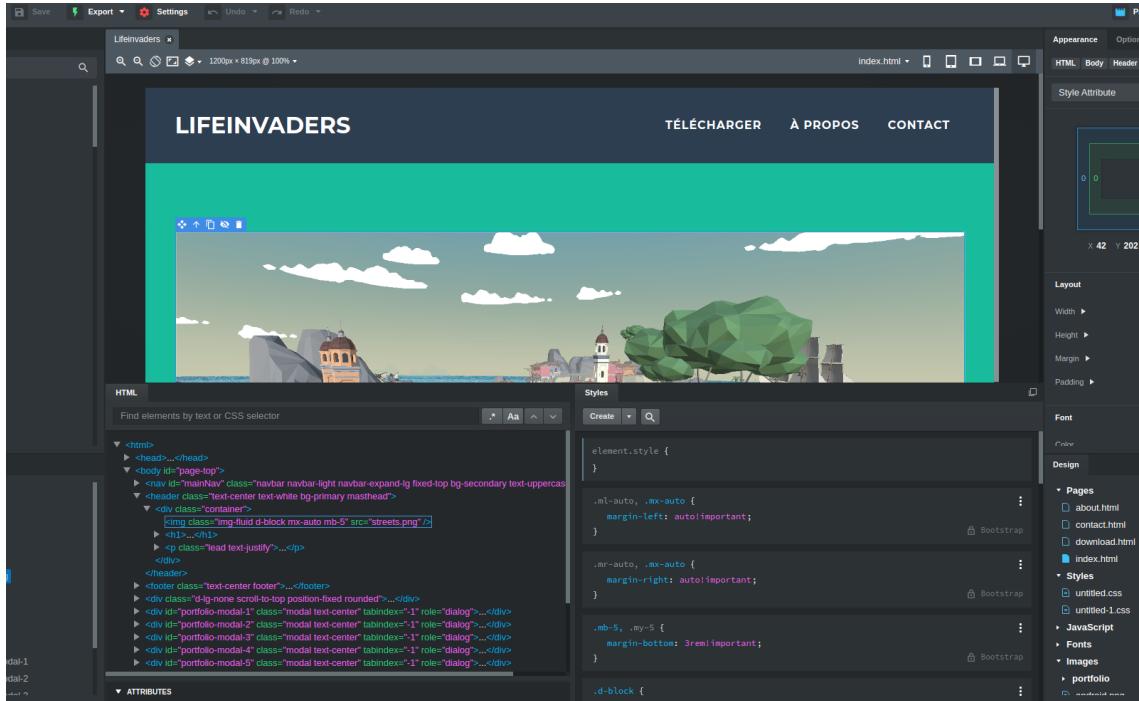


FIGURE 1 – Le logiciel Bootstrap Studio

La conception du site s'est donc déroulée en deux parties. En premier, la création d'une base grâce à BootStrap Studio, en modifiant les images, textes et titres présents, ainsi que d'autres chose un peu plus minutieuses, comme la modification de tableaux et des encarts personnalisés. Ensuite, la modification plus poussée des options verouillées par BootStrap Studio, comme l'ajout de liens sur les images, ou encore la modification des titres et favicons.

2.3.2 Hébergement

L'hébergement de petits sites comme celui-ci n'étant pas très contraignant, nous avons décidé d'utiliser un hébergeur gratuit, car ces derniers sont généralement largement suffisants. Après avoir cherché parmi les solutions proposées, nous avons décidé d'utiliser la solution *Github Pages*, qui permettait d'avoir une extension "sérieuse" (nous préférions une site qui finit par github.io que par wix.com), ainsi qu'une gestion de ce dernier très simplifiée, grâce au gestionnaire de versions. Ainsi, tout comme pour le projet, les versions sont gérées en trois commandes (git add, git commit , git push), et la limite de taille de 1 Go est plus que suffisante pour quatre pages html.

2.3.3 Améliorations futures

La création et la maintenance d'un page recensant les nouveautés apportées par chaque version est un des principaux points prévus pour la dernière soutenance, afin de pouvoir voir l'évolution du projet au fil des mois et d'être informé des dernières mises à jour du jeu. Un autre aspect important à développer est l'identité visuelle du site, afin de le rendre unique. Pour cela, nous comptons utiliser de polices de caractères originales et de couleurs vives et attrayantes. Des éléments dynamiques seront également nécessaires à rendre le site agréable à consulter.

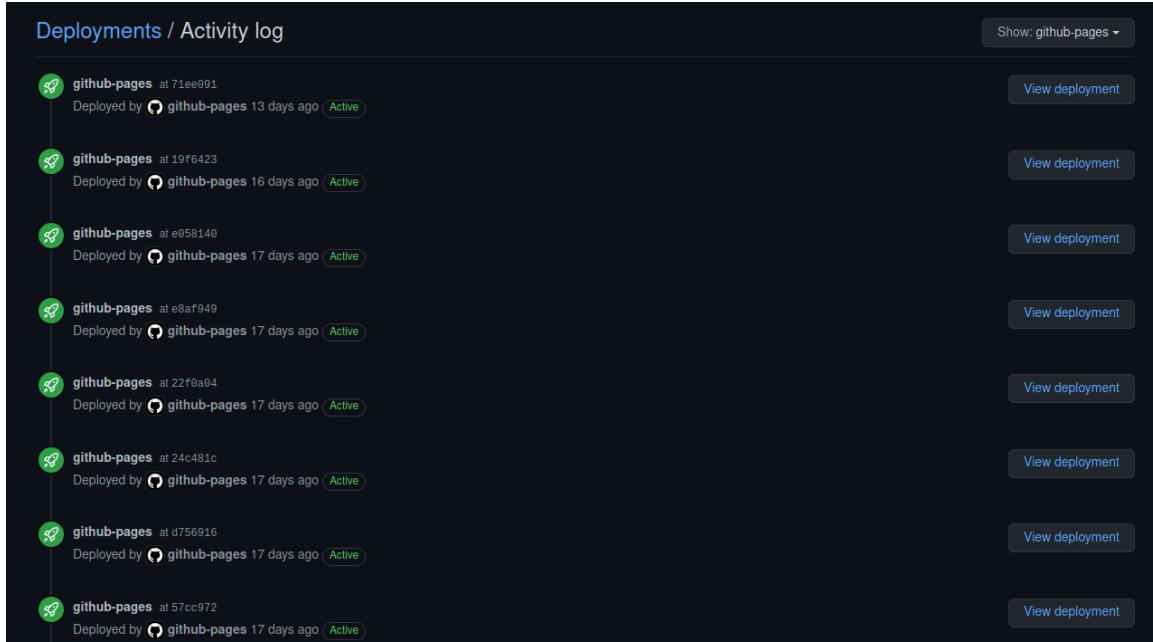


FIGURE 2 – Liste des différentes versions du site

2.4 Réseau

2.4.1 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 à l'aide d'un timer, qui est synchronisé entre tous les joueurs, y compris ceux rejoignant le lobby après son lancement. La synchronisation se fait grâce à une fonctionnalité des salles Photon permettant de créer des propriétés spécifiques, couplé à la propriété PhotonNetwork.Time qui est identique pour tous les clients d'une salle au même moment, permettant une synchronisation "parfaite" des timers. 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, quand le serveur est complet, le temps d'attente est réduit à 30 secondes. À la fin du timer, le Master Client charge la map de jeu, qui est synchronisé avec tous les joueurs.

2.4.2 Synchronisation du lancement de partie

Une fois la transition effectuée, il est nécessaire de synchroniser l'initialisation des différents composants permettant le fonctionnement d'une partie. Par exemple, il faut attendre que tous les joueurs finissent de charger la nouvelle scène avant d'instancier les joueurs sur la carte. Pour cela, un script s'occupe d'activer les différentes phases du lancement de partie suivant certaines conditions. Dans le cas de l'instantiation des joueurs, le script observe une propriété des joueurs déterminant si ces derniers ont chargé la map. Ainsi, le script de spawn des joueurs ne débute qu'une fois que tous les joueurs ont indiqué avoir chargé la carte.

2.4.3 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 cela : l'apparition (spawn) des joueurs. Pour cela, 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 apparaître chaque joueur à une position unique sur la carte : un point = un joueur, pas plus. Une fois cela fait, il faut également synchroniser la réapparition des joueurs, pour cela on applique le même système, en ne prenant en compte que les joueurs morts.

2.4.4 Synchronisation de l'IA

La grande difficulté du système multijoueur est la synchronisation des PNJs. En effet, c'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. Le Master Client instancie les personnages grâce à Photon ; ils sont donc au départ placés et visibles de la même façon pour tout les joueurs.

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. Il faut également synchroniser leur apparence. Encore une fois, cette dernière est déterminé par le Master Client puis partagé aux autres joueurs grâce à une méthode RPC.

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 cela, plusieurs méthodes ont été envisagées :

-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, fonctionnant 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. Cela 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 envoie uniquement l'array de positions générée par le Master Client. Ce système permet d'avoir une synchronisation satisfaisante des déplacement et une utilisation minime de la bande passante.

De plus, une fois le premier chemin créé et partagé aux joueurs de la salle, il est nécessaire d'activer le mouvement des PNJ de façon la plus simultanée possible. Le processus est le même que celui permettant de synchroniser les timers : on crée une propriété de la salle qui indique le moment où les IAs sont activés.

2.4.5 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 cela, il a été décidé d'utiliser le système 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 là que Photon rentre en jeu.

En effet, ce dernier permet de synchroniser le déclenchement d'évènements. Pour cela, 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 cet 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 reçue. Les détails du système de morts sont dans la partie gameplay.

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 système 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 reçue. Les détails du système de morts sont dans la partie gameplay.

2.4.6 Synchronisation du déroulé de partie

De la même manière, un système d'event a été mis en place permettant de synchroniser les différentes étapes du jeu.

2.5 Gameplay

2.5.1 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 films¹. 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 surposer les unes sur les autres.

2.5.2 Finishers

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 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 carte 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. Si le personnage tué était un joueur, alors il réapparaît discrètement ailleurs sur la map lors de sa réapparition. Le personnage tué disparaît de la carte au bout d'une dizaine de secondes avec un shader fait avec Shader Graph.

Voici quelques finishers que nous avons réalisé :

1. voir Effets Graphiques



FIGURE 3 – Quelques finishers

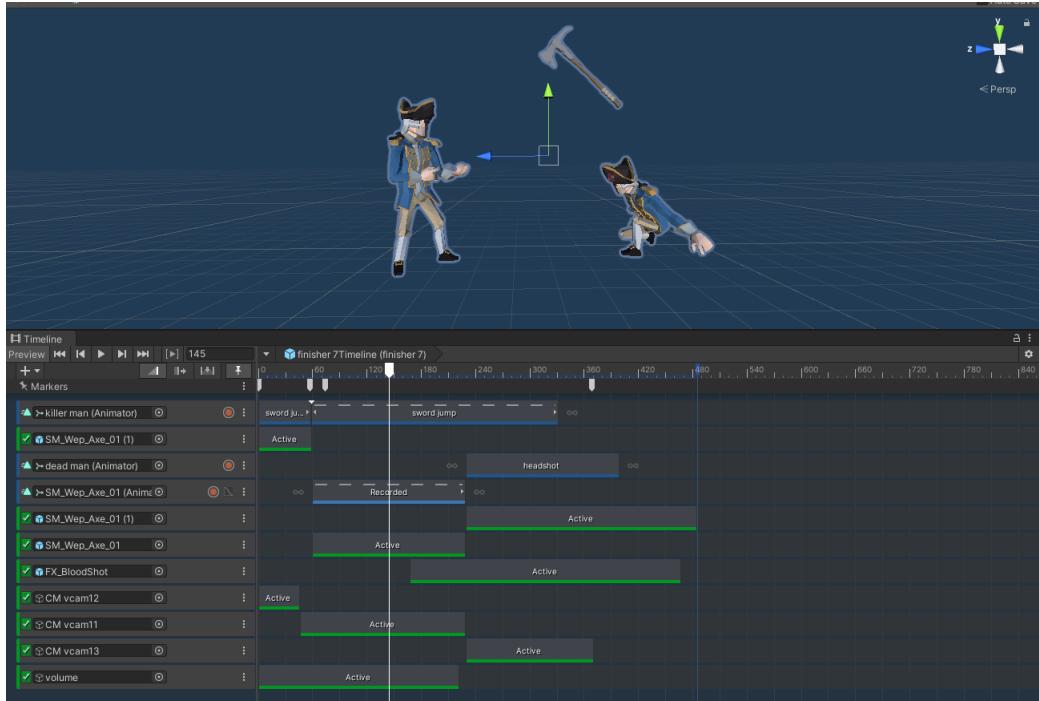


FIGURE 4 – Timeline du lancer de hache

2.5.3 Système de manche

Le système actuellement utilisé pour le déroulement d'une partie se base sur des manches. Le jeu se fait en deux phases : -Une période de 'grâce' de 30 secondes où les joueurs attendent l'assignation d'une cible. Ils sont libre de se déplacer, pour se cacher par exemple.

-Une période de 'chasse'. Les cibles sont assignés et le combat peut commencer. Elle est au départ de 3 minutes, mais ce temps diminue à chaque mort de joueur pour pousser les participants au meurtre.

2.5.4 Système de mort

Joueurs comme IA peuvent être tués. Dans le cas d'un joueur, un système de mort a été mis en place. Ainsi, quand le joueur se fait tuer, son personnage est désactivé temporairement et il rentre en mode "spectateur". Il peut se balader sur la map, mais il ne peut en aucun cas interagir avec l'environnement, et il n'est pas visible des autres joueurs.

2.5.5 Système de point

Un système de point a également été implémenté. Il fonctionne de la façon suivante :

-Tuer sa cible rapporte des points. Le premier joueur tuant sa cible gagne plus de points, les autres en gagnent moins.

-Tuer une IA fait perdre des points ! Il faut donc faire attention à ne pas tuer n'importe qui. Celà encourage la réflexion et pas simplement un carnage afin de trouver sa cible.

-Tuer la cible de quelqu'un d'autre ne fait pas perdre de points, mais un système de compensation a été mis en place. Ainsi, se faire voler sa cible par un autre joueur apporte des points de compensation. Encore une fois, il n'est donc pas dans l'intérêt des joueurs de tuer le premier venu !

-Une autre façon de gagner des points et d'être encore en vie à la fin d'une manche.

Bien évidemment, tout cela est dans le but de déterminer le vainqueur.

2.6 Graphismes

Pour rendre le jeu plus beau, nous avons intégré plusieurs effets. Afin de les créer, nous avons utilisé plusieurs outils de Unity : Shader Graph, TimeLine, Visual Effect FX Graph, Cinemachine.

Voici plusieurs exemples des éléments graphique que nous avons pu réaliser :

2.6.1 Shader Eau

L'eau que nous avions de base était plate mais surtout pas animé. Plutôt que d'acheter un asset permettant d'avoir de l'eau tout prête, nous avons opté pour une réalisation manuelle via l'outil du Shader Graph.

Il y a trois blocs pour les vagues, les petites, les moyennes et les grandes, plus des blocs sinusoïdaux pour que chaque bloc d'eau soit aligné et synchronisé les uns les autres.

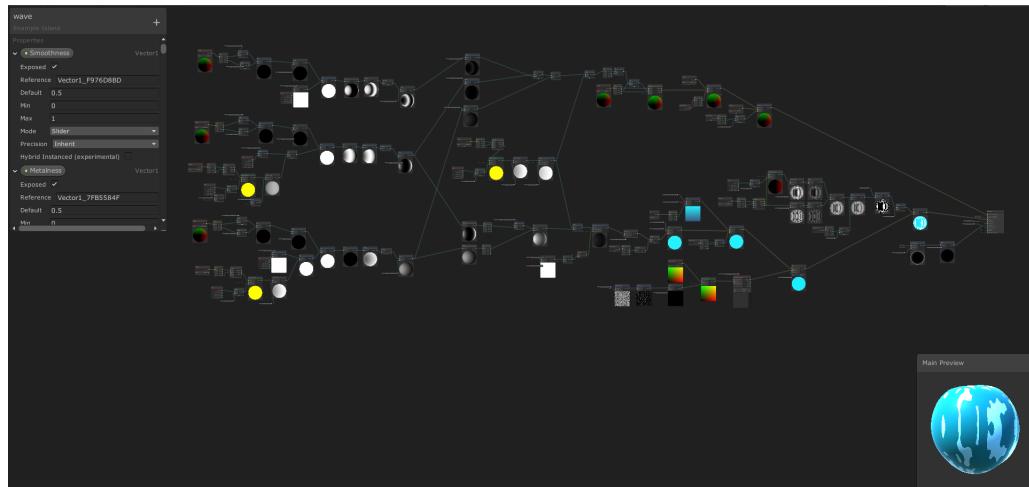


FIGURE 5 – Shader Graph de l'eau

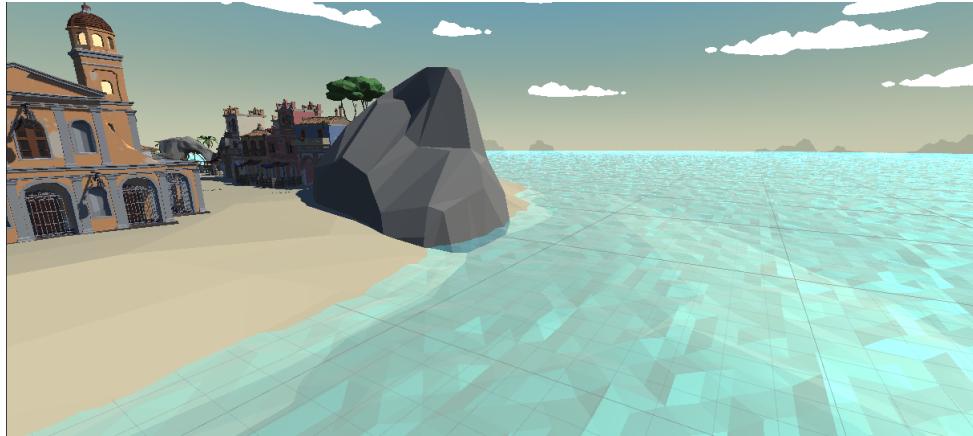


FIGURE 6 – L'eau avant

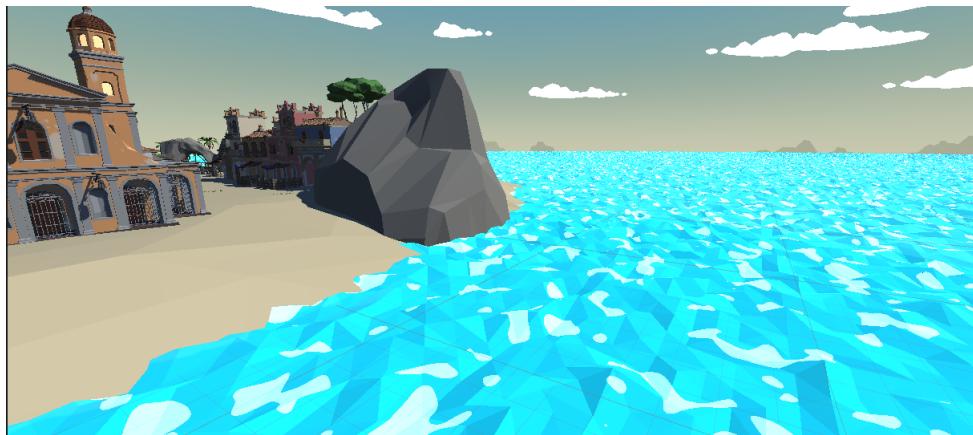


FIGURE 7 – L'eau après

2.6.2 Mode Jour/Nuit

Le mode jour/nuit permet au lancement de la partie de choisir soit le jour, soit la nuit. Si l'on choisit le mode nuit, un script désactive la lampe de jour, pour la remplacer par une autre plus sombre. Des FX de pluie ont été rajoutés en plus d'effets post processing. Mais l'effet le plus important est d'activer toutes les lampes de la carte.

2.6.3 Shader de disparition/apparition

Pour éviter de faire apparaître ou faire disparaître en un instant les personnages, nous avons décidé de créer un shader qui selon un matériau en paramètre, fait une transition de l'état initial vers l'état final.

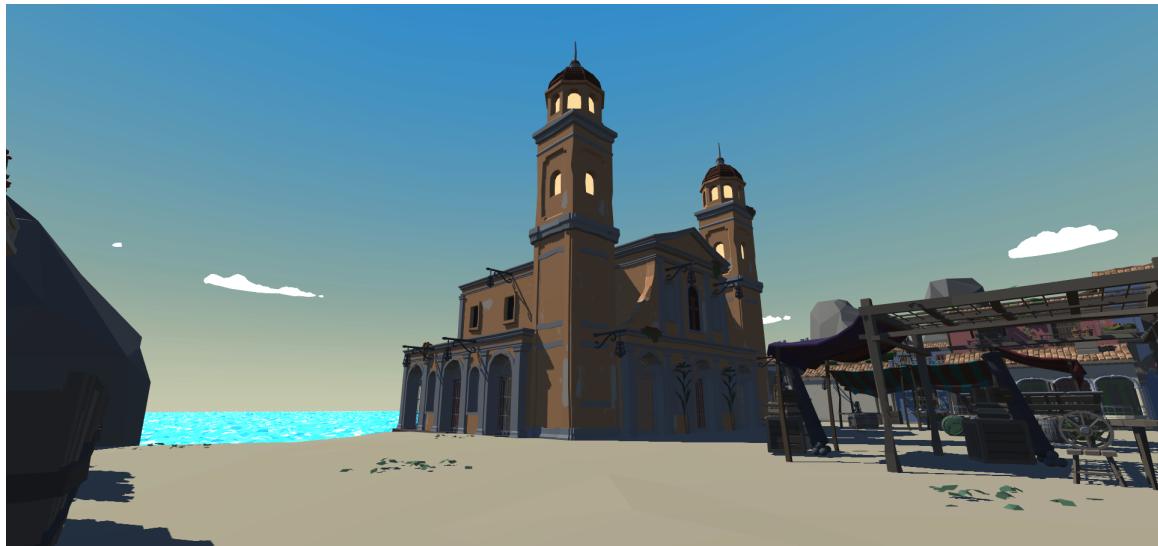


FIGURE 8 – La carte de jour...



FIGURE 9 – ...et de nuit.



FIGURE 10 – Application du shader sur un personnage

Le problème que nous avions rencontré était que lorsque nous appliquions le matériau avec le shader, les paramètres de l'instance du shader sur le matériau étaient communs. Résultat : lorsque un premier personnage mourrait puis dans la foulée un autre mourait également, alors le premier mort recommençait l'animation de transition en plus de changer de couleur de vêtements plus de peau. La solution fut toute simple, il nous a suffit en runtime de créer un nouveau matériau et d'appliquer les paramètres du matériau dans le script.

2.6.4 Post Processing

Nous avons ajouté le package Post Processing de Unity qui avec l'URP (Universal Render Pipeline) nous a permis de créer des ambiances et effets visuels sympathiques. Nous avons créé des volumes, où quand la caméra rentre dans le volume, alors les effets sont activés sur la caméra active. Cela nous a permis notamment d'avoir un effet vieux film/sépia lorsque le joueur passe en mode verrouillage



FIGURE 11 – Mode verrouillage

Si l'on s'attarde sur cette figure, nous voyons que l'effet "sépia" n'est pas visible sur certains objets. Ce fut toute la difficulté d'avoir des effets de Post Processing différents sur chaque layer. Nous avons donc ajouté plusieurs caméra Overlay superposées sur la caméra principale qui affichent chacune une partie de l'image. Après avoir réglé des soucis de profondeur sur les caméras, nous avons l'effet ci dessus.

3 Conclusion

Table des figures

1	Le logiciel Bootstrap Studio	3
2	Liste des différentes versions du site	4
3	Quelques finishers	7
4	Timeline du lancer de hache	8
5	Shader Graph de l'eau	9
6	L'eau avant	10
7	L'eau après	10
8	La carte de jour...	11
9	...et de nuit.	11
10	Application du shader sur un personnage	12
11	Mode verrouillage	13

Liste des tableaux