

Panic At Tortuga

Rapport de projet

Juin 2021



Table des matières

1	Introduction	2
2	Développement	3
2.1	Chronologie du projet	3
2.2	Progression	3
2.2.1	Première soutenance	3
2.2.2	Deuxième soutenance	4
2.2.3	Dernière soutenance	4
2.3	Launcher	4
2.4	Intelligence Artificielle et NPC	5
2.4.1	Première soutenance	5
2.4.2	Seconde soutenance	6
2.5	Carte du jeu	6
2.5.1	Première soutenance	6
2.5.2	Deuxième soutenance	9
2.5.3	Dernière soutenance	9
2.6	Interface utilisateur	9
2.6.1	Cahier des charges	9
2.6.2	Première soutenance	10
2.6.3	Deuxième soutenance	11
2.6.4	Troisième soutenance	11
2.7	Site Internet	12
2.7.1	Première soutenance	12
2.7.2	Seconde soutenance	13
2.8	Réseau	15
2.8.1	Cahier des charges	15
2.8.2	Première soutenance	15
2.8.3	Deuxième soutenance	16
2.9	Mécaniques de jeu	18
2.9.1	Cahier des charges	18
2.9.2	Première soutenance	19
2.9.3	Deuxième soutenance	21
2.10	Graphismes	24
2.10.1	Première soutenance	24
2.10.2	Shader Eau	24
2.10.3	Mode Jour/Nuit	25
2.10.4	Shader de disparition/apparition	26
2.10.5	Post Processing	27
2.11	Avancées et prévisions	28
2.11.1	Cahier des charges	28
2.11.2	Première soutenance	28
2.11.3	Prévisions pour la deuxième soutenance	28
2.11.4	Deuxième soutenance	29
2.11.5	Prévisions pour la troisième soutenance	29
2.12	Troisième soutenance	31
3	Conclusion	32

1 Introduction

L'heure du rendu final a sonné. Cette dernière période de travail n'a pas été la plus calme, car rythmée par un TD de maths et les partiels de fin d'année. Cependant, le projet n'a pas cessé d'évoluer pour atteindre sa forme finale, plutôt aboutie.

2 Développement

2.1 Chronologie du projet



FIGURE 1 – Frise chronologique du développement du projet

2.2 Progression

2.2.1 Première soutenance

Le système de progression, qui reste pour l'instant une tâche secondaire, a tout de même convenablement avancé. En effet même s'il n'y a pas encore de mécanique de jeu permettant à

ce système de prendre sens, l'objectif principal pour cette première soutenance a été réalisé : le système de sauvegarde.

Système de sauvegarde Un système de sauvegarde complet a été implémenté. Une classe "PlayerDatabase" a été créée et permet de stocker l'ensemble des variables que nous souhaitons sauvegarder. Ce système est utilisé non seulement pour sauvegarder la progression du joueur, mais il se trouve qu'il a également une grande utilité pour sauvegarder d'autres paramètres, comme par exemple la persistance du changement des contrôles effectués par le joueur (cf. menu d'accueil). Il n'existe qu'une seule instance de cette classe pour tout le jeu (singleton). Cette instance est sérialisée(processus permettant de convertir l'information dans un autre format, ici dans le but de la stocker, permettant ainsi sa persistance) puis sauvegardée dans un fichier. Au lancement du jeu, ce fichier est désérialisé(processus inverse de la sérialisation) est l'instance récupérée remplace donc l'instance unique présente en jeu. Il est important de faire cette démarche le plus tôt possible, afin que le chargement de l'instance soit faite avant toute tentative d'accès à cette dernière. Il faut également faire attention à sauvegarder régulièrement, notamment lors de la modification de l'instance.

Contenu débloquable Même si le système de progression n'a pas encore été créé, nous avons déjà réfléchi sur la "récompense" de ce système de progression. Il serait donc intéressant d'explorer la possibilité de débloquer des effets "cosmétiques" qui serait visible dans le lobby¹. Un système de customisation de personnage a été créé, et nous avons pour objectif de permettre au joueur de débloquer ces éléments de customisation.

Système de customisation Même si cette mécanique est encore basique, le joueur a la possibilité de choisir l'apparence de son personnage dans le menu d'accueil². Cette dernière sera vue des autres joueurs du lobby. Le fonctionnement est le suivant : chaque caractéristique est associée à une variable. Ces variables composent un code unique associé à une combinaison spécifique qui sera alors affichée. De plus, ces variables sont sauvegardées et le choix est persistant (sauvegardé dans un fichier). Avant la fin du projet, l'objectif est de bloquer l'accès à certains éléments de customisation, et de permettre au joueur de les débloquer grâce à sa progression.

2.2.2 Deuxième soutenance

Le système de progression a peu avancé car nous l'avons trouvé d'une importance moindre que le multijoueur ou l'IA. Cependant, un système d'expérience, acquise à la fin de chaque partie en fonction du nombre de points obtenus, a été implémenté. Il y a également un système de niveau. Pour obtenir un niveau, il faut acquérir une certaine quantité d'expérience, calculé en fonction du niveau actuel. Plus le niveau est élevé, plus l'expérience demandé est élevé. Mais ce système n'a pas encore d'utilité dans le cadre de la progression.

2.2.3 Dernière soutenance

Le système de progression est fonctionnel, et inclue la sélection / sauvegarde des apparences, ainsi qu'un système de niveaux basé sur des points d'expérience gagnés à chaque fin de partie. Même si cela reste assez basique, il permet deux choses : permettre au joueur de choisir un skin qui lui correspond, et activer le système de récompense du cerveau après chaque partie gagnée, afin de rendre le jeu un peu addictif (oui, c'est mal).

2.3 Launcher

Le lanceur du jeu est un élément crucial, car il permet d'installer le jeu et de le maintenir à jour. Une première version du launcher avait été créée avec Avalonia, un Framework C#. Cependant, il faisait appel à des librairies externes, ce qui l'alourdissait et rendait son fonctionnement instable. C'est pour cette raison que nous avons décidé, pour la deuxième soutenance, de nous tourner vers

1. Cf. Multijoueur

2. Cf. Menu d'accueil

le Framework Electron, qui est parfaitement compatible avec toutes les plateformes car il repose sur du HTML.

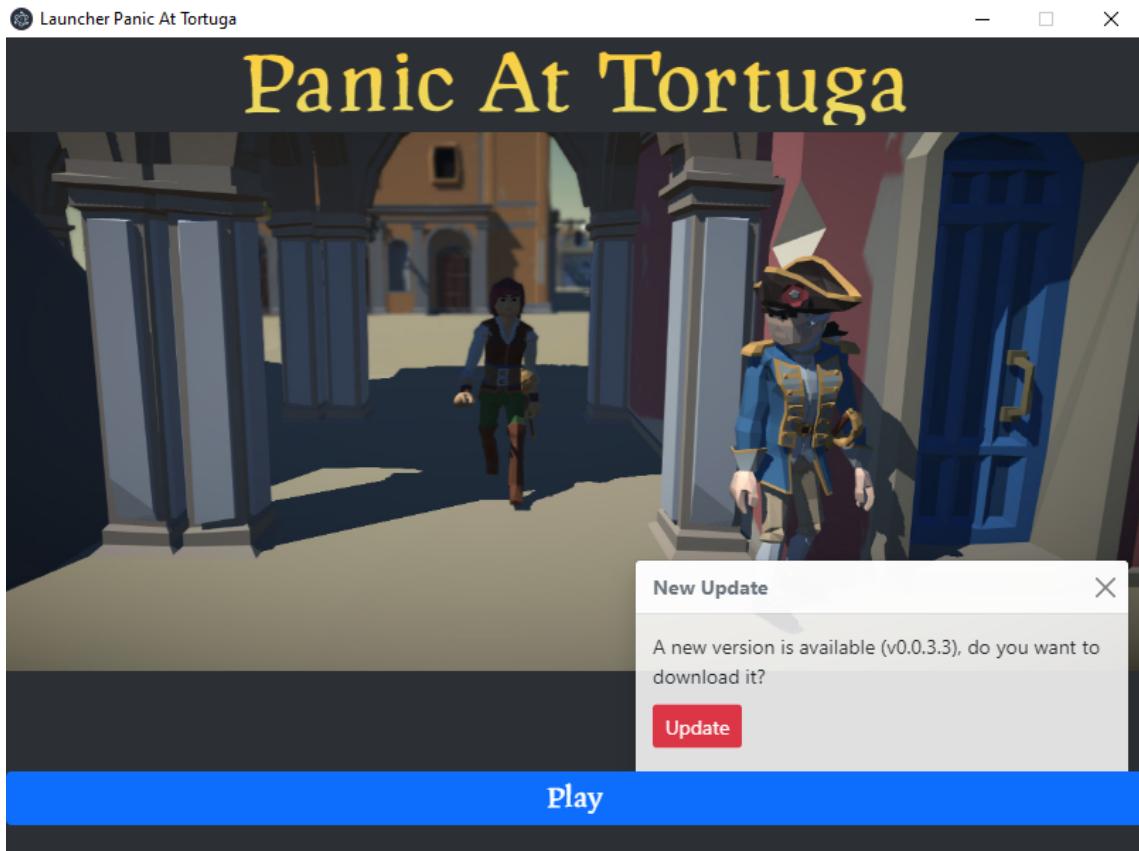


FIGURE 2 – Aperçu du Launcher

Le lanceur du jeu sous Electron est capable de se connecter au repo Github et de récupérer la dernière version du jeu afin de la comparer à la version actuelle pour proposer une éventuelle mise à jour. Cela permet de constamment maintenir le jeu à jour, pour éviter des conflits de versions ou toutes autres erreurs. Le html utilisé permet aussi de s'adapter parfaitement à toutes les formes de fenêtres sans problèmes d'échelles ou d'overlay.

2.4 Intelligence Artificielle et NPC

2.4.1 Première soutenance

Pour rendre notre jeu plus vivant et permettre aux joueurs de se mêler à la foule, il nous fallait créer des IA se déplaçant dans la ville. Pour cela, nous avons décidé d'utiliser des NavMesh pour créer des zones où les NPC peuvent se balader d'un point à un autre. Pour ne pas rendre leur comportement linéaire et trop prévisible, ce qui gâcherait inéluctablement le jeu, ils se déplacent de manière aléatoire vers un point quelconque défini dans une liste de coordonnées. Les NPC ont donc des comportements pseudo-aléatoires (même s'ils empruntent toujours le chemin le plus court entre deux objectifs) qui permettent aux joueurs de se fondre plus facilement dans la masse. En d'autres termes, en faisait se comporter les IA comme des joueurs, on évite aux joueurs d'avoir à se comporter comme des robots.

L'outil de navigation est assez poussé, et permet de définir les dimensions des personnages, ainsi que la hauteur de laquelle ils peuvent sauter et les pentes qu'ils peuvent emprunter. Ici, la pente maximale est de 33.4°, mais si ce chiffre avait été plus élevé, les pentes seraient devenues praticables (bleues)³, et les personnages auraient pu monter sans utiliser les escaliers (ce qui n'est

3. Voir figure 3

évidemment pas le but).

Pour rendre ces IA plus humaines, les joueurs peuvent les bousculer et les étourdir en courant vers eux. Ils reprennent leur trajet au bout de quelques secondes. Il faudra ainsi, dans les prochaines versions, améliorer la navigation afin que le chemin emprunté ne soit pas toujours le plus court.



FIGURE 3 – Problème rencontré lorsque de nombreuses IA vont au même endroit

2.4.2 Seconde soutenance

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 coordonnées prédéfinie. Le tracé était de ce fait trop prévisible et ne permettait pas de se déplacer librement 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 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.

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, parlent, avant de repartir au bout d'une durée de temps aléatoire.

2.5 Carte du jeu

2.5.1 Première soutenance

La carte étant un élément crucial du jeu, au même titre que les mécaniques, il nous a semblé important de faire des schémas et que le groupe soit d'accord sur la direction artistique, afin que



FIGURE 4 – Représentation vectorielle des path des IA



FIGURE 5 – Zone de discussion

Paul, la personne en charge de la map, puisse être sûr de la vision de la carte du jeu avant de commencer. Ainsi, la carte finalement retenue devait avoir suffisamment de relief pour rendre les échelles intéressantes, et suffisamment spacieuse pour pouvoir la remplir avec au moins 150 personnages non-joueurs (afin de pimenter le jeu). Dans un premier temps, une forme ronde avait été retenue pour la carte, mais a ensuite évolué pour une forme en L, cette dernière rendant l'environnement plus naturel et augmentant les chances des personnages de se croiser.

Ensuite, afin d'ajouter du relief à la carte, une colline a été créée. Cette dernière s'étend sur environ un quart de la carte, et possède quatre niveaux afin d'en permettre l'accès par de petits escaliers successifs. Une colline étant un terrain irrégulier, il a été décidé que les bâtiments placés sur cette dernière ne seraient pas parallèles, mais répartis afin de créer un imbroglio de maisons



FIGURE 6 – Vue aérienne de la carte

rappelant le style méditerranéen dont les îles comme celle-ci sont inspirées.



FIGURE 7 – Colline de la carte, organisée par étages

Enfin, l'architecture de la ville elle-même devait aussi avoir une influence ibérique, les maisons ont été dessinées basses et organisées autour de places et marchés animés. Les tonnelles et les nombreuses lanternes rendent l'environnement plus chaleureux, et les arcades, dotées de portes qui se ferment lorsqu'un joueur les passe en courant, ajoutent une mécanique de fuite au jeu⁴. L'organisation du village se fait autour de la place de l'église, qui fait office de place du marché.

4. Cf. Gameplay/Environnement

2.5.2 Deuxième soutenance

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 venelles reliant les avenues. En outre, les nombreux NPC ainsi que les marchandises exposées au milieu des rues font aussi de bonnes diversions. Enfin, l'ajout d'escaliers offrant un second accès à la colline permettent non seulement de désengorger la butte, envahie par les NPC, mais aussi de redynamiser la digue qui était jusque là exempte de tout intérêt : pas de bâtiments, pas de cachettes...

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. Le mode nuit applique des effets de post-processing à toutes les textures de la carte, et assombrissant les couleurs et en appliquant certains effets visuels se traduisant en jeu par un environnement plus sombre (deonc nocturne). Ce mode nuit 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.

Il reste à peaufiner les détails, comme l'ajout de hautes herbes dans les rues, le placement de torches utilisant la lumière dynamique (mode jour / nuit) ou encore le réajustement de petites erreurs de placements.

2.5.3 Dernière soutenance

2.6 Interface utilisateur

2.6.1 Cahier des charges

Afin de faire l'UI, nous avons majoritairement utilisé l'outil de Unity Canvas. Pour la direction artistique, compte tenu du thème de notre jeu portant sur les pirates, nous avons décidé de nous inspirer du menu principal du jeu Sea Of Thieves.

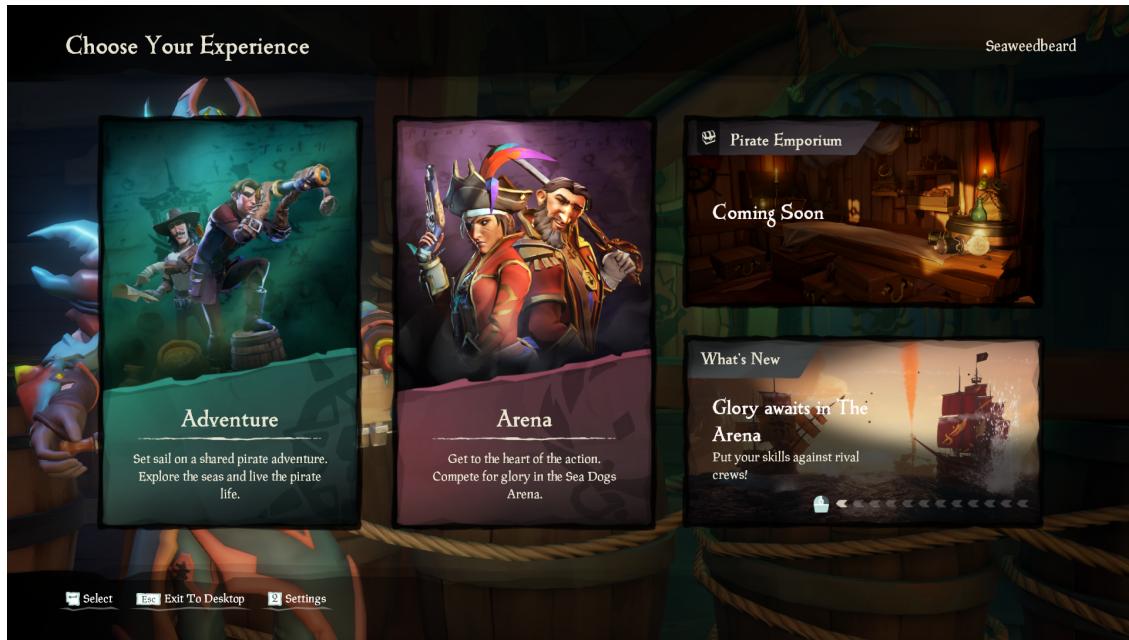


FIGURE 8 – Menu principal du jeu Sea of Thieves



FIGURE 9 – Préversion de notre menu

Afin de rendre le menu plus vivant, nous pourrions utiliser des animations sur les boutons ainsi que sur l'arrière-plan du menu. Notre menu intègrera 4 boutons et un champ. Le champ permettra au joueur d'entrer son nom dans le jeu.

Les 4 autres boutons serviront à :

- Se connecter à une partie
- Quitter le jeu
- Régler les paramètres du jeu
- Personnaliser son personnage

2.6.2 Première soutenance

Tableau des scores Pendant le déroulement du jeu, le joueur a besoin de connaître certaines informations concernant son personnage, la partie et les autres joueurs. Pour fournir ces informations, nous avons opté pour un tableau des scores disponible à n'importe quel moment de la partie. Ce tableau permet également un classement simple, clair et rapide des joueurs en fonction de leurs points. La difficulté principale de ce tableau des scores est la synchronisation entre un script côté client qui doit actualiser à chaque pression de la touche TAB le score ainsi que la présence de chaque joueur et la récupération des données de chaque Player, mises à jour en temps réel sur le serveur.



FIGURE 10 – Première version du tableau des scores

2.6.3 Deuxième soutenance

Menu principal Le menu principal a été amélioré, afin d'y ajouter des paramètres comme la sélection du mode de fenêtre (fenêtré ou plein-écran) ou le format de la fenêtre (compatible avec des formats 16 :9, 4 :3 et 16 :10). Il comporte maintenant les quatre options annoncées dans le cahier des charges, à savoir jouer, modifier le personnage, régler les paramètres et quitter.

Un début de boussole a aussi été implémenté, mais n'est pas accessible en jeu car encore au stade de développement. Cette dernière, présente en bas à droite de l'écran, permet de localiser sa cible (avec une précision proportionnelle à la distance à la cible).

2.6.4 Troisième soutenance

Menu Principal Le menu principal a encore été amélioré, et est maintenant très semblable à celui de Sea of Thieves, avec des boutons recouverts par des images pour donner du cachet au jeu. Des danses de victoire ayant également été ajoutées au jeu, un menu de sélection est maintenant disponible dans les options de configuration du joueur, au même endroit que le choix de l'apparence.

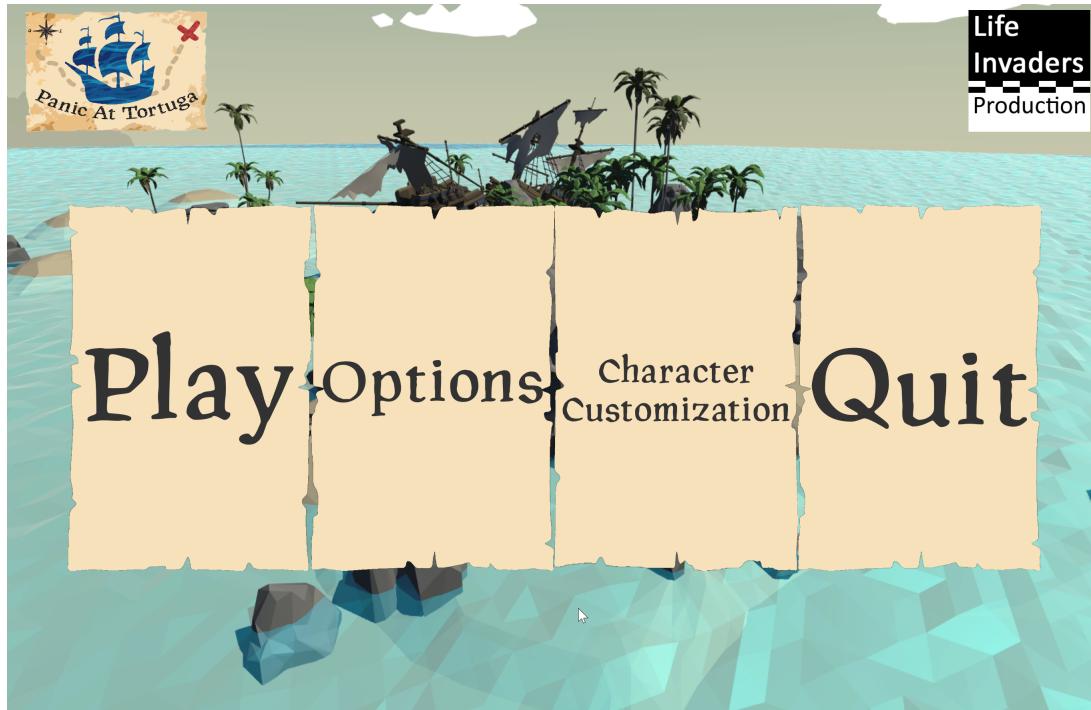


FIGURE 11 – Version antérieure du menu principal



FIGURE 12 – Dernière version du menu

2.7 Site Internet

2.7.1 Première soutenance

Au moment de la première soutenance, seule une page HTML de démo était disponible sur le site, afin de vérifier que la mise en place de GitHub Pages avait bien marché. Un template HTML, plus abouti, était également hébergé en local, mais nous avons finalement décidé d'utiliser GitHub Pages, premièrement pour son accessibilité rendant le site modifiable plus rapidement par n'importe quel membre de l'équipe, et aussi pour des raisons évidentes de sécurité et de disponibilité (pas terrible d'ouvrir un port de sa box, et je n'allais pas ouvrir un port à chaque fois lors de mes

déplacements).

2.7.2 Seconde soutenance

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, 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.

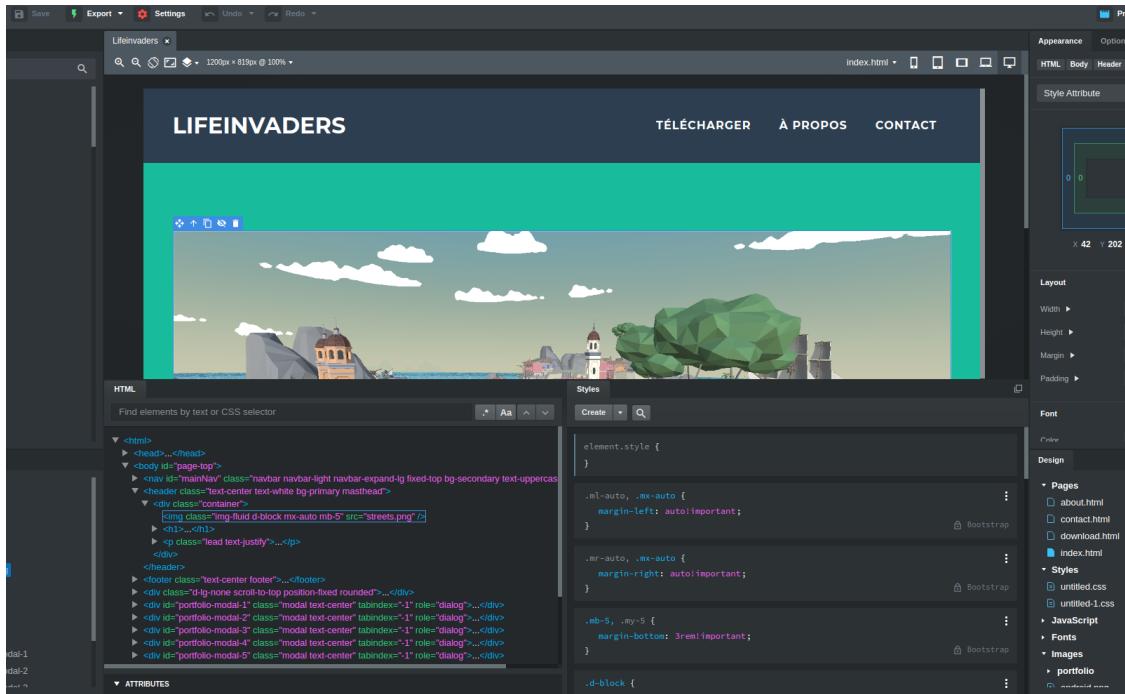


FIGURE 13 – Le logiciel Bootstrap Studio

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 choses un peu plus minutieuses, comme la modification de tableaux et des encarts personnalisés. Ensuite, la modification plus poussée des options proposées par Bootstrap Studio, comme l'ajout de liens sur les images, ou encore la modification des titres et favicons.



FIGURE 14 – Aperçu de notre site

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`.

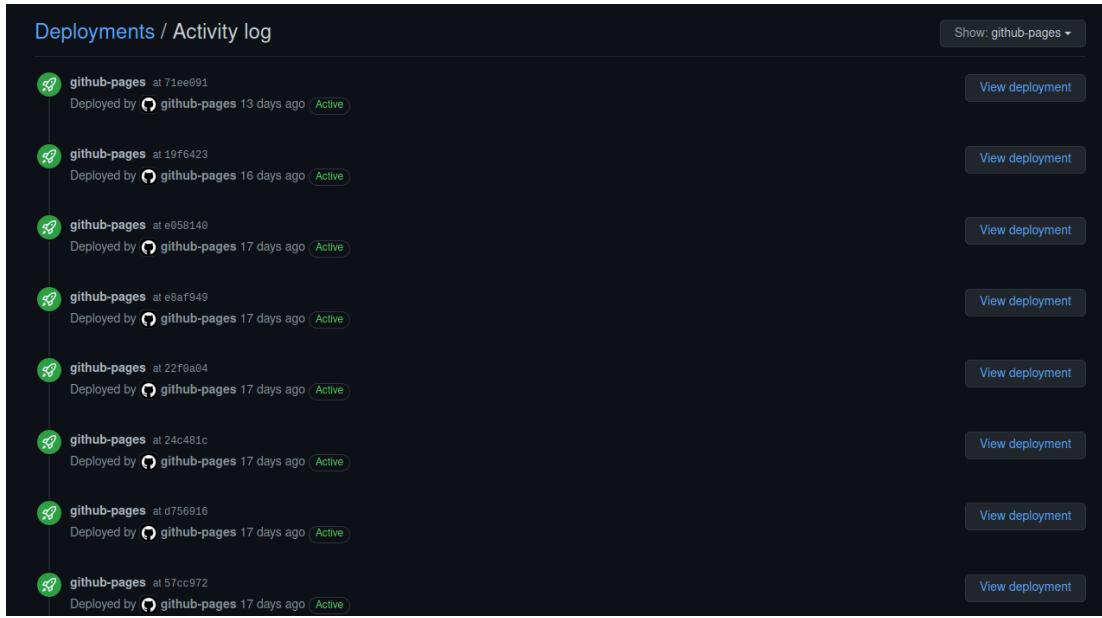


FIGURE 15 – Liste des différentes versions du site

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'indentité 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.

2.8 Réseau

2.8.1 Cahier des charges

Réalisation du multijoueur Afin de développer le cœur de notre jeu, le système permettant de connecter les joueurs ensemble, nous avons utilisé la framework Photon, et plus précisément sa version la plus aboutie, Photon Unity Network 2 (PUN2).

Photon est un outil pour Unity qui joue le rôle de framework dans la création d'environnements multijoueurs. Il offre de nombreuses fonctionnalités comme :

- Des serveurs dédiés à couverture mondiale
- La création de lobbies permettant aux joueurs de facilement se connecter à une partie
- Un système RPC permettant la communication joueur-joueur.

Photon fonctionne grâce à un système de méthodes RPCs. Un joueur génère un message RPC qui est distribué par un serveur Photon aux autres joueurs selon un filtre. On peut donc facilement synchroniser divers paramètres essentiels à la réalisation d'un jeu multijoueur.

Dans le cadre de notre jeu, ce framework est nécessaire pour connecter les joueurs entre eux et synchroniser les déplacements des joueurs et de l'IA ainsi que certains événements tels que le commencement d'une partie ou le partage des cibles, mais également pour communiquer les actions réalisées par les joueurs (tel que l'exécution d'une cible) à tout le lobby.

2.8.2 Première soutenance

Connexion aux serveurs Photon Avant toute chose, il est essentiel de pouvoir connecter le joueur à un serveur de jeu. Grâce à Photon, qui fournit non seulement un serveur gratuit d'une capacité de 20 joueurs simultanés, mais également un accès facile à ce dernier grâce à son API, connecter les joueurs au serveur est chose aisée. Il suffit d'un appel de méthode pour connecter et déconnecter le joueur. Simple et efficace.

Création d'une salle Photon se base sur un système de salle. Ces dernières permettent aux joueurs présent à l'intérieur de communiquer entre eux. Il est actuellement possible de créer ainsi que de rejoindre une salle. Ces deux processus se font automatiquement, sans effort du joueur, mais il est prévu de donner un plus grand contrôle sur ce système aux clients. En effet, il suffit d'appuyer sur un bouton pour tenter de rejoindre une salle. Si aucune salle est disponible, alors le client en crée une qui se rend disponible aux autres joueurs. Ce système a été implémenté par Julien et Harrys.

Instantiation et synchronisation C'est Harrys qui s'est occupé de cette partie. Une fois que le joueur accède à une salle, celui-ci charge une scène, la même que tous les autres joueurs. Photon permet alors d'instancier son personnage, qui sera visible par tout les autres joueurs présent dans la salle. Cependant cela ne suffit pas, car il faut ensuite permettre la synchronisation des mouvements des joueurs. Pour cela, on utilise un composant appelé PhotonView. Ce dernier permet de synchroniser diverses variables spécifiques à un objet comme sa position ou ses animations. De plus ce composant est nécessaire pour la communication par "RPC" qui est essentiel à la réalisation du multijoueur. Certains éléments, tels que ceux permettant de contrôler le personnage grâce à l'utilisation du clavier ou bien d'une manette, doivent cependant être pris en compte lors de ce processus. Chaque client doit désactiver les composants problématiques des personnages qu'il ne "possède" pas. On utilise le système d'appartenance d'objet qui, tout comme la synchronisation de mouvement, se fait grâce au composant PhotonView. La scène actuel d'instantiation servant de lobby, divers ajouts "cosmétiques" ont été effectués comme l'apparition des pseudos au dessus de chaque joueur ou bien l'implémentation du choix de l'apparence. Un système de "timer", synchronisé pour tous les joueurs d'une salle est actuellement en cours de développement. Par la suite, les différentes actions pouvant être réalisées par les joueurs devront également être synchronisés. Cependant le travail effectué jusqu'à présent facilitera grandement cette tâche.



FIGURE 16 – Photo de groupe en multijoueur

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 tout 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 tout 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. A la fin du timer, le Master Client charge la map de jeu, qui est synchronisé avec tout les joueurs.

2.8.3 Deuxième soutenance

Synchronisation du lancement de partie Une fois la transition effectué, il est nécessaire de synchroniser l'initialisation des différents composants permettant le fonctionnement d'une partie. Par exemple, il faut attendre que tout 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 chargés la map. Ainsi, le script de spawn des joueurs ne débute qu'une fois que tout les joueurs ont indiqué avoir chargé la carte.

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. cela 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.

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 exact où les IAs sont activés, une fonctionnalité de Photon permettant l'accès à une valeur identique sur tout les clients au même instant (PhotonNetwork.Time).

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 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 la que `Photon` rentre en jeu.

En effet, ce dernier permet de synchroniser le déclenchement d'évènement. 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 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`.

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, avec une période de synchronisation des IA en début de partie, différentes manches et enfin une phase de fin avec les stats.

Chat Photon Enfin, un chat a été implémenté grâce à une librairie de `Photon`. Ce chat permet aux joueurs d'une salle de communiquer entre eux, dans le lobby et en jeu.

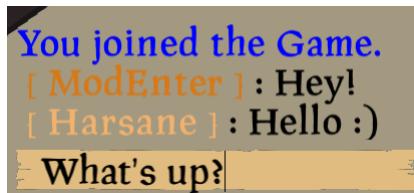


FIGURE 17 – Démonstration du chat Photon

2.9 Mécaniques de jeu

2.9.1 Cahier des charges

Objectif du jeu Chaque joueur se voit attribuer une cible et est lui-même la cible d'un autre joueur. Nous intégrerons donc un système de sélection de cibles, et un moyen pour le joueur de sélectionner un personnage (joueur ou non) à proximité de lui.

Le but est de tuer le plus de joueurs cibles dans le temps imparti. Chaque assassinat rapporte X points aux joueurs.

À l'aide de classes de personnage, nos personnages **pourraient** avoir des pouvoirs spécifiques, tel que se déguiser en quelqu'un d'autre pendant une durée limitée, pouvoir tuer à moyenne distance, ralentir et affaiblir un joueur (l'empêcher de courir, d'utiliser des capacités par exemple), avoir une boussole plus précise...

Nous avons également rajouté des objets pouvant être utilisés par les joueurs (Echelles⁵, Portes) et comptons en rajouter d'autres.

Organisation de partie L'organisation de partie concerne la création des scripts et ressources permettant à chaque partie multijoueur de se dérouler suivant un schéma prédéterminé. Cela comprend le début et la fin de manche (choix du gagnant, présentation du classement de fin de partie

5. Voir Figure 4

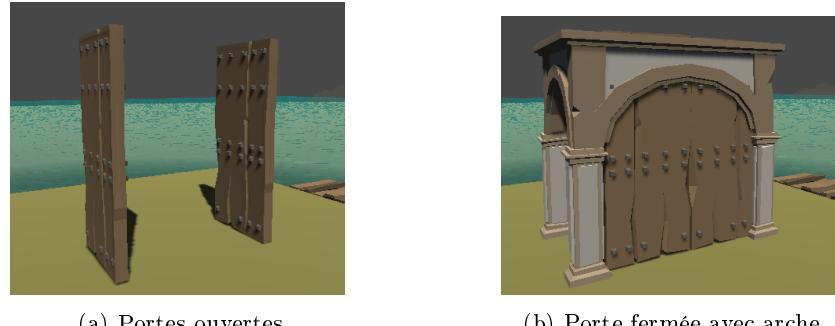


FIGURE 18 – Exemple des portes que nous avons réalisé

par exemple), le placement des éléments dynamiques de la scène, les timers, le comptage des points, l’assignation des cibles...

Pour offrir la meilleure expérience de jeu possible, cette organisation doit permettre à chaque manche d’être unique et imprévisible. Pour cela, nous introduirons de la complexité dans les différentes mécaniques de jeu. Par exemple, on peut attribuer plus de points au premier joueur ayant abattu sa cible ou à un joueur ayant survécu un certain temps sans se faire tuer.

2.9.2 Première soutenance

Environnement Nous nous sommes beaucoup inspirés du mode multijoueur des premiers Assassin’s Creed. Le système de grimpe s’étant montré trop complexe à mettre en place, nous avons réutilisé quelques idées de gameplay pour l’environnement, comme les échelles et les portes.



FIGURE 19 – Joueur grimpant sur l’échelle

Les échelles permettent de créer un peu de verticalité dans nos niveaux. Elles ne peuvent être utilisées que par les joueurs, mais ont leurs avantages comme leurs inconvénients : ainsi, prendre de la hauteur permet d'emprunter des raccourcis, mais retire la discréetion (car personne de civilisé

devrait être sur les toits!)

Les portes permettent de barrer le passage pour échapper à son agresseur. Elles se ferment quand le joueur passe dessus en courant et se rouvrent au bout de quelques secondes.



FIGURE 20 – Porte fermée après le passage d'un joueur

Ces interactions environnementales ont été réalisé par Renaud-Dov.

Déplacement du personnage C'est Renaud-Dov qui a réalisé les contrôles de déplacement du personnage :

Tout jeu est très rapidement limité par la capacité de déplacement du personnage et sa vitesse. Nous avons donc réalisé un script de déplacement qui permet au joueur de marcher, courir, sauter et grimper aux échelles.

Pour le côté technique, nous avons dans un premier temps utilisé l'Input System de base proposé par Unity. Mais plusieurs problèmes se sont posés :

- Les configurations des keymaps sont assez limitées
- Paramétrier des périphériques autre que le clavier/souris est compliqué. Il faudrait avoir des scripts différents pour chaque périphérique, et donc des prefabs différents

C'est pour cette raison que nous avons migré vers le New Input System. Celui ci est ergonomique, multiplateforme et permet l'utilisation de périphériques divers, comme la manette ou le clavier par exemple. Certains scripts ont dû être modifiés pour devenir compatibles avec ce système.

Il est donc actuellement possible de jouer au clavier/souris ou à la manette (pas les deux à fois).

Attribution des cibles Harrys a réalisé le script permettant l'attribution des cibles, qui utilise des méthodes RPC (communication inter-clients par l'intermédiaire d'un serveur). Le "Master-Client", un client qui est désigné dans chaque salle pour faire office de maître de jeu, est celui qui sélectionne la cible de chaque joueur. Il communique ensuite à chacun des clients présents dans la salle sa cible désignée. Cette attribution se fait de façon aléatoire, mais selon certaines règles :

- Un joueur ne peut (évidemment) pas être sa propre cible
- Deux joueurs ne peuvent pas être la cible l'un de l'autre (Cela permet une plus grande interaction entre les joueurs et pas simplement des scénarios en "1 contre 1").

— Plusieurs joueurs ne peuvent pas avoir la même cible.

Cependant même si ce script est fonctionnel, il n'a pas encore été implémenté. Le système d'élimination de la cible fera l'objet d'un travail important pour la prochaine soutenance.

Système de verrouillage Le système qui attribue des cibles à chaque joueur n'est pas entièrement implémenté, mais le joueur peut déjà tuer une cible, qu'elle soit la bonne ou non. Pour cela, il passe en mode verrouillage, et les personnages pointés par le viseur surbrillent. Pour les sélectionner, un coup de molette suffit, et le contour devient alors jaune, pour indiquer que la cible est verrouillée. Il suffit alors d'être à moins d'un mètre pour l'éliminer⁶.



(a) Personnage au centre de l'écran en surbrillance



(b) Personnage sélectionné et verrouillé



(c) Disparition des corps après mort

FIGURE 21 – Système de verrouillage

Pour le material qui fait disparaître les corps, nous avons créé manuellement un shader avec l'outil Shader Graph de Unity⁷. Nous pouvons donc créer des shaders personnalisés et dynamiques (avec comme exemple l'utilisation du temps).

2.9.3 Deuxième soutenance

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 et un effet réalisé grâce au post processing de Unity permet de donner un effet sépia/vieux films⁸. Un contour blanc autour des personnages visés au centre de l'écran permettent voir quel

6. Voir fig 6

7. Voir fig 7

8. voir Effets Graphiques

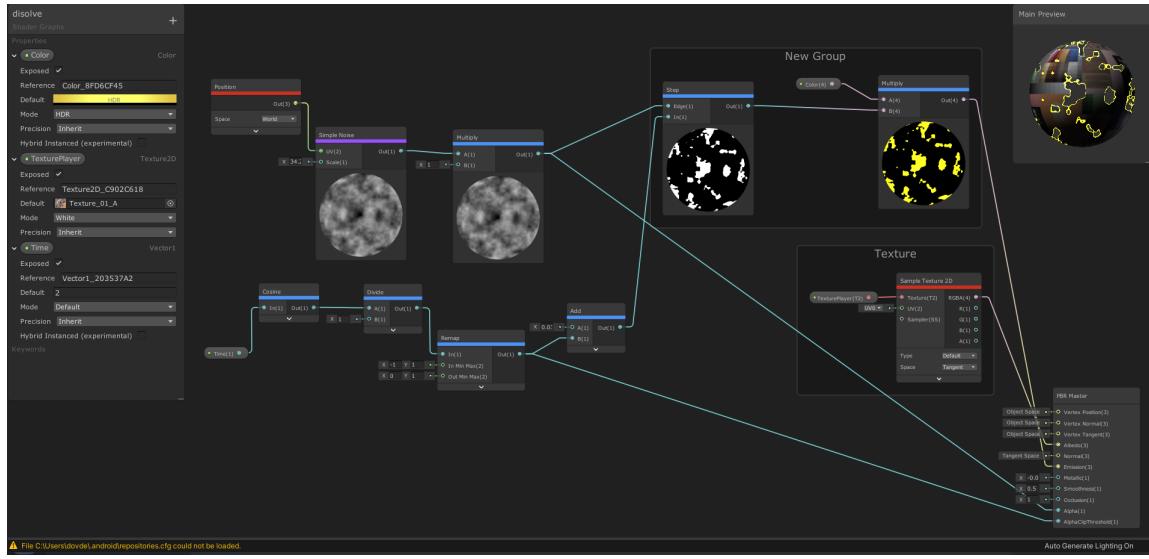


FIGURE 22 – Vue du shader créé depuis Shader Graph

personnage va être sélectionné.

Une fois sa cible sélectionnée, le joueur peut l'éliminer à condition d'en être suffisamment proche.

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

Finishers Nous avons fourni un vrai travail sur les différentes animations de mort 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 parfaitement synchronisées. Après avoir regardé plusieurs types de solutions, nous nous sommes tournés vers un outil préintgré appelé 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 la victime 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 terminée, un signal est envoyé à un script qui réaffiche alors le joueur qui était masqué. Si le personnage tué était un joueur, alors il réapparait 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 :



FIGURE 23 – Quelques finishers

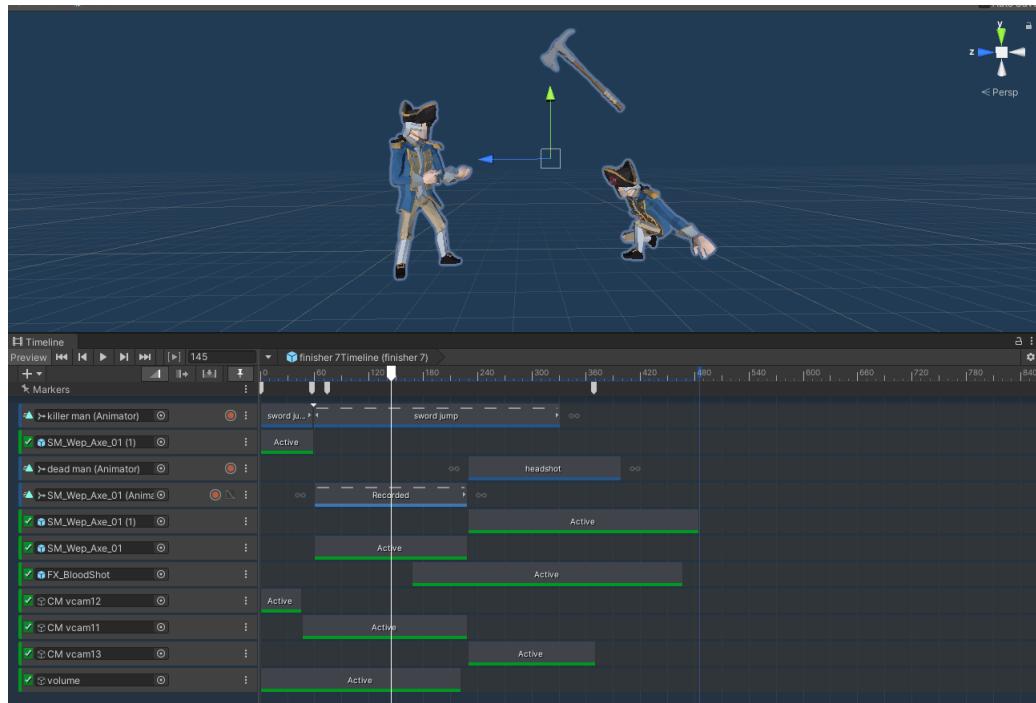


FIGURE 24 – Timeline du lancer de hache

*

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 libres de se déplacer, pour se cacher par exemple.

- Une période de 'chasse'. Les cibles sont assignées 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.

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. Une fois la manche terminée, le joueur réapparaît aléatoirement sur la map.

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 point, les autres en gagne de moins en 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. Tous ces éléments permettent de déterminer le classement de la partie.

Système de fin de partie Finir une partie est tout un processus. Il faut d'abord désactiver le mouvement des joueurs. On présente ensuite plusieurs informations comme le rang final, le nombre d'assassinats et de mort, ainsi que l'expérience acquise. Il y a également une sauvegarde de ses statistiques et une mise à jour du système de niveau dans le but de permettre leur utilisation éventuel par le système de progression. Ensuite, on propose au joueur de rejouer. S'il refuse, il quitte la salle et peut fermer le jeu ou rejoindre une autre salle. Sinon, quand tous les joueurs ont décidé de redémarrer ou de partir, les joueurs retournent sur le lobby dans l'attente d'une nouvelle partie.

2.10 Graphismes

2.10.1 Première soutenance

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 quelques exemples des éléments graphiques que nous avons pu réaliser :

2.10.2 Shader Eau

L'eau que nous avions était plate mais surtout n'était pas animée. Plutôt que d'acheter un asset permettant d'avoir de l'eau animée, nous avons opté pour une réalisation manuelle à l'aide de l'outil Shader Graph.

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

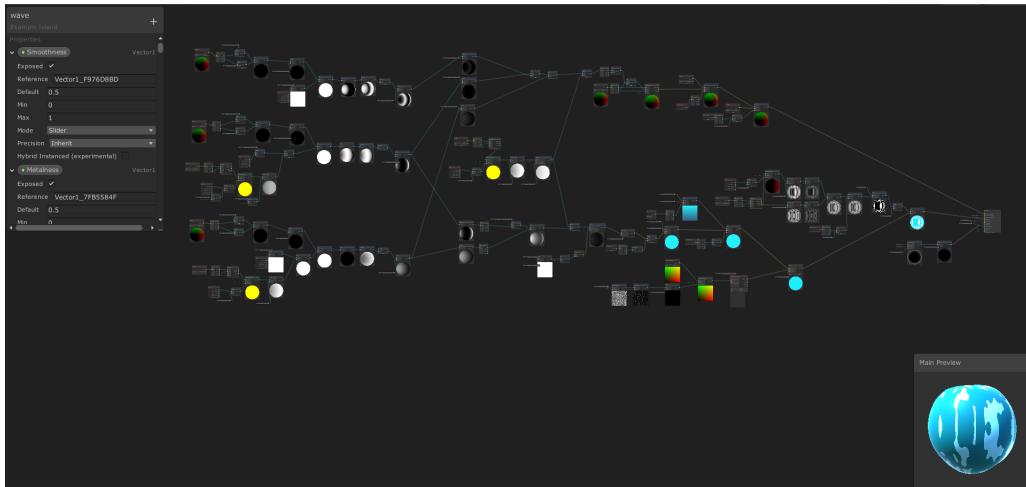


FIGURE 25 – Shader Graph de l'eau

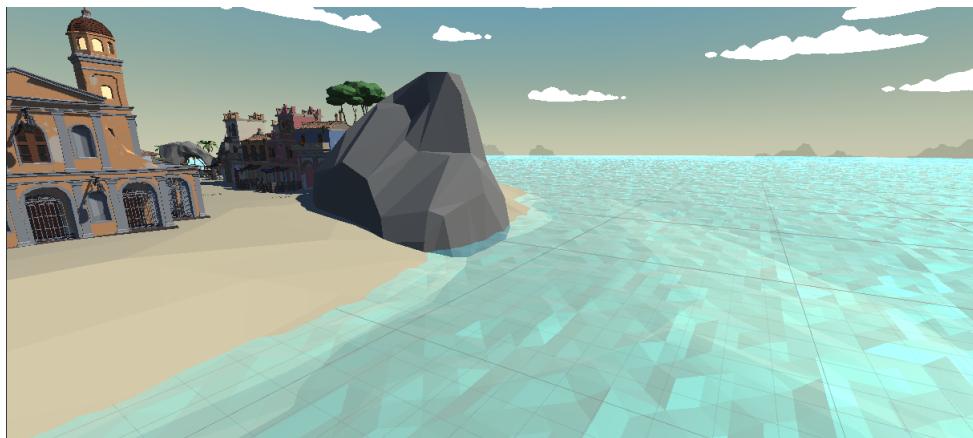


FIGURE 26 – L'eau avant

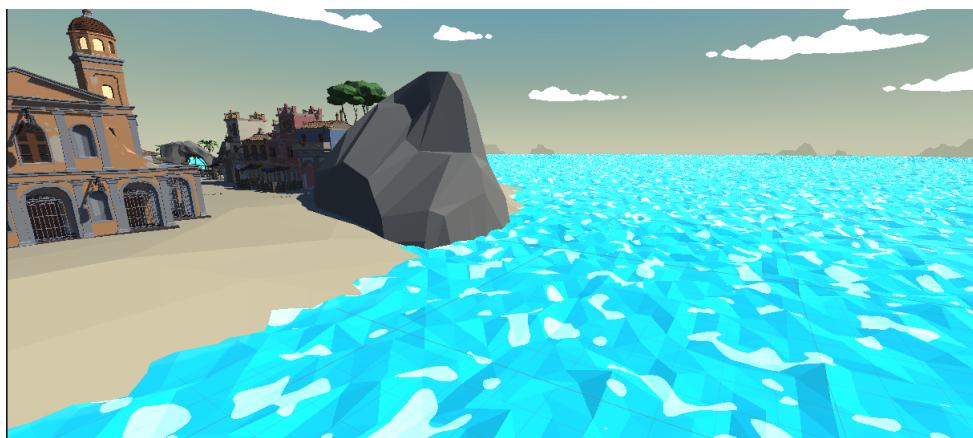


FIGURE 27 – L'eau après

2.10.3 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 effets de pluie ont été rajoutés en plus d'effets de post processing. Mais l'effet le plus important est d'activer toutes les lampes de la carte.



FIGURE 28 – La carte de jour...



FIGURE 29 – ...et de nuit.

2.10.4 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 30 – Application du shader sur un personnage

Nous avons rencontré un problème lorsque nous appliquions le matériau avec le shader, les paramètres de l'instance du shader sur le matériau étant communs. Ainsi, lorsqu'un premier personnage mourrait juste après un autre, le premier mort recommençait l'animation de transition en plus de changer de couleur de vêtements et de peau. La solution trouvée, toute simple, consiste à créer un nouveau matériau et d'appliquer ses paramètres dans le script.

2.10.5 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 agréables. Nous avons créé des volumes qui activent les effets de caméra. Cela nous a permis notamment d'avoir un effet vieux film/sépia lorsque le joueur passe en mode verrouillage.



FIGURE 31 – Mode verrouillage

Si l'on s'attarde sur cette figure, nous voyons que l'effet "sépia" n'est pas visible sur certains objets. C'est là toute la difficulté d'avoir des effets de Post Processing différents sur chaque couche. Nous avons donc ajouté plusieurs caméras **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 obtenu l'effet précédent.

2.11 Avancées et prévisions

2.11.1 Cahier des charges

Répartition des tâches Les tâches étaient à l'origine, réparties comme suit, mais certaines modifications ont dû y être apportées à cause de certains contremorts.

Tâche	Responsable	Suppléant
Carte		
Création de la carte	Paul	Renaud-Dov
Création du lobby	Renaud-Dov	Paul
Réseau		
Implémentation du multijoueur	Julien	Harrys
IA		
Réalisation des différents types d'IA	Renaud-Dov	Paul
Menus		
Interface	Julien	Harrys
HUD	Harrys	Julien
Game Core		
Contrôle du personnage	Renaud-Dov	Harrys
Animations	Renaud-Dov	
Objets dynamiques	Renaud-Dov	Harrys
Déroulé d'une partie	Harrys	Paul
Autre		
Système de progression	Harrys	
Réalisation et maintenance du site Web	Paul	-

Tâche	Soutenance		
	Soutenance 1	Soutenance 2	Soutenance 3
Mouvement	50%	70%	100%
Interfaces/HUD	40%	60%	100%
Cartes	40%	80%	100%
Réseau	20%	50%	100%
IA	30%	60%	100%
Mécaniques de jeu	50%	70%	100%
Progression	20%	60%	100%
Site internet	0%	100%	100%

2.11.2 Première soutenance

2.11.3 Prévisions pour la deuxième soutenance

Map Afin d'améliorer l'expérience des joueurs, nous réfléchissons actuellement à la création d'une seconde map, une fois la première peaufinée.

Interface Sur l'interface, nous prévoyons d'intégrer la catégorie « Liste des serveurs » au menu Multijoueur. Les joueurs pourront ainsi choisir librement le serveur qu'ils souhaitent rejoindre. Ce menu permettrait aussi aux joueurs souhaitant s'amuser en petit comité de créer des parties privées grâce à un onglet "Créer un serveur" à partir de paramètres personnalisables. Il sera également

Partie \ Tâche	Prévu	Réalisé
Mouvement	50%	80%
Interface/HUD	40%	40%
Cartes	40%	40%
Réseau	20%	50%
IA	30%	40%
Mécaniques de jeu	50%	40%
Progression	20%	30%

TABLE 1 – Tableau des avances et retards dans les différentes parties

possible de connaître les noms des joueurs connectés à la partie en pressant la touche TAB grâce à un menu présent en jeu.

Réseau Nous souhaitons ajouter un chat textuel dans le lobby et au sein des parties mêmes afin d'augmenter la sociabilité et l'adversité entre les joueurs.

Intelligence Artificielle Pour la seconde période, notre objectif serait de pouvoir avoir une IA plus autonome et intelligente, qui n'emprunte plus forcément le chemin le plus court. De plus, les joueurs peuvent actuellement bousculer les NPC qui s'arrêtent alors de marcher, mais ils ne peuvent jamais les pousser ; ceci sera donc à paufiner.

Mécaniques de jeu Nous prévoyons d'intégrer pour la deuxième soutenance des pouvoirs comme un écran de fumée ou encore des couteaux, pour rendre le jeu plus intéressant et dynamique. Nous comptons également implémenter le script de choix de la cible. Même si celui-ci a été réalisé il n'est pas encore utilisé en jeu. Il faudra principalement donner un sens à l'attribution des cibles, sous la forme de points octroyés au joueur.

Autres Pour tester le multijoueur avec des invités, nous avons réalisé un début de launcher qui télécharge les dernières mises à jour et les installe. Nous aimerais continuer à le développer pour avoir un lanceur qui se mette à jour automatiquement sans à avoir à le réinstaller.

2.11.4 Deuxième soutenance

Partie \ Tâche	Prévu	Réalisé
Mouvement	70%	70%
Interface/HUD	60%	50%
Cartes	80%	90%
Réseau	50%	70%
IA	60%	80%
Mécaniques de jeu	70%	80%
Progression	60%	50%

TABLE 2 – Tableau des avancées et prévisions

Avancées

2.11.5 Prévisions pour la troisième soutenance

Le jeu est à présent dans un état "jouable". Cependant, il faut encore implémenter de nombreux systèmes pour finaliser notre vision pour le projet.

Multiplayer Certains systèmes ont été créés, comme les finishers ou les pouvoirs, mais n'ont pas encore été implémentés en multijoueur. L'objectif pour la prochaine soutenance sera donc d'intégrer les mécaniques de jeu qui ne l'ont pas encore été au réseau Photon pour permettre aux joueurs d'y accéder. De plus, le travail continue sur la synchronisation i.e. des personnages joueurs et non-joueurs, notamment il est question d'optimisation pour le mouvement des IA qui génère encore une utilisation de bande passante importante. Il serait intéressant de tenter l'utilisation d'une méthode de compression pour les vecteurs afin de diminuer la quantité d'information envoyée.

Progression Bien qu'un système d'expérience et de niveau ait été implémenté, celui-ci n'a pour l'instant pas d'utilité. Mais de nombreuses opportunités de progression se sont ouvertes, notamment grâce au travail de Dov. Par exemple, nous comptons permettre le débloquage des finishers grâce à ce système de progression. Nous pourrons également modifier le système actuel de customisation de personnage et restreindre l'utilisation de certaines options d'apparence à un certain niveau.

Tableau des scores Le scoreboard reste très simple et manque d'informations. Nous souhaiterions donner au tableau des scores un affichage plus épuré et plus simple. Une mise en tableau des différentes informations permettrait d'éviter au texte de dépasser les bordures du canvas. Elle facilitera grandement l'ajout, la suppression ou la modification d'une valeur ou d'un joueur pendant la partie ainsi que l'alignement des valeurs appartenant à la même catégorie. Cette nouvelle version du tableau comprendrait l'affichage des points, du nombre de fois où le joueur est mort, le nombre de fois où il a tué un joueur, le nombre de fois où il a tué une IA et son PING avec le serveur. Ce formatage sous forme de tableau permet aussi de faire une sauvegarde rapide et temporaire du tableau dans les fichiers du jeu. Il serait ainsi beaucoup plus simple pour le logiciel d'y accéder et de l'éditer.



FIGURE 32 – Seconde version du tableau des scores

HUD Certaines informations sont essentielles pour le joueur. Elles lui permettent de savoir en cours de partie et à n'importe quel moment quel pouvoir il peut utiliser, combien de temps il lui reste avant de pouvoir l'utiliser (si son pouvoir nécessite un temps de recharge), la cible qui lui a été désigné par le jeu ou encore de pouvoir lire le chat général en direct.

Carte Pour la prochaine soutenance, l'objectif est d'avoir une map terminée dans les moindres détails. Comme cet objectif semble raisonnable, une seconde carte est aussi une possibilité envisagée pour l'ultime soutenance.



FIGURE 33 – Schéma du futur HUD

2.12 Troisième soutenance

3 Conclusion

En définitive, le projet est sur une bonne lancée en dépit de quelques contre-temps. L'implémentation du multijoueur progresse à grands pas, de la même façon que les éléments client-side (animations, mouvements, shaders...) et que la carte. Le site internet permet maintenant de suivre le projet mois par mois, et le launcher affiche les nouveautés et permet de rester à jour.

Table des figures

1	Frise chronologique du développement du projet	3
2	Aperçu du Launcher	5
3	Problème rencontré lorsque de nombreuses IA vont au même endroit	6
4	Représentation vectorielle des path des IA	7
5	Zone de discussion	7
6	Vue aérienne de la carte	8
7	Colline de la carte, organisée par étages	8
8	Menu principal du jeu Sea of Thieves	9
9	Préversion de notre menu	10
10	Première version du tableau des scores	11
11	Version antérieure du menu principal	12
12	Dernière version du menu	12
13	Le logiciel Bootstrap Studio	13
14	Aperçu de notre site	14
15	Liste des différentes versions du site	15
16	Photo de groupe en multijoueur	16
17	Démonstration du chat Photon	18
18	Exemple des portes que nous avons réalisé	19
19	Joueur grimpant sur l'échelle	19
20	Porte fermée après le passage d'un joueur	20
21	Système de verrouillage	21
22	Vue du shader créé depuis Shader Graph	22
23	Quelques finishers	23
24	Timeline du lancer de hache	23
25	Shader Graph de l'eau	25
26	L'eau avant	25
27	L'eau après	25
28	La carte de jour...	26
29	...et de nuit.	26
30	Application du shader sur un personnage	27
31	Mode verrouillage	27
32	Seconde version du tableau des scores	30
33	Schéma du futur HUD	31

Liste des tableaux

1	Tableau des avances et retards dans les différentes parties	29
2	Tableau des avancées et prévisions	29