RAPPORT DE PROJET

TREASHUNT

de IBOREN™

Paul VOGELEISEN

Cyril KOHLER

Léo SIBOUR

Baptiste MARX

Juin 2024

Table des matières

1 Introduction

Ce rapport vise à présenter en détail les progrès accomplis et le fruit de notre travail tout au long du second semestre sur le développement de notre jeu vidéo Treashunt, un hard-plateformer 3D au style graphique low poly. Ce projet incarne l'essence de l'aventure, du défi et de l'exploration dans un monde en 3D.

Nous débuterons par un bref rappel de ce qu'est notre projet, puis un retour sur l'état de l'art, mettant en lumière les références et les influences qui nous ont inspirées. En outre, nous vous présenterons l'équipe chargée du développement de Treashunt.

Nous exposerons également la répartition des tâches fixée lors du projet. Enfin, nous détaillerons l'avancement des tâches attribuées à chaque membre de l'équipe.

1.1 Notre projet

Treashunt a pour objectif principal de procurer une expérience de jeu captivante, adaptée à un large public. En combinant des univers variés, avec une immersion narrative modeste à travers un mode histoire, nous cherchons à susciter l'intérêt des joueurs de tous horizons.

Le joueur endosse le rôle d'un courageux aventurier en quête d'un trésor légendaire. Sa mission : réussir cette quête principale en revenant sain et sauf avec le trésor tant convoité. Mais son chemin sera périlleux et il devra faire face à des parcours insolites semés d'embûches, mettant ainsi à l'épreuve ses compétences...

Cependant, Treashunt ne se limite pas qu'à l'expérience solo. En effet, à la demande du client, nous avons inclus un mode multijoueur : les joueurs s'affrontent sur les parcours des chapitres du mode histoire et le premier arrivé l'emporte. Ce mode ajoute une dimension compétitive et sociale au jeu, renforçant ainsi son attrait pour un public varié. De plus, l'intégration de l'intelligence artificielle par les ennemis et les pièges dans le mode solo pimente davantage cette aventure épique que représente Treashunt.

Treashunt encourage le développement du sens logique des joueurs. Dans un univers riche de défis, ils devront constamment faire preuve de réflexion pour emprunter le meilleur chemin et surmonter des obstacles pour avancer.

1.2 Etat de l'art

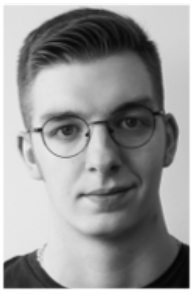
Le jeu que nous développons appartient au genre des jeux d'action-aventure, plus particulièrement à la catégorie des jeux de plateformes dits *hard-platformer*. L'émergence de ce genre remonte aux débuts des jeux vidéo, se concrétisant en 1990 avec le jeu *Alpha Waves*, pionnier en la matière. Ce titre a défini les bases de défis intellectuels et physiques complexes intégrés au gameplay.

Depuis lors, les *hard-platformer* ont évolué pour donner naissance à des titres renommés tels qu'AltF4 (2018), Only Up (2015) et Fall Flat (2016) dont nous nous sommes inspirés.

À l'heure où les jeux d'exploration 3D ont connu une évolution significative tel que la série Uncharted, notre jeu, Treashunt, se distingue par sa fusion d'éléments d'action, d'énigmes et de diversités environnementales. Ses principes fondamentaux incluent une immersion totale du joueur dans l'exploration, une mécanique de plateforme exigeante (le joueur doit escalader et surmonter divers obstacles), des énigmes stimulantes, des combats occasionnels, et enfin, un style graphique *low poly*, qui confère au jeu un look distinctif tout en optimisant ses performances.

1.3 Notre équipe

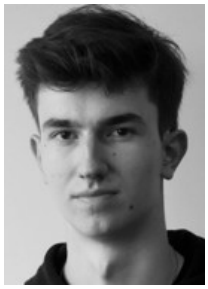
Suite au départ de Benjamin Mathieu au début du S2, nous sommes désormais une équipe de quatre membres engagés dans le développement de Treashunt. Face à cette nouvelle organisation, nous avons dû réajuster nos stratégies et répartir les tâches en conséquence. Benjamin étant chargé du Sound Design, c'est Paul qui a repris cette tâche.



Baptiste Marx
Directeur Artistique

Léo Sibour
Game/Web Designer

Cyril Kohler
Directeur Technique



Paul Vogeleisen
Chef de Projet

FIGURE 1 – Membres d'IBOREN

1.4 Répartition des tâches

Il se trouve qu'il y a eu quelques échanges de tâches qui se sont fait naturellement au cours du projet permettant d'avancer encore plus rapidement. Cyril s'est chargé des déplacements du joueur et Léo s'est chargé des interactions avec l'environnement et a souvent poussé encore plus loin le design des maps dans le détail.

	Paul	**Cyril**	**Léo**	**Baptiste**
Programmation				
Multijoueur	S	R		
Mécanique du jeu				
Déplacement du joueur	S	R		
Gestion inventaire Interface			R	S
Interactions environnement	S		R	
Intelligence Artificielle				
Pièges et défenses	S	R		
Apparition des ennemis	R	S		
Game Design				
Conception des maps				
Lobby : plage			S	R
Chapitre 1 : fôret			R	S
Chapitre 2 : montagnes			S	R
Chapitre 3 : volcan			S	R
Environnement				
Scénario et narration	S			R
Musiques et sons	R		S	
Gestion des assets			S	R
Site Web				
Page d'accueil			R	S
Section téléchargement			R	S
Section à propos			R	S

TABLE 1 – Tableau récapitulatif : répartition des tâches

R : Responsable | **S** : Suppléant

2 Problèmes rencontrés

Avant de rentrer dans le vif du sujet, laissez nous vous récapituler quelques problèmes majeurs que nous avons rencontré lors du développement de Treashunt :

Tâche	Gravité	Problème	Résolu par
Multijoueur	CRITIQUE	Multijoueur HS : le client se spawn pas et se met en mode spectateur de l'hôte	Cyril et Paul
IA : ennemis	SERIEUX	l'ennemi est bien présent sur la map mais ne chasse pas et n'attaque pas le joueur à proximité	Paul
Inventaire joueur	MODERE	Les sprites ne s'ajoutent pas à l'interface graphique	Léo
Animation joueur	MODERE	Empêcher le joueur de se déplacer quand il meurt	Léo

TABLE 2 – Tableau récapitulatif : problèmes et bugs du projet

Nous avons évidemment rencontré de nombreux autres bugs et passé quelques nuits blanches dessus (peu de temps avant le rendu notamment...) mais nous préférons vous citer les plus problématiques uniquement. Chaque membre expliquera en détails dans la partie qui suit comment il a résolu son problème.

3 Avancement du projet

Dans cette section, vous trouverez toutes les informations détaillées concernant la réalisation et l'avancement des tâches de chacun au cours du développement de Treashunt.

3.1 Baptiste - Directeur Artistique

3.1.1 Design menu principal

Le menu principal s'ouvre sur une scène qui plonge immédiatement les joueurs dans l'univers captivant de la piraterie. Les options de menu, telles que SOLO, MULTIPLAYER et QUIT, sont présentées avec des boutons en forme de planches de bois, ornés d'une police blanche stylisée qui évoque le thème pirate du jeu. Chaque détail renforce cette atmosphère, depuis le choix des matériaux jusqu'à la typographie.

En arrière-plan, une île tropicale s'étend avec une plage de sable jaune baignée par une mer turquoise. Des palmiers verts se balancent doucement au vent, entourés d'une végétation luxuriante qui crée un cadre idyllique mais mystérieux. Amarré près de l'île se dresse un imposant navire pirate à voiles marron, évoquant une silhouette menaçante et robuste. À l'horizon, un autre navire pirate navigue fièrement sur les vagues.

FIGURE 2 – Menu Principal

Des éléments disséminés ajoutent au réalisme de la scène : un petit feu de camp cerné de pierres, émettant une lueur chaleureuse à la tombée de la nuit, et un drapeau pirate noir orné d'une tête de mort qui flotte au sommet d'une colline. Deux personnages, assis non loin sur la plage, contemplent l'horizon avec une expression mêlée de détermination et de curiosité, captivés par l'appel de l'aventure.

En bas à gauche de l'écran, une discrète icône en forme d'engrenage sur une planche de bois attire l'attention et symbolise l'accès aux paramètres du jeu. Ce détail fonctionnel s'intègre harmonieusement dans le décor sans rompre l'immersion dans l'univers visuel riche et détaillé.

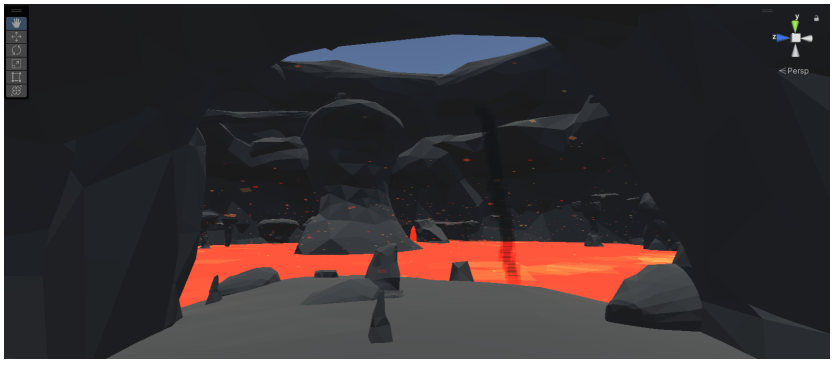
Le style graphique Low Poly est utilisé avec brio pour donner une clarté visuelle tout en optimisant les performances du jeu. Chaque aspect de cette composition artistique contribue à transporter les joueurs dans une aventure palpitante de chasse au trésor en haute mer, capturant l'essence même de l'excitation et de la romance de la vie de pirate.

3.1.2 Level design : chapitre 3

Ce troisième chapitre, situé à l'intérieur d'un volcan, est conçu pour offrir une expérience de jeu intense et immersive, qui contraste fortement avec les environnements précédents. Après avoir parcouru les sentiers sinueux et les sommets escarpés des montagnes, le joueur pénètre dans les entrailles de la terre par un tunnel sombre et étroit. Cette transition entre les sommets aériens et les profondeurs volcaniques crée une montée en tension dramatique, renforçant le sentiment de progression et de danger.

Le couloir initial du volcan sert de guide naturel, forçant le joueur à avancer sans possibilité de retour, accentuant ainsi l'impression d'un voyage sans retour possible. Ce passage étroit s'ouvre sur un vaste cratère, révélant une mer de lave en fusion qui bouillonne et éclaire sinistrement les parois rocheuses. La seule ouverture visible est le sommet du cratère, offrant un mince aperçu du ciel et symbolisant la liberté et la réussite qui attendent le joueur une fois ce défi surmonté. Cette vision fugace du monde extérieur renforce l'ambiance oppressante et angoissante de ce chapitre.

(a) Croquis du volcan



(b) Rendu du volcan

FIGURE 3 – Level design du chapitre 3

L'étendue de lave en fusion représente un obstacle majeur que le joueur doit surmonter. De plus, les particules incandescentes ajoutent une dimension épique à ce dernier chapitre. Pour atteindre la sortie située en hauteur, le joueur doit naviguer avec précision le long des parois du volcan, sautant de rebord en rebord. La conception des plateformes évoquant un gigantesque escalier en colimaçon exige une habileté et une précision accrues, ajoutant un niveau de défi supplémentaire. À chaque saut, le joueur est rappelé du sort tragique des aventuriers précédents par la présence de squelettes éparpillés le long du chemin, témoignant des dangers mortels qui l'attendent.

En somme, le choix d'un volcan pour le troisième chapitre n'est pas seulement une décision esthétique mais une stratégie de conception visant à maximiser l'engagement et l'immersion du joueur. En le confrontant à des défis croissants et en créant une atmosphère de danger palpable, ce chapitre pousse le joueur à repousser ses limites et à maîtriser pleinement les compétences acquises dans les niveaux précédents. La récompense de la réussite, symbolisée par l'atteinte du sommet et la vision du ciel, offre une gratification puissante, incitant le joueur à persévérer malgré les obstacles.

3.1.3 Scénario / narration

Dans ce jeu d'aventure, le joueur incarne un explorateur intrépide se lançant dans une quête à travers trois environnements distincts : la forêt, la chaîne montagneuse et le volcan. Chaque chapitre représente une étape cruciale de son périple, ponctuée de défis uniques et de découvertes captivantes.

Dans le premier chapitre, notre héros pénètre dans une épaisse forêt, où il doit naviguer à travers des sentiers sinueux tout en évitant divers pièges naturels. Ce cadre initial met l'accent sur l'exploration et la survie, préparant le joueur aux difficultés à venir. Pendant son parcours, le joueur rencontrera des PNJ (Personnages Non Joueurs) qui lui fourniront des informations essentielles. Ces dialogues immersifs et concis permettent de maintenir le rythme du jeu tout en offrant des indices précieux.

Le deuxième chapitre conduit le joueur à travers une chaîne montagneuse escarpée, où il doit affronter des crevasses et des chemins abrupts. Fort de l'expérience acquise dans la forêt, le joueur utilise ses compétences pour surmonter les obstacles montagneux, rendant ce passage non seulement plus difficile mais aussi plus gratifiant. Là encore, des PNJ seront présents pour guider le joueur. Ces conseils, bien que brefs, enrichissent l'expérience en ajoutant une dimension narrative tout en offrant une aide précieuse.

Enfin, le héros atteint le volcan, un territoire hostile où la lave bouillonnante et les geysers explosifs représentent les ultimes défis. Grâce à ses compétences affinées au cours des chapitres précédents, il progresse à travers ce paysage infernal. Au cœur de ce dernier environnement, il découvre un trésor inestimable, la récompense tant attendue de son périple ardu. Cette découverte marque l'apogée de son aventure, offrant une conclusion satisfaisante et enrichissante à son voyage.

3.2 Léo - Game/Web Designer

3.2.1 Menu principal

Le Menu Principal du jeu est conçu pour immerger immédiatement le joueur dans l'univers de Treashunt. À l'arrière-plan, une vidéo cinématique de la plage du jeu tourne en boucle, créant une ambiance engageante et cohérente avec le thème de l'aventure. Le Menu Principal comporte plusieurs options :

1. Solo : Lance une partie en mode solo, permettant au joueur de s'immerger dans l'histoire et les défis de Treashunt de manière individuelle.

2. Multiplayer : Accède au mode multijoueur, où les joueurs peuvent se connecter et jouer ensemble, ajoutant une dimension sociale et compétitive au jeu.

3. Quit : Permet de quitter le jeu de manière sécurisée, en sauvegardant la progression du joueur si nécessaire.

4. Paramètres : Ce bouton ouvre un sous-menu offrant des options supplémentaires : Informations sur l'équipe : Présente des détails sur les développeurs et les créateurs du jeu. Contrôle du son : Permet de désactiver ou d'ajuster le son du menu principal, offrant une personnalisation audio selon les préférences du joueur. L'interface du Menu Principal est intuitive et stylisée pour s'aligner avec l'esthétique générale du jeu, utilisant des éléments graphiques et des polices de texte cohérentes avec le thème de Treashunt. L'animation de la plage en arrière-plan ajoute une touche dynamique et immersive, invitant les joueurs à plonger dans l'aventure dès les premiers instants.

3.2.2 Level design : chapitre 1 - chapitre 2

Chapitre 1 : La Forêt
Lors de la conception de la carte du chapitre 1, intitulé "La Forêt", nous avons d'abord conceptualisé la forme générale de la carte sans inclure les objets principaux. Cela nous a permis d'avoir une vision claire du parcours à ajouter et de définir la structure globale du niveau.

Pour optimiser les performances, nous avons utilisé des objets 3D dans le style Low Poly. Ce choix stylistique non seulement améliore les performances du jeu en réduisant la charge graphique, mais il s'harmonise également avec l'esthétique visuelle de Treashunt, offrant un rendu visuel épuré et stylisé.

Nous avons récupéré une grande partie des assets sur Unity Store, ce qui nous a permis de gagner du temps et de nous concentrer davantage sur la conception et l'intégration des éléments dans le jeu. Les assets Low Poly disponibles sur Unity Store ont été soigneusement sélectionnés pour s'intégrer parfaitement dans l'univers de Treashunt, tout en respectant les contraintes de performance.

Le chapitre 1 comprend également deux mini-jeux captivants :

1. Naviguer un Bateau : dans ce mini-jeu, les joueurs ont la possibilité de naviguer un bateau pour traverser un lac hostile. Ce segment ajoute une dimension supplémentaire à l'aventure, demandant aux joueurs de naviguer prudemment tout en évitant divers obstacles et dangers présents sur le lac. Ce mini-jeu non seulement diversifie le gameplay, mais il offre aussi une expérience unique et palpitante au sein du chapitre.

2. Mini-Jeu du Village : après avoir traversé le lac des pirates, le joueur pénètre dans un nouveau décor : un village. Dans ce mini-jeu, nous avons ajouté des PNJ appelés « traders » qui jouent un rôle essentiel dans cette partie puzzle du jeu. Le but est d'échanger des objets d'un trader à l'autre, en fonction des indications flottant au-dessus de la tête de chacun. Le joueur doit également récupérer une émeraude posée sur un bateau pirate en effectuant un parcours sur des rochers. Avec les deux derniers objets obtenus, un échangé en porte-à-porte et l'autre récupéré lors du parcours, le joueur échange ces items avec le dernier trader, le « SpecialTrader ». Celui-ci remettra une clé permettant au joueur de sortir du village et de continuer son aventure.

(a) Bateau du joueur

(b) Zone du village

FIGURE 4 – mini-jeux chapitre 1

Aspect Technique du Mini-Jeu du Village

Pour rendre ce mini-jeu interactif, les traders possèdent un AnimatorController qui leur permet de s'animer lorsque le joueur s'approche. Nous avons recherché et téléchargé des animations sur le site Mixamo.com, propriété de l'entreprise Adobe, que nous avons ensuite intégrées dans Unity3D. Les traders ont deux animations principales : un signe de la main pour représenter un échange réussi et un refus de la tête si le joueur n'a pas l'objet demandé. Le joueur est également guidé par des bulles de bande dessinée contenant l'image de l'objet demandé, facilitant ainsi la vision lors de l'échange.

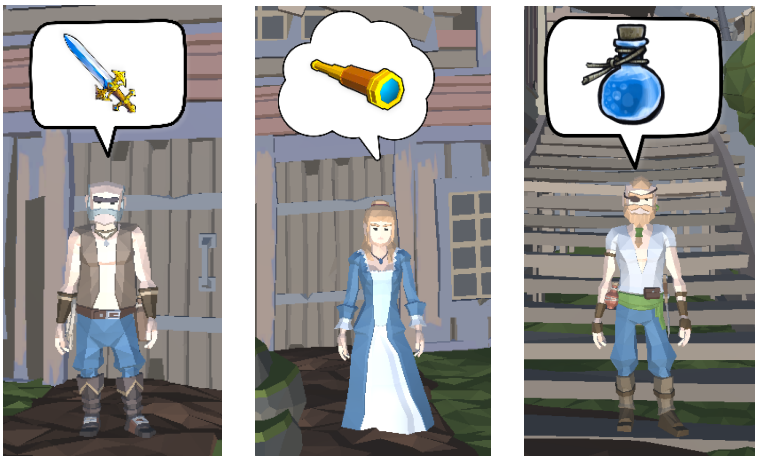
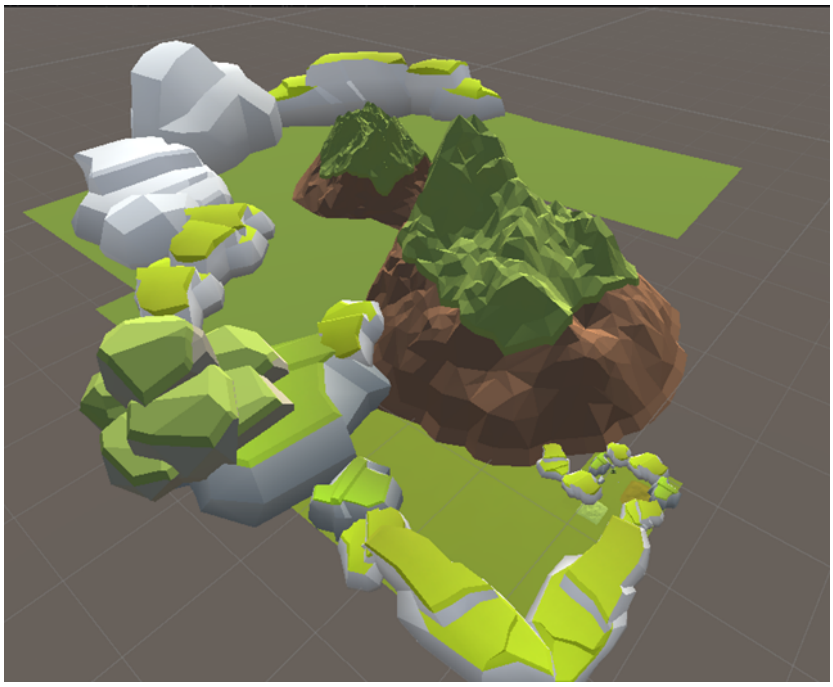


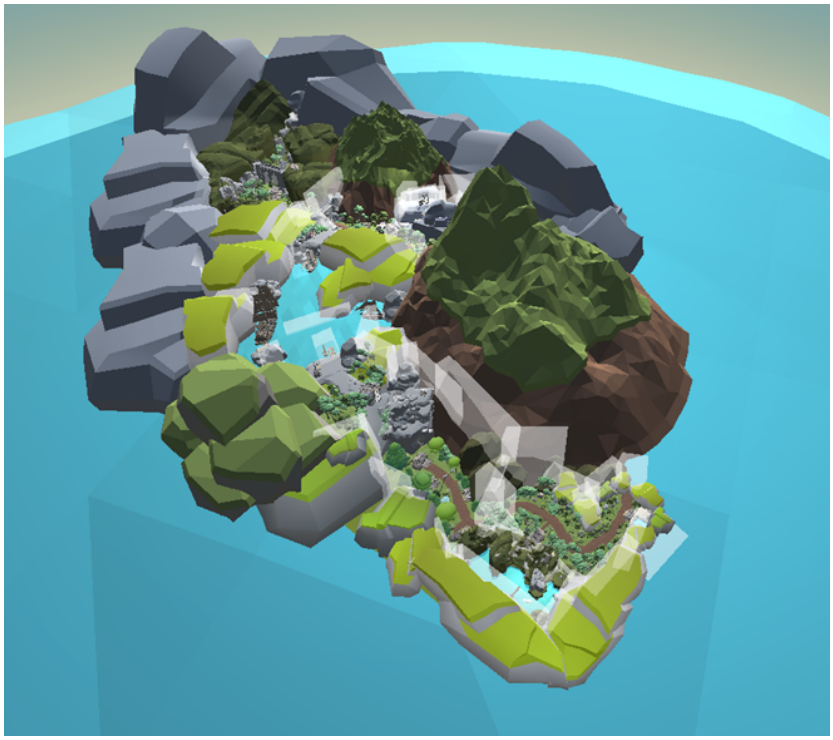
FIGURE 5 – Exemple de traders du village

La conception du village consiste en un chemin relativement droit bordé de maisons et de végétation. Nous avons ajusté manuellement les hitboxes des objets 3D, tels que les arbres, pour éviter toute collision qui pourrait empêcher le joueur de poursuivre son aventure. L'algorithme d'échange des traders ne consiste pas seulement à faire disparaître l'objet demandé de l'inventaire du joueur et à ajouter en conséquence l'objet de récompense. Lors d'un échange, si le joueur est dans une zone autour du trader et possède l'item demandé, son objet disparaît de l'inventaire virtuel ainsi que de l'interface et de son affichage physique (l'objet apparaissant dans le dos du joueur lorsqu'il est sélectionné). Ensuite, l'item donné par le trader est ajouté, impactant ces mêmes champs (virtuel, physique, affichage).

Grâce à cette approche, nous avons pu créer un environnement immersif et cohérent pour le joueur, tout en assurant une fluidité optimale du gameplay. Le parcours dans la forêt est ainsi pensé pour offrir à la fois des défis et des opportunités d'exploration, intégrant des éléments interactifs et des objets clés pour progresser dans l'aventure. En résumé, la conception du chapitre 1 de Treashunt reflète un équilibre entre esthétique et performance, utilisant des techniques de design Low Poly et des ressources de Unity Store pour créer un niveau captivant et fluide pour les joueurs.



(a) 3 mars 2024



(b) 15 juin 2024

FIGURE 6 – Avancement map chapitre 1

Chapitre 2 : Les Montagnes

Le chapitre 2, intitulé "Les Montagnes", est davantage axé sur le parcours et l'esquive d'ennemis, offrant une expérience de jeu dynamique et exigeante. La mission principale du joueur est de terminer le parcours, que ce soit en mode solo ou en multijoueur. Ce chapitre se distingue par un environnement montagneux avec des piliers rocheux sur lesquels le joueur doit sauter tout en évitant de tomber.

Conception et Développement des Décors

Comme son nom l'indique, le décor des Montagnes est principalement constitué de piliers rocheux, de falaises escarpées et de ponts naturels. Ces éléments créent un parcours complexe et stimulant, demandant aux joueurs d'exercer précision et habileté dans leurs sauts et leurs mouvements.

Le processus de création des décors a commencé par des croquis réalisés à la main, qui ont servi de base pour la conception. Ces croquis détaillés nous ont permis de visualiser les différentes sections du parcours et de planifier les défis que le joueur rencontrerait. Ensuite, nous avons utilisé des assets disponibles sur Unity Store pour recréer fidèlement ces décors en 3D dans Unity3D. Ces assets, principalement dans le style Low Poly, ont été choisis pour leur compatibilité avec le design général de Treashunt, assurant une cohérence visuelle et des performances optimales.

Parcours et Mécaniques de Jeu

Le parcours dans les Montagnes est conçu pour offrir une série de défis progressifs. Le joueur doit sauter de pilier en pilier, tout en évitant de tomber dans les ravins en contrebas. Certains piliers peuvent être instables, ajoutant une dimension supplémentaire de difficulté. De plus, des ennemis patrouillent certaines sections du parcours, obligeant le joueur à esquiver leurs attaques ou à les éliminer pour progresser.

En mode multijoueur, le parcours devient une compétition où les joueurs doivent non seulement surmonter les obstacles, mais aussi rivaliser entre eux pour atteindre la fin en premier. Ce mode ajoute une tension supplémentaire, rendant chaque saut et chaque esquive cruciaux pour la victoire.

Intégration des Ennemis et des Objets

Les ennemis dans ce chapitre sont intégrés de manière à augmenter la difficulté sans rendre le jeu frustrant. Ils possèdent des patterns de mouvement et d'attaque prédéfinis, permettant aux joueurs d'apprendre et de maîtriser leurs comportements pour mieux les éviter. En outre, des objets spéciaux peuvent être trouvés tout au long du parcours, offrant des bonus temporaires ou des améliorations pour aider le joueur à surmonter les défis plus facilement.

Aspect Visuel et Immersion

Le design visuel des Montagnes est conçu pour être immersif et spectaculaire. Les textures rocheuses, les jeux de lumière et les effets de particules contribuent à créer une atmosphère réaliste et engageante. Les assets Low Poly, tout en étant performants, sont utilisés de manière à renforcer l'aspect artistique du jeu, offrant une esthétique à la fois simplifiée et attrayante.

En conclusion, le chapitre 2 de Treashunt, "Les Montagnes", est une combinaison harmonieuse de conception artistique et de mécaniques de jeu. Grâce à un travail minutieux de croquis préliminaires et à l'utilisation d'assets Unity Store, nous avons pu créer un niveau captivant et défiant, fidèle à l'esprit du jeu et engageant pour les joueurs en mode solo comme en multijoueur.

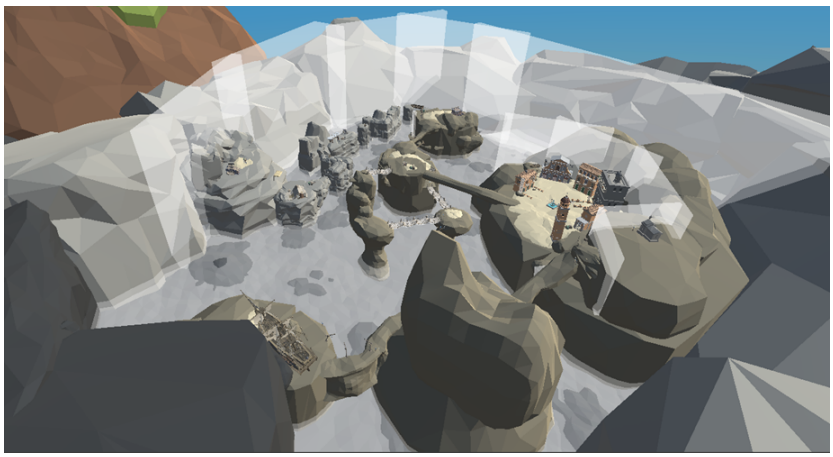


FIGURE 7 – map chapitre 2 avec mur invisible

3.2.3 Système PNJ

Les PNJ (personnages non-joueurs) jouent un rôle crucial dans l'histoire de Treashunt, fournissant des indices essentiels et indiquant les objets à trouver pour certains mini-jeux, comme celui du village dans le chapitre 1. PNJ de Narration Un PNJ de narration

dans Treashunt est inspiré des personnages de bandes dessinées, avec une narration effectuée via des bulles de texte au-dessus de sa tête. Le joueur peut faire défiler ces bulles une par une en appuyant sur la touche 'E' lorsqu'il est à proximité du PNJ. Les bulles de narration sont équipées d'un script nommé "IndicationRotation.cs", qui permet aux bulles de toujours faire face à la caméra du joueur, peu importe où il regarde. Interaction et Animation Les PNJ sont également équipés d'un Animator et de sons

designés pour enrichir l'interaction. Lorsque le joueur interagit avec un PNJ, ce dernier exécute une animation "IsSpeaking", explicite d'un geste de bras montrant qu'il parle, et un son "BlaBla" se déclenche simultanément pour simuler la conversation.

Ces animations et effets sonores contribuent à rendre les PNJ plus vivants et immersifs, renforçant leur rôle dans la progression et l'immersion du joueur dans l'histoire.

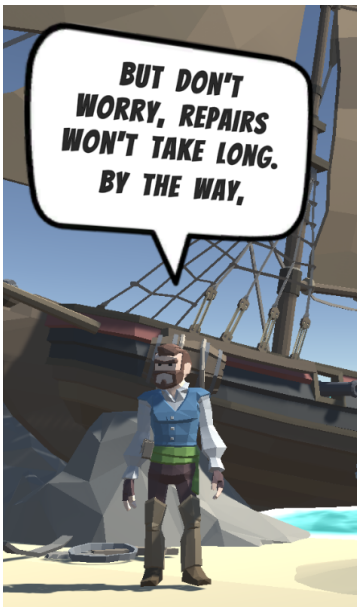


FIGURE 8 – PNJ plage

Indications et Mini-Jeux

Les PNJ sont essentiels pour indiquer les objets nécessaires dans divers mini-jeux. Par exemple, dans le mini-jeu du village du chapitre 1, les PNJ traders fournissent des indices sur les objets à échanger. Grâce à leurs bulles de narration et leurs indications, les joueurs peuvent suivre les instructions et réussir les puzzles proposés.

Design et Immersion

Les PNJ sont soigneusement conçus pour s'intégrer harmonieusement dans l'univers visuel de Treashunt. Le style graphique des personnages, leurs animations et leurs interactions sont pensés pour renforcer l'immersion du joueur dans l'aventure, tout en apportant des éléments narratifs et fonctionnels essentiels à la progression du jeu.

En résumé, les PNJ dans Treashunt sont plus que de simples personnages : ils sont des guides, des narrateurs et des éléments interactifs clés qui enrichissent l'expérience de jeu et aident les joueurs à naviguer dans les défis et les aventures qui les attendent.

3.2.4 Inventaire et interfaces

Cet ajout joue un rôle crucial dans Treashunt, car il est indispensable pour compléter certaines quêtes et progresser dans l'aventure. Il se compose de deux éléments distincts : une interface graphique et une représentation physique des objets 3D sélectionnés par le joueur.

Nous avons conçu l'inventaire de manière à pouvoir contenir plusieurs exemplaires d'un même objet. Bien que l'inventaire dispose de seulement deux emplacements, cela suffit pour l'avancée du joueur dans l'aventure. Cette fonctionnalité est étroitement liée à l'acquisition et à la suppression d'objets, que ce soit par une interaction directe avec l'environnement ou en récupérant des objets au sol. Virtuellement, l'inventaire est représenté par un tableau de GameObject dans un script (Inventory.cs) associé au joueur.

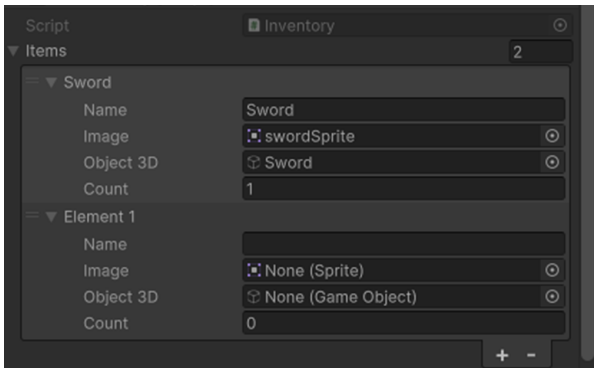


FIGURE 9 – Inspecteur inventaire

Ce tableau peut être manipulé à l'aide de différentes méthodes telles que AddItem(), RemoveItem() et IsInside(). La méthode IsInside() est principalement utilisée dans le mini-jeu du village avec les marchands, pour déterminer la possibilité ou non d'un échange. Par ailleurs l'ajout d'item est prévu pour être le plus à gauche possible selon le contenu.Lors de la sélection d'un objet, le joueur utilise les touches de clavier "1" ou "2" pour choisir l'emplacement dans l'inventaire. Cette action modifie la couleur du cadre correspondant dans l'interface graphique de l'inventaire.

FIGURE 10 – UI inventaire

Les objets jouent un rôle essentiel dans le jeu, car ils permettent la réalisation du mini-jeu du village et offrent la capacité de conduire un bateau en mode solo. Virtuellement, un objet est créé dans le fichier « Item.cs » et possède plusieurs propriétés : Name, Sprite, Object3D, et Count. Ces propriétés signifient respectivement le nom de l'objet, son image associée pour s'ajouter à l'interface de l'inventaire, un objet 3D qui se positionne dans le dos du joueur lorsqu'il est sélectionné, et « Count », le nombre d'unités de ce même objet que le joueur possède dans son inventaire.

Les objets sont intégrés de manière à enrichir l'expérience de jeu. Par exemple, certains objets sont nécessaires pour interagir avec les personnages et les défis du mini-jeu du village, ajoutant une dimension stratégique et interactive au gameplay. D'autres objets sont essentiels pour naviguer et conduire un bateau, ouvrant de nouvelles possibilités d'exploration et d'aventure en mode solo.

Cette fonctionnalité est gérée par les scripts HUD.cs et Border.cs, qui sont attachés à des GameObjects présents dans le Canvas de la scène, assurant ainsi une cohérence visuelle dans la sélection des emplacements de l'inventaire. De plus, la représentation physique des objets sur le dos du joueur est conditionnée par la présence de l'objet dans l'inventaire et sa sélection. Si tel est le cas, le joueur verra à la fois l'objet représenté sous forme de sprite dans sa barre d'inventaire et physiquement sur le dos de son personnage. Sur le personnage, tous les objets existants sur la map sont présents sur son dos.

Création d'un Objet
Un objet est représenté en 3D sur la carte et est détruit une fois récupéré par le joueur. Il est équipé d'un collider avec l'option IsTrigger activée, permettant ainsi au joueur de le récupérer par collision. De plus, il est composé d'un script Item.cs, entre autres. Ce script permet d'initialiser l'objet avec un nom, un sprite (une image 2D), une quantité et une limite de quantité. Ces informations facilitent la communication avec d'autres fonctionnalités du jeu.

Problèmes Rencontrés et Solutions Apportées : Durant le développement de l'inventaire, nous avons rencontré des difficultés liées à l'ajout d'images dans les emplacements de l'interface graphique. En effet, les images PNG n'avaient pas toutes les mêmes dimensions que l'image transparente de base. L'emplacement vide est représenté par une image totalement transparente, et lorsqu'un objet est ajouté à cet emplacement, cette image est remplacée par celle de l'objet associé. Cette différence de dimensions entraînait un dépassement du cadre de l'emplacement de l'interface. Pour résoudre ce problème, nous avons dû redimensionner chaque image d'objet afin qu'elles s'intègrent correctement dans l'interface graphique de l'inventaire.

3.2.5 Les Items

Les objets jouent un rôle essentiel dans le jeu, car ils permettent la réalisation du mini-jeu du village et offrent la capacité de conduire un bateau en mode solo.

Virtuellement, un objet est créé dans le fichier « Item.cs » et possède plusieurs propriétés : Name, Sprite, Object3D, et Count. Ces propriétés signifient respectivement le nom de l'objet, son image associée pour s'ajouter à l'interface de l'inventaire, un objet 3D qui se positionne dans le dos du joueur lorsqu'il est sélectionné, et « Count », le nombre d'unités de ce même objet que le joueur possède dans son inventaire.

Les objets sont intégrés de manière à enrichir l'expérience de jeu. Par exemple, certains objets sont nécessaires pour interagir avec les personnages et les défis du mini-jeu du village, ajoutant une dimension stratégique et interactive au gameplay. D'autres objets sont essentiels pour naviguer et conduire un bateau, ouvrant de nouvelles possibilités d'exploration et d'aventure en mode solo.

```
1  using UnityEngine;
2  using Sprite = UnityEngine.ProBuilder.Shapes.Sprite;
3
4  [System.Serializable]
5
6  public class Item
7  {
8      public string name;
9      public UnityEngine.Sprite image;
10     public GameObject Object3D;
11     public int count;
12     public Item(string itemName,UnityEngine.Sprite thisImage,GameObject object3D,int itemCount)
13     {
14         name = itemName;
15         image = thisImage;
16         count = itemCount;
17         Object3D = object3D;
18     }
19 }
```

FIGURE 11 – Script Item

Chaque objet est soigneusement conçu pour s'intégrer parfaitement dans l'univers de Treashunt. Le Sprite, ou image 2D, permet de visualiser l'objet dans l'inventaire, tandis que le modèle 3D apporte une dimension tangible lorsqu'il est équipé par le joueur. Le système de comptage permet de suivre facilement la quantité d'objets collectés, assurant ainsi une gestion efficace de l'inventaire. La création et la gestion des objets dans « Item.cs » permettent une flexibilité et une extensibilité du système, facilitant l'ajout de nouveaux objets et fonctionnalités au fur et à mesure que le jeu évolue. En somme, les objets sont un élément clé de l'expérience de jeu, offrant à la fois des défis et des opportunités pour les joueurs de découvrir et d'explorer l'univers riche de Treashunt. Les Objets sont ainsi présents physiquement dans les map et peuvent être récupérés grâce à un script Pickup.cs, qui s'occupe donc d'ajouter l'item à l'inventaire du joueur si ce dernier est entré en collision avec.

3.2.6 La barre de vie

La barre de vie a été conçue pour être simple et en harmonie avec le design de Treashunt. Nous avons décidé de la modifier dès que le joueur subit des dégâts infligés par un ennemi, par exemple lorsqu'il reçoit un coup d'épée. Une mauvaise chute, considérée comme une mort instantanée, ne fera cependant pas perdre de points de vie au joueur.

Initialement, le joueur dispose de 100 points de vie. Chaque coup reçu de la part d'un ennemi lui fait perdre 25 points, ce qui signifie qu'il peut résister à quatre coups avant de mourir et de réapparaître au dernier point de contrôle.

En ce qui concerne son design, nous avons opté pour un aspect en bois reflétant la coque d'un bateau pirate, tout en conservant une simplicité qui gêne le moins possible le champ de vision du joueur.

FIGURE 12 – Barre de vie à 100%

FIGURE 13 – Barre de vie à 75%

Lorsque le joueur perd un point de vie, une animation de transition se déclenche, montrant l'état de la barre de vie avant et après la perte. Un léger agrandissement puis rétrécissement de l'échelle se produit, accompagné d'une teinte rouge sur l'image, indiquant la prise de dégâts.

D'un point de vue technique, cette fonctionnalité est composée de 5 images dont une représentant une barre de vie vide et une organisation d'activation ou nom de la bonne image selon la quantité de points de vie du joueur.

3.2.7 Le Minuteur

Le minuteur est une fonctionnalité essentielle pour informer le joueur du temps qu'il a passé dans le jeu. Il permet de suivre le temps écoulé jusqu'à ce que le joueur ouvre le menu « Jeu en Pause ». En mode multijoueur, le minuteur est continuellement visible à l'écran, car il s'agit d'une course contre les autres joueurs, et le suivi du temps est crucial pour la compétition.

En revanche, en mode solo, le minuteur est affiché uniquement lorsque le joueur accède au menu « Jeu en Pause », afin de ne pas encombrer l'écran pendant le gameplay.

Le minuteur est intégré de manière à être discret mais visible, utilisant une police de texte claire et une position stratégique sur l'écran pour minimiser les distractions. En mode multijoueur, il est placé dans un coin de l'écran, facilement visible sans gêner l'action principale. En mode solo, il apparaît de manière subtile dans le menu de pause, permettant au joueur de vérifier son temps sans perturber l'immersion dans le jeu.

Cette fonctionnalité est particulièrement utile pour les joueurs qui souhaitent suivre leur progression et améliorer leurs performances, que ce soit en solo ou en multijoueur. De plus, le minuteur peut également servir de référence pour les défis chronométrés ou les compétitions organisées au sein de la communauté de joueurs.

En offrant une visualisation claire et constante du temps de jeu, le minuteur contribue à une expérience de jeu plus engageante et compétitive.

FIGURE 14 – Timer

3.2.8 Site Web

Design du Site Web de Treashunt

Le design de notre site web a été minutieusement pensé pour offrir une navigation intuitive et agréable, permettant aux utilisateurs de découvrir facilement des informations sur le jeu Treashunt et notre entreprise, Iboren. Voici une description détaillée de nos choix de design, couleurs et effets visuels utilisés.

Choix des Couleurs et Thème

Le choix des couleurs pour le site web repose sur une palette neutre, complétée par des touches de couleurs vives pour ajouter de l'originalité. Les couleurs neutres, comme les tons de gris et de beige, servent de toile de fond apaisante, mettant en valeur les éléments visuels sans distraire l'utilisateur. Les accents de vert forêt et de bleu océan rappellent le thème naturel et aventureux de Treashunt.

Ces choix de couleurs visent à refléter l'ambiance du jeu et les valeurs de notre entreprise, Iboren, qui met en avant l'innovation et la créativité tout en respectant l'environnement.

Effets Visuels

L'effet parallax est un élément clé de notre design. Ce choix n'est pas seulement esthétique, il sert à immerger les visiteurs dans l'univers de Treashunt dès leur arrivée sur le site. Le parallax crée une impression de profondeur et de mouvement, renforçant l'expérience utilisateur en ligne avec le thème de la nature et de l'aventure du jeu.

Accueil : Introduit par un effet parallax avec des éléments naturels en mouvement tels que des arbres et des montagnes, plongeant le joueur dans l'univers de Treashunt dès les premiers instants.

Téléchargement : Utilise également un effet parallax, mettant en scène des éléments du jeu pour maintenir une cohérence visuelle tout en ajoutant une dimension dynamique.

À propos : Présente un parallax représentant une plaine et des nuages, ajoutant une touche de tranquillité et d'élégance. En plus de l'effet parallax, nous avons ajouté des ombres derrière certains arrière-plans pour donner de la profondeur au site. Ces ombres créent un effet de relief, rendant les éléments visuels plus distincts et ajoutant une dimension supplémentaire à l'expérience utilisateur.

Conception de la Navigation (Nav Bar)

La barre de navigation (navbar) est conçue pour être dynamique et réactive, facilitant l'accès à toutes les sections du site. Elle permet aux visiteurs de se rendre directement à l'information qui les intéresse, qu'il s'agisse de la chronologie du projet ou des origines de création d'Iboren.

Cette navbar est fixée en haut de la page, permettant aux utilisateurs de naviguer facilement entre les sections. Les éléments de la liste sont centrés et espacés uniformément, garantissant une apparence propre et organisée. Les effets de survol ajoutent une touche interactive, rendant la navigation encore plus fluide et moderne.

Organisation des Sections

Accueil : Présente l'univers de Treashunt avec des illustrations clés et une introduction à Iboren. L'effet parallax et les ombres ajoutées renforcent l'immersion visuelle.

Téléchargement : Offre un accès facile à tous les documents essentiels, comme le cahier des charges, les guides d'installation et le jeu lui-même, avec une introduction par un effet parallax pour maintenir la cohérence visuelle.

À propos : Comprend une frise chronologique de l'avancement du projet et une présentation des membres de l'équipe. Le parallax représentant une plaine et des nuages, combiné aux ombres, apporte une touche de profondeur et de sophistication.

En conclusion, le site web de Treashunt est conçu pour être à la fois esthétique et fonctionnel. Les choix de couleurs, l'utilisation de l'effet parallax, l'ajout d'ombres pour la profondeur, et une conception soignée de la navbar offrent une expérience utilisateur agréable. Ce design reflète fidèlement l'univers du jeu et les valeurs de notre entreprise.

3.3 Cyril - Directeur Technique

3.3.1 Multijoueur

Ancien système

Précédemment nous utilisions juste Netcode for Game Object pour le multijoueur. Ce système présentait plusieurs inconvénients majeurs en matière de sécurité et d'expérience utilisateur. Pour que les joueurs puissent se connecter entre eux, il était nécessaire de partager leur adresse IP ainsi que le port réseau utilisé pour la connexion, ce qui posait un problème de sécurité.

De plus, les utilisateurs devaient ouvrir manuellement des ports dans le pare-feu de leur PC pour autoriser les connexions entrantes. Cette exigence non seulement compliquait le processus de connexion, mais présentait également des risques supplémentaires pour la sécurité. Ces étapes techniques créaient également une mauvaise expérience utilisateur, décourageant de nombreux joueurs moins expérimentés ou soucieux de la sécurité de participer à des parties multijoueurs.

FIGURE 15 – Ancien système multijoueur

Nouveau système

C'est pourquoi nous avons choisi de faire un refonte complète du système multijoueur. Pour cela nous avons décidé d'intégrer Unity Lobby et Unity Relay en complément de Netcode for GameObject.

Unity Lobby et Unity Relay sont des services fournis par Unity qui simplifient grandement la création et la gestion de parties multijoueurs en ligne.

Voici une brève présentation de ces services et les raisons pour lesquelles nous les avons choisies :

- Unity Lobby

Unity Lobby permet aux joueurs de créer, rejoindre et naviguer facilement dans des lobbies multijoueurs. Les principaux avantages de ce service incluent :

Simplicité de Création et de Gestion des Parties : Avec Unity Lobby, les joueurs peuvent rapidement créer des lobbies ou en rejoindre d'existants, simplifiant ainsi le processus de mise en relation des joueurs.

Personnalisation et Configuration Flexible : Les lobbies peuvent être configurés pour répondre aux besoins spécifiques de notre jeu, permettant des options de personnalisation telles que la sélection des parcours d'obstacles, le nombre de participants, et d'autres paramètres de jeu.

Intégration Facile avec Unity : Unity Lobby s'intègre de manière fluide avec Unity et NGO, facilitant son implémentation dans notre projet.

- Unity Relay

Unity Relay complète Unity Lobby en fournissant un moyen efficace de gérer le routage des données de jeu entre les différents joueurs sans nécessiter de serveur dédié. Ses avantages incluent :

Faible Latence et Haute Disponibilité : Unity Relay offre une transmission de données rapide et fiable, essentielle pour un jeu de course où la réactivité et la précision sont cruciales.

Évolutivité : Ce service permet de gérer facilement un grand nombre de joueurs simultanément, garantissant une expérience de jeu fluide même en cas d'augmentation de la charge.

Économie de Coûts : En évitant la nécessité de mettre en place et de maintenir des serveurs dédiés, Unity Relay réduit les coûts de développement et d'infrastructure.

Raisons du Changement

La transition de Netcode for GameObjects seul à l'utilisation combinée de Unity Lobby et Unity Relay s'explique par plusieurs facteurs clés :

Facilité d'Utilisation : L'intégration de Unity Lobby permet de simplifier considérablement le processus de création et de gestion des parties multijoueurs, offrant une meilleure expérience utilisateur.

Amélioration des Performances : Unity Relay assure une transmission de données rapide et fiable, ce qui est crucial pour la réactivité nécessaire dans un jeu de course multijoueur.

Flexibilité et Évolutivité : Les services Unity sont conçus pour évoluer facilement avec le nombre de joueurs et les besoins du jeu, sans nécessiter de modifications majeures de l'infrastructure.

Support et Documentation : Unity fournit une documentation exhaustive et un support actif pour ses services, facilitant leur implémentation et leur utilisation optimale.

L'adoption de Unity Lobby et Unity Relay nous a permis d'améliorer significativement le système multijoueur de notre jeu, offrant ainsi une expérience de jeu plus fluide, réactive et agréable pour nos joueurs.

Détail du multijoueur

Pour le nouveau système multijoueur, nous avons décidé d'utiliser une nouvelle scène qui centralise tous les menus relatifs au multijoueur, afin de simplifier et d'améliorer l'expérience utilisateur. Cette scène se compose de plusieurs pages, chacune ayant un rôle spécifique dans la gestion et la participation aux parties multijoueurs.

Page d'Authentification

La première étape pour les joueurs est de s'authentifier. Cette page d'authentification permet aux joueurs de choisir leur pseudonyme et de se connecter aux services d'Unity. L'authentification est essentielle pour gérer les identités des joueurs et garantir la sécurité des interactions en ligne. Une fois connecté, le joueur peut accéder aux autres fonctionnalités multijoueurs, assurant une expérience personnalisée et sécurisée.

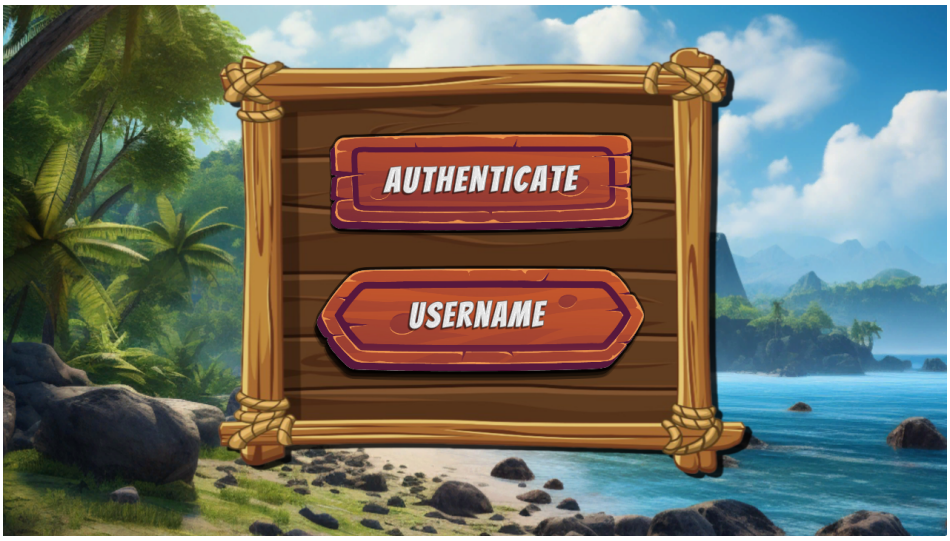


FIGURE 16 – Page d'authentification

Page de Liste des Lobbies

Après s'être authentifié, le joueur accède à la page de liste des lobbies. Cette page affiche tous les lobbies existants, permettant au joueur de naviguer et de choisir un lobby à rejoindre. Les lobbies sont présentés avec des informations telles que le nom du lobby, le nom de la carte choisie pour la course, et le nombre maximum de joueurs autorisés. Cette fonctionnalité facilite la recherche et la participation aux parties avec ses amis ou d'autres joueurs en ligne, améliorant ainsi la convivialité et la fluidité de l'expérience multijoueur.

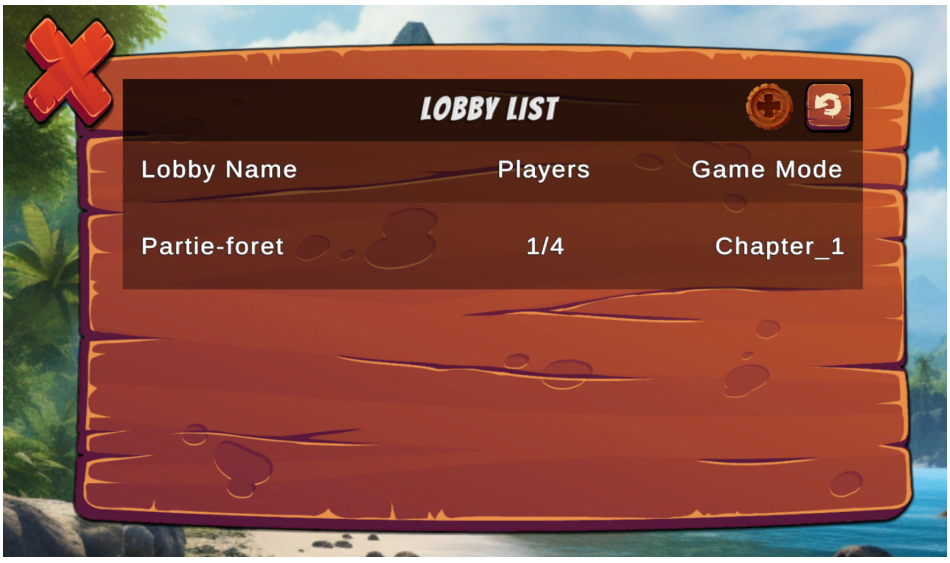


FIGURE 17 – Fenêtre liste des lobbys

Page de Création de Lobby

Pour les joueurs souhaitant créer leur propre groupe, la page de création de lobby offre une interface simple et intuitive. Ici, le joueur peut entrer un nom pour son lobby, ce qui le rend identifiable par les autres joueurs. De plus, le créateur du lobby peut choisir la carte sur laquelle se déroulera la course et définir le nombre maximum de joueurs autorisés dans ce lobby. Cette personnalisation permet de configurer des parties selon les préférences et les besoins du créateur, assurant ainsi une expérience de jeu adaptée et agréable.

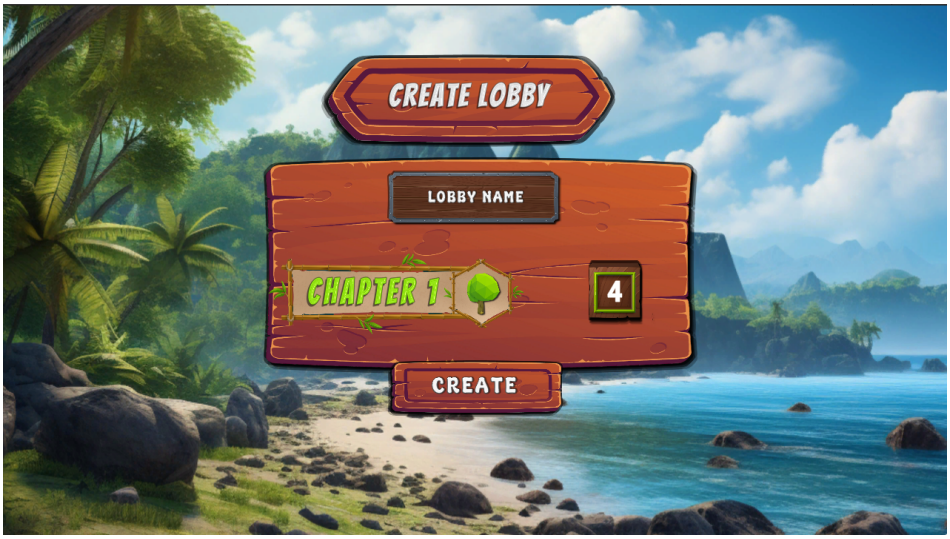


FIGURE 18 – Page de création du lobby

Page du Lobby pour rejoindre

Une fois qu'un joueur a rejoint un lobby, il est redirigé vers la page du lobby rejoint. Sur cette page, il peut voir la liste des différents joueurs présents dans le même groupe, ainsi que des informations cruciales comme la carte sélectionnée et le nombre maximum de joueurs. Le créateur du lobby dispose de pouvoirs supplémentaires sur cette page : il peut expulser n'importe quel joueur du lobby si nécessaire, et il peut lancer la partie lorsque tout le monde est prêt. En lançant la partie, tous les joueurs sont automatiquement téléportés sur la carte choisie, prêts à commencer la course. Cette organisation permet une gestion efficace et fluide des parties, assurant que tout se passe sans accroc et que les joueurs peuvent se concentrer sur le jeu.

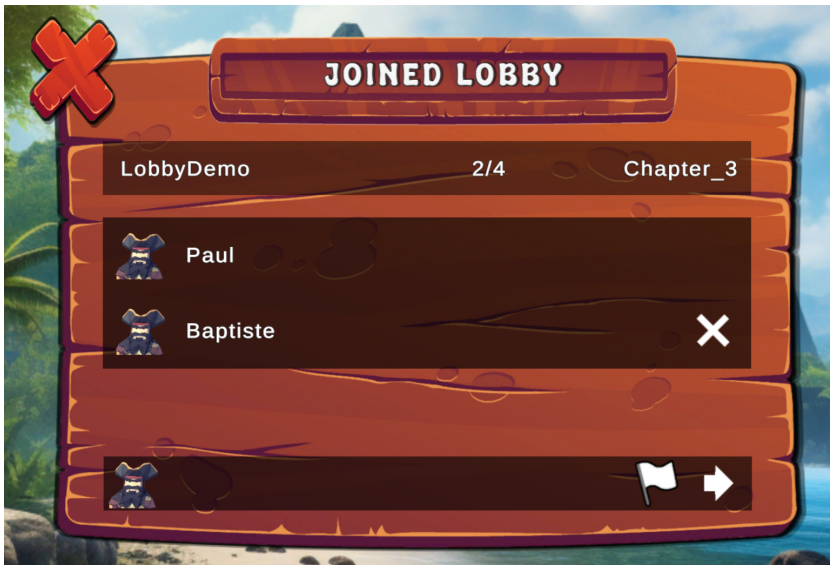


FIGURE 19 – Page du lobby pour rejoindre

En centralisant toutes ces fonctionnalités dans une seule scène dédiée au multijoueur, nous avons créé une interface cohérente et intuitive qui simplifie considérablement l'organisation et la participation aux parties multijoueurs. Chaque page est conçue pour répondre à des besoins spécifiques, de l'authentification à la gestion des lobbies, en passant par la création et la navigation des lobbies. Ce nouveau système multijoueur, intégrant Unity Lobby et Unity Relay, offre une expérience améliorée et sécurisée, répondant aux attentes des joueurs tout en simplifiant les aspects techniques de la gestion multijoueur.

Toutes ces différentes pages sont gérées par des scripts dédiés à chacune d'elles. L'ensemble permet une interaction fluide et simple avec Unity Lobby et Relay.

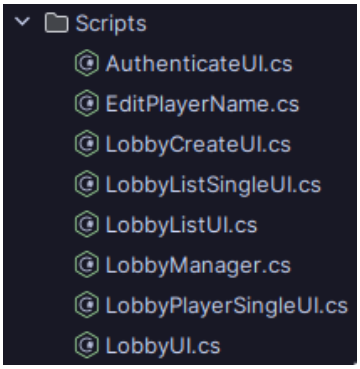


FIGURE 20 – Scripts utiles au lobby

3.3.2 Séparation multijoueur / jeu solo

Bien que notre jeu propose un mode solo et un mode multijoueur, nous utilisions auparavant le même préfab pour le joueur dans les deux modes, ce qui a entraîné d'importants problèmes.

Problème de Joueurs en Mode Spectateur

Au cours du développement initial de notre mode multijoueur, nous avons rencontré un bug majeur qui affectait la jouabiliter du jeu. Lorsqu'un joueur rejoignait une partie, il arrivait souvent que l'un des deux joueurs soit placé en mode spectateur, incapable de bouger ou de contrôler son personnage. Ce problème rendait le mode multijoueur complètement inutilisable et devait absolument être résolu.

La cause de ce bug était liée à la présence d'un script de gestion de l'inventaire du joueur. Une fois ce script retiré, le mode multijoueur fonctionnait à nouveau sans problème.

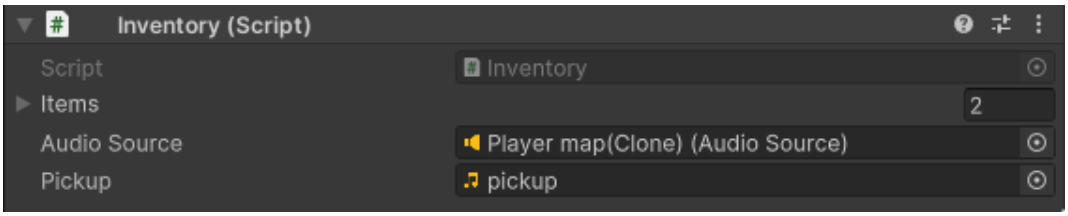


FIGURE 21 – Componnent problématique

Séparation multijoueur / jeu solo

À la suite de ce bug, nous avons décidé de séparer les modes multijoueur et solo en utilisant des préfab différents pour chaque mode. Cela signifie que les deux joueurs n'auront pas les mêmes composants. Le joueur multijoueur n'inclut donc pas le script de gestion de l'inventaire, qui est de toute façon inutile dans les courses multijoueurs.

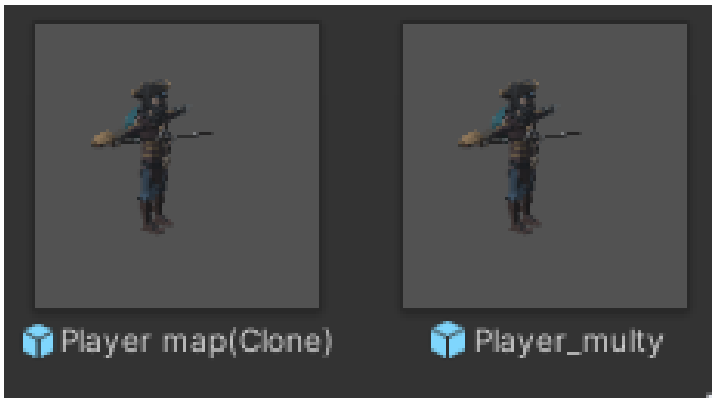


FIGURE 22 – Deux prefab du joueur

Cette différenciation des joueurs inclut également l'utilisation de scripts et de composants différents pour les préfab des modes multijoueur et solo.

3.4 Paul - Chef de Projet

3.4.1 IA : ennemis

Dans le cadre du développement de Treashunt, nous avons intégré deux systèmes d'IA principaux : les ennemis et les pièges. Alors que Cyril s'est concentré sur la création des pièges, je me suis consacré au développement de l'intelligence artificielle des ennemis. Cet aspect du jeu améliore grandement l'expérience utilisateur en ajoutant de la difficulté à l'aventure.

Objectif des ennemis

Une particularité de notre jeu est que le joueur ne peut pas attaquer les ennemis car c'est avant tout un hard-platformer où le coeur de l'aventure réside dans les parcours. Cette décision de gameplay simplifie la gestion des scripts et se concentre sur la nécessité d'évasion et de stratégie plutôt que sur le combat. Les ennemis ont pour principal objectif de chasser et d'attaquer le joueur, augmentant ainsi le défi et l'intensité du parcours. Leurs comportements sont programmés pour détecter le joueur, le pourchasser et l'attaquer une fois à proximité. Nous aborderons ces aspects techniques en détails dans les pages suivantes.

Prefab de l'ennemi

Pour la conception visuelle des ennemis, nous avons opté pour des modèles de squelettes provenant de l'asset pack Polygon Pirate d'Unity Store. Ces squelettes, avec leur allure sinistre, s'intègrent parfaitement dans l'univers de Treashunt. Afin de faciliter l'intégration des scripts et leur réplication sur les différentes maps, j'ai créé un prefab de l'ennemi. Ce prefab permet de cloner facilement les ennemis et de les paramétrer indépendamment les uns des autres.

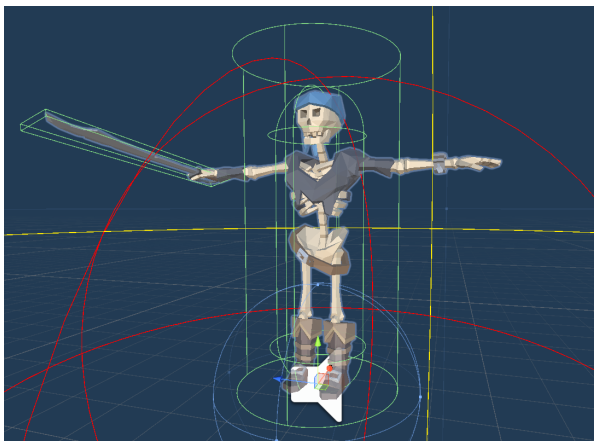


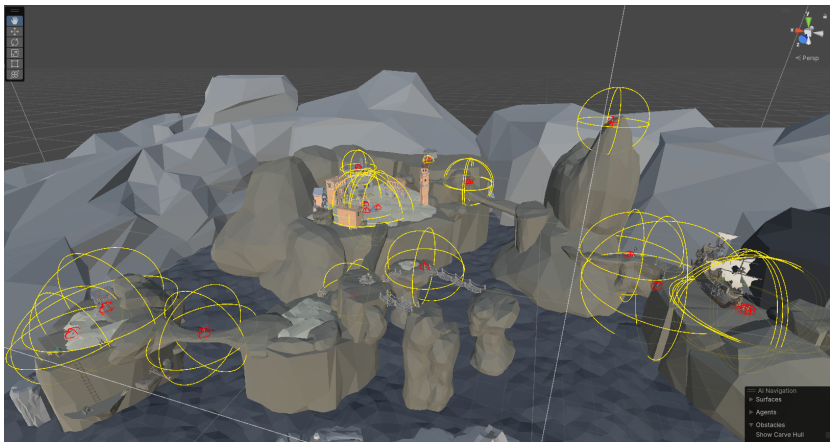
FIGURE 23 – Prefab de l'ennemi avec aperçu des colliders (en vert)

De plus, j'ai ajouté et positionné une épée dans la main de l'ennemi. Cette épée est conçue pour infliger des dégâts au joueur dès que son box collider entre en contact avec le capsule collider du joueur et que l'animation d'attaque est jouée. Nous verrons par la suite le fonctionnement de son script et le système d'animation de l'ennemi.

Placement des ennemis

Les ennemis ne sont pas placés aléatoirement sur les maps. Leur positionnement est minutieusement planifié pour maximiser la pression sur le joueur et compliquer son parcours. En effet, ils sont placés à des endroits stratégiques, notamment dans des passages étroits, près des objectifs importants ou dans des zones où le joueur doit accomplir des actions spécifiques. Cela oblige le joueur à rester constamment vigilant et dans la maitrise de ses mouvements afin de ne pas mourir et de reprendre au denier checkpoint !

(a) Un point stratégique - Chap 1



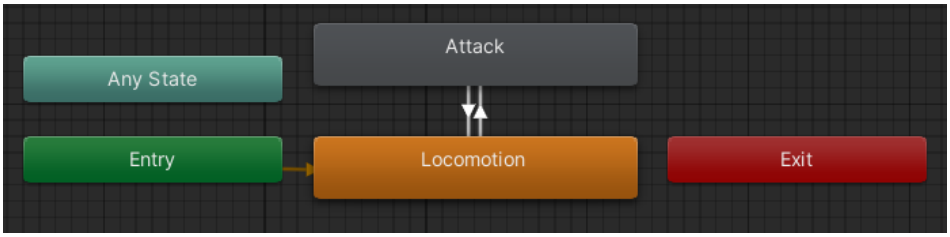
(b) Positionnement des ennemis - Chap 2

FIGURE 24 – Exemples de positionnement des ennemis

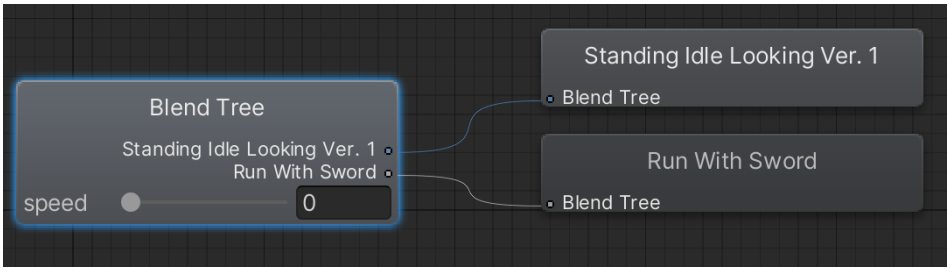
Animation de l'ennemi

La prefab de l'ennemi est équipée d'un animator qui gère ses différentes animations. J'ai veillé à ce que la logique d'animation reste simple et efficace. Nous utilisons un Blend Tree "Locomotion" pour gérer la transition entre les états de repos (idle) et de course (run) via un float "speed" variant de 0 à 1.

Ce Blend Tree est également relié à l'animation d'attaque, permettant une transition fluide et réaliste lorsque l'ennemi détecte et chasse le joueur. Toutes les animations du jeu, y compris celles des ennemis, ont été sélectionnées et téléchargées depuis le site Mixamo.



(a) Animator ennemi



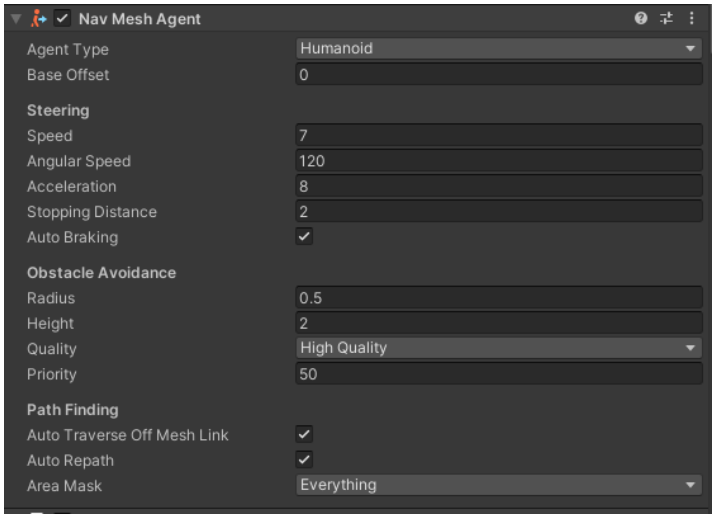
(b) Blend Tree Locomotion

FIGURE 25 – Animation de l'ennemi

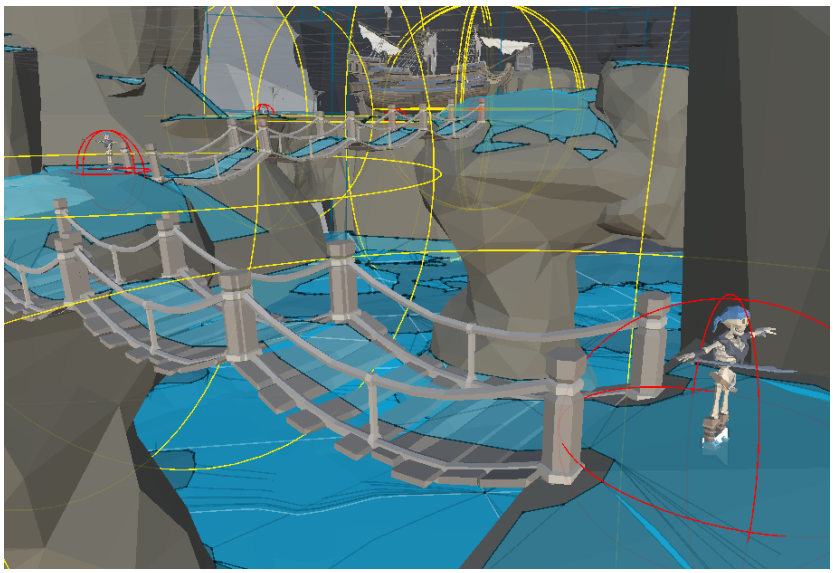
IA de l'ennemi et scripts

Intéressons-nous maintenant au cœur du fonctionnement de l'IA de l'ennemi. Pour doter les ennemis de capacités de navigation intelligentes, j'ai ajouté le composant Nav Mesh Agent à la prefab. Ce composant, fourni par Unity, permet à l'IA de naviguer de manière autonome sur une surface définie par un Nav Mesh. En d'autres termes, le Nav Mesh Agent permet à l'ennemi de calculer le chemin le plus efficace pour se déplacer vers une cible, en évitant les obstacles et en suivant les contours du terrain.

Solution au problème : Attention à ne pas oublier de générer le Nav Mesh Surface sur votre map ! C'est la raison pour laquelle mon ennemi ne se déplaçait pas et ne chassait pas le joueur sur le map : je n'en avais pas généré !



(a) Nav Mesh Agent ennemi



(b) Nav Mesh Surface (en bleu)

FIGURE 26 – Nav Mesh Agent et Surface

Grâce au Nav Mesh Agent, nous pouvons ajuster plusieurs paramètres cruciaux pour le comportement de l'ennemi. Le premier paramètre est la vitesse de déplacement de l'IA, qui détermine à quelle rapidité l'ennemi se déplace sur la carte. Cela nous permet de créer des ennemis avec des vitesses variées, ajoutant ainsi de la diversité dans les rencontres du joueur. Le second paramètre important est la distance d'arrêt (stopping distance). Ce paramètre gère la distance à laquelle l'IA s'arrête par rapport au joueur.

En réglant correctement la stopping distance, nous évitons que l'ennemi ne se superpose au joueur lors de l'attaque, ce qui améliore l'immersion et le réalisme des interactions.

Le script de l'ennemi gère les aspects critiques du comportement de l'IA, notamment la détection du joueur, la chasse et l'attaque. Les paramètres de combat que l'on a mis par défaut incluent un délai d'attaque (attackCD) de 3 secondes, une portée d'attaque (attackRange représentée par la sphère rouge dans les images précédentes) de 1 mètre, et une portée d'aggroRange (zone de détection représentée par la sphère jaune dans les images précédentes) de 4 mètres :

```
6  public class Enemy : NetworkBehaviour
7  {
8      [Header("Combat")]
9      [SerializeField] float attackCD = 3f;
10     [SerializeField] float attackRange = 1f;
11     [SerializeField] float aggroRange = 4f;
12
```

En réalité, après avoir testé dans le jeu, nous avons changé l'attackCD à 1 seconde car c'était trop lent, l'attackRange à 3 secondes et l'aggroRange à 30 mètres. Certains ennemis ont d'ailleurs une porté d'aggro plus ou moins grande selon leur emplacement dans la map pour des raisons d'optimisations, de bugs etc.

Lorsqu'un joueur entre dans l'aggroRange, l'ennemi commence à le poursuivre en définissant la position du joueur comme nouvelle destination (le second if sert à jouer les bruits de pas et gémissements de l'ennemi, j'en reparlerai dans la partie Sound Design) :

```
if (newDestinationCD <= 0 && distanceToPlayer <= aggroRange)
{
    newDestinationCD = 0.5f;
    agent.SetDestination(player.transform.position); // chasing Player
    if (!isChasing && !animator.GetCurrentAnimatorStateInfo(0).IsName("Attack"))
    {
        isChasing = true;
        footstepCoroutine = StartCoroutine(PlayFootstepSounds());
        randomSoundCoroutine = StartCoroutine(PlayRandomSounds());
    }
}
```

Si le joueur est à portée d'attaque, l'ennemi déclenche une animation d'attaque, vérifiant la distance et réinitialisant le délai d'attaque :

```
if (timePassed >= attackCD)
{
    if (distanceToPlayer <= attackRange)
    {
        animator.SetTrigger(IsAttacking);
        // play attack sound
        timePassed = 0;
    }
}
timePassed += Time.deltaTime;
```

Enfin lorsque le joueur sort de l'aggroRange, l'ennemi arrête de le poursuivre :

```
else if (distanceToPlayer > aggroRange)
{
    if (isChasing)
    {
        isChasing = false;
```

Concernant le script de l'épée, lorsqu'elle entre en collision avec un autre objet (OnTriggerEnter), le script vérifie si cet objet est le joueur. Si c'est bien le cas, et que l'ennemi est en train d'attaquer (IsAttacking), la méthode TakeDamage est appelée sur le HealthManager du joueur pour lui infliger des dégâts :

```
private void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("Player"))
    {
        if (playerHealth != null && IsAttacking())
        {
            playerHealth.TakeDamage();
```

Nous reparlerons en détail du HealthManager et de la fonction TakeDamage dans la partie Système santé joueur. La méthode IsAttacking utilise l'Animator pour vérifier si l'ennemi est en train de jouer l'animation d'attaque :

```
private bool IsAttacking()
{
    if (animator != null)
    {
        return animator.GetCurrentAnimatorStateInfo(0).IsName("Attack");
    }
    return false;
}
```

3.4.2 Menu pause

Un menu pause est indispensable dans un jeu vidéo, permettant au joueur de mettre le jeu en pause et de modifier certains paramètres selon ses envies. Dans Treashunt, nous avons adopté un menu des paramètres composé de trois boutons : Reprendre, Paramètres et Menu. Les boutons Reprendre et Menu permettent respectivement de revenir au jeu ou de quitter vers le menu principal. Le bouton Paramètres ouvre une nouvelle interface permettant de régler le volume de la musique du jeu ainsi que les effets sonores à l'aide de curseurs.

Le design du menu reste fidèle au style du jeu, avec un aspect bois et une police de texte uniforme, tout en restant simple et intuitif. Pour accéder au menu des paramètres, il suffit d'appuyer sur la touche ECHAP du clavier, sans avoir besoin de cliquer sur un bouton Option, ce qui pourrait cacher le gameplay au joueur.

Ce menu est un GameObject présent et désactivé dans chaque scène du jeu. Il s'active et s'affiche donc logiquement quand le joueur appuie sur ECHAP. Il est géré par le script PauseMenu. La méthode Update vérifie constamment si le joueur appuie sur cette touche ECHAP. Si ECHAP est pressé, le script bascule entre les états de pause et de reprise du jeu :

```
void Update()
{
    if (Input.GetKeyDown(KeyCode.Escape))
    {

        Cursor.visible = gameIsPaused;
        if (gameIsPaused)
        {
            Resume();

        }
        else
        {
            Paused();
        }
    }
```

La méthode Resume est appelée pour reprendre le jeu. Elle désactive l'interface utilisateur du menu de pause, remet l'échelle de temps à 1 (ce qui fait que le jeu reprend son cours normal) et met à jour l'état de pause :

```
public void Resume()
{
    pauseMenuUi.SetActive(false);
    Time.timeScale = 1;
    gameIsPaused = false;
    Cursor.lockState = CursorLockMode.None;
    Timer.timer.ShowTimer(false);

}
```

La méthode Paused est appelée pour mettre le jeu en pause. Elle active l'interface utilisateur du menu de pause, arrête le temps du jeu en réglant l'échelle de temps à 0 et met à jour l'état de pause :

```
void Paused()
{
    Cursor.lockState = CursorLockMode.Locked;
    pauseMenuUi.SetActive(true);
    openSound.SetActive(false);
    Time.timeScale = 0;
    gameIsPaused = true;

    Timer.timer.ShowTimer(true);

}
```

Deux méthodes supplémentaires permettent de gérer l'affichage de la fenêtre des paramètres : SettingsButton, active la fenêtre des paramètres et CloseSettingsWindow, désactive la fenêtre des paramètres :

```
public void SettingsButton()
{
    settingsWindow.SetActive(true);
}

public void CloseSettingsWindow()
{
    settingsWindow.SetActive(false);
}
```

3.4.3 Système santé joueur

Le système de santé du joueur est géré par un GameObject présent dans toutes les scènes du jeu, assurant ainsi une continuité et une cohérence de la santé du joueur tout au long de l'aventure. Le script principal responsable de la gestion de ce système est le HealthManager, dont les fonctionnalités et la logique sont détaillées ci-dessous.

Ce script gère l'état de la santé, les animations associées à la perte ou au gain de vie, et la mise à jour visuelle de l'interface utilisateur pour refléter ces changements. Comme l'a décrit Léo dans la barre de vie, nous utilisons des images pour l'interface utilisateur stockées dans une liste de GameObjects.

La méthode TakeDamage diminue la santé du joueur de 25 points lorsqu'il subit des dégâts, met à jour l'interface utilisateur et lance une animation de dégâts. C'est cette même méthode qui est appelée dans le script de l'épée sur la vie du joueur quand elle entre en colision avec :

```
public void TakeDamage()
{
    if (current != 0)
    {
        healthAmount -= 25f;
        ShowImage(current-1);
        StartCoroutine(PlayDamageAnimation());

    }
}
```

La méthode ShowImage gère l'affichage des images de santé dans l'interface utilisateur. Elle active ou désactive les images en fonction de l'état actuel de la santé :

```
public void ShowImage(int act)
{
    for (int i = 0; i < 5; i++)
    {
        if (i == act)
        {
            images[i].SetActive(true);
            current = i;
        }
        else
        {
            images[i].SetActive(false);
        }
    }
}
```

3.4.4 Sound Design

Le sound design de Treashunt est sans doute l'une des parties les plus cruciales et est essentielle pour immerger les joueurs dans l'univers que nous avons créé. Tous les sons et musiques utilisés dans le jeu sont libres de droits et ont été téléchargés sur les sites *OPENGAMEART.ORG* et *FREESOUND.ORG*. Ces plateformes offrent une vaste bibliothèque de sons et de musiques de qualité, adaptés à une variété de besoins et de thèmes, et nous ont permis d'incorporer des effets sonores et des musiques d'ambiance qui enrichissent l'expérience de jeu.

Musique

Chaque scène du jeu possède une musique spécifique qui correspond au thème de la map. Ces musiques, gérées par un AudioManager, sont conçues pour tourner en boucle, assurant une ambiance continue pour le joueur. La sortie audio pour la musique est configurée sur la piste Music du MainMixer, ce qui permet de contrôler indépendamment le volume de la musique par rapport aux autres sons du jeu.
L'AudioManager est un GameObject présent dans chaque scène qui contient une source audio. La gestion de la musique est simple mais efficace, grâce à un script basique qui se contente de jouer l'audio :

```
public class AudioManager : MonoBehaviour
{
    public AudioClip background;
    public AudioSource audiosource;
    void Start()
    {
        audiosource.clip = background;
        audiosource.Play();
    }
```

Ce script initialise la source audio avec le clip de musique de fond spécifique à la scène et commence à le jouer dès le début de la scène. Cette approche garantit que chaque map ait son propre thème musical.

AudioManager et MainMixer

Pour gérer la distinction entre la sortie de la musique et des effets sonores, j'ai mis en place un MainMixer. L'AudioManager se concentre uniquement sur la partie musique de chaque map, permettant une gestion centralisée de l'ambiance musicale. Le MainMixer est configuré pour séparer les canaux audio, avec une piste dédiée à la musique et une autre aux effets sonores. Cela permet de régler individuellement les niveaux de volume de la musique et des effets sonores, assurant un équilibre sonore optimal.

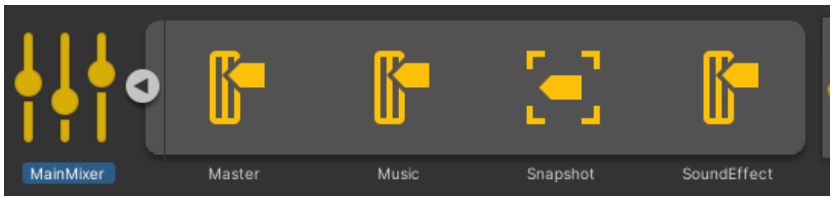


FIGURE 27 – MainMixer aves les différents canaux audio

Le joueur pourra régler le volume de la musique et des effets sonores depuis le menu pause grâce à ce MainMixer. Nous en reparlerons en détail par la suite.

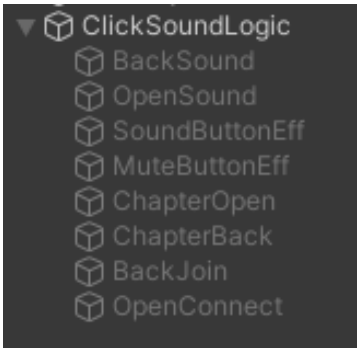
Effets sonores

Les effets sonores sont un véritable pilier dans le feedback et l'interactivité de Treashunt. Chaque action du joueur, chaque interaction avec l'environnement et chaque événement spécifique sont accompagnés d'effets sonores appropriés. Ces sons sont également configurés pour être diffusés via le MainMixer, mais cette fois-ci sur la piste Sound Effect. Cela permet de maintenir une clarté et une distinction entre les différents types de sons, améliorant ainsi l'expérience auditive globale du jeu.

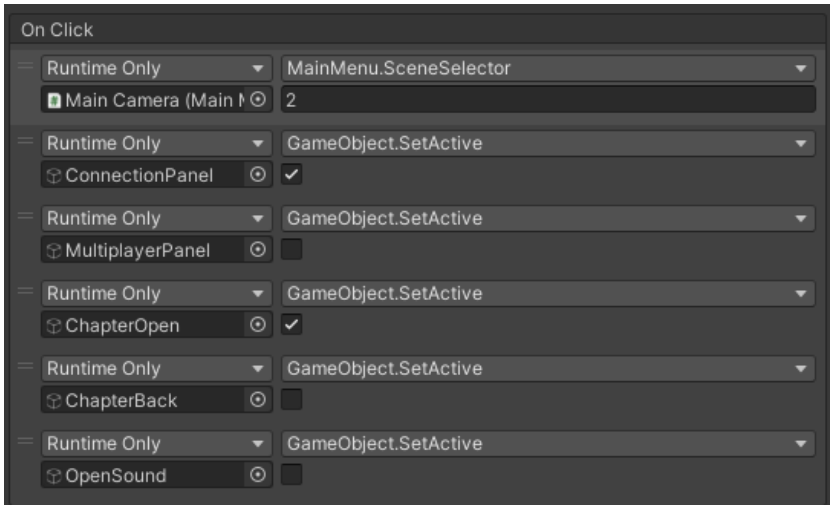
Les effets sonores incluent des éléments tels que les bruits de pas, des sauts, de la mort, les sons d'attaque des ennemis, les interactions avec les objets et échanges, les sons des clics de l'UI et bien d'autres encore... Chacun de ces sons a été soigneusement choisi et intégré pour correspondre parfaitement aux actions et événements qu'ils accompagnent.

Menu principal, menu multijoueur et menu pause

Les effets sonores du menu principal et du menu du multijoueur sont les clics de l'utilisateur sur les différents boutons. Le menu pause reprend la même implémentation avec un son bonus quand le joueur appuie sur ECHAP. Pour cela j'ai implémenté une logique de GameObject ayant une source audio qui, en s'activant jouer le son du clic ou le son du retour en arrière. Pour pouvoir les réutiliser au fur et à mesure il est important de les activer et désactiver à certaines étapes. C'est ce qui est fait pour chaque bouton via le système OnClick dans l'inspecteur qui appelle la méthode GameObject.SetActive pour activer le GameObject et donc jouer le son :



(a) Logique des GameObjects



(b) Système OnClick appellant la méthode

FIGURE 28 – Effets sonores menu principal / multi / pause

Plage (chapitre 0)

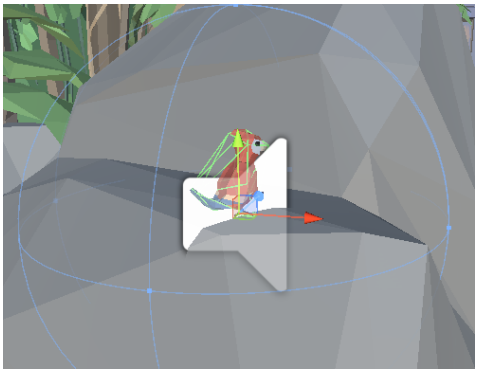
Pour les effets sonores de la plage j'ai ajouté un petit perroquet qui pousse un cri toutes les 8 à 15 secondes. Cet intervalle est géré par ce script :

```
public class ParrotSoundPlay : MonoBehaviour
{
    public AudioSource audioSource;
    public float minDelay = 7f;
    public float maxDelay = 15f;

    private void Start()
    {
        StartCoroutine(PlaySoundWithRandomDelay());
    }

    private IEnumerator PlaySoundWithRandomDelay()
    {
        while (true)
        {
            float delay = Random.Range(minDelay, maxDelay);
            yield return new WaitForSeconds(delay);
            audioSource.Play();
        }
    }
}
```

(a) Script perroquet



(b) Asset perroquet

FIGURE 29 – Effets sonores menu principal

Il y a également la présence d'un PNJ narrateur qui a son propre effet sonore et qui est le même pour tous les PNJ narrateur de Treashunt que je vais expliquer de suite.

PNJ Narrateur

L'effet sonore des PNJ Narrateur se joue lorsque le joueur change de bulle narrative en cliquant sur E. Cela donne l'impression que le PNJ parle donc rend le jeu plus immersif. Un des trois audios est sélectionné aléatoirement et se joue via cette fonction :

(a) Exemple PNJ narrateur

```
private void PlayRandomAudioClip()
{
    if (audioClips.Length > 0)
    {
        AudioClip chosenClip = audioClips[UnityEngine.Random.Range(0, audioClips.Length)];
        audioSource.PlayOneShot(chosenClip);
    }
}
```

(b) Fonction audio

FIGURE 30 – Fonctionnement PNJ Narrateur

Joueur

Le joueur a le sound design le plus développé du jeu. Tout d'abord c'est le seul qui a le composant audio listener. C'est à dire qu'il est le seul à entendre toutes les sources de son du jeu. Il émet également du son, c'est pourquoi il a une source audio.

Commençons par les sons des pas du joueur. Pour que ce soit réaliste, j'ai mis 2 sons différents pour les 2 pieds. J'appelle cette fonction dans le script du joueur quand il marche pour jouer les sons en boucle avec un délai d'intervalle entre les deux. Ce délai est réduit pour que les bruits de pas se jouent plus rapidement quand le joueur se met à courir.

Le délai entre les pas quand il marche (minTimeBetweenStepsWalk) est de 0.45 secondes. Et de 0.35 secondes quand il court (minTimeBetweenStepsRun). On ne le voit pas dans le code mais ces valeurs ont été changé dans l'inspecteur sur Unity. Voici la fonction qui joue ces sons dans les 2 actions (remplacer la variable timeSinceLastStepWalk par timeSinceLastStepRun pour la seconde fonction) :

```
    [Header("Audio Sound")]
    public AudioSource audioSource;
    public AudioClip deathsound;
    public AudioClip jumpSound;
    public AudioClip walkSound1;
    public AudioClip walkSound2;
    public AudioClip ambiant;

    public static bool isDead = false;
    private float timeSinceLastStepWalk = 0f;
    public float minTimeBetweenStepsWalk = 0.4f;
    private float timeSinceLastStepRun = 0f;
    public float minTimeBetweenStepsRun = 0.4f;

private void PlayFootstepSound() {
    if (timeSinceLastStepWalk >= minTimeBetweenStepsWalk && !isDead)
    {
        AudioClip chosenClip = Random.Range(0, 2) == 0 ? walkSound1 : walkSound2;
        audioSource.PlayOneShot(chosenClip);
        timeSinceLastStepWalk = 0f;
    }
```

FIGURE 31 – Fonctionnement audio pas du joueur

Ensuite j'ai ajouté un son quand le joueur saute (jumpsound) qui se joue dans l'appel de la fonction Jump :

```
private void Jump() {
    // reset y velocity

    audioSource.PlayOneShot(jumpSound);
    rb.velocity = new Vector3(rb.velocity.x, 0f, rb.velocity.z);
    rb.AddForce(transform.up * jumpForce, ForceMode.Impulse);
}
```

FIGURE 32 – Script saut du joueur

Le joueur émet aussi un son quand l'épée de l'ennemi le touche. Voilà la partie de script en question dans le script de l'épée :

```
playerHealth.TakeDamage();

AudioSource playerAudioSource = other.GetComponent<AudioSource>();
if (playerAudioSource != null && playerDamageSound != null)
{
    playerAudioSource.PlayOneShot(playerDamageSound);
}
```

FIGURE 33 – Script contact épée-joueur

Puis un son quand le joueur meurt (deathsound) qui se joue dans l'appel de la fonction Dying :

```
private void Dying()
{
    if (HealthManager.healthManager != null && HealthManager.healthManager.healthAmount == 0 && !isDead)
    {
        isDead = true;
        audioSource.PlayOneShot(deathsound);
        anim.SetBool(IsDying,true);
        StartCoroutine(ResetDying());
    }
}
```

FIGURE 34 – Script mort du joueur

Inventaire

Pour l'inventaire j'ai ajouté un son quand le joueur récupère un item. J'ai mis la ligne de code dans le script de l'inventaire, dans la fonction AddItem :

```
        if (items[i] != null && items[i].name == itemToAdd.name)
        {
            items[i].count += itemToAdd.count;
            audioSource.PlayOneShot(pickup);
            Debug.Log(itemToAdd.count + " " + itemToAdd.name + " added to inventory.");
            return true;
        }
    }
    else
    {
        if (items[i].name == "")
        {
            items[i] = itemToAdd;
            audioSource.PlayOneShot(pickup);
            Debug.Log(itemToAdd.count + " " + itemToAdd.name + " added to inventory.");
            return true;
        }
```

FIGURE 35 – Script inventory

Echelles

J'ai également ajouté un son lorsque le joueur monte ou descend une échelle. Cela suit le meme principe qu'avec les pas. C'est la même structure de fonction mais avec des sons de pas différents.

FIGURE 36 – AudioSource de l'échelle

Ennemis

Le sound design des ennemis est semblable au joueur. J'utilise la même logique pour les bruits de pas que le joueur. La nouveauté réside dans le son de l'attaque qui se joue dans l'animation de l'attaque à un moment clé.

Enfin pour une meilleure immersion, j'ai ajouté une fonction qui fait que l'ennemi joue des gémissements de monstre à des moments aléatoires. Il y en a 3 différents. Elle est appelée lorsque l'ennemi chasse le joueur. Voici la fonction qui gère cela :

```
private IEnumerator PlayRandomSounds()
{
    while (isChasing && !PlayerMovementAdvanced.isDead)
    {
        if (!animator.GetCurrentAnimatorStateInfo(0).IsName("Attack"))
        {
            AudioClip randomSound = randomSounds[UnityEngine.Random.Range(0, randomSounds.Length)];
            audioSource.PlayOneShot(randomSound);
            yield return new WaitForSeconds(randomSoundInterval);
        }
        else
        {
            yield return null;
        }
    }
```

FIGURE 37 – Fonction gémissements ennemis

Mini-jeu : Village Trader

Pour les échanges avec les Trader, j'ai décidé de vraiment pousser le Sound Design. En tout j'ai 4 audios différents : deux voix hommes et deux voix femmes qui respectivement, remercient le joueur et disent "non" pour indiquer au joueur qu'il n'a pas le bon item à échanger. Je l'ai intégré dans le script des Traders, à la suite des animations IsHappy et IsRefusing :

```
            anim.SetBool(IsHappy, true);
            StartCoroutine(PlayAudioClipsSequentially(accept, thanks));
            hasExchanged = true;
            StartCoroutine(ResetIsHappy());

        }
        else
        {
            anim.SetBool(IsRefusing, true);
            StartCoroutine(PlayAudioClipsSequentially(decline, no));
            StartCoroutine(ResetIsRefusing());
        }
```

FIGURE 38 – Script Trader Sound Design

Bumper

Lorsque le joueur saute sur le bumper, un son se joue aussi dans la fonction OnTriggerEnter :

```
private void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("Player"))
    {
        Rigidbody playerRigidbody = other.GetComponent<Rigidbody>();
        if (playerRigidbody != null)
        {
            playerRigidbody.AddForce(Vector3.up * Force, ForceMode.Impulse);
            audioSource.PlayOneShot(bounce);
        }
    }
```

FIGURE 39 – Fonctionnement audio bumper

Tresor

Enfin, dans le dernier chapitre lorsque le joueur arrive près du trésor, il entendra un son continu mystique lui indiquant qu'il a réussi sa quête !

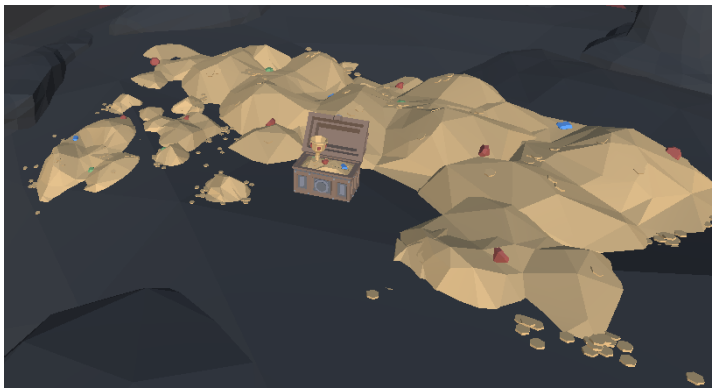


FIGURE 40 – Tresor dernier chapitre

Précision technique AudioSource

En conclusion de cette partie assez conséquente du Sound Design, je souhaite ajouter que la plupart des objets et PNJ ont un AudioSource avec leur Spatial Blend réglé sur 3D. Cela signifie que le volume de la source audio varie selon la distance du joueur comme dans la réalité. Ce détail ajoute vraiment une touche de réalisme importante à l'univers de Treashunt.

4 Conclusion

Nous sommes tous extrêmement fiers d'être arrivés au bout de ce projet. La création de TreasHunt a été une aventure incroyable, remplie de défis, de moments de doute et de tensions, mais aussi de nombreuses satisfactions. Certains membres de notre équipe ont investi énormément de temps et d'efforts pour donner vie à ce jeu, et nous sommes heureux de pouvoir enfin partager le résultat de notre travail acharné.

Ce projet a été une expérience très exigeante. Il a sollicité toutes nos compétences, que ce soit en programmation, en conception graphique, en sound design ou en gestion de projet. Nous avons fait face à de nombreux obstacles, mais grâce à notre détermination et à notre esprit d'équipe, nous avons réussi à les surmonter. Chacun de nous a tout donné pour que ce jeu soit le meilleur possible, et nous avons beaucoup appris tout au long de cette aventure.

Cher lecteur, nous espérons sincèrement que notre jeu TreasHunt vous plaira. Nous avons mis tout notre cœur et notre passion dans ce projet, et nous sommes impatients de voir vos réactions et d'entendre vos retours. Merci de prendre le temps de découvrir notre travail. Nous espérons que vous trouverez autant de plaisir à jouer à TreasHunt que nous en avons eu à le créer.