Rapport de projet

Treep

Kromm Studio

Kristen Couty - Romain Dischamp - Oscar Cornut
Mahé De Berranger - Milo Moisson

D1

Mai 2025

Tables des matières

1 Introduction

Ce cahier des charges a pour mission de définir et de présenter en détail les modalités de réalisation du projet en accord avec les critères et les attentes du client. Il constitue le premier pas vers la concrétisation du jeu vidéo *Treep*, un Rogue-like en deux dimensions. Il est possible à l'avenir que les objectifs qui y sont définis évoluent et soient légèrement modifiés selon l'état de son avancement. Bien entendu, ces éventuelles modifications n'impacteront en rien la finalité attendue du projet.

Dans *Treep*, le joueur incarne l'âme d'une gigantesque colonie d'insectes vivant au pied d'un arbre majestueux. Cependant, cet écosystème autrefois prospère est menacé par un champignon parasite, guidant inévitablement la colonie à sa perte. L'objectif est donc de parvenir à le sauver pour assurer l'avenir de la colonie. Pour cela, vous allez être confrontés à une armée d'insectes déchus, parasités par le champignon, protégeant le cœur du parasite. Malheureusement, aucun combat ne se gagne sans sacrifice et tous les insectes de la colonie n'y survivront pas. *Treep* vous plonge dans un cycle de mort et de réincarnation vous laissant ainsi plusieurs opportunités pour sauver la colonie. *Treep* est à la fois une aventure dynamique et un hommage à des mécaniques de jeu intemporelles.

Kromm Studio est une entreprise française et indépendante de développement de jeux vidéos récemment créé par cinq amis : Milo Moisson, développeur senior et chef de projet de l'équipe, ainsi que Romain Dischamp, Mahé De-Berranger, Oscar Cornut et Kristen Couty. Ainsi, l'équipe de développement de *Kromm Studio* est constituée de passionnés et d'experts dans leurs domaines respectifs, chacun apportant à l'équipe une expertise complémentaire, nous permettant de créer un jeu innovant. Nous avons l'ambition de nous démarquer sur la scène indépendante en créant un jeu qui allie parfaitement plaisir et innovation, sans oublier nos racines, l'amour du jeu vidéo avant tout.

Ce cahier des charges a été rédigé à l'intention d'EPITA, qui pourrait à terme investir dans le développement de *Kromm Studio* et ses futurs projets. Il constitue également une feuille de route interne pour l'équipe afin de structurer nos attentes, contraintes et ressources tout au long du développement de *Treep*. Notre objectif est clair, construire un partenariat avec EPITA pour faire de *Kromm Studio* un acteur incontournable du jeu vidéo indépendant.

2 Présentation du projet

Treep a pour objectif principal de procurer une expérience de jeu captivante, adaptée à un large public. Treep est un mélange entre *"tree"*, arbre en anglais, qui est le décor principal de l'histoire et *"trip"*, voyage en anglais, qui décrit ce que le joueur ressent en jouant à notre jeu.

L'histoire de *Treep* se déroule dans un grand chêne malade, gangréné par un champignon. Le joueur incarne l'âme de la colonie d'insectes vivant au pied de l'arbre. Mais un jour, un gardien de la colonie, apparait devant l'entrée principale, tacheté de blanc, les yeux vides et les antennes baissées. Dans sa folie, il tue d'innombrables insectes avant de se faire abattre par d'autres gardiens. C'est le premier contact de la colonie avec les anomalies que provoque le parasite. Ainsi, le champignon aliène peu à peu tous les habitants de l'arbre. Le joueur doit donc atteindre le champignon, puis le détruire pour sauver la colonie et maintenir l'équilibre de la forêt.

Il faudra ainsi explorer l'intérieur du chêne et combattre les sbires pour se débarrasser de la menace champignon. De plus, à chaque mort, le joueur se réincarne en un nouveau membre de la colonie, abandonnant son ancien corps au parasite, ce qui donne une dimension punitive à la mort, car le corps, une fois parasité, se retournera contre son ancien hôte.

2.1 Origine et nature du projet

L'histoire de *Treep* a été inspirée par le livre *Les Fourmis* de Bernard Werber, mais aussi par différents documentaires sur les insectes et sur la faune et la flore. Le jeu vidéo *Ori and the Blind Forest* de Moon Studios, a aussi été une source d'inspiration pour l'histoire. Pour la partie technique, nous nous sommes inspirés à la fois de *Dead Cells*, une référence du genre Rogue-like (cf 2.2), pour la génération procédurale de contenu (cf 3.7) et de *Hollow Knight*, une référence du genre Metroidvania (cf 2.2), pour les mécaniques de mouvement et de combat.

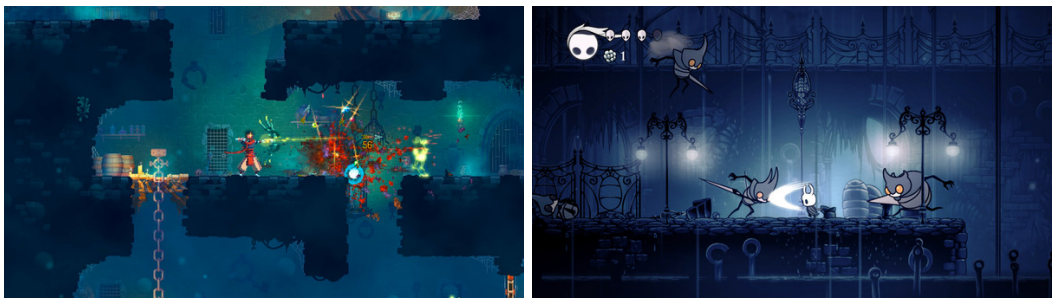

Figure 1: Images des jeux *Dead Cells* et *Hollow Knight*

2.2 État de l'art

Treep est un Rogue-like qui emprunte aussi certaines mécaniques au genre Metroidvania. Nous allons revenir sur ces genres du jeu vidéo avant d'expliquer leurs évolutions respectives.

Le terme Rogue-like désigne l'ensemble des jeux partageant les spécificités suivantes : la génération procédurale, la mort permanente et une progression dans la mort. Les trois caractéristiques précédentes induisent une grande rejouabilité puisque le contenu du jeu est infini. Le terme vient du jeu éponyme, développé par Michael Toy et Glenn Wichman en 1980. Il est inspiré par les jeux de rôle *Donjon et Dragon* qui sera précurseur dans le domaine de la génération procédurale, étant l'un des premiers à l'implémenter pour construire aléatoirement ses niveaux.

Plus récemment, *Hadès* sorti en 2020 et *Outer Wilds* sorti en 2019 sont tous deux des jeux appartenant à ce genre ayant eu un grand succès.

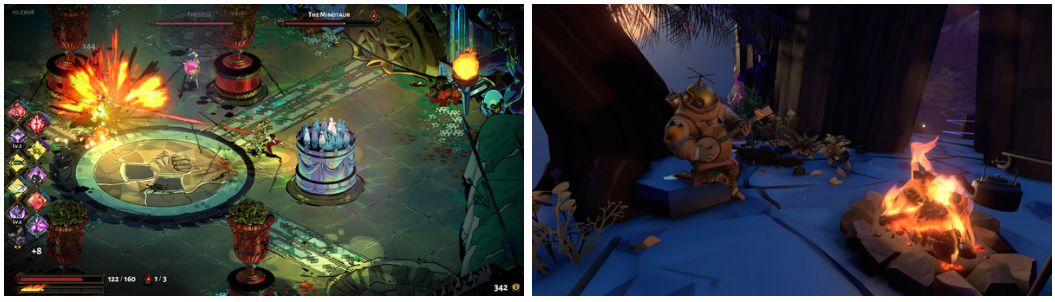

Figure 2: Images des jeux *Hadès* et *Outer Wilds*

Metroidvania est un genre de jeu vidéo, il vient du mot composite formé à partir du nom des jeux Metroid et Castlevania, les premiers du genre. Un jeu Metroidvania, généralement en 2D, possède une grande carte. Le joueur doit alors explorer l'ensemble du monde disponible et acquérir au fur et à mesure de nouvelles capacités (double saut, la possibilité de sauter sur les murs, propulsion avant, etc.) pour lui permettre d'accéder au combat final et de le gagner. La clé d'un Metroidvania est donc l'exploration, obligeant le joueur à retourner sur ses pas pour accéder à des lieux de la carte auparavant inaccessibles. Les récompenses offertes à un joueur curieux sont des bonus, des nouvelles compétences ainsi que des morceaux de l'histoire. Cette dernière est intimement liée à l'expérience de jeu, puisque que les évolutions du personnage et de l'environnement sont des explications à l'histoire globale du jeu.

Plus récemment, *Hollow Knight* sorti en 2017 et *Ori and the Blind Forest* sorti en 2015 sont tous deux des jeux appartenant à ce genre ayant eu un grand succès.

Figure 3: Image du jeu *Ori and the Blind Forest*

2.3 Objet de l'étude

Ce projet vise à développer un jeu sur un an avec le moteur de jeu *Unity*, nous permettant ainsi de nous mettre au défi tout en découvrant de nouvelles compétences techniques. La répartition des tâches au sein du groupe est un élément clé du projet, car elle favorise la collaboration et l'apprentissage mutuel. Chaque membre peut apporter ses forces, que ce soit en programmation, en conception graphique ou en écriture narrative. D'un point de vue technique, ce projet nous permet d'approfondir nos connaissances en programmation à travers le langage *C#* et de nous familiariser avec *Unity*.

D'un point de vue artistique, la nécessité de créer une histoire captivante pour notre jeu et pour notre entreprise nous a permis de développer nos talents d'écriture et d'imagination. De plus, la conception d'assets en deux dimensions et d'éléments musicaux a permis à chacun de développer sa créativité, d'expérimenter et de découvrir différents styles.

Enfin, innovation et expérimentation sont demandées pour trouver, de façon efficace, des solutions aux différents problèmes et défis pouvant apparaître lors de la création d'un jeu. De plus, ces défis étant techniques et créatifs, ils nous apprennent à sortir de notre zone de confort, ce qui est essentiel pour notre développement personnel et professionnel.

En somme, ce projet est une opportunité précieuse qui nous prépare individuellement aux réalités du monde professionnel, tout en nous apprenant à travailler efficacement en groupe.

2.4 L'entreprise (Kromm Studio)

Kromm Studio est un studio de jeu vidéo indépendant français fondé en 2024. Il est le produit de la collaboration entre cinq amis, tous reliés par leur passion pour le dixième art. C'est ce lien qui donne ainsi naissance à l'acronyme KROMM, pour Kristen, Romain, Oscar, Mahé et Milo. Réunis, les profils uniques qui forment le studio permettent une symbiose liant la technique à l'artistique.

Chacun des membres a eu la chance de voyager, de s'amuser et de rêver grâce à différentes œuvres du jeu vidéo. C'est pour ça que nous avons pour but de créer de véritables aventures permettant, à notre tour, de faire voyager les joueurs à travers des jeux vidéos au concept amusant, éprouvant ou encore émouvant. Car ces jeux sont faits par des passionnés pour des passionnés.

Mais pour l'heure, d'innombrables défis restent encore à surmonter. Le studio néophyte doit se faire une place parmi les grands studios de ce monde, qui ont tendance à prioriser l'argent et leurs actionnaires au détriment de l'expérience des joueurs. *Kromm Studio* a donc pour unique but de transmettre et de faire perdurer une passion affaiblie par l'avidité de certains studios et de faire renaître la véritable essence du jeu vidéo.

2.5 Membres du projet

2.5.1 Milo Moisson

Je m'appelle Milo Moisson, développeur et chef de projet. Je n'ai jamais fait de jeu vidéo, mais j'ai été emballé par l'idée de rejoindre *Kromm Studio* pour aider à superviser le développement de *Treep*, un projet intéressant tant techniquement qu'artistiquement. Mon expérience en programmation me permet de guider l'équipe dans leurs choix techniques.

Sur ce projet, j'ai un double rôle. En tant que développeur, je m'occupe principalement de l'algorithme de génération procédurale des niveaux qui est fortement liée au *level design*. L'algorithme doit pouvoir être flexible pour répondre à nos besoins, robuste pour être sûr de pouvoir générer un niveau à chaque fois et suffisamment efficient pour qu'aucun joueur ne remarque son exécution. En tant que chef de projet, je dois coordonner les membres sur les diverses tâches que nous devons effectuer pour mener à terme cette mission et m'assurer de la cohérence artistique du projet.

2.5.2 Romain Dischamp

Je suis Romain Dischamp, développeur, graphiste et animateur 2D, membre de *Kromm Studio* depuis sa création. Ayant commencé à dessiner très jeune, j'ai été plongé tôt dans la création artistique. Les *art books* et expositions sur le monde du jeu vidéo m'ont toujours fasciné et donné l'envie de développer à mon tour un jeu vidéo. La programmation, quant à elle, me passionne tout autant, et nourrit aujourd'hui mon besoin de création et d'apprentissage. J'ai donc choisi de rejoindre

Kromm Studio pour réaliser un objectif personnel, mais surtout pour rejoindre et travailler avec un groupe partageant des valeurs similaires aux miennes.

Mon profil mélange ainsi artistique et technique, ce qui me permet d'être à la fois créateur et responsable des assets 2D. Pour garder une vision commune du projet, je suis aussi en charge de coordonner les différents membres entre eux sur le plan artistique. D'un point de vue technique, je m'occupe de l'implémentation des différentes mécaniques de base et j'assiste aussi d'autres membres dans la conception d'une forme d'*IA*[5].

2.5.3 Mahé De Berranger

Je m'appelle Mahé de Berranger, développeur, concepteur sonore et compositeur chez *Kromm Studio*. Je suis passionné par la musique et le développement de jeux vidéo depuis l'enfance. Pour ce projet, je m'occupe de la conception des *HUD*, des menus, des mécaniques de combat, ainsi que la création d'environnements sonores immersifs. Travailler sur l'aspect audio d'un jeu vidéo me permet de fusionner mes deux passions pour la musique et la technologie.

Avant d'intégrer *Kromm Studio*, j'avais déjà participé au développement de plusieurs jeux vidéos, ce qui m'a permis d'acquérir des compétences solides en sound design et en programmation. Aujourd'hui, en plus de composer la musique et de concevoir les effets sonores qui donnent vie à notre univers, j'aide à la création des assets. Je collabore par ailleurs au développement du site web du studio. Ce travail d'équipe nous permet de rester aligné sur une vision artistique et technique commune.

2.5.4 Oscar Cornut

Je suis Oscar Cornut, développeur et *game designer*. Je suis passionné par le code depuis mon plus jeune âge. J'ai participé à plusieurs concours comme les trophées NSI ou la nuit du code qui m'ont permis d'acquérir une rigueur et une capacité de travail avec une échéance. Déjà expérimenté grâce la réalisation de deux jeux vidéos durant le lycée, j'ai décidé de rejoindre l'aventure *Kromm Studio*.

Mon rôle dans l'équipe est plus technique que graphique avant tout car c'est dans ce domaine que j'ai le plus de compétences en plus d'être mon secteur de prédilection. Je m'occupe de l'implémentation de l'*IA* des ennemis et des *PNJ*, mais également de construire une histoire créative et cohérente. En revanche, j'aime beaucoup raconter des histoires ou en lire, c'est pourquoi j'occupe une place importante dans le développement du *lore* et des personnages.

2.5.5 Kristen Couty

Je m'appelle Kristen Couty, et je suis développeur chez *Kromm Studio*. Depuis toujours, je suis passionné par la programmation et la conception des mécanismes backend pour tout type de projet. Le développement de jeux vidéos n'est pas mon secteur de prédilection, mais j'ai rejoint *Kromm Studio* après avoir été séduit par

leur vision du jeu vidéo et en quête de défis technique à relever. J'adore travailler en équipe afin de concrétiser des concepts inédits grâce à des solutions techniques novatrices.

Au sein du studio, je suis principalement assigné au développement plutôt qu'à l'aspect artistique, car mon niveau technique dépasse largement mes compétences artistiques. Cela me permet de me concentrer sur la création de mécaniques de jeu solides et performantes. Je supervise aussi le développement du système multijoueur. Cette lourde tâche inclut la gestion des interactions entre les joueurs, la synchronisation des données en temps réel et l'optimisation des performances réseau, autant d'éléments essentiels pour offrir une expérience multijoueur fluide.

3 Spécifications techniques

3.1 Mécanique de jeu et gameplay

3.1.1 Interface du jeu

Treep est un jeu 2D en vue de côté. Le HUD du jeu comporte les éléments de bases de HUD d'un jeu c'est-à-dire :

- Affichage de la vie et des capacités avec leur temps de rechargement
- Un inventaire permettant au joueur de gérer tous les objets qu'il a récupérés au cours de son aventure
- Une map interactive pour faciliter l'évolution du joueur dans le niveau et connaître la position des autres joueurs dans le mode multijoueur.

3.1.2 Déplacement

Nous avons pensé les commandes de déplacement de *Treep* pour rendre l'expérience de jeu intuitive et ergonomique. Ainsi tous les contrôles sont situés sur le clavier ce qui permet de jouer au jeu n'importe où et n'importe quand. Les contrôles de déplacement sont ceux communs à tous les jeux : Z, Q, S, D et espace. Ainsi qu'une touche supplémentaire pour intéragir avec l'environnement (E), et enfin une touche par capacité. Les contrôles sont modifiables dans le menu principal du jeu si ceux définis par défaut ne conviennent pas au joueur.

3.1.3 Structure du jeu

Le mode solo et multijoueur de *Treep* sont tous les deux constitués de trois niveaux de difficultés croissantes correspondant à trois parties de l'arbre: la souche, le tronc, et la cime. A chaque niveau le joueur doit traverser l'ensemble des salles de ce dernier afin d'arriver au combat final, un mini-boss pour les deux premiers niveaux et un boss pour le troisième.

Cependant, le joueur ne pourra pas explorer et compléter l'entièreté du niveau du premier coup, la difficulté étant devenu trop grande pour le niveau du personnage que le joueur incarne. Il devra donc retourner dans les niveaux inférieurs pour les réexplorer et développer son personnage avant de revenir à des niveaux plus complexes.

3.2 Moteur de jeu

D'un point de vue purement technique, nous développons notre jeu avec la version 6 du moteur de jeu *Unity* sortie le 17 octobre 2024. Cette mise à jour est particulièrement intéressante, car elle apporte de nombreuses nouveautés pour le développement du multijoueur et des jeux en deux dimensions facilitant le travail de développement et de *testing/QA*.

3.3 Graphisme

Pour les graphismes du jeu, nous avons choisi la deux dimensions, avec une résolution de 32x32 *pixels*. Ce choix permet de garder un jeu beau et original visuellement tout en limitant considérablement les ressources et le temps nécessaire à la réalisation des *assets* graphiques. De plus, de nos jours, peu de jeux adoptent ce style, ainsi, nous pensons que ce choix visuel permettra à notre jeu de se distinguer des autres.

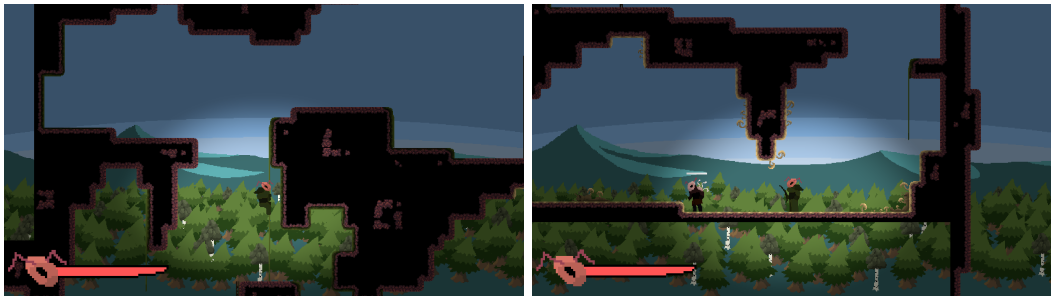

Figure 4: Images de notre jeu *Treep*

Nous avons pour le moment exclu la possibilité d'avoir recours à des packs d'assets graphiques déjà conçus, cependant nous n'abandonnons pas totalement l'idée dans l'éventualité où nous rencontrerions des difficultés quant à leur conception ou des retards de développement.

3.4 Musique et son

Les moments de calme sont accompagnés par des musiques douces et apaisantes composées principalement à partir de mélodies jouées à la guitare. Ce choix renforce l'immersion du joueur en l'ancrant dans l'atmosphère paisible des phases d'exploration. La simplicité des accords permet de ne pas détourner l'attention tout en soutenant discrètement l'ambiance du jeu.

Lorsque l'action monte en intensité, la musique évolue vers des morceaux plus rythmés. La base harmonique reste la même que celle des musiques calmes, mais des tambours puissants sont ajoutés pour renforcer la tension. Cela permet de créer une continuité sonore tout en accentuant l'urgence des combats. Le changement de rythme suffit à transformer l'ambiance sans provoquer de rupture brutale.

La plupart des musiques ont été composées spécifiquement pour le jeu, ce qui a

permis d'adapter chaque morceau à son contexte précis. Cela facilite également les transitions, car les compositions partagent des éléments communs.

Pour les effets sonores, une partie a été enregistrée ou modifiée à partir de packs d'assets existants. Ils couvrent différents aspects du jeu comme les déplacements, les attaques, les interactions avec l'environnement ou l'interface. Le choix de chaque son a demandé une attention particulière pour assurer leur qualité et leur cohérence.

Un travail a été fait pour équilibrer le volume des sons afin qu'ils ne couvrent pas la musique ou les autres effets. Certains sons sont liés aux menus et peuvent être ajustés via un curseur.

3.5 Multijoueur

Parmi les contraintes se trouve l'obligation d'intégrer un mode multijoueur à notre jeu. Dans ce mode, les joueurs ont la possibilité de former une équipe de deux à cinq personnes afin d'unir leur force pour affronter les menaces qui pèsent sur la colonie et tenter de la sauver. Leur bonne coopération est cruciale pour triompher des épreuves auxquelles ils vont être confrontés. En effet, chaque joueur incarne un rôle spécifique au sein de l'équipe, chaque rôle disposant d'un ensemble de capacités d'attaques, de défense et de support uniques, rendant chaque joueur indispensable au reste de l'équipe. Selon nous, l'ajout de rôles distincts permet une expérience de jeu plus immersive où la contribution de chaque membre est essentielle à la progression.

De plus, le défi est à la hauteur de la taille du groupe puisque la force et la stratégie du parasite évolue proportionnellement à la taille de l'équipe qui essaye de le détruire. Ainsi, plus l'équipe est nombreuse, plus l'ennemi est coriace, rendant les affrontements plus complexes, ce qui exige une véritable coordination entre les membres du groupe.

Pour renforcer l'aspect stratégique du multijoueur, les niveaux sont conçus de manière à répartir les joueurs à travers tout le niveau les forçant à faire preuve d'entraide et de synchronisation. Chaque membre de l'équipe se retrouve ainsi dans une zone différente de ses partenaires, mais les actions de chacun ont des répercussions directes sur les autres joueurs. Cela accentue la coopération entre les joueurs, qualité essentielle pour progresser dans les niveaux les plus complexes.

De plus, ce mode est doté de mécanismes permettant l'interaction entre les joueurs comme un système de vote ou encore de pointeur sur la carte pour leur permettre de prendre collectivement des décisions importantes comme choisir un itinéraire, indiquer des zones d'intérêt, signaler des dangers imminents, ou encore suggérer des points de rassemblement, tout cela dans le but d'encourager une participation active de chaque membre de l'équipe.

D'un point de vue plus technique, tous les joueurs du groupe se connectent à un serveur distant commun et dédié à leur partie. Ce serveur centralise et synchronise toutes les données envoyées par les clients, comme leurs positions, leurs actions ou encore leurs interactions avec l'environnement. Cette architecture permet de s'assurer que chaque joueur dispose d'une représentation à jour de la partie en cours, minimisant ainsi les désynchronisations. Cette architecture est mise en place avec la

librairie *Netcode* fournie par *Unity* permettant d'intégrer facilement du multijoueur dans son jeu.

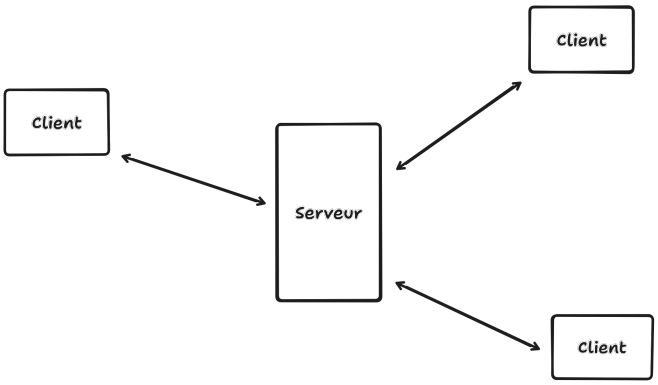

Figure 5: Schéma simplifié de l'architecture du multijoueur

3.6 Intelligence artificielle

L'IA est maintenant présente dans la plupart des nouveaux jeux, et atteint différents niveaux de complexité en fonction du problème qu'elle résout : dans *Red Dead Redemption*, un jeu de Rockstar Games, les personnages non-joueurs ont tous une histoire différente sur laquelle ils s'appuient lors de leurs interactions avec le joueur.

Par exemple, si le joueur tue un des compères du *PNJ*, il sera plus agressif et déterminé à tuer le joueur. Si, au contraire, un joueur aide un *PNJ*, il agira de façon plus clémente à son égard. Ainsi, dans cet exemple, les IA agissent en fonction des actions que le joueur est en train d'effectuer, mais aussi en fonction des actions qu'il a effectuées auparavant.

Des IA plus basiques sont aussi présentes, sans mémoire, comme dans *Pac-Man* où les fantômes chassent tous le joueur d'une manière différente, là où le fantôme rouge suit le joueur à la trace, le fantôme rose utilisera une méthode plus "réfléchie" en essayant de prédire vers où le joueur se dirige.

Conformément à la demande du client, *Treep* intégrera de l'intelligence artificielle pour le fonctionnement de ses ennemis.

- Un ennemi solitaire attaque le joueur, combat à distance si sa vie est faible, ou se cache dans un recoin pour surprendre le joueur.
- Un groupe d'ennemis attaque le joueur, fuit si la vie moyenne du groupe passe en dessous d'un certain seuil ou protège un membre affaibli.
- Les IA agissent donc en fonction des actions du joueur, du terrain sur lequel elles évoluent et enfin en fonction de leurs propres caractéristiques telles que leurs compétences ou encore l'évolution de leur vie.

L'objectif des IA étant de dynamiser l'expérience de jeu, notre but est de créer un simulacre de réflexion en élargissant le panel des comportements des *IA*. Un exemple de suite d'actions faites par une *IA* est le suivant : *Le joueur est loin, je me rapproche. le joueur est maintenant assez proche, je l'attaque. Il m'attaque, ma vie est en dessous du seuil, je fuis.*

3.7 Génération procédurale de contenu

Pour un Rogue-like comme *Treep*, les joueurs traversent, à chaque *run*, les mêmes niveaux. Cela induit certains problèmes de *game design*. Tout d'abord, dans le cas de niveaux statiques, en seulement quelques parties, les joueurs sont aptes à se repérer et n'ont plus aucune surprise des dangers ou de l'emplacement de certains bonus. Cela peut rapidement lasser les joueurs. De plus, les joueurs peuvent commencer à optimiser leur itinéraire pour éviter les obstacles au lieu de s'y confronter pour augmenter ses capacités techniques et ses réflexes de combat. Ses observations sont tirées de *playtests* d'autres *Rogue-likes* rendus publics.

Pour pallier à ce problème, nous avons choisi de faire un moteur de génération procédurale de contenu qui nous permet de générer des niveaux aléatoirement tout en gardant un certain contrôle. Pour chaque niveau, nous devons ainsi créer plusieurs pièces qui seront assemblées aléatoirement pour créer un niveau. En décrivant un niveau par un graphe, nous avons la possibilité de décider de sa forme générale. Par exemple, à quel endroit de la progression du niveau est placé un marchand de bonus.

Devoir faire plusieurs pièces pour chaque niveau demande plus de travail, mais offre un large avantage puisqu'il permet de garder une cohérence avec l'univers du jeu sur le plan graphique et le plan scénaristique. De plus, cela nous permet d'introduire manuellement des pièces qui mettent en œuvre les capacités précédemment acquises, par exemple le joueur est forcé de faire un double saut.

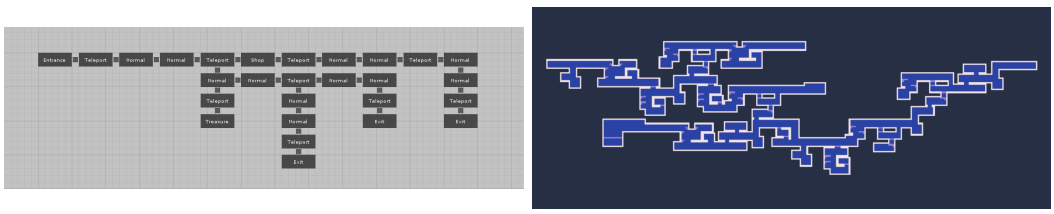

Figure 6: Le graphe de description d'un niveau et un exemple de ce niveau généré procéduralement

4 Bilan des réalisations

4.1 Mécaniques de déplacement

4.1.1 Bilan

Dans la conception d'un jeu vidéo, l'un des piliers fondamentaux d'une expérience captivante réside dans la fluidité et la réactivité des actions du personnage.

Un déplacement avec un délai ou différent de celui attendu peut nuire considérablement à l'immersion du joueur, c'est pourquoi une attention particulière a été portée à l'amélioration et à la diversification des mécaniques de mouvement. Pour ce faire, différentes actions ont été intégrées ou perfectionnées afin d'offrir un gameplay plus riche, plus dynamique et toujours plus immersif.

L'une des premières améliorations majeures concerne l'implémentation d'un système de position accroupie. Ce système permet au joueur de se baisser, mais uniquement dans des contextes où l'environnement le permet, garantissant ainsi un comportement réaliste et cohérent. Cela offre de nouvelles possibilités de gameplay, notamment en introduisant des passages étroits, des cachettes ou des zones secrètes accessibles uniquement en se faufilant. Cette mécanique apporte une dimension tactique supplémentaire, encourageant l'exploration et la prudence dans les déplacements.

Le dash est une autre mécanique qui vient enrichir les déplacements du joueur. Dans un souci d'équilibrage, celui-ci ne peut être utilisé qu'une seule fois après chaque contact avec le sol, empêchant ainsi tout abus ou enchaînement excessif. La distance du dash est également ajustée dynamiquement selon l'environnement et les statistiques du joueur, permettant d'éviter les collisions imprévues avec des éléments du décor et de garantir une transition fluide entre chaque phase de mouvement. Cette fonctionnalité augmente à la fois la mobilité du joueur et la sensation de dynamisme.

La capacité à grimper sur des éléments verticaux, comme les échelles ou les lianes, a également été revue. Une tilemap spécifique, dotée du tag "ladder", a été ajoutée pour détecter automatiquement si le joueur est en contact avec un élément grimpable. Une animation synchronisée à la vitesse a été intégrée, rendant cette interaction beaucoup plus fluide et naturelle. Cette fonctionnalité permet au joueur de passer d'un déplacement horizontal à une ascension verticale de manière cohérente, sans rupture d'immersion.

Afin de rendre le jeu encore plus réactif et satisfaisant, une synchronisation précise entre les actions du joueur et les animations du personnage a été mise en place. Chaque mouvement ou interaction génère une réponse visuelle immédiate, ce qui améliore significativement le ressenti du joueur. Par exemple, l'animation de course s'adapte dynamiquement à la direction, tandis que les différentes phases du saut sont liées à des animations spécifiques qui renforcent la lisibilité de l'action.

Le saut est une mécanique complexe qui a été soigneusement divisée en cinq phases distinctes :

Grounded : le joueur est au sol
PrepareToJump : phase de préparation
Jumping : impulsion initiale
InFlight : le joueur est en l'air
Landed : atterrissage

Cette séparation fine permet une gestion précise des états du personnage et facilite l'ajout d'effets visuels ou sonores contextuels, rendant le tout plus vivant. Par exemple, une animation spécifique peut être jouée uniquement pendant PrepareToJump, tandis qu'une autre s'active à Landed, offrant ainsi une meilleure cohérence visuelle.

Un travail important a été réalisé sur le système de collisions, souvent source de complications dans *Unity*, notamment lors des interactions avec les tilemaps. Les collisions entre le joueur et l'environnement sont désormais gérées de manière plus fine grâce à l'utilisation combinée du composant *Rigidbody2D* et du *TileMapCollider2D*. Des raycasts sont utilisés pour détecter les collisions avant qu'elles ne se produisent, ce qui permet d'ajuster en temps réel la position et la vélocité du joueur en cas d'impact avec un mur ou un plafond. Ce système garantit une meilleure fluidité et prévisibilité des mouvements.Cette fonctionnalité est utilisée pour les échelles et la détection de pièges devant éliminer le joueur tombé à l'intérieur.

En parallèle, de nombreuses variables essentielles telles que la gravité, la vitesse de déplacement, de dash ou de grimpe sont accessibles et modifiables directement dans l'éditeur Unity permettant un ajustement rapide et précis des paramètres de gameplay sans nécessiter de modifications lourdes du code source.

4.1.2 Défis rencontrés

Étant une partie essentielle dans un jeu, nous avons passé beaucoup de temps à travailler les fonctionnalités de déplacement, et par conséquent rencontré plusieurs problèmes.

Au début de la conception, le personnage se déplaçait trop rapidement et sautait trop loin. En réduisant la vélocité du personnage, cela a donné un effet plus naturel au gameplay. De plus, une impression de "glissade" apparaissait lorsque l'on courait assez vite, qui a été supprimé pour faire ressentir au joueur qu'il court sur du bois sec et non de la glace.

De même, un comportement anormal avait été rencontré quand on sautait en se collant au mur. En effet, nous étions bloqués. Le problème venait d'une boucle, qui permettait de faire descendre le joueur lorsque sa boîte de collision touchait un mur ou un plafond. La suppression de cette boucle engendrait un autre bug, moins problématique, qui laissait le joueur au plafond pendant toute la période de son saut. Ce problème a aussi été réglé d'une autre manière, en vérifiant qu'il ne touche pas le plafond durant son saut.

L'implémentation des échelles a aussi été revue à plusieurs reprises pour corriger des bogues comme l'absence de collisions à certains endroits. De plus, le personnage devait coller aux murs avec une échelle pour le monter, mais un problème de collision laissait le joueur grimper aux échelles sans qu'il ne les touche. Ce problème a été résolu en créant une tilemap spécifique aux échelles. Une vérification a été ajoutée pour s'assurer que le joueur ne puisse se relever que si l'espace au-dessus de lui est suffisant. Cette vérification détecte si un plafond empêche le redressement. Cela garantit un comportement plus réaliste et évite les problèmes de collision.

Initialement, le système d'accroupissement ne prenait pas en compte l'environnement au-dessus du joueur. Cela signifiait que le personnage pouvait occasionnellement se relever alors qu'un obstacle (comme un plafond bas) se trouvait juste au-dessus, entraînant des problèmes de collision et un possible blocage du joueur dans les décors.

La synchronisation des animations comme par exemple entre le saut et son animation a été compliquée, car il est constitué de différents états (e.g. impulsion, en vol, atterrissage, etc.). Ce à quoi s'est ajouté par la suite, les animations d'attaque, de propulsion qui pouvaient intervenir simultanément.

Ce n'est pas le seul problème que nous avons eu avec le saut, un bug permettait au personnage de convertir la vitesse de son saut en propulsion vers le haut. Cela était dû à la génération automatique des *hitboxes* par Unity, qui ne correspondait pas toujours précisément à la forme des cubes composant la carte.

Au lieu d'utiliser les *hitboxes* générées automatiquement, une *hitbox* a été définie manuellement. Cela a permis de mieux contrôler les interactions du personnage avec l'environnement et d'éviter tout comportement anormal, comme le fait d'être projeté dans les airs.

Similairement, un bogue apparaissait lorsque l'on sautait d'une certaine manière sur le plafond, ce qui brisait l'immersion du joueur, car des problèmes de collision apparaissaient et nous faisaient traverser le plafond.

L'utilisation du *dash* a aussi été une source de bogues, le joueur pouvait parfois traverser les murs, ce qui posait un problème majeur. Ce problème survenait principalement parce que la vitesse élevée du *dash* permettait au joueur de passer à travers les collisions automatiques d'Unity.

4.2 Mécaniques de combat

4.2.1 Bilan

Dans ce projet de jeu vidéo réalisé avec *Unity*, nous avons mis en place un système de combat pour le joueur. Il permet au joueur d'attaquer des ennemis de manière dynamique, selon la direction dans laquelle il regarde (gauche, droite, haut ou bas). Ce système est conçu pour fonctionner dans un jeu multijoueur. Le joueur peut effectuer une attaque au corps-à-corps en appuyant sur une touche. Lorsqu'il attaque, plusieurs actions sont déclenchées :

- Le jeu détermine où se situe l'ennemi en fonction de la direction dans laquelle le joueur regarde.
- Une animation d'attaque est jouée pour rendre le mouvement visible.
- Un son de coup est joué pour améliorer l'immersion.
- Si un ennemi se trouve dans la zone d'attaque, il reçoit des dégâts, et le joueur peut gagner de l'argent en fonction des dégâts infligés.

Le jeu ajuste automatiquement la position du coup en fonction de la direction du joueur. Par exemple, si le joueur regarde à droite, l'attaque sera déclenchée un peu à droite de lui. Cela permet d'avoir un système plus réaliste où le joueur doit viser les ennemis avec précision.

Le joueur peut utiliser plusieurs armes, chacune avec ses propres caractéristiques :

Poing : attaque simple, sans arme équipée.
Bâton : portée moyenne, dégâts modérés.
Lance : longue portée, plus lente mais couvre une grande zone.
Épée : bon équilibre entre puissance, portée et rapidité.

Ces armes peuvent être changées en cours de partie via la boutique. Chaque arme a ses propres valeurs de dégâts et de vitesse d'attaque, ce qui permet au joueur d'adapter son style de jeu.

Lorsque le joueur attaque, le jeu vérifie si un ennemi est touché. Il le fait en imaginant une zone invisible devant le joueur. Si un ennemi se trouve dans cette zone, il est considéré comme touché. Cette zone peut être ronde (comme pour un coup de poing) ou rectangulaire (comme pour une attaque de lance). Cela permet d'avoir des coups différents selon l'arme utilisée.

Pour rendre le jeu plus vivant, chaque attaque déclenche des animations adaptées aux armes du joueurs ainsi que des effets sonores.

Cette gestion du combat permet une bonne diversité lors des différents combats du jeu.

4.2.2 Défis rencontrés

De nombreuses embûches se sont dressées pendant la création des mécaniques de combats, étant donné que ce n'est pas la même personne qui a construit les mécaniques de mouvements et celle de combat, il y a eu un travail de compréhension de code important, cela a induit une communication entre les membres de l'entreprise.

Suite à ça, nous avons pu nous concentrer sur les nouveautés à implémenter. Nous avons donc d'abord implémenté le plus basique des ennemis pour servir de test. Ensuite, nous avons pu créer des attaques sur les côtés. Nous nous sommes alors confrontés à un problème, celui d'empêcher que le joueur puisse frapper très rapidement, nous avons donc rajouté une caractéristique à l'arme : le nombre de coups par secondes autorisé. Cette caractéristique étant un attribut de la classe "weapon", cet attribut pourra être changé en fonction de l'arme.

Enfin, à part quelques bugs mineurs, surtout autour de l'implémentation des animations, le développement de cette partie du jeu s'est déroulé sans réel problème. Nous avons réussi à bien gérer notre calendrier pour ne pas prendre de retard sur cette partie.

4.3 Conception des assets

4.3.1 Bilan

La réalisation des assets est composée de 6 parties :

1. La réalisation des 3 ennemis, du joueur, des pnj et enfin du boss final et inspirations. 2. Le processus d'animations pour les différents elements vu précédemment, et de slicing pour les animations du player. 3. La création de plusieurs tilemap pour les différents niveaux ou salles (lvl 1, lvl 2, anthill, marchand) 4. La réalisation de tree breach pour les salles du niveau 1. 5. HUD/UI 6. Visuels du site web

1. Les deux ennemis implémentés dans le jeu représentent des membres déchus de la colonie, gangrénés par le champignon parasitant l'arbre. Ces deux ennemis sont à différents stade de l'évolution du parasite dans leur corp. Leur design reprennent l'esthetique de zombies présents dans plusieurs jeux vidéos comme The Last of us ou Resident Evil. le troisième ennemi, lui, n'a malheureusement pas été implémenté dans le jeu par faute de temps mais il représente une termite, qui devait être présente dans la cité des termites. Elle est munie de deux grandes épées et d'une paire d'ailes dans le dos qu'elle n'a pas la loisir d'utiliser car elle vie uniquement dans le bois creusé pas ses congénères.

Ces sbires du champignon sont combattu par le joueur, une fourmi de la colonie. Cette fourmi étais originellement vêtue de marron mais ces couleurs étant aussi utilisées dans d'autres elements du décors, un changement vers différentes teinte de vert fût adopté. Les ennemis, quant à eux, ont conservé les couleurs initiales, les rendant, plus sombres, ce qui colle avec l'image qu'il doivent véhiculer.

Les PNJ (personnages non joueurs), sont composés d'une nourrisse, d'un forgeron,

de la reine de la colonie et enfin d'un marchand.

La nourrisse représente une fourmi frêle, vêtue d'une robe jaune clair, robe inspirée de celle que porte le personnage Padmé Amidala dans *Star Wars* III. Cette nourrisse est la personnification de la faiblesse de la colonie face à la menace grandissante du champignon, mais aussi le fait de choisir de réaliser un personnage jeune comme nourrisse permet d'ouvrir le champ des possibilités sur un futur lore autour de ce personnage.

Le forgeron est une petite fourmi, avec une chemise bouffie blanche, des lunettes de protection sur le front, une salopette et des gants en cuir avec un marteau à sa ceinture. Ses vêtements rappellent directement le communisme des colonies de fourmi, ce forgeron étant le Stakhanov de la colonie. Ce personnage bombe le torse car il veut faire valoir ses qualités, malgré le fait que son rôle n'est pas sur le champ de bataille.

La Reine est imposante. Couverte sous d'inombrables étoffes, cela montre son autorité dans la colonie. Cependant, certains signes comme la longueur de ses antennes ou sa longue tête, montrent la longévité de ce personnage. Le design de la reine a été la première esquisse de ce que devait être le joueur, mais ce modèle était trop imposant pour être assez mobile et rendre le gameplay assez fluide.

Enfin, le marchand représente un boulier (l'insecte), transportant sa boule à laquelle est accrochée les différents articles que le joueur peut acheter. Il échange des armes et de la nourriture contre des petits os que le joueur peut trouver sur le cadavre des ennemis qu'il tue. Dans l'histoire, le marchand recherche ces os car ils sont très durs, et cela lui permet de renforcer le cœur de sa boule.

Le dernier personnage présent dans le jeu est le boss final de la première zone. Ce personnage, deux fois plus grand que le joueur, représente une amas de cadavres de fourmis gangrenées par le champignon. Plusieurs crânes de fourmis se trouvent tout le long de son corp et des champignons poussent sur son dos. Sa tête est le crâne d'une fourmi pendant à son buste.

Ce boss, à l'image des deux sbires du champignon, doit donner l'impression d'être désarticulé et complétement fou.

2. Les ennemis ont 5 animations différentes : idle, walk, hitted, attack et dead. Dans le cas des deux ennemis, les animations sont saccadées, comme pour si ils étaient en proie à des spasmes musculaires.

Les *PNJ* n'ont qu'une seule animation, celle de idle.

Le boss, lui, a différentes animations, uniques, répondant au différents besoins du gameplay.

Enfin le joueur avait, initialement, plusieurs animation qui ont du être toutes refaites dans l'optique d'implémenter plusieurs armes. En effet les premières animations n'avaient été pensées que pour l'usage d'une seule arme et cela rendait presque impossible l'ajout de différentes armes car cela aurait demandé de refaire, à chaque fois, toutes les animations du joueur. C'est pourquoi nous avons décidé de séparer les animations du joueur en 3 parties : "l'upper arm" qui est composé du bras droit et

de la tête, le “body” composé des jambes et du ventre, et enfin le “weapon”, composé du bras gauche et de l’arme.

Cette séparation a permis d’animer plus facilement les différentes armes, en effet, pour rajouter une arme, il ne manquait que de refaire les animations du weapon.

Les animations du joueur sont composées de : idle, walk, crouch, crouch walk, climb, dash, dash-attack.

Enfin, les différentes armes implémentées sont : les poings (ou fist en anglais), le bâton, l’épée et la lance. Le bâton et l’épée font partie des animations “small attack”, en effet cette catégories enveloppe toute les animations ayant comme arme une arme une-main.

Les mains font parties des animations “hand attack”, qui sont les animations utilisées lorsque le joueur se bat au poing ou alors avec des armes proches des poings comme les poings américains. Pour l’instant, une seule arme appartient à cette catégorie.

La spear appartient à la catégories des armes deux-mains, lourdes. Les animations changent car les armes deux-mains sont plus lourdes que la moyenne, nécessitant plus de force et de lenteur pour les manier. Ici aussi, qu’une seule arme fait parti de cette catégorie.

Malgré le fait que seulement 4 armes ont été implémentées, cela a permit d’implémenter les différentes catégories d’armes, rendant l’ajout de nouvelles armes très rapide, et si nous souhaitons continuer à travailler sur ce projet dans le futur, rajouter 10 armes dans le jeu serait réalisable.

3. Pour créer des rooms, nous avons besoin de différentes tilemap, ainsi 4 tilemaps ont été crées pour la la zone 1, la zone 2, la fourmilière et le marchand.

La tilemap de la zone 1 est composée de 3 variantes de murs : mur vierge, mur mousseux et mur champignonné. Ces murs représentent l’interieur de l’arbre. De plus, pour briser la monotonie du noir autour des rooms, des aspérités dans le bois ont été rajoutées. Cette tilemap est aussi composée d’une échelle et de sa variante en mousse/liane. Enfin, les éléments de décor sont 2 fourmis mortes, momifiées par le champignon et 10 champignons différents poussant sur différentes inclinaisons de surfaces. Le décor est autant important que le sol car il donne l’essence au jeu, et il permet de rendre les rooms plus uniques.

La tilemap de la zone 2 ne représente plus l’interieur d’un arbre sain mais celui d’un arbre rongé par les termites, c’est pourquoi les couleurs sont plus jaune, comme si l’arbre était entrain de mourir. Pour cette tilemap, 2 variantes de murs a été réalisée, une vierge et une champignonnée. De plus, une nouvelle variante de coin est présente sur la tilemap, et est utilisée lorsque nous avons des gouffres mortelles dans les rooms. Comme pour la première tilemap, il y a des aspérités dans le bois pour ne pas avoir des bords monocolore. Enfin des ponts avec 3 variantes de dégradation ont été réalisés et une échelle. Comme mentionné, il y a des gouffres mortels pour les joueurs, or les fourmis peuvent survivre à de grande chute, c’est pourquoi les termites ont mis au fond des trous des piques pour tuer les fourmis et les empêcher

de venir sur leur territoire. C'est pourquoi sur la tilemap il y a des piques que nous plaçons à chaque fois au fond des trous. Enfin, differents éléments de décors ont été rajoutés comme des cadavres de fourmis tué par les termites, des étendars avec la version abimée. Des grandes piques en bois, à laquelle les termites ont accroché des os et enfin des cranes sur le boût de piques, sur lesquels il y a des bougies. Pour l'armurerie des termites, il y a un tonneau avec les armes des termites en vrac dedans et un panneau de bois auquel sont accroché des éléments d'armures des termites.

La tilemap de la fourmilière comprends une version de murs, dérivée des murs de la tilemap de la zone 1, pour les décors, il y a plusieurs tas d'oeufs premier plan et arrière plan pour donner un effet de profondeur, un fourneau et des portes armures pour le forgeron et des torches et un trône pour la reine de la fourmilière.

La tilemap du marchand est uniquement composée de la boule du boulier. Cette boule est, comme décrite précédemment, recouverte d'etoffes, avec des armes plantées dedans et une ceinture à laquelle sont accrochés des flacons et autres nourritures achetables par le joueur.

4. Pour garder une logique dans le gameplay, des cassures dans l'écorce personnalisées pour chacune des salles du niveau 1 ont été réalisées. En effet, la fourmi parcourt l'interieur de l'arbre et ce n'est pas logique qu'elle voit l'exterieur de l'arbre sans problème. C'est pourquoi les trous dans les écorces apportent de la logique dans le gameplay. Pour la zone 2, les salles n'ont pas besoin d'équivalent car le fond représente l'interieur de l'arbre.

5. La partie artistique de l'HUD/UI est composée de plusieurs boutons et interfaces. Les interfaces avaient pour nécessité de ressembler à la coupe horizontale d'une bûche pour rester dans la même direction artistique tout le long du jeu.

Pour le vendeur, l'interface du menu a été reprise en ajoutant des visuels pour les produits disponibles à l'achat.

6. Pour le site web, 6 illustrations ont été utilisées pour illustrer différents aspects de notre jeu. Ces animations ont été faites sur mesure pour donner de la personnalité au site web et teaser notre jeu auprès des futurs clients que pourrait avoir le jeu.

4.3.2 Défis rencontrés

Lors de la conception des assets, de nombreux défis sont apparus.

Premièrement, les animations ont été longues à réaliser, car le mode par défaut de Pixel Studio ne permettait pas d'animer rapidement. Mais nous avons trouvé un mode plus avancé qui a permis d'optimiser la réalisation de ces animations, réduisant le temps passé dessus. Cependant, lorsque nous avons décidé de slice toutes les animations, cela a demandé de retravailler toutes les animations, ce qui prit un temps conséquent. De plus, pour un soucis de réalisme, plusieurs animations ont été changées plusieurs fois pour permettre d'avoir quelque chose qui colle avec la direction artistique du jeu.

Deuxièmement, les tiles de la zone 1 étaient trop sombres. En effet, sans assembler

les assets créés, nous n'avions pas remarqué que l'ensemble était très sombre et les couleurs peu diverses, tendant souvent vers un marron écorce trop terne qui ne donnait pas envie de jouer au jeu en plus de faire mal aux yeux à la longue. De plus, ce fût très difficile de trouver un façon de réaliser une façon pour représenter les galleries des termites, et le résultat finale n'était pas evident lors de sa réalisation.

Troisièmement, par défaut, *Unity* lisse les images pour que les pixels ne se voient pas. Cependant, dans notre cas, avoir de larges pixels est un choix. Il a fallu indiquer à *Unity* de changer la méthode de rendu de nos assets.

Enfin, il y avait un problème de fentes entre certaines tiles du niveau, qui apparaissaient lorsque le personnage bougeait. Pour régler ce problème, nous avons utilisé un *Sprite atlas*, permettant de rassembler nos assets pour optimiser les performances et régler des problèmes de calcul de position à l'éxecution. Ce *Sprite atlas*, lors de la réalisation des écorces, n'était pas pleinement fonctionnel et encore sur la dernière version du jeu, il nous arrive d'avoir des problèmes de *Sprite atlas* qui apparaissent comme ils disparaissent, sans raison apparente.

4.4 Conceptions des niveaux

4.4.1 Bilan

Suite à la création d'un tileset fixée pour la zone 1, la zone 2 et la fourmilière, les membres responsables de la création de niveaux, ont réalisé environ 15 salles pour le level1 et 5 salles pour le level 2 afin d'alimenter l'algorithme de génération procédural.

Pour créer ces salles, il a fallu respecter certains points :

Le joueur doit pouvoir parcourir la salle sans difficultés, en effet la course étant rapide et les sauts assez hauts, il ne faut pas qu'ils soient entravés par des plafonds trop bas ou des marches mal disposées par exemple. Mais malgré cette contrainte, les niveaux devaient rester organiques, pour rappeler au joueur qu'il parcourt un arbre.

De plus, les salles doivent être toutes différentes dans leur forme et dans leur parcourt pour que le jeu ne soit pas monotone, donnant une impression de découverte au joueur à chaque partie lancée.

Le level 1 a une entrée et une sortie, un shop, une salle de boss et 12 salles en plus. Le level 2 quant à lui a aussi une entrée une sortie; un shop et 3 rooms.

De plus, une salle sur mesure représentant la colonie a été implémentée comme lobby de départ du jeu.

4.4.2 Défis rencontrés

La réalisation de salles était plus longue que prévu, car il fallait réaliser une réelle réflexion préalable à la réalisation de chacune d'entre elles, réflexion renseignée dans la description de l'avancement de cette partie.

De plus, la taille moyenne des salles étant de 80 par 40 *tiles*, et chaque *tile* étant de

16 par 16 pixels, un nombre important de tiles devait être placé et mûrement réfléchi pour créer un niveau original et optimal.

Enfin, avec l'évolution du jeu, de notre réflexion et le développement de nos capacités, une partie des premières salles réalisées sont devenues obsolètes, car ne répondant pas aux nouveaux critères que nous nous sommes fixés, obligeant leur suppression du projet.

Pour les salles du niveau 2, plusieurs salles ont été réalisées mais le niveau 2 n'a pas pût être implémenté par manque de temps.

4.5 Multijoueur

4.5.1 Bilan

Pour l'implémentation du multijoueur, nous avons choisi d'utiliser une librairie externe nommée *Mirror*. Nous avons fait ce choix, car c'est un point sensible et complexe au cours du développement d'un jeu. Comme nous travaillons en temps limité, nous avons préféré concentrer nos efforts sur le gameplay directement en simplifiant le travail un maximum sur ce point. Mirror ajoute un nouvel composant *Unity*, le *NetworkManager*, et des nouvelles classes permettant la synchronisation des variables et des scripts exécutés entre le joueur et le serveur. Les mouvements du joueur, par exemple, ne sont plus définis comme un MonoBehavior mais un *NetworkBehavior*.

Malgré ce que nous avions annoncé dans le cahier des charges, le multijoueur ne fonctionne pas avec un serveur dédié. Un joueur doit lancé une partie qu'il peut décider de faire tout seul ou non et les autres joueurs peuvent le rejoindre. Une fois que tous les joueurs se sont déclarés "prêt", la partie est lancée grâce à notre système de *state machine* qui est responsable du déclenchement de la génération procédurale et du changement d'état pour tous les joueurs connectés.

Ensuite grâce à *Mirror*, la position, l'orientation et les animations des joueurs sont synchronisés, il en est de même pour les ennemis. Le serveur vérifie régulièrement l'état de chaque joueur pour changer l'état de la partie en conséquence. C'est comme cela que la partie est lancé lorsque tous les joueurs sont prêts ou encore que les joueurs sont téléportés au *lobby* lorsque tout le monde est mort. Lorsque tous les joueurs ont atteint la dernière salle du niveau, la *state machine* est chargé de lancé la génération du niveau suivant.

Notre système multijoueur est relativement complet et permet d'avoir une expérience multijoueur immersive et amusantes mais à cause de plusieurs évènements nous n'avons pas pu aller au bout de nos idées. Nous avions notamment prévu de donner un rôle unique à chaque joueur pour renforcer l'entraide au sein de l'équipe ainsi que la possibilité de communiquer.

4.5.2 Défis rencontrés

En intégrant le multijoueur à notre jeu, nous avons très vite remarqué que la position du client qui n'était pas l'hôte de la partie était remise à zéro dès que l'on essayait de bouger. Il s'avère que le problème venait du fonctionnement du multijoueur dans *Mirror*. En effet, il existe deux façons principales de penser le multijoueur dans un jeu. Dans la première manière de penser, le client a l'autorité, c'est-à-dire que le client se déplace et le serveur partage cette information au client. Dans la deuxième, le serveur doit valider les déplacements des clients et plus généralement toutes les données qui circulent entre les joueurs. Or, dans notre implémentation initiale, nous avions conçu le script de déplacement en local, avec l'idée que chaque client contrôlerait librement son propre personnage. En réalité, seuls les mouvements du joueur hébergeur étaient pris en compte, les autres étant automatiquement corrigés (et annulés) par le serveur. Pour pallier ce problème, nous avons donc dû revoir la manière dont les déplacements du joueur étaient gérés et interagissaient avec *Mirror*. La façon la plus simple était de déléguer l'autorité des objets d'un joueur au client qui les contrôlait.

Nous avons aussi eu des difficultés pour passer d'un jeu qui marchait localement à un jeu avec des clients synchronisés. Tout d'abord, nous avons changé d'outil pour synchroniser les différents clients Unity. Nous avons dû gérer à la fois le mode solo qui fonctionne avec un serveur local et le mode multijoueur que nous avons choisi d'architecturer autour d'un serveur distant. Nous avons fait le choix d'utiliser une *state machine*[15] pour gérer l'état d'une partie, faire apparaître les niveaux, déclencher la génération procédurale avec la même graine pour tous les clients et téléporter les joueurs. Apprendre à utiliser les différents composants de Mirror, notre librairie de réseau, pour s'envoyer de telles informations a été un des principaux problèmes. En effet, si la librairie semble simple à prendre en main, dès lors qu'il faut synchroniser des états plus complexes, il faut bien comprendre tous les outils à notre disposition. De plus, la mécanique de client "hébergeur" qui a à la fois un statut de client et de serveur rajoute de la complexité à gérer.

Enfin à cause de notre organisation dans laquelle nous avons choisi de nommer une personne en charge du multijoueur, les fonctionnalités du jeu étaient d'abord pensées et programmées sans penser à leur implémentation en multijoueur ce qui rendait souvent la tâche plus compliquée par la suite et donnait lieu à de nombreux problèmes de synchronisation. Nous avons donc dû, après avoir développé les mouvements du joueur, retravailler le script pour l'adapter au multijoueur. Cela aussi a été le cas pour les ennemis, et de manière plus générale, pour le système de combat.

La partie la plus compliquée à synchroniser et qui peut encore mal fonctionner est la synchronisation des ennemis. En effet, il fallait s'assurer que leur vie était identique entre tous les clients, mais aussi le joueur qu'il cible, leur déplacement, leur animation, etc. Les déplacements ont été particulièrement compliqués à faire correspondre d'un client à l'autre car l'algorithme de *Path Finding* tournait en local pour chaque joueur et non pas sur un serveur ce qui a compliqué la tâche.

Malheureusement le manque de temps et nos problèmes d'organisation nous ont empêchés de revoir plus profondément l'architecture de notre multijoueur en le ren-

dant plus centralisé autour du serveur que des clients ce qui malgré le fait que cela aurait été un peu plus contraignant en termes de développement cela aurait permis d'avoir un système multijoueur plus stable et fonctionnel.

4.6 Intelligence artificielles

4.6.1 Bilan

L'intelligence artificielle (*IA*) occupe une place centrale dans notre jeu, notamment à travers les comportements des ennemis contrôlés par le système.

L'ennemi ne poursuit pas le joueur en permanence. Il commence à le faire uniquement s'il le détecte dans un certain rayon autour de lui. Cette détection se fait à l'aide d'un système basé sur une bulle invisible autour de l'ennemi, qui vérifie régulièrement si un joueur est entré dans la zone. Si c'est le cas, l'ennemi entre en mode actif et commence à réagir.

Une fois qu'un joueur est détecté, l'*IA* utilise un système de pathfinding pour calculer un chemin jusqu'au joueur et le suivre. Cela permet à l'ennemi de contourner les obstacles et de ne pas se bloquer contre les murs. Il avance automatiquement dans la direction du joueur, avec une limite de vitesse pour rendre les mouvements naturels et maîtrisés.

Pour éviter que l'ennemi ne reste coincé devant un obstacle, un système de saut intelligent a été ajouté. Si le joueur est situé bien plus loin en hauteur et que l'ennemi est à l'arrêt, il déclenche un saut pour tenter de reprendre la poursuite.

L'IA peut aussi attaquer. Lorsqu'un joueur entre dans une zone d'attaque proche de l'ennemi, celui-ci lance une attaque automatiquement. Cette attaque inflige des dégâts au joueur. Il y a un délai entre deux attaques pour éviter que l'ennemi ne frappe en continu, ce qui rend le combat plus juste.

L'ennemi possède un système de points de vie. Lorsqu'il est touché par le joueur, ses points de vie diminuent. S'ils tombent à zéro, l'ennemi meurt. Lorsqu'il prend un coup, une animation est jouée et un son est émis pour renforcer le retour visuel et sonore. À sa mort, une autre animation est lancée laissant pour seul trace son corps inerte.

Chaque action importante de l'*IA* (attaque, mort) est accompagnée de sons spécifiques. Cela permet de rendre les combats plus immersifs.

Le système est conçu pour un jeu multijoueur. Ainsi, toutes les actions de l'*IA* (dommages, animations, mort...) sont synchronisées entre les joueurs grâce à des mécanismes de réseau. Cela garantit que tous les participants voient les mêmes événements en temps réel.

Ce système d'*IA* permet de créer des ennemis crédibles, capables de réagir intelligemment à l'environnement et au joueur. Il offre une base solide pour enrichir les mécaniques de jeu et rendre les combats plus dynamiques.

L'ajout d'un boss dans le jeu a représenté une étape importante du développement, à la fois sur le plan technique et en termes de game design. Ce combat devait proposer un réel challenge.

Le boss a été structuré autour d'un enchaînement de phases. Cette organisation permet de rythmer le combat. Le déroulement suit une alternance entre des moments d'attaque intense et des phases de vulnérabilité, durant lesquelles le joueur peut infliger des dégâts.

La première phase d'attaque prend la forme d'une pluie de rochers. Ceux-ci tombent de manière aléatoire sur l'aire de combat, forçant le joueur à se déplacer constamment pour éviter les zones d'impact. Une fois cette attaque terminée, une première phase de vulnérabilité s'ouvre, pendant laquelle le boss cesse toute action et devient sensible aux attaques.

Le combat enchaîne ensuite sur une seconde mécanique offensive : un lancer de pierres directement ciblé vers le joueur. Cette attaque est plus précise et demande une bonne lecture des positions pour les esquiver verticalement. Elle est suivie d'une nouvelle période de vulnérabilité.

Dans la phase suivante, les deux attaques précédentes sont combinées. Le joueur doit ainsi gérer simultanément la pluie de rochers aléatoire et les projectiles ciblés. Cette combinaison augmente fortement la difficulté. Elle est suivie d'une dernière phase de vulnérabilité, plus longue, offrant une dernière opportunité au joueur d'achever le combat.

Ce combat de boss permet de conclure une séquence de jeu de manière intense et remarquable.

4.6.2 Défis rencontrés

L'intégration de l'intelligence artificielle dans un environnement multijoueur a présenté plusieurs défis techniques importants. Le premier concernait la mise en œuvre du pathfinding. L'algorithme de navigation devait permettre à l'ennemi de suivre le joueur tout en contournant correctement les obstacles du décor. Cependant, cet algorithme s'exécutant localement sur chaque client, il entraînait des comportements incohérents entre les machines, notamment des déplacements non synchronisés ou des ennemis qui se retrouvaient bloqués différemment selon le client. Pour résoudre ce problème, nous avons dû recentraliser le calcul du chemin sur le serveur et synchroniser précisément les actions (mouvements, attaques, réactions) de l'*IA* auprès de tous les clients via le système de Mirror.

Un autre point délicat a été d'éviter que les ennemis ne se coincent dans le décor. Le pathfinding ne suffisait pas toujours à gérer les différences de hauteur ou les zones étroites. Nous avons donc ajouté un système de détection de blocage et une logique de saut déclenchée dans certaines conditions (par exemple, si l'ennemi reste immobile devant un obstacle et que la cible est en hauteur). Cela a demandé plusieurs ajustements pour éviter les sauts aléatoires ou incohérents.

Le multijoueur, lui aussi, a apporté sa part de complexité. Comme pour les joueurs, il a fallu choisir quelle instance (client ou serveur) contrôlait les ennemis. Nous avons opté pour une approche centralisée où seul le serveur pilote les *IA*, ce qui a permis de garantir une cohérence de leur comportement sur tous les clients. Cela a impliqué de nombreuses modifications dans notre architecture initialement pensée pour du local, notamment pour gérer les points de vie, les animations et les effets sonores des ennemis, tout en assurant leur synchronisation réseau.

Enfin, la gestion des phases du boss a nécessité une structure claire et robuste. Le système de phases devait à la fois permettre une alternance fluide entre attaque et vulnérabilité, et rester compréhensible pour le joueur tout en étant piloté correctement par le serveur. L'ajout de transitions conditionnelles (par exemple, attendre la fin d'une animation ou le déclenchement d'un timer) a parfois généré des bugs de synchronisation entre clients.

4.7 Génération procédurale

4.7.1 Bilan

Dans notre jeu, nous avons mis en place un système de génération procédurale afin de garantir une grande rejouabilité et de créer des environnements dynamiques à chaque nouvelle partie.

Ce système repose sur un plan directeur, appelé blueprint, qui définit la structure globale du niveau sous la forme d'une suite de types de salles à assembler. À partir d'une graine aléatoire (*seed*), qui permet de reproduire un niveau identique si nécessaire, le système sélectionne dans une bibliothèque de salles les éléments correspondant au blueprint, puis les positionne intelligemment dans l'espace de jeu en s'assurant que l'ensemble reste cohérent et accessible.

Une fois le niveau construit, les salles sont instanciées et placées dans la scène, avec leurs volumes, leurs formes et leurs entrées correctement ajustées. Chaque salle contient des zones spécifiques prévues pour accueillir des ennemis. Le système va alors parcourir ces points de spawn et y placer aléatoirement un des deux types d'ennemis disponibles, en suivant une probabilité définie à l'avance. Cette génération d'ennemis est entièrement intégrée au mode multijoueur : tous les ennemis sont synchronisés entre les clients grâce à la librairie Mirror, ce qui permet à tous les joueurs de voir les mêmes ennemis, au même endroit, avec les mêmes comportements. Cela garantit une expérience de jeu fluide, équitable et cohérente en ligne.

Le système est également conçu pour définir automatiquement des points d'apparition pour les joueurs. Ces spawn points sont choisis de manière à permettre un départ équilibré, avec un bon placement par rapport aux premières zones à explorer. De plus, les limites du niveau sont calculées dynamiquement en fonction de la disposition des salles, ce qui permet de bien cadrer la zone de jeu et d'ajuster éventuellement des éléments comme la caméra ou les effets visuels.

En somme, cette approche procédurale offre une solution efficace et flexible pour

construire des niveaux variés. Elle nous permet de proposer une expérience renouvelée à chaque session, sans compromettre la qualité du gameplay ou la stabilité en multijoueur.

4.7.2 Défis rencontrés

L'une des premières difficultés que nous avons rencontrées lors de la programmation du *PoC* pour la génération procédurale était de s'assurer de la validité des niveaux produits. L'un des premiers moyens a été de montrer toutes les informations de chaque pièce dans le graphe et de vérifier à la main. Cette méthode marche au début, mais devient limitante dès que l'on essaye de faire un niveau de plus de trois salles.

Pour générer une représentation graphique, nous avons d'abord utilisé une bibliothèque graphique pour faire une image pixels par pixels. Par défaut, tous les lecteurs d'image lissent les pixels, ce qui impacte la visibilité du rendu. Après un certain temps passé à essayer de contourner le problème, nous nous sommes tournés vers le format SVG, bien plus souple.

4.8 Musiques et sons

4.8.1 Bilan

La création de l'univers sonore du jeu a été pensée comme un prolongement direct de l'expérience de jeu. Elle vise à renforcer l'immersion, à souligner les émotions des différentes séquences et à donner vie à notre univers. Cela passe à la fois par un travail approfondi sur la musique d'ambiance, les musiques dynamiques comme celle des combats de boss, et l'intégration précise d'effets sonores adaptés à chaque situation de gameplay.

La création de la musique principale a nécessité plusieurs essais pour parvenir à une ambiance parfaitement cohérente avec l'univers du jeu. Une première version a été composée plus tôt dans le développement, en se basant sur des intentions initiales autour de l'exploration et de la découverte. Cependant, cette version ne retranscrivait pas fidèlement l'atmosphère recherchée. Elle manquait d'émotion, de texture et surtout d'un lien avec l'identité visuelle et narrative du projet. L'ambiance musicale semblait trop neutre, voire déconnectée de l'environnement du jeu, ce qui pouvait nuire à l'immersion du joueur.

Ce constat a conduit à une réévaluation des choix musicaux. Une seconde version a alors été enregistrée avec une approche différente. L'accent a été mis sur un son plus riche et un rythme plus en phase avec le ton du jeu. Cette nouvelle piste s'intégrait plus naturellement dans le gameplay, renforçant l'aspect exploration du jeu. Elle a également été pensée dès sa conception pour permettre une transition fluide vers une musique plus dynamique en cas de changement de situation, comme l'arrivée d'un combat de boss. L'enjeu était d'éviter une rupture auditive qui casserait l'immersion, et de proposer au contraire une montée en puissance naturelle.

Pour garantir cette fluidité sonore, la musique de boss a été construite en conser-

vant les mêmes accords fondamentaux que la musique d'ambiance, ce qui assure une cohérence harmonique. Cependant, afin d'apporter une intensité dramatique propre à ces moments clés, la structure musicale a été profondément modifiée. Le rythme a été accéléré, les motifs mélodiques plus affirmés, et des éléments rythmiques ont été introduits. L'ajout de percussions puissantes, notamment des tambours profonds et résonnants, vient renforcer le sentiment de tension et de danger.

Ce travail permet une transition fluide : la musique évolue naturellement, sans coupure ni changement brutal, créant un effet de montée en intensité progressive. Le joueur perçoit ainsi que la situation évolue, mais reste immergé dans un univers sonore familier. Ce choix accentue l'importance du combat que le joueur s'apprête à réaliser.

En parallèle de la composition musicale, un soin tout particulier a été apporté à la conception et à l'intégration des effets sonores. Ces éléments, souvent discrets mais essentiels, jouent un rôle majeur dans la perception du jeu. Ils renforcent la sensation de réactivité, améliorent la lisibilité des actions, et participent pleinement à l'ambiance générale.

La première étape a consisté à identifier les besoins précis en matière de sound design. Les effets ont été classés selon leur fonction : interface utilisateur, déplacements, interactions environnementales, et combats. Pour chaque catégorie, des sons adaptés ont été recherchés, testés et modifiés si nécessaire. Les sons liés à l'interface permettent au joueur de naviguer dans les menus et options sans effort, tout en recevant un retour clair et immédiat à chaque action.

Du côté des déplacements, chaque action du joueur a reçu un traitement sonore spécifique. Les bruits de pas, les sauts, dash et phases de grimpe bénéficient de sons distincts, renforçant la lisibilité des mouvements. Ce travail de détail permet au joueur de mieux sentir son personnage et d'anticiper ses actions, ce qui améliore l'expérience globale du jeu. Enfin, les effets sonores liés au combat ont été pensés pour donner de la force et du poids aux impacts. Un menu pour modifier les sons permet d'adapter le volume dynamiquement pour s'intégrer naturellement à l'ambiance sans couvrir la musique ni les autres éléments du jeu selon les désirs du joueur.

4.8.2 Défis rencontrés

La création de l'univers sonore du jeu a présenté plusieurs défis, tant sur le plan technique que sur le plan de l'intégration dans Unity. Du côté de la musique, le principal obstacle a été de produire une ambiance cohérente avec l'univers du jeu. Une première version de la musique d'ambiance avait été composée mais ne reflétait pas suffisamment l'identité visuelle et narrative du projet. Une nouvelle version a alors été enregistrée, avec un ton plus immersif.

Pour accompagner les phases de combat, une version alternative a été conçue à partir des mêmes accords de base que la musique principale. Ce choix harmonique a permis une transition fluide entre les deux morceaux sans effort technique particulier, car l'oreille du joueur reste guidée par des repères musicaux constants, évitant ainsi

toute dissonance.

En revanche, l'intégration des effets sonores a soulevé de nombreuses difficultés. L'un des premiers défis a été de lier correctement les effets au système de menus. Il fallait que les niveaux de volume définis par le joueur via les menus (musique, effets, master) soient correctement appliqués à chaque scène. Or, il arrivait que certains sons ne tiennent pas compte de ces paramètres au chargement d'un nouveau niveau, ou qu'ils soient joués au volume par défaut. Un autre problème récurrent concernait l'équilibrage du volume entre les différents effets sonores. Certains sons de combat étaient trop forts comparés à d'autres sons. Il a donc fallu ajuster finement les niveaux sonores pour assurer une bonne lisibilité de l'ensemble sans nuire à l'ambiance globale.

Le timing de déclenchement des effets représentait également un point critique. Il était essentiel que chaque son se joue précisément au moment où l'action a lieu, sans délai ni décalage, afin de renforcer la sensation de réactivité. Cela nécessitait une bonne synchronisation avec les animations, les collisions et les triggers d'événements.

Une difficulté particulière est apparue lors des transitions entre scènes : certains sons ne se jouaient pas correctement, voire étaient complètement ignorés si la transition avait lieu au même moment qu'un événement sonore. Ce bug s'expliquait par la destruction ou le rechargement du système audio avant que le son n'ait eu le temps de se déclencher. Pour corriger cela, des sons ont été relocalisés dans des objets persistants ou des appels différés ont été utilisés, afin de garantir qu'ils puissent se jouer correctement même lors d'un changement de scène.

4.9 Site internet

4.9.1 Bilan

Nous avons décidé de concevoir notre site internet avec la même palette de couleur que notre jeu pour garder l'ambiance et rester cohérent dans notre charte graphique et refléter au mieux l'esprit de notre jeu.

Le but premier du site internet est de servir de vitrine à notre jeu et surtout de permettre son téléchargement Cependant, nous avons aussi mis à disposition tous les documents relatifs au projet du cachier des charges au rapport de projet. De même nous avons mis les liens et les sources de toutes les ressources et applications utlisées pour réaliser *Treep*. Il est aussi possible d'y retrouver une présentation de notre entreprise *Kromm Studio* et de ses membres.

Lors de sa conception nous avions un seul objectif, le garder simple, minimaliste et épuré. Nous ne voulions pas nous lancer dans la création d'un site internet complexe avec beaucoup d'effets visuels et de fonctionnalités.

Pour ce qui est de la partie technique, nous avons utilisé les languages *HTML*, *CSS* et *Javascript* pour arriver à ce résultat.

4.9.2 Défis rencontrés

La conception du site internet n'a pas posé de défis majeurs. Le seul point plus difficile a été de maîtriser correctement le *CSS* pour trouver les bons accords de couleurs, les bonnes dimensions et les bonnes formes pour les différents éléments du site.

Nous avions commencé à réfléchir et à travailler sur un site internet auto adaptatif, c'est à dire visualisable sur tout type d'écran mais nous n'avons pas trouvé le temps de parfaire cela à cause du manque de temps, de la longueur et la complexité de la tâche.

4.10 Autres fonctionnalités

4.10.1 Bilan

Notre jeu est aussi composé d'autres fonctionnalités essentielles qui n'entraient dans aucune catégorie.

Interface utilisateurs

Les interfaces utilisateurs étant la première interaction que le joueur à avec le jeu, nous avons essayé de les faire aussi intuitives et attirantes possible pour l'utilisateur. Pour ce faire nous avons animé certaines parties des menus, notamment le menu principal du jeu. Nous avons aussi limité au maximum le nombre de boutons et d'éléments affichés pour ne pas trop chargé l'écran et perdre le joueur.

Ce dernier a donc la possibilité de changer les contrôles du jeu ainsi que le volume de la musique et des effets sonores. Cela permet au joueur de personnaliser son expérience et de faire correspondre à ses préférences. Les paramètres du jeu sont aussi changeable en cours de partie.

Le joueur a aussi le choix de rejoindre la partie de son choix en renseignant l'ip et le port dun serveur de jeu qu'il veut rejoindre.

Fog of War

Le *fog of war* ou brouillard de combat permet d'ajouter de la difficulté au jeu ainsi qu'une capacité pouvant être amélioré. Le brouillard de combat désigne la zone de flou noir qui entoure le joueur et masque sa vision l'empêchant de voir les ennemis et autres obstacles de loin. Pour arriver à ce résultat nous avons superposer plusieurs layers de textures, un auquel nous avons appliqué un shader noir et fou, l'autre sur lequel nous avons mis un *mask* démasquant certains sprites configurés pour agir de cette façon.

Magasin de capacités

Dans le *lobby*, la salle où les joueurs apparaissent avant d'entrer dans le niveau, les joueurs peuvent parler avec un marchand itinérant pour acheter de meilleures armes ainsi que des améliorations pouvant l'aider tout au long de son aventure. Le joueur a donc la possibilité d'acheter trois nouvelles armes et quatres nouvelles capacités pour améliorer ses dégats, sa vision, sa propulsion et sa vie. Le magasin permet aussi au joueur d'avoir un aperçu de ses statistiques qu'il ne peut voir que à cet endroit. Le joueur doit néanmoins céder des restes de fourmis gangrénées en échange de ces améliorations. La création de ce magasin nous a forcé a implémenter plusieurs statistisques en plus sur le joueur mais aussi de nouveaux menus.

Cache de l'arrière-plan

La génération procédurale place les pièces mais ne s'occupe pas de remplir les bords. Cela laisse donc des trous qui montrent l'arrière-plan. Pour y remédier, on utilise la même technique que pour le fog of war. On place un grand cache de la couleur de notre choix

Le parallax

Pour donner un effet de profondeur aux salles, un effet de parallax ont été ajouté pour la zone 1 et 2. Le parallax consiste en la technique de faire bouger les différentes couches de paysage à des vitesses différentes en fonction de la distance que nous voulons faire ressentir.

Le parallaxe de la zone 1 utilise 10 couches différentes pour donner une réelle impression de profondeur. Ces couches forment un paysage sylvestriel, entouré de montagnes. Les différentes couches d'arbre permettent d'utiliser pleinement le script de parallax.

Le parallaxe de la zone 2 utilise 5 couches différentes et représente les entrailles de l'abre, creusé par les termites. Si on fait attention, on peut voir des ponts entre les parois et des lumières au loin, donnant presque une sensation de vertige devant l'immensité de l'arbre. La verticalité de l'arbre est bien représentée contrairement à dans la zone 1 où le terrain était assez plat.

Pour s'assurer qu'il y ait toujours un fond présent derrière les rooms, le script de parallax s'occupe de répéter indéfiniment les couches.

Git et Github

Pour nous aider dans notre travail nous avons essayé d'exploité au maximum les fonctionnalités mises à dispostions par *Git* et *Github*.

Nous avons donc mis en place une *C.I Github* qui compile le jeu à chaque modification du code sur la branche principale. Cela nous permet du temps lorsque nous devons build le jeu mais ajoute aussi une vérification supplémentaire nous indiquant si oui ou non le jeu fonctionne. De plus, nous avons réussi à faire en sorte que les binaires produit par la *C.I* soit automatiquement rendu disponible au téléchargement sur le site internet.

Nous avons utilisé un autre service de *Github* pour héberger notre site internet qui est automatiquement mis à jour et republier à chaque modification du code de ce dernier. Nous avons lié le domaine fourni par *Github* à notre propre nom de domaine.

4.10.2 Défis rencontrés

Interface utilisateurs

La mise en place des éléments visuels des interfaces utilisateurs n'a pas posé de problèmes, ce qui a pu se montrer plus complexe est la mise en place de la logique que les divers boutons et *sliders* doivent exécutés. Pour donner la possibilité au joueur de changer ses touches il a fallu prendre en main le nouveau système d'entrées de *Unity* ce qui a par la suite impliqué de changer la façon dont on gérait le contrôle de tous nos déplacements.

Pour les réglages sonores, nous avons du effectuer plusieurs conversion pour passer d'un pourcentage allant de zéro à cent à une valeur comprise entre moins quatre vingt et vingt décibels qui est la plage sonore imposé par *Unity*.

Fog of War

La mise en place du *fog of war* a n'a pas posé de problème majeure mais cependant elle a nécessité de comprendre la mise en place et le fonctionnement des *sprites renderer* et des *sprites masks* et comment ils peuvent intéragir entre eux. Il a fallu aussi prendre en main la création de *shader* dans Unity, comment leur donné des paramètres pour les contrôler en temps réel pendant la partie.

Au début, notre masque cachait les effets du *shader* ce qui créait une zone ou aucun effet ne fonctionnait rendant l'effet complétement non fonctionnel. Il a fallu se renseigner d'avantage sur les *layers* et leur fonctionnement pour que chaque composant applique sont effet au bon endroit.

Magasin de capacités

Pour ce qui est de la conception du magasin de capacité, le design des menus n'a pas posé de problèmes. Cependant les interactions entre le *PNJ* et le joueur ont été plus compliqué à mettre en place. Au début de sa conception, nous n'arrivions pas à configurer corrrectement les colliders pour que le joueur puisse intéragir avec le *PNJ*. Pour finir, il a aussi fallu créér des fonctions spécifiques pour mettre à jour les statististiques du joueur.

Cache de l'arrière-plan

Les défis rencontrés pour la mise en place de cette fonctionnalités sont les mêmes que pour le *fog of war* soit la gestion des layers et des masques et leur interaction les uns avec les autres.

Le parallax

Les couches pour le *parallaxe* de la zone 1 ont beaucoup changé, ce qui a amené beaucoup de problèmes de mauvaise configurations de ces couches. De plus, pour faire les tree breach le logiciel pixel studio est utilisé, or la taille maximum de feuille est de 1024 par 1024 *pixels*. Or certaines rooms nécessitent des trous dans l'écorce de 2 à 3 fois la taille de la feuille maximale. Il a ainsi fallu séparer et faire en plusieurs fois les trous dans l'écorce.

5 Récit de la réalisation

Dans l'ensemble nous sommes satisfait par l'expérience de créer un projet de A à Z sur un an. Nous avons tous individuellement et collectivement appris beaucoup sur de nombreux aspects.

Premièrement d'un point de vue technique, nous avons commencé l'année sans connaître ni *Unity* ni *C-Sharp* et nous sommes maintenant en Juin avec un jeu jouable et presque amusant ou l'on pourrait lancer et jouer ensemble comme nous le faisons avec ceux des grands studios de jeux vidéos. Nous avons aussi appris à ce servir d'outils essentiel dans l'industrie de l'informatique comme *Git* et *Github* qui sera sans doute une compétence dont nous allons nous reservir plus tard.

D'un point de vue organisation nous avons pu avoir un premier aperçu de ce que la gestion de projet demande, les documents à rédiger comme le cahier des charges de début de projet mais aussi l'organisation et la rigueur qu'il faut avoir. Nous pensons que c'est ce point le plus important et celui sur lequel nous avons le plus appris et le plus à retenir de cette expérience. A plusieurs reprises dans ce rapport nous mentionnons notre manque d'organisation, notamment le manque de régularité ce qui nous a empêché d'accomplir tout ce que nous avions en tête.

En plus de ces aspects, ce projet nous a permis de découvrir plusieurs réalités concrètes du développement de jeu vidéo. Nous avons par exemple pris conscience de l'importance du game design, non seulement dans les mécaniques mais aussi dans le ressenti global du joueur. Concevoir un jeu amusant et équilibré demande un travail d'observation, de test et d'ajustement que nous n'avions pas anticipé. Nous avons également appris à travailler en équipe sur un même code et à gérer la répartition des tâches en fonction des compétences et disponibilités de chacun. La coordination, la communication et les réunions régulières se sont révélées être des points essentiels. D'un point de vue plus technique, nous avons été confrontés à des bugs complexes résultant de l'interaction entre différentes briques du jeu (*IA*, interface, multijoueur, physique...), et nous avons progressé dans notre capacité à enquêter, tester et corriger. Nous avons aussi compris l'utilité d'une documentation claire : bien documenter le code, les systèmes, les étapes de développement ou les processus de test est un gain de temps considérable. Enfin, nous avons vu les limites du prototypage rapide, souvent utile pour tester une idée, mais qui, sans cadre structuré, peut rendre le projet difficile à maintenir.

Pour conclure, ce projet nous aura beaucoup appris sur de nombreux points et nous sommes globalement fier de ce que nous avons accomplie mais avec une touche de déception et de frustration de ne pas avoir travaillé plus régulièrement.

6 Gestion du projet

6.1 Répartition et avancement du travail

	Répartition					Avancement		
Soutenance	Mahé	Oscar	Kristen	Romain	Milo	Janvier	Mars	Mai
Programmation								
Génération procédurale			S		R	90%	100%	100%
Réseau multijoueur			R		S	50%	100%	100%
Intelligence artificielle		R		S		0%	50%	100%
Mécaniques de base (mouvements, etc.)				R	S	50%	75%	100%
Mécaniques de combat	S	R				25%	50%	100%
Interface utilisateur (menus, HUD, etc.)	R	S				50%	100%	100%
Game design								
Histoire/Lore		R		S		90%	100%	100%
Conception des niveaux			S		R	40%	75%	100%
Conception graphique	S			R		25%	75%	100%
Conception sonore		S	R			10%	50%	100%
Musique	R			S		10%	50%	100%
Site web								
Conception du site web	S		R			90%	100%	100%

Table 1: R correspond au rôle de Responsable et S correspond au rôle de Suppléant

Pour ce qui est de l'organisation de notre flu de travail, nous avons créé une *monorepo* sur *Github* contenant le code de notre jeu, de notre site, de nos PoC ainsi que les assets graphiques et les documents liés au projet.

Nous ajoutions du code au jeu grâce à des *pull request*, chaque *PR* associé à une fonctionnalité. Les autres membres du groupe pouvait ensuite aller sur la branche associé, tester le code et approuver ou non les changements.

6.2 Gestion des coûts

Le développement de notre jeu repose principalement sur des outils gratuits ou bénéficiant de licences étudiantes (Unity, Rider, GitHub). Cela nous permet de limiter fortement les dépenses logicielles.

Le poste principal de dépense concerne les salaires. Nous avons estimé un coût de 2 500 € par mois et par personne, pour une équipe de 5 personnes pendant 8 mois, soit un total de 100 000 €. Ce montant inclut les développeurs, graphistes et responsables techniques.

À cela pourraient s'ajouter des frais annexes comme la communication (campagnes promotionnelles, site vitrine professionnel), l'hébergement avancé ou la publication sur des plateformes (Steam), ainsi que des frais juridiques pour la protection du projet (dépôt de la marque, CGU, etc.). En anticipant ces dépenses, nous pouvons estimer un budget élargi à environ 100 000 à 110 000 € au total.

Besoins	Moyens utilisés	Coût
Moteur de jeu	Unity 3D (Gratuit)	0 €
Éditeur de code	Rider (License étudiante)	0 €
Hébergement du code	GitHub (Gratuit)	0 €
Hébergement du site	GitHub Pages (Gratuit)	0 €
Nom de domaine	OVH	10 €
Design des assets	Pixel Studio (Gratuit)	0 €
Salaires	2500 €/mois pour 5 personne	100000€
Eventuel frais de publication	Steam	100€
Frais juridique	Marque, CGU	500€
Communication/Marketing	Réseaux sociaux, visuels, campagnes	1500 €
Total		Environ 102 000€

6.3 Diagramme de Gantt

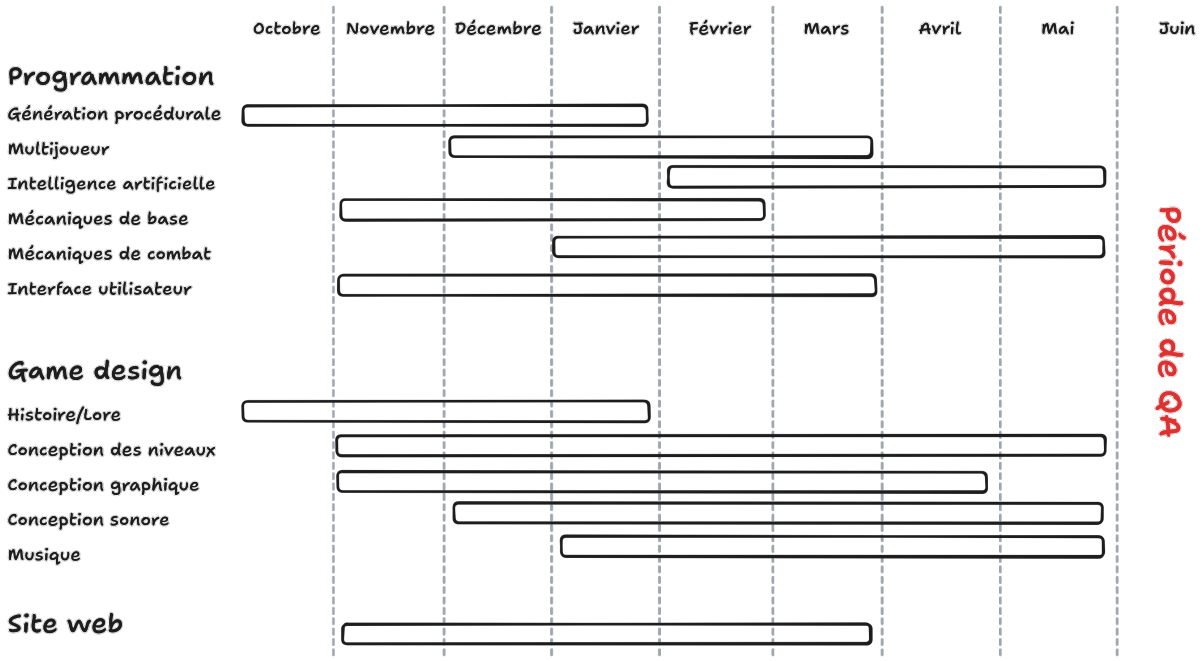

Figure 7: Diagramme de Gantt pour la répartition des tâches dans le temps

7 Annexes

7.1 Termes techniques

Définitions de tous les termes techniques et anglicismes utilisés dans ce cahier des charges :

Unity : Moteur de jeu largement utilisé pour créer des jeux vidéo.

C# : Principal langage de programmation utilisé pour le développement de jeux vidéo sur Unity.

Level design : Art de concevoir des niveaux (level en anglais), pour permettre au joueur d'avoir une expérience fluide tout au long du jeu.

Art books : Livre répertoriant le processus de création d'un artiste centré autour d'un projet (jeu, personnage, film, etc.).

IA : L'intelligence artificielle, dans le domaine du jeu vidéo, correspond au comportement (attaque, fuite, etc.) des personnages non-joueurs (ennemis, marchands, etc.) présents dans le jeu.

HUD : Le Heads-Up Display, littéralement affichage tête haute, désigne tous les éléments affichés en permanence sur l'écran d'un joueur (vie, endurance, munitions, etc.).

Lore : Correspond au cadre narratif établi d'un jeu. Il influe sur toute la partie artistique du jeu : son histoire, ses personnages, son environnement, ses musiques, etc.

PNJ : Un personnage non-joueur est un personnage entièrement contrôlé par une IA. Il peut avoir plusieurs rôles comme celui d'un ennemi, de vendre des bonus, de donner des informations, etc.

Testing/QA : Étape dans la création d'un jeu vidéo où le jeu est testé, poussé à ses limites, pour faire ressortir d'éventuels bugs, problèmes lors de la progression ou incohérence narrative.

Assets : Ensemble des éléments graphiques du jeu vidéo, regroupant sprites 2D, modèles 3D, animations, textures, musiques et effets sonores.

Netcode : Ensemble de protocoles permettant la synchronisation de différents composants tels que les joueurs et leurs actions au sein d'un jeu en ligne.

Run : Désigne la partie d'un joueur sur un jeu, du lancement de la partie, à la mort ou à la victoire du joueur. Ce terme est beaucoup utilisé dans les jeux du genre Rogue-like ou le joueur a tendance à relancer beaucoup de parties.

Game design : processus de création et de mise au point des règles et autres éléments constitutifs d'un jeu.

Playtest : Phases de test faites par une petite partie de joueurs désigné par le studio, ayant pour but de tester le jeu, de trouver les failles et les incohérences au

niveau du gameplay et de l'histoire.

Rogue Like : Sous genre du jeu vidéo, inspiré du jeu Rogue.

Dash : Action très présente dans les jeux vidéos, consistant à faire bondir le joueur vers l'avant.

Input : Ensembles des entrées claviers et/ou souris du joueur, guidant à une action.

UI : L'UI pour User Interface, désigne toutes les parties visuelles qui ne font pas partie directement du jeu : inventaire, menu, barre de vie...

Script : Les scripts désignent toutes les parties "techniques" permettant au jeu de fonctionner. Ces scripts sont écrit, dans notre cas, majoritairement en Csharp.

Lobby : Dans les jeux vidéos, les lobby agissent comme une zone pour permettre aux joueurs d'attendre le début de la partie.

Pixel : Le plus petit élément constitutif d'une image produite ou traitée électroniquement, définie par sa couleur et sa luminosité.

Tileset : Image, répertoriant toutes les tiles utilisables dans le jeu.

Branche : Une branche est une partie d'un dépôt Github, permettant aux développers d'un projet de ne pas tous travailler au même endroit, permettant une avancée sur le projet plus fluide.

Fix : "Réparer" en anglais, cela désigne l'action de régler un problème ou un bug.

Idle : Les animations "Idle" désignent celle qui sont appelées quand le personnage reste statique sans rien faire. Elles sont réalisées entre autre pour montrer au joueur que le jeu n'a pas crash.

Hitboxes : Zone invisible permettant de définir le contour de collision d'un personnage ou d'une tile de la tilemap.

PNJ : Personnage Non Joueur, désigne tous les personnages qui ne sont pas contrôlés par un joueur.

State Machine Système responsable de la gestion de l'état de la partie.

Peer to peer : Modèle d'échange en réseau où chaque entité est à la fois client et serveur.

MonoBehavior : Classe Unity basiques de Unity.

NetworkBehavior : Equivalent de *MonoBehavior* dans Mirror.

Rogue-Like : Sous genre du jeu vidéo, désignant les jeux ressemblant de près ou de loin au jeu Rogue.

PoC : Ou Proof of Concept, sert à montre la faisabilité du projet.

Seed : Ou graine en français, est une chaine de caractère ou suie de chiffres permettant de générer du contenu aléatoirement.

SVG : Scalable Vector Graphics, est un format de données uutilisépour représenter des images.

Tilesmap : Objet du jeu, créé à partir de tiles côte à côte.

Tiles : Case réalisée dans le but de construire une tilemap.

Tilemap collider : Composant de Unity, permettant de générer les zones de collision de la tilemap.

RigidBody2D : composant de base afin d'activer le comportement du moteur physique à un objet.

Monorepo : Une architecture logicielle dans laquelle plusieurs projets peuvent coexister au sein d'un même référentiel.

Pull request : Mécanisme qui permet à un développeur de prévenir les membres de son équipe qu'il a terminé une fonctionnalité.

Sprite Atlas : Composant Unity permettant de réorganiser les tiles entre-elles pour optimiser le rendu de la tilemap de Unity.

Path Finding : Type d'algorithme permettant de chercher le chemin le plus court, dans un labyrinth par exemple.

Shader : Les shaders sont des programmes utilisés en infographie pour définir l'apparence des modèles 3D et contrôler des aspects tels que l'éclairage, la texture, la couleur

Parallaxes : L'impact d'un changement d'incidence d'observation, c'est-à-dire du changement de position de l'observateur, sur l'observation d'un objet.

7.2 Sources

Liens des images utilisées :

- *Hollow Knight* : `https://store.steampowered.com/app/367520/Hollow_Knight/`
- *Dead Cells* : `https://store.steampowered.com/app/588650/Dead_Cells/`
- *Ori and the Blind Forest* : `https://store.steampowered.com/app/261570/Ori_and_the_Blind_Forest/`
- *Hades* : `https://store.steampowered.com/app/1145360/Hades/`
- *Outer Wilds* : `https://store.steampowered.com/app/753640/Outer_Wilds/`

7.3 Ressources mentionnées

- *Mirror Networking* : https://mirror-networking.com/
- *Edgar Unity* : https://ondrejnepozitek.github.io/Edgar-Unity/
- *Pixel Studio* : https://store.steampowered.com/app/1204050/Pixel_Studio__pixel_art_editor/
- *Git* : https://git-scm.com/
- *Unity* : https://unity.com/
- *Rust* : https://www.rust-lang.org/
- *Dead Cells* : https://store.steampowered.com/app/588650/Dead_Cells/
- *Site internet de Treep* : https://treep.world/
- *Rust* : https://www.rust-lang.org/
- *Mirror Networking* : https://mirror-networking.com/