Rapport de Soutenance

Treep

Kromm Studio

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

D1

Janvier 2025

Tables des matières

1 Introduction

Ce rapport a pour mission de définir et présenter en détail l'avancement de Treep et les étapes restantes à sa concrétisation.

Le nom du jeu vient de la fusion des mots *tree* (pour arbre) et *trip* (pour voyage), qui sont les deux axes majeurs de notre jeu.

En effet, dans *Treep* le joueur, incarne l'âme d'une gigantesque colonie d'insectes vivant au pied d'un arbre millénaire. 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, le joueur va être confronté 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.

Treep sera donc un Rogue-like, multijoueur, bénéficiant d'un algorithme de génération procédurale permettant à chaque partie d'être unique.

2 Bilan des réalisations

Dans l'ensemble, nous avons bien suivi les impératifs et le calendrier que nous nous étions fixées en début d'années. Nous n'avons pas fait de gros ajustements de ce dernier ni accumulé de retard particulièrement important dans un domaine.

2.1 Mouvement

Dans la conception de jeux vidéo, l'objectif est d'offrir aux joueurs une expérience fluide et captivante. Le mouvement en est responsable en partie. C'est pour ça que nous avons choisi de terminer le plus rapidement possible les différents éléments du mouvement.

Le saut a cinq phases : *Grounded*, *PrepareToJump*, *Jumping*, *InFlight* et enfin *Landed*. Ces cinq phases, permettent, par la suite, de faire une animation de saut détaillée.

De plus, l'animation de course change de sens en fonction de la direction vers laquelle le joueur se déplace.

Pour l'escalde, une autre tilemap a été rajoutée, avec des propriétés différentes de la première tilemap. Cette tilemap ayant le tag *ladder*, permet de vérifier si le joueur est en contact avec un objet ayant ce tag, pour savoir s'il est en contact avec une échelle. Des phases d'escalade sont aussi implémentées, permettant de rajouter une animation de grimpe prochainement.

Enfin, un grand nombre de variables est modifiable à l'intérieur même de Unity comme la gravité, la vitesse de course du joueur ou encore la vitesse de grimpe, nous permettant de les modifier facilement.

Les collisions entre le joueur et les tilemaps sont gérées avec le composant *Rigidbody2D* du joueur et par *TileMapCollider2D*.

Un *cast* est ainsi fait dans la direction du mouvement pour vérifier les collisions, puis ajuste la position et la vélocité du joueur en conséquence s'il heurte un mur ou un plafond déterminé par la normale du cast.

En résumé, ce code est une bonne base pour des mécaniques de jeu solides. Il permet des déplacements horizontaux fluides, des sauts réactifs et des interactions verticales avec les échelles.

Chaque fonctionnalité a été pensée pour rendre le jeu agréable et facile à jouer. Le code est aussi assez flexible pour pouvoir être amélioré et étendu dans le futur, offrant ainsi un bon point de départ pour le développement du jeu.

2.2 Multijoueur

Nous avons dès le début pensé à l'intégration du multijoueur dans notre jeu pour éviter les complications éventuelles liées à une implémentation trop tardive de ce dernier. Ainsi, le multijoueur a été l'une de nos principales priorités.

Pour faciliter le développement, le multijoueur est pour l'instant en *peer to peer*[1] avec un hôte qui héberge la partie. Cela nous permet de facilement tester le fonctionnement de la synchronisation entre les clients. Pour ce faire, nous utilisons la nouvelle fonctionnalité de Unity qui permet de cloner rapidement son projet pour lancer deux instances du jeu. Par la suite, le système peer to peer sera complété ou remplacé par un système dans lequel les clients doivent se connecter à un serveur distant indépendant.

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 sur ce point.

Mirror ajoute un nouveau 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*[2] mais un *NetworkBehavior*[3].

2.3 Génération procédurale

Une des difficultés dans un *Rogue-like*[4] est de proposer une expérience différente à chaque partie tout en gardant un contrôle sur le déroulement d'un niveau. C'est pour cette raison que la génération procédurale est un des concepts pilier du jeu.

Avant de nous lancer dans notre propre implémentation, nous avons décidé de regarder ce qui avait déjà été fait sur le sujet. Un article de blog fait par un développeur de *Dead Cell*, un jeu Rogue-like sorti en 2017, explique comment leur génération procédurale marche. Ils construisent leurs environnements salle par salle. L'algorithme de génération procédurale vient ensuite assembler ces dernières dans un ensemble cohérent à l'aide d'un arbre qui décrit le niveau fait lui aussi à la main. Cette manière de procéder permet donc aux développeurs de garder une large part de design manuel qui lui permet de s'assurer de l'expérience du joueur.

Une implémentation d'algorithme de génération procédurale par *Ondrej Nepozitek*, appelée Edgar, est disponible en ligne. Il apporte un soin tout particulier à l'intégration de son algorithme avec Unity. L'idée de l'implémentation ressemble fortement à celle employée par l'équipe de *Dead Cells*.

Pour correspondre au maximum à nos besoins, nous avons décidé de repartir de zéro pour avoir un contrôle maximum sur le fonctionnement de l'algorithme. Cela nous permettra d'implémenter nos propres passages pour répartir des objets ou des ennemis lors de leur implémentation.

Nous avons décidé de faire un premier *PoC*[5] de la génération procédurale. Celui-ci est en *Rust* car Milo, responsable de ce domaine, est plus à l'aise dans ce langage que le *Csharp*. La première étape a été de modéliser les différents composants. L'algorithme repose sur deux concepts :

- un *level blueprint* qui décrit l'architecture d'un niveau en donnant un type (e.g. couloir, boss, etc.) à chaque salle et qui liste toutes les liaisons entre elles. Pour des raisons de complexité algorithmique, mais aussi de game design, il est impossible d'avoir une boucle dans le niveau. Ce level blueprint est implémenté avec un graphe orienté, mais sera sûrement implémenté avec un simple arbre dans le code final.
- *un room provider* qui s'occupe de donner des pièces dans un ordre aléatoire à l'algorithme. Plus on construit de pièces que le room provider peut distribuer, plus les possibilités de génération sont grandes.

Le tout est régi par générateur d'aléatoire qui se base sur une valeur numérique de départ, la *seed*[6]. La génération d'un niveau est donc entièrement déterministe. Cela est utile pour coordonner les clients lorsque l'on joue en réseau et simplifie aussi le développement.

Si pour un room provider donné, le level blueprint est solvable, la sortie de l'algorithme est un second arbre avec la position de chaque pièce choisie. Cette représentation intermédiaire permet de transformer l'arbre pour différents cas d'usage. Dans le cas du PoC, celui est un rendu en *SVG*[7] qui permet d'inspecter le résultat. Dans le jeu, il sera utilisé pour placer les pièces préfabriquées. Cette même représentation nous donne aussi l'occasion d'effectuer des passages pour placer des objets rares ou répartir les ennemis uniformément.

L'algorithme procède par tâtonnement pour trouver un agencement correct. Pour chaque salle dans le *level blueprint*, le programme sélectionne une salle pré-construite du même type. Puis pour chaque porte du niveau actuel, il essaye de trouver un porte de même taille dans une des salles suivantes. On vérifie ensuite que la nouvelle salle sélectionnée ne s'intersecte pas avec le reste de l'agencement. Plusieurs simples vérifications comme marquer les portes déjà utilisées permettent de rapidement arrêter de chercher dans un sens quand celui-ci n'a aucune possibilité d'être correct plus tard.

Un point important que nous avons noté lorsque nous avons essayé Edgar-Unity est le travail fait autour de l'intégration entre le moteur de génération procédurale et l'éditeur pour simplifier le travail des game designers. Sans aller aussi loin que le projet mentionné, il est possible de simplifier le processus de création d'une pièce.

Le travail sur l'intégration permet donc aux game designers de composer des pièces pour la génération procédurale sans qu'ils aient à sortir de l'éditeur pour changer des lignes de code de configuration.

On peut observer plusieurs détails sur le rendu SVG de notre graphe. Tout d'abord, les salles sont représentées par des rectangles noirs avec le nom du template en bleu

aligné en haut à gauche. On peut aussi noter les portes, indiqués par un rectangle rouge, qui sont jointes et relient l'ensemble du niveau.

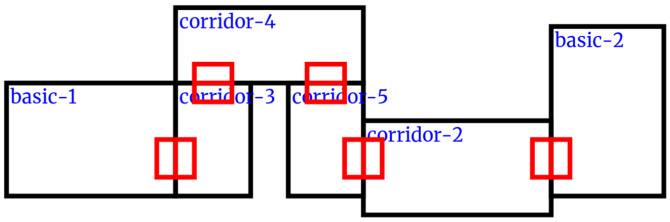

Figure 1: Exemple de niveau rendu avec notre PoC.

2.4 Site web

En vue des attendus de la première soutenance, nous avons commencé le développement du site web relativement tôt. Pour le moment, il est constitué uniquement de HTML, de CSS et de JavaScript. Étant donné la sobriété du site internet, nous ne voulions pas nous lancer dans l'utilisation d'outils plus compliquée que ce dont nous avons réellement besoin.

Pour ce qui est du design global, nous avons intentionnellement fait le choix de garder un style minimaliste et épuré par préférence stylistique et simplicité de réalisation. Nous avons donc opté pour une architecture à quatre pages.

- index.html : présente notre projet en expliquant son histoire, ses principales fonctionnalités et propose des liens pour télécharger la version la plus récente du jeu .
- appendices.html : permet d'accéder aux ressources utilisées tout au long du développement de notre jeu, ainsi que les liens pour télécharger les documents liés au projet.
- about.html : présente dans une première partie l'entreprise Kromm Studio puis dans un second temps, l'équipe accompagné d'une biographie de chaque membre.
- chronology.html : détaille l'avancement du projet, les problèmes rencontrés et les solutions choisies.

Étant donné nos expériences passées dans la création de site web, nous avions déjà un bagage de connaissances, ce qui nous a permis d'être efficace dans la création du site web. Nous avons choisi de mettre notre site en anglais pour le rendre accessible à une plus grande audience. À long terme, nous aimerions intégrer plusieurs langues.

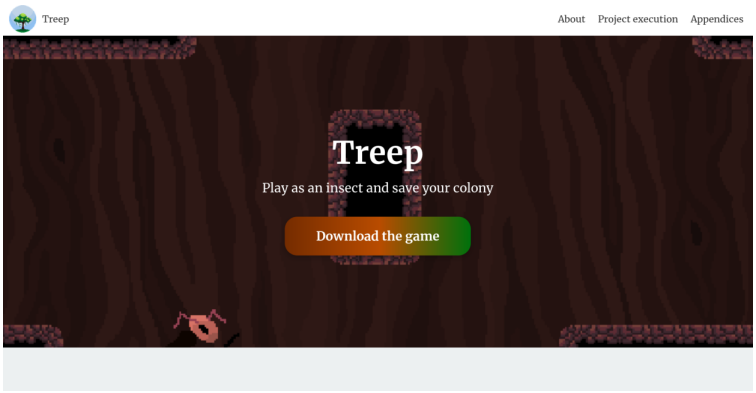

Figure 2: Page d'accueil de note site internet.

2.5 Conception des assets

Nous avons choisi d'utiliser des *tilemaps*[8] pour réduire le temps passé sur la conception graphique. En effet, il est possible de composer une large gamme de niveaux avec une ensemble de *tile*[9] réduit.

La conception des assets a été un défi majeur sur les trois derniers mois. En effet, l'ensemble du groupe s'est mis d'accord sur le fait que la conception d'assets étant compliquée, il fallait commencer au plus tôt pour ne pas prendre de retard. Ainsi, une grande partie des assets a été réalisée dans l'optique de proposer dès la première soutenance un prototype agréable à regarder.

La génération procédurale a besoin de niveaux, les mécaniques de mouvement nécessitent des animations et l'implémentation des éléments du *HUD*[10] nécessite des graphismes dans l'esprit du jeu.

La conception des assets a donc été premièrement autour d'un PoC pour trouver la direction artistique du jeu. Un gros travail sur les animations a ensuite été effectué.

De plus, la réalisation d'un PoC a permis de créer un pack d'assets utilisable dans le futur. Par exemple des ennemis, différents joueurs pour le multijoueur, une barre de vie ou encore un *tileset*[11] pour une future zone pas encore exploitée. Tous ces assets permettront une avancée plus fluide lors de notre processus de création.

Nous avons choisi d'utiliser un tileset avec des tiles de 16 par 16 pixels. Cette résolution a été choisie pour permettre de rester dans les codes des jeux 2D du type Rogue-like, tout en ayant un bon rendu visuel.

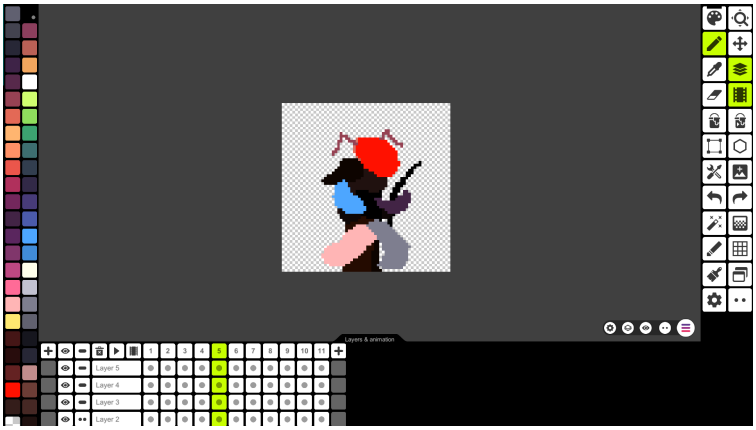

Figure 3: Logiciel de conception des assets

2.6 Réalisation des niveaux

Pour la réalisation des niveaux, comme expliqué dans la conception des *assets*[12], les niveaux sont réalisés à la main avec une tilemaps. Le premier niveau, effectué pour la soutenance, devait répondre à deux contraintes : créer un niveau esthétique et présenter les différentes fonctionnalités implémentées dans le jeu. Ainsi, trois zones avec deux différents types d'échelles ont été réalisés et deux grandes salles pour montrer le saut. Enfin des zones à deux cases de hauteur pour montrer le bon fonctionnement des collisions du personnage avec son environnement. Nous avons utilisé un *tilemap collider*[13], pour permettre au personnage d'interagir avec le sol (marcher) et les murs (se faire bloquer par des murs).

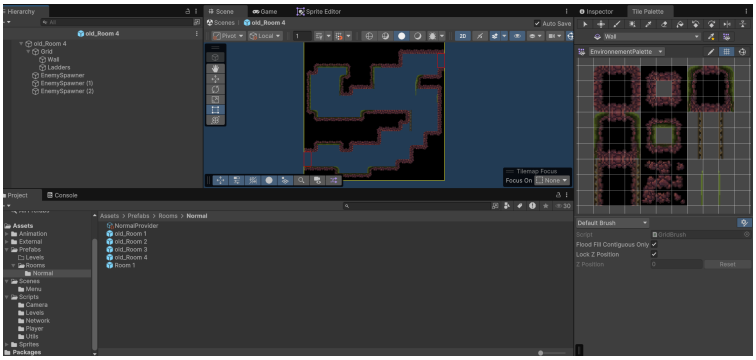

Figure 4: Démonstration conception d'un niveau

3 Retards et défis rencontrés

L'avancement du projet que nous avions planifié en octobre est pour le moment relativement bien respecté, à l'exception des effets sonores, des mécaniques de combat et du HUD.

D'autres tâches comme les effets sonores où les interfaces utilisateurs ont été survalorisées dans l'ordre d'importance à la conception du planning. En effet, commencer à se concentrer sur les effets sonores, la musique et les menus à ce stade du projet n'est pas le plus judicieux et c'est pour cette raison que nous avons retardé leur développement.

Un autre challenge jusqu'à présent était de créer un environnement de travail propre pour avancer de façon organisée. Il fallait donc, en plus de travailler sur le jeu, se former à l'utilisation des différents outils techniques que nous utilisons pour pouvoir travailler efficacement.

Pour faciliter l'organisation et la fluidité du développement du projet, nous avons créé une *monorepo*[14] Git pour centraliser le code de notre jeu, du site internet, du PoC, etc. Cette organisation nous a demandé à tous un effort de prise en main de *Git* que nous utilisons pour gérer les versions de notre code. Cependant, nous sommes maintenant tous parfaitement à l'aise avec cet outil. À chaque modification que nous souhaitons apporter, nous créons une branche dérivée de la branche principale. Nous utilisons ensuite les *pulls requests*[15] pour nous coordonner, voir l'avancement de certaines fonctionnalités et approuver mutuellement notre code.

3.1 Mouvement et Gameplay

La base des mécaniques de mouvement est présente, mais quelques ajustements restent à effectuer. Par exemple, le personnage se déplace rapidement et saute loin. En réduisant la vélocité du personnage, cela donne un effet plus naturel au gameplay. De plus, un effet de "glissade" apparait lorsque l'on court assez vite qui a été supprimé pour faire ressentir au joueur qu'il court sur du bois sec et non de la glace.

Un bogue apparait lorsque l'on saute d'une manière précise sur le plafond, ce qui brise l'immersion du joueur, car des problèmes de collision apparaissent et nous font traverser le plafond.

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 boite 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 était à revoir pour corriger des bogues comme l'absence de collisions à certains endroits. De plus, le personnage devait coller les murs avec

une échelle pour le monter, mais un problème de collision laissait le joueur grimper les échelles sans qu'il ne les touche. Ce problème a été résolu en créant une tilemap spécifique aux échelles.

Enfin, la synchronisation entre le saut et son animation a été compliqué, car il est constitué de différents états (e.g. impulsion, en vol, atterrissage, etc.).

3.2 Multijoueur

En intégrant le multijoueur à notre jeu, nous avons très vite remarqué que la position client qui n'était 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*. Dans cette architecture, l'hôte possède l'autorité sur toutes les données de la partie et contrôle les mouvements des autres joueurs qui ne sont pas autorisés à se déplacer comme ils le souhaitent. Il est possible de déléguer cette autorité à tous les clients.

3.3 Site web

Comme expliqué précédemment, 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.

3.4 Conception des assets et des niveaux

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.

Deuxièmement, les assets 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.

Troisièmement, par défaut, Unity lisse les images pour que les pixels ne se voient pas. Cependant, dans notre case, 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*[17], permettant de rassembler nos assets pour optimiser les performances et régler des problèmes de calcul de position à l'exécution.

3.5 Génération procédurale

L'une des premières difficultés 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 Planification à venir

4.1 Interface Utilisateur

L'interface utilisateur a pris un peu de retard. La plupart des tâches que l'on s'était attribué en amont sur le HUD ne se sont pas révélées prioritaires. Réaliser un menu avant un jeu est contradictoire. Voici donc les tâches à réaliser dans le futur :

- L'implémentation d'un système de sauvegarde : le joueur pourra choisir de quitter en sauvegardant sa partie ou non. De plus, on pourrait imaginer faire différents emplacements pour stocker des sauvegardes du joueur pour qu'il puisse lancer plusieurs parties différentes.
- L'implémentation d'un menu de réglages : le joueur pourra changer ses touches (mouvement, combat, interaction, etc.), la luminosité ou encore le volume sonore.
- Enfin, le HUD du joueur sera composé de deux barres indicatives. Une pour la vie et l'autre pour l'endurance et l'utilisation des capacités. Leur fonctionnement est pour l'instant basique et leur taux de remplissage diminue ou augmente proportionnellement à la vie ou à l'endurance que le joueur perd ou gagne.

4.2 Intelligence artificielle

Nous allons implémenter des systèmes d'intelligence artificielle. Parmi ces comportements, il y aura :

- Un ennemi solitaire qui 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 qui 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 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.

Pour le déplacement, nous utiliserons un algorithme de *path finding*[16], ayant pour but de trouver le chemin le plus court entre lui et sa cible. L'algorithme utilisé pourrait être l'algorithme $A*$ ou celui de *Dijkstra* en fonction du type d'implémentation.

4.3 Mouvement et gameplay

Quand toutes les animations de mouvement seront implémentées dans le jeu, les membres responsables des mécaniques de mouvement devront s'assurer que l'animation est raccord avec la vitesse de déplacement. En effet, lors d'une course par exemple, le pied ne devra pas bouger quand il sera au sol, pour une question de réalisme et d'immersion.

De plus, dans le futur, des IA seront implémentées au jeu, ce qui demandera aux membres responsables du mouvement et du gameplay de réaliser un set de mouvement personnalisé par personnages non-joueurs. Si un ennemi vole, il faudra gérer la physique et la gravité, ainsi que réfléchir à son endurance et à ses mécaniques de combat.

Le personnage joué par le joueur aura aussi besoin d'un set de mécaniques de combat et l'implémentation de mécaniques de mouvements comme la propulsion en avant, le double saut ou d'autres pour rendre le gameplay plus dynamique.

4.4 Multijoueur

Les fondements du multijoueurs sont posés, cependant nous devons continuer de l'intégrer au jeu au fur et à mesure que nous ajoutons des fonctionnalités. Nous devons aussi nous concentrer sur la création d'un véritable serveur de jeu pour remplacer le peer to peer où un joueur fait office de serveur pour les autres membres de la partie. Cela était indiqué dans le cahier des charges techniques.

Du côté du jeu, ce changement n'a pas d'impact, mais implique de penser à la création d'un serveur central pour héberger les parties et aussi la mise en place d'une architecture pour l'héberger.

4.5 Génération procédurale

Nous disposons d'un PoC fonctionnel pour la génération procédurale que nous allons dès à présent adapter en Csharp pour l'intégrer dans notre jeu. Ce passage est important, car une grande partie du jeu repose sur cet algorithme. On pourra ajouter des propriétés aux pièces au fur et à mesure de l'implémentation de l'IA, des ennemis et d'autres fonctionnalités.

4.6 Site Web

Bien que majoritairement fini, il nous reste encore quelques détails à ajouter ou modifier sur le site internet.

Nous voudrions le rendre plus complet sur plusieurs points, notamment la présentation du jeu qui manque de détails tant sur les descriptions que dans le contenu audiovisuel qu'elle présente. Un de nos objectifs est donc d'ajouter une bande-annonce du jeu. Du point de vue du design, nous aimerions trouver une alternative au fond blanc de la page principale et retravailler différents éléments comme le menu de navigation.

4.7 Conception des assets et des niveaux

La plupart des assets ont été réalisés. En effet, des design d'ennemis et leurs animations seront nécessaires pour la suite de la réalisation du projet.

Nous avons produit un tileset pour la première zone de l'arbre, mais nous avons prévu de réaliser trois autres zones, nécessitant, pour chacune d'entre elles, un tileset personnalisé. Différents arrière-plans devront aussi être réalisés pour les différents niveaux, permettant de donner un effet de progression lors du changement de niveau.

Il faut également que l'on ajoute plus d'animations au personnage, notamment pour le saut, les échelles et les mécaniques de combat. De même, nous allons continuer de travailler les animations du décor pour le rendre plus dynamique et moins pauvre en ajoutant des effets de particules et des petites animations à certains éléments du fond.

Pour ce qui est de la réalisation des niveaux, il faudra concevoir un grand nombre de salles en tout genre pour diversifier les différentes parties pouvant être lancées.

Les salles pourront être de trois tailles différentes : petite, moyenne ou large. Les salles petites auront 0, ou un ennemi, les moyennes pourront avoir jusqu'à trois ennemis et les salles larges pourront avoir jusqu'à sept ennemis. De plus, elles seront agencées de façons aléatoires, et n'apparaitront toutes pas dans une partie pour laisser un roulement de salles. Mais certaines salles seront toujours présentes, ces salles spéciales sont : les salles de boss, les salles de marchand ou encore salles didacticiel.

5 Récapitulatif de l'avancement et de la répartition

	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				10% (-15%)	50%	100%
Interface utilisateur (menus, HUD, etc.)	R	S				0% (-50%)	100%	100%
Game design								
Histoire/Lore		R		S		90%	100%	100%
Conception des niveaux			S		R	25% (-15%)	75%	100%
Conception graphique	S			R		45% (+20%)	75%	100%
Conception sonore		S	R			0% (-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. Les pourcentages entre parenthèses au delta entre l'avancement actuel et l'avancement initial

6 Conclusion

Jusqu'à maintenant, nous avons quasiment fini le développement du site internet, comme vous avez pu le voir précédemment, les mouvements de base, même s'ils pourront tous deux être retravaillés par la suite.

Nous avons également grandement avancé dans la conception des assets avec le modèle des personnages, mais aussi des tiles pour le design des niveaux. Dans le même domaine, nous avons aussi bien travaillé les animations et la conception des niveaux.

Les éléments complexes comme le multijoueur et la génération procédurale ont aussi été travaillés avec la création d'un PoC pour la génération procédurale.

Un autre challenge jusqu'à présent était de créer un environnement de travail propre pour avancer de façon organisée sur le projet. Il fallait donc tous se former à l'utilisation des différents outils techniques que nous utilisons pour pouvoir travailler efficacement.

Cependant, il reste encore beaucoup à faire, notamment sur l'intelligence artificielle, l'ambiance sonore et les mécaniques de combat que nous n'avons pas commencé. Parmi ces trois domaines, celui de l'intelligence artificielle est sûrement celui qui va être le plus compliqué si l'on veut atteindre nos attentes.

7 Annexes

7.1 Définitions

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

Peer to peer[1] : Modèle d'échange en réseau où chaque entité est à la fois client et serveur.

MonoBehavior[2] : Classe Unity basiques de Unity.

NetworkBehavior[3] : Equivalent de *MonoBehavior* dans Mirror.

Rogue-Like[4] : Sous genre du jeu vidéo, désignant les jeux ressemblant de près ou de loin au jeu Rogue.

PoC[5] : Ou Proof of Concept, sert à montre la faisabilité du projet.

Seed[6] : Ou graine en français, est une chaine de caractère ou suie de chiffres permettant de générer du contenu aléatoirement.

SVG[7] : Scalable Vector Graphics, est un format de données uutilisépour représenter des images.

Tilesmap[8] : Objet du jeu, créé à partir de tiles côte à côte.

Tiles[9] : Case réalisée dans le but de construire une tilemap.

HUD[10] : Ou Head Up Display, désigne l'ensemble des éléments appartenant à l'interface graphique.

Tileset[11] : Un répertoire de tiles.

Asset[12] : Elément de la partie artistique de la création de jeu.

Tilemap collider[13] : Composant de Unity, permettant de générer les zones de collision de la tilemap.

Monorepo[14] :

Pull request[15] :

Sprite Atlas[16] : Composant Unity permettant de réorganiser les tiles entre-elles pour optimiser le rendu de la tilemap de Unity.

Path Finding[17] : Type d'algorithme permettant de chercher le chemin le plus court, dans un labyrinth par exemple.

7.2 Références bibliographiques

- *Article de blog de Dead Cells* : `https://www.indiedb.com/games/dead-cells/news/the-level-design-of-a-procedurally-generated-metroidvania`

- *Algorithme de Dijkstra* : https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm
- *Algorithme de recherche A** : https://en.wikipedia.org/wiki/A*_search_algorithm

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://mrnossiom.github.io/treep/