Abyssal Ascension

Rapport de soutenance 2

BOUDJEMA Elyas - MONSELLIER Jonathan - VASSAUX Clément - XIA Frédéric

Table des matières

1. Introduction

1.1. Présentation du projet

Abyssal Ascension est un jeu de plateforme 2D de type Metroidvania, inspiré de grands jeux tels que Hollow Knight et Ori. Le projet vise à proposer une expérience immersive mélangeant exploration, réflexion et progression dans notre univers.

Le jeu plonge le joueur dans un monde mystérieux, initialement sombre et oppressant, où la lumière constitue un élément central de la progression. Au fil de l'aventure, l'environnement s'éclaircit, cela symbolise à la fois l'ascension du personnage vers les hauteurs et donc son évolution au sein de cet univers.

Le joueur devra progresser à l'aide d'un nombre limité d'objets lui permettant d'avancer dans le jeu. Cette contrainte volontaire cherche à mettre l'accent sur la réflexion et l'observation de l'environnement plutôt que de juste monter en puissance.

L'exploration occupe une place centrale dans le jeu. Le monde est structuré de manière interconnectée pour encourager le joueur à revisiter des zones déjà explorées à l'aide de nouveaux objets découverts. Chaque zone possède sa propre identité visuelle et sonore afin d'obtenir un univers le plus riche et réaliste possible.

Le projet intègre également un système de jeu en multijoueur local. Ce système permet aux joueurs de coopérer ou d'interagir de différentes manières.

L'ensemble des graphismes, des sprites et des animations des personnages sont entièrement originaux afin de renforcer le visuel du projet.

De plus, nous pensons que la bande sonore joue un rôle important dans l'immersion du joueur. C'est pour cela que nos musiques originales et nos effets sonores fait maison ont été créés pour accompagner les différentes phases du jeu et renforcer l'atmosphère globale de notre jeu.

Abyssal Ascension est donc notre vision complètement originale du metroidvania, tel que nous l'imaginons : une expérience centrée sur l'exploration et la réflexion où l'ambiance du jeu est un élément clé pour l'immersion du joueur.

Ce projet représente également pour nous une opportunité de mettre en pratique les compétences apprises lors de notre première année à l'EPITA , que ce soit

en programmation, en organisation ou en rigueur. Abyssal Ascension est donc un jeu que nous avons imaginé et qui montre notre travail, notre créativité et notre capacité à créer une expérience complète.

1.2. Présentation de Caféine

Nous sommes Caféine, un groupe qui tire son nom de notre motivation pour ce projet et de notre addiction à la caféine (pas le café par contre, c'est pas bon). Ce nom montre à la fois notre énergie, notre implication et notre volonté de mener ce projet jusqu'au bout.

Notre équipe est composée de profils variés et complémentaires. Certains membres ont déjà eu l'occasion de participer à la création de jeux, ce qui nous apporte une première expérience dans le domaine. D'autres possèdent de bonnes bases en programmation, notamment en Python, tandis que certains se distinguent davantage par leur logique et leur capacité à résoudre des problèmes. Cette diversité est un grand atout pour le développement de notre projet.

Ce mélange de compétences nous permet d'apprendre les uns des autres tout au long de la conception du jeu. Chacun peut apporter ses idées, partager ses connaissances et progresser dans des domaines qu'il maîtrise moins. Cela crée une dynamique de groupe où la collaboration est essentielle.

Nous répartissons les tâches selon les points forts de chacun, tout en restant flexibles. Nous cherchons à avancer efficacement tout en permettant à chaque membre de s'impliquer autant que possible.

2. Conception

2.1. World Design

Sans monde, il n'y a pas d'Abyssal Ascension. Il faut donc travailler sur le world building.
Pour cela, j'utilise le logiciel LDtk (Level Design ToolKit). Ce logiciel permet la conception de monde de manière simple et efficace.

Pour le début de la conception du monde, on a choisi de prendre un design (tileset) simple composé de sol en pierre et d'un fond violet (cf. Figure 2 et 3) qu'on a projeté sur un layer avec des règles pour pouvoir faciliter la conception des niveaux. (cf. figure 7)

De plus, chaque zone dans le monde a une dimension de 960x480px et chaque cases (tiles) dans la zone a une taille de 24px par 24px (cf. Figure 4). L'agencement des zones entre eux est également rendu très simple grâce à l'interface de la gestion du monde (cf. Figure 5 et 8), il suffit juste de prendre et glisser la zone à l'endroit qu'on souhaite.

Lors de la création des zones, on choisit donc un layer (couche, comme stone ou dirt) et on dessine le terrain que l'on souhaite obtenir) (cf. figure 9).

Les portes permettant de relier deux niveaux entre eux fonctionne de la même façon qu'une couche. On vient créer une sous partie dans le layer avec la classe entities et on lui ajoute une valeur pour qu'il puisse agir dans tous les niveaux sans restriction. Puis on choisit dans le premier niveau l'endroit qu'on souhaite qui va la relier au deuxième niveau et la même chose dans le second niveau. La liaison entre les deux portes se fait automatiquement.

Ce logiciel exporte la map en png et JSON. Le format JSON est constitué ainsi : la grande sous-partie "Level" contient les différents niveaux créés, qui se segmentent ensuite en plusieurs propriétés techniques indispensables.

Chaque niveau possède d'abord un identifiant unique ainsi que des dimensions précises en pixels avec les champs "pxWid" et "pxHei", ce qui permet au script de définir les limites de la zone de jeu.

À l'intérieur de ces niveaux, on retrouve les "layerInstances", qui représentent les différentes couches de la carte. La plus importante est la couche de collision, cette liste permet au code de traduire chaque case en une donnée logique, où une valeur spécifique comme le chiffre 2 indique un obstacle infranchissable, tandis que le 0 correspond à une zone vide.

Parallèlement, les autoLayerTiles s'occupent de la partie visuelle en faisant correspondre chaque coordonnée de la grille aux textures sources du tileset.

Enfin, le fichier répertorie également les "entityInstances", qui servent à placer les objets dynamiques, c'est-à-dire les joueurs et les ennemis, et les points d'intérêt sur la carte. C'est ici que sont enregistrées les coordonnées précises du point d'apparition du joueur, le "PlayerSpawn", ou encore les positions des ennemis.

L'ensemble de cette structure est conçu pour être lu par le script ldtk_loader.py, qui ne conserve que ces données essentielles afin d'alléger le chargement et de charger le map dans le jeu.

2.2. IA

Après avoir conçu la map, il faut qu'il y ait un premier niveau de difficulté pour le joueur. C'est pourquoi nous avons développé une entité ennemie dotée d'un

comportement simple pour les ennemis banaux que le joueur va rencontrer sur son chemin.

Le comportement de l'ennemi repose sur un algorithme de patrouille unidimensionnelle. C'est-à -dire que l'entité se déplace horizontalement entre deux bornes définies par une position d'origine et un rayon de patrouille paramétrable (cf. figure 15).

À chaque frame, le système évalue deux conditions distinctes pouvant entraîner un changement de direction : le dépassement des limites de la zone de patrouille d'une part, et la détection d'une collision avec un obstacle latéral d'autre part. Cette seconde condition est particulièrement importante, car elle permet à l'ennemi de réagir dynamiquement à la géométrie du niveau sans nécessiter de connaissance préalable de la carte. Ainsi, même dans des environnements complexes, l'entité adopte un comportement visuellement cohérent et crédible, sans recourir à des techniques de navigation avancées telles que le pathfinding.

La gestion des collisions suit le même principe que celui employé pour le joueur. Chaque axe est géré de manière indépendante et séquentielle, les collisions horizontales sont d'abord détectées et modifiées, puis les collisions verticales sont traitées dans un second temps.
Cette méthode permet d'éviter les imprécisions liées aux collisions diagonales, qui peuvent survenir lorsque les deux axes sont traités simultanément.

La gravité de l’entité utilise la même variable que l’entité joueur afin d’obtenir un même comportement durant une chute. La gravité augmente la vitesse de chute de l’entité progressivement jusqu’à atteindre une vitesse maximale. Cette méthode permet de simuler l'effet de la pesanteur et d'assurer un comportement physique cohérent avec le reste du jeu.

2.3. Multijoueur

Un des piliers d’Abyssal Ascension est la coopération entre les joueurs. Nous avons implémenté une architecture client-serveur en LAN (Local Area Network) permettant une synchronisation fluide et cohérente entre deux instances du jeu tournant sur le même réseau local.

Pour synchroniser les données en python, j’utilise la bibliothèque native python `socket`. Le système tourne autour de 2 entités :

- Le serveur (host) : héberge la logique du jeu et distribue aux clients l’état du monde (positions de l’ensemble des joueurs, ennemis, statut de la partie…). Le serveur à toujours l’identifiant 0.

- Les clients : transmettent la position de leur joueur au serveur après l'application des inputs (entrées claviers / souris) permettant le déplacement du joueur. Chaque client à un identifiant unique, il commence avec un identifiant temporaire de -1 (indiquant qu'il n'est pas encore connecté au serveur), puis une fois connecté, le serveur lui attribue un identifiant libre (par exemple 1 si ce client est le premier à se connecter sur le serveur).

(cf. figure 14)

L'échange des informations s'effectue via le format JSON. Ce format permet de structurer des données complexes (coordonnées, états, liste d'ennemis…) sous forme de chaînes de caractères lisibles, facilitant ainsi le débogage et la communication entre les instances.

Pour maintenir une expérience multijoueur fluide tout en limitant la surcharge du réseau, les données sont synchronisées toutes les 3 frames. Le jeu tournant à 60 fps, cela revient environ à une synchronisation toutes les 50ms.

Cependant, j'ai rencontré un problème majeur lors des tests du système multijoueur. Nous utilisons le protocole TCP (Transmission Control Protocol), et celui-ci ne garantit pas toujours la réception des messages sous la même forme qu'ils ont été envoyés. Par exemple, si on envoie le message "Bonjour" à une machine, celle-ci pourrait recevoir deux messages différents, le premier "Bon" et le second "jour". Cela faisait planter le système de chargement du JSON de la machine de réception car elle pouvait donc recevoir des messages JSON incomplets.

Pour résoudre ce problème, j'ai utilisé la méthode du préfixe de taille du message. Chaque message envoyé est précédé de la taille totale du message en bytes. Par exemple le message JSON suivant : `{"x":10,"y":20}` fait 16 bytes et sera donc envoyé sous la forme suivante : `[00000016][{"x":10,"y":20}]`, où les 4 premiers bytes contiennent la taille du message.

La machine qui reçoit les données lit d'abord les bytes reçus jusqu'à en obtenir 4, ce qui correspond à la taille du message. Elle continue ensuite la lecture jusqu'à avoir reçu l'intégralité des bytes correspondant au message.

Cette méthode permet donc à la machine de réception de reconstituer correctement le JSON, même si celui-ci a été découpé en plusieurs morceaux pendant la transmission.

J’ai également intégré le changement de niveau dans le multijoueur. Lorsqu’un joueur prend une porte pour changer de zone, son niveau actuel est synchronisé avec les autres joueurs via le réseau. Ainsi, si deux joueurs se trouvent dans des niveaux différents, ils ne se voient plus à l’écran ce qui évite l’effet d’un joueur qui “vole” dans le vide au milieu d’une zone qui n’est pas la sienne.

2.4. Menus & Interface utilisateurs

Le menu principal a pour image de fond, une image sombre représentant l’ambiance du début du jeu (les abysses).
Le thème principal est également joué en arrière plan pour rendre le menu principal plus complet et vivant.

Pour faciliter la connexion avec des amis, on a décidé d’afficher l'adresse IP locale de la machine dans le menu principal. Cela permet de la partager et de pouvoir héberger puis rejoindre une partie multijoueur plus rapidement.

Trois modes de jeu sont proposés :

- Solo : Lancement immédiat d’une partie locale.
- Héberger (multijoueur) : Création d’un serveur et lancement immédiat de la partie hébergée.
- Rejoindre (multijoueur) : Redirige vers une interface de saisie.

Dans le menu “Rejoindre”, le joueur saisit l’IP du serveur à rejoindre. Un indicateur visuel valide la saisie en temps réel selon le format standard (présence des points et chiffres compris entre 0 et 255).

Lorsque le joueur se retrouve en jeu (en mode solo ou multijoueur), il peut voir de nombreuses informations de débogage en haut à gauche de son écran. Ces informations correspondent au mode de jeu actuel, au nombre de joueurs connectés sur le serveur, à l’identifiant du joueur, au nombre de FPS, à la position et à la vélocité du joueur local, ainsi qu’à sa vie et à son statut (au sol / en l'air).

2.5. Textures & Animations

Pour tout ce qui touche à l’animation, nous avons choisi le style "pixel art". De plus, chaque animation et graphique a été dessiné à la main. Le site internet que nos designers utilisent actuellement est "Piskel". L’interface est rapide à prendre en main et très pratique.

Pour ce faire, il faut suivre plusieurs étapes. Sur le logiciel de création utilisé, on doit dans un premier temps dessiner ce que l’on appelle des "sprites" (cf. figure

10). Ils correspondent à une image unique. Via le logiciel, on va ensuite disposer plusieurs sprites à la suite et les faire défiler pour créer du mouvement (cf.figure 11). Cela permet alors d'importer chaque animation dans le jeu vidéo et de les lier à chaque action. Les graphismes présents sur l'écran d'accueil, les futures cinématiques ou encore le site internet ont également été créés via ces techniques.

Pour l'implémentation dans le jeu, il n'y aura rien de très compliqué. Lors de l'écriture du code, nous avons des variables globales qui définissent le statut du joueur (course / marche / chute…). Lorsque l'on veut déclencher une animation spécifique, il suffira de relier les sprites correspondants au bon moment en fonction de la du statut du joueur. Tant que la condition est remplie, on fait défiler les sprites correspondants. Le personnage avance donc dans le jeu et l'animation tourne en boucle jusqu'à ce que ce dernier s'arrête.

Pour la suite, on ajoute les sprites directement dans un fichier du jeu prévu pour l'animation afin de pouvoir les afficher selon certaines conditions. Les images sont enregistrées en fichier .PNG afin de faciliter leurs utilisations. Chaque image est renommée d'une façon précise : 0-Running-0 signifie dans l'ordre qu'il s'agit du personnage N°1; de l'animation de course (running); du sprite N°0. Cela permet de s'y retrouver très facilement lorsque nous sommes amenés à coder pour les animations. Il faut aller chercher la bonne image au bon endroit. (cf. figure 12)

Pour rendre les animations de déplacements plus fluides, nous nous sommes basés sur de courts extraits de vidéos ou de films pour pouvoir décortiquer le mouvement dans son ensemble. Il nous a ensuite suffi de visionner ces extraits encore et encore afin de sélectionner les meilleures images, pour pouvoir les reproduire nous-mêmes à la main. Lorsque l'animation en question n'est autre qu'un mouvement perpétuel, il suffit alors de faire tourner une boucle dans le code qui passe en continu les mêmes images, afin de donner l'impression que le mouvement est continu.

Pour afficher ces animations, il y a plusieurs conventions à respecter. Deux animations différentes ne peuvent pas avoir lieu en même temps. En effet, si le joueur est dans l'état initié "en saut" mais qu'il continue à avancer vers l'avant, l'animation qui doit être affichée est celle du saut et non celle de course. Il y a donc des priorités à déterminer lorsque l'on parle d'affichage d'animations. De plus, les obstacles doivent empêcher les animations d'avoir lieu. Si le joueur se heurte à un mur, sa course doit naturellement s'arrêter. Il y a par conséquent des conditions supplémentaires à implémenter pour toutes les interactions avec l'environnement. Des déplacements spécifiques tels que le sliding sur les murs sont aussi à prendre en compte lors de la création de ces règles.

Par la suite, implémenter l'affichage des animations n'est pas compliqué. Les états prédéfinis lors de la création de la classe "Personnage" servent à détecter les mouvements du joueur. Lorsque l'état qui nous intéresse est actif, il suffit de déclencher au même moment une boucle qui chargera à l'emplacement du joueur trois à quatre images a la suite. Ces images auront un temps d'apparition différent en fonction du type de mouvement. Pour la course par exemple, chaque sprite sera affiché durant 5 frames. Le jeu tournant à 60 FPS en permanence, cela permet de bien décortiquer le mouvement et de voir toutes les étapes de l'animation apparaître.

Pour ce qui est du multijoueur, chaque personne connectée aura le même design de personnages, mais avec une couleur de cape différente. D'autres petites modifications apparaissent également (cf. figure 13). Les couleurs permettent donc de différencier les joueurs mais rien de plus. Cela suffit pour l'aspect technique du multijoueur.

Pour tout ce qui est des monstres : le jeu Abyssal Ascension a comme ambiance une grotte plutôt sombre et mystérieuse. Les monstres à choisir pour affronter le joueur doivent donc être en accord avec le type du jeu ainsi que l'ambiance. Nous avons donc opté pour des zombies pour tout ce qui est au corps à corps, des squelettes pour du potentiel combat à distance (cela reste à déterminer lors de la dernière étape de création du jeu). Des monstres volants seront aussi implémentés pour l'aspect de déplacement. En effet, cela ajoute une dimension importante au jeu. Le joueur ne pourra pas qu'esquiver afin de passer aux salles suivantes. Il sera donc contraint d'apprendre les mécaniques de déplacements ainsi que d'esquive grâce à ces monstres particuliers. Les animations d'attaques des monstres ont été sauvegardées dans un fichier spécifique du jeu, mais n'ont pas encore été implémentées car le système de point de vie et d'attaque est en cours d'implémentation.

L'étape suivante est le design du second plan du jeu. L'affichage du world building fonctionne grâce à des tuiles. Ces dernières sont des planches sur lesquelles sont dessinées un environnement simple contenant quelques obstacles permettant de faire des tests facilement lors des implémentations. De la même manière que pour les animations de monstres ou du personnage principal, il suffit de dessiner des sprites assez grands pour pouvoir les superposer sur les tiles. Ces sprites ne changeront jamais car l'environnement reste fixe. Les tiles changeront en fonction du passage du joueur dans des salles différentes, mais chaque objet aura sa propre sprite qui lui sera attribué, donc chaque changement d'environnement sera fait automatiquement.

2.6. Son & Musique

Nous avions décidé de créer notre propre musique pour le jeu afin de renforcer l'immersion du joueur. Plutôt que d'utiliser des ressources externes, nous avons fait le choix de produire une bande sonore entièrement originale, en accord avec l'ambiance du projet.

A l'aide de l'expérience que j'ai accumulée après dix ans d'années de pratique de piano et de composition musicale, j'ai utilisé le logiciel gratuit MuseScore pour composer un thème principal pour le jeu. Ce travail de composition m'a permis d'adapter précisément la musique à l'univers du jeu, en réfléchissant à la fois aux émotions à transmettre et au rôle de la musique dans l'expérience du joueur.

L'objectif des musiques d'ambiances est de créer une atmosphère lourde et pesante, en cohérence avec l'univers sombre du début du jeu. Pour cela, j'ai principalement utilisé des tonalités mineures (comme le La mineur ou le Mi mineur), qui permettent d'installer une tension. Ces choix musicaux permettent de renforcer le ressenti global du joueur.

Pour le thème principal, j'ai choisi le sol dièse mineur (G# mineur), une tonalité qui évoque des émotions profondes et intenses. J'ai également utilisé un son de piano "felt", qui donne un rendu plus feutré et intime, ce qui correspond bien à l'ambiance du jeu. J'ai créé ce thème comme un élément central de l'identité sonore du projet, comme un thème principal. (cf. Figure 1 pour voir la partition)

Pour les effets sonores, nous avons fait le choix de les enregistrer nous-mêmes afin de garder une cohérence avec le reste du projet. Nous avons principalement utilisé notre voix pour simuler certains sons, notamment pour les actions comme le saut ou le dash. Ce procédé nous a permis d'obtenir des résultats originaux et personnalisés.

Nous avons également enregistré nos propres bruits de pas pour les effets sonores de déplacement (course). Cela nous a permis d'ajuster précisément les sons en fonction des animations et du jeu.

2.7. Site Web

Afin d'accompagner le projet, nous avons conçu un site web composé de plusieurs rubriques et organisé par un style css simple et coloré. Nous avons choisis des couleurs plutôt sombres comme un bleu très foncé #0a0e27 ou encore un bleu marin #1e3a8a qui représentent bien les couleurs et l'ambiance de notre jeu.

Pour chaque rubrique nous avons créé des classes pour pouvoir manipuler leur style (dimensions / couleurs) séparément depuis le fichier css afin d'obtenir un fichier structuré.

Le site est hébergé par github pages depuis le repository principal du jeu dans le dossier 'docs' situé à la racine du projet.
A chaque nouveau commit, le site se met à jour automatiquement par les actions github (ensemble d'actions automatisées par github).

Il est possible de télécharger l'ensemble des rapports de soutenance depuis le site dans l'onglet "Rapports".

2.8. Contrôles

J'ai implémenté les contrôles de base pour le joueur, qui constituent le cœur du gameplay et permettent une prise en main rapide et intuitive du jeu. L'objectif était de proposer des commandes simples, efficaces et accessibles, tout en laissant suffisamment de précision pour les phases de plateforme (mouvement).

Le joueur peut utiliser les touches Q / D ou les flèches gauche / droite afin de se déplacer horizontalement. Ce double choix de touches permet de s'adapter aux préférences de chaque joueur, certains étant plus à l'aise avec les touches directionnelles classiques, tandis que d'autres préfèrent utiliser le clavier en configuration AZERTY.

Pour sauter, le joueur peut utiliser la touche espace ou la flèche vers le haut. Le saut est une mécanique essentielle dans un jeu de plateforme. C'est pour cela que nous avons peaufiné les paramètres de cette action afin de nous satisfaire le plus possible.

De plus, pour effectuer un dash, le joueur peut utiliser les touches shift gauche ou shift droite. Cette capacité permet au joueur de se déplacer rapidement sur une courte distance, que ce soit pour franchir des obstacles, éviter des dangers ou atteindre des zones autrement inaccessibles. Le dash permet d'enrichir les possibilités de déplacement.

En plus de ces mécaniques de base, nous avons ajouté des déplacements avancés afin d'enrichir encore plus le gameplay et de rendre les phases de plateforme plus dynamiques.

Notamment, le joueur peut effectuer un wall jump (saut contre un mur) ainsi qu'un wall slide (glissade le long d'un mur). Lorsque le personnage est en contact avec un mur en étant en l'air, il peut s'y appuyer temporairement et

ralentir sa chute. Cela permet au joueur de mieux contrôler sa descente et de se repositionner plus facilement.

Le wall jump permet lui de prendre appui sur un mur pour rebondir dans la direction opposée. Cette mécanique ouvre de nouvelles possibilités en termes de level design, notamment pour atteindre des zones en hauteur ou enchaîner les wall jump dans un couloir vertical.

L'entièreté du système de déplacement du joueur repose sur un fonctionnement basé sur la vélocité, ce qui permet d'obtenir des mouvements plus naturels et progressifs. Plutôt que de déplacer instantanément le joueur à une vitesse fixe, le jeu applique une accélération lorsque le joueur commence à se déplacer, puis fixe une décélération lorsqu'il s'arrête. Cela donne une sensation de poids et d'inertie, rendant les déplacements plus fluides et réalistes.

Ce système s'applique à la fois aux mouvements horizontaux et aux mouvements verticaux. Par exemple, la gravité (variable PLAYER_FALL_ACCELERATION) influence progressivement la chute du joueur, ce qui signifie qu'il accélère en tombant jusqu'à atteindre une vitesse maximale. De la même manière, certaines actions comme le saut, le dash ou le wall jump modifient directement cette vélocité.

2.9. Système de sauvegarde

Afin de garantir la persistance des données entre les sessions de jeu, nous avons mis en place un système de sauvegarde automatique reposant sur le format JSON. Ce choix s'explique par la lisibilité du format, sa compatibilité avec Python via le module `json` de la bibliothèque standard, et sa légèreté pour des données de jeu simples.

L'architecture du système est écrite dans une classe dédiée, `SaveManager`, respectant ainsi le principe de séparation des responsabilités. Cette classe prend en charge trois fonctionnalités principales : la sérialisation de l'état du jeu (conversion des objets Python en texte JSON), la désérialisation lors du chargement (reconstruction des objets Python à partir du fichier JSON lu), et le déclenchement automatique de la sauvegarde à intervalle régulier.

Les données sauvegardées comprennent la position et les points de vie du joueur, l'index du niveau en cours, ainsi que des métadonnées telles que l'horodatage et le temps de jeu cumulé. La sauvegarde est écrite dans un fichier `save.json` placé dans un sous-dossier `saves/` créé automatiquement si absent. (cf. figure 16)

L'autosauvegarde est déclenchée toutes les 300 frames, soit environ toutes les cinq secondes à 60 FPS, grâce à un compteur incrémenté à chaque appel de la méthode `update()`. Le joueur peut également déclencher manuellement une sauvegarde ou un chargement via les touches F5 et F9. Ce système est volontairement limité au mode hors-ligne, le mode multijoueur nécessitant une gestion de la progression côté serveur.

3. Réalisation

3.1. Fonctionnalités jouables actuelles

Pour cette deuxième soutenance, le joueur peut lancer le jeu et choisir plusieurs options dans le menu d'accueil avec le thème musical principal du jeu en arrière plan.

Il peut choisir entre créer une partie en solo (hors ligne), héberger une partie ou rejoindre une partie déjà hébergée ailleurs par une autre machine du même réseau.

Les joueurs peuvent donc créer une partie avec un joueur serveur, et des joueurs clients. Dans la partie multijoueur, les joueurs peuvent se voir et se déplacer en utilisant des mouvements tels que, la course (gauche / droite), le saut, le double saut, dash, le wall jump et le wall slide.

Les joueurs se retrouvent dans la map que nous avons créé de toute pièce et qui a été chargée avec les textures correspondantes dans le jeu. Ils peuvent changer de zone en traversant les “portes” (zones désignées permettant de changer de zone) et explorer la map.

La map contient également des IA simples de patrouille qui font un gauche droite sur une zone prédéfinie et qui changent de direction si elles rencontrent un mur / un obstacle.

Chaque joueur possède également une variable représentant sa vie avec des fonctions permettants d’infliger des dégâts et de redonner de la vie (ces fonctions ne sont cependant pas encore implémentés dans les éléments du jeu, c’est à dire que les fonctions sont présentes et fonctionnelles mais ne sont jamais appelées pour le moment).

3.2. Problèmes & Solutions

Lors de la création des premiers designs du personnage principal, nous avons remarqué que le niveau de détails ne convenait pas à nos attentes pour le jeu. Nous avons donc pris la décision de passer d'animations de 32x32 pixels a 64x64. Ces animations étant dessinées à la main, nous avons alors choisi de garder un design simple et efficace afin de ne pas perdre trop de temps.

Nous avons rencontré des conflits Git au cours du développement. Cela signifie que plusieurs membres de l'équipe ont modifié les mêmes fichiers en parallèle, puis ont effectué des commits de leur côté. Lors de la mise en commun des modifications (merge), Git n'était plus capable de déterminer automatiquement quelle version du code devait être conservée.

Ces conflits nous ont obligés à intervenir manuellement pour comparer les différentes versions du code et choisir les modifications à garder, voire les combiner. Cette étape est délicate car une mauvaise résolution de conflit peut entraîner des bugs ou la suppression de certaines fonctionnalités.
Cette difficulté nous a aussi permis d'apprendre à mieux utiliser Git.

4. Synthèse des expériences individuelles

4.1. Elyas

Au cours de ce projet, j'ai dû réapprendre certaines compétences techniques qui s'étaient estompées avec le temps, notamment le HTML pour la création du site web de présentation. Cette remise à niveau, bien que nécessaire, a engendré une perte de temps non négligeable dans la phase de conception du site. En parallèle, j'ai découvert et pris en main GitHub, outil que je n'avais jusqu'alors jamais utilisé de manière sérieuse. La gestion de versions, le système de branches et la synchronisation du travail en équipe via des dépôts distants ont constitué un apprentissage à part entière, mais se sont rapidement révélés indispensables à la bonne organisation du projet.

Sur la partie gameplay, j'ai réfléchi à un schéma de contrôles simples afin de garantir un confort optimal au joueur. Le choix des touches Q et D pour le déplacement horizontal, de la barre espace pour le saut et de la touche Shift pour le dash découle d'une inspiration des standards du genre. J'ai ensuite enrichi ce système de contrôles en implémentant un mécanisme de wall jump, permettant au joueur de rebondir contre les murs pour enchaîner les déplacements verticaux, ainsi qu'un wall slide, qui ralentit la chute du joueur lorsqu'il est plaqué contre une paroi et maintient la direction correspondante.

Enfin, la réalisation la plus complexe de ma contribution a été la mise en place d'un système de sauvegarde automatique. Pour cela, j'ai créé un nouveau fichier, `save_manager.py`, contenant une classe `SaveManager` et ses méthodes. Cette classe gère la sérialisation de l'état du jeu vers un fichier JSON, sa désérialisation au chargement, ainsi que le déclenchement automatique de la sauvegarde à intervalle régulier.

4.2. Jonathan

J’ai pu grandement participer au début de la création du jeu. En effet, lors de mon année de terminale, j’ai créé un jeu vidéo dans le cadre d’un projet et j’ai donc une petite expérience sur le sujet. J’ai ainsi pu expliquer à mes camarades comment bien structurer leur code ou encore les étapes à suivre lors du développement.

Pour les aider davantage (hormis la création de décors et d’animations), j’ai appris à corriger les erreurs qui rendent notre code inopérant. Cela peut être qualifié de debugging. J’ai également appris le fonctionnement d’une map et des hitbox, et je compte approfondir la création d’animations.

Depuis la première soutenance, les animations ont grandement gagné en qualités. Chaque mouvement spécifique du joueur a sa propre animation ce qui rend le gameplay actuel beaucoup plus fluide. L’ajout des IA ainsi que de leur design rendent également la partie beaucoup plus vivante et permettent de se sentir dans un vrai environnement de jeu vidéo, bien que le fond ne soit pas encore présent pour plonger le joueur dans l’ambiance convenue pour Abyssal Ascension. J’ai pu découvrir lors de mes essais que la texture des monstres était importante. En effet, un mauvais mélange de couleur fait perdre toute crédibilité a une créature hostile, et par conséquent mes premiers essais de texture étaient un échec. Les couleurs plus sombres que j’ai choisies iront de paire avec la création du fond, ce qui fera en sorte de garder un équilibre cohérent dans les graphismes du jeu.

L’unique problème que je trouve à la direction artistique est que le travail reste un peu toujours le même, et par conséquent je me suis intéressé au comportement IA afin d’aider mes collègues dans la création de ces dernières.

4.3. Frédéric

Ces 6 mois de travail sur le projet m'ont permis de découvrir les bases du world building et de la conception de niveaux pour un jeu vidéo. Il m’a permis de

penser comme un concepteur. Il ne suffit pas seulement de créer un beau décor, il fallait qu'il soit fonctionnel pour le joueur et techniquement compatible avec le code.

J'ai dû apprendre à utiliser l'outil LDtk, un éditeur de niveaux externe, afin de concevoir et structurer les environnements du jeu. Cela m'a permis de mieux comprendre comment un niveau est construit, du placement des tuiles jusqu'à la définition des zones de collision, en passant par la gestion des différentes couches graphiques, comme le fond ou les plateformes avec lesquelles le joueur interagit.

Concernant l'intelligence artificielle, j'ai pu appréhender les principes fondamentaux qui régissent le comportement des ennemis dans un jeu vidéo. Même si l'IA que nous avons développée reste très simple pour l'instant, sa conception m'a amené à réfléchir à des notions telles que la gestion des états, la détection de l'environnement ou encore la prise de décision automatique, ce qui me permettra par la suite du projet a l'appliquer sur des ennemis plus complexes. J'ai notamment dû concevoir un système de patrouille permettant à l'ennemi de se déplacer de manière autonome tout en réagissant aux obstacles qui se présentent à lui. Cette expérience m'a donné un premier aperçu concret de ce que représente le développement d'une IA dans un contexte de jeu.

4.4. Clément

Après 6 mois à travailler sur le projet, j'ai appris énormément de choses sur la partie multijoueur des jeux vidéo en LAN (Local Area Network). En effet, j'ai dû apprendre les bases de la librairie python socket afin de transmettre des données entre deux machines du même réseau. Cela m'a permis de mieux comprendre comment les jeux multijoueur gèrent la communication entre plusieurs joueurs, notamment l'envoi et la réception d'informations comme les positions, les actions ou encore les états du jeu.

Cependant, cette partie du projet a également apporté son lot de difficultés. Par exemple, nous avons rencontré des problèmes liés à l'utilisation du protocole TCP. Contrairement à ce que l'on pourrait penser, TCP ne garantit pas que les messages envoyés soient reçus en un seul morceau. Il peut arriver qu'un message soit découpé en plusieurs parties lors de la transmission, ou au contraire que plusieurs messages soient regroupés ensemble à la réception. Cela m'a obligé à réfléchir à une manière de reconstruire correctement les données reçues.

Ces 6 mois m'ont également permis de travailler sur la théorie musicale et la composition de musique. J'ai dû réfléchir aux choix de tonalités, d'instruments et d'ambiances afin que la musique corresponde à l'univers du jeu.
En même temps, nous avons cherché différentes solutions pour produire nous-mêmes des effets sonores, ce qui nous a demandé de faire preuve de créativité.

Ce projet m'a également appris à chercher des solutions face aux nombreux problèmes rencontrés durant le développement. Que ce soit des bugs techniques, des difficultés de conception ou des contraintes liées au travail en groupe. Cela nous a aussi appris à mieux travailler en équipe, à communiquer et à nous mettre d'accord sur les différentes fonctionnalités et décisions à prendre pour le projet.

Enfin, cette expérience m'a permis de mieux comprendre l'ensemble des étapes de création d'un jeu vidéo, des premières idées à leur implémentation.

5. Prévision

5.1. Accord avec le planning

Tâches	Progression actuelle soutenance 2	Progression visée soutenance 2
World building	70%	70%
Character Design	80%	90%
Musique	70%	70%
IA	50%	60%
Contrôles joueurs	100%	100%
FX	70%	70%
Physique et collisions	100%	100%
Playtest	70%	50%
Interface / UI	80%	80%
Optimisation	80%	80%
Multijoueur	80%	65%
Système de sauvegarde	75%	70%
Site web	70%	50%

% = en retard
% = dans les temps
% = en avance

5.2. World Design

Pour le world design, on devra ajouter de nouvelles zones et également ajouter plus de détails aux zones actuelles.

Les nouvelles zones devront contenir des mécaniques que nous avons récemment ajoutées telles que le wall jump et le dash. Ces zones devront être accessibles qu'en utilisant ces capacités afin d'avoir une map un minimum exigeante.

Intégrer les différentes zones et les transitions entre les zones.

5.3. IA

Pour l'IA, il faudra ajouter des ennemis capables de pourchasser le joueur quand il entre dans une zone et de l'attaquer automatiquement.

Il faudra que les IA (ennemis) soient capable de faire des dégâts par les joueurs et de mourir / disparaître.

5.4. Multijoueur

Pour le multijoueur, il va falloir ajouter d'autres informations à transmettre en plus de la position des joueurs comme leur arme actuelle, leur vie, leur état (en saut / en dash), la position des ennemis et leur statut.

Il va aussi falloir implémenter un système de déconnexion (voulue, le joueur se déconnecte ou non voulue, le joueur perd la connexion).

5.5. Menus & Interface utilisateurs

Pour les menus et l'interface utilisateurs, il faudra retirer les informations de débogage une fois qu'elles seront inutiles.

Il faudra également créer un menu pause, une interface pour que le joueur ait un moyen d'accéder aux informations de son joueur.

5.6. Textures & Animations

L'objectif est de rendre chaque animation plus fluide en lui ajoutant 2 à 3 sprites.

Il faudra aussi s'occuper des graphismes de fonds, (ex : menu pause, lors des dialogues et autres). Les graphismes des premiers ennemis sont également prévus, avec celui du premier boss si possible.

5.7. Son & Musique

Pour les sons et les musiques, il faudra ajouter plus de bruitages (effets sonores) pour de nouvelles actions comme l'attaque, la mort du joueur et les sons de l'interface (son joué sur un clic de bouton par exemple).

Il faudra également composer plusieurs autres musiques originales pour d'autres zones et contextes (dialogue, combat, animation…).

5.8. Site Web

Pour le site web, il faudra rajouter du contenu tel que les captures d'écrans du jeu, un lien d'installation et un guide d'installation.

Il faudra également le rendre plus élégant sur interfaces mobiles (en ce moment, l'interface est légèrement endommagé quand ouverte sur téléphone portable).

5.9. Contrôles

Pour les contrôles, il va falloir ajouter la possibilité d'attaquer en ajoutant une nouvelle touche pour le combat.

Pour la suite du jeu, il est prévu que les combats se basent sur différentes utilisations du même objet, ce qui inclut donc des combinaisons de touches ou actions à coder.

5.10. Système de sauvegarde

Il faudrait pour finir pouvoir sauvegarder certaines données qui ne sont pas encore présentes dans le jeu telles que les niveaux et les objets possédés par le joueur.

6. Conclusion

Notre travail depuis le début de ce projet s'est réalisé sans accroc et les objectifs fixés ont été respectés dans leur ensemble malgré la charge de travail.
Chacun continue de bien travailler au sein du groupe et l'aide est apportée dès que nécessaire si un des membres de l'équipe à des soucis durant son travail.
La charge de travail restante est tout de même très importante, mais l'avancée de la partie fonctionnelle et programmation va nous permettre de passer rapidement aux

étapes suivantes afin de se rapprocher de l'objectif final du projet avant la date d'échéance.

7. Annexes

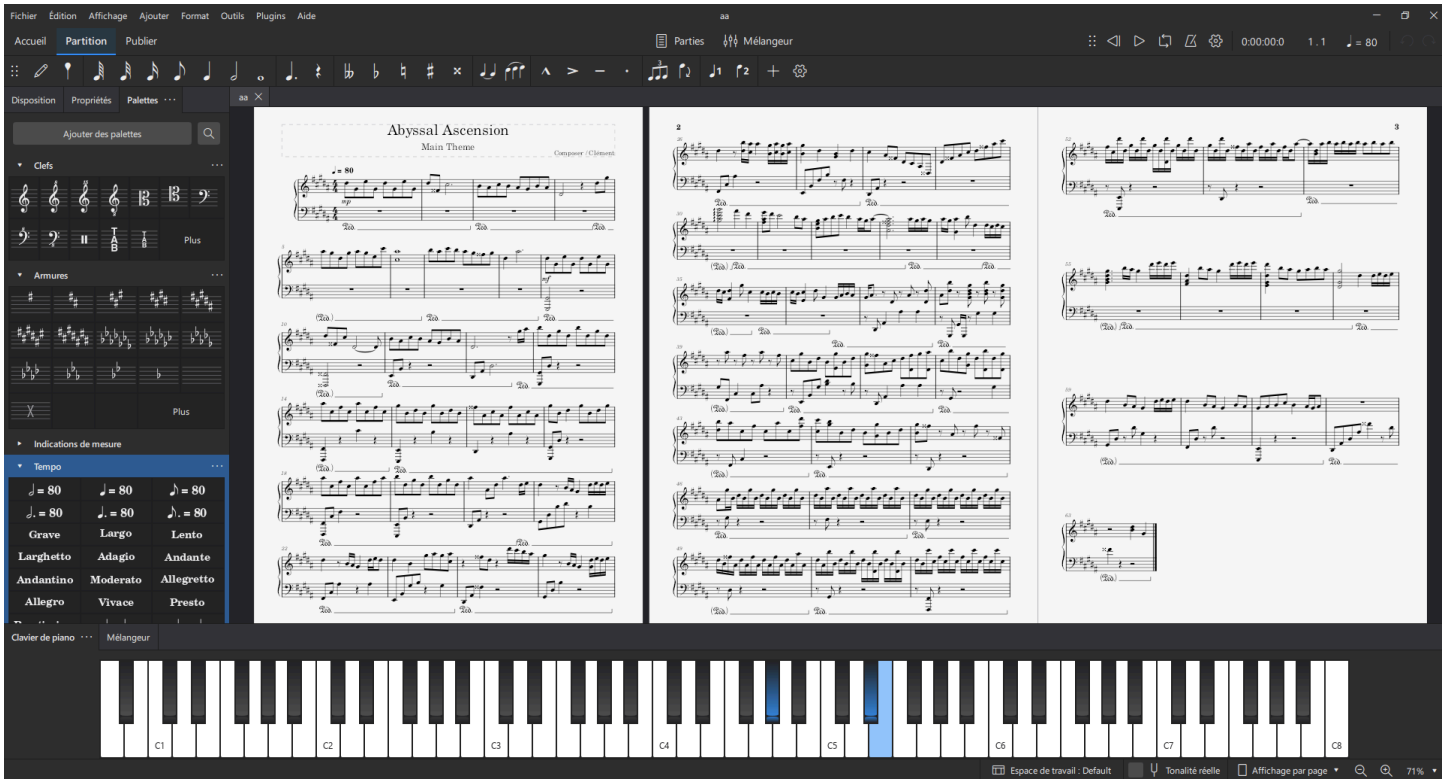


Figure 1 : Partition de musique du thème principal composée sur MuseScore.

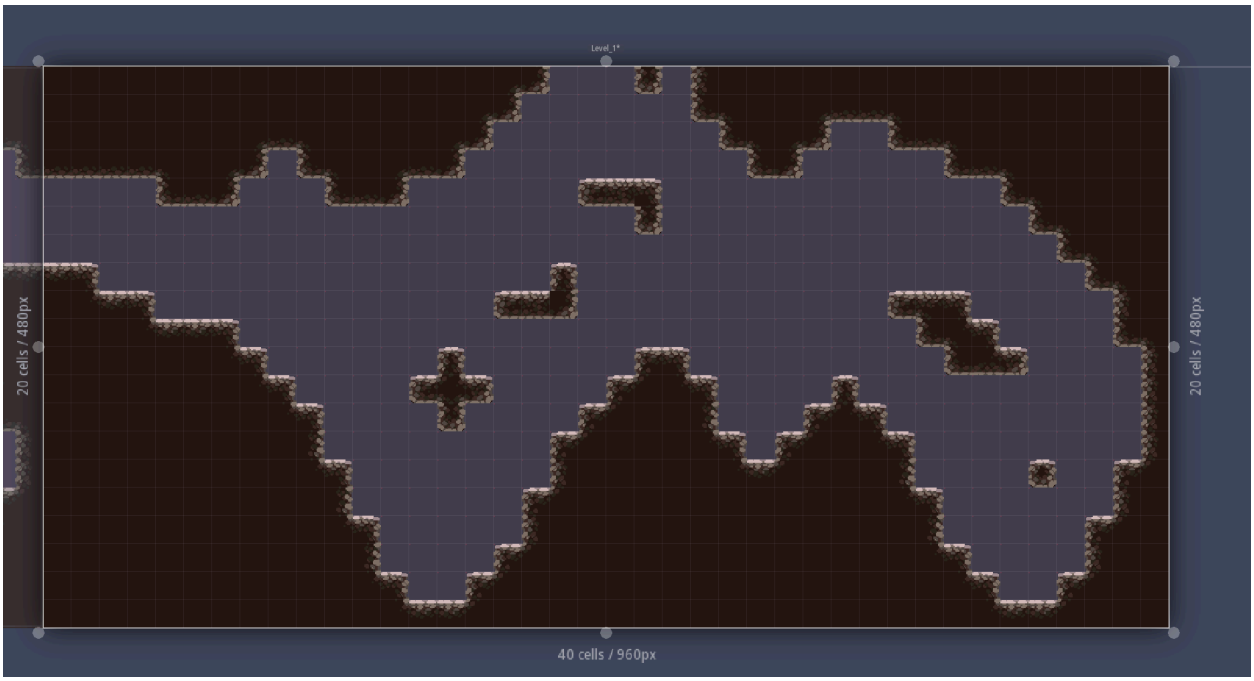


Figure 2 : Première zone jouable

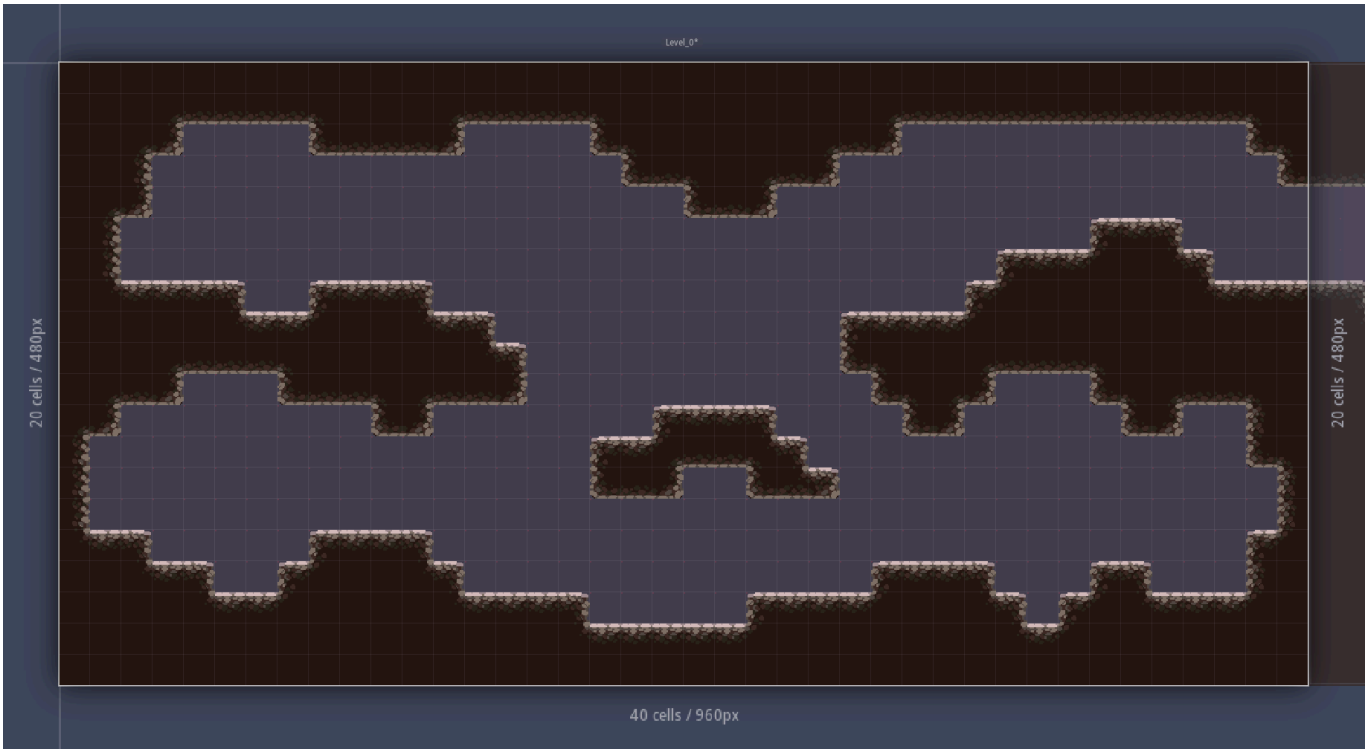


Figure 3 : 2e zone jouable

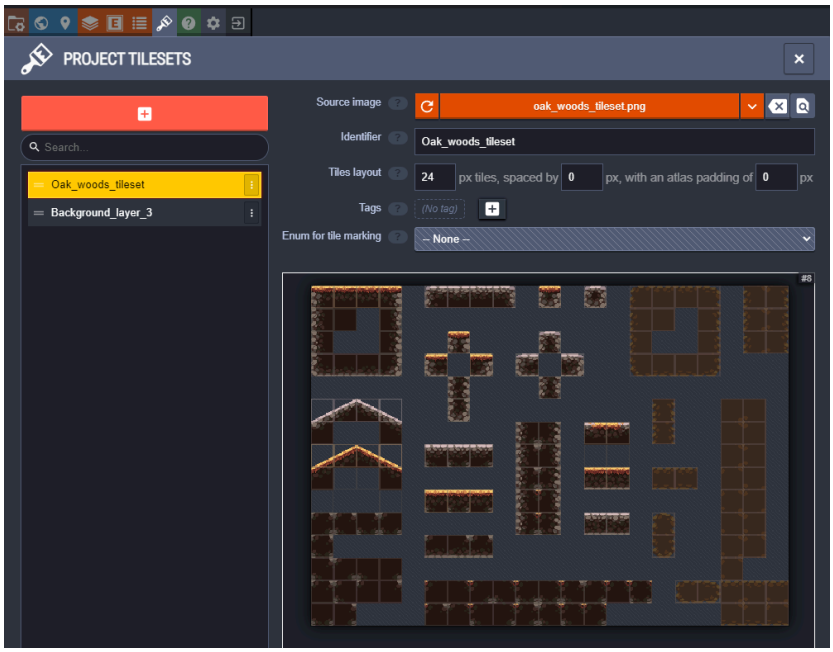


Figure 4 : Interface de configuration des tilesets sous LDtk

Figure 5 : Map globale du jeu sur LDtk.

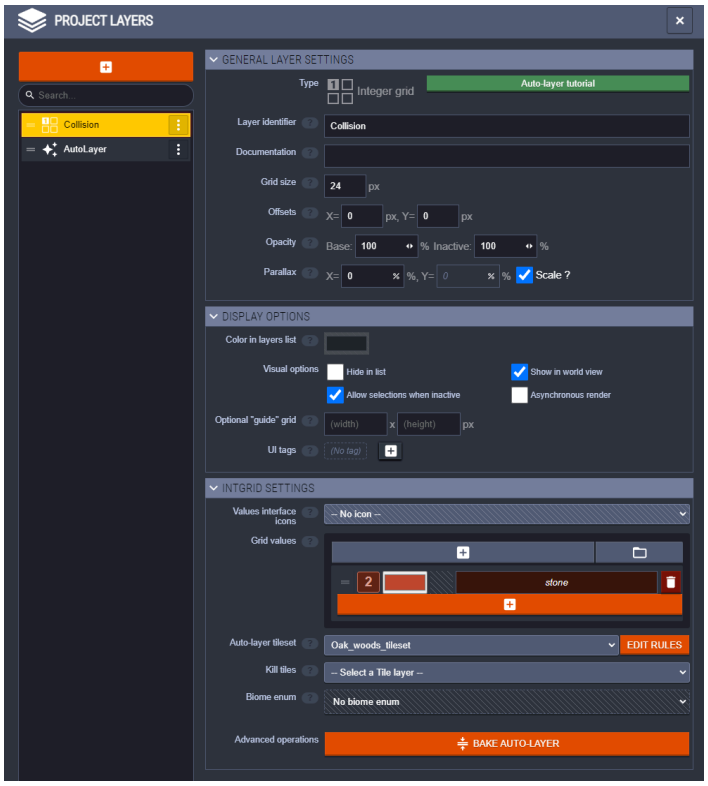


Figure 6 : Interface de gestion du World Building et des collisions

Figure 7 : Interface de configuration des motifs pour le rendu automatique du décor

Figure 8 : Configuration des paramètres globaux du monde et du niveau dans LDtk

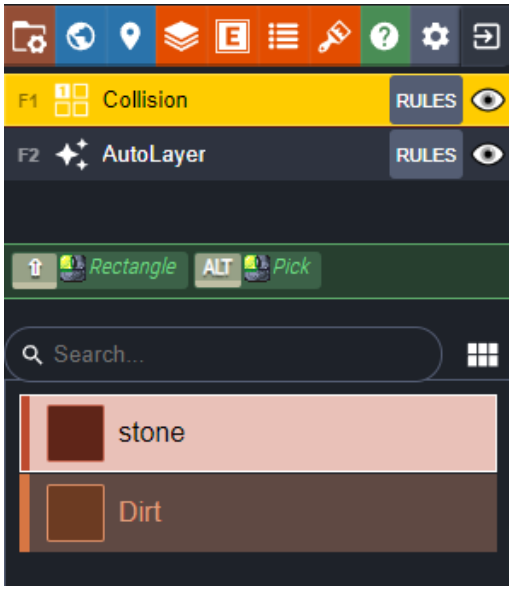


Figure 9 : Interface de sélection des matériaux pour le world building.

Figure 10 : Création d'un sprite pour le personnage principal.

Figure 11 : Création d'une animation de course en enchaînant rapidement plusieurs sprites afin de créer du mouvement.

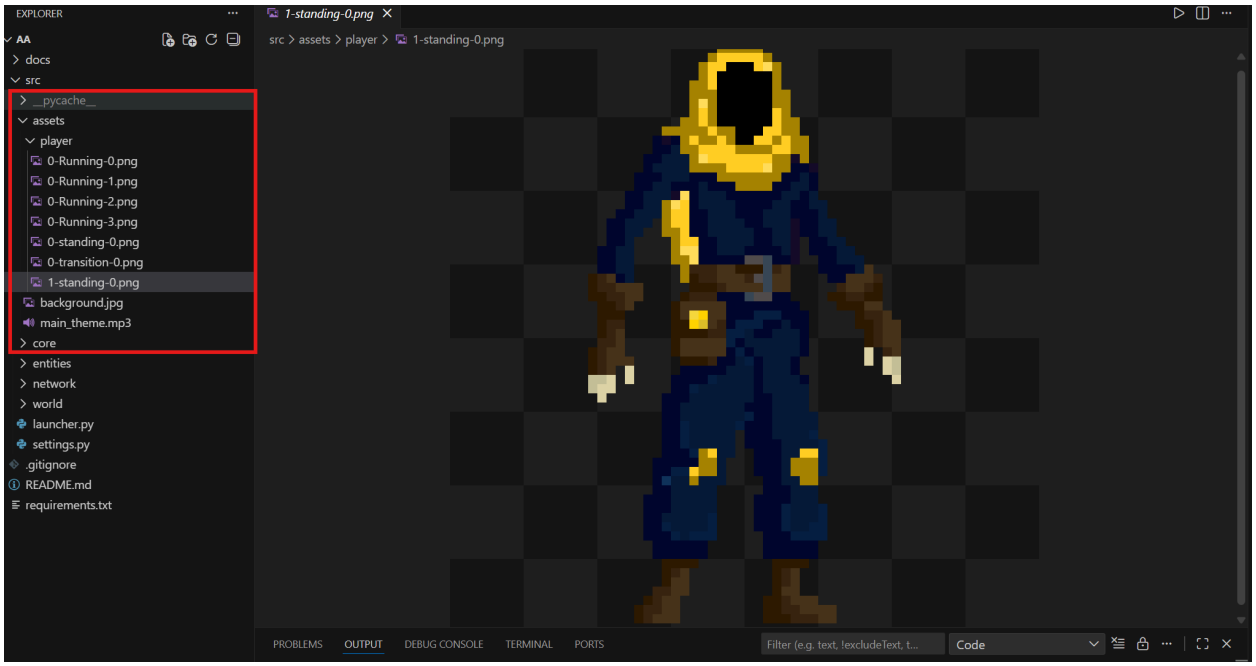


Figure 12 : Vue de l'organisation des fichiers pour les animations dans l'éditeur.

Figure 13 : Player 1 Player 2

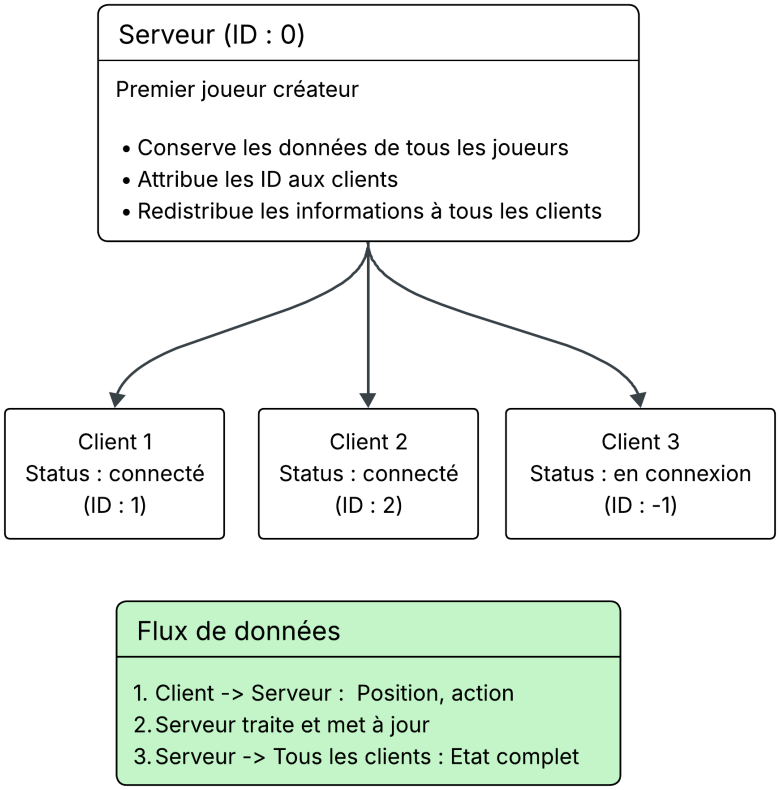


Figure 14 : Schéma représentatif du système multijoueur client-serveur

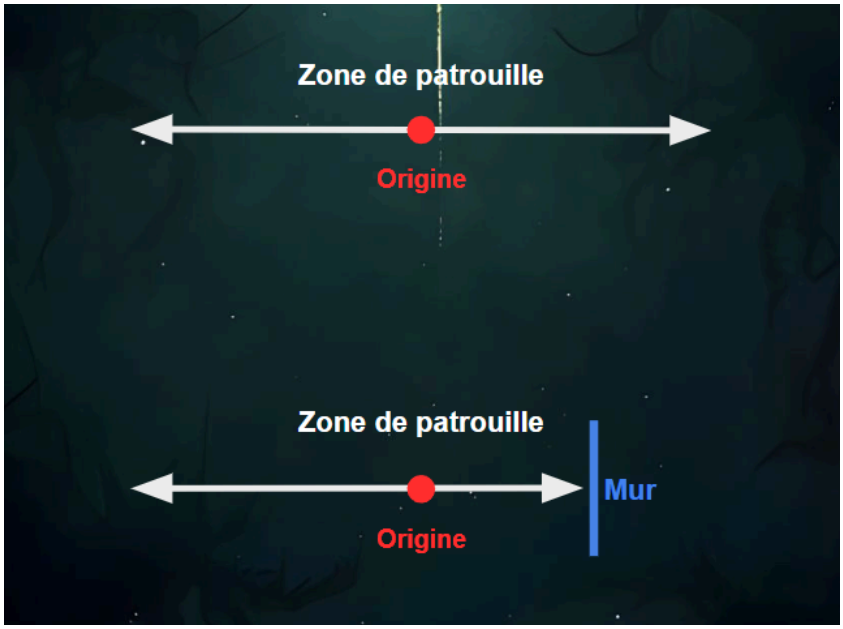
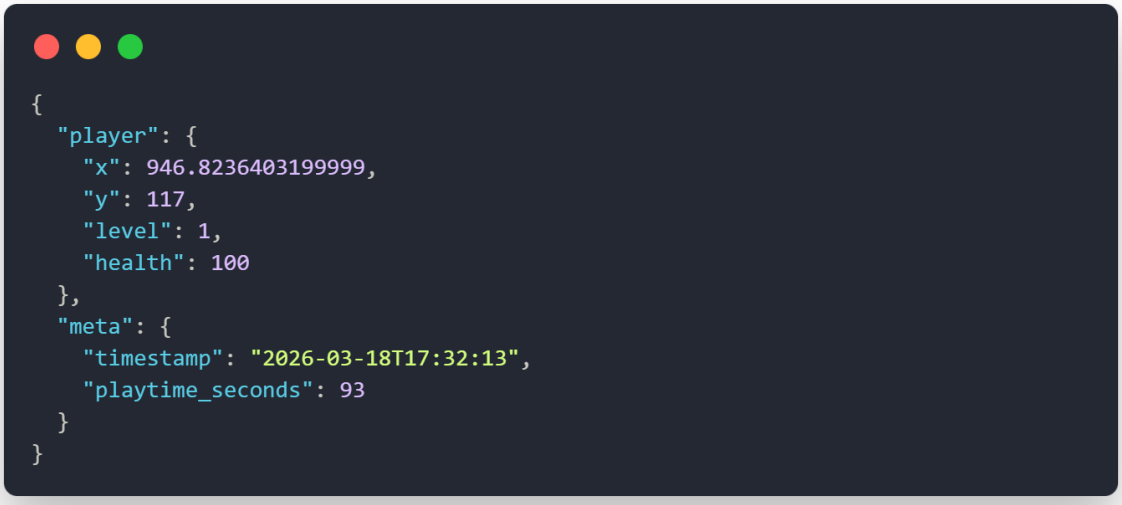


Figure 15 : Schéma de la patrouille de l'IA



```
{
  "player": {
    "x": 946.8236403199999,
    "y": 117,
    "level": 1,
    "health": 100
  },
  "meta": {
    "timestamp": "2026-03-18T17:32:13",
    "playtime_seconds": 93
  }
}
```

Figure 16: Exemple de fichier save.json