UNDERGROUND

SOMMAIRE

Introduction générale

Dans le cadre de la SAE J3D 2025 de première année à l'EPITA, il nous a été demandé de réaliser un projet informatique de longue durée, consistant en la conception et le développement d'un jeu vidéo en groupe de quatre à cinq étudiants. Ce projet s'étend sur l'ensemble de l'année scolaire et a pour objectif de mettre en pratique les connaissances acquises en cours et en travaux pratiques, tout en développant des compétences supplémentaires liées au travail en équipe, à la gestion de projet et à la conception logicielle.

Notre groupe, composé de quatre étudiants, a choisi de développer un jeu vidéo intitulé Underground. Il s'agit d'un jeu de survival-horror en 2D, programmé en Python à l'aide du framework PyGame, conformément aux contraintes imposées par le sujet. Le projet intègre également les éléments obligatoires demandés, notamment une intelligence artificielle simple et un mode multijoueur en réseau, basé sur une architecture client/serveur en réseau local (LAN).

L'objectif principal de ce projet est de concevoir un jeu fonctionnel, cohérent et techniquement maîtrisé, tout en respectant les contraintes de performance et de faisabilité liées à l'environnement Python. Le jeu repose sur un gameplay orienté survie, sans combat direct, mettant l'accent sur la fuite, la discrétion et la coopération entre joueurs. Cette approche permet à la fois de proposer une expérience originale et de rester réaliste quant aux moyens techniques et humains dont dispose le groupe.

La première étape du projet a consisté en la rédaction d'un cahier des spécifications techniques, document fondamental visant à définir précisément le concept du jeu, ses mécaniques, son architecture technique et l'organisation du travail au sein de l'équipe. Ce document a servi de base pour le démarrage du développement et pour la répartition des tâches entre les membres du groupe.

Ce rapport de soutenance a pour but de présenter l'avancement actuel du projet, les choix techniques et conceptuels réalisés, ainsi que le travail individuel de chaque membre du groupe. Il est structuré de manière à offrir une vision globale du projet, puis à détailler les contributions personnelles, conformément aux attentes de la SAE J3D. Enfin, il permet de mettre en évidence les objectifs à court et moyen terme en vue des prochaines étapes du développement.

Présentation du jeu

Cette partie correspond à la présentation synthétique du projet, telle qu'elle peut être utilisée à l'oral ou dans le rapport.

Présentation générale

Underground est un jeu vidéo de survival-horror en 2D, développé en Python avec PyGame.

Il est jouable en solo ou en coopération jusqu'à trois joueurs via un réseau local (en LAN).

Le jeu se déroule dans des salles générées aléatoirement, chacune contenant des ennemis, des objets ou des énigmes. L'objectif est de survivre et de s'échapper.

Gameplay

Le gameplay repose sur :

- la fuite,
- la discrétion,
- l'observation,
- et la coopération.

Les joueurs doivent utiliser l'environnement pour éviter les monstres et progresser.

Multijoueur en LAN

Le jeu fonctionne selon une architecture client/serveur :

- le serveur gère la logique du jeu, l'IA et la génération,
- Le client gère l'affichage, les entrées et le son.

  Ce choix permet une synchronisation efficace et répond aux exigences du projet EPITA.

Cahier des charges technique

1 Choix techniques généraux

Le projet Underground est développé en Python, en utilisant le framework graphique PyGame. Ces choix techniques ont été faits en priorité afin de respecter les contraintes imposées par la SAE J3D, mais également pour assurer la faisabilité du projet sur la durée. Le choix de ces technologies permet de se concentrer sur la logique du jeu, l'architecture logicielle et les mécaniques de gameplay, plutôt que sur des problématiques trop complexes liées au moteur.

2 Utilisation de Python et PyGame

PyGame est utilisé pour :

- l'affichage des éléments graphiques (joueurs, ennemis, objets, décor),
- la gestion des entrées utilisateur (clavier),
- la gestion des collisions et des hitbox,
- la lecture des sons et musiques,
- la gestion du temps et des FPS côté client.

Le jeu est conçu pour fonctionner sous Windows 10/11, comme exigé par le cadre du projet. Les performances ont été prises en compte dès la phase de conception, afin de garantir une expérience fluide malgré les limites de Python.

Le rendu graphique repose sur un style 2D en pixel art, ce qui permet de limiter la charge graphique tout en conservant une identité visuelle forte et cohérente avec l'ambiance du jeu.

3 Architecture client / serveur

Le jeu fonctionne de la manière suivante :

- un serveur est lancé par un joueur hôte,
- les autres joueurs se connectent en tant que clients,
- le serveur centralise la logique du jeu,
- La partie client se charge de l'affichage et des interactions utilisateur.

Cette séparation permet :

- une meilleure synchronisation entre joueurs,
- un contrôle centralisé des règles du jeu,
- une réduction des risques de désynchronisation,
- une architecture claire et évolutive.

4 Réseau LAN : sockets et échanges JSON

Ce choix est motivé par la simplicité de mise en œuvre et la fiabilité des échanges dans un environnement local, sans dépendance à une connexion Internet externe.

Les données échangées entre le serveur et les clients sont encodées au format JSON. Ce format a été choisi car il est :

- lisible,
- simple à sérialiser et désérialiser en Python,
- adapté à l'échange de structures de données (positions, états, événements).

Les messages échangés comprennent notamment :

- la position et l'état des joueurs,
- l'état des ennemis,
- les informations de salle,
- les actions des joueurs.

Ce système permet de maintenir une cohérence globale de la partie tout en limitant le volume de données échangées.

5 Répartition des responsabilités serveur / client

La répartition des rôles entre le serveur et les clients a été définie dès le cahier des charges afin d'éviter toute ambiguïté lors du développement.

Serveur

Le serveur est responsable de :

- la génération aléatoire des salles,
- la gestion de l'intelligence artificielle des ennemis,
- la logique du jeu (règles, progression, défaite),
- la gestion des shops et des objets,
- la synchronisation des états entre joueurs,
- la collecte des statistiques de fin de partie.

Client

Le client est responsable de :

- l’affichage graphique,
- la gestion des entrées clavier,
- la lecture des sons et musiques,
- l’affichage du HUD (interface),
- l’envoi des actions du joueur au serveur.

Cette séparation garantit une architecture propre et facilite le débogage ainsi que les évolutions futures.

6 Contraintes techniques identifiées

Plusieurs contraintes techniques ont été identifiées et intégrées au cahier des charges :

- Nombre de joueurs : limité à 3 joueurs maximum, afin de réduire la charge réseau et la complexité de synchronisation.
- Tick serveur : fixé à 25 mises à jour par seconde, offrant un bon compromis entre réactivité et stabilité.
- Fréquence d’affichage client : 60 FPS.
- Stockage limité : nécessité d’optimiser les ressources graphiques et sonores.
- Perte d’un membre du groupe (Clément Sappey) en fin décembre.

Ces contraintes ont fortement influencé les choix de conception et de priorisation des fonctionnalités ainsi que la réorganisation des tâches.

7 Justification globale des choix techniques

L’ensemble des choix techniques présentés dans ce cahier des charges vise à trouver un équilibre entre :

- les exigences pédagogiques de la SAE J3D,
- la faisabilité du projet,
- la stabilité du jeu,
- et l’ambition du concept.

Travail individuel

I. Benjamin Zanibellato

Rôle et responsabilités dans le projet

Dans le cadre du projet *Underground*, j'ai principalement travaillé sur la conception globale du jeu, la rédaction du cahier des charges, ainsi que sur la création et la structuration du lore et de l'univers narratif.

Dès les premières phases du projet, mon objectif a été de poser une vision claire et cohérente du jeu afin de servir de base au travail technique du reste de l'équipe. Cette phase de conception a permis d'anticiper les contraintes imposées par le cadre EPITA tout en garantissant la faisabilité du projet sur le long terme.

1. Conception et rédaction du cahier des charges

Objectifs

Le cahier des charges avait pour but de :

- définir précisément le concept du jeu,
- encadrer le développement technique,
- respecter les contraintes du projet SAE J3D (Python, PyGame, IA, multijoueur réseau),
- assurer une vision commune au sein du groupe.

Il s'agissait de transformer une idée de jeu en un document structuré, clair et exploitable.

Mécaniques principales définies

Les mécaniques détaillées dans le cahier des charges incluent :

- Déplacements : déplacement fluide en 2D et gestion du positionnement.
- Système de survie : points de vie, dégâts ou mort instantanée selon les ennemis.
- Cachettes : possibilité de se dissimuler dans l'environnement.
- Objets : pièces, soins, bonus de vitesse, boucliers.
- Économie : shops générés dans certaines salles.
- Ennemis :
    - monstres mobiles (patrouille, détection),
    - monstres immobiles (gardiens de salle),
    - monstre traversant l'intégralité de la salle.
- HUD : affichage des informations essentielles (vie, monnaie, salle et niveau de difficulté).

Ces mécaniques ont été pensées pour renforcer la tension sans alourdir inutilement la complexité du jeu.

Contraintes identifiées

Le cahier des charges prend également en compte plusieurs contraintes :

- stockage limité avec PyGame,
- performances réseau (tick serveur, FPS),
- niveau technique hétérogène du groupe.

Ces contraintes ont directement influencé les choix de conception, notamment le nombre limité de joueurs et la simplicité volontaire de certaines mécaniques.

2. Création du lore et de l’univers narratif

Objectifs narratifs

Le lore du jeu n’est pas omniprésent. Il a été conçu pour :

- poser un contexte crédible,
- renforcer l’ambiance oppressante,
- laisser une liberté d’interprétation au joueur.

L’histoire sert avant tout de support à l’immersion.

Histoire du jeu

Le personnage principal part faire du camping lorsqu’il se perd dans la forêt en allant chercher du bois. Un orage éclate soudainement, l’obligeant à se réfugier dans une grotte.Un éboulement survient et l’entrée s’effondre, le piégeant à l’intérieur. Assommé, il perd connaissance.Il se réveille plusieurs jours plus tard, désorienté, et tente de trouver une issue. Le sol se dérobe alors sous ses pieds, le faisant chuter dans des souterrains inexplorés depuis des années. Son seul objectif devient alors de trouver la sortie.

Ambiance et direction artistique

Cette histoire influence directement :

- l’ambiance sombre et claustrophobe,
- la visibilité réduite,
- le sentiment d’isolement.

Le choix du pixel art permet de rester cohérent avec les contraintes techniques tout en renforçant l’atmosphère du jeu.

3. Conception des objets, mécaniques interactives et système économique

En complément du travail de conception globale, j'ai également participé activement à l'élaboration des mécaniques d'objets et du système économique du jeu.

L'objectif était d'enrichir l'expérience de survie en proposant des éléments interactifs cohérents avec le gameplay et favorisant la progression des joueurs.

J'ai ainsi conçu plusieurs types d'objets utilisables par les joueurs :

Bouclier

Le bouclier est un objet défensif achetable dans la boutique. Lorsqu'un joueur en possède un, celui-ci absorbe une attaque d'un monstre à sa place.

Après avoir bloqué un coup, le bouclier se brise et disparaît.

Cette mécanique permet d'offrir une seconde chance au joueur sans rendre le gameplay trop permissif.

Boost de vitesse

Cet objet améliore légèrement la vitesse de déplacement du joueur. Il permet d'augmenter les chances de fuite face aux ennemis et facilite l'exploration des salles.

L'amélioration reste volontairement modérée afin de préserver l'équilibrage du jeu.

Boost de vie

Le boost de vie permet de régénérer une partie des points de vie du joueur sans jamais dépasser la limite maximale fixée à 100 points de vie.

Il s'agit d'un objet stratégique permettant aux joueurs de se maintenir en état de survie lors des phases d'exploration prolongées.

Potions et soins

Un système de potions à efficacité variable a été mis en place afin d'introduire une dimension stratégique liée aux ressources :

- Petite potion : restaure 20 points de vie,
- Potion moyenne : restaure 40 points de vie,
- Grande potion : restaure tous les points de vie.

Le prix des potions varie en fonction de leur puissance, obligeant les joueurs à gérer leurs dépenses avec réflexion.En complément des potions, les joueurs peuvent également retrouver directement au sol des soins représentés sous forme de médicaments ou de pilules.Ces objets permettent de récupérer uniquement 20 points de vie et servent principalement à aider les joueurs durant l'exploration sans rendre le jeu trop facile.

Clés

Les clés sont des objets de progression indispensables. Elles permettent d'ouvrir des portes verrouillées bloquant l'accès à certaines zones.Ce mécanisme structure l'exploration et permet de rythmer l'aventure en introduisant des objectifs intermédiaires.Les clés ne peuvent pas être achetées dans le shop.

4. Mécaniques de cachette et menace dynamique

Afin de renforcer la tension et l'aspect survival-horror, j'ai conçu un système de cachettes intégrées à l'environnement.

Armoires

Les armoires permettent aux joueurs de se dissimuler pour éviter certains ennemis.Cette mécanique devient essentielle face à un monstre spécifique qui apparaît de manière scriptée et traverse l'intégralité de la carte.Ce monstre est impossible à éviter par la fuite seule : le joueur doit impérativement se cacher dans une armoire pour ne pas subir de dégâts.

Cette mécanique :

- renforce la tension,
- oblige à observer l'environnement,
- favorise l'anticipation et la coopération en multijoueur.

Les armoires apparaissent directement dans certaines salles et ne peuvent pas être achetées.

5. Système économique et boutique

J’ai également participé à la conception du système économique du jeu.

Monnaie du jeu

La monnaie est représentée par des crânes, élément cohérent avec l’ambiance sombre et souterraine du jeu.

Cette ressource est récupérable lors de l’exploration des salles.

Les crânes permettent d’acheter :

- boucliers,
- potions,
- boosts.

Les éléments de cachette comme les armoires ne sont pas achetables, car ils font partie intégrante du décor.

Boutique (Shop)

Une salle spéciale dédiée à la boutique a été développée et le système du shop est désormais terminé et entièrement fonctionnel.Cette zone permet aux joueurs d’acheter leurs objets entre les phases d’exploration.

Le shop comprend :

- une interface d’achat,
- la gestion de la monnaie,
- la disponibilité des objets,
- les interactions avec les joueurs.

Afin d’améliorer l’immersion visuelle et l’ambiance des salles, plusieurs éléments de décor ont également été ajoutés :

- tonneaux,
- coffres,
- décorations souterraines.

Ces éléments permettent de rendre l’environnement plus vivant et cohérent avec l’univers du jeu.

6. Travail graphique

En parallèle du game design, j’ai recherché et sélectionné plusieurs ressources graphiques cohérentes avec la direction artistique du projet :

- graphismes des portes verrouillées et déverrouillées,
- visuels de la monnaie (crânes),
- sprites des armoires,
- spirtes des leviers,
- éléments de décor comme les tonneaux et les coffres.

Ces éléments participent à renforcer l’immersion et l’identité visuelle du projet *Underground*.

II. Emma Villard

1. Mon rôle dans le projet

Dans ce projet, mon rôle principal était la cheffe d'équipe, je devais faire en sorte que le travail en équipe se passe bien. Mais aussi j'étais responsable des graphismes et des monstres, ainsi mon rôle concernait également la conception et l'intégration des éléments visuels liés aux personnages, aux monstres et à l'ambiance générale du jeu. J'ai travaillé sur le sprite du personnage principal, sur les monstres, sur l'intégration des murs, des portes, des objets, du fond de la map et de certains éléments d'interface.
Je me suis aussi occupé d'une partie importante du système de monstres. J'ai participé à leur conception graphique, mais aussi à leur fonctionnement dans le jeu : leur apparition, leurs déplacements, leurs réactions lorsque le joueur est caché, et leurs interactions avec le système de dégâts. Chaque monstre devait avoir une identité propre, à la fois visuelle et comportementale.
J'ai également intégré la musique et participé à l'introduction. Même si le lore a principalement été écrit par Benjamin, j'ai travaillé sur la façon de le rendre présent dans l'expérience du joueur. J'ai aussi participé au menu, notamment aux boutons, à l'entrée du prénom et à l'ambiance générale de l'interface.
Enfin, j'ai effectué plusieurs retouches techniques sur les collisions, les objets et les déplacements. Ces corrections étaient nécessaires pour que les graphismes ne soient pas seulement décoratifs, mais qu'ils fonctionnent réellement avec le gameplay. L'objectif était que le joueur comprenne naturellement ce qu'il voit à l'écran et que les interactions soient cohérentes.

2. Conception du personnage principal et direction artistique

Dans un premier temps, j'ai souhaité me concentrer sur la création des sprites du personnage principal. Mon intention était de concevoir un protagoniste volontairement simple, presque banal, afin qu'il puisse représenter « n'importe qui ». Ce choix n'était pas anodin : je voulais que l'attention du joueur ne soit pas focalisée sur un personnage trop caractérisé ou trop héroïque, mais plutôt sur l'atmosphère générale du jeu et sur l'environnement dans lequel il évolue. Le personnage devait donc s'effacer partiellement au profit de l'ambiance visuelle et narrative. C'est aussi pour cette raison que, sur la jaquette du jeu, le personnage principal n'est pas clairement mis en avant et qu'il ne possède pas de véritable prénom défini.

Vers la mi-novembre, j'ai proposé un premier prototype de personnage à mon équipe. Il s'agissait d'une silhouette très allongée et sombre, rappelant une ombre, avec pour seuls éléments distinctifs de grands yeux lumineux. La toile utilisée faisait 64 x 64 pixels. Ce design volontairement minimaliste et mystérieux renforçait l'aspect inquiétant du jeu et s'intégrait bien à l'univers sombre que nous envisagions. Dans un premier temps, cette proposition a été validée par mes coéquipiers, ce qui m'a permis de poursuivre le travail artistique à partir de cette base.

Cependant, après une réflexion collective, nous avons remarqué que ce sprite correspondait davantage à un monstre qu'au personnage principal. Il donnait une impression trop inquiétante et trop étrange pour permettre au joueur de s'identifier facilement à lui. Nous avons donc décidé de conserver cette idée, mais de la réutiliser pour un ennemi. C'est ainsi que ce premier prototype est devenu la base d'Ombrelame.

J'ai ensuite entrepris de créer un nouveau sprite pour le personnage principal. Cette fois-ci, je me suis inspiré du personnage à capuche de Little Nightmares, dont le design simple mais expressif fonctionne très bien dans un univers sombre. Je me suis aussi appuyé sur différentes références trouvées sur Pinterest. Le nouveau personnage est plus petit que le précédent, avec des proportions plus proches de celles d'un enfant. Ce choix accentue sa vulnérabilité face à l'environnement hostile et aux monstres présents dans les salles.

La capuche est devenue un élément central de son identité visuelle. Elle permet au personnage de rester simple tout en étant reconnaissable. En mode multijoueur, la couleur de cette capuche change selon le joueur, ce qui permet de repérer plus facilement son propre personnage à l'écran. Le joueur principal apparaît avec une

couleur rouge, tandis que les autres joueurs peuvent apparaître en bleu ou en vert. Ce choix améliore la lisibilité du jeu tout en restant cohérent avec la direction artistique.
Au départ, je voulais réaliser un personnage en 32 x 32 pixels. Cependant, il était difficile d'obtenir un rendu satisfaisant avec une taille aussi limitée, surtout pour un personnage animé. Les proportions variaient selon les mouvements, notamment pendant les animations de marche. Finalement, les sprites du personnage principal font en moyenne 50 x 80 pixels selon les frames utilisées.
Dans le jeu, il a fallu adapter cette taille. Si le personnage était affiché à sa taille réelle, il paraissait trop grand par rapport aux décors, aux objets et à la map. Nous avons donc choisi de l'afficher avec une taille de 25 x 40 pixels. Cela permet de mieux l'intégrer à l'échelle générale du jeu. Sa vitesse a également été ajustée : elle était initialement trop élevée, ce qui donnait l'impression que le personnage courait. Nous l'avons donc réduite afin de renforcer l'impression de marche lente. Ce choix participe aussi à l'ambiance angoissante du jeu, car le joueur se sent moins puissant et moins capable de fuir facilement.

3. Réflexion sur les décors et l'ambiance visuelle

Une fois le personnage principal validé, je me suis tourné vers la conception du fond du jeu. Pour nourrir ma réflexion, je me suis largement appuyé sur des recherches visuelles, notamment via Pinterest. Cette plateforme m'a permit de trouver de nombreuses inspirations en lien avec les grottes, les ambiances sombres, les environnements oppressants et les jeux en pixel art. Afin d'organiser ces références, j'ai créé un tableau Pinterest dédié. Cela m'a permis de centraliser mes inspirations et d'assurer une certaine cohérence visuelle tout au long du projet. Ce tableau a également servi de support de discussion avec les autres membres de l'équipe, notamment Benjamin.
Avant même de dessiner les premiers décors, j'ai réfléchi à une palette de couleurs cohérente. Il me semblait essentiel de définir des teintes dominantes afin de donner une identité forte au jeu. J'ai donc choisi des nuances de marron pour évoquer la roche, la terre et l'aspect organique d'une grotte. J'ai aussi utilisé du rouge pour rappeler le sang et renforcer l'aspect violent, inquiétant et dérangeant de l'univers. Enfin, le noir occupe une place très importante, car il accentue la profondeur, le mystère et l'oppression.

4. Conception de la map et adaptation aux écrans

Pour la map, je voulais éviter une map trop rigide ou trop contrainte par les dimensions de l'écran. En parcourant Pinterest, j'ai trouvé une approche visuelle intéressante : ne pas faire correspondre directement les bords de la zone jouable aux bords de l'image. Le principe consiste à placer la zone de jeu au centre, entourée d'un décor plus large et d'une marge sombre.

Ce choix présente plusieurs avantages. Le jeu peut être affiché sur différents écrans sans déformer la zone jouable. Le fond noir agit comme une marge adaptative et permet de conserver une expérience visuelle homogène, quelle que soit la résolution. La zone jouable du jeu mesure 800 x 400 pixels. Comme une tuile fait 20 x 20 pixels, cela correspond à une map de 40 tuiles de largeur et 20 tuiles de hauteur. Cette structure en tuiles facilite la gestion des murs, des portes, des objets et des collisions.

Le fond que j'ai réalisé représentait une grotte. Ce décor devait instaurer immédiatement une atmosphère lourde et inquiétante. J'y ai ajouté des taches de sang afin de suggérer que d'autres personnes étaient déjà passées par cet endroit et qu'il s'y était probablement déroulé des événements violents. Au départ, j'avais envisagé d'utiliser des teintes plus brunâtres pour ces taches, afin de donner l'impression d'un sang ancien et séché. Cependant, lors des tests visuels, ces couleurs se confondaient trop avec la boue. Pour éviter toute ambiguïté, j'ai finalement choisi un rouge plus franc, identifiable immédiatement comme du sang.

Pour obtenir un rendu pixelisé, j'ai d'abord travaillé sur une toile de 272 x 152 pixels. Cependant, je n'étais pas satisfait du rendu final. La map était trop sombre et certains éléments étaient difficiles à distinguer dans le jeu. J'avais aussi essayé de représenter la sortie de la salle sous la forme d'un trou, mais le résultat n'était pas assez joli niveau design. J'ai donc refait la map en choisissant une forme plus simple et plus rectangulaire, ce qui facilitait aussi la création de la zone accessible par le personnage.

Même après cette nouvelle version, je trouvais encore le décor trop simple. Plutôt que de recommencer entièrement une troisième fois, j'ai utilisé une IA pour accentuer et détailler ce que j'avais déjà créé. L'objectif n'était pas de remplacer mon travail, mais de l'améliorer en conservant l'esprit pixel art et l'ambiance initiale. La version finale de l'image de fond mesure 1681 x 936 pixels. Dans le jeu, elle est affichée derrière une zone jouable de 800 x 400 pixels. (Soit la taille de l’écran de jeu)

5. Le design des monstres

1. Ombrelame

Ombrelame est directement issu du premier prototype du personnage principal. Lorsque nous avons réalisé que ce sprite correspondait davantage à un monstre qu'à un protagoniste, nous avons décidé de le réutiliser comme ennemi. Ce choix permettait de ne pas perdre le travail déjà effectué et de donner une identité forte à l'un des monstres du jeu. (Surtout que nous aimons beaucoup ce monstre)

Visuellement, Ombrelame est une silhouette sombre et allongée, avec de grands yeux lumineux. Son apparence évoque une ombre vivante, ce qui correspond bien à son comportement dans le jeu. Il traverse les salles et peut poursuivre le joueur s'il le repère. Son design devait donc inspirer une forme de menace discrète, presque silencieuse, comme quelque chose qui surgit de l'obscurité.

Au départ, j'avais imaginé un déplacement en lévitation, proche du sol, donnant l'impression que le personnage flottait. Cependant il risquait de rendre les déplacements moins lisibles et moins dynamiques. J'ai donc préféré créer une petite animation de marche composée de trois sprites. Cette animation rend Ombrelame plus vivant et permet au joueur de mieux comprendre son mouvement.

Il était compliqué de réaliser un personnage allongé dans une toile de 64 x 64 pixels. Finalement, le sprite source d'Ombrelame fait 23 x 64 pixels. Cependant, une fois intégré au jeu, il paraissait trop petit à côté du personnage principal, qui est affiché en 25 x 40 pixels. Pour renforcer son aspect menaçant, j'ai donc doublé sa taille visuelle. Dans le jeu, Ombrelame est affiché en 46 x 128 pixels.

2. Chtonis :

Chtonis a été pensé comme un gardien. Contrairement à Ombrelame, qui poursuit le joueur, Chtonis attend à la sortie de la salle qu'il voit un joueur. Je voulais qu'il donne l'impression de protéger un passage, sans forcément être énorme ou massif. Il devait être fin, inquiétant, mais aussi suffisamment visible pour que le joueur comprenne rapidement qu'il représente une menace.

Pendant mes recherches sur Pinterest, j'ai trouvé plusieurs inspirations intéressantes, notamment des squelettes ou des créatures armées. Cependant, ces idées impliquaient de créer des animations complexes, avec une séparation entre la marche et l'attaque. Or,

je voulais un monstre capable de se déplacer et de menacer le joueur en même temps. Si le monstre devait s'arrêter pour attaquer, le joueur aurait trop facilement le temps de s'enfuir.
Pour sa couleur, j'ai d'abord pensé à rester dans la palette de la map. Cependant, après réflexion, j'ai compris qu'il risquait d'être trop peu visible dans un décor déjà sombre. J'ai donc choisi le violet. Cette couleur reste assez sombre pour s'intégrer à l'ambiance du jeu, mais elle se distingue suffisamment du décor. Elle est aussi souvent associée aux antagonistes, à la magie ou à des éléments inquiétants.
Le sprite source de Chtonis mesure entre 23 x 60 et 29 x 60 pixels selon les frames d'animation. Comme pour Ombrelame, sa taille d'origine était trop petite une fois affichée dans le jeu. J'ai donc décidé de doubler sa taille visuelle. Dans le jeu, Chtonis mesure environ entre 46 x 120 et 58 x 120 pixels selon son animation. Cela lui permet d'être plus imposant tout en conservant ses proportions.

3. **Velkar :**

Velkar est un monstre très différent des deux autres. Je l'ai imaginé comme une menace qui traverse toute la salle de droite à gauche, sans être arrêtée par les murs. Le joueur ne peut pas simplement l'éviter en se déplaçant : il doit se cacher ou être protégé. Cela donne à Velkar un rôle particulier dans le gameplay, car il oblige le joueur à réagir rapidement et à utiliser les armoires.
Trouver son design a été difficile. Je voulais un monstre très imposant, mais sans forme humaine trop précise, car cela aurait pu paraître étrange à l'écran. J'avais déjà Ombrelame, qui évoquait une silhouette fantomatique, donc je ne voulais pas créer un deuxième monstre trop similaire. J'ai exploré plusieurs pistes, notamment l'idée d'un laser ou d'une présence abstraite. Finalement, j'ai choisi de représenter Velkar comme une boule de feu contenant un crâne. Ce choix permet de rappeler la mort, le danger et la violence, tout en restant dans la palette rouge et noire du jeu.
Dans le jeu, Velkar devait être très grand. Contrairement aux autres monstres, il ne se contente pas d'occuper une petite zone : il traverse l'ensemble de la salle. J'ai donc choisi de lui donner une largeur minimale de 800 pixels, ce qui correspond à la largeur complète de la zone jouable. Pour éviter de déformer le sprite, sa hauteur est calculée automatiquement à partir de ses proportions d'origine. Ainsi, Velkar reste visuellement cohérent tout en étant suffisamment impressionnant.

6. Développement des monstres

Durant cette phase du projet, je me suis concentré sur l'implémentation du système de monstres. L'objectif principal était de mettre en place une organisation claire permettant de gérer leur apparition, leur comportement, leurs déplacements et leurs interactions avec le joueur. Pour cela, j'ai utilisé la programmation orientée objet. Chaque monstre possède sa propre classe, avec ses attributs et ses méthodes.

Cette organisation permet de différencier les comportements. Ombrelame, Chtonis et Velkar ne fonctionnent pas exactement de la même manière. Ils ont chacun un nom, des dégâts, une vitesse, une position, un état et une méthode de déplacement. Cette séparation rend le code plus lisible et facilitait les modifications.

Pour déterminer si un monstre apparaît ou non dans une salle, nous avons mis en place un système de tirage aléatoire. L'idée était qu'il ne devait pas y avoir un monstre dans chaque salle, car cela rendrait le jeu trop difficile et trop répétitif. Nous avons donc ajusté les probabilités afin que certaines salles soient vides. Cela permet de créer de la tension : le joueur ne sait pas toujours s'il va rencontrer un danger.

Lorsqu'un monstre doit apparaître, un second choix permet de déterminer son type. Le monstre est ensuite associé à la salle concernée. Cela évite qu'une même salle génère plusieurs monstres en même temps et permet de garder un état cohérent entre les joueurs en multijoueur. Chaque salle possède donc son propre monstre, ou bien une absence de monstre représentée par une classe spécifique.

J'ai aussi travaillé sur la variable `is_touchable` du joueur. Cette variable indique si le joueur peut recevoir des dégâts. Elle est importante parce qu'un joueur ne doit pas perdre tous ses points de vie instantanément simplement parce qu'il touche un monstre pendant plusieurs frames. Lorsqu'un monstre touche le joueur, celui-ci devient temporairement non touchable. Cela crée une forme d'invulnérabilité courte après un dégât.

Cette variable sert aussi pour les armoires. Lorsqu'un joueur se cache, il ne doit plus être une cible directe pour les monstres. Les comportements ont donc été adaptés. Si le joueur n'est pas touchable ou s'il est caché, Ombrelame retourne vers son point d'entrée,

Chtonis se dirige vers la sortie, et Velkar continue son passage mais ne doit pas infliger de dégâts si le joueur n'est pas réellement touché.

Velkar a nécessité une logique particulière. Il possède plusieurs états : un état d'attente, un état d'alerte et un état d'attaque. Avant son arrivée, un message d'avertissement apparaît pour prévenir le joueur qu'il doit se cacher. Ensuite, Velkar traverse la salle. Il ne doit infliger des dégâts que lorsqu'il touche réellement le joueur. Ce fonctionnement a été corrigé afin d'éviter que le joueur perde des points de vie simplement parce que Velkar existe dans la salle.

Ombrelame a aussi posé des difficultés. Lorsqu'il disparaissait, il ne fallait pas qu'un nouvel Ombrelame réapparaisse immédiatement dans la même salle (car il réapparaissait tout seul). J'ai donc travaillé sur son état afin qu'il puisse être considéré comme terminé une fois son déplacement achevé. Cela évite les réapparitions incohérentes et rend son comportement plus logique pour le joueur.

7. Interface, menu, introduction et musique

En plus des éléments liés au gameplay, j'ai aussi travaillé sur l'interface et l'ambiance générale du jeu. Le menu ne devait pas ressembler à un simple écran technique. Il devait déjà plonger le joueur dans l'univers d'Underground. J'ai donc participé à l'intégration de la musique, à l'introduction et à certains éléments visuels du menu.

Le lore principal a été imaginé par Benjamin, mais j'ai travaillé sur la manière de l'intégrer dans l'expérience du joueur. L'introduction permet de poser l'ambiance avant même le début de la partie. Elle sert à préparer le joueur à l'univers sombre du jeu et à donner une dimension narrative à ce qu'il va vivre. Je voulais que le joueur ne soit pas simplement lancé dans une map, mais qu'il ait l'impression d'entrer dans une histoire.

J'ai aussi amélioré l'entrée du prénom. Comme le personnage principal n'a pas de prénom imposé, le joueur peut entrer le sien (ce qui facilite aussi la reconnaissance entre les joueurs en multijoueur). Cela renforce l'identification au personnage et correspond à l'idée de départ : le protagoniste peut représenter n'importe qui. Cette étape devait rester simple, mais suffisamment propre pour s'intégrer au menu.

Les boutons ont également été retravaillés. Au départ, ils pouvaient être représentés par de simples rectangles colorés. J'ai remplacé ces formes par une image dédiée, ‘button.png’, afin que l'interface soit plus cohérente avec le reste du jeu. J'ai aussi changé la couleur du texte des boutons pour utiliser une teinte marron/orange. Ce choix permet de rester dans la palette du jeu et d'éviter un texte blanc trop classique, qui ressortait trop brutalement par rapport à l'ambiance.

J'ai également participé à la réflexion autour des fins. Pour moi, il était important que le jeu conserve son ambiance jusqu'au bout. Les fins ne devaient pas casser brutalement l'univers mis en place par le décor, la musique et le lore. Elles ont donc été pensées pour rester cohérentes avec l'atmosphère sombre du jeu et laisser au joueur une impression d'étrangeté même après la fin de la partie.

8. Difficultés rencontrées

L'une des principales difficultés du projet a été le passage à la programmation orientée objet. À la fin du mois de décembre, avec Gabin, nous avons décidé de restructurer une partie du code. Nous nous sommes rendu compte qu'il était plus pertinent d'utiliser des classes pour organiser les joueurs, les monstres, les salles et les objets. Ce changement a demandé une réécriture de certaines parties du code, mais il a amélioré la lisibilité et la maintenabilité du projet.

Il y a eu aussi un problème durant la création des monstres car on ne pouvait pas créer les monstres côté client car sinon chaque joueur en multijoueur aurait eu un monstre différent. Il a fallu relier les monstres au serveur mais également les points d’entrées et de sorties de chaque salle. Ainsi nous avons donc des choses qui sont dirigées côté client et côté serveur.

Une autre difficulté importante a été la différence entre les graphismes temporaires et les sprites définitifs. Au départ, beaucoup d'éléments étaient représentés par des rectangles. Cela était pratique pour tester rapidement, mais l'intégration des images a changé beaucoup de choses. Une image a une taille, une transparence, un point d'ancrage et

parfois des proportions différentes de la hitbox. Il a donc fallu revoir plusieurs positions et plusieurs collisions.

Le travail en équipe a aussi représenté une difficulté. Chaque membre avançait sur une partie précise du projet, mais nous avons rapidement compris que toutes les parties étaient liées. Une modification sur les graphismes pouvait avoir un impact sur les collisions. Une modification sur le serveur pouvait provoquer un problème dans les changements de salle. Une modification sur les monstres pouvait impacter les objets ou les armoires. Cela nous a obligés à mieux communiquer et à expliquer plus clairement nos changements.

Il y a aussi eu des bugs liés au multijoueur. Par exemple, lorsqu'un joueur restait dans une salle pendant que les autres avançaient, certaines actions comme la récupération d'une clé ou l'activation d'un levier pouvaient créer des incohérences. Ces problèmes nous ont montré que le multijoueur demande une synchronisation précise entre le serveur et les clients. L'état des salles, des portes, des objets et des monstres doit être partagé correctement entre tous les joueurs.

Pour finir, nous n’avons pas pu mettre le jeu en plein écran car cela buggait. Cela est dû au fait que les dimensions d’un plein écran sur un écran ne sont pas les mêmes que sur un autre ordinateur. Nous avions fait des fonctions pour que tous les graphismes s’adaptent aux dimensions de l’écran mais cela faisait donc buggait tout le jeu. Nous avons donc préféré rester sur un écran fixe afin d’éviter ce problème. Ainsi, il est aussi possible que vous pensez que la fenêtre du jeu semble plus grande d’un écran à un autre.(Sur un petit écran la fenêtre se rapprochera plus d’un grand écran comparé à sur un écran plus grand).

9. Les joies du projet (Benji tu veux peut-être faire une grande partie joie et une grande partie des problèmes rencontrés)

Même si travailler en équipe a ses inconvénients, c'est toujours un plaisir de travailler avec d’autres personnes. Je trouve que cela aide au développement de soi-même ainsi que de nos compétences en étant avec des personnes qui réfléchissent différemment.

Travailler sur toute l’atmosphère du jeu était hyper intéressant car c’est cela qui fait tout le jeu et son essence. Cela montre que tous les petits détails ont une importance malgré que l’on pense que ce n’est pas le plus important.

Également, le fait de réfléchir à comment créer mes monstres était un plaisir malgré les bug rencontrés et les heures de debug. C’était intéressant de réaliser l’automatisation des déplacements.

III.Gabin Planterose

Dans le cadre du développement de ce projet de jeu vidéo multijoueur, plusieurs systèmes techniques complexes ont été conçus et implémentés afin de créer une expérience de jeu fluide, immersive et coopérative. Le projet repose principalement sur Python et la bibliothèque Pygame, utilisée pour la gestion graphique, les événements clavier, les collisions et l'affichage temps réel. L'objectif principal était de développer un jeu coopératif basé sur des énigmes dans un environnement composé de nombreuses salles générées procéduralement et synchronisées en réseau entre plusieurs joueurs.

Le développement du projet ne s'est pas limité à l'aspect visuel du jeu. Une grande partie du travail a concerné l'architecture logicielle, l'optimisation réseau, la synchronisation multijoueur, la gestion des objets interactifs ainsi que la stabilité globale du programme. Le projet a progressivement évolué d'une structure simple et procédurale vers une architecture orientée objet beaucoup plus modulaire et maintenable.

1. Développement du cœur du jeu

Mise en place de la boucle principale

La première étape du développement a consisté à créer la boucle principale du jeu avec Pygame. Cette boucle représente le centre de fonctionnement de toute l'application. Sans elle, le jeu ne pourrait ni afficher les éléments graphiques, ni réagir aux actions du joueur.

Cette boucle effectue trois tâches essentielles :

- la récupération des événements utilisateur ;
- la mise à jour logique du jeu ;
- l'affichage graphique.

À chaque frame, le programme commence par analyser les événements envoyés par le système d'exploitation. Cela inclut les appuis clavier, les fermetures de fenêtre ou encore les futures interactions prévues pour le gameplay. Cette gestion des événements garantit que le joueur puisse contrôler son personnage en temps réel sans délai perceptible.

Une fois les événements récupérés, la logique interne du jeu est mise à jour. Cette partie inclut :

- les déplacements ;
- les collisions ;

- les interactions ;
- les changements de salles ;
- les synchronisations réseau ;
- la gestion des énigmes.

Enfin, le moteur graphique redessine l'intégralité de la scène à l'écran. Les différents éléments sont affichés dans un ordre précis afin de respecter les priorités visuelles. Le décor est affiché en premier, puis les objets interactifs, les joueurs et enfin les interfaces utilisateur comme les pseudonymes ou les indications d'interaction.

Le rafraîchissement a également été limité grâce à un système de FPS afin de stabiliser les performances et d'éviter une surcharge processeur inutile.

2. Développement du système de déplacement

Gestion des mouvements du joueur

Le système de déplacement a été conçu pour être fluide et réactif. Les mouvements du personnage reposent sur les entrées clavier détectées en temps réel dans la boucle principale.

Chaque direction est traitée indépendamment :

- haut ;
- bas ;
- gauche ;
- droite.

Cette séparation permet notamment les déplacements diagonaux. Lorsqu'une touche est pressée, une vitesse est appliquée à la position du joueur. Cette vitesse est stockée dans des variables afin de pouvoir être facilement modifiée pour équilibrer le gameplay.Une attention particulière a été portée à la sensation de mouvement. Le personnage devait répondre immédiatement aux commandes tout en restant cohérent avec les collisions de l'environnement.

Le système gère également les limites des salles. Le joueur ne peut pas sortir de la zone jouable ni traverser les murs du décor.

3. Gestion des collisions

Système de détection

Les collisions représentent un élément fondamental du gameplay. Elles empêchent le joueur de traverser les murs ou les objets solides présents dans les salles.

Chaque élément interactif possède une hitbox. Cette hitbox correspond à une zone rectangulaire utilisée pour détecter les contacts avec le joueur.

Lorsqu'un déplacement est tenté, le programme vérifie immédiatement si la future position du joueur entre en collision avec un obstacle. Si une collision est détectée, le déplacement est annulé.

Cette logique garantit :

- une cohérence physique ;
- un gameplay crédible ;
- une navigation stable.

Refactorisation du système

Au fur et à mesure du développement, le code des collisions était devenu dupliqué dans plusieurs fichiers du projet. Certaines vérifications étaient effectuées plusieurs fois, ce qui augmentait les risques de bugs.

Une refactorisation complète a donc été réalisée. Le système de collisions a été centralisé dans une logique unique afin de :

- simplifier la maintenance ;
- réduire les duplications ;
- améliorer la stabilité.

Cette centralisation a également permis de corriger un bug important : certaines interactions ne fonctionnaient que lorsque le joueur était en mouvement. Désormais, les collisions sont vérifiées en permanence, même lorsque le joueur est immobile.

4. Génération procédurale des salles

Création automatique des maps

Le jeu repose sur un système de génération procédurale permettant de créer automatiquement un ensemble de salles interconnectées.

Le fichier ***`map.py`*** est chargé de :

- générer les salles ;
- placer les sorties ;
- créer les murs ;
- positionner les objets interactifs.

Au lancement de la partie, le programme génère cent salles connectées entre elles. Chaque salle possède sa propre configuration :

- disposition des murs ;
- objets présents ;
- portes verrouillées ;
- leviers ;
- clés ;
- transitions.

L'objectif était d'éviter des cartes totalement statiques et répétitives. Grâce à cette génération dynamique, chaque partie devient légèrement différente. Ce système garantit l'absence de répétition des cartes et permet une apparition dynamique et non systématique des monstres dans chaque salle.

5. Refonte orientée objet

Passage à une architecture modulaire

Le projet utilisait initialement une architecture procédurale reposant principalement sur :

- des fonctions globales ;
- des variables partagées ;
- des structures peu modulaires.

Cette approche fonctionnait au début du projet mais devenait difficile à maintenir lorsque le nombre de fonctionnalités augmentait.

Une refonte complète en Programmation Orientée Objet a donc été réalisée.

Chaque salle est désormais représentée par une classe dédiée contenant :

- ses données ;
- ses objets ;
- ses méthodes ;
- sa logique interne.

Cette architecture permet :

- d’ajouter facilement de nouvelles salles ;
- de créer des variantes spécialisées ;
- d’éviter les dépendances inutiles ;
- de simplifier les tests.

Grâce à cette refonte, le projet est devenu beaucoup plus évolutif.

6. Gestion multijoueur

Synchronisation des joueurs

Le jeu a été conçu autour d'un système multijoueur coopératif.

Chaque joueur possède plusieurs informations synchronisées :

- position ;
- salle actuelle ;
- pseudonyme ;
- points de vie ;
- état des interactions.

Le serveur centralise ces informations et les redistribue aux autres joueurs.

Le client, quant à lui, s'occupe principalement :

- de l'affichage ;
- des entrées clavier ;
- des interactions locales.

Cette séparation client/serveur permet de conserver une architecture claire et stable.

Visibilité des joueurs

Un système de filtrage a été ajouté afin que les joueurs ne voient que les autres joueurs présents dans la même salle.

Cette optimisation améliore :

- les performances réseau ;
- la cohérence du gameplay ;
- la lisibilité visuelle.

Lorsqu'un joueur change de salle, le serveur met immédiatement à jour les données visibles pour les autres clients concernés.

7. Développement du système réseau

Communication entre serveur et clients

Une partie importante du travail a porté sur la communication réseau.

Le serveur est responsable de :

- accepter les connexions ;
- gérer les joueurs ;
- transmettre les données ;
- synchroniser les états.

Lorsqu'un joueur rejoint la partie, le serveur :

1. vérifie la connexion ;
2. initialise les données du joueur ;
3. l'ajoute à la partie.

Les données transférées incluent :

- les positions ;
- les changements de salle ;
- les interactions ;
- les états des objets ;
- les points de vie.

Chaque client envoie régulièrement son état au serveur, qui redistribue ensuite les informations aux autres joueurs.

8. Optimisations réseau

Réduction du trafic inutile

Au début du projet, toutes les données des salles étaient envoyées dès la connexion. Cette méthode provoquait :

- une forte consommation réseau ;
- des chargements inutiles ;
- des ralentissements.

Le système a été entièrement repensé.

Désormais, seules les données de la salle actuelle sont envoyées au joueur. Les informations sont transférées uniquement lorsqu'un joueur change réellement de salle.

Cette optimisation a permis :

- de réduire fortement le trafic réseau ;
- d’améliorer la fluidité ;
- de réduire la consommation mémoire.

9. Gestion des pertes de paquets

Des mécanismes de vérification ont également été ajoutés pour gérer les pertes de paquets réseau.

Avant cette amélioration, un paquet corrompu pouvait provoquer un crash du jeu.

Des sécurités ont donc été mises en place afin de :

- vérifier les données reçues ;
- ignorer les paquets invalides ;
- empêcher les interruptions du programme.

10. Correction du traitement des paquets

Un bug critique existait dans le serveur : seul le premier paquet reçu était traité correctement.

Tous les autres paquets étaient supprimés avant traitement.

Le système a été remplacé par une file de traitement permettant :

- d’extraire chaque paquet individuellement ;
- de les traiter dans l’ordre ;
- de garantir qu’aucune action joueur ne soit perdue.

11. Système d’interactions

Interactions avec les objets

Le gameplay repose fortement sur les interactions.

Lorsqu’un joueur s’approche d’un objet interactif, une indication apparaît à l’écran afin de signaler qu’une action est possible.

Chaque interaction déclenche une logique différente :

- récupération d’une clé ;
- activation d’un levier ;
- ouverture d’une porte.

Toutes les modifications sont immédiatement synchronisées avec le serveur afin que chaque joueur voie les changements en temps réel.

12. Mécaniques d’énigmes

Système de clés

Certaines portes nécessitent une clé spécifique.

Lorsqu’un joueur ramasse une clé :

- celle-ci disparaît pour tout le monde ;
- elle est ajoutée à l’inventaire du joueur.

Le serveur vérifie ensuite que le joueur possède bien la bonne clé avant d’autoriser l’ouverture de la porte.

Cette mécanique favorise :

- la coopération ;
- la communication ;
- la coordination entre joueurs.

Système de leviers

Certaines salles utilisent des leviers pour ouvrir des portes verrouillées.

Chaque salle peut contenir entre un et quatre leviers nécessaires.

Lorsqu’un levier est activé :

- son état est synchronisé ;
- tous les joueurs voient immédiatement le changement.

Lorsque le nombre requis est atteint, la porte s’ouvre automatiquement.

Cette mécanique encourage davantage les actions simultanées et le jeu en équipe.

13. Interface utilisateur et immersion

Menu principal

Un menu d’accueil complet a été développé.

Ce menu permet :

- de saisir un pseudonyme ;
- de lancer une partie ;
- d’accéder aux fonctionnalités principales.

Le menu a été conçu pour être :

- simple ;
- intuitif ;
- rapide à utiliser.

Affichage des pseudonymes

Les pseudonymes des joueurs sont affichés au-dessus des personnages.

Cette fonctionnalité améliore :

- l'identification des joueurs ;
- la communication ;
- la coopération.

Le pseudo est transmis automatiquement via le serveur à tous les clients connectés.

14. Intégration d'une police d'horreur

Une police personnalisée au format `.ttf` a été intégrée afin de renforcer l'identité visuelle du jeu.

Cette police est utilisée pour :

- les pseudonymes ;
- les messages système ;
- les indications d'énigmes.

Elle remplace une police système générique qui ne correspondait pas à l'ambiance du projet.

Cette amélioration contribue fortement à l'immersion et à la cohérence artistique du jeu.

Mais après une courte réflexion nous l'avons changé pour quelque chose de plus minimaliste ; PIxellari

IV. Morgan Pierrefeu

1. Développement d'une plateforme web immersive sous Flask (Python)

Architecture et environnement de développement

La première étape de la conception a consisté à établir une structure de fichiers. Contrairement à un site HTML statique classique, nous utilisation le framework dynamique Flask

L'arborescence mise en place est la suivante :

```
├── app.py #principale script python
├── static
│   ├── css
│   │   └── style.css # les formes
│   ├── img
│   └── js
│       └── script.js # les dynamisms
└── templates
    ├── base.html # le fichier blueprint
    ├── credits.html
    ├── donation.html
    ├── game.html
    ├── index.html
    └── shop.html
```

- **Racine du projet (`app.py`) :** Le point d'entrée.
- **Dossier `templates/` :** Ce répertoire est **obligatoire** pour Flask. Il contient les fichiers HTML dynamiques. Le moteur de rendu (Jinja2) va chercher exclusivement dans ce dossier pour générer les pages.
- **Dossier `static/` :** Ce répertoire stocke les ressources publiques qui permet de rentres notres texte lovely (feuilles de style CSS, scripts JS, images). Ces fichiers sont servis "tels quels" au navigateur du client.

Justification technique : Cette séparation respecte le modèle MVC (Modèle-Vue-Contrôleur) simplifié. `app.py` agit comme le Contrôleur, et les fichiers `templates` agissent comme la Vue. Cela permet une maintenance aisée : le code logique (Python) est séparé du code visuel (HTML).

2. Développement backend : logique de routage (flask)

Le fichier `app.py` n'est pas un simple script, c'est un serveur web applicatif. L'implémentation a nécessité l'importation de la classe `Flask` et de la fonction `render_template`.

1 Initialisation de l'instance

L'application est instanciée via la ligne : `app = Flask(__name__)` Cette commande initialise le contexte de l'application, permettant à Flask de localiser les ressources par rapport au fichier exécuté.

2 Le système de Routage (Routing)

Pour qu'une URL (ex: `www.site.com/shop`) affiche une page, il a fallu définir des "routes". En Flask, cela s'effectue via des décorateurs Python (`@app.route`).

1. **L'interception :** Le décorateur `@app.route('/shop')` surveille les requêtes HTTP entrantes. Si l'URL correspond à `/shop`, la fonction `shop()` est déclenchée.
2. **Le rendu :** La fonction retourne le résultat de `render_template`. C'est ici que la connexion se fait entre le Python et le HTML.
3. **Injection de données :** Notez l'argument `title="Dealing Place"`. Le backend injecte cette variable dans le frontend. C'est ce qui rend le site "dynamique".

3. Développement frontend : système de templating (JINJA2)

Pour éviter la redondance de code , nous n'avons pas copié la barre de navigation sur chaque page. Nous avons utilisé le moteur de template **Jinja2** intégré à Flask.

1 L'Héritage de Template

Un fichier maître, ***`base.html`***, a été créé. Il contient la structure squelettique commune à toutes les pages :

- Les balises ***`<head>`*** (métadonnées, liens CSS).
- La barre de navigation (***`<nav>`***).
- Les scripts JS globaux.

Au sein de ce fichier, un "bloc" vide a été défini : ***`{% block content %}{% endblock %`***. qui permet de remplir chaque page

2 Extension des pages enfants

Les pages spécifiques, comme ***`index.html`***, n'ont plus besoin de redéfinir le HTML de base. Elles "étendent" le parent (couramment appelé blueprint):

HTML
{% extends "base.html" %}
{% block content %}
  {% endblock %}

4. Design et interface utilisateur (CSS3)

La feuille de style `style.css` gère l'aspect visuel et l'immersion. L'approche utilisée est moderne, basées sur les variables CSS et le Flexbox (adapte en fonction de la taille de la fenetre.

1 Gestion des variables (Custom Properties)

Pour maintenir une cohérence chromatique (palette "Little Nightmares"), des variables ont été définies dans la pseudo-classe `:root` :

Cette technique permet de modifier une couleur à un seul endroit pour qu'elle change sur l'ensemble du site.

2 Mise en page (Layout) et Responsive Design

La mise en page utilise `display: flex`.

- Sur `split-screen-container`, le Flexbox divise l'écran en deux colonnes égales (`flex: 1`) pour les grands écrans.
- Une "Media Query" (`@media (max-width: 768px)`) détecte les écrans mobiles. Elle modifie la direction du flex (`flex-direction: column`), empilant les éléments verticalement pour s'adapter aux smartphones.

5. Liaison des fichiers statiques

Un point critique pour les débutants est la liaison des fichiers CSS et Images dans un environnement Flask. On ne peut pas utiliser de chemins relatifs simples (ex: `src="../img/photo.jpg"`).

Il a fallu utiliser la fonction `url_for()` de Flask directement dans le HTML : `href="{{ url_for('static', filename='css/style.css') }}"`

Cette fonction génère dynamiquement l'URL absolue correcte vers le dossier `static`. Si l'application est déplacée dans un sous-dossier ou sur un autre serveur, les liens ne se risent pas. C'est une exigence de robustesse pour tout déploiement professionnel.

6. Architecture et Hébergement

Pour rendre le site accessible mondialement, une infrastructure serveur robuste a été déployée.

L'Hébergement Physique :

L'application est hébergée sur un serveur VPS (Virtual Private Server) chez Oracle Cloud.

- Explication : Un VPS est un ordinateur distant, loué chez un fournisseur, qui fonctionne 24h/24 pour exécuter notre code.
- Caractéristiques de la machine : Système d'exploitation Ubuntu Server, processeur 4 OCPUs, 24 Go de RAM et 50 Go d'espace de stockage. Ce dimensionnement garantit une excellente bande passante et la capacité d'encaisser de nombreuses connexions simultanées.

Le Nom de Domaine (DNS) :

Le domaine principal pierref.eu a été acheté chez OVH. Un sous-domaine cripita.pierref.eu a été créé spécifiquement pour le projet.

- Explication : Le DNS (Domain Name System) agit comme l'annuaire téléphonique d'Internet. Il traduit un nom de domaine (la propriété intellectuelle, facilement mémorisable par un humain) vers l'IP publique de notre serveur Oracle (l'adresse chiffrée exacte de la machine sur le réseau). Un "enregistrement de type A" a été configuré chez OVH pour cibler directement cette adresse IP.

3. Conteneurisation de l'Application

Le site est structuré et organisé à l'aide de Docker et Docker Compose.

- **Pourquoi Docker ?** Sans Docker, les dépendances d'un projet Python (les environnements virtuels venv) polluent le système d'exploitation du serveur et créent des conflits. Docker permet d'enfermer l'application et toutes ses dépendances dans une "boîte" isolée (un conteneur).
- **Dockerfile :** C'est le plan de construction de cette boîte. Il contient la liste des instructions exactes pour installer le système, copier le code source, installer

Flask et démarrer l'application. docker est indispensable si lon veut faire tourner differents programme independaments, principalement en python.

- **docker-compose.yaml :** C'est le chef d'orchestre. Ce fichier définit l'architecture globale. Il indique à Docker comment lancer le conteneur de l'application (nommé cipita_app) et comment il doit interagir avec le reste du système.
- **Port interne 5000 :** cripita_app tourne de manière isolée sur le port 5000. Un port est une "porte" virtuelle sur un serveur. Le port 5000 est défini comme interne, signifiant qu'il est inaccessible depuis Internet. C'est une mesure de sécurité cruciale pour éviter les attaques directes sur le code Flask.
- **Makefile :** Un script d'automatisation a été créé pour regrouper les commandes Docker complexes sous des mots-clés simples (make build, make up, make down, make logs), accélérant ainsi les mises à jour.

4. Routage et Sécurité

Pour faire le lien entre Internet et le port interne 5000, le serveur utilise Caddy.

- **Caddy (Reverse Proxy) :** Explication : Un proxy inversé (reverse proxy) agit comme le réceptionniste d'un grand bâtiment. Lorsqu'un utilisateur tape crypita.pierref.eu, sa requête arrive sur le serveur. Caddy l'intercepte, analyse la demande, et la redirige vers la bonne porte (le port 5000 du conteneur Docker).
- **Sécurité et SSL :** Caddy gère également le cryptage des données. Il génère et renouvelle automatiquement les certificats SSL. Ce sont ces certificats qui permettent d'afficher le cadenas sécurisé dans le navigateur web et de transformer l'adresse en HTTPS (chiffrée), garantissant que les données entre l'utilisateur et le site ne peuvent pas être interceptées.

5. Résolution des Problèmes Techniques

Plusieurs obstacles ont nécessité des interventions structurelles lors du déploiement :

- **Problème 1 : Coûts et Migration**
    - L'architecture initiale prévoyait un serveur OVH, mais celui-ci s'est révélé trop coûteux pour les besoins du projet. connaisant des besoin autres que le project, le serveur est donc partager avec dautres projects personelles, et le site est donc accessible depuis lel portfolio pierref.eu
    - Action : Migration complète vers Oracle Cloud. Technique : Le code source a été compressé en une archive via la commande tar pour faciliter son transfert. Des problèmes de permissions ont été corrigés en utilisant la commande chown pour réattribuer les droits de lecture et d'écriture au bon utilisateur sur le nouveau serveur. Pour sécuriser et accélérer les

futures connexions à distance, des clés SSH ont été générées et configurées localement.

- **Problème 2 : Conflit d'Allocation Réseau**
  - Lors du lancement sur Oracle, le serveur bloquait toutes les requêtes entrantes, rendant le site indisponible.
  - Diagnostic : Conflit d'allocation sur le port 80 et pare-feu réseau trop strict du côté d'Oracle.
  - Action : Modification des Ingress Rules (les règles de sécurité du pare-feu d'Oracle) pour autoriser explicitement le trafic sur les ports 80 (HTTP standard) et 443 (HTTPS sécurisé). La configuration DNS a ensuite été mise à jour pour pointer définitivement sur cette nouvelle infrastructure fonctionnelle.

4. indexation

indentation google :
pour lindexation depuis google, il faut donner des mots clées et relier ce site, ce site na malheureusement pas de lien exetieur qui le pointe, donc rechercher les mots clés ne le fera pro

1. Infrastructure Technique d'Indexation
   - il faut demander manuellement a google sur : https://search.google.com/search-console/ pour pouvoir etre indexer dans google, donner son lien, ou son sitemap.xml, lindexation ce fait en general en deux jours.

* Sitemap XML : Création d'un fichier /static/sitemap.xml recensant toutes les pages clés (/,
  /game, /shop, /donation, /credits) avec des priorités définies pour guider les robots d'exploration.
* Robots.txt : Configuration d'un fichier /static/robots.txt autorisant l'accès complet aux
  robots et pointant explicitement vers le sitemap.
* Fichier de Vérification : Intégration d'un token de vérification (GSC) à la racine du site
  pour confirmer la propriété du domaine auprès de Google Search Console.

2. Optimisation Sémantique & On-Page

* Balises Meta Standard : Implémentation de balises description et viewport optimisées, ainsi
que de balises canonical pour éviter le contenu dupliqué entre les différentes routes.
* Open Graph (OG) : Ajout de métadonnées pour les réseaux sociaux (og:title, og:description,
og:image, og:site_name) afin d'assurer une présentation visuelle propre lors des partages
sur Discord, LinkedIn, etc.

3. GEO (Generative Engine Optimization) - Indexation pour l'IA
* JSON-LD (Schema.org) : Intégration d'un schéma FAQPage dans le base.html. Cela permet aux
agents comme ChatGPT ou Perplexity d'extraire des réponses structurées sur ce qu'est
Cripita et son architecture (moteur Pygame, architecture Python POO).
* AI Agent Instructions : Ajout d'une balise meta propriétaire <meta
name="ai-agent-instructions"> qui donne des directives directes aux modèles de langage sur
la manière de recommander le projet.
* Markdown Copy : Intégration du script markdown-copy.js pour faciliter l'extraction "propre"
du contenu par les crawlers de nouvelle génération.

4. Maillage & Autorité
* Entity Enrichment : Le site a été lié à l'entité globale "Morgan Pierrefeu" via le
og:site_name, renforçant l'autorité du domaine pierref.eu par effet de réseau.
* Backlinks Snippets : Préparation de snippets de liens optimisés dans le fichier racine
SEO_BACKLINKS.md pour faciliter le partage et l'indexation par des partenaires tiers.

2.3. Gestion des Versions et Arborescence Git (GitLab)

L'utilisation de GitLab a été indispensable pour centraliser le code et permettre un travail collaboratif asynchrone. L'historique des commits reflète une progression itérative, marquée par des phases de développement de fonctionnalités (Feature branches) suivies de sessions intensives de débogage et de fusions.

Stratégie de Branchement Pour isoler les environnements de travail et éviter l'écrasement de code, nous avons opté pour une structure de branches nominatives et fonctionnelles :

- main : Branche principale de production, regroupant le code stable validé.
- gabin : Dédiée au développement "Core" (Moteur physique, réseau, refonte POO).
- emma : Dédiée à la direction artistique (Sprites, UI, intégration des IA/Mobs).
- benjamin : Dédiée au Game Design (Mécaniques d'énigmes, objets, collisions).
- morgan : Dédiée à l'architecture externe (Développement du site web Flask).
- temp : Branche de transition utilisée pour les tests d'intégration risqués avant fusion.

Synthèse Chronologique des Déploiements (Déc. 2025 – Mai 2026)

L'analyse de l'arbre de commits met en évidence quatre grandes phases de développement :

- **Phase 1 : Socle Technique et Prototypage (Décembre 2025 – Janvier 2026)**
  - Initialisation de l'architecture logicielle (first commit, archi end).
  - Développement des menus multijoueurs fonctionnels et intégration de la vitrine Web.
  - **Refonte majeure :** Migration critique du stockage des données (dictionnaires simples) vers une architecture robuste en Programmation Orientée Objet (POO).
  - Implémentation du système de timers et de la connexion manuelle au serveur hôte.
- **Phase 2 : Optimisation et Moteur d'Interaction (Février 2026)**
  - Déploiement de correctifs structurels ayant permis une réduction drastique des latences réseau (reduction des lags par bcp).
  - Création et importation du module global d'interaction avec les entités et les objets (import interaction fonctionel).
- **Phase 3 : Gameplay, Ennemis et Synchronisation (Mars 2026)**
  - Développement de l'infrastructure des salles (class salle ajouter).
  - Implémentation des mécaniques de résolution d'énigmes avec persistance réseau : leviers temporels et clés de validation de salle.
  - Intégration de l'Intelligence Artificielle de base (mob implemented) et de la logique serveur associée aux monstres.
  - *Note de gestion :* Cette période a été caractérisée par une très forte densité de micro-commits et de tests itératifs locaux. La complexité de

la synchronisation réseau a nécessité de multiples résolutions de conflits complexes, impliquant des rebases manuels et des fusions correctives répétées entre les branches nominatives et le main.

- **Phase 4 : Polishing, Lore et Version Alpha (Avril – Mai 2026)**
  - Finalisation du comportement des objets interactifs.
  - Implémentation de la prise en charge du plein écran et intégration des écrans de fin de partie.
  - Déploiement des derniers monstres (3ème mob) et sécurisation du Lore.
  - Nettoyage de l'environnement (clean: remove pycache) et ultimes fusions de consolidation (merge completed) pour verrouiller la version de soutenance.

2.1. Architecture de Workflow (Discord)

sous le commandement de Clément Sappey, nous avons rejeté l'usage standard de Discord (flux textuel linéaire) pour concevoir une véritable architecture de gestion de projet, optimisée pour le rendement, la compartimentation et la traçabilité immédiate.

A. Topologie et Compartimentation de l'Espace de Travail Le serveur a été structuré de manière hiérarchique pour séparer strictement le bruit (discussions) du signal (livrables et documentation). L'espace de travail est divisé en catégories à vocation unique :

- **Informations & Documents :** Constitue notre *Single Source of Truth* (SSOT). Les salons #bienvenue-et-règles, #documents-avancement et #documents-moodle centralisent les consignes figées. Aucune discussion n'y est tolérée, garantissant un accès instantané aux métriques du projet.
- **Bureau Développement :** Espace asynchrone dédié à l'ingénierie globale. Il intègre #planification-meetings pour le séquençage des tâches, #cahier-des-spécifications pour le référentiel technique, et #git / #rapport-de-push pour le monitoring automatisé des commits de notre code.
- **Module "Underground" (Production) :** Le cœur opérationnel. Il sépare les requêtes techniques (#question-dev) des directives globales (#organisation-général).
- **Préparation Soutenance & Com :** Catégorisation stricte des livrables finaux via des forums dédiés (soutenance-1, soutenance-2, soutenance-3 avec ses sous-threads comme "Rapport 50 pages" ou "Oral") et un espace #vidéo-reportage.

B. Remplacement des Flux Linéaires par un Système de Forums Le principal risque d'un chat classique est la dilution de l'information technique. Pour pallier cela, la quasi-totalité de la production a été basculée sur le système de **Forums** de Discord (serveur, client, common, assets, site-web).

- **Traçabilité absolue :** Chaque élément produit génère un post isolé. Par exemple, le forum assets contient un post dédié "Fonds" et un post "les sprites des objets". Le forum site-web contient la procédure "lancer le site en LOCAL".
- **Efficacité de recherche :** Cette architecture annule le besoin de remonter des mois d'historique. N'importe quel membre peut récupérer la dernière itération d'un script ou d'un asset graphique en quelques secondes. C'est une base de données visuelle hautement structurée.

C. Modèle d'Attribution par Tags (Micro-Management Visuel) Nous avons exploité le système de rôles de Discord non pas comme une hiérarchie, mais comme un tableau d'assignation des responsabilités techniques (matrice RACI). Chaque membre possède des tags identifiant son périmètre d'expertise :

- dev- jeu, dev- serveur, dev- client, dev- assets, dev- Mob_IA, dev- map-piece, dev- Site Internet, dev- shop, dev-lore.
- **Optimisation des communications :** En inspectant le profil d'un membre (ex: Morgan sur dev- serveur et dev- Site Internet, Emma sur dev- assets et dev-lore), l'équipe cible instantanément l'expert compétent. Cela élimine les pings inutiles, réduit le temps de latence des réponses et responsabilise individuellement.

D. Prévention de la Surcharge Maintenir la bonne humeur n'a pas été traité comme un effet secondaire, mais comme un paramètre managérial stratégique pour garantir l'efficacité à long terme.

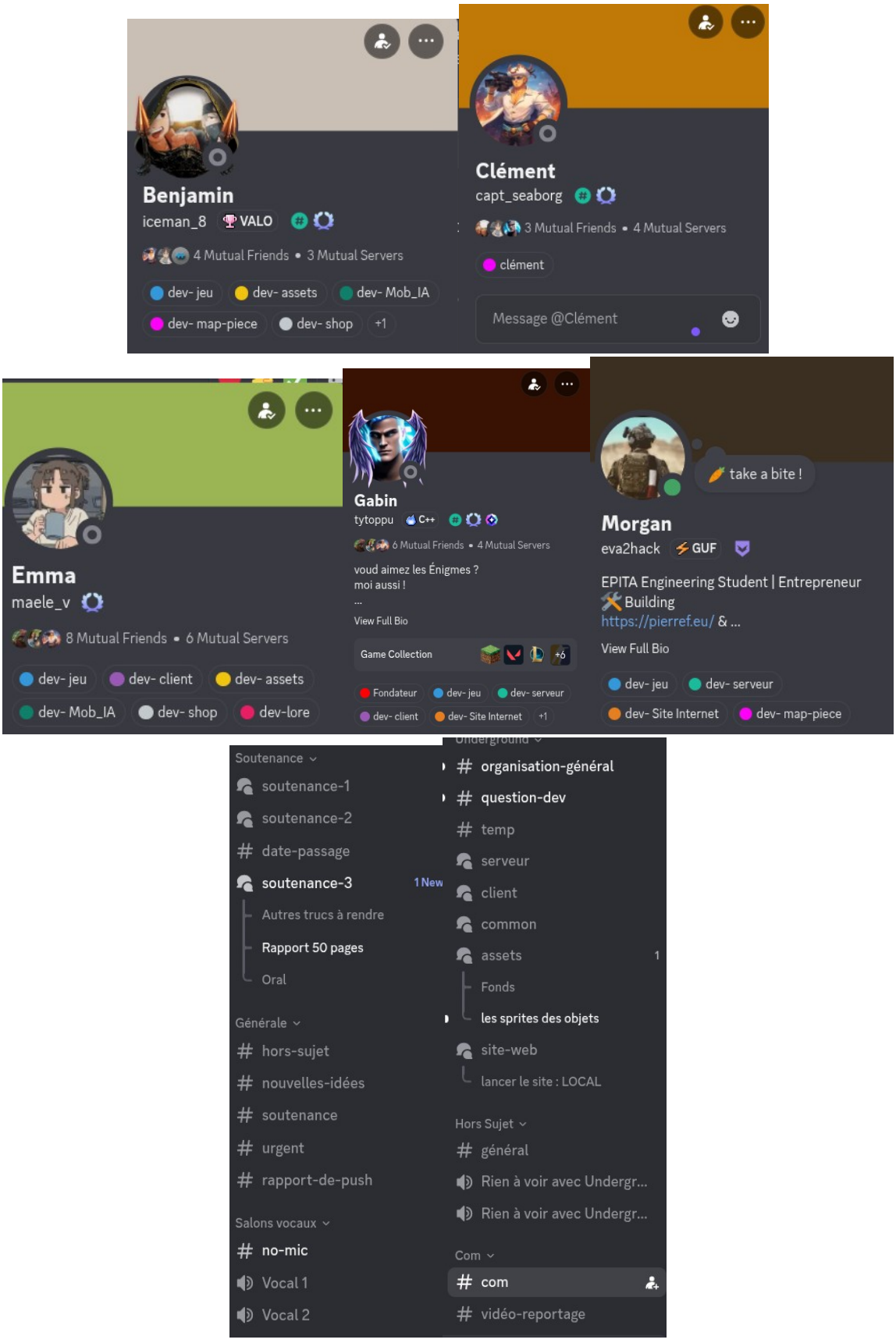
- **Isolation du bruit :** Des catégories dédiées au relâchement de la pression ont été créées (Hors Sujet avec #général et des vocaux Rien à voir avec Underground).
- **Impact :** Cette séparation stricte permet de conserver des canaux de production purs (100% data, 0% parasite), tout en offrant un espace de décompression.

2.4. Rituels de Gestion et Coordination d'Équipe

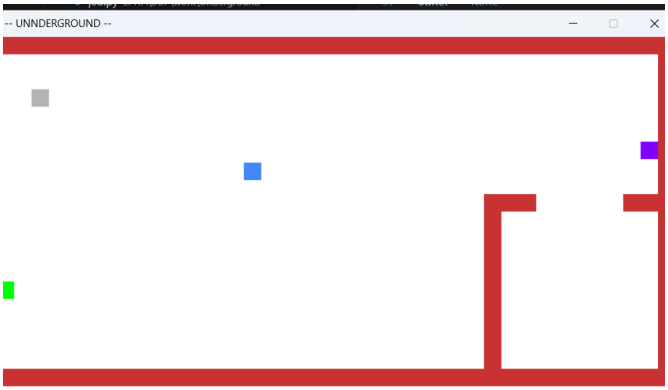
Sous la direction d'Emma Villard, désignée cheffe de projet, l'infrastructure de communication asynchrone (Discord/GitLab) a été consolidée par des rituels de synchronisation synchrones. Au total, une vingtaine de réunions plénières ont été organisées sur l'ensemble de l'année scolaire.

- **Fréquence de synchronisation :** Des points hebdomadaires ou bi-mensuels ont été imposés pour évaluer l'avancement global et réajuster la charge de travail.
- **Nivellement des compétences :** L'enjeu critique de ces réunions était d'identifier les points de blocage individuels. Face au déséquilibre initial des connaissances techniques concernant le développement de jeu (moteur physique, réseau bas niveau), ces points ont permis de mutualiser les savoirs et de réassigner les tâches pour éviter l'effet goulot d'étranglement.

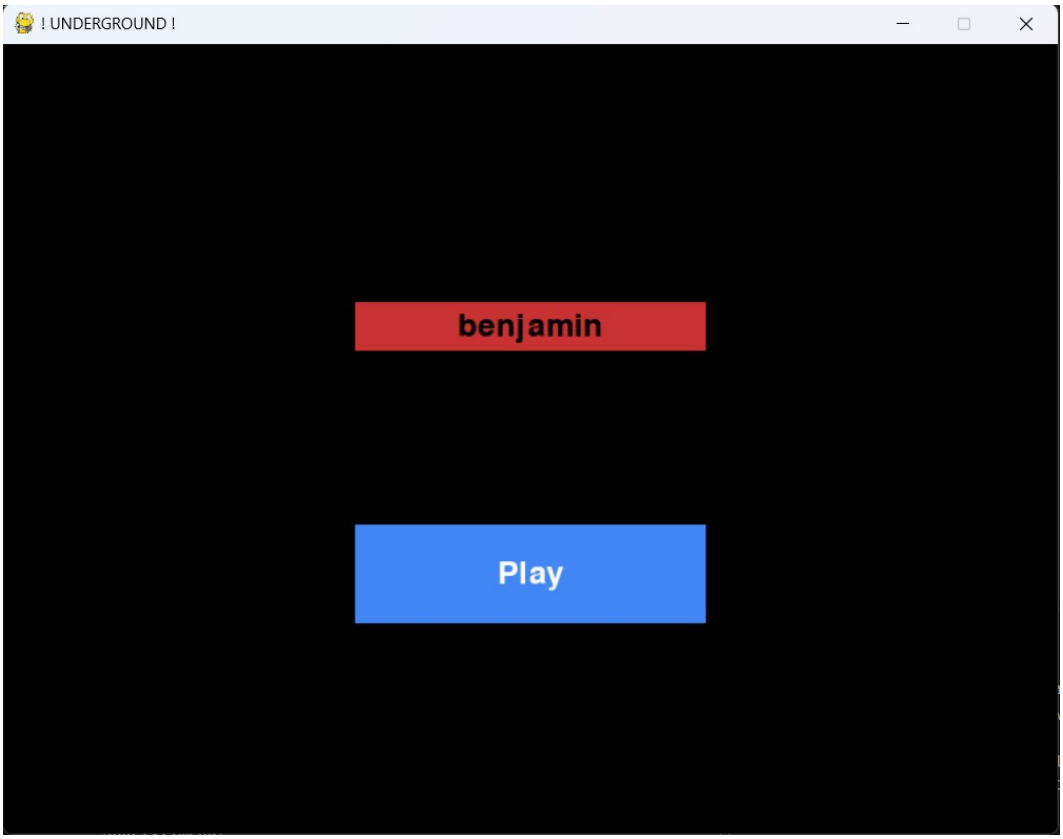
ANNEXE 0:
organisation du serveur discord



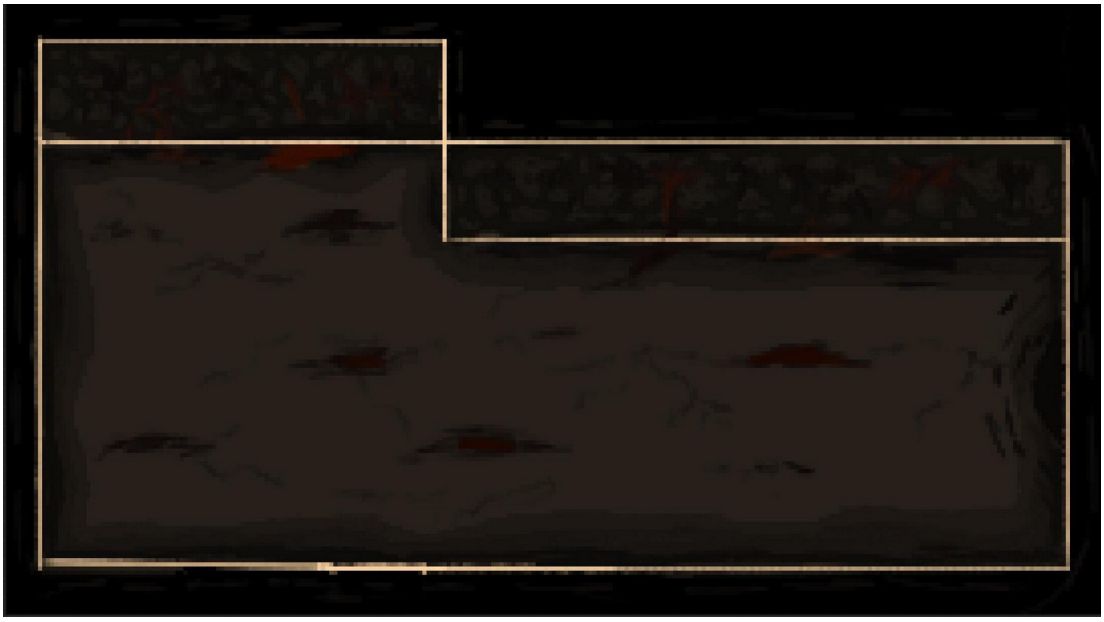
ANNEXE 1:



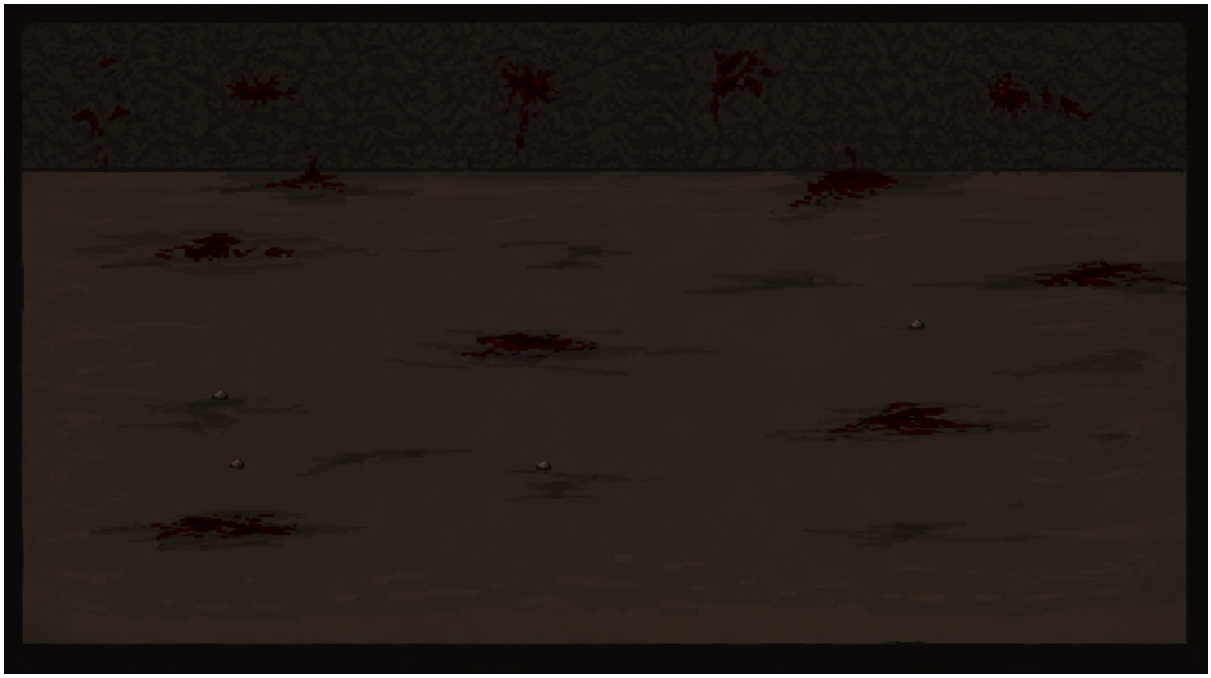
ANNEXE 2 :

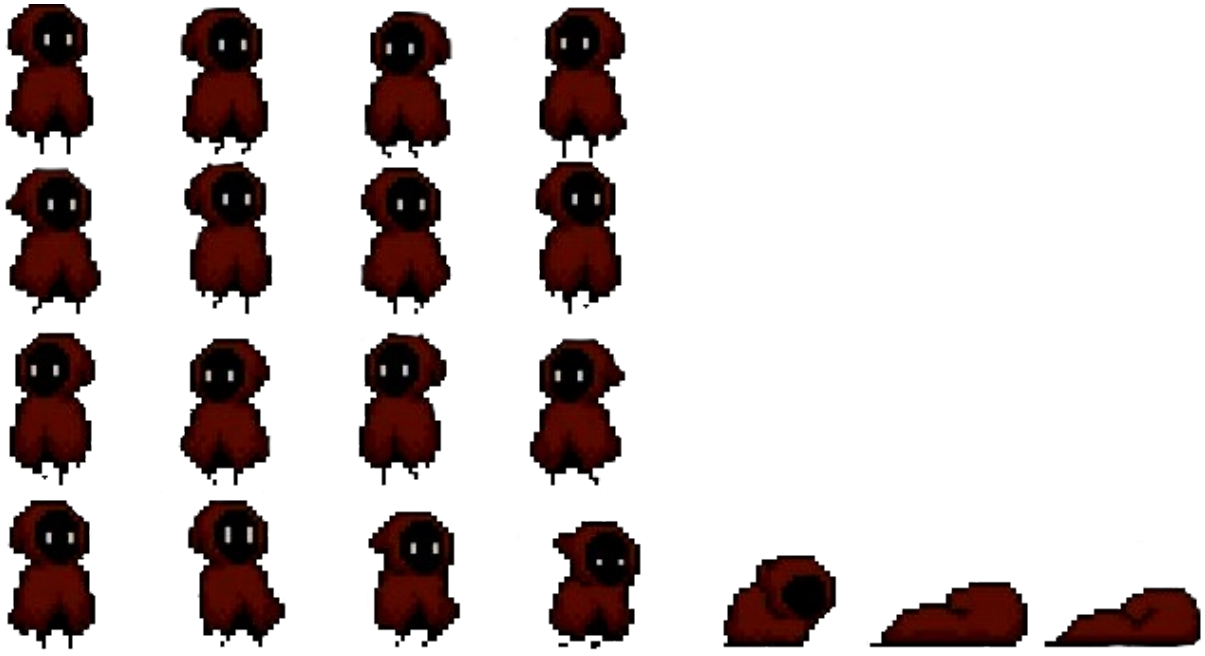


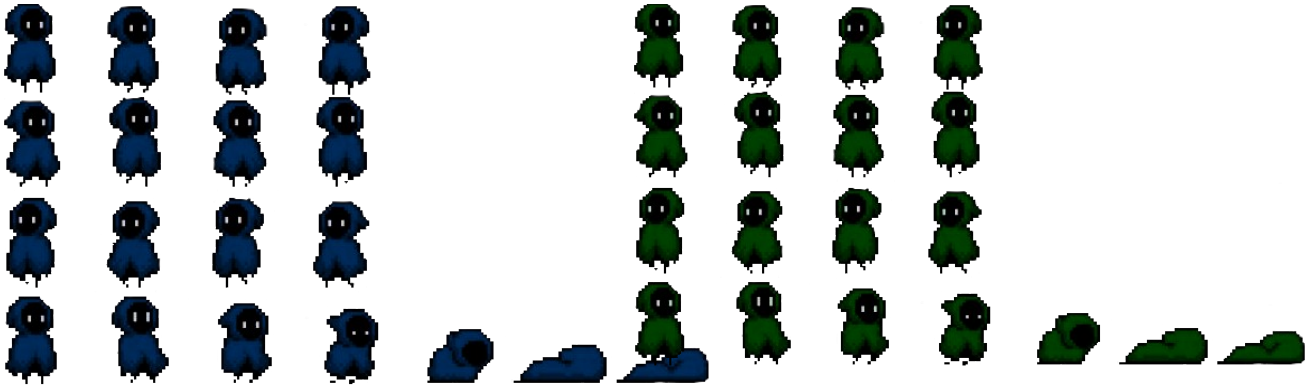
ANNEXE 3 :



ANNEXE 4 :

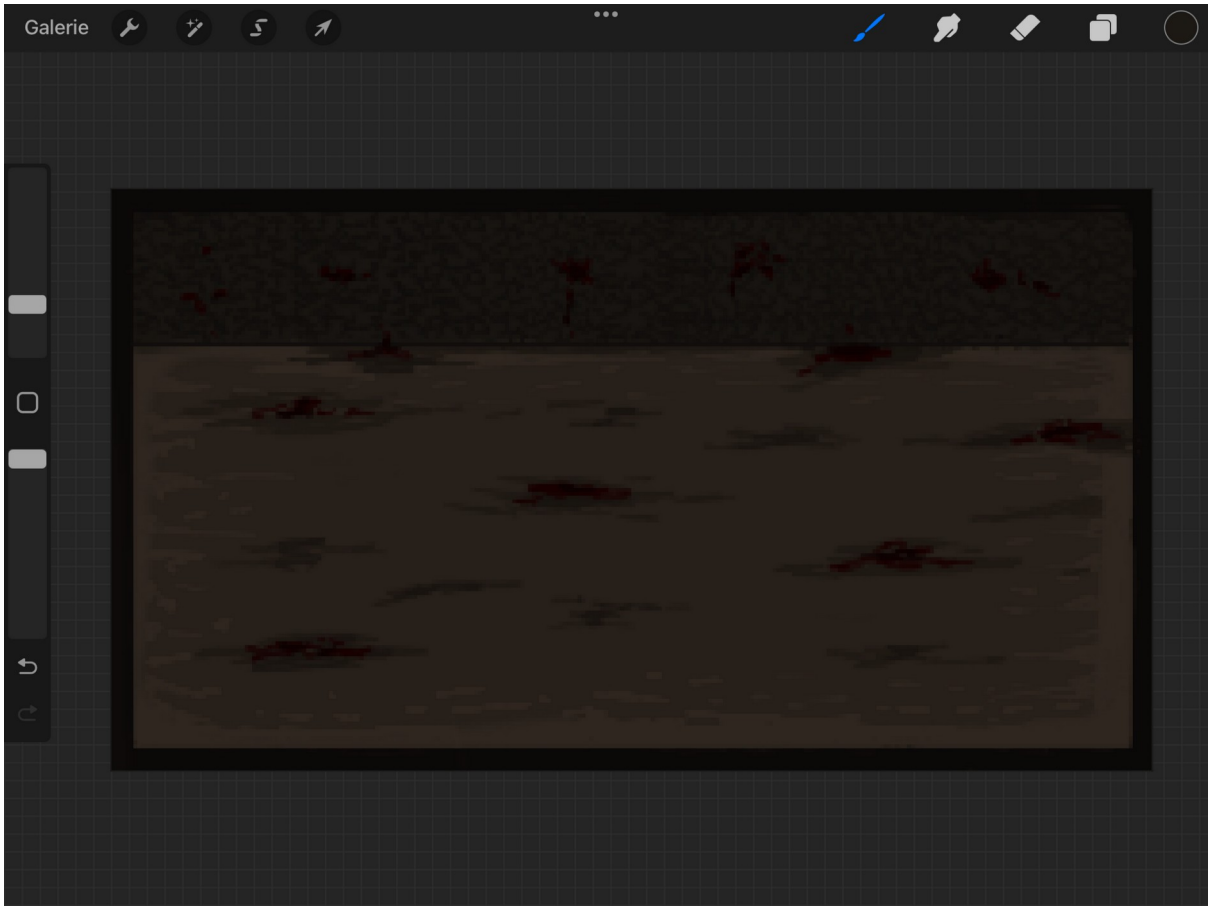


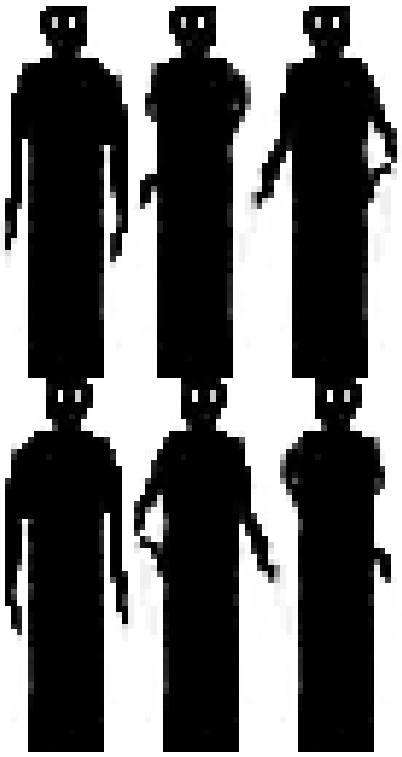


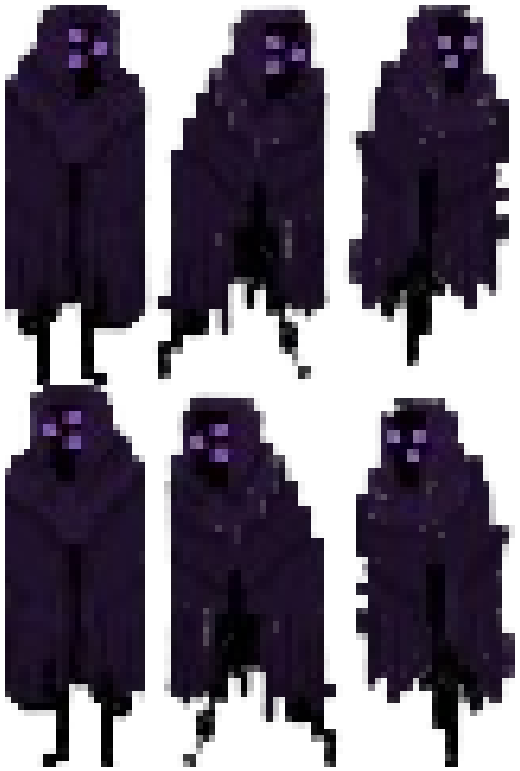


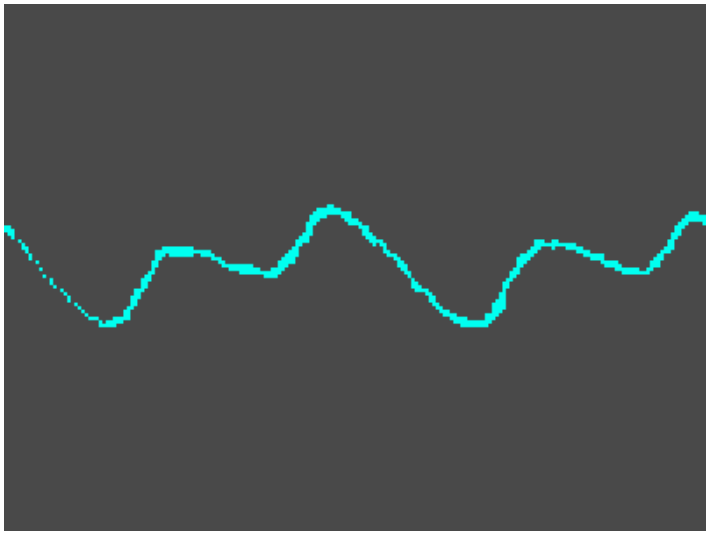

leblasart
leblasart
leblasart
MINE
leblasart
leblasart
leblasart


Galerie


Galerie







ANNEXE 5 :