RAPPORT DE SOUTENANCE 1

Mars 2022

Alexandre Agostinho, Sébastien Combe, Nathan Fontaine, Dimitri Brancourt

Avant-propos

Bonjour cher lecteur, ce rapport de soutenance vous permettra d'avoir une meilleure vision de l'avancement de notre projet

Table des matières

1 Introduction

1.1 Rappel

Notre projet est un jeu plateformer 2D de style pixelart développé à l'aide de Unity.

1.2 La team frelu

Notre équipe s'est répartie les tâches de la manière suivante :

Tâches	Alex.A	Dimi.B	Séba.C	Nath.F
Objects Design		S	S	R
Mobs/Characters Design			S	R
Game Design			R	S
GUI Design		S		R
GUI Programing		R	S	S
Gameplay Programing	S	R	S	S
IA Programing	R	S		
Website	R		S	
Multiplayer	R	S	S	
Sound Design				R

R : responsable, S : suppléant

1.3 Soutenance 1

Voici l'avancement actuel du projet

Tâches	Alex.A	Dimi.B	Séba.C	Nath.F	Total
Objects Design	0%	5%	0%	0%	5%
Mobs/Characters Design	0%	0%	0%	5%	5%
Game Design	0%	0%	2%	0%	2%
GUI Design	0%	0%	0%	2%	2%
GUI Programing	0%	10%	0%	50%	60%
Gameplay Programing	5%	40%	0%	5%	50%
IA Programing	20%	0%	0%	0%	20%
Website	25%	0%	0%	0%	25%
Sound Design	0%	0%	0%	5%	5%

2 Travail d'équipe

Afin d'ordonner un maximum notre travail, nous avons eu besoin de mettre en place une manière de travailler et aussi d'apprendre à utiliser certains outils.

2.1 Ordonner les nouveaux items

Pour pouvoir garder une efficacité dans notre travail, il nous faut bien sûr organiser nos créations. En partant des fichiers : dans 'Asset', organiser les préfabs dans le dossier 'Prefabs', les scripts dans le dossier 'Scripts', les scènes dans le dossier 'Scenes'... ainsi que tous les autres éléments dans les dossiers qui leur correspondent. Evidemment, nous avons fait de même avec le contenu de ces fichiers. Cette organisation se retrouve aussi dans le montage des scènes : on place les éléments en ordre en utilisant un maximum la hiérarchie. Enfin, nous avons pris soins de nommer les noms des différents élements que l'on a créé de manière claire et facilemment compréhensible pour que les autres puissent plus simplement les utiliser plus tard.

2.2 Git

Pour réussir à développer à plusieurs et surtout de manière plus ordonnée, nous avons utilisé l'outil Git. Il s'agit d'un puissant outil utilisé par une grande majorité de devloppers pour gérer leur projet. De manière générale, Git permet de versionner un projet, et de naviguer à travers ces versions. De plus il permet l'interaction avec un server Git, ce qui permet non seulement de sauvegarder et d'accèder à son projet n'importe où, mais aussi et surtout de le partager avec d'autres personnes, et donc de travailler en équipe. Dans son utilisation, Git utilise des 'commits' (des versions du projet) labellé sous différante branches. Le but étant d'avoir une branche par fonctionnalité et de 'merge' (de fusionner) les branches entre elle quand le développement d'une fonctionnalité nécessite les fonctionnalités d'une autre. Aussi, on merge les branches entre elle pour produire la version complète du projet. Travailler sur des branches séparémment permet, en outre de mieux imager le développement, d'éviter de casser les implémentations des autres lors de la mise sur le serveur des nouvelles fonctionnalités. Cependant, il arrive que des conflits apparaîssent lors de la fusion de branches, mais Git permet de gérer cela assez efficacement.

Pour notre projet, nous nous sommes servis de Gitlab comme seveur Git. Le projet Git à été organisé de la manière suivante :

- branche 'main' : version fonctionelles / complète
- branche 'dev' : version incomplète - sert pour les création de branches et récupérer des fonctionnalités
- autres branche : leurs nom correspond à leur fonctionnalité (interactions, GFX, AI...)

Gitlab nous permet d'interagir avec les autres membres du groupe sur les différents merges et fonctionnalités à implémenter, mais nous nous servons surtout aussi d'un serveur discord, que j'expliquerai plus bas. Alexandre, le chef de projet c'est chargé de créé le projet, et de le paramètrer. La structure du projet à été décidé avec une consertation de l'enemble du groupe et précisé dans le README du dépot Git.

En outre, nous utilisons aussi Git pour la construction de ce rapport : chacun peut alors écrire une partie et cela nous permet de mixer le tout plus efficacement.

2.3 Discord

Comme énoncé plus haut, nous avons mis en place un serveur discord que l'on utilise très fréquemment. La communication étant plus qu'importante dans ce type de projet, il s'agit en effet d'un outil essentiel à la bonne organisation du projet. Nous avons des canaux textuels réservé au différentes conversations sur les fonctionnalités du jeu, mais aussi pour tout les projets annexe (comme pour l'écriture de ce rapport de soutenance). Nous utilisont beacoup les canaux vocaux qui nous permmettent d'échanger beacoup plus facilemment.

2.4 Trello

Pour finir, afin de garder une trace des avancements sur le projet, nous avons mis en place un tableau Trello. Celui-ci nous permet de visualiser les tâches faites et celle qui doit être faites, ainsi que les informations sur les deadlines et d'autre informations utiles.

Nous avons construit notre tableau en mettant une liste pour chaque tâches (répertoriée dans le tableau plus haut). Dans ces listes nous rajoutons des cartes pour grouper les fonctionnalités selons leur hiérarchie dans la timeline et leurs points communs. On utilise des checklists pour dresser la liste des fonctionnalités sous ces cartes.

Par exemple : Sous la liste 'AI',
- 'Basic mob mouvement'
  - 'Mouvements prédéfinit sur une zone'
  - 'Mouvements aléatoire dans une zone'
- 'Advenced AI'
  - 'Pathfinding'
  - 'Mouvements by Pathfinding'
- ect...

**Representation schematique de la vue de l'application*

De plus, cet outil est disponible sur smartphone, ce qui nous permet à tout momment, notamment pour le chef de projet, de vérifier l'avancée du développement, et d'intervenir plus rapidemment si un écart se creuse entre les prédictions de deadlines et l'état actuel des fonctionnalités du projet.

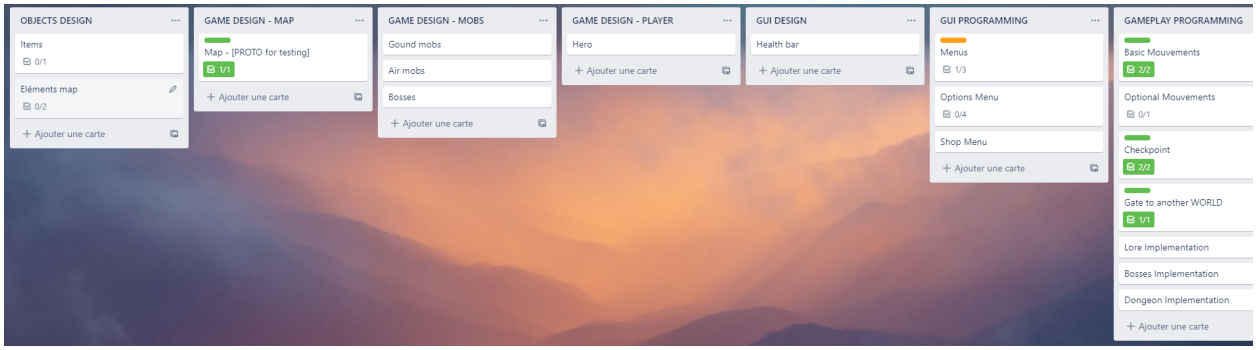

FIGURE 1 – Aperçu Trello

3 Avancement du projet

Nous allons maintenant vous présenter plus en détail ce que nous avons fait et comment nous voyons la suite du projet.

3.1 Menus d'affichages

De bon menus d'affichages sont une partie essentielle dans la mise en place d'un jeu, c'est pourquoi il est important que nous les implémentons correctement. J'ai tout d'abord cherché dans la documentation unity pour trouver les plugins qui permettent de gérer des scènes ainsi que les panels, de ce fait nous avons implémenté un script qui permet de se mouvoir entre les différentes scènes. Les fonctionalités de ces scripts sont mises en places grâce à l'ajout de boutons à l'écran et d'interaction avec des touches. Pour l'instant, je n'ai créer que des menus fonctionnels sans vrai design graphique car pour juste qu'ils soient fonctionnels étant donné que nous créerons un désign personnalisé plus tard dans l'avancement du projet.

3.1.1 Menu Options

Le menu options se décompose en 4 parties : Les boutons "video", "audio", "help" et "commands". Chacun d'entre eux permettra d'ouvrir un panel qui permettra au joueur d'effectuer des modifications ou actions vis-à-vis du paramètre sélectionné.

Figure 2 – Aperçu test Menu Options

3.1.2 Menu Principal

Le menu Principal est le menu de démarrage du jeu, il sera quant à lui composé des boutons "Resume"(ou "Start" en fonction de si le joueur à une save ou non), "Load", "Options", "Help" et "Quit game".

FIGURE 3 – Aperçu test Menu Principal

3.1.3 Menu Pause

Le menu Pause est constitué des boutons "Resume", "Options"(amenant donc au menu options), "Main Menu", "Quit Game".

FIGURE 4 – Aperçu test Menu Pause

3.2 Graphismes

Nous utilisons Gimp pour faire les graphismes en style pixelart, pour cela nous créeons des calques de 32x32 ou de 16x16 et on y applique une grille pour pouvoir faire du pixel par pixel. Ensuite nous regroupons ces différents calques d'une même catégorie dans une plus grande image appelée une "sprite sheet" qui permettra de créer des tilemaps(le décors du jeu/map, les platformes ...) ainsi que des animations pour des personnages, des objets, des mobs ... Pour ce faire, nous utilisons l'outil intégré de Unity qui permet de créer des animations en prenant des éléments précis de la "sprite sheet" afin de les placés à un timing voulue ce qui permettra de créer une animation. Pour gérer ces animations, nous utilisons l'option "Animator" intégré à unity qui permet de sélectionner une animation en fonction d'intéractions de l'utilisateur avec le jeu (par exemple, une animation de marche vers la droite quand le joueur va à droite).

3.3 TileMap et cycle jour/nuit

Cette partie sera centrée sur la création de la tilemap mais aussi que du codage du cycle jour/nuit,

3.3.1 Prise en main de Gimp

Pour commencer parlons de quelque chose des plus important : la prise en main de l'outil de travail Gimp.

Gimp est un outil d'édition et de retouche d'image, facile à utiliser en apparence il a fallu s'adapter à cet nouvel outil que nous utilisons pour créer les maps, les personnages et principalement tous les designs présent dans le projet.

Cependant Gimp n'est pas si difficile à prendre en main, quelques tutoriels par ci par là et le tour était joué (ou presque). Malgré cette prise en main accès simple nous avons quelque chose en moins... La fibre artistique, en effet nous ne sommes pas des graphistes dans l'âme et les premiers designs ressemblait plus à des bouillies de pixels qu'autre chose, malgré tout nous nous sommes améliorés et nous avons réussis à faire naitre quelques designs.

3.4 Création des tilemaps

Avant la création de notre première tilemap il a fallu réfléchir à se que nous voulions, un design ne se fait pas au hasard.
Nous avons donc réfléchis (pas énormément mais un peu), pour savoir à quoi pouvais ressembler notre premier niveau.

La première tilemap est donc assez basique, faite avec nos compétences actuelles, elle est vouée à évoluer selon nos besoins, pour le moment elle est simplement "fonctionnelle", présentant les terrains basiques nous permettant de construire notre premier niveau.

Cette création de tilemap nous a permis d'en apprendre plus sur les tilesheet, d'en apprendre plus sur la construction d'un niveau

En effet, une tilemap doit être construite de façon à ce qu'elle puisse être utilisé telle une palette d'un peintre pour construire nos niveaux, ainsi chaque "pièces" d'une tilemap doit être construite pour que chaque "pièces" s'emboite parfaitement.

Ainsi la principale difficulté n'a pas été de s'adapter à l'outil de travail ou bien technique mais bien artistique, le métier de graphiste n'est pas aisé loin de là, construire et designer une tilemap est un travail de longue haleine.

3.4.1 Un travail de longue haleine

Finalement la création de la tilemap n'a pas posé de soucis majeurs, néanmoins ce travail était long et fastidieux, créer et designer chaque partie de la tilemap, une par une, prend un temps monstreux.

La partie artistique pose toujours problème et est fortement responsable du temps passer à créer une tilemap.
En effet créer une tilemap cent pour cent fonctionnelle n'est pas complexe, mais créer quelque chose d'artistique et d'agréable à l'oeil est cent fois plus complexe, l'aspect artistique a été le plus souvent le principal responsable d'un revirement de style ou tout simplement d'un reset comple.

La création des tilemap est donc très chronphage et est l'une des parties les plus longues et fastitieuse.

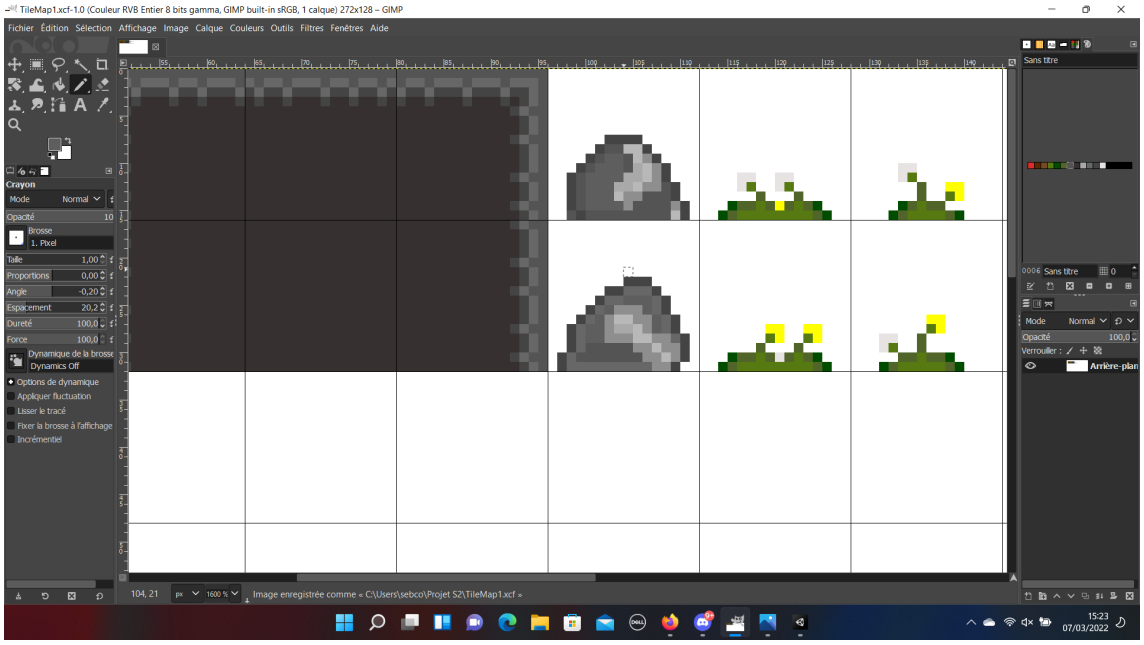

FIGURE 5 – Aperçu d'une tilemap

3.4.2 Pixelart

Pour notre jeu nous avons choisi le pixelart pour le visuel de notre jeu, cela peux paraître simple mais faire et designer des décors en pixelart peux causer certains problèmes.

Le pixelart, outre une style, peux rapidement devenir une contrainte, il devient difficile (par exemple avec un cercle) de faire certaines formes, et représenter certaines choses devient rapidement moche.

3.5 Cycle jour/nuit

Entrons dans la partie coding, le cycle jour nuit est central dans notre jeu, en effet lors de la nuit le nombre de monstres augmenter, il faut donc commencer par quelque chose : l'implémentation du cycle jour/nuit.
L'implémentation d'un cycle jour/nuit n'est pas si compliqué, il suffit de changer l'intensité de notre lumière principale, pour cela créons un script qui compte chaque image par seconde, avec ce compteur créons des variables représentant les secondes, les minutes et les heures. La valeur de l'intensité changeara en fonction de ces variables, et voilà nous avons notre cycle jour/nuit fonctionnel.

3.6 Interactions

La partie Interactions sera centrée sur le Gameplay programing, cette partie englobe donc les actions du personnages et ses interactions avec d'autres objets ...

3.6.1 Prise en main de Unity

Parlons maintenant d'un point important de l'initialisation du projet, la prise en main de Unity !

Unity a beau être un outil puissant et relativement simple à utiliser, nous avons quand même ressenti quelques difficultés au début. Maintenant nous sommes plus à l'aise avec cet outil.

La plus grosse difficulté avec Unity est son côté visuel. En effet ce côté qui rend Unity puissant, simple à utiliser et permet à l'utilisateur de changer des paramètres dans une interface appelée inspecteur possède un petit désavantage. Au début il est assez difficile de se retrouver avec toutes les possibilités existantes.

Bien heureusement, des tutos en ligne et l'utilisation de la documentation nous a permit de mieux comprendre comment commencer et avancer dans notre projet. Gardons tout de même en mémoire que des erreurs ou bugs sont apparus jusqu'à récemment.

3.6.2 Prise en main de la programmation dans unity

Un autre point abordé lors du début du projet est la programmation. En effet Unity a une manière de fonctionner bien à lui. L'intégralité du gameplay et de l'interactivité dans Unity est basée sur trois principes : les GameObjects, les composants et les variables.

Tout objet d'un jeu est un GameObject : c'est le cas des personnages, de la map, des accessoires, bref, de tout.

Les GameObjects ne peuvent rien accomplir seuls. Pour qu'il puisse interagir et faire des actions particulières, il lui faut des propriétés, obtenues par l'ajout de Composants (Components).

Les Composants définissent et contrôlent le comportement des GameObjects auxquels ils sont attribués. Un simple exemple est l'ajout d'un Composant Rigidbody à un objet pour qu'il tombe, ou même l'ajout d'un collider pour gérer les collisions de l'objet.

Les Composants ont de nombreuses propriétés modifiables, ou variables, qui peuvent être ajustées dans l'inspecteur, dans l'Éditeur Unity et/ou via un script. Nous pouvons dans l'exemple ci-dessus, régler la friction des collider, la transparence des sprites, qui sont les apparences des objets.

Ici, la difficulté était juste de s'approprier le fonctionnement de la programmation dans unity, finalement les tutos et la doc ont permit de gérer les différentes classes et méthodes de unity.

3.6.3 Déplacements du perso

Finalement les déplacements du personnage ne sont pas si compliqués, il faut utiliser les propriétés de Rigidbody pour les déplacements horizontaux. Il faut stocker le mouvement du joueur dans une variable pour pouvoir faire tourner le personnage quand la vitesse est négative et jouer l'animation de course.

La plus grosse difficulté est d'implémenter le saut. Pour cela plusieurs possibilités existent.
Celle que nous avons retenue est de vérifier si le personnage est au sol ou non via une variable. Enclencher le saut passera par la vérification de cette variable.
En plus du saut normal un deuxième saut plus grand a été implémenté (via une seconde variable notamment)

Un autre déplacement particulier mais plus difficile à gérer est la montée aux échelles.
Cette fois nous nous servons d'une box collider sur l'échelle pour savoir si le personnage est a portée. Si oui le joueur peut maintenant appuyer sur E pour activer la possibilité de monter verticalement au niveau de l'échelle.

3.6.4 Gestion de la barre de vie et des dégâts

La barre de vie est gérée avec 9 images de coeurs : chaque coeur se voit attribué un demi-coeur en dessous de lui puis un coeur vide, ce qui offre la possibilité de cacher le coeur complet pour ne voir qu'un demi-coeur puis le demi-coeur pour voir un coeur vide.

Il y a trois manières de perdre des coeurs : toucher un ennemi, appuyer sur h (pour les tests, cela disparaitra dans la version finale) et enfin de tomber dans le vide.

3.6.5 gestion de la deathzone

Quand le personnage tombe, il verra une animation d'écran noir puis réapparaitra là où se trouve le point PlayerSpawn.
De plus il perdra un coeur. Cette zone est en fait un collider que l'on peut ajuster, si le joueur rentre dedans alors l'animation d'écran noir se joue et le personnage est replacé au niveau de PlayerSpawn et perd un coeur.

3.6.6 gestion des Checkpoints

Les checkpoints sont des collider, qui permettent de déplacer PlayerSpawn à leurs position, lorsque le personnage rentre dedans et appuie sur E.
Le personnage regagne tous ses coeurs lors de cette action, c'est pour l'instant le seul moyen de regagner de la vie.

Le principe de nos checkpoints est de pouvoir les réutiliser à chaque fois que l'on veut. Cela permet donc les retours en arrière et ainsi une plus grande liberté.

3.6.7 creation de portails

Cette fois deux types de portails on été implémentés. Il y a d'abord le portail dans la même scène et ensuite le portail qui mène à une scène différente.

Le portail dans la même scène est bien sûr le plus simple, il s'inspire du principe de la dethzone.
Donc lorsque le personnage rentre dans le collider du portail et appuie sur E, cela active l'animation de l'écran noir. Ensuite le personnage est téléporté au second portail. Le principe marche dans les deux sens.

Maintenant, le portail vers une autre scène possède quelques ressemblances.
Il téléporte le joueur lorsqu'il appuie sur E en étant en contact avec le collider du portail. Il y aura encore une fois l'animation d'écran noir.
La principale différence avec le portail normal est que ce portail doit posséder une méthode qui sauvegarde certaines données : personnage, caméra, santé ... On appelle donc DontDestroyOnLoad.

3.6.8 ouverture de coffres

Cette intéraction particulière est plutôt simple, elle est constituée d'une animation d'ouverture de coffre, qui s'active lorsque le personnage est en contact avec le collider du coffre et le joueur appuie sur E.
Cette interaction fera appel à l'inventaire et sûrement aux items de notre jeux, qui ne sont pas encore mis en place.
La création de l'inventaire des items puis leur sauvegarde s'annonce être la partie la plus difficile. Il ne restera plus que les PNJ à ajouter après cela.

3.6.9 problèmes et difficultés

Cette partie se termine sur les problèmes et difficultés rencontrés, même s'il n'y a pas eu de très gros problèmes.
La principale difficulté au début était la prise en main de unity. Elle a ensuite été la capacité à se retrouver dans le projet et à suivre de manière précise chacune des actions effectuées.
En effet les problèmes rencontrés était principalement dus à des erreurs d'inattention ou des fonctionnalités activées qui rentraient en conflit avec certains autres objets.
Le problème le plus embêtant sont quand on veut vérifier et faire plusieurs choses en même temps. Dans ce cas, il faut bien séparer les actions et les variables.

3.7 L'intelligence Artificielle

3.7.1 Description

Dans ce type de jeu, l'intelligence artificielle n'a pas le besoin d'être très poussée. En effet, nous avons prévu pour les mobs de base de les faire suivre un chemin généré entre la position de l'ennemi et celle du joueur. Le mob ne pourra cependant pas quitter une certaine zone, le but étant de ne pas regrouper tout les ennemi de la carte sur le joueur (cela pourrai peut-être devenir un mod du jeu Hardcore). Le but serai aussi d'éviter que les mobs tombent dans le vide en allant chercher le joueur.

3.7.2 Implémentation

Pour implémenter l'intelligence artificielle, nous avons choisi d'utiliser le package externe d'Unity 'Astar'. Il comprend des scripts de base pour la création de chemins (un pathfinder pour la zone de recherche et un seeker pour utiliser cette zone). Il comprend aussi des scripts plus avancés et automatisés mais nous ne les avons pas utilisé.

FIGURE 6 – Aperçu du Pathfinding

3.7.3 Difficultés

Lors de l'implémentation, nous avons rencontré beaucoup de difficultés. La première était la bonne mise en place des paramètres des éléments mis en jeu. De la définition de la zone du pathfinder au éléments composant l'ennemi, certains paramètres ont du être réglé au centième près.
La seconde est certainemment celle qui nous à pris le plus de temps et d'énergie à régler. Le message d'erreur indiquait qu'il ne trouvait pas de point d'arrivé. Après des heures de recherche, de déboggage, de recompilation des bibliothèques d'Unity et aussi de plusieurs git rebase, l'erreur fut enfin trouvée. Elle provenait d'une ligne de code manquante dans le script du mouvement du joueur. Cette ligne de code permet la mise à jour de la position de l'objet 'Transform' rataché au Player. Le chemin ne pouvais alors pas trouver de point d'arrivé puisque la position n'était pas réglée.

4 Conclusion

Merci d'avoir eu le courage d'être arrivé jusqu'ici et d'avoir lu notre rapport de soutenance, nous espérons que vous appréciez notre projet et notre avancement, et nous vous invitons à nous donner des conseils pour améliorer quoi que ce soit.

Nous vous prions d'agréer nos plus sincères salutations.
La team FRELU ;)