Rapport Soutenance

Julien Tuan Khoi Bui
Andrei Vartiuc
Anh-Khoa Nguyen
Clement Hautier
Ayoub El Asri

Sommaire

Introduction

Ce rapport de soutenance permet de réunir l'avancée du projet en vue de la deuxième soutenance. Il contient un bilan de ce qui a été accompli ainsi que des éléments qui restent à finaliser. Des illustrations et des exemples concrets viennent appuyer l'état d'avancement, permettant de mieux visualiser les résultats obtenus jusqu'à présent.

Ce document se concentre sur les réalisations concrètes du projet tout en mentionnant les aspects qui demeurent à compléter. Il inclut des explications sur les obstacles rencontrés et les solutions envisagées pour les surmonter, afin de maintenir la progression.

Présentation Jeu

Nous allons ici rappeler quel est l'objectif que nous souhaitons atteindre afin de poser des bases solides. Le jeu que nous allons créer et réaliser est un jeu du style "top-view" (vue de haut) en 2D. C'est un jeu de survie dans lequel il sera possible d'explorer, de créer, de combattre et de découvrir de nouvelles choses tout au fur et à mesure du jeu. Notre objectif est d'implémenter toutes sortes de mécaniques présentes dans ce genre de jeu 2D afin de le rendre le plus complet possible. Le jeu auquel il devra le plus ressembler est un jeu nommé Necesse, qui constitue une référence proche par son style et ses mécaniques.

2. Interface

Cette Section réunit toutes les informations concernant l'interface.

2.1 Canva

Pour la conception de notre projet, nous avons utilisé Canva, une plateforme en ligne permettant de créer facilement des visuels, notamment des titres avec des polices spécifiques aux jeux vidéo. Cet outil nous a aidés à concevoir certains éléments graphiques pour améliorer l'aspect visuel du projet.

Toutefois, notre utilisation de Canva est restée limitée. Nous l'avons employé de manière ponctuelle, principalement pour des besoins spécifiques de mise en forme, sans en faire un élément central de notre conception.

2.2 Conception

Pour l'aspect visuel de notre projet, nous avons porté une attention particulière à l'harmonie graphique afin de rester cohérents avec l'ambiance de notre jeu.

Le fond que nous avons choisi est une capture d'écran de notre propre jeu, permettant d'immerger immédiatement le joueur dans l'univers que nous avons créé. Cette approche renforce l'identité visuelle et donne un avant-goût du gameplay dès l'écran d'accueil.

Le titre, Eternal Forest, a été conçu avec Canva. Nous avons utilisé une police pixélisée, parfaitement adaptée au style rétro du jeu, et nous l'avons incrusté dans un élément de terre, ajoutant ainsi une touche visuelle immersive qui rappelle l'environnement du jeu.

Concernant les boutons, nous avons fait le choix de les personnaliser directement dans l'éditeur Godot. Plutôt que d'utiliser des assets génériques, nous avons conçu notre propre design, en privilégiant

un style simple et efficace, garantissant une bonne lisibilité et une navigation fluide pour le joueur.

Enfin, pour assurer une cohérence visuelle à travers toutes les pages, nous avons conservé la même mise en page et les mêmes éléments graphiques, ne modifiant que le nom des boutons en fonction du contexte.

2.3 Fonctionnement

Dans Godot, nous avons conçu plusieurs pages de menu, chacune ayant une utilité spécifique. L'architecture de navigation a été pensée pour être intuitive et efficace.

Le menu principal constitue le point d'entrée du jeu. Depuis ce menu, l'utilisateur peut accéder à la page solo ou à la page multijoueur.

La page multijoueur propose deux options :
- Héberger une partie, ce qui redirige vers une nouvelle page permettant soit de créer une partie, soit de continuer une partie existante.
- Rejoindre une partie, qui permet de se connecter à une session en cours au lieu d'en héberger une.

Le fonctionnement du menu repose sur la détection des clics sur les boutons. Lorsqu'un bouton est activé, un script en C# s'occupe de rediriger l'utilisateur vers la page correspondante. Cela est réalisé grâce à la méthode :

GetTree().ChangeScene("chemin/vers/la/page.tscn");

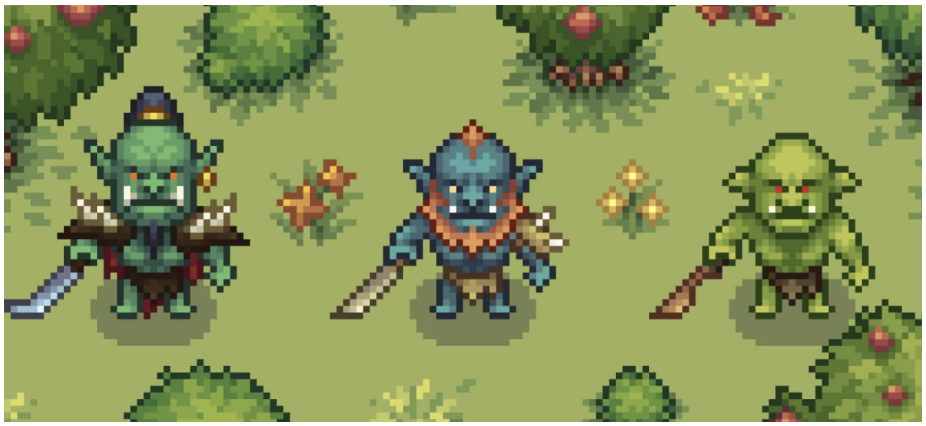
Ce principe est utilisé de manière homogène dans tout le menu, y compris pour lancer le jeu, garantissant ainsi une navigation fluide et réactive.

3. Mobs

Cette Section réunit toutes les informations concernant les Mobs.

3.1 Assets

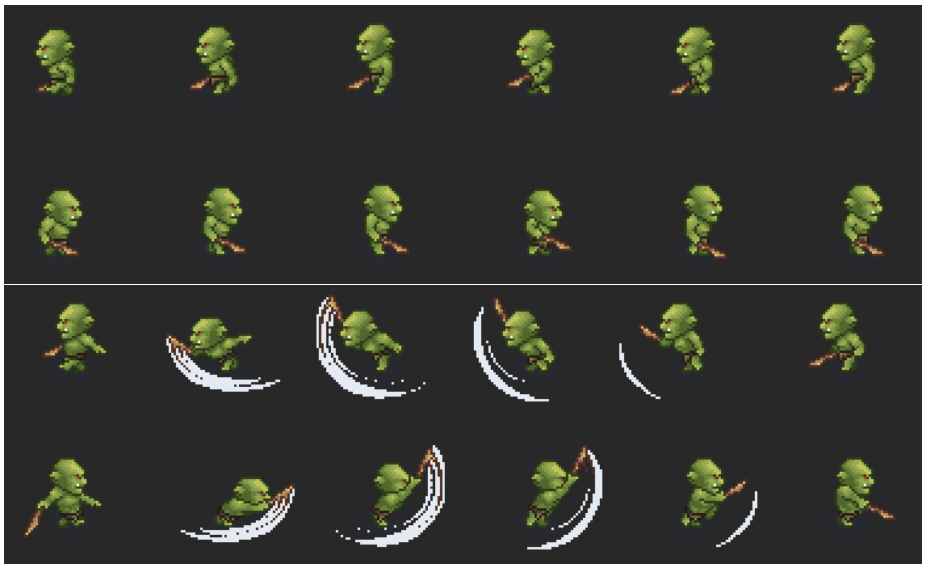
Pour l'intégration des mobs dans notre jeu, nous avons d'abord sélectionné des assets graphiques cohérents avec notre style visuel. Nous avons opté pour des textures gratuites disponibles sur le site CraftPix.net, qui proposait des sprites correspondant parfaitement à l'ambiance de notre projet.



Chaque mob était fourni avec tout le nécessaire dans son dossier d'assets, incluant les animations et les différentes poses, ce qui nous a permis de les intégrer facilement sans avoir à créer nous-mêmes les visuels.

3.2 Mouvements

Chaque mob possède ses propres animations de mouvement et d'attaque, qui sont présentes dans leur dossier respectif. Pour les mouvements, nous avons utilisé un algorithme similaire à celui du personnage principal (Player) pour gérer les déplacements vers la gauche et la droite. Cependant, nous avons fait le choix de ne pas implémenter de mouvements verticaux (haut et bas) dans notre jeu, ce qui a simplifié la conception du gameplay.



Concernant les attaques, les mobs peuvent uniquement attaquer à gauche et à droite. Cette restriction a été décidée pour maintenir une certaine simplicité dans les mécaniques de combat et garantir une expérience de jeu fluide.

3.3 Ce qui n'a pas été fait

Une fonctionnalité qui n'a pas encore été implémentée est la barre de vie. Cette décision a été prise de manière délibérée. Nous avons choisi de ne pas la prioriser pour l'instant, préférant nous concentrer sur le développement du système de dégâts dans son ensemble. L'idée derrière cette approche était d'éviter de se disperser dans le projet et de garantir une gestion plus fluide des mécaniques de jeu lorsque nous aborderons l'intégration de la barre de vie.

4. Intelligence Artificielle

Cette Section réunit toutes les informations concernant l'intelligence artificielle des mobs.

4.1 Fonctionnement

Pour l'implémentation de l'IA de nos mobs, nous avons conçu un algorithme complexe permettant à l'ennemi de réagir de manière intelligente aux mouvements du joueur. L'IA commence par déterminer sa propre position dans l'espace, puis elle calcule la distance qui la sépare du joueur. Une fois cette distance calculée, l'IA détermine la direction qu'elle doit prendre pour se rapprocher du joueur.

Nous avons également défini un rayon de détection autour du mob. Si la position du joueur entre dans ce rayon, le mob commence à attaquer. Cette fonctionnalité permet à l'IA de réagir uniquement lorsque le joueur est suffisamment proche, créant ainsi une interaction réaliste entre les deux.

Lorsque le mob ne détecte pas le joueur, il passe en mode patrouille. Dans ce mode, le mob se déplace de 100 pixels vers la gauche, puis fait demi-tour pour parcourir les 100 pixels suivants vers la droite. Cette alternance de mouvement permet d'éviter que le mob reste immobile. La patrouille s'active uniquement si aucun joueur n'est détecté, assurant que le mob se déplace de manière autonome lorsqu'il n'est pas en contact avec le joueur.

Lorsque l'IA détecte le joueur, elle commence à se diriger vers lui. Cependant, elle ne peut se rapprocher qu'à une distance pré-définie pour éviter que le mob ne poursuive le joueur indéfiniment, ce qui pourrait être causé par la hitbox. Une fois cette distance minimale atteinte, le mob lance une attaque dans la direction du joueur.

4.1 Problèmes rencontrées

Lors de l'implémentation de l'IA, plusieurs problèmes ont surgi. Le premier problème à commencé au tout début lorsque le mob se dirigeait en haut à gauche de la carte. Après avoir examiné le code, on a compris que nous avons pris la mauvaise position dans le nœud de la scène Player. En corrigeant cette erreur, le comportement du mob est devenu beaucoup plus cohérent.

Un autre problème était que l'ennemi courait à l'infini sur le joueur même si celui ci ne bougeait pas. Pour résoudre ce problème, nous avons dû définir une distance de limitation afin d'empêcher le mob de se coller constamment au joueur. Cela a permis de rendre le comportement de l'IA plus naturel, sans qu'elle ne suive sans fin le joueur.

Enfin, un autre bug est apparu lorsque le mob continuait à attaquer indéfiniment dès qu'il atteignait le joueur, même si ce dernier se déplaçait loin de lui. Le mob restait bloqué en mode attaque au même endroit. Pour régler cela, il a fallu ajouter plusieurs variables, dont un timer pour limiter le temps d'attaque, ainsi qu'une variable d'état pour indiquer si le mob attaquait ou non. Une fois ces variables bien implémentées, le problème a été résolu et l'IA a fonctionné correctement.

4.1 Ce qui n'a pas été fait

Bien que l'IA soit fonctionnelle, il reste encore des améliorations à apporter pour la rendre plus réaliste et plus sophistiquée. L'un des aspects qui manque actuellement est la capacité de l'IA à faire le tour des objets qui l'entourent. Actuellement, l'IA se concentre uniquement sur le joueur, sans tenir compte des obstacles qui se trouvent sur son chemin, comme les rochers, arbres, plantes, et autres éléments de la carte. Cela fait que le mob se heurte fréquemment à ces objets, perturbant son comportement.

Pour améliorer cela, il serait nécessaire d'ajouter une logique de navigation qui permette à l'IA de détecter ces obstacles et de les éviter, en modifiant sa trajectoire pour contourner les objets plutôt que de les percuter. Une fois cette fonctionnalité implémentée, l'IA

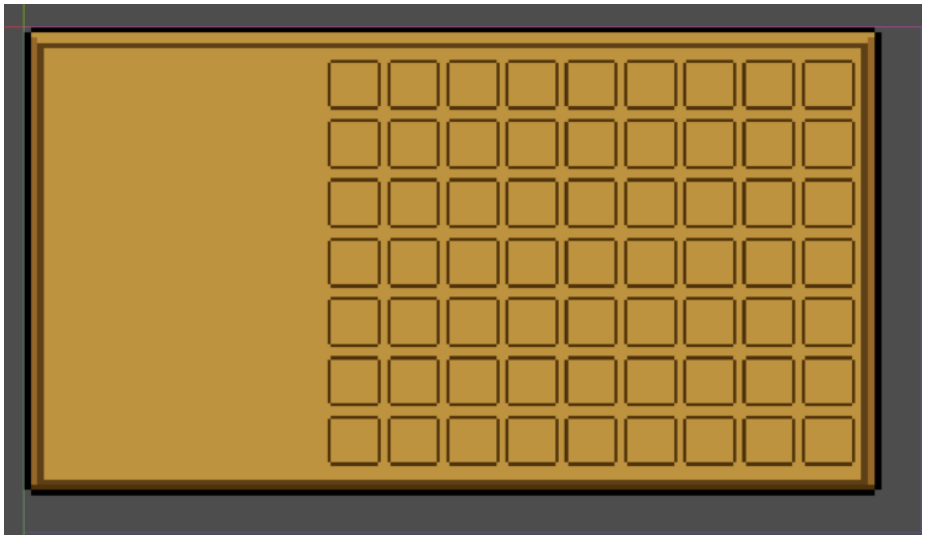
sera beaucoup plus fluide et réaliste, offrant une meilleure expérience de jeu.

6. Inventaire

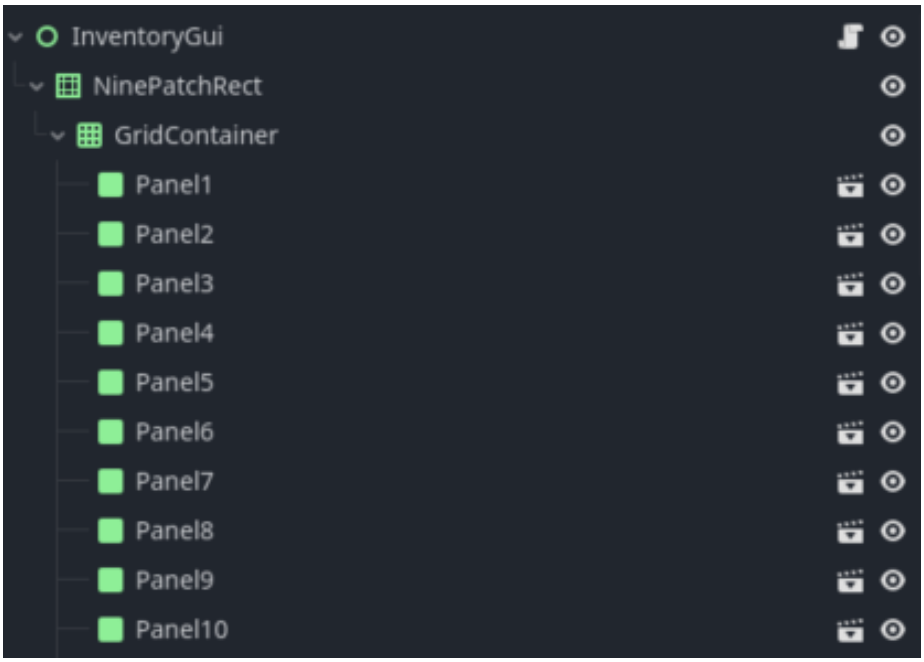
6.1 Le fonctionnement de l'inventaire

Pour l'inventaire, nous commençons par un node Control. Cette classe gère l'état de l'inventaire avec une variable booléenne qui nous permet d'ouvrir et fermer l'inventaire. Dans le script, on crée une fonction qui initialise l'inventaire en le rendant invisible et en activant le traitement des entrées d'utilisateurs. On crée une autre fonction qui détecte lorsque le joueur appuie sur la touche "E" et bascule l'état de l'inventaire entre ouvert et fermé donc respectivement Open et Close.

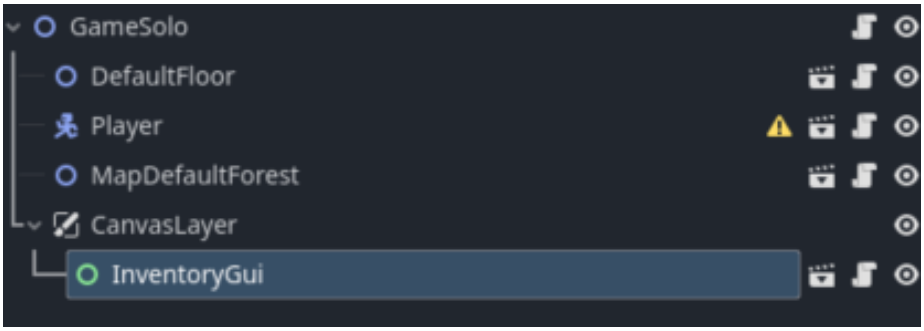
Voici donc notre inventaire, c'est tout de quoi il y a de plus basique, mais s'intéressons nous à sa structure.



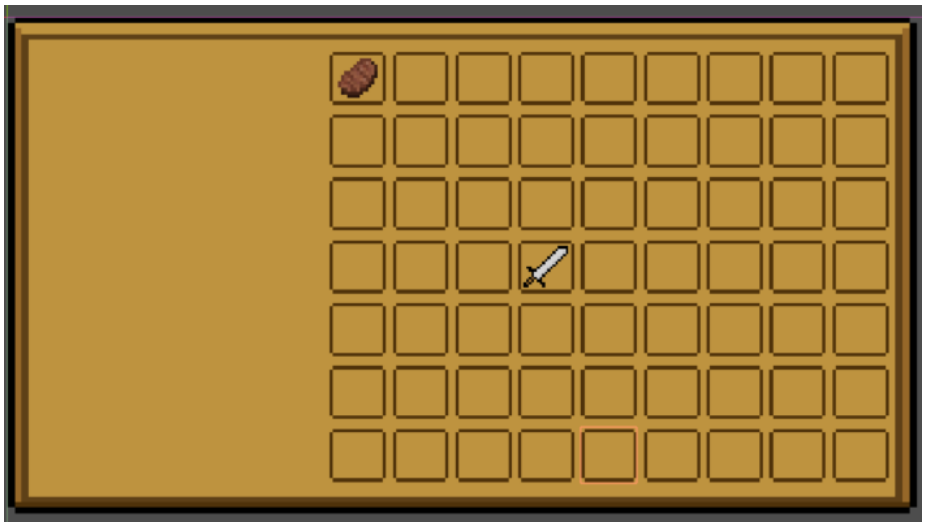
Voici son node Control qu'on renomme "InventoryGui", en Godot un node Control est un type de noeud utilisé pour créer des interfaces utilisateur (UI). Il fait partie de la hiérarchie des noeuds de Godot et est essentiel pour développer des éléments interactifs et visuels dans un jeu. Dedans on retrouve le noeud enfant "NinePatchRect" et "GridContainer" qui servent à répartir les "Panel" qui sont l'espaces ou nous allons stockés les Items.



Maintenant qu'on a crée notre inventaire il reste plus qu'à introduire le code, mais ?! OU EST L'INVENTAIRE ??!!?? Pas de panique, l'inventaire a été bien crée sauf un petit soucis, la postion de l'inventaire. On veut que l'inventaire suit le joueur donc on va utilisé la node "CanvasLayer" pour que la position de l'inventaire soit tout le temps devant nous.



Maintenant que nous pouvons ouvrir et fermer l'inventaire ainsi l'avoir devant nous il faut maintenant faire en sort que chaque cases de l'inventaire dont les "slots" doivent faire apparaître les items qu'on a. Pour cela on créer on nouveau script intitulé "inventoryItem" qui hérite de Resource qui sera utilisé pour les items. On crée ensuite un autre script pour l'inventaire en générale qui va nous



permettre d'accéder au slots.

6.2 Problèmes rencontrées

Quelques problèmes sont apparus dans pendant la mise en place de l'inventaire.

Le premier défi était la mise en place des nodes car la nodes "Control" ne pouvais pas être mise en place avec un node2D, ce problème a nécéssité une restructuration des nodes et ainsi avoir une meilleure optimisation dans la recherche des nodes précise.

Pendant le développemnt de l'inventaire plusieurs problèmes intervient, le premier problème rencontrer était l'IDE de Godot. Celle-ci ne nous averti pas s'il y a des problèmes de code donc on doit lancer le projet à chaque fois pour voir si des problèmes sont présent. La mise en place des touches étaient assez difficile car Godot ne détecter pas les touches nécessaires au fonctionnement.

Une fois le code écrit et le problème avec le node Control résolu il faut crée l'inventaire avec les nodes enfant que Godot propose. Quand nous avons crée l'inventaire et confirmer que le code fonctionne. L'inventaire s'affiche mais au mauvais endroit dur la carte donc nous a ajouté la node "Inventaire" à un node "CanvasLayer" pour que l'inventaire s'affiche de manière correct. Après tout cela on lance le projet et l'inventaire et le code fonctionne à merveille.

Cependant, un autre défi nous attendait : l'accès aux différents

slots de l'inventaire. Pour cela, nous avons dû créer plusieurs éléments héritant de la classe "Resource". Malheureusement, Godot ne détectait pas toujours ces éléments, ce qui a nécessité une investigation approfondie. Nous avons passé beaucoup de temps à comprendre pourquoi certains éléments n'étaient pas toujours reconnus. Après plusieurs essais et erreurs, nous avons finalement identifié que le problème résidait dans la manière dont les ressources étaient chargées et référencées dans le code.

Pour résoudre ce problème, nous avons dû revoir notre approche de gestion des ressources. Nous avons opté pour une méthode plus robuste de chargement et de référencement des éléments, en utilisant des chemins d'accès absolus et en vérifiant systématiquement l'existence des ressources avant de les utiliser. Cette solution a considérablement amélioré la fiabilité de notre système d'inventaire.

Pour conclure cette partie, le développement de l'inventaire dans Godot a été un processus riche en défis techniques. Chaque problème rencontré nous a poussés à approfondir notre compréhension de l'environnement de développement et à trouver des solutions innovantes. Grâce à une restructuration des nodes, une gestion rigoureuse des ressources et une attention particulière aux détails, nous avons pu surmonter ces obstacles et créer un inventaire fonctionnel et performant.

6.3 Ce qui n'a pas été fait

Pour compléter l'implémentation de l'inventaire, plusieurs étapes restent à accomplir. Tout d'abord, il est essentiel d'ajouter des éléments dans l'inventaire. Cela nécessite la création d'une classe dédiée pour représenter les objets. Cette classe devra inclure des attributs tels que le nom de l'objet, sa description, son poids, sa valeur, et éventuellement d'autres caractéristiques pertinentes comme sa rareté ou ses effets spéciaux. Une fois cette classe définie, il faudra implémenter des méthodes pour ajouter, supprimer et afficher les objets dans l'inventaire. Ces méthodes devront être robustes et capables de gérer différents types d'objets, tout en assurant que l'inventaire reste cohérent et bien organisé.

Ensuite, il est important d'implémenter des fonctionnalités permettant d'interagir avec les objets de l'inventaire. Par exemple, il pourrait être utile de permettre aux utilisateurs d'utiliser des ob-

jets, de les équiper ou de les jeter. Pour ce faire, il faudra définir des méthodes spécifiques pour chaque type d'interaction. Par exemple, une méthode "utiliser" pourrait déclencher un effet particulier en fonction de l'objet, tandis qu'une méthode "équiper" pourrait ajouter l'objet à une liste d'équipements actifs. La méthode "jeter" devra quant à elle supprimer l'objet de l'inventaire de manière définitive.

En ce qui concerne les armures, il faudra également créer une sous-classe spécifique pour représenter ces objets particuliers. Les armures auront des attributs supplémentaires comme la défense qu'elles procurent, leur durabilité, et éventuellement des bonus spécifiques. Il sera nécessaire d'implémenter des méthodes pour équiper et déséquiper les armures, en tenant compte des éventuelles restrictions d'équipement.

Il sera également crucial de tester l'inventaire dans différentes situations pour s'assurer de son bon fonctionnement. Cela inclut des tests unitaires pour vérifier que chaque méthode fonctionne correctement, ainsi que des tests d'intégration pour s'assurer que l'inventaire interagit correctement avec d'autres composants du système. Il faudra également tester l'inventaire dans des scénarios réels pour identifier les problèmes éventuels et optimiser les performances. Par exemple, il pourrait être utile de tester l'inventaire avec un grand nombre d'objets pour s'assurer qu'il reste performant. vien discord

7. Système de sauvegarde

7.1 Fonctionnement

Tout session de jeu mérite une pause, pour les jeux en ligne ceci est impossible mais c'est possible pour les jeux "Singleplayer". Mais à quoi ça sert, un système de sauvegarde dans un jeu vidéo est essentiel pour plusieurs raisons. Il permet aux joueurs de conserver leur progression, de reprendre leur partie à un point précis, et d'éviter de perdre leurs avancées en cas de déconnexion ou de fermeture inattendue du jeu.

Son implémentation est simple, on crée un menu de pause dans un jeu. Il permet aux joueurs de mettre le jeu en pause, de sauvegarder leur progression et de charger une partie sauvegardée. Le menu de pause apparaît lorsque le joueur appuie sur une touche spécifique.

Pour le mettre en place on crée deux buttons un pour sauveg-

arder le jeu et un autre pour charger une partie sauvegardée ainsi une variable pour savoir si le jeu est en pause ou non. On crée une fonction récupère les boutons de sauvegarde et de chargement à partir de la scène, elle cache le menu de pause au démarrage en appellant "HidePauseMenu" et elle connecte les boutons "OnSave-Pressed" et "OnLoadPressed" pour gérer les actions de sauvegarde et de chargement. On crée une autre fonction qui gère les entrées utilisateur non traitées. Elle vérifie si l'utilisateur a appuié sur la touche "pause". Si le jeu n'est pas en pause, elle affiche le menu de pause.

7.2 Problèmes rencontrées

Le système de sauvegarde ainsi que le menu de pause sont essentiels au fonctionnement du jeu car on a besoin que les coordonnées du joueur sont sauvegarder et qu'il puisque reprendre le jeu là où ils se sont arrêtés. Il est également important pour la progression du joueur et la gestions des erreurs en cas de bug ou de crash, un système de sauvegarde permet de récupérer la partie sans perdre tout le progrès.

Quels sont les problèmes rencontrées ? Et bien tout d'abord pour commencer on n'a aucune idée comment marche ou implémenter un système de sauvegarde donc on a commencé à apprendre les bases du développemnt de jeu. Une fois les bases acquis on a commencé à le travailler.

Un autre problème courant est la gestion des versions de sauvegarde. Si on apporte des modifications au jeu qui affectent la structure des données sauvegardées, on doit s'assurer que les anciennes sauvegardes restent compatibles avec les nouvelles versions du jeu. Cela peut nécessiter la mise en place de systèmes de migration de données, ce qui peut être complexe et sujet à des erreurs.

On doit gérer les différents états du jeu et décider quelles informations doivent être sauvegardées. Par exemple, on peut vouloir sauvegarder la position du joueur, les objets collectés, les quêtes en cours, etc. Cependant, sauvegarder trop d'informations peut entraîner des fichiers de sauvegarde volumineux et des temps de chargement plus longs. À l'inverse, ne pas sauvegarder suffisamment d'informations peut entraîner une perte de progression pour le joueur.

On doit tester rigoureusement le système de sauvegarde pour

s'assurer qu'il fonctionne correctement dans toutes les situations. Cela inclut la vérification que les sauvegardes sont correctement créées, chargées et que les données sont cohérentes. Les tests doivent également couvrir les cas de figure où le jeu pourrait être interrompu de manière inattendue, comme un crash ou une déconnexion, pour s'assurer que les sauvegardes ne sont pas corrompues.

Conclusion

En conclusion, ce projet a permis de développer plusieurs aspects essentiels d'un jeu vidéo en utilisant le moteur de jeu Godot. Nous avons abordé diverses fonctionnalités, allant de la gestion du personnage à la création de la carte, en passant par l'implémentation de l'inventaire et du système de sauvegarde.

Pour le personnage, nous avons créé des animations de marche gauche et droite en utilisant des assets inspirés et adaptés à nos besoins. Le script gère les animations de manière fluide et cohérente, en utilisant AnimatedSprite2D pour lancer les animations correspondantes en fonction des entrées du joueur. La caméra, centrée sur le personnage, suit ses mouvements avec un léger effet de décalage pour une expérience plus fluide.

La carte du jeu a été conçue avec des assets provenant de OpenGameArt, comprenant des éléments de décor comme l'herbe et les bordures. Nous avons utilisé un script pour générer des objets et des ressources de manière aléatoire, créant ainsi un environnement dynamique. Cependant, la génération procédurale de la carte et la réinitialisation après 12 heures de jeu restent à implémenter.

L'inventaire, géré par un noeud Control, permet d'ouvrir et de fermer l'inventaire en appuyant sur la touche "E". Nous avons structuré l'inventaire avec des noeuds enfants comme NinePatchRect et GridContainer pour organiser les espaces de stockage des items. Pour que l'inventaire suive le joueur, nous avons utilisé le noeud CanvasLayer. Les prochaines étapes incluent l'ajout d'éléments dans l'inventaire, l'implémentation de fonctionnalités d'interaction avec les objets, et des tests pour optimiser les performances.

Enfin, le système de sauvegarde permet aux joueurs de mettre le jeu en pause, de sauvegarder leur progression et de charger une partie sauvegardée. Le menu de pause, activé par une touche spécifique, offre des boutons pour sauvegarder et charger le jeu, assurant ainsi une expérience de jeu fluide et sans interruption.

Ce projet a non seulement permis de développer des compétences techniques en programmation et en design de jeu, mais il a également mis en lumière l'importance de la planification et de l'organisation dans le développement de fonctionnalités complexes

Ce qui reste à faire pour les prochains soutenances

Mouvements du Personnage :

Attaques et récolte de Ressources : Les mouvements d'attaque et de récolte de ressources n'ont pas encore été réalisés. Ces fonctionnalités sont cruciales pour enrichir l'expérience de jeu et permettre aux joueurs d'interagir plus profondément avec l'environnement. Nous devons implémenter ces mouvements d'ici la prochaine soutenance.

Carte du Jeu :

Génération procédurale : La génération procédurale de la carte n'a pas encore été mise en place. Cette fonctionnalité permettrait de créer des environnements variés et dynamiques, offrant une expérience de jeu plus riche et diversifiée. Nous devons développer un algorithme qui génère de nouvelles zones de la carte lorsque le joueur atteint certaines coordonnées.

Réinitialisation de la carte : Il manque également l'ajout de la fonctionnalité qui réinitialise toute la carte, à l'exception du bloc central, après 12 heures de jeu. Cette fonctionnalité est importante pour maintenir l'intérêt et le défi du jeu sur le long terme.

Inventaire :

Pour les prochaines soutenances, il sera essentiel d'ajouter davantage d'items afin de rendre le jeu plus divertissant. Une variété plus large d'objets permettra aux joueurs d'avoir plus d'options et d'interactions, enrichissant ainsi leur expérience de jeu. Il faudra également finaliser les petits détails restants pour garantir une expérience utilisateur fluide et agréable. Ces ajustements incluent la correction de bugs mineurs, l'amélioration de l'interface utilisateur, et l'optimisation des performances globales de l'inventaire. En s'assurant que ces éléments sont bien en place, le jeu pourra offrir une expérience plus immersive et captivante pour les joueurs.

Système de Sauvegarde :

Améliorations : Bien que le système de sauvegarde soit fonctionnel, nous pouvons envisager des améliorations, comme l'ajout de sauvegardes automatiques à intervalles réguliers ou la possibilité de sauvegarder plusieurs fichiers de progression.

Interface Utilisateur : Nous pouvons également améliorer l'interface utilisateur du menu de pause pour rendre les options de sauvegarde et de chargement plus intuitives et accessibles.