Quality Plan

Historique des versions

Version	Date	Changements
1	24/01/2024	Version initiale
1.1	29/01/2024	Modification de l'OBS + ajout des normes et pratiques + tests + tickets + livraison
1.2	30/01/2024	Modification de l'OBS
1.3	19/02/2024	Modification de la méthodologie (Scrum → V)
1.4	02/04/2024	- Précisions sur les rôles - Précision sur l'utilisation de Github - Ajout d'un encart pour signaler la malléabilité des outils utilisés - Modification de la méthodologie de travail - Ajout d'une description des attributs des tickets - Modification des conditions de livraison
1.5	29/04/2024	- Ajout d'une mention à Unity - Ajout d'explications sur les branches "dev" et "master" - Changement de nom sur la partie Scrum

Introduction

Objectif

Ce document définit le plan qualité pour le développement de **Untitled God Game**, un jeu en réalité augmentée sur Meta Quest 3. Il énonce les règles, les processus, les normes, et les meilleures pratiques qui seront appliquées tout au long du projet pour garantir la qualité du produit final.

Il sera révisé régulièrement pour s'adapter aux besoins changeants du projet et pour garantir la livraison d'un produit final de haute qualité.

Organisation de l'Équipe

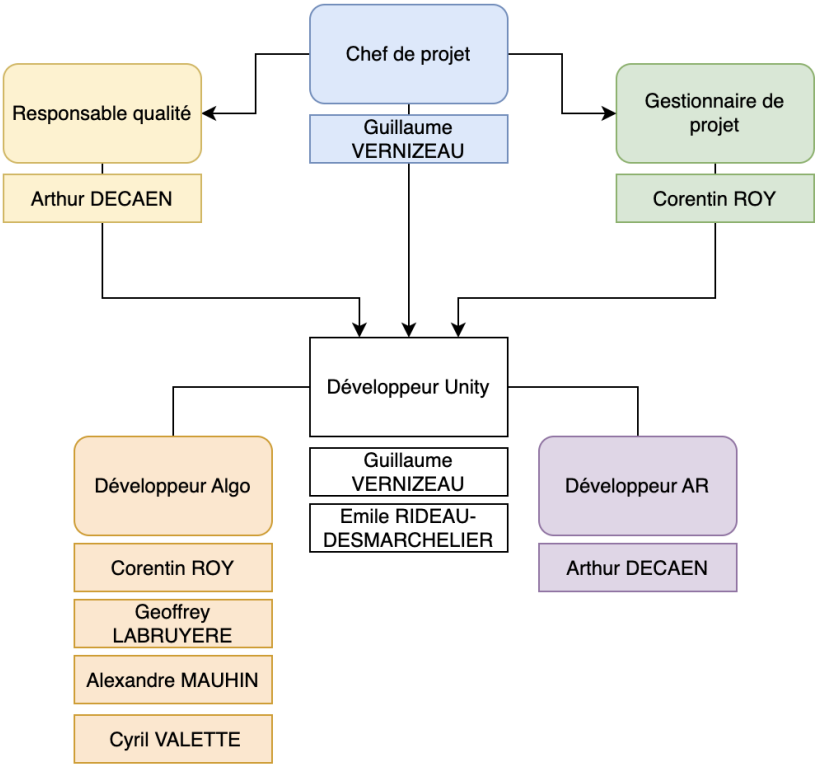
Organisation

Chaque membre du groupe a un droit de parole équivalent et participe aux décisions sur l'avenir du projet. En cas de questionnement plus complexe, c'est au Responsable Technique ou au Chef de projet de trancher.

Rôles et Responsabilités

Rôle	Responsabilités
Chef de projet	Responsable de la coordination générale, de la planification et de la communication.
Gestionnaire de projet	Responsable des tâches, des délais et de la répartition des ressources.
Responsable de la qualité	Garantit la conformité aux normes et aux pratiques. Il tranche également lors des décisions sur l'ajout d'options au projet.
Développeurs Unity	Chargés de la conception et de la programmation générale du jeu. Sans spécialité précise, il est apte à gérer tout types de problématiques et pourra être positionné en soutiens aux équipes spécialisées.
Développeur AR	Chargés de la conception, de la programmation et de l'expérience utilisateur en lien avec la Réalité Augmentée. Au besoin, il pourra aider sur les aspects plus généraux du projet.
Développeur Algo	Chargés de la conception et de la programmation en lien avec les comportement des animaux et de la génération procédurale. Au besoin, il pourra aider sur les aspects plus généraux du projet.

OBS



Méthodologie de travail

La méthodologie de travail appliquée sur ce projet utilise une base Kanban combinée à des réunions régulières pour mettre à jour le backlog. Elle s'appuie sur des points clefs décrits dans les paragraphes suivants.

Backlog

Avant le début du projet, un backlog est établi pour répertorier les tâches à accomplir. Ce backlog sera basé sur les exigences initiales du projet mais sera également régulièrement mis à jour lors des réunions pour s'adapter aux nouveautés et aux changements de priorités.

Cycles de travail

Le projet est découpé en cycles de travail où, contrairement à un Scrum, il n'y a pas d'exigence de rendu. Ces cycles s'étalent sur des périodes de **deux semaines**. Chaque cycle démarre par une réunion visant à définir les tâches à réaliser à partir du backlog pour cette période. Cette réunion est souvent informelle et ne requiert pas de documentation détaillée. Elle offre à l'équipe l'opportunité de discuter des progrès réalisés, des difficultés rencontrées et des améliorations potentielles à apporter au processus de travail.

Tableau Kanban

Le tableau Kanban est utilisé pour visualiser le flux de travail et gérer les tâches à mesure qu'elles avancent à travers le processus. Il sera divisé en 6 catégories.

Nom	Description
Backlog	• Contient toutes les tâches planifiées pour le projet, initialement définies dans le backlog du projet. • Les éléments dans cette colonne peuvent être des idées de fonctionnalités, des bugs signalés, des demandes de changement, etc. • Les tâches dans cette colonne sont généralement décrites de manière succincte et peuvent être priorisées en fonction de leur importance et de leur urgence. • Les éléments de cette colonne pourront être modifiés pendant les réunions de début de sprint.
To Do (À faire)	• Contient toutes les tâches qui devront être traitées pendant le sprint en cours.
Work In Progress (En cours)	• Les tâches qui sont actuellement en cours de traitement sont déplacées dans cette colonne. • Les tâches de cette colonne sont forcément attribuées à des personnes.
Verifying (Vérification)	• Une fois qu'une tâche est terminée par un membre de l'équipe, elle est déplacée dans cette colonne pour être vérifiée. • Dans cette colonne, une revue ou un test peut être effectué pour s'assurer que la tâche est complète et conforme aux normes définies.
Done (Terminé)	• Une fois qu'une tâche a été vérifiée et approuvée, elle est déplacée dans cette colonne. • Cette colonne indique que la tâche a été validée et est valide pour ce sprint. • Une fois la tâche terminée, elle est merge sur notre branche “dev”.
Archived (Archives)	• Contient toutes les tâches qui ont été complétées et finalisées avec succès. • En fin de cycle, toutes les tâches “Done” sont transférées dans les archives. • En fin de cycle, toutes les tâches présentent sur “dev” sont déplacées sur la branche “master“.

Outils

Unity

Unity sera le moteur de jeu principal utilisé pour le développement de **Untitled God Game**. C'est un outil puissant et flexible qui prend en charge de manière simple la réalité augmentée.

Github

Pour la gestion des versions, l'équipe utilisera **Git** au travers de la plateforme Github**,** permettant un suivi précis des modifications apportées au code source et facilitant la collaboration.

Notion

Egalement, l'équipe optera pour **Notion**, qui sera utilisé comme un hub central étant donné sa nature customisable (tous types de documents peuvent être créés) et participative (pensé pour être utilisé en organisation). Nous avons préféré cette solution à un Trello, moins polyvalent, ou à un Jira, plus lourd.

Le
Notion participatif hébergera donc :
→ la gestion des sprints et la tenue du tableau
Kanban (backlog, todo, wip, ..)
→ la documentation interne
→ le ou les calendriers
→ les notes diverses

> 💡 Il est à noter que les outils pourront être modifiés si des limitations se font ressentir pendant le développement.

Normes et Pratiques

Normes des commits

Les commits liés au projet se basent sur la norme “Commits Conventionnels”. Pour résumer, les commits devront être de la forme suivante : `<Mot_clef>: <Description_brève>`. Si la description brève n’est pas assez précise sur les

modifications apportées, il est alors préférable d'ajouter une description détaillée.

Mot Clef	Utilisation
FEAT	Ajout d'une nouvelle fonctionnalité.
UP	Modifications mineures.
DOC	Mise à jour des documentations, commentaires...
DEL	Suppression d'éléments du projet.
FIX	Correction d'un bug ou d'une fonctionnalité bancale.

Utilisation des branches

Chaque fonctionnalité est développée sur sa propre branche, nommée au format `<Mot_clef>/<Fonctionnalité>`.

Mot Clef	Utilisation
feat	Ajout d'une fonctionnalité
fix	Correction d'un bug

Afin d'éviter tout problème en production, tout rebase vers la branche `master` doit être vérifié par une tierce personne lors d'une Pull request. Le projet se doit de fonctionner correctement sur cette branche en tout temps.

Bonnes pratiques (soft)

Cette partie liste les bonnes pratiques à adopter pour rendre son code le plus propre possible et permettre aux autres développeurs de trouver facilement ce qu'ils cherchent.

- Utilisation systématique du **camelCase** pour les variables, fonctions, types... (ex: `maFonction`).
- Utilisation systématique du **PascalCase** pour les méthodes et les classes (ex: `MaClasse`).
- Donner des noms logiques à ses éléments (pas de variables a, z, test...).
- Chaque classe doit être dans un fichier différent.
- Privilégier au maximum les attributs **private** au sein des classes.

- Utiliser au maximum l’héritage pour éviter de répéter inutilement du code.
- Essayer de limiter la taille des lignes et des fonctions pour rendre le code lisible (ex: 100 char par lignes, 30 lignes par fonctions...).
- Chaque fonction doit avoir sa propre responsabilité et ne pas multiplier les tâches.

Tests de l'Application

Durant chaque sprint, les tâches sont vérifiées avant d’être validées. Cette vérification s’appuie sur divers tests décrits ci-après. Il n’est pas nécessaire d’effectuer chaque type de test pour valider une tâche.

Types de Tests

1. **Tests d'Intégration : P**our s'assurer que les différentes parties du jeu interagissent correctement les unes avec les autres.
2. **Tests d'Acceptation :** Pour vérifier que les fonctionnalités répondent aux exigences spécifiées par les parties prenantes. Cette partie sera principalement à la charge du **Responsable Qualité**.
3. **Tests de Performance :** Pour garantir que le jeu fonctionne de manière optimale, même dans des conditions d'utilisation intensives.

> 💡 L’écriture d’une séquence de tests sera effectuée durant le développement afin de s’assurer que tout le monde suive un protocole similaire. Il n’est malheureusement pas possible de prévoir à l’avance les différentes étapes ce qu’il contiendra.

Gestion des Tickets

Outil de Suivi des Tickets

L'équipe utilisera **Notion** pour attribuer, suivre et résoudre les problèmes et les tâches. Chaque ticket sera assigné à un membre spécifique de l'équipe pour assurer la responsabilité et la transparence.

Cycle de Vie des Tickets

1. **Création :** Les tickets seront créés en identifiant les problèmes, les nouvelles fonctionnalités ou les améliorations nécessaires.
2. **Attribution :** Chaque ticket sera attribué par le Gestionnaire de projet à un membre de l'équipe en fonction de ses compétences et de sa charge de travail actuelle.
3. **Développement :** Une fois attribué, le membre de l'équipe commencera à travailler sur la résolution du problème ou le développement de la fonctionnalité.
4. **Test :** Après le développement, le ticket passera par une phase de test pour assurer sa qualité.
5. **Validation :** Une fois que le ticket a passé les tests, il sera validé par le Responsable de la Qualité avant d'être considéré comme résolu.

Attributs des tickets

Les tickets auront différents attributs afin de pouvoir facilement les trier, classés, etc... Ces attributs sont amenés à être modifiés durant le cycle de développement afin de satisfaire au mieux l'équipe de développement. Cependant, quelques attributs sont immuables et sont listés dans le tableau suivant :

Nom	Définition
Standby	`Vrai/Faux` Permet de définir si un ticket est en attente (diverses raisons sont possibles).
Story Point	`Entier` Difficulté estimée du ticket sur une échelle de 1 à 5. Doit être spécifié avant un passage en WIP.
Team	`Personne` Membre(s) de l'équipe en charge de la gestion du ticket.
Priority	`Entier` sur une échelle de 1 à 3.
Blocked by...	`Liste de tâches` Permet de définir les tâches qui bloquent la mise en chantier du ticket.

Definition of Ready (DoR) pour un Ticket

Un ticket est considéré comme prêt à être traité lorsqu'il remplit tous les critères suivants :

- Il n'est bloqué par aucun autre ticket.

- Son objectif est clairement défini à travers son titre et sa description.
- La priorité du ticket est définie.

Ces éléments garantissent que le ticket est clair, priorisé, et prêt à être pris en charge par l'équipe.

Les autres attributs des tickets ne sont obligatoires qu’au moment du passage en WIP.

Definition of Done (DoD) pour un Ticket

Un ticket est considéré comme validé à partir du moment où il respecte les points suivants :

- La fonctionnalité livrée est conforme à la description de la tâche.
- La fonctionnalité a été testée manuellement et validée.
- Tous les tests automatisés relatifs à la tâche passent sans erreurs.
- Le code a été relu et approuvé par un pair (merge request/code review).
- La fonctionnalité a été livrée sur la branche principale.
- La documentation a été mise à jour.
- La tâche est marquée comme "done" dans Notion.

Gestion des Livraisons

Aucune livraison n’est prévue avant la fin du développement.

Le client peut demander une démo lors des différents suivis, montrant l’avancement à un moment T. Aucun engagement n’est fait vis à vis de la stabilité de la démo. Nous nous réservons le droit de ne pas fournir de démo sans qu’un motif ne soit nécessaire.

A la fin du développement, un fichier `.apk` sera disponible en téléchargement sur Github. Ce fichier sera accompagné d’une documentation détaillée des différentes options offertes à l’utilisateur dans le cadre de l’ajout de créatures personnalisées.

KPIs

Techniques

- Nombre d’animaux sans perte de FPS.
- Pourcentage des joueurs qui remonte un bug bloquant.

Jeu

- Durée moyenne d’une session.
- Pourcentage des joueurs terminant le tutoriel.
- Pourcentage des utilisateurs voulant recommencer une partie après une première session.
- Pourcentage des utilisateurs qui recommande le jeu à un tiers.
- Nombre de création d’un nouvel animal à partir des outils fournis.