Compte Rendu Global du Projet

Introduction

Le projet Blue Prince est un travail réalisé dans le cadre du module de Programmation Orientée Objet. Il consiste à créer un jeu d'exploration dans un manoir généré dynamiquement. Le joueur doit progresser de salle en salle, gérer ses ressources, franchir des portes verrouillées et atteindre l'antichambre finale située au sommet du manoir. Ce projet nous a permis de pratiquer la conception logicielle, le travail en équipe, la génération procédurale et l'organisation d'un projet réel.

Objectifs du projet

• Développer un jeu complet en Python basé sur la POO.
• Structurer un programme autour de plusieurs classes interconnectées (Room, Door, Player, Inventory, Manor).
• Mettre en place une génération dynamique et cohérente du manoir.
• Gérer les interactions du joueur avec les salles et les portes.
• Collaborer efficacement via GitHub et organiser un véritable projet d'équipe.

Répartition du travail

Notre groupe était composé de trois membres : **Souleymane**, **Sergen** et **Gabriel**.

• **Souleymane** : responsable de la structure interne du manoir (door.py, room.py, manoir.py). Il a géré les portes, les salles, la génération, et les effets associés.

• **Sergen** : responsable de la partie graphique sous Pygame. Il a conçu l'affichage, la gestion des touches et la représentation du manoir et du joueur.

• **Gabriel** : responsable du joueur et de l'inventaire. Il a géré les déplacements, les ressources, les clés, la consommation de pas, ainsi que de nombreux tests.

Démarche Suivie

1. Analyse du sujet

Nous avons identifié les classes nécessaires et leur rôle dans le fonctionnement global du manoir. Notre priorité fut de structurer correctement la logique interne avant d'ajouter l'affichage.

2. Conception modulaire du projet

Nous avons réparti le projet en plusieurs fichiers Python distincts : door.py, room.py, manoir.py, inventory.py, player.py et game.py. Cette approche a permis une meilleure lisibilité, une simplification des tests, et une répartition claire du travail.

3. Développement progressif

Souleymane a construit la structure générale du manoir et la logique des salles. Sergen a intégré cette logique dans Pygame pour permettre une visualisation claire. Gabriel a implémenté le joueur et son inventaire, garantissant une interaction cohérente avec le manoir.

4. Collaboration GitHub

Chaque membre travaillait sur une branche personnelle afin d'éviter les conflits. Les Pull Requests permettaient d'intégrer le travail de chacun proprement en assurant une revue et une validation systématique.

5. Phase de tests

Nous avons testé la cohérence du manoir, les effets des salles, l’ouverture des portes, le comportement de l’inventaire et l'affichage général sous Pygame.

Problèmes rencontrés

1. Git et VS Code
Au début, Git n’était pas reconnu car le projet n’était pas ouvert dans le bon dossier. Nous avons appris à utiliser correctement Git, les branches, les push et les Pull Requests.

2. Imports circulaires
Certaines classes s’importaient mutuellement, provoquant des erreurs Python. La solution fut d'utiliser des imports localisés dans certaines méthodes.

3. Génération déséquilibrée
La génération procédurale créait parfois des zones trop difficiles. Un équilibrage a été fait pour ajuster les probabilités.

4. Ajustements graphiques
Plusieurs tests ont été nécessaires pour rendre l’affichage clair et fluide dans Pygame.

Conclusion

Ce projet nous a permis de progresser fortement en POO, en conception logicielle et en travail collaboratif. Nous avons obtenu un jeu fonctionnel, modulable et bien structuré, prêt à être étendu avec de nouvelles fonctionnalités.

Choix de Conception

1. Architecture orientée objet stricte
Nous avons choisi de structurer le jeu entièrement autour de classes, afin de permettre une modularité maximale. Chaque élément du jeu possède son comportement propre : Room pour les salles, Door pour les portes, Manor pour la grille globale, Player pour les déplacements et Inventory pour les ressources.

2. Séparation logique / graphique
La logique interne du jeu fonctionne même sans interface Pygame. Ce choix permet de modifier ou remplacer l’affichage sans toucher aux règles du jeu, ce qui rend le projet plus propre et maintenable.

3. Génération progressive du manoir
Plutôt que de générer toutes les salles à l’avance, nous générons les nouvelles salles au fur et à mesure que le joueur progresse.

4. Système de salles simple et efficace
Les salles ont été conçues avec des effets clairs : nourriture, piège, trésor, neutre. Cela rend le gameplay lisible tout en permettant un équilibre cohérent.

5. Niveaux de verrous pour les portes
Nous avons défini trois niveaux : unlocked, locked, double locked. Ce système introduit une vraie progression de difficulté : facile en bas, plus dur en haut.