Rapport Projet Python

Groupe :
Albert Boghossian
Joy Baroudi
Thierry Huang

Recréation:

BLUE PRINCE®

Responsable UE :
Louis Annabi

UM4RBT11

Rapport :

Introduction:

Ce projet consiste à recréer une version inspirée du jeu Blue Prince, un jeu d'exploration procédurale dans lequel le joueur découvre un manoir salle par salle.
L'objectif principal était de construire une architecture logicielle modulaire permettant :

- la génération dynamique des salles, coffres, casiers et loot,
- la gestion d'un inventaire complexe avec objets consommables, nourriture et permanents,
- la persistance de la progression du joueur,
- la gestion d'interfaces avec Pygame (affichage, HUD, commandes, boutiques).

Le projet a nécessité la création d'un moteur de jeu complet, avec un modèle interne du manoir, un catalogue de salles, un système de tirage probabiliste, et une interface graphique interactive.

Étapes de réalisation du projet:

La conception du projet Blue Prince s'est faite par étapes successives, chacune construite sur les décisions et apprentissages des précédentes. Dans un premier temps, nous avons posé les fondations du fonctionnement interne du jeu en développant les différentes classes liées aux objets : consommables, permanents, nourriture, et système d'inventaire. L'objectif était de disposer d'un noyau logique suffisamment solide pour permettre, par la suite, une intégration plus naturelle avec le reste du gameplay.(le backend étant la priorité initiale)

Parallèlement à cette phase, nous avons commencé à expérimenter avec la bibliothèque Pygame, que nous ne connaissions pas du tout au départ. Nous avons donc créé une version minimale de l'affichage, qui est une fenêtre simple, quelques éléments graphiques statiques, et la détection des événements clavier. Cette familiarisation précoce avec le moteur graphique s'est révélée utile, car elle nous a permis d'éviter de découvrir Pygame trop tard dans le développement.

Une fois ces bases en place, nous nous sommes attaqués à l'un des aspects les plus déterminants du projet : la génération aléatoire des salles. Nous avons développé un système de tirage reposant sur plusieurs facteurs (rareté, couleur, prix en gemmes, contraintes de placement) pour parvenir à une génération cohérente dans l'espace, mais suffisamment variée pour garder un intérêt ludique. Cette partie du travail a nécessité de nombreux ajustements, car les premières versions donnaient soit des tirages trop restreints, soit des configurations incohérentes du manoir.

La phase suivante a consisté à étendre cette logique aléatoire aux objets présents dans les salles : coffres, casiers, digspots, objets au sol, ainsi que la manière dont ils interagissent avec l'inventaire du joueur. Nous avons mis en place un système complet de conteneurs, chacun avec ses règles d'ouverture et de génération de contenu. Il a également fallu gérer la persistance du loot, pour qu'une salle explorée ne produise pas un nouveau contenu à chaque visite. Cette étape nous a obligés à structurer davantage le SalleManager et à introduire un stockage local du contenu généré.

Enfin, une fois la plupart des mécanismes internes stabilisés, nous avons entrepris la tâche, plus exigeante, d'intégrer ces fonctionnalités dans une interface visuelle cohérente. Cela incluait : les déplacements dans le manoir, l'affichage du joueur, la gestion correcte des portes, les interactions avec les conteneurs et le loot, ainsi que l'intégration d'une boutique pour les salles jaunes. Cette dernière phase nous a confrontés à de nombreux problèmes pratiques : synchronisation entre l'état interne et l'affichage, erreurs liées aux imports circulaires, gestion correcte des collisions et des directions, et un important travail d'ajustement général pour rendre l'expérience de jeu fluide et compréhensible.

Au final, la progression du projet s'est apparentée à une construction incrémentale : d'abord un moteur logique abstrait, ensuite une couche procédurale plus riche, puis une gestion durable de l'état du monde, et enfin une interface visuelle qui relie tous ces éléments dans un ensemble jouable. Cette structuration par étapes nous a permis d'identifier progressivement les points faibles du système et de les améliorer jusqu'à obtenir un résultat cohérent. Mais Biensur, toutes ces étapes étaient accompagnés de plusieurs obstacles, dont on va détailler dans la page suivante.

Difficultées Rencontrées :

Au fur et à mesure du développement, nous avons été confrontés à une série de difficultés techniques qui ont parfois nécessité de revoir des pans entiers de notre architecture. Certaines erreurs étaient isolées et faciles à corriger ; d'autres, en revanche, étaient beaucoup plus subtiles car elles se combinaient entre elles, ce qui rendait leur origine difficile à identifier.

1. Conditions de placement trop strictes ou incohérentes

Au début, les conditions que nous avions imposées pour déterminer si une salle pouvait être placée étaient beaucoup trop restrictives. Nous voulions éviter des situations impossibles (ex : une salle sans porte alignée), mais les filtres étaient tellement stricts que presque aucune salle n'était tirée dans certaines directions.
Cela créait de fausses impressions de bug dans le tirage aléatoire lui-même. Nous avons dû revoir la logique, assouplir certains contrôles et mieux structurer la manière dont peut_etre_placee() interprète la géométrie du manoir.

2. Gestion des rotations des salles

Les salles étaient définies avec leurs portes dans un ordre précis, mais notre système de rotation n'était pas compatible avec les images des salles affichées dans Pygame.

Résultat :

- certaines salles avaient une porte sur l'image mais pas dans leurs données logiques,
- et inversement, ce qui menait à des chemins totalement incohérents.

Nous avons finalement décidé d'abandonner la rotation visuelle pour éviter que l'image ne contredise les données internes.

3. Problèmes de déplacement dans le manoir

Cette partie a été l'une des plus délicates, car elle combinait plusieurs sources de bugs simultanément.

Nous avons rencontré :

- Le joueur pouvait traverser des murs quand les tests de collision étaient mal synchronisés avec les portes.
- Certains déplacements étaient autorisés alors qu'il n'existait pas de porte dans la salle adjacente (problème de porte inversée manquante).
- Dans d'autres cas, le joueur ne pouvait pas passer même lorsque la porte était bel et bien déclarée ouverte.
- Des portes ouvertes d'un côté n'étaient pas enregistrées de l'autre côté, ce qui créait un comportement asymétrique.

Ce qui rendait la correction difficile était le fait que chaque bug en cachait un autre.
Un mauvais index de porte entraînait un mauvais tirage, qui induisait un mauvais test de cohérence, qui donnait un déplacement illogique.
Nous avons dû reconstruire la gestion des directions, unifier les systèmes d'indexation (Sud/Ouest/Nord/Est), et introduire un mécanisme de "porte miroir" pour garantir la symétrie.

4. Pooling des salles beaucoup trop restreint

À cause d'un mauvais enchaînement entre la rareté, les conditions et les poids probabilistes, certaines salles étaient presque impossibles à tirer. Même avec un catalogue riche, seules deux ou trois salles revenaient constamment.

La difficulté venait du fait que le problème ne se manifestait pas comme une erreur, mais comme un comportement appauvrissant du tirage.

Nous avons réécrit la logique de pooling pour s'assurer que :

- les poids soient normalisés correctement,
- les contraintes soient filtrées dans le bon ordre,
- et qu'au moins une salle gratuite apparaisse dans la première colonne.

5. Compteur total des salles non décrémenté

Il existait un bug important où le compteur des salles disponibles ne se réduisait pas lors du placement.
Cela menait à des situations absurdes où certaines salles apparaissaient bien plus souvent que prévu, même si leur nombre théorique était très limité.
Le correctif a consisté à intégrer la décrémentation au bon moment, et non pas dans plusieurs endroits du code, ce qui créait des duplications ou des oublis.

6. Génération des items et persistance du contenu des salles

Nous avons fait face à plusieurs dysfonctionnements successifs :

- Les salles réaffichaient du nouveau loot à chaque entrée, car le stockage local n'était pas encore en place.
- Certains conteneurs étaient régénérés alors qu'ils avaient déjà été ouverts.
- Les digspots spawnaient de manière incohérente, parfois dans des salles qui n'auraient pas dû en avoir.

Ces bugs étaient liés au fait que différentes parties du gestionnaire de salles stockaient chacune leur propre version des données.
Nous avons créé un modèle unique via RoomCell qui a permis d'atteindre une persistance cohérente.

7. Intégration théorique ↔ interface graphique (dans Pygame)

C'est probablement la partie la plus chronophage.
L'interface graphique ne se contente pas d'afficher le jeu : elle doit refléter exactement l'état interne.
Nous avons dû corriger :

- des décalages de coordonnées entre la grille logique et les pixels,
- la mise à jour incorrecte de l'écran après la fermeture de la boutique,
- des artefacts visuels causés par la redéfinition de la surface Pygame,
- le clignotement de certains éléments lors d'un changement de salle,
- des événements clavier interceptés dans le mauvais contexte (menu vs déplacement).

8. Gestion de dizaines de fichiers et risque de dépendances circulaires

Au fur et à mesure que le projet grandissait, les imports devenaient difficiles à gérer.
Plusieurs modules dépendaient les uns des autres, et il nous est arrivé de tomber dans des dépendances circulaires qui provoquaient des crashes au lancement.
Pour corriger cela, nous avons dû repenser la structure des classes, centraliser certaines fonctionnalités et éviter que des modules logiquement indépendants s'appellent mutuellement.

9. Menus du shop qui redimensionnaient le tableau principal

Lors de la fermeture de la boutique, l'écran principal récupérait parfois une surface Pygame incorrecte ou réduite.
Ce bug était dû au fait que la boutique manipulait directement la surface active.
Nous avons réparé cela en uniformisant l'accès via pygame.display.get_surface() et en évitant de redéfinir la fenêtre.

Diagramme UML :

Telecharger l'image du diagramme haute-définition ici🔗