

Rapport Python POO Projet

HAFEJI Taaha, TLILI Israa, CULLET Thibault

Dans le cadre du projet de programmation objet en Python, nous avons développé une version simplifiée du jeu *Blue Prince*. L'objectif de ce projet était de mettre en pratique les notions vues en cours en réalisant un petit jeu complet, avec une interface graphique créée à l'aide de **Pygame**.

Notre programme reprend les bases du jeu original : le joueur explore un manoir, découvre de nouvelles pièces tirées aléatoirement et gère différentes ressources comme les pas, les clés ou les gemmes.

L'utilisation de classes dans ce projet Blue Prince permet de manipuler les aspects fondamentaux du jeu plus facilement. Chaque classe représente une entité du jeu avec ses données (attributs) et ses comportements (méthodes).

Pièce :

Très utile d'implémenter une classe pour les pièces notamment pour accéder et modifier facilement ses attributs, on y a aussi implémenté des méthodes comme l'attribution des objets aléatoirement.

Ses attributs sont, le nom de la pièce, ses connexions (portes), sa rareté, son image, son coût en gemmes. Ces attributs nous permettent de modifier une pièce facilement et en ajouter en se renseignant sur ses attributs à l'aide du wiki du jeu Blue Prince .

- L'attribut du nom permet de gérer les effets uniques à certaines pièces comme la Bedroom
- L'attribut 'connexions' qui est très important nous indique les directions de déplacement possibles avec la pièce (N,S,E ou W). Cela va nous permettre de gérer la rotation des pièces pour proposer au joueur 3 pièces qui respectent les conditions des pièces comme celles où aucune connexion ne doit viser l'extérieur du manoir.
- L'attribut de rareté comme son nom l'indique permet à certaines pièces d'être plus rare ou plus commune que les autres en fonction de leur niveau de rareté allant de 1 pour les communes à 3 pour les rares
- L'attribut du coût en gemmes correspond aux prix de sélection de chaque pièce, le cahier des charges nous dit que si on a 0 gemmes une pièce avec un coût à 0 gemmes doit forcément être proposée
- En connexion avec la classe Objet on doit peut-être gérer les objets présents aléatoirement dans chaque pièce en plus d'effets garantis par certaines pièces.
- Chaque pièce possède des portes qui vont avoir un niveau de verrouillage allant de 0 à 2 et on veut savoir donc si une porte a été déverrouillée ou pas et en fonction des connexions de la pièce décrire aléatoirement le verrouillage de ces connexions

En somme la classe Pièces permet de gérer toutes les interactions du joueur dans une pièce, quand il va déverrouiller une porte et choisir les pièces suivantes, quand il entre dans la pièce s'il trouve des choses à l'intérieur et comment la pièce va être, dans quel sens.

Objet :

La classe objets est simple et permet de définir l'objet "Objet" avec uniquement 4 attributs :

- Son nom : utilisé pour y appliquer son effet, par exemple si c'est de la nourriture, quel nombre de pas le joueur va gagner. De plus ceci permet de gérer l'inventaire et l'affichage pour choisir une icône de l'objets si il est dans l'inventaire
- chance_app : un entier entre 1 et 3 de sa chance d'apparition, Si c'est a 1 l'objet est rare, si c'est a 2 l'objet est peu commun, si c'est a 3 l'objet est commun. Ceci est en connexion avec la classe Pieces car on applique cette chance d'apparition d'un objets a chaque pièces
- is_collected : Pour savoir si l'objet est déjà collecté dans une pièce et ne pas le récupérer 2 fois si on repasse par cette pièce.
- Unique : Pour définir un objet comme unique et donc trouvable qu'une seul fois, ces objets unique ont des pouvoir spéciaux comme ouvrir les portes de niveau 1 ou augmenter les chances d'apparition de certain objets, quand on les trouves une premières fois on ne peux plus les trouver après il reste jusqu'à la fin dans l'inventaire

La classe objets à aussi quelques méthodes qui facilitent leur intégration comme :

- ajouter_au_joueur qui ajoute un certain nombre de clés, de dés ou de gemmes si le joueur en trouve dans les pièces
- retirer_du_joueur qui consomme l'objet si on l'utilise c'est-à-dire que le nombre d'exemplaires de cet objets diminue par 1. On ne peut pas consommer un objets unique

La classe objets créer tout un système aléatoire sur les objets trouvables dans les pièces et définit tous les objets du jeu, et leurs effets si ce sont des consommables ou des objets uniques.

Joueur :

La classe est utilisée pour implémenter les différents aspects du joueur, sa position[colonne, ligne], son nombre d'objets (gemmes, or, clé, dé), ses pas et ses objets permanents.

Tous les attributs sont modifiés avec les actions du joueur en utilisant ses objets qu'il possède dans l'inventaire du joueur qu'on initialise dans la classe Joueur

Les objets uniques vont avoir une valeur en binaire 0 ou 1 pour savoir si on possède ou non l'objet dans l'inventaire

La classe comporte aussi des méthodes comme utiliser_pas, ajouter_pas, ajouter_or qui permettent de modifier très facilement ses attributs en connexion avec la classe objets et la classe pièces permettant de modifier la valeur des attributs en fonction des actions du joueur, par exemple dépenser son or ou des gemmes ou bien perdre 1 pas a chaque déplacement

Conclusion :

Finalement, on instancie ces classes dans la fonction main, ou toute la logique du jeu se trouve, la fonction main crée les classes et les modifie dynamiquement en fonction de la situation, (déplacement, acquisition d'objets...).

Le joueur représente "l'état" du jeu actuel pendant la partie.

L'utilisation de classe en somme :

- Facilite le débogage car les classes en tant que tels ne se connecte que dans le main; ils sont indépendants
- La méthode copier() d'une classe permet de dupliquer ses classes pour qu'un changement de classe ne soit pas générale
- Améliore grandement la lisibilité (le main est très clair)
- Permet d'ajouter très facilement des éléments de jeu (des pièces ou des nvx objets par exemple)
- Réduit les erreurs car il y a moins de cafouillage entre les fichiers python