MOCUMBI TOMAS Thindeka Claudia
DAGHARI Jihen
de VILLARDI de MONTLAUR Cyril

Rapport

Projet de programmation - Blue Prince

Enseignant : Louis ANNABI
Année universitaire : 2025-2026

Présentation générale

Blue Prince est un jeu de type exploration/tirage inspiré des jeux de plateau. Le joueur progresse dans un manoir généré dynamiquement case par case. Chaque nouvelle salle tirée dépend de la forme (portes possibles), de la couleur (type de salle), et des contraintes imposées par la grille déjà construite.

Le jeu repose donc sur trois éléments centraux : Le joueur (déplacement, inventaire, interactions), le manoir (grille, pièces, portes) et le système de tirage (pioche, sélection des salles).

L'ensemble est orchestré par la classe Game, qui gère la boucle principale et les transitions d'état.

Architecture

Nous avons structuré le code selon une logique modulaire :

→ src/ : logique métier du jeu (autre objet, game, grille, inventaire, joueur, objet permanent, piece, pioche, porte)

→ ui/ : affichage et gestion du clavier (Renderer, InputHandler)

→ main.py : point d'entrée, boucle pygame

Ce découpage est fait pour qu'on puisse isoler la logique du jeu de l'affichage, rendre chaque classe indépendante et testable et faciliter l'ajout d'objets, de salles ou d'actions.

Structure

↘ / →	Autre Objet	ObjetPermanent	Pioche 2	Porte	Pièce2	Joueur	Inventaire	Grille	Game	FormePiece	CouleurPiece	EndroitCreuser	Coffre	Casier	Renderer	Input_handler
Autre Objet	-	-	-	-	-	A	A	-	-	-	-	-	-	-	-	-
ObjetPermanent	-	-	-	-	-	A	Ag	-	-	-	-	-	-	-	-	-
Pioche 2	-	-	-	-	C	-	-	A	A	-	-	-	-	-	-	-
Porte	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	-
Pièce2	-	-	-	-	-	-	-	A	-	Ag	Ag	-	-	-	-	-
Joueur	-	-	-	-	-	-	C	A	A	-	-	-	-	-	-	-
Inventaire	A	Ag	-	-	-	-	-	-	A	-	-	-	-	-	-	-
Grille	-	-	-	C	C	-	-	-	-	-	-	C	C	C	-	-
Game	-	-	C	-	-	C	C	C	-	-	-	-	-	-	A	A
FormePiece	-	-	-	-	A	-	-	-	-	-	-	-	-	-	-	-
CouleurPiece	-	-	-	-	A	-	-	-	-	-	-	-	-	-	-	-
EndroitCreuser	-	-	-	-	-	A	-	A	-	-	-	-	-	-	-	-
Coffre	-	-	-	C	-	A	A	A	-	-	-	-	-	-	-	-
Casier	-	-	-	C	-	A	A	A	-	-	-	-	-	-	-	-

Renderer	-	-	-	-	-	A	-	A	A	-	-	-	-	-	-	-
Input_handler	-	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-

Figure 1 : le tableau UML des relations (agrégation, composition, association)

LEGENDE : **H** = héritage **A** = association **Ag** = agrégation **C** = composition **–** = aucune relation directe

On a deux classe majeur mère : deux familles `AutreObjet` et `ObjetPermanent`

Et dix classes mineur fille : *Pomme, Gateau, Banane, Sandwich, Repas
**Pelle, Marteau, KitCrochetage, DetecteurMetaux, PatteLapin

MERE/FILLE	Pomme	Banane	Gâteau	Sandwich	Repas	Pelle	Marteau	KitCrochetage	DetecteurMetaux	PatteLapin
AutreObjet	H	H	H	H	H	-	-	-	-	-
ObjetPermanent	-	-	-	-	-	H	H	H	H	H

Figure 2 : le tableau UML des relations (héritage)

Les classes AutreObjet et ObjetPermanent structurent les deux familles d'objets (consommables et permanents) via l'héritage. L'Inventaire agrège ces objets, tandis que le Joueur compose son inventaire. La Pioche2 construit un catalogue de Piece2, qui sont définies par une `FormePiece` et une `CouleurPiece`. Le Game compose la Grille, la Pioche et le Joueur, et coordonne les interactions entre ces éléments.

Déroulement des classes

`AutreObjet` est la classe mère des objets consommables : Pomme, Banane, Gâteau, Sandwich, Repas.
`ObjetPermanent` est la classe mère des objets permanents : Pelle, Marteau, KitCrochetage, DetecteurMetaux, PatteLapin.

Les autres classes ne font pas d'héritage entre elles. Ces objets appliquent leurs effets sur l'inventaire (par exemple ajouter des pas), mais ils ne possèdent pas l'inventaire.

La classe `ObjetPermanent` est la classe mère de tous les objets permanents : Pelle, Marteau, KitCrochetage, DetecteurMetaux, PatteLapin. L'inventaire, lui, contient les objets permanents dans une liste : Il regroupe plusieurs objets permanents, mais ils peuvent exister même en dehors. L'inventaire est donc comme un sac à dos qui contient des objets, mais ces objets ne sont pas "créés" par le sac.

La classe `Pioche2` gère toutes les `Piece2` possibles du jeu (les salles). Au démarrage, la pioche crée un catalogue de `Piece2`.

Chaque `Piece2` a une `FormePiece` (où sont les portes : croix, couloir, T, etc.), et une `CouleurPiece` (jaune, verte, bleue, rouge…). `Piece2` utilise `FormePiece` et `CouleurPiece` mais ne les "possède" pas vraiment (ce sont des constantes utilisées).

La méthode `_garder_pieces_compatibles` de `Pioche2` appelle `Piece2.peut_etre_posee(grille, ...)` : Pioche2 regarde la Grille pour savoir quelles pièces sont posables. Elle ne crée pas la grille.

La Grille stocke : toutes les `Piece2` placées (`self.__pieces`), et toutes les `Porte` entre les cases (`self.__portes`). Elle crée les `Porte` dans `garantie_porte`, et place les `Piece2` avec `placer_piece`. Les salles et portes **font partie** de la grille : si la grille disparaît, elles n'ont plus de sens.

`Game` crée la grille, le joueur et la pioche : elles appartiennent à l'init de cet affichage.

Le joueur se déplace en utilisant la grille : `grille.deplacement_permis(...)` et le `Game` appelle les méthodes du joueur (`handle_deplacement`).

Sous-explication

- Class AutreObjetClass (Classe de base pour tous les objets utilisables dans le jeu) : en interne : relation d'héritage (mère-fille) avec les class Pomme, Gateau, Banane, Sandwich, Repas, permettant de transmettre un attribut (le nom) et 1 méthode (appliquer à l'inventaire) entre les classes. Cela évite fortement les répétitions. Représenté par une flèche vide en UML. Ils appliquent les objets à l'inventaire du joueur. Ils sont donc instanciés uniquement par inventaire.
- Class ObjetPermanent
- Class Pioche2 utilise la class Piece2, FormePiece, CouleurPiece afin de créer une pioche de départ. Contient une méthode _garder_pieces_compatibles faisant appel à Grille de filtre les pièces posables. Pioche2 est instancié uniquement par la class Game
- Class Porte est définit par son statut (0=deverouillee, 1=verouillée, 2=verouillée à double tour) et par son ouverture (vrai ou faux)
- Class Piece2 est instancié uniquement par Pioche2.
- Class Joueur, représente le joueur ou la joueuse, est instancié uniquement par Game. On lui attribut un inventaire et un emplacement sur la grille.

Remarque

Le jeu est fait autour de la gestion de quatre fonctionnement distinct : Le système de portes, le tirage multiple, la gestion de la game loop et la gestion des ressources :

Tout d'abord chaque pièce possède des portes directionnelles définies par FormePiece. Les portes sont créées par la Grille. Leur état (ouverte/verrouillée) influe sur les déplacements et sur l'utilisation de clés ou outils.

Dans l'état `tirage`, le joueur peut naviguer entre les propositions de salles et choisir une pièce à poser sur la grille. Ensuite, dans le game loop, le Main.py initialise pygame, Game et Renderer. À chaque frame InputHandler lit les touches, la classe Game applique les règle et Renderer affiche les résultats

L'inventaire centralise toutes les probabilités : chance d'obtenir un objet, chance d'obtenir des pièces d'or, chance d'obtenir une clé ou un diamant. Certains objets permanents modifient ces probabilités.

Conclusion

Le projet Blue Prince repose sur une architecture clairement séparée entre : la logique métier (Game, Grille, Joueur, Pioche2, Inventaire), les données et objets du jeu (Piece2, Porte, AutreObjet, ObjetPermanent…), l'interface graphique affiché avec pygame (Renderer), et le système d'entrées (InputHandler).