
**Sorbonne Université**

Master Systèmes Intelligents
UE : Programmation Orientée Objet (POO)
Année universitaire 2024 - 2025

# Projet de Programmation

## Implémentation du jeu « Blue Prince »

**Rapport présenté par :**

KHAMMAR Mohammed Ayoub

KESSOUAR Abderraouf Tarek

WEBER Moritz

**Enseignant :**

Louis Annabi

# Table des matières

# 1 Introduction

Dans le cadre de l'UE *Programmation Orientée Objet*, ce projet avait pour objectif de développer une version simplifiée du jeu vidéo *Blue Prince*. Ce projet constitue une occasion d'appliquer concrètement les notions fondamentales de POO : encapsulation, polymorphisme, héritage, abstraction, modularité et séparation des responsabilités.

Le jeu repose sur la génération dynamique d'un manoir composé de salles aux propriétés variées : portes, objets, rareté, effets spéciaux, conditions de placement et interactions avec le joueur. Notre tâche consistait à concevoir une architecture suffisamment robuste pour accueillir ces mécaniques, tout en restant simple, lisible et extensible.

Nous avons utilisé Python et la bibliothèque `pygame` pour gérer l'affichage et les interactions utilisateur. L'objectif final était de produire un jeu fonctionnel, modulaire, évolutif et fidèle aux mécaniques essentielles du concept original.

Ce rapport détaille le fonctionnement général du jeu, la structure logicielle développée, les choix d'architecture, les interactions clés et les difficultés rencontrées. Une attention particulière est portée à l'organisation orientée objet et aux justifications techniques des choix effectués.

# 2 Présentation globale du fonctionnement du jeu

Cette section présente le déroulement d'une partie et les principales mécaniques du jeu, sans entrer dans les détails techniques de l'implémentation. Elle offre une vue d'ensemble du fonctionnement du jeu tel que l'utilisateur final l'expérimente, avant d'aborder l'architecture logicielle plus en profondeur dans la section suivante.

## 2.1 Objectif et déroulement général

Le jeu propose au joueur d'explorer un manoir composé de salles dont l'agencement se construit progressivement. Le joueur débute dans l'*Entrance Hall* et doit atteindre l'*Antechamber*, située au sommet du manoir. Ce parcours prend la forme d'une progression stratégique où chaque déplacement, chaque porte ouverte et chaque salle choisie influence la suite de la partie.

Une partie se joue en gérant différentes ressources : pas, clés, gemmes, or et objets — tout en prenant en compte les effets des salles déjà placées et celles à venir. Le manoir se construit au rythme des décisions du joueur, ce qui rend chaque partie différente.

## 2.2 Exploration du manoir

Le manoir est représenté par une grille rectangulaire de 5 colonnes et 9 lignes. Initialement, seules quelques salles sont connues ; l'essentiel de l'exploration consiste à ouvrir des portes pour révéler de nouvelles possibilités.

Lorsqu'une porte mène vers une case encore vide, elle déclenche le tirage de nouvelles salles potentielles. Le joueur doit alors choisir laquelle ajouter au manoir, ce qui lui permet de façonner sa progression. Cette mécanique donne au joueur un contrôle partiel sur la structure du manoir : il ne découvre pas un monde figé, mais participe à sa construction.

## 2.3 Déplacements et gestion des ressources

Chaque déplacement consomme un nombre limité de pas. Le joueur doit donc anticiper ses choix pour éviter de se retrouver bloqué. Certaines salles permettent de regagner des pas ou d'obtenir des objets utiles, mais ces opportunités ne sont jamais garanties.

Les portes peuvent être verrouillées selon différents niveaux ; certaines exigent une clé, d'autres acceptent un kit de crochetage. Cette contrainte ajoute une dimension tactique, puisque certains chemins deviennent inaccessibles ou coûteux.

## 2.4 Choix et tirage des salles

À chaque ouverture de porte, trois salles compatibles sont tirées. Ces possibilités varient en fonction de la rareté, des contraintes de placement et des caractéristiques de la salle en cours. Le joueur doit alors choisir celle qui s'intègre le mieux à sa stratégie :

— compléter des chemins existants ;
— sécuriser des bonus ;
— limiter les coûts futurs ;
— éviter des configurations défavorables.

Ce système de tirage mélange hasard contrôlé et planification, rendant chaque partie unique tout en laissant une marge de décision réelle au joueur.

## 2.5 Objets et interactions

Au fil de sa progression, le joueur peut trouver ou obtenir divers objets :

— **consommables**, offrant des bonus immédiats ;
— **permanents**, modifiant durablement la partie ;
— **interactifs**, comme les coffres et les casiers.

Ces objets enrichissent le gameplay en introduisant de nouveaux leviers d'optimisation. Ils encouragent également l'exploration, puisqu'ils apparaissent souvent dans des salles spécifiques ou à la suite de choix particuliers.

## 2.6 Résumé du fonctionnement du jeu

Le cœur du jeu repose sur :

— une exploration progressive du manoir ;
— des choix influencés par les ressources disponibles ;
— un tirage contrôlé de salles déterminant la suite de la partie ;
— une gestion d'objets variés permettant d'adapter la stratégie ;
— une tension constante liée au risque de perdre par manque de pas.

Cette présentation générale pose les bases de la compréhension du jeu. La section suivante détaille la manière dont cette expérience a été traduite en une architecture logicielle structurée et modulaire.

# 3 Architecture générale et conception orientée objet

L'architecture du projet repose sur une organisation claire, inspirée des principes fondamentaux de la programmation orientée objet. L'ensemble a été structuré de façon à séparer nettement la logique métier, la gestion du monde et l'affichage, afin de favoriser la maintenabilité du code et la possibilité d'ajouter de nouvelles fonctionnalités sans modifier les composants existants.

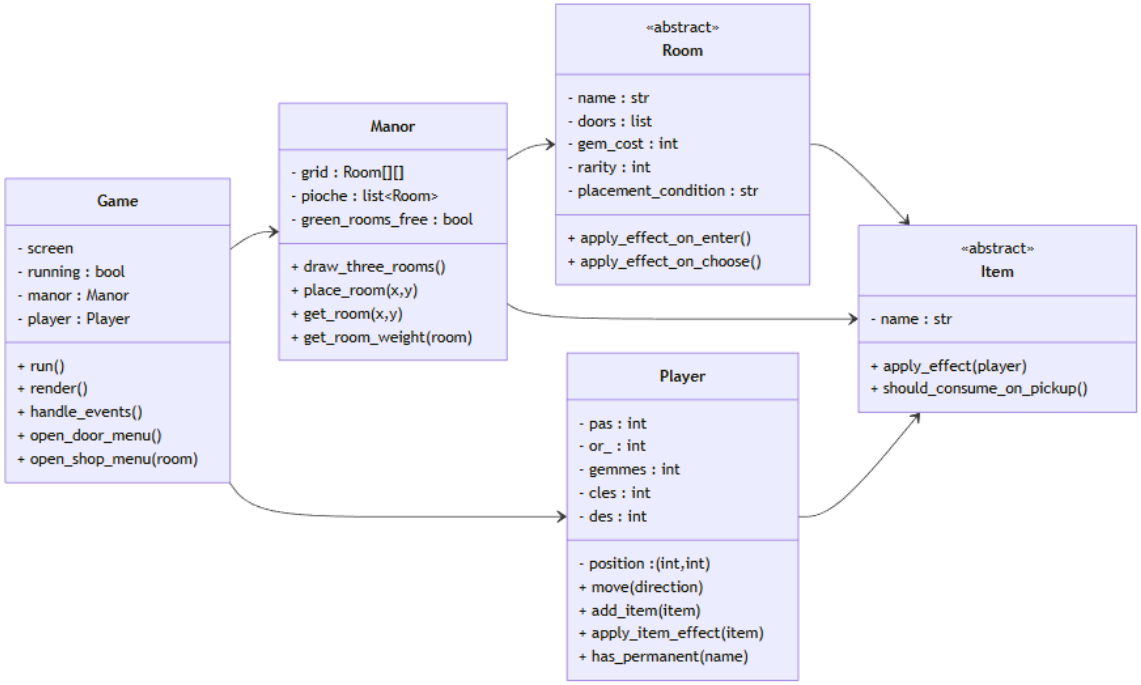


FIGURE 1 – Architecture générale du projet

Cette structure modulaire facilite l'évolution du projet : ajouter une nouvelle salle, introduire un nouvel objet ou modifier le système de tirage ne nécessite pas de modifications dans les modules d'affichage ou dans le contrôleur principal. L'organisation en modules indépendants contribue ainsi à la clarté et à la stabilité globale du code.

## 3.1 Le module `Game` : orchestration et interface

Ce module constitue la couche « contrôle + vue » du projet. Il gère la boucle de jeu, l'écoute des événements clavier et l'affichage du manoir et du HUD via la bibliothèque `pygame`. Il ne contient ainsi aucune règle métier : toutes les décisions liées au placement des salles, aux effets ou à la gestion des ressources sont confiées aux modules spécialisés.

Cette séparation permet d'isoler le code graphique du reste du programme, et rend l'interface interchangeable : une interface console ou même web pourrait être ajoutée sans réécrire la logique du jeu.

## 3.2 Le module `Manor` : modèle principal du monde

Le module `Manor` gère l'ensemble de la structure du manoir : la grille 5×9, le placement des salles, la pioche, les règles de compatibilité et la génération procédurale des salles. Il

centralise également les mécanismes globaux comme la rareté, la rotation des pièces ou les effets persistants de certaines salles (par exemple Terrace ou Conference Room).

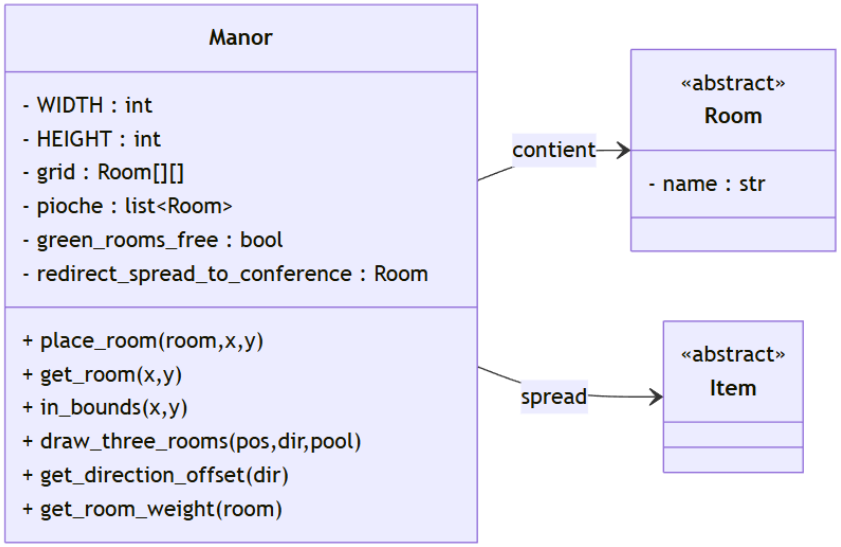


FIGURE 2 – Structure du module `Manor`

En regroupant ces responsabilités au sein d'un seul module, l'ensemble des règles du manoir demeure cohérent et peut évoluer sans affecter les autres composants du projet.

## 3.3 Le module `Player` : état et actions du joueur

La classe `Player` gère toutes les informations relatives au joueur : son inventaire, ses ressources (pas, clés, or, gemmes, dés) et sa position dans la grille. Elle regroupe également les mécanismes de déplacement et l'application des effets d'objets.

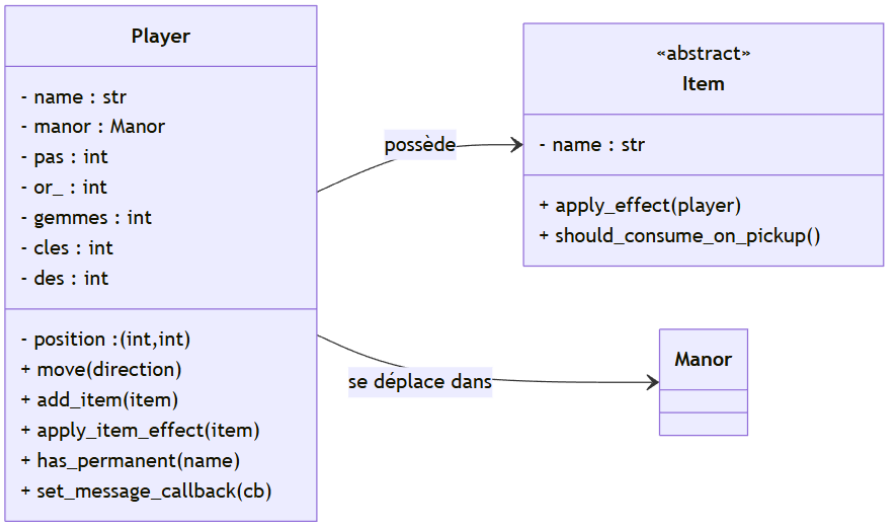


FIGURE 3 – Structure de la classe `Player`

Cette organisation permet au joueur de rester indépendant de la génération du manoir : il interagit avec les salles et les objets, mais ne gère ni le tirage ni les règles de construction

du monde. Ce découplage renforce la lisibilité du code et facilite l'ajout de nouvelles mécaniques.

## 3.4 Système des salles : classe abstraite `Room`

La classe `Room` constitue la base de toutes les salles du manoir. Elle définit les attributs essentiels (portes, couleur, rareté, coût en gemmes) ainsi que les deux méthodes fondamentales permettant de déclencher les effets lors du tirage ou lors de l'entrée dans la salle.

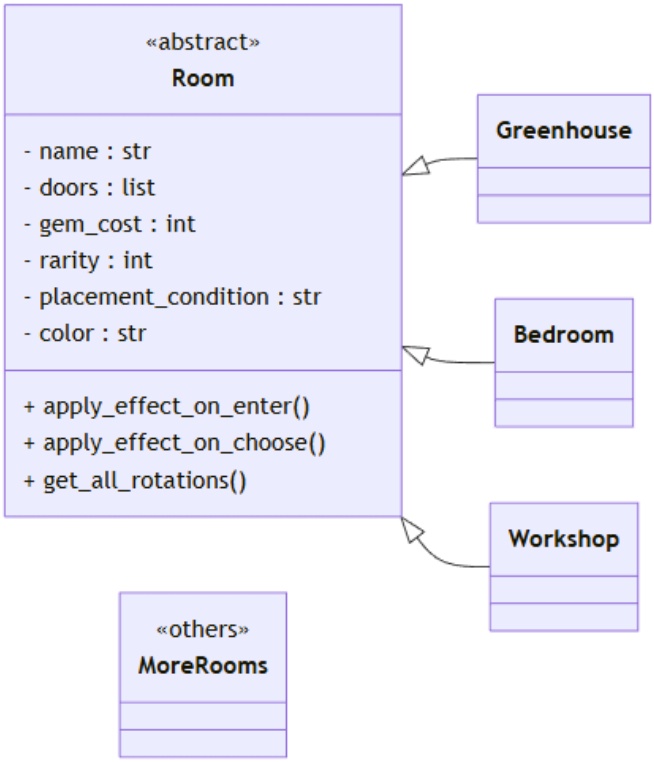


FIGURE 4 – Hiérarchie des salles (`Room`)

Chaque salle spécialisée hérite de cette structure et implemente uniquement la logique qui lui est propre, ce qui rend l'ajout de nouvelles salles très simple et limite le risque d'erreurs dans les mécanismes existants.

## 3.5 Système des objets : classe abstraite `Item`

Les objets trouvés dans le manoir héritent tous de la classe abstraite `Item`. Chaque objet (qu'il soit consommable, permanent ou interactif) implémente son propre comportement via une méthode d'effet appliquée au joueur.

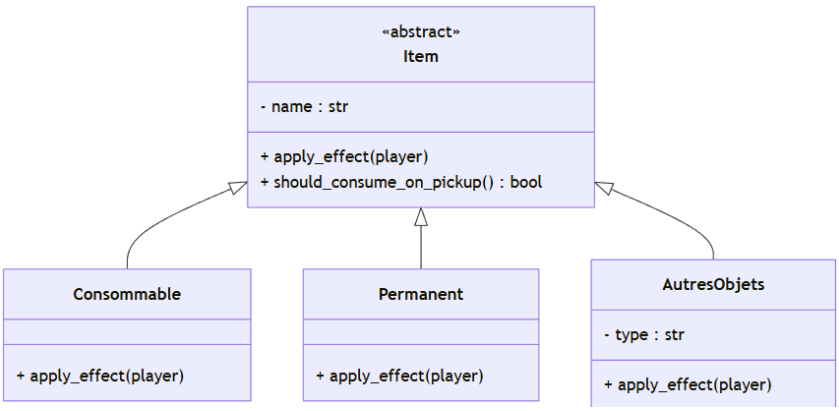


FIGURE 5 – Hiérarchie des objets (`Item`)

Ce fonctionnement polymorphique unifie la gestion de l'inventaire : le joueur peut manipuler n'importe quel objet sans connaître sa nature exacte, et le système peut être étendu en ajoutant simplement une sous-classe.

### 3.6 Synthèse de l'architecture

L'architecture adoptée permet ainsi :

- une séparation nette entre logique métier et interface graphique ;
- une grande flexibilité lors de l'ajout de nouveaux éléments (salles, objets, effets, mécaniques) ;
- une meilleure lisibilité du code grâce à des modules spécialisés ;
- la préservation de la cohérence interne grâce à la centralisation des règles du manoir dans un module unique.

Cette structure a permis d'intégrer de manière cohérente la génération procédurale, les effets de salle, le système d'objets et l'interaction continue entre le joueur et le manoir, tout en conservant un code lisible et extensible.

## 4 Conception orientée objet

La conception du projet s'appuie sur les principes essentiels de la programmation orientée objet, appliqués de manière cohérente à l'ensemble des modules. Cette section présente les mécanismes conceptuels qui structurent le code et les illustre par des exemples concrets issus de l'implémentation.

### 4.1 Héritage

Les classes abstraites `Room` et `Item` constituent les fondations de deux hiérarchies importantes du jeu. Chaque salle spécialisée (par exemple `Greenhouse`, `Bedroom`, `Workshop`) hérite de `Room` et redéfinit uniquement la logique spécifique de son effet. De même, tous les objets (consommables, permanents ou interactifs) dérivent de `Item`. Ce mécanisme permet d'étendre facilement le jeu en introduisant de nouvelles pièces ou de nouveaux objets sans modifier les composants existants.

## 4.2 Polymorphisme

Le polymorphisme est utilisé pour garantir un comportement uniforme malgré la diversité des éléments du jeu. Ainsi, toutes les salles possèdent une méthode `apply_effect_on_enter`, et toutes les sous-classes d'objets implémentent `apply_effect`. Le module `Player` peut donc manipuler n'importe quel objet sans connaître sa nature précise, ce qui simplifie considérablement la gestion de l'inventaire.

## 4.3 Exemples tirés du code

Quelques exemples illustrent l'application concrète des concepts :

- la méthode `room.apply_effect_on_enter(player)` est redéfinie par chaque salle spécialisée pour implémenter son comportement propre ;
- le tirage pondéré des salles dans `Manor.draw_three_rooms()` utilise l'héritage pour manipuler des objets de types différents mais compatibles.

L'ensemble de ces mécanismes contribue à une architecture modulaire, cohérente et simple à faire évoluer.

# 5 Conclusion

Ce projet nous a permis d'appliquer de manière concrète les notions de programmation orientée objet étudiées durant l'UE. La conception d'un jeu dynamique fondé sur de nombreuses interactions, des ressources variées et une génération procédurale nous a confrontés à des problématiques complexes et enrichissantes.

Nous avons conçu une architecture modulaire, maintenable et extensible, tout en mettant en œuvre des mécanismes avancés tels que :

— la gestion de l'aléatoire pondéré ;
— les effets de salles et leur propagation ;
— l'inventaire polymorphique du joueur ;
— l'intégration d'une interface graphique ;
— la manipulation d'une grille dynamique orientée contraintes.

Le travail collaboratif a également représenté un volet essentiel du projet. L'usage rigoureux de Git a favorisé la coordination, la qualité du code et la résolution efficace des conflits.

Enfin, ce projet ouvre la voie à de nombreuses améliorations : enrichissement des salles, optimisation du moteur de tirage, amélioration de l'interface ou intégration de nouvelles mécaniques inspirées du jeu original.