



---

## Shopping Adventure: La quête du Meilleur Deal !

---



## SOMMAIRE DU RAPPORT

### **1.guide utilisateur**

- 1) Comment jouer au jeu : description détaillée du projet**
- 2) les conditions de victoire et de défaite**
- 3) les commandes**
- 4) Les pièces créées, les éléments utilisés et sorties possibles**

### **2.guide développeur**

- 1) les diagramme de classes**
- 2) Importation de modules**
- 3) Vidéo de présentation**

### **3.perspectives de développement**

- 1) Améliorations possibles**
- 2) Réflexions durant le développement**

## 1. guide utilisateur

### 1) Comment jouer au jeu : description détaillée du projet

*Shopping Adventure* est un jeu textuel interactif développé en Python, qui offre aux joueurs une expérience immersive de shopping. Le but du jeu est de simuler une journée de shopping où le joueur doit gérer son budget, faire des choix stratégiques et explorer différentes sections d'un magasin tout en respectant un délai imparti.

Le joueur commence le jeu en créant son personnage, de sa couleur de cheveux à son style vestimentaire. Une fois le personnage créé, le joueur se retrouve dans le hall d'un grand magasin, avec une carte cadeau d'une valeur aléatoire comprise entre **50€ et 200€**.

Cependant, la valeur exacte de la carte cadeau reste inconnue au départ.

Le but est d'explorer différentes sections du magasin, qui contiennent divers articles à acheter. Le joueur doit ajouter des articles à son caddie tout en surveillant son budget et en veillant à ne pas dépasser la limite de sa carte cadeau.

De plus, le joueur est soumis à une contrainte de **temps** : il doit effectuer ses achats dans un délai déterminé. A la fin, il doit avoir réussi à rassembler assez d'articles pour proposer une tenue.

### 2) les conditions de victoire et de défaite

La victoire est obtenue lorsque le joueur parvient à remplir son caddie avec des articles qu'il souhaite acheter, tout en ne dépassant pas le montant de sa carte cadeau et en terminant ses achats dans le délai imparti.

La défaite survient lorsque l'une des conditions de jeu n'est pas remplie correctement. Il existe deux principaux scénarios de défaite :

Dépassement de budget : Si le joueur dépasse le montant de sa carte cadeau pendant ses achats, il perd immédiatement la partie. La défaite est alors expliquée par un message indiquant que le joueur a été trop généreux dans ses achats et a dépassé son budget.

Temps écoulé : Si le joueur n'a pas terminé ses achats avant la fin du temps imparti, c'est également un échec. Le jeu se termine avec un message indiquant que le joueur a pris trop de temps pour choisir ses articles et qu'il n'a pas pu finaliser ses achats à temps.

### 3) les commandes

- **help** : affiche une liste des commandes disponibles
- **quit** : permet au joueur de quitter le jeu.
- **go** : permet au joueur d'aller dans une autre section du magasin.
- **back** : retourne dans la pièce précédente que le joueur a visité.
- **look** : voir l'inventaire des articles dans la pièce actuelle
- **take** : Prendre un article de la pièce actuelle
- **drop**: reposer un article pris dans la pièce actuelle
- **buy**: finaliser son shopping
- **play**: commencer ou recommencer le jeu
- **receipt**: donne le reçu du joueur
- **talk** : parler à une personne dans la pièce actuelle

### 4) Les pièces créées, les éléments utilisés et sorties possibles

Le joueur a 12 pièces à sa disposition dans lesquelles il peut se balader librement en fonction des sorties possibles : Il ya :

- the hall\_entry
- the hall\_exist
- the hall\_center
- the coats\_section
- the sweaters\_section
- the tops\_section
- jewelry
- the shoes\_section
- the accessories\_section
- the sunglasses\_section
- the bottoms\_section
- the checkout

Dans chaque pièces, le joueur peut sélectionner un certain nombre d'éléments

Ces items sont spécifiques à une pièce en particulier. Chaque item contient 3 à 4 éléments de différentes couleurs, matières et coupes.

#### Pour la sweaters\_section:

- sweater(15)
- turtleneck (18)
- jumper (20)
- sweatshirt(12)

#### Pour la coats\_section:

- coat (25)
- puffer (30)
- trench (35)
- fur\_coat (40)

#### Pour la bottoms\_section:

- jeans (14)
- pants (10)
- skirt (15)
- shorts (12)

Pour la tops\_section:

- bandeau (8)
- t shirt (10)
- shirt (14)
- crop top (12)

Pour la sunglasses section:

- leopard\_sunglasses (15)
- square\_sunglasses (12)
- oversized\_sunglasses (18)

Pour la accessories\_section:

- scarf1 (10)
- scarf2 (9)
- beanie (6)
- gloves (8)

Pour la shoes section:

- heels (25)
- boots (30)
- slippers (10)
- sneakers(20)

Pour la jewelry:

- ring (18)
- necklace (20)
- earrings (25)
- watch (40)

Les sorties possibles dans chaque pièce sont :

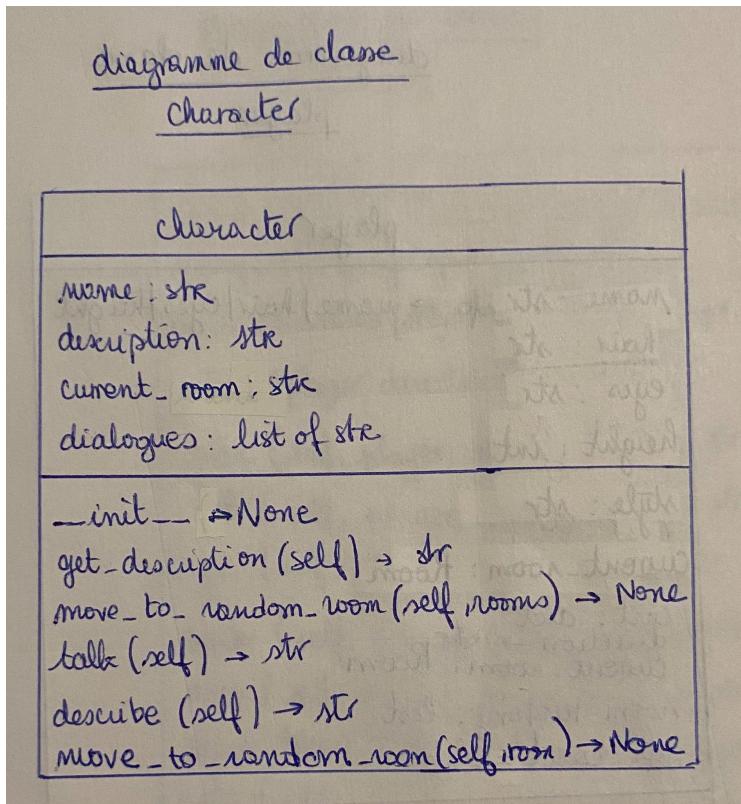
#Create exits for rooms

```
hall_entry.exits={"N":hall_center, "S": None , "W":checkout, "E":coats_section}
hall_center.exits={"N":hall_exit, "S": hall_entry , "W":sunglasses_section, "E":sweaters_section}
hall_exit.exits={"N":accessories_section, "S": hall_center , "W":jewelry, "E":bottoms_section}
coats_section.exits={"N":sweaters_section, "S": None , "W": hall_entry , "E": None}
sweaters_section.exits={"N":bottoms_section, "S": coats_section, "W":hall_center , "E":None}
tops_section.exits={"N":None, "S": bottoms_section , "W":accessories_section , "E": None}
bottoms_section.exits={"N":tops_section, "S": sweaters_section , "W":hall_exit , "E": None}
sunglasses_section.exits={"N":jewelry, "S": checkout , "W":shoes_section , "E": tops_section}
accessories_section.exits={"N": None, "S": hall_exit , "W":shoes_section, "E":tops_section }
jewelry.exits={"N":shoes_section, "S": sunglasses_section , "W":None, "E":hall_exit }
shoes_section.exits={"N":None, "S": jewelry,"W":None, "E": accessories_section}
checkout.exits={"N":sunglasses_section , "S":None , "W":None , "E":hall_entry}
```

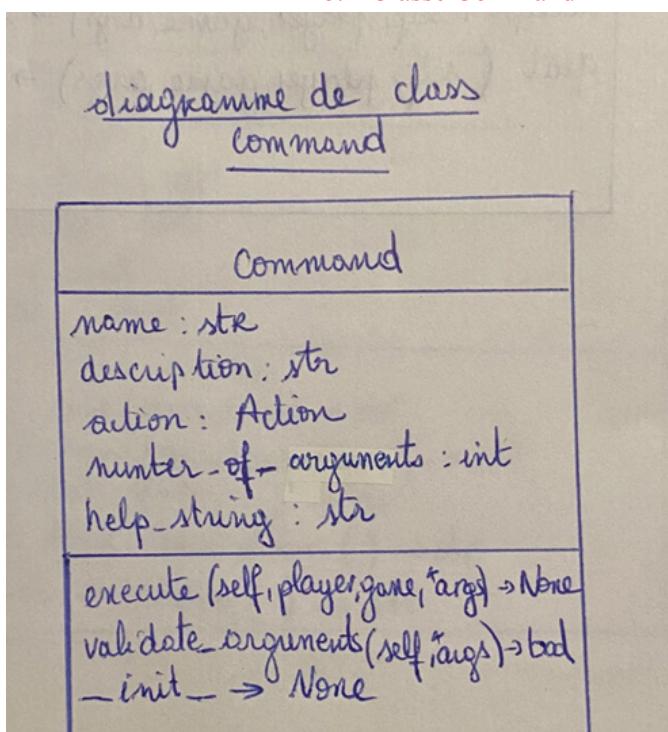
## 2.guide développeur

### 1) les diagramme de classes

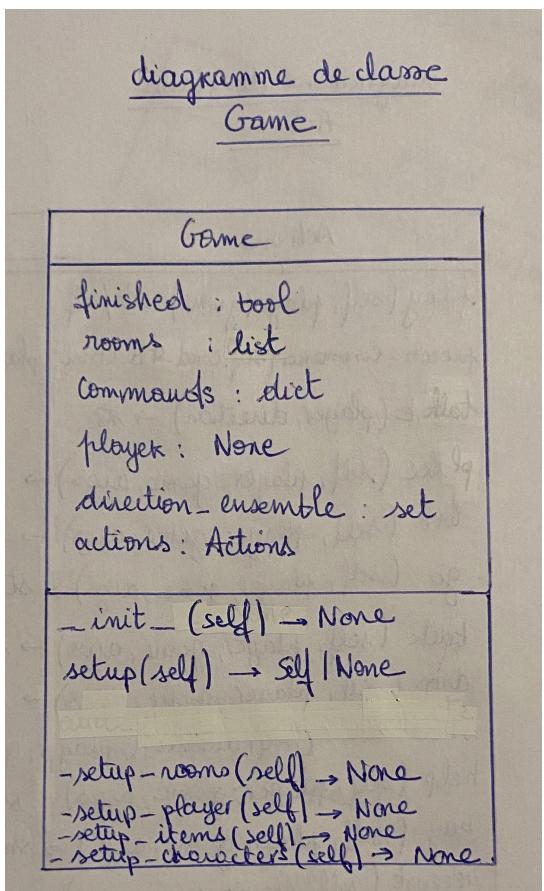
#### a. Class Character



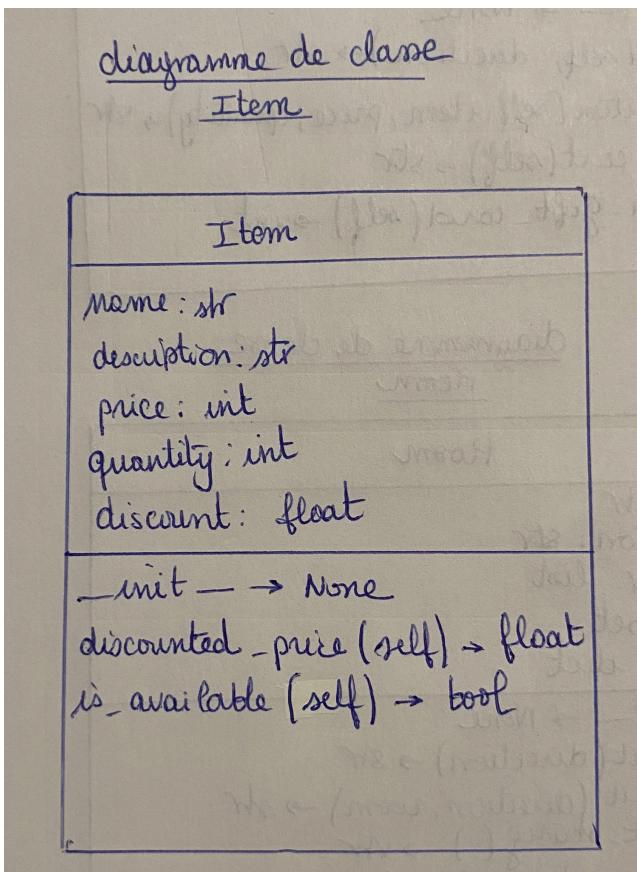
#### b. Classe Command



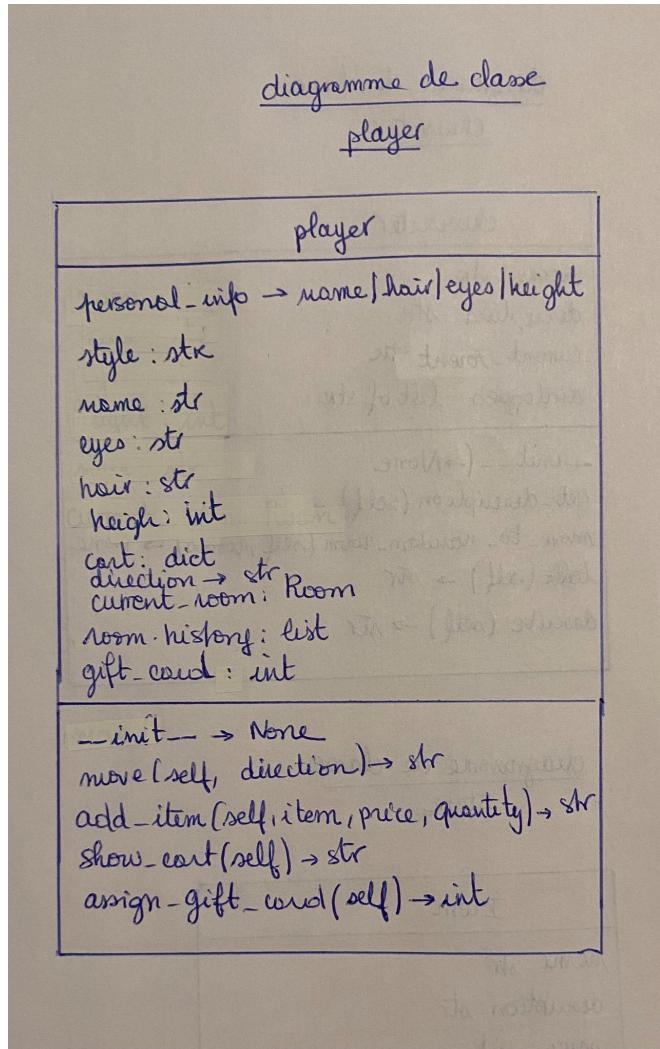
c. Class Game



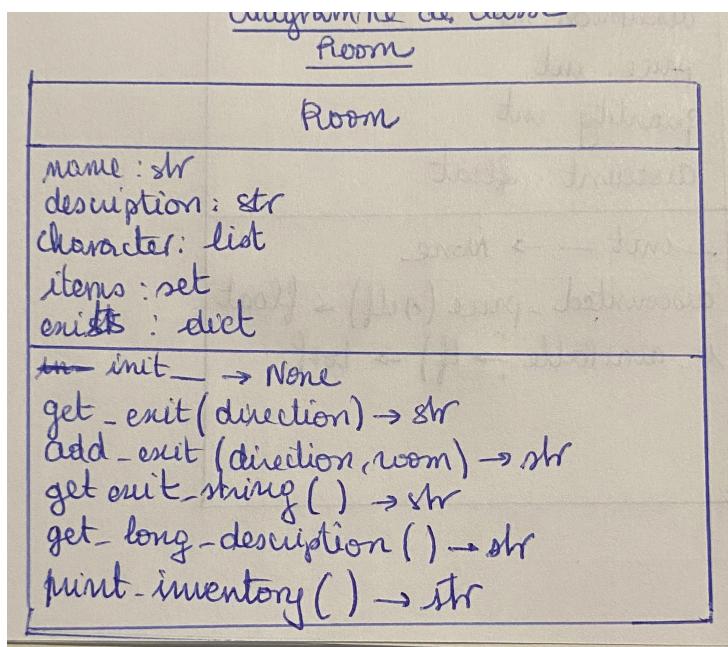
d. Class Item



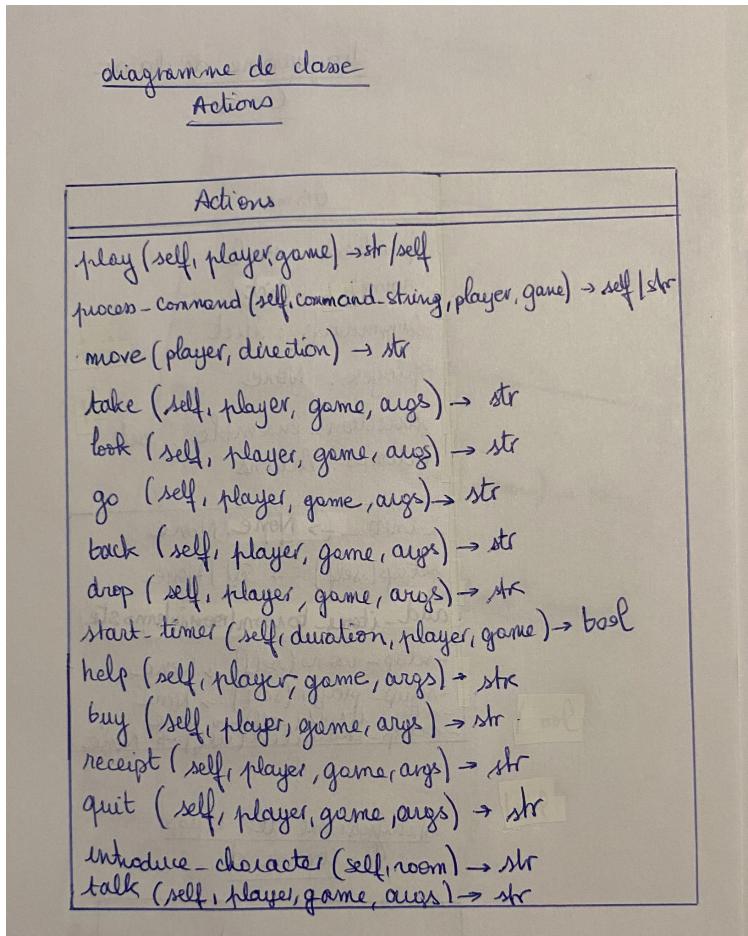
e. Class Player



f. Class Room



### g. Class Actions



### h. Importation des modules

#### • Import random :

Le module random permet de générer des nombres ou de faire des choix aléatoires avec des fonctions comme random.randint(), random.choice() et random.shuffle(), ajoutant du hasard dans un programme.

#### • Import threading :

Le module threading permet de gérer l'exécution simultanée de plusieurs fils d'exécution en Python, améliorant les performances pour des tâches parallèles, comme dans les jeux.

#### • Import time

Le module time permet de gérer le temps, suspendre l'exécution (time.sleep()) et mesurer la durée d'un code.

## 4) Vidéo de présentation

[https://youtu.be/S8oumC37TO0?si=OdF3T5wjyf\\_Vmt9A](https://youtu.be/S8oumC37TO0?si=OdF3T5wjyf_Vmt9A)

### 3.perspectives de développement

#### 1) Améliorations possibles

- Une évolution des prix selon la demande

On pourrait implanter une variation des prix basée sur l'offre et la demande. Par exemple, les prix peuvent augmenter pour certains articles populaires ou baisser pour les articles invendus.

Pour cela, il faudrait mettre à jour le prix d'un article après chaque achat, en fonction de l'offre et de la demande. On pourrait rajouter une méthode pour ajuster les prix de manière dynamique.

- L'ajout de mini-jeux ou de quêtes

On pourrait inclure des mini-jeux ou des quêtes secondaires que le joueur pourrait accomplir pour gagner plus d'argent ou débloquer des articles spéciaux. Pour cela, il faudrait ajouter une quête ou un mini-jeu à chaque fois que le joueur entre dans une nouvelle zone ou magasin. De fait, le joueur pourrait être récompensé avec de l'argent ou des objets rares.

- Marché de l'occasion (échange d'articles)

Un système de marché de l'occasion pourrait être ajouté où les joueurs peuvent échanger des objets ou vendre des articles qu'ils ne veulent plus. Cela peut introduire un élément économique supplémentaire dans le jeu.

#### 2) Réflexions durant le développement

Lors du développement du jeu, nous avons rencontré des défis techniques et conceptuels. En effet, ces défis ont nécessité des réflexions approfondies pour garantir une expérience utilisateur fluide et agréable. Voici un résumé des principales réflexions :

a) La gestion des erreurs de saisie :

Le joueur pouvait saisir des commandes incorrectes ou incomplètes. Pour cela et pour éviter que le jeu plante, on a implémenté un système robuste de gestion des erreurs.

b) La synchronisation du minuteur avec la boucle principale du jeu:

La gestion du minuteur nous a posé un défi particulier. Un thread séparé a été utilisé pour décompter le temps, avec une synchronisation assurée via (game.finished). Cela nous a permis d'arrêter le jeu dès que le temps était écoulé ou que le joueur avait terminé ses achats.

c) Equilibre de la difficulté du jeu :

Le jeu devait captiver l'utilisateur sans le frustrer par des défis trop complexes ni l'ennuyer avec des tâches trop simples. Pour cela, nous avons équilibré trois éléments : la pression du minuteur, la contrainte du budget limité, et la nécessité d'explorer stratégiquement les pièces du magasin, afin de créer une expérience stimulante et agréable.