RAPPORT DU
PROJET TBA
IGI_3008

DATE: 18 Janvier 2025

DEBBAGH Chama - JABRY Rana

SOMMAIRE :

1.Guide utilisateur :

1.1Titre et contexte du jeu :

Titre du jeu : Exploration d'une maison hantée.

Contexte du jeu :

Ce jeu d'aventure textuelle plonge le joueur dans une maison hantée mystérieuse. Il a pour mission d'explorer les différentes pièces de la maison tout en résolvant des énigmes qui empêchent sa progression dans le jeu.

Notre projet **La Maison Hantée** est un jeu d'aventure textuelle conçu pour plonger le joueur dans une expérience immersive et interactive. L'objectif est de stimuler la réflexion et l'exploration à travers une série d'énigmes intégrées dans un environnement fictif mystérieux.

Dans le scénario, le joueur incarne un explorateur qui s'aventure dans une maison réputée hantée. L'ensemble de la maison est structuré autour de pièces interconnectées, chacune verrouillée par une énigme. Pour progresser, le joueur doit résoudre ces énigmes afin de déverrouiller les passages et accumuler des points. L'objectif ultime est d'obtenir **trois points**, synonymes de victoire et de la découverte des secrets de la maison.

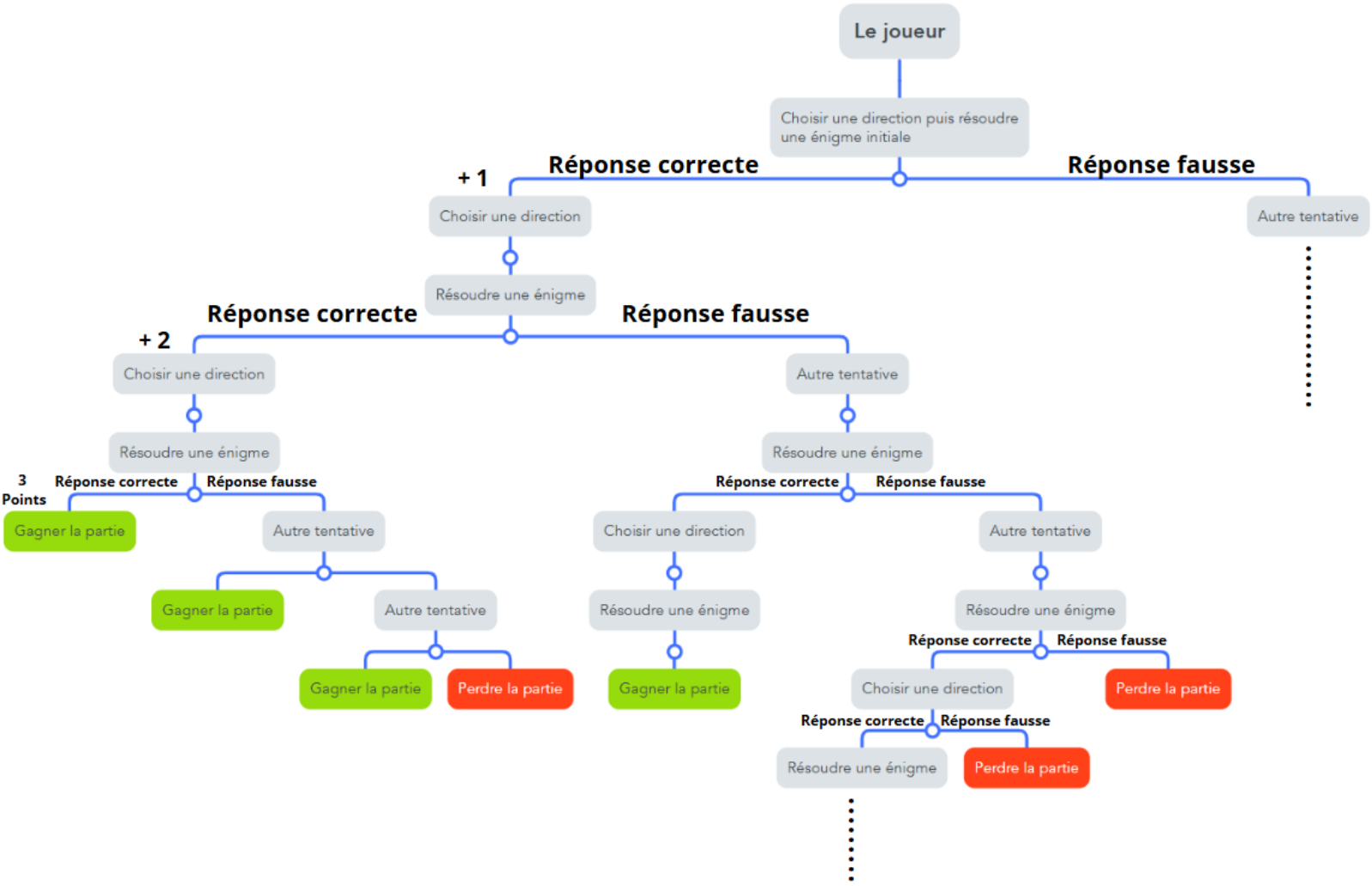
Toutefois, la progression est limitée par une contrainte majeure : le joueur ne dispose que de **trois tentatives** . En cas d'échec, la partie se termine, et le joueur est déclaré perdant, condamné à rester enfermé dans la maison. Cette mécanique de jeu introduit une dimension stratégique où chaque réponse doit être soigneusement réfléchie.

Les principales fonctionnalités du jeu incluent :

- **Exploration** : Le joueur peut se déplacer entre les différentes pièces de la maison.
- **Résolution d'énigmes** : Chaque pièce est associée à une énigme unique à résoudre.
- **Gestion des erreurs** : Un système de comptabilisation des tentatives limite les erreurs possibles.
- **Condition de victoire/défaite** : Le joueur gagne en accumulant trois points ou perd après trois erreurs.

Ce projet met en valeur plusieurs aspects techniques, notamment la gestion d'un système d'interactions entre objets (pièces, énigmes, joueur), la validation des entrées utilisateur, et l'intégration d'un scénario immersif.

1.2. Diagramme du jeu :



2.Guide développeur :

2.1. Class Game :

Dans la classe Game, nous avons d'abord créé les énigmes en définissant quatre questions, chacune accompagnée de sa réponse correcte.

CodesProjetTBA > game.py

```
6   class Game:
10      def __init__(self):
17
18      def setup(self):
19          """Configure le jeu avec les pièces et les énigmes."""
20          # Création des énigmes
21          riddles = [
22              Riddle("Je te suis partout. Je disparais chaque fois que la lumière arrive. Qui suis-je ?", "ombre"),
23              Riddle("Je monte et je descends, mais je ne bouge jamais. Qui suis-je ?", "escaliers"),
24              Riddle("Combien font 12 divisé par 3, multiplié par 2 ?", "8"),
25              Riddle("Plus j'ai de gardiens, moins je suis gardé. Moins j'ai de gardiens, plus je suis gardé. Qui suis-je ?", "secret"),
26              Riddle("Je peux monter jusqu'au ciel mais je ne peux pas voler. Qui suis-je ?", "fumee"),
27              Riddle("Dans une cave sombre, on me trouve souvent pleine. Qui suis-je ?", "bouteille")
28          ]
```

Ensuite, nous avons configuré le jeu en créant les différentes pièces de la maison et en attribuant à chaque pièce une énigme spécifique. Cela permet de structurer le jeu et de lier chaque espace à un défi à résoudre pour progresser.

```
30          # Création des pièces du rez-de-chaussée
31          hall = Room("Hall d'entrée", "un hall sombre et poussiéreux", riddles[0])
32          salon = Room("Salon", "une pièce avec des meubles recouverts de draps blancs", riddles[1])
33          cuisine = Room("Cuisine", "une cuisine abandonnée avec des ustensiles rouillés", riddles[2])
34          bibliotheque = Room("Bibliothèque", "une vaste bibliothèque aux étagères couvertes de toiles d'araignées", riddles[3])
35
36
37          # Création des pièces de l'étage
38          grenier = Room("Grenier", "un grenier poussiéreux rempli d'objets anciens", riddles[4])
39          chambre = Room("Chambre", "une chambre avec un vieux lit à baldaquin", None)
40
41
42          # Création des pièces du sous-sol
43          cave = Room("Cave", "une cave humide avec des toiles d'araignées", riddles[5])
44          laboratoire = Room("Laboratoire", "un ancien laboratoire secret", None)
```

On a aussi configuré les sorties possibles (N, S, E, O, U, D) qui sont (Nord, Sud, Est, Ouest, Up, Down) avec les différentes pièces qu'on a créé.

```python
# Configuration des sorties du rez-de-chaussée
hall.exits = {
    "N": salon,
    "E": cuisine,
    "O": bibliotheque,
    "U": None,  # On ne peut pas monter depuis le hall
    "D": cave   # On peut descendre à la cave
}

salon.exits = {
    "S": hall,
    "E": bibliotheque,
    "U": chambre,  # On peut monter à la chambre
    "D": None
}

cuisine.exits = {
    "O": hall,
    "N": bibliotheque,
    "U": None,
    "D": laboratoire  # On peut descendre au laboratoire
}

bibliotheque.exits = {
    "S": cuisine,
    "E": salon,
    "U": grenier,  # On peut monter au grenier
    "D": None
}
```

```python
# Configuration des sorties de l'étage
grenier.exits = {
    "D": bibliotheque,  # On peut descendre à la bibliothèque
    "N": None,
    "S": None,
    "E": None,
    "O": None,
    "U": None
}

chambre.exits = {
    "D": salon,  # On peut descendre au salon
    "N": None,
    "S": None,
    "E": None,
    "O": None,
    "U": None
}

# Configuration des sorties du sous-sol
cave.exits = {
    "U": hall,  # On peut monter au hall
    "N": None,
    "S": None,
    "E": laboratoire,
    "O": None,
    "D": None
}

laboratoire.exits = {
    "U": cuisine,  # On peut monter à la cuisine
    "N": None,
    "S": None,
    "E": None,
    "O": cave,
    "D": None
}
```

Nous avons ajouté de nouveaux attributs qui jouent un rôle central dans la gestion de la progression et des tentatives du joueur :

1. **point** : Cet attribut représente le score du joueur. Chaque énigme correctement résolue permet d'attribuer un point. Atteindre **3 points** est la condition de victoire dans le jeu.

2. **wrong_attempts** : Le nombre de tentatives incorrectes que le joueur a effectuées lors de la résolution d'énigmes. À chaque mauvaise réponse donnée par le joueur, cette valeur est incrémentée. Si wrong_attempts atteint la valeur de max_wrong_attempts cela déclenche la f in de la partie, et le joueur perd, ayant épuisé toutes ses tentatives.

3. **max_wrong_attempts** : La valeur par défaut est de 3, ce qui signifie que le joueur a droit à trois mauvaises réponses avant de perdre la partie

```python
def __init__(self, name):
    self.name = name
    self.current_room = None
    self.points = 0
    self.wrong_attempts = 0
    self.max_wrong_attempts = 3
    self.history = []  # Liste des pièces visitées
    self._inventory = {}  # Inventaire du joueur
```

2.2. Class Player :

La classe Player modélise le joueur et ses interactions avec le jeu. Cette classe gère les informations clés du joueur, comme son nom, sa position actuelle dans la maison (current_room), et son score.

```
2   class Player:
6       def __init__(self, name):
7           self.name = name
8           self.current_room = None
9           self.points = 0
10          self.wrong_attempts = 0
11          self.max_wrong_attempts = 3
12          self.history = []  # Liste des pièces visitées
13          self._inventory = {}  # Inventaire du joueur
```

Elle permet au joueur de se déplacer entre les pièces en résolvant des énigmes. Chaque énigme correcte ajoute un point, tandis que les erreurs sont comptabilisées. Après trois erreurs, le joueur perd la partie. Il détermine alors les conditions de victoire (trois points) et de défaite (trois erreurs), jouant ainsi un rôle central dans la logique et la progression du jeu.

```
2   class Player:
15      def move(self, direction):
24          if self.current_room:
26
27          if not next_room.solved and next_room.riddle:
28              print(f"\nÉnigme pour entrer dans {next_room.name}:")
29              print(next_room.riddle.question)
30              answer = input("Votre réponse: ")
31
32              if next_room.riddle.check_answer(answer):
33                  print("\nBonne réponse! Vous gagnez un point.")
34                  next_room.solved = True
35                  self.points += 1
36                  self.current_room = next_room
37                  print(self.current_room.get_long_description())
38                  self.print_history()  # Affiche l'historique après chaque déplacement
39
40                  if self.points >= 3:
41                      print("\nFélicitations! Vous avez gagné le jeu en résolvant 3 énigmes!")
42                      return "win"
43              else:
44                  self.wrong_attempts += 1
45                  remaining_attempts = self.max_wrong_attempts - self.wrong_attempts
46                  print(f"\nMauvaise réponse! Il vous reste {remaining_attempts} tentatives.")
47
48                  if self.wrong_attempts >= self.max_wrong_attempts:
49                      print("\nGame Over! Vous avez épuisé toutes vos tentatives.")
50                      return "lose"
51                  return False
```

Le fichier inclut également un suivi de l'historique des pièces visitées et la gestion d'un inventaire pour collecter des objets.

```
def print_history(self):
    """Affiche l'historique des pièces visitées."""
    if self.history:
        print("\nPièces visitées:")
        for room in self.history:
            print(f"    - {room}")
    else:
        print("\nVous n'avez pas encore visité de pièces.")
```

2.3. Class Riddle :

La classe Riddle représente une énigme dans le jeu, comprenant une question et une réponse correcte.

CodesProjetTBA > riddle.py

```python
class Riddle:
    """
    Classe représentant une énigme avec sa question et sa réponse.
    """
    def __init__(self, question, answer):
        self.question = question
        self.answer = answer.lower()  # Convertit la réponse en minuscules

    def check_answer(self, player_answer):
        """Vérifie si la réponse du joueur est correcte."""
        return player_answer.lower() == self.answer
```

Lors de l'initialisation de la classe, deux attributs sont définis : question, qui contient la question posée au joueur, et **answer**, qui est la réponse correcte à l'énigme.

La réponse est convertie en minuscules pour garantir que la comparaison avec la réponse donnée par le joueur soit insensible à la casse.

La méthode **check_answer** permet ensuite de comparer la réponse donnée par le joueur à la réponse correcte. Si elles sont identiques, elle retourne **True**, sinon **False**. Cette classe permet ainsi de créer et de vérifier les énigmes dans le jeu.

2.4. Diagramme de Classes :

2.4.1.Class Game :

Attributs :

- **finished** : Boolean **:** Indique si le jeu est terminé.
- **rooms** : List[Room] **:** Liste des pièces du jeu.
- **player** : Player **:** Instance du joueur.
- **game_state** : String **:** État actuel du jeu.
- **commands** : Dict[String, Command] **:** Dictionnaire des commandes disponibles.
- **directions** : Set[String] **:** Ensemble des directions valides.

Méthodes :

- **__init__**(self) → None **:** Initialise les attributs de la classe.
- **setup**(self) → None **:** Configure les pièces, les commandes, et le joueur.
- **do_go**(self, args) → Boolean **:** Gère les déplacements du joueur.
- **do_quit**(self, args) → String **:** Quitte le jeu.
- **do_help**(self, args) → Boolean **:** Affiche les commandes disponibles.
- **do_look**(self, args) → Boolean **:** Affiche la description de la pièce actuelle.
- **play(self)** → None **:** Lance la boucle principale du jeu.

2.4.2.Class Command

Attributs :

- **command_word** : String **:** Mot-clé de la commande.
- **help_string :** String **:** Description de la commande.
- **action :** function **:** Fonction associée à la commande.
- **number_of_parameters** : Int **:** Nombre de paramètres requis.

Méthodes :

- **__init__**(self, command_word, help_string, action, number_of_parameters) → None **:** Initialise une commande.
- **__str__**(self) → String **:** Retourne une description textuelle de la commande.

2.4.3.Class Riddle:

Attributs :

- **question** : String **:** Question de l'énigme.
- **answer** : String **:** Réponse correcte.

Méthodes :

- **__init__**(self, question, answer) → None **:** Initialise une énigme.
- **check_answer**(self, player_answer) → Boolean **:** Vérifie si la réponse du joueur est correcte.

2.4.4.Class Actions

Méthodes :

- **go**(game, list_of_words, number_of_parameters) → Boolean **:** Gère le déplacement du joueur.
- **quit**(game, list_of_words, number_of_parameters) → Boolean **:** Permet de quitter le jeu.
- **help**(game, list_of_words, number_of_parameters) → Boolean **:** Affiche l'aide des commandes.
- **back**(game, list_of_words, number_of_parameters) → None **:** Retourne à la pièce précédente.
- **look**(game, list_of_words, number_of_parameters) → Boolean **:** Montre les détails de la pièce actuelle.
- **take**(game, list_of_words, number_of_parameters) → Boolean **:** Permet de prendre un objet.
- **drop**(game, list_of_words, number_of_parameters) → Boolean **:** Permet de déposer un objet.
- **display_items_in_inventory**(player) → None **:** Affiche l'inventaire du joueur.

2.4.5.Class Player

Attributs :

- **name** : String **:** Nom du joueur.
- **current_room** : Room **:** Pièce actuelle du joueur.
- **points** : int **:** Nombre de points accumulés.
- **wrong_attempts** : int **:** Nombre d'erreurs faites.
- **max_wrong_attempts** : int **:** Nombre maximal d’erreurs autorisées.
- **history** : List[String] **:** Historique des pièces visitées.
- **_inventory** : Dict[String, Item] **:** Inventaire du joueur.

Méthodes :

- **__init__**(self, name) → None **:** Initialise le joueur avec son nom.
- **move**(self, direction) → Boolean **:** Déplace le joueur dans une direction.
- **print_history**(self) → None **:** Affiche l'historique des pièces visitées.
- **get_inventory**(self) → String **:** Retourne le contenu de l'inventaire.

2.4.6.Class Room

Attributs :

- **name** : String **:** Nom de la pièce.
- **description** : String **:** Description de la pièce.
- **exits** : Dict[String, Room] **:** Directions possibles vers d'autres pièces.
- **riddle** : Riddle **:** Énigme associée à la pièce.
- **solved** : Boolean **:** Indique si l'énigme a été résolue.
- **inventory** : List[Item] **:** Liste des objets disponibles dans la pièce.

Méthodes :

- **__init__**(self, name, description, riddle=None) → None **:** Initialise une pièce avec son nom, sa description, et une énigme.
- **get_exit**(self, direction) → Room **:** Retourne la pièce accessible dans une direction donnée.
- **get_exit_string**(self) → String **:** Retourne une chaîne décrivant les sorties disponibles.
- **get_long_description**(self) → String **:** Retourne une description complète de la pièce.

3.Perspective/ axes de développement :

Dans notre jeu, nous avons choisi d'utiliser uniquement les commandes **GO, QUIT, HELP et LOOK.**

La commande **LOOK** permet au joueur d'obtenir une description détaillée de la pièce, **GO** lui permet de se déplacer dans une direction donnée, **QUIT** pour quitter le jeu et **HELP** pour afficher la liste des commandes possibles.

Nous n'avons pas utilisé les commandes **BACK, DROP, TAKE et CHECK**, qui étaient présentes dans la première version inspirée du cours, car elles étaient liées à la gestion des objets.

Dans notre jeu, le joueur doit répondre aux énigmes pour se déplacer d'une pièce à l'autre, et non pas interagir avec des objets. Toutefois, nous avons réfléchi à une amélioration pour le jeu. Nous envisageons de lier les énigmes à des indices physiques présents dans les pièces. Par exemple, chaque pièce contiendra des objets que le joueur devrait collecter pour obtenir des indices cruciaux pour résoudre l'énigme et ainsi pouvoir se déplacer dans la pièce. Pour cela, nous ajouterons les commandes **TAKE** pour ramasser les objets, **DROP** pour les reposer, et **CHECK** pour consulter l'inventaire. Pour implémenter cette fonctionnalité, nous pourrions ajouter une nouvelle classe **ITEM** en complément de la classe **RIDDLE**, qui gère actuellement les énigmes. Cette évolution permettrait d'enrichir l'expérience du joueur en ajoutant une dimension supplémentaire à la résolution des énigmes