Rapport Projet TBA :

Achraf ICHKARRAN / Amine EL MOUTTAKI

IGI-3008

Bienvenue dans Backrooms, un jeu d'exploration et de survie où le joueur navigue de salle en salle dans un environnement mystérieux et captivant. Chaque salle possède ses propres caractéristiques, quêtes et personnages non-joueurs (PNJ) avec lesquels le joueur peut interagir. Votre mission est de progresser dans cet univers tout en évitant les dangers et en relevant les défis.

1. Guide Utilisateur - Règles et Commandes

Règles du jeu et conditions de victoire/perte :

- Condition de victoire :

- Atteindre la salle finale.

- Condition de défaite :

- Perdre tous ses points de vie.

Commandes disponibles :

- **go**: Utilisez la commande 'go <direction>' pour passer d'une salle à une autre (N, E, S, O).

- **prendre / poser**: Utilisez 'prendre <objet>' ou 'poser <objet>'.

- **Parler** : Tapez 'parler <nom_du_pnj>'.

- **inspecter** : Tapez 'inspecter'.

- **Historique** : Tapez 'historique'.

- **Quit**.: Tapez 'quit'.

- **attaquer**: taper attaquer <PNJ>

- **help** : tapez help

2. Guide Développeur - Modules

Les classes :

1- Actions :

La classe Actions regroupe l'ensemble des fonctions permettant au joueur d'interagir avec le jeu, que ce soit en se déplaçant, en manipulant des objets, ou en interagissant avec les PNJ et l'environnement.

Les méthodes de cette classe :

- go(game, list_of_words, number_of_parameters) : Déplace le joueur dans une direction donnée, en gérant les obstacles comme les portes verrouillées.
- quit(game, list_of_words, number_of_parameters) : Met fin à la partie et affiche un message d'au revoir.
- help(game, list_of_words, number_of_parameters) : Affiche la liste des commandes disponibles.
- historik(game, list_of_words, number_of_parameters) : Affiche l'historique des pièces visitées par le joueur.
- back(game, list_of_words, number_of_parameters) : Ramène le joueur à la salle précédemment visitée.
- inventorix(game, list_of_words, number_of_parameters) : Affiche l'inventaire du joueur.
- look(game, list_of_words, number_of_parameters) : Montre les objets et PNJ présents dans la pièce actuelle.
- attack(game, list_of_words, number_of_parameters) : Permet au joueur d'attaquer un PNJ, avec une probabilité de réussite aléatoire.
- take(game, list_of_words, number_of_parameters) : Ramasse un objet dans la pièce et l'ajoute à l'inventaire du joueur.
- drop(game, list_of_words, number_of_parameters) : Dépose un objet de l'inventaire du joueur dans la pièce actuelle.

 talk(game, list_of_words, number_of_parameters) : Engage une interaction avec un PNJ, affichant un message spécifique.

2- La classe Game :

La classe Game gère la logique principale du jeu d'aventure textuel, incluant la configuration des éléments (salles, commandes, items, PNJ), le traitement des commandes du joueur et le déroulement de la partie

Attributs :

- finished : Indique si le jeu est terminé (True ou False).
- rooms : Liste des objets **Room**, représentant les différentes salles du jeu.
- commands : Dictionnaire associant des mots-clés de commande à des objets **Command**.
- player : Instance de la classe **Player**, représentant le joueur.
- victory_room : Salle de victoire, définissant l'objectif final.
- DIRECTIONS : Dictionnaire de standardisation des directions ("N", "North", "Sud", etc.).

Méthodes :

- **__init__()** : Initialise les attributs principaux du jeu.
- **centralise_direction(input_direction)** : Standardise une direction donnée en une forme cohérente ("north" devient "N", par exemple).
- **setup()** : Configure les salles, commandes, objets, PNJ, portes, et initialise le joueur.
- **play()** : Lance la boucle principale du jeu où le joueur peut interagir jusqu'à la fin de la partie.
- **process_command(command_string)** : Analyse et exécute une commande saisie par le joueur.
- **print_welcome()** : Affiche un message d'introduction et la description de la pièce où commence le joueur.
- **main()** : Point d'entrée du jeu, démarrant la partie avec la méthode play.

3- La classe Player :

La classe **Player** modélise le joueur dans le jeu, en gérant ses déplacements, son historique, son inventaire, ses points de vie, et ses interactions avec les PNJ et l'environnement.

Attributs :

- name : Nom du joueur.
- current_room : Salle actuelle où se trouve le joueur (instance de la classe **Room**).
- history : Liste des noms des salles visitées par le joueur.
- inventory : Dictionnaire des objets possédés par le joueur.
- pv : Points de vie du joueur, initialisés à 5.

Méthodes :

- **__init__(self, name, depart=None)** : Initialise le joueur avec un nom, une salle de départ, un historique vide, un inventaire vide, et 5 points de vie.
- **move(self, direction)** : Déplace le joueur vers une direction donnée, si la sortie correspondante existe.
- **get_history(self)** : Retourne l'historique des salles visitées par le joueur sous forme de chaîne de caractères.
- **get_inventory(self)** : Retourne une liste des objets présents dans l'inventaire, ou indique qu'il est vide.
- **take_damage(self, amount)** : Réduit les points de vie du joueur et déclenche la mort si les PV tombent à 0.
- **heal(self, amount)** : Augmente les points de vie du joueur, jusqu'à un maximum de 5.
- **die(self, game)** : Gère la mort du joueur et termine la partie.
- **attack(self, character)** : Permet au joueur d'attaquer un PNJ, avec une probabilité de réussite aléatoire, et inflige des dégâts si l'attaque réussit.

4- La classe Room :

La classe Room modélise une salle du jeu, en décrivant ses caractéristiques (nom, description, inventaire, PNJ, sorties disponibles) et en permettant d'interagir avec son contenu.

Attributs :
- name : Nom de la salle.
- description : Description textuelle de la salle.
- inventory_rooms : Ensemble des objets présents dans la salle (instances de la classe **Item**).
- exits : Dictionnaire des sorties disponibles, associant une direction (clé) à une destination (valeur).
- character_rooms : Dictionnaire des personnages présents dans la salle, associant un nom (clé) à une instance de la classe **Character**.

Méthodes :
- **__init__(self, name, description)** : Initialise une salle avec son nom, sa description, un inventaire vide, des sorties et des personnages vides.
- **get_exit(self, direction)** : Retourne la salle associée à une direction donnée, ou None si aucune sortie n'existe dans cette direction.
- **get_exit_string(self)** : Renvoie une chaîne listant toutes les directions accessibles depuis la salle.
- **get_long_description(self)** : Renvoie une description détaillée de la salle, incluant les sorties disponibles.
- **get_inventory(self)** : Fournit une description des objets et personnages présents dans la salle ou indique que la salle est vide.

5- La classe command :

La classe Command représente une commande dans le jeu, incluant son mot-clé, une description d'aide, l'action associée, et le nombre de paramètres attendus.

Attributs :

- command_word : Le mot-clé de la commande (e.g., "go", "help").
- help_string : Une description d'aide expliquant la fonction de la commande.
- action : La fonction ou méthode à exécuter lorsque la commande est appelée.
- number_of_parameters : Le nombre de paramètres requis pour exécuter la commande.

Méthodes :

- __init__(self, command_word, help_string, action, number_of_parameters) : Initialise une commande avec son mot-clé, sa description, l'action associée, et le nombre de paramètres attendus.
- __str__(self) : Retourne une représentation textuelle de la commande, combinant le mot-clé et la chaîne d'aide.

6- La classe item :

La classe **Item** représente un objet manipulable dans le jeu, avec des attributs tels que son nom, sa description et son poids.

Attributs :

- name : Nom de l'objet.
- description : Description textuelle de l'objet.
- weight : Poids de l'objet (en kg ou une autre unité de mesure).

Méthodes :

- **__init__(self, name, description, weight)** : Initialise un nouvel objet avec son nom, sa description et son poids.
- **__str__(self)** : Retourne une chaîne lisible décrivant l'objet, par exemple : "Épée: Une épée tranchante (Weight: 3 kg)".
- **to_dict(self)** : Convertit l'objet en un dictionnaire contenant son nom comme clé, et une liste [description, poids] comme valeur.

7- La classe Character :

Attributs :

- name : Nom du personnage.
- description : Description textuelle du PNJ.
- current_room : Salle actuelle où se trouve le PNJ (instance de la classe **Room**).
- msgs : Liste de messages que le PNJ peut partager.

- hp : Points de vie du PNJ.
- can_move : Booléen indiquant si le PNJ peut se déplacer ou non.

Méthodes :

- **__init__(self, name, description, current_room, msgs, hp, can_move)** : Initialise un PNJ avec ses caractéristiques, sa salle actuelle, et ses points de vie.
- **__str__(self)** : Retourne une description textuelle du PNJ, incluant son nom et ses messages.
- **to_dict(self)** : Convertit les attributs du PNJ en dictionnaire.
- **get_msg(self)** : Récupère le premier message du PNJ, le retire de sa liste, et le retourne.
- **move(self)** : Déplace le PNJ aléatoirement dans une pièce adjacente, si les conditions le permettent.
- **take_damage(self, amount)** : Réduit les points de vie du PNJ de la quantité spécifiée et déclenche sa mort si ses PV tombent à 0.
- **die(self)** : Gère la mort du PNJ en le retirant de la pièce et en affichant un message.
- **attack(self, player)** : Le PNJ attaque le joueur, infligeant un montant de dégâts aléatoire.

8- La classe door :

La classe **Door** représente une porte reliant deux pièces, pouvant être verrouillée et nécessitant un code pour être déverrouillée.

Attributs :

- name : Nom ou identifiant de la porte.
- code : Code nécessaire pour déverrouiller la porte.
- locked : Booléen indiquant si la porte est verrouillée (True) ou déverrouillée (False).

Méthodes :

- **__init__(self, name, code)** : Initialise une porte avec un nom, un code et la verrouille par défaut.
- **unlock(self, input_code)** : Tente de déverrouiller la porte en comparant le code fourni au code de la porte. Retourne True si le code est correct.
- **is_locked(self)** : Retourne True si la porte est verrouillée, sinon False.

3. Perspectives de Développement

Perspectives de développement:

1. Ajouter davantage de salles avec des thématiques uniques.

2. Enrichir les interactions avec les PNJ (arcs narratifs, dialogues).

3. Introduire des quêtes secondaires.

Optimisations:

- Améliorer les performances pour gérer un plus grand nombre de salles.

- Ajouter des sauvegardes et chargements de parties.

Extensions possibles:

- Multijoueur : Collaboration pour explorer les Backrooms.

- Graphismes améliorés avec Pygame ou une autre bibliothèque.

- Version mobile ou web.

Conclusion :

Mon jeu Python est un projet d'aventure textuel conçu de manière modulaire pour être évolutif et immersif. Chaque élément, comme les pièces, les PNJ, les portes et les objets, est représenté par des classes distinctes, ce qui facilite l'ajout de nouvelles fonctionnalités. Le joueur peut explorer les pièces, interagir avec les personnages, résoudre des énigmes et affronter des adversaires. Les portes verrouillées et les indices cachés ajoutent une dimension stratégique, tandis que l'inventaire et l'historique suivent sa progression. Ce projet constitue une base solide pour créer un univers captivant et personnalisable.