Projet PYTHON TBA – RAPPORT

Coddeville Daryl
Samba Diallo
E3FD

Guide utilisateur

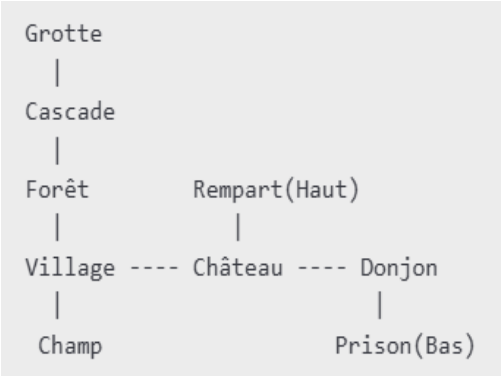
Ce jeu python prend place dans un univers médiéval. Le but du jeu est d'aller vaincre le dragon qui se trouve dans la grotte.

Pour accéder à la grotte, il faut au préalable avoir récupérer la carte pour savoir où se trouve l'antre du dragon.

Scénario de victoire :
Le joueur entre dans la grotte, avec l'épée en main, et tue le dragon.
Le message de victoire s'affiche, le jeu s'arrête.

Scénario de défaite :
Le joueur entre dans la grotte, sans l'épée. Le dragon le mange tout crû.
Le message de défaite s'affiche, le jeu s'arrête.

Carte du jeu :



Liste des commandes :

- **help** : affiche de l'aide.
- **quit** : quitte le jeu.
- **go** : permet de se déplacer dans une direction.
- **back** : permet de revenir en arrière.
- **check** : permet de vérifier les objets dans son inventaire.
- **look** : permet de regarder les objets dans la carte actuelle.
- **take** : permet de prendre un objet.
- **drop** : permet de reposer un objet.
- **talk** : permet de parler à un pnj.

Liste des personnages :

Immobile

- **Le roi** : *Le souverain du royaume, portant une couronne d'or et un manteau royal.*
  Il reste au château, et nous demande d'aller tuer le dragon.
  Réplique : "Salut aventurier ! Un terrible dragon menace notre royaume... Si tu es assez courageux, arme toi d'une épée et récupère la carte dans mon donjon, tu sauras où ce trouve l'antre de ce monstre ! Je te couvrirai d'or à ton retour ! Si tu reviens bien sûr..."

- **Le forgeron** : *Un artisan complètement chauve et musclé.*
  Il reste au village, et il fabrique l'épée pour tuer le dragon.
  Réplique : "Je peux forger une épée pour vous."

- **Le prisonnier** : *Un homme enchaîné, avec une barbe longue et des yeux fatigués*
  Il reste dans la prison, il n'apporte rien à la narrative du jeu.
  Répliques : "Les murs ici semblent m'étouffer.", "Si seulement quelqu'un pouvait m'aider à m'échapper..."

Mobile (Personnages qui ont 50% de chance de se déplacer lorsqu'on traverse le lieu où ils se trouvent. Ils n'apportent rien à la narrative du jeu mais donne du dynamisme.)

- **Le bûcheron** : *Un robuste homme avec une hache, prêt à couper du bois dans la forêt.*
  Apparaît dans fôret.
  Répliques : "Je suis Pierre, le bûcheron. Si vous avez besoin de bois, vous savez où me trouver.", "Ce bois est parfait pour l'hiver, vous ne trouvez pas ?", "La forêt est grande, mais je connais chaque arbre ici."

- **Le villageois** : *Un homme vêtu simplement, qui semble toujours occupé par les tâches du village.*
  Apparaît dans village. Réplique : "Êtes-vous l'aventurier venu nous sauver ?"

Liste des objets :

- **epee** = Item("epee", "*une épée au fil tranchant comme un rasoir*", 5)
  Condition de victoire pour tuer le dragon.

- **clef** = Item("clef", "*une clé dorée ouvrant une porte secrète*", 1)
  Élément de décor

- **pain** = Item("pain", "*un morceau de pain rassis*", 1)
  Élément de décor

- **carte** = Item("carte", "*une carte ancienne montrant des chemins oubliés*", 1)
  Permet d'accéder à la carte grotte.

- **buche** = Item("buche", "*un énorme bout de bois, certainement trop lourd pour être porté seul*", 20)
  Objet impossible à récupérer car il est trop lourd.

Guide développeur

Classe Actions

La classe Actions contient plusieurs méthodes statiques qui sont utilisées pour exécuter des actions spécifiques dans le jeu. Chaque méthode prend trois paramètres principaux :

- game: L'objet principal du jeu.
- list_of_words: Une liste des mots de la commande saisie par l'utilisateur.
- number_of_parameters: Le nombre de paramètres attendus par la commande.

Les méthodes suivantes sont définies dans cette classe :

1. go(game, list_of_words, number_of_parameters)
   - Déplace le joueur dans la direction spécifiée (N, S, E, O, U, D).
   - Vérifie si le nombre de paramètres est correct.
   - Si la direction est valide, le joueur est déplacé dans la salle correspondante.
   - Si la direction est inconnue, affiche un message d'erreur.
2. quit(game, list_of_words, number_of_parameters)
   - Permet au joueur de quitter le jeu.

3. help(game, list_of_words, number_of_parameters)
    - Affiche la liste des commandes disponibles.
    - Vérifie si le nombre de paramètres est correct et, si c'est le cas, imprime toutes les commandes disponibles du jeu.
4. back(game, list_of_words, number_of_parameters)
    - Permet au joueur de revenir à la dernière salle visitée.
    - Si l'historique des salles est vide ou contient seulement la salle actuelle, affiche un message d'erreur.
    - Sinon, affiche la salle précédente et met à jour la position du joueur.
5. check(game, list_of_words, number_of_parameters)
    - Affiche la liste des items présents dans l'inventaire du joueur.
6. look(game, list_of_words, number_of_parameters)
    - Affiche la description de la pièce actuelle, ainsi que les items présents dans cette pièce.
7. take(game, list_of_words, number_of_parameters)
    - Permet au joueur de prendre un item dans la pièce actuelle.
    - Vérifie si l'item est présent dans la pièce et si le joueur peut le prendre (en fonction du poids).
    - Si l'item est pris avec succès, il est ajouté à l'inventaire du joueur et retiré de l'inventaire de la pièce.
8. drop(game, list_of_words, number_of_parameters)
    - Permet au joueur de poser un item dans la pièce actuelle.
    - Vérifie si l'item est présent dans l'inventaire du joueur, puis le retire et l'ajoute à l'inventaire de la pièce.
9. talk(game, list_of_words, number_of_parameters)
    - Permet au joueur de parler à un PNJ (personnage non joueur) dans la pièce actuelle.
    - Si le nom du PNJ est précisé et que ce dernier se trouve dans la pièce, le message du PNJ est affiché.
    - Si aucun PNJ n'est trouvé avec ce nom, un message d'erreur est affiché.

Classe Character

Représente un personnage non-joueur (PNJ) dans le jeu. Elle contient des attributs qui définissent le PNJ, ainsi que plusieurs méthodes permettant de gérer ses interactions et ses déplacements.

Attributs :

1. name (str) : Le nom du personnage.
2. description (str) : La description du personnage.
3. current_room (Room) : La pièce où se trouve le personnage.
4. msgs (list) : Liste des messages associés au personnage.
5. _msg_index (int) : Un index utilisé pour garder une trace du message actuellement affiché.

Méthodes :

1. __init__(self, name, description, current_room, msgs) :
    - Initialise un personnage non-joueur avec un nom, une description, la pièce actuelle et une liste de messages.
2. __str__(self) :
    - Retourne une représentation textuelle du personnage sous la forme : nom : description.

3. move(self) :
    - Permet au personnage de se déplacer dans une pièce voisine, à moins que le personnage soit immobile (comme un roi, un forgeron ou un prisonnier).
    - Si le personnage est mobile, il choisit une salle voisine aléatoirement parmi celles disponibles dans sa pièce actuelle et se déplace dans cette salle.
    - Le déplacement est indiqué par un message dans la console (ou en mode debug si activé).
4. get_msg(self) :
    - Retourne un message cyclique associé au personnage.
    - Le message est choisi de manière cyclique à partir de la liste msgs. Après chaque utilisation d'un message, l'index du message est mis à jour pour que le message suivant soit retourné à l'appel suivant.
    - Si la liste msgs est vide, un message indiquant que le personnage n'a rien à dire est retourné.

Classe Command

La classe Command représente une commande dans le jeu. Chaque commande se compose d'un mot-clé, d'une description d'aide, d'une action associée et d'un nombre de paramètres attendus. Elle permet de structurer et de gérer les commandes que le joueur peut entrer dans le jeu.

Attributs :

1. command_word (str) : Le mot-clé de la commande, qui est utilisé pour identifier et exécuter la commande (par exemple, "go", "look", "take").
2. help_string (str) : La description de la commande, qui explique son fonctionnement et comment l'utiliser.
3. action (function) : La fonction à exécuter lorsque la commande est appelée. Cette fonction est liée à l'action spécifique qui sera réalisée dans le jeu lorsque la commande est donnée.
4. number_of_parameters (int) : Le nombre de paramètres attendus par la commande. Cela permet de vérifier que la commande est correctement formatée lors de son exécution.

Méthodes :

1. __init__(self, command_word, help_string, action, number_of_parameters) :
    - Le constructeur de la classe, qui initialise les attributs de la commande.
    - command_word : mot-clé de la commande.
    - help_string : description de la commande.
    - action : fonction associée à l'action à exécuter.
    - number_of_parameters : nombre de paramètres attendus.
2. __str__(self) :
    - Retourne une représentation textuelle de la commande sous la forme du command_word suivi de la help_string. Cette méthode permet de fournir une description simple de la commande lors de l'affichage.

Classe game

La classe Game gère le flux global du jeu, y compris l'initialisation des salles, des personnages, des objets, et l'exécution des commandes du joueur. Elle contient également la logique pour lancer et contrôler le jeu jusqu'à ce qu'il se termine.

Attributs :

1. finished (bool) : Indique si le jeu est terminé ou non.
2. rooms (list) : Une liste de toutes les salles présentes dans le jeu.
3. commands (dict) : Un dictionnaire contenant les commandes disponibles, associées à leurs descriptions et actions.
4. player (Player) : L'instance du joueur, qui interagit avec le monde du jeu.

Méthodes :

1. __init__(self) :
    - Le constructeur de la classe, qui initialise les variables et prépare le jeu pour la suite.
2. setup(self) :
    - Configure les commandes, les salles, les objets et les personnages.
    - Elle définit des commandes comme "help", "quit", "go", etc., et les associe à des actions spécifiques (par exemple, la fonction Actions.go pour la commande "go").
    - Elle initialise les salles, leurs descriptions et leurs connexions (exits).
    - Elle crée des objets (items) et les place dans des salles spécifiques.
    - Elle instancie des personnages non-joueurs (PNJ) et les associe aux différentes pièces du jeu.
3. play(self) :
    - Le cœur du jeu. Elle appelle la méthode setup() pour tout préparer, puis affiche un message de bienvenue et attend l'entrée du joueur.
    - Elle traite les commandes jusqu'à ce que l'attribut finished soit True.
4. process_command(self, command_string) :
    - Analyse la commande entrée par le joueur. Elle sépare la chaîne en mots et détermine si le mot de commande est valide.
    - Si la commande est reconnue, elle appelle l'action correspondante en passant la liste de mots et le nombre de paramètres attendus.
5. print_welcome(self) :
    - Affiche un message de bienvenue et donne des informations sur la situation actuelle du jeu, comme la description de la salle dans laquelle se trouve le joueur.

Classe Item et Inventory

Les classes Item et Inventory sont utilisées pour gérer les objets dans le jeu.

La classe Item représente un objet du jeu avec trois attributs principaux :

1. name (str) : Le nom de l'objet, comme "épée", "clé", ou "carte".
2. description (str) : Une description de l'objet, qui explique ce qu'il est ou son utilité.
3. weight (float) : Le poids de l'objet, exprimé en kilogrammes (kg).

Méthodes :

- __init__(self, name, description, weight) : Le constructeur de la classe qui initialise un objet Item avec les valeurs fournies pour name, description, et weight.
- __str__(self) : La méthode spéciale __str__ qui permet de représenter l'objet sous forme de chaîne de caractères. Elle renvoie une chaîne formatée comprenant le nom, la description et le poids de l'objet.

La classe Inventory gère l'inventaire des objets du joueur ou d'une salle. L'inventaire est représenté par un ensemble (set), ce qui permet de garantir l'unicité des objets.

1. items (set) : Un ensemble contenant les objets qui font partie de l'inventaire.

Méthodes :

- __init__(self) : Le constructeur initialise un inventaire vide, représenté par un ensemble set vide.
- add(self, item) : Ajoute un objet à l'inventaire.
- remove(self, item) : Retire un objet de l'inventaire.
- contains(self, item_name) : Cette méthode permet de vérifier si un objet avec un nom spécifique est présent dans l'inventaire. Cela peut être utile pour déterminer si un joueur possède un certain objet avant de lui permettre de réaliser une action spécifique, comme entrer dans une grotte si le joueur possède la clé.
- get_inventory_description(self, context) : Cette méthode génère une description de l'inventaire. Le paramètre context peut être "player" ou "room", et il définit comment l'inventaire doit être affiché. Par exemple :
    - Si le contexte est "player", cela décrit l'inventaire du joueur.
    - Si le contexte est "room", cela décrit les objets présents dans la salle actuelle.

Classe Player

La classe Player représente le joueur dans le jeu, avec des attributs et des méthodes permettant de gérer son inventaire, ses déplacements et ses interactions avec le monde. Voici un détail des fonctionnalités et de l'implémentation de cette classe.

Attributs :

1. name (str) : Le nom du joueur.
2. current_room (Room) : La salle dans laquelle le joueur se trouve actuellement. C'est une instance de la classe Room qui devrait être définie ailleurs dans le code.
3. history (list) : Une liste qui enregistre les salles que le joueur a visitées.
4. inventory (Inventory) : L'inventaire du joueur, qui est une instance de la classe Inventory (probablement définie dans un autre fichier). L'inventaire contient les objets que le joueur possède.
5. max_weight (int) : Le poids maximal que le joueur peut porter. Par défaut, il est fixé à 15 kg.

Méthodes :

1. __init__(self, name, max_weight=15) :
    - Le constructeur initialise un joueur avec un nom donné, une salle de départ (current_room est initialement None), un inventaire vide, et un poids maximal de 15 kg.
2. set_starting_room(self, starting_room) :
    - Cette méthode permet de définir la salle de départ du joueur et ajoute cette salle à l'historique des salles visitées (history).

3. move(self, direction) :
    - Cette méthode permet de déplacer le joueur d'une salle à l'autre, en fonction de la direction spécifiée (direction). Si le joueur tente d'aller dans une direction qui n'a pas de sortie, un message d'erreur est affiché.
    - Si le joueur entre dans une salle appelée "Grotte", la méthode vérifie s'il possède les objets nécessaires (la carte et l'épée) pour gagner le jeu. Si le joueur possède ces objets, il entre dans la grotte et le jeu se termine avec une victoire. Sinon, un message d'erreur est affiché.
    - Les PNJ dans la salle sont déplacés après chaque déplacement du joueur en appelant la méthode move_pnj().
4. move_pnj(self) :
    - Cette méthode déplace les PNJ (personnages non-joueurs) présents dans la salle actuelle après que le joueur ait effectué son déplacement. Elle itère sur les personnages de la salle et appelle la méthode move() pour chaque personnage.
5. get_history(self) :
    - Retourne une chaîne de caractères décrivant les salles déjà visitées par le joueur. Cela permet de suivre l'historique des déplacements du joueur.
6. get_inventory(self) :
    - Retourne une chaîne de caractères qui décrit l'inventaire du joueur en appelant la méthode get_inventory_description() de la classe Inventory.
7. get_total_weight(self) :
    - Calcule le poids total des objets dans l'inventaire du joueur en additionnant le poids de chaque objet. Cela est utile pour vérifier si le joueur dépasse son poids maximal.

Classe Room

La classe Room représente une pièce ou un lieu dans le jeu, avec des attributs et des méthodes qui gèrent les éléments présents dans cette pièce, ainsi que les différentes sorties possibles. Elle permet de définir les interactions du joueur avec l'environnement.

Attributs :

1. name (str) : Le nom de la salle. Par exemple, "Salle d'entrée" ou "Grotte".
2. description (str) : Une description textuelle de la salle. Cela permet de donner un aperçu du lieu au joueur.
3. exits (dict) : Un dictionnaire contenant les sorties de la pièce. Les clés sont des directions, et les valeurs sont les pièces adjacentes dans ces directions.
4. inventory (Inventory) : Un inventaire des objets présents dans la pièce. L'inventaire est une instance de la classe Inventory, qui contient des objets que le joueur peut ramasser.
5. characters (dict) : Un dictionnaire des personnages non-joueurs (PNJ) présents dans la pièce. Les clés sont les noms des personnages, et les valeurs sont des instances de la classe Character.

Méthodes :

1. __init__(self, name, description) :
    - Le constructeur initialise la salle avec un nom et une description donnés, crée un inventaire vide, et initialise un dictionnaire vide pour les personnages et les sorties.
2. get_exit(self, direction) :
    - Cette méthode permet de récupérer la pièce dans une direction donnée. Si la direction existe dans les sorties de la pièce, elle retourne la salle correspondante, sinon elle retourne None.

3. get_exit_string(self) :
    - Cette méthode génère une chaîne de caractères qui décrit toutes les sorties possibles de la pièce. Elle parcourt le dictionnaire des sorties et génère une liste de directions. Cela permet au joueur de savoir dans quelles directions il peut se déplacer.
4. get_long_description(self) :
    - Cette méthode retourne une description détaillée de la pièce, qui inclut la description de la salle et une liste des sorties disponibles. Elle utilise les méthodes description et get_exit_string pour générer cette description complète.
5. get_inventory(self) :
    - Cette méthode retourne une description de l'inventaire de la pièce, c'est-à-dire les objets présents dans la pièce et les personnages non-joueurs. Elle appelle la méthode get_inventory_description de la classe Inventory et parcourt le dictionnaire characters pour fournir des informations sur les PNJ.
6. add_character(self, character) :
    - Cette méthode ajoute un personnage non-joueur (PNJ) à la pièce. Le personnage est ajouté au dictionnaire characters de la pièce avec son nom comme clé et l'objet Character comme valeur.
7. remove_character(self, character_name) :
    - Cette méthode retire un personnage non-joueur de la pièce, en supprimant le personnage du dictionnaire characters à l'aide de son nom.