Avant toute chose,nous tenons à nous excuser pour le retard par rapport à la deadline du projet et aussi aviser qu'en plus des conseils et recommandations de notre intervenante, nous avons eu recours à l'IA afin de nous aider à affiner notre raisonnement par rapport à certains exercices du projet mais également d'éclairer certaines zones d'ombres.

TBA: Scary escape.

Nous avons nommé notre TBA **"scary escape"**,qui se traduit littéralement par *"évasion effrayante".*

Synopsis: Votre voiture est tombée en panne au bord d'une forêt lugubre lors d'un voyage et vous apercevez un manoir au loin.
Vous entrez dans la forêt pour rejoindre le manoir. Vous arrivez devant le manoir ,il semble abandonné. Vous décidez donc de faire demi-tour.
Malheureusement vous êtes perdue et décidez donc de retourner prendre refuge dans le manoir.
Une fois entré,vous vous rendez compte que vous ne pouvez plus sortir et en parcourant les pièces vous voyez qu'elles sont très macabres. Au fil du temps vous vous rendrez compte que vous n'êtes pas seul(e) car le manoir s'avère être habité et il semble que son propriétaire(un tueur en série) n'apprécie pas que quelqu'un y soit entré sans son autorisation.

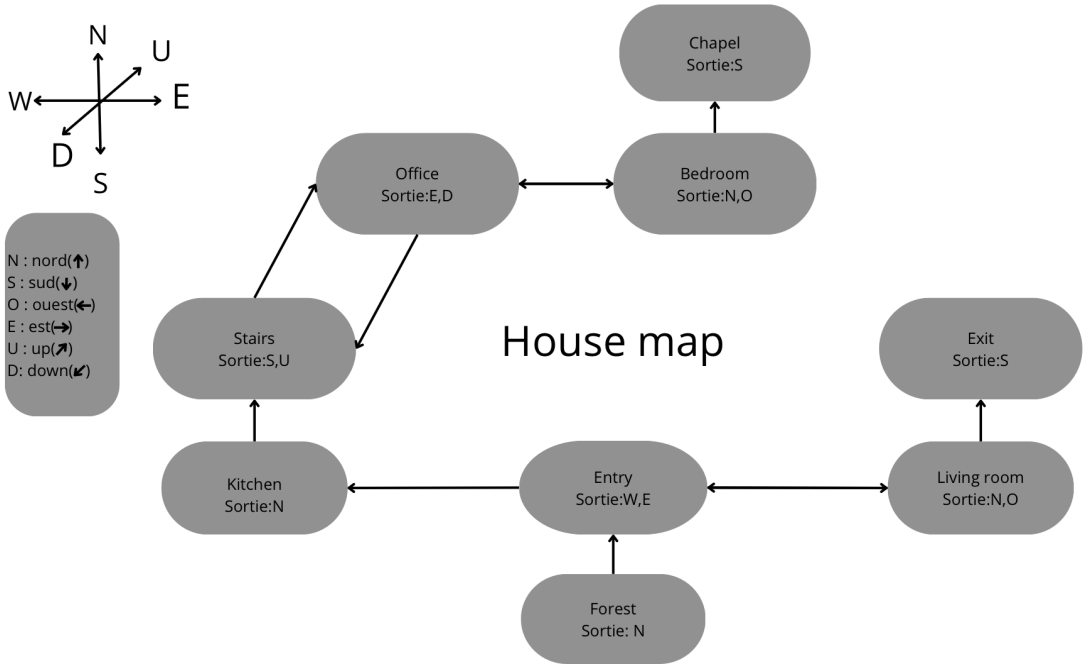
Guide d'utilisateur
Pour jouer à *Scary Escape*, vous avez à votre disposition plusieurs commandes à entrer dans le terminal vous permettant de progresser dans le jeu et d'interagir avec lui:

python game.py:pour lancer le jeu.
quit: pour quitter le jeu.
go *direction*: pour se déplacer dans la map.
look: pour observer l'environnement,voir quels sont les objets présents dans la pièce où vous êtes.
take *nom de l'objet*: pour prendre un objet et donc l'ajouter dans votre inventaire.
drop *nom de l'objet*: pour déposer un objet,donc le retirer de votre inventaire.
check: pour consulter votre inventaire et donc voir les objets que vous possédez.
back: pour revenir dans pièce précédente.
help: pour afficher les commandes disponibles.
talk: pour déclencher une action talk sur le pnj.

Il y a plusieurs pièces dans le manoir contenant divers objets tels qu'une lampe, un fusil(avec une seule balle dans le chargeur),un couteau et des clés.
Pour gagner à *Scary Escape,* vous devez récupérer les clés de la sortie et également les clés de la voiture située à la sortie du manoir afin de pouvoir espérer rentrer chez vous.
Les clés de la sortie se trouvent dans l'une des pièces et celles de la voiture dans une autre.

Vous perdez si vous ne récupérez pas les clés de la sortie et celles de la voiture. Le cas échéant, le propriétaire du manoir vous attrapera et vous tuera.

MAP



Guide développeur

- **Une première version**

- *Passage interdit:* Pour cette étape au niveau de la classe **Game** nous avons mis des passages interdits entre plusieurs lieux. Par exemple pour *living_room* et *kitchen* le joueur ne passe pas directement entre les deux. Au niveau des deux lieux nous avons remplacé par *None* des directions.

- *Sens unique:* Le joueur ne peut pas aller de l'entrée à la forêt. Pour ça on a remplacé la direction de la forêt par *None* au niveau de l'entrée.

- *Commande vide:* Si le joueur entre une commande qui n'est pas valide, un message d'erreur s'affiche. Pour celà:

```
# If the command is not recognized, print an error message
        if command_word not in self.commands.keys():
            print(f"\nCommande '{command_word}' non reconnue. Entrez 'help' pour voir la liste des commandes disponibles.\n")
```

Cette syntaxe va vérifier si la commande entrée par le joueur est reconnue parmi les commandes valides. L'attribut *commands* de la classe **Game** est un dictionnaire initialisé à vide et la syntaxe *if command_word not in self.commands.keys():* permettra d' itérer sur le dict des commandes valides.

- **Votre propre univers**

- *Ajouter des lieux:* Il y a neuf lieux dans la MAP. Pour ajouter un lieu (au niveau de la classe **Game**) nous l'avons instancié et l'attribut *self.exit* (dictionnaire initialisé à vide) sert de référencement pour l'instanciation.

- *Gestion direction inconnue:* Si le joueur entre une direction qui n'est pas valide, un message d'erreur s'affiche. Pour celà au niveau de la méthode *go* de la classe **Action** nous avons mis:

```
if valid_direction is None:
            print(f"\nDirection: '{direction_input}' non reconnue.\n")
            return False
```

Cette syntaxe va vérifier si la direction entrée par le joueur est reconnue parmi les directions valides et afficher la direction entrée suivie de *"non reconnue* si elle n'est pas bonne.

- **Ajouter un historique**

- *Construire un historique:* Pour cette étape un attribut *history*(initialisé comme liste vide) a été créé au niveau de la classe **Player.**
- *Revenir en arrière:* Pour celà la commande *back* a été créé au niveau de classe **Game**:

La syntaxe *self.commands["back"]* = permet d'ajouter la commande look au dictionnaire dans lequel les commandes du jeu sont stockées.

Ainsi au niveau de la classe **Actions** une méthode *def look()* a été créé:

- **Ajouter des objets**

- Créer des objets: Une classe Item a été créée afin de représenter les objets que le joueur pourra trouver dans les différents lieux de la MAP.

```
class Item:
    def __init__(self,name,description,weight):
        self.name= name
        self.description= description
        self.weight= weight
    def __str__(self):
        return f"{self.name} : {self.description} ({self.weight})"
```

```
fusil= Item('fusil','un fusil à pompe prêt à être dégainer à bout portant','3.5kg')
couteau=Item('couteau','un couteau tranchant pour un massacre amusant','70g')
clé_sortie=Item('clé de la sortie','les clés de la porte de sortie','1g')
lampe=Item('une lampe','pour tenter de térasser l obscurité','70g')
clé_voiture=Item('clé de la voiture','les clés de la voiture','1g')
```

- *Ajouter un inventaire (joueur):* Au niveau de la classe **Player** un attribut *inventory*(initialisé comme dictionnaire à vide) a été créé. Dans cette même classe une méthode *get_inventory()*

```
def get_inventory(self):
```

```
if len(self.inventory) == 0:
    print("Il n'y a rien dans votre inventaire.")
else:
    print("Vous avez dans votre inventaire :")
    for item in self.inventory.values():
        print(f"    - {item.name} : {item.description} ({item.weight})")
```

a été créée afin que lorsque le joueur interroge son inventaire avec la commande *check*, si il n'est pas vide il y a itération sur ce dernier afin d'afficher pour chaque item: son nom,sa description et son poids notamment avec la synthaxe *print(f"   - {item.name} : {item.description} ({item.weight})").*

- *Ajouter un inventaire (lieu):* Pour celà au niveau de la classe Room un attribut inventory_items (initialisé comme dictionnaire à vide) a été créé. Une méthode *get_inventory_items()*

```
def get_inventory_items(self):
    if len(self.inventory_items) == 0:
        print("Il n'y a rien ici dans cette pièce")
    else:
        print("La pièce contient :")
        for item in self.inventory_items.values():
            print(f"    - {item.name} : {item.description} ({item.weight})")
```

a été créée afin que chaque pièce contiennent un historique des différents objets qu'elle contient.

Ceci étant fait, des modifications ont donc été apportées au niveau de la classe **Game** afin d'ajouter à chaque pièce les objets voulus. Au niveau de cette classe,par exemple pour la pièce *'entry'*(l'entrée du manoir) on a:

```
entry = Room("Entry", "dans l'entrée du manoir. Vous êtes dans un hall sombre et effrayant.")
        self.rooms.append(entry)
        lampe = Item('lampe', 'une lampe pour tenter de térasser l\'obscurité', '70g')
        entry.inventory_items["lampe"] =lampe
```

On ajoute l'item *lampe* à l'inventaire des objets de la pièce *entry*.

- *Observer l'environnement:* Pour celà la commande *look* a été créé au niveau de classe **Game**:

La syntaxe *self.commands["look"] = look* permet d'ajouter la commande look au dictionnaire dans lequel les commandes du jeu sont stockées.
Ainsi au niveau de la classe **Actions** une méthode *def look()* a été créé:

```
def look(game, list_of_words, number_of_parameters):
        """Affiche les items présents dans la pièce actuelle."""
        game.player.current_room.look()
```

Plus spécifiquement la syntaxe *game.player.current_room.look()* va permettre d'appeler la méthode look sur l'objet current_room du joueur ce qui signifie que chaque salle possède une méthode look qui gère l'affichage des objets et des éléments présents dans celle-ci.

NB: L'ajout de commande se fait de manière analogique pour toutes les autres commandes du jeu.

- *Prendre / reposer un item:* Pour celà les commandes *take* et *drop* ont été créées au niveau de classe **Game:**

```
take= Command("take", "nom_item : Pour prendre un objet", Actions.take, 1)
            self.commands["take"] = take
drop = Command("drop", "nom_item : Pour déposer un objet", Actions.drop, 1)
      self.commands["drop"] = drop
```

Ainsi au niveau de la classe **Actions** des méthodes *def look()* et *def drop()* ont été créées :

```
def take(game, list_of_words, number_of_parameters):
    """Permet au joueur de prendre un item."""

    if len(list_of_words) != number_of_parameters + 1:
        print("Erreur dans le nombre de paramètres.")
        return False

    name_item = list_of_words[1].lower()
    game.player.current_room.take(name_item, game.player)

    return True


  def drop(game, list_of_words, number_of_parameters):
    if len(list_of_words) != number_of_parameters + 1:
        print("Erreur dans le nombre de paramètres.")
        return False

    name_item = list_of_words[1].lower()
    game.player.current_room.drop(name_item, game.player)

    return True
```

Ici spécifiquement la syntaxe *game.player.current_room.drop(name_item, game.player)* appelle la méthode drop sur l'objet current_room du joueur impliquant

que chaque salle a une méthode drop qui gère la logique pour permettre au joueur de déposer l'objet spécifié.

.

- *Vérifier son inventaire:* Pour celà la commande check a été créée au niveau de classe **Game**:

```
check = Command("check", ": afficher votre inventaire", Actions.check, 0)
        self.commands["check"] = check
```

Ainsi au niveau de la classe **Actions** une méthode *def check()* a été créé:

```
def check(game, list_of_words, number_of_parameters):
    """Affiche l'inventaire du joueur."""
    player = game.player
    if len(player.inventory) == 0:
        print("Il n'y a rien dans votre inventaire.")
    else:
        print("Vous avez dans votre inventaire :")
        for item in player.inventory.values():
            print(f"    - {item.name} : {item.description} ({item.weight})")
```

Spécifiquement ici la syntaxe *player=game.player* permet l'extraction de l'objet player de l'objet game ce qui va permettre d'accéder plus facilement aux propriétés et méthodes du joueur.

- **Ajouter des PNJ**
- *Créer un PNJ:*
- *Intégrer les PNJ à la map:*
- *Déplacer les PNJ:*
- *Interagir avec les PNJ:*
-
- La classe Character possède plusieurs attributs :

- nom, description, pièce actuelle
- messages, référence au jeu, mode de débogage
- Nous avons définit trois méthodes principales:
- move() : qui gère le déplacement du personnage, notamment vers l'entrée si le joueur est dans la chapelle
- get_msg() :qui renvoie un message du personnage, avec une logique spéciale si le joueur est dans la chapelle
- __str__() :qui fournit une représentation textuelle du personnage
- La classe intègre une logique de jeu spécifique, notamment autour des pièces "Chapel" (Chapelle) et "Entry" (Entrée).
- Il y a une interaction entre le personnage et l'état du jeu, notamment la position du joueur.
-

Perspective de développement

Dans notre jeu nous aurions aimé améliorer plusieurs aspects:

- Nous aurions aimé avoir une interface graphique mais le temps ne nous a pas permis de le faire.
- Nous aurions aimé avoir plusieurs PNJ dans le jeu,mais le temps de compréhension et de perception de cette partie du projet ne nous l'a pas permis.
- Nous aurions également aimé pouvoir ajouter une énigme de telle sorte à ce que si le joueur est dans la pièce de *manor_exit* et qu'il ait les deux clés, il doit en plus de cela résoudre cette énigme et pour cela il aurait eu deux tentatives pour le faire.

Vidéo démonstration du jeu

-Présentation de l'univers du jeu
-démonstration d'un parcours perdant
-démonstration d'un parcours gagnant

```
"""Partie principale du jeux"""
# Description: Game class
# Import modules"""Partie principale du jeux"""
# Description: Game class
# Import modules

from item import Item
from room import Room
```

```
from player import Player
from command import Command
from character import Character

class Game:
    """Classe principale du jeu."""
    # Constructor
    def __init__(self):
        """Initialise les attributs du jeu."""
        self.finished = False
        self.rooms = []
        self.commands = {}
        self.player = None
        self.debug = True
        self.npc = None
    # Setup the game
    def setup(self):
        from actions import Actions
        """Initialise le jeu en configurant les commandes,
          les pièces, les objets, le joueur et les PNJ."""
        self.actions = Actions()
        self.setup_commands()
        self.setup_rooms()
        self.setup_item()
        self.setup_player()
        self.setup_npcs()
    # L'affichage du DEBUG
    def debug_print(self, message):
        """Affiche un message de debug si le mode debug est activé."""
        if self.debug:
            print(f"DEBUG: {message}")
    def setup_commands(self):
        """
       Configure les commandes du jeu.

       Cette méthode initialise toutes les commandes disponibles dans
le jeu,
       en définissant leur nom, description, action associée et nombre
de paramètres.
       """
        # Setup commands
        self.commands["help"] = Command({"command_word":"help",
```

```
                                        "help_string":" : afficher
cette aide",
                                        "action": self.actions.help,
                                        "number_of_parameters": 0})
        self.commands["quit"] = Command({"command_word":"quit",
"help_string": " : quitter le jeu",
                                        "action":
self.actions.quit,"number_of_parameters": 0})
        self.commands["go"] = Command({"command_word":"go",
                                       "help_string": " <direction> :
se déplacer dans une "
                                      "direction cardinale (N, E, S,
O)",
                                      "action":
self.actions.go,"number_of_parameters": 1})
        self.commands["back"] = Command({"command_word":"back",
                                        "help_string": " : revenir à
la pièce précédente",
                                        "action":
self.actions.back,"number_of_parameters": 0})
        self.commands["look"] = Command({"command_word":"look",
                                        "help_string": ": observer
l'environnement",
                                        "action":
self.actions.look,"number_of_parameters": 0})
        self.commands["take"] = Command({"command_word":"take",
                                        "help_string": "nom_item :
Pour prendre un objet",
                                        "action":
self.actions.take,"number_of_parameters": 1})
        self.commands["drop"] = Command({"command_word":"drop",
                                        "help_string": "nom_item :
Pour déposé un objet",
                                        "action":
self.actions.drop,"number_of_parameters": 1})
        self.commands["check"] = Command({"command_word":"check",
                                         "help_string":": afficher
votre inventaire",
                                         "action":
self.actions.check,"number_of_parameters": 0})
        self.commands["talk"] = Command({"command_word":"talk",
                                        "help_string":" <personnage> :
parler à un personnage",
```

```
                                        "action":
self.actions.talk,"number_of_parameters": 1})
    def setup_rooms(self):
        """Configure le jeu en initialisant les pièces et les
sorties."""
        # Setup rooms
        forest = Room("Forest", "dans une forêt lugubre."
                      " Vous entendez une brise légère à travers la
cime des arbres.")
        self.rooms.append(forest)
        entry = Room("Entry", "dans l'entrée du manoir."
                     " Vous êtes dans un hall sombre et effrayant.")
        self.rooms.append(entry)
        kitchen = Room("Kitchen", "dans une cuisine macabre."
                       " La piece est couverte de sang et le frigo est
rempli d'organes humain.")
        self.rooms.append(kitchen)
        living_room = Room("Living room", "dans un salon maudit remplis
d'âmes errante.")
        self.rooms.append(living_room)
        stairs= Room("Stairs", "devant la cage d'escalier.")
        self.rooms.append(stairs)
        manor_exit = Room("Manor exit", "devant la sortie mais la porte
est fermée à clé."
                      " Vous voyez une voiture a l'extérieure qui"
                      " peut vous aider à sortir de la forêt.")
        self.rooms.append(manor_exit)
        office = Room("Office", "dans un bureau remplis d'armes."
                      " La pièce ressemble a une chambre de torture.")
        self.rooms.append(office)
        chapel = Room("Chapel", "dans une pièce ammenager come une
chapelle."
                      "Elle est completement detruite")
        self.rooms.append(chapel)
        bedroom = Room("Bedroom", "dans une chambre plein de tache de
sang."
                       " La chambre est completement rouge.")
        self.rooms.append(bedroom)
        self.rooms.extend([forest, entry, kitchen, living_room,
                           stairs, manor_exit, office, chapel,
bedroom])
        # Create exits for rooms
```

```
        forest.exits = {"N" :entry, "E" : None, "S" : None, "O" :
None,"U": None,"D": None}
        entry.exits = {"N" : None, "E" : living_room, "S" : None, "O" :
kitchen,"U": None,"D": None}
        living_room.exits = {"N" : manor_exit, "E" : None,
                             "S" : None, "O" : entry,"U": None,"D":
None}
        kitchen.exits = {"N" : stairs, "E" : entry, "S" : None, "O" :
None,"U": None,"D": None}
        stairs.exits = {"N" : None, "E" : None, "S" : kitchen, "O" :
None,"U" : office}
        office.exits = {"N" : None, "E" : bedroom, "S" : None, "O" :
None,"U": None,"D": stairs}
        bedroom.exits = {"N" : chapel, "E" : None, "S" : None, "O" :
office,"U": None,"D": None}
        chapel.exits = {"N" : None, "E" : None, "S" : bedroom, "O" :
None,"U": None,"D": None}
        manor_exit.exits = {"N" : None, "E" : None,
                            "S" : living_room, "O" : None,"U":
None,"D": None}
    def setup_item(self):
        """
        Configure les objets du jeu.

        Cette méthode crée tous les objets du jeu et les place
        dans leurs pièces respectives.
        """
        # Setup inventory
        exit_key= Item("exit_key", "les clés de la porte de sortie",
"1g")
        self.rooms[6].inventory_items["exit_key"] = exit_key
        flashlight = Item("flashlight", "une lampe pour tenter de
térasser l'obscurité", "70g")
        self.rooms[1].inventory_items["flashlight"] = flashlight
        knife = Item("Knife", "un couteau tranchant pour un massacre
amusant", "70g")
        self.rooms[2].inventory_items["Knife"] = knife
        gun = Item("Gun", "un fusil à pompe prêt à être dégainé à bout
portant", "3.5kg")
        self.rooms[3].inventory_items["Gun"] = gun
        car_key = Item("car_key", "les clés de la voiture", "1g")
        self.rooms[8].inventory_items["car_key"] =car_key
    def setup_player(self):
```

```
        """
        Configure le joueur.

        Cette méthode initialise le joueur, lui demande son nom,
        et définit la pièce de départ ainsi que son historique.
        """
        # Setup player and starting room
        self.player = Player(input("\nEntrez votre nom: "))
        self.player.current_room = self.rooms[0]
        self.player.history.append(self.player.current_room.name)
    def setup_npcs(self):
        """
        Configure le personnage non-joueur.

        Cette méthode crée le PNJ unique du jeu et
        l'ajoute à sa pièce de départ.
        """
        # Creating the NPC
        forest_room = next((room for room in self.rooms if room.name ==
"Forest"), None)
        if forest_room:
            self.npc = Character({"name": "Anthony",
                                  "description": "Un tueur en série à
l'apparence de Jeffrey Dahmer",
                                  "current_room": forest_room,
                                  "msgs": ["Je chasse ma prochaine victime
!"]},
                                 game=self, debug=self.debug)
        forest_room.characters[self.npc.name.lower()] = self.npc
    # Play the game
    def play(self):
        """
        Démarre la boucle principale du jeu.

        Cette méthode initialise le jeu, affiche un message de
bienvenue,
        et traite les commandes du joueur jusqu'à ce que le jeu soit
terminé.
        """
        self.setup()
        self.print_welcome()
        while not self.finished:
            self.npc.move()
```

```
            self.process_command(input("> "))
    # Process the command entered by the player
    def process_command(self, command_string) -> None:
        """
        Traite une commande entrée par le joueur.

        Args:
            command_string (str): La commande entrée par le joueur
            sous forme de chaîne de caractères.

        Cette méthode analyse la commande, vérifie si elle est valide,
        et exécute l'action correspondante.
        """
        # Split the command string into a list of words
        list_of_words = command_string.split(" ")
        command_word = list_of_words[0].lower()
        if command_word not in self.commands:
            print(f"\nDirection '{command_word}' non reconnue."
              " Entrez 'help' pour voir la liste des commandes
disponibles.\n")
        else:
            command = self.commands[command_word]
        if command_word == "go" and self.player.current_room.name ==
"manor_exit":
            #vérifie si les objest nécessaires sont dans l'inventaire
            print("Avez-vous les clés de la sortie et de la voiture ?")
            answer=input("Répondez par oui ou non ")
            if answer == 'oui':
                print('vous avez gagné')
            else:
                print("vous n'avez pas toutes les clés. Trouvez les et
revenez")
        else:
            command.action(self, list_of_words,
command.number_of_parameters)
    # Print the welcome message
    def print_welcome(self):
        """
        Affiche le message de bienvenue et l'introduction du jeu.

        Cette méthode imprime un message de bienvenue personnalisé pour
le joueur,
```

```
        décrit le contexte initial du jeu, et fournit des instructions
de base.
        Elle affiche également la description de la pièce de départ du
joueur.
        """
        print(f"\nBienvenue {self.player.name} dans ce jeu d'horreur et
de survie !")
        print("Votre voiture est tombée en panne au bord d'une forêt "
              "lors d'un voyage et vous apercevez un manoir au loin.")
        print("Vous entrez dans la forêt pour rejoindre le manoir.")
        print("vous arrivez devant le manoir qui a l'air d'être
abandoné,"
              "donc vous décidez de faire demi-tour")
        print("Malheureseument vous êtes perdue et decidez donc de "
              "retourner prendre refuge dans le manoir")
        print("Entrez 'help' si vous avez besoin d'aide.")
        print(self.player.current_room.get_long_description())
def main():
    """Fonction principale pour lancer le jeu."""
    Game().play()
if __name__ == "__main__":
    main()


"""Partie principale du jeux"""
# Description: Game class
# Import modules

from item import Item
from room import Room
from player import Player
from command import Command
from character import Character

class Game:
    """Classe principale du jeu."""
    # Constructor
    def __init__(self):
        """Initialise les attributs du jeu."""
        self.finished = False
        self.rooms = []
        self.commands = {}
        self.player = None
```

```
        self.debug = True
        self.npc = None
    # Setup the game
    def setup(self):
        from actions import Actions
        """Initialise le jeu en configurant les commandes,
          les pièces, les objets, le joueur et les PNJ."""
        self.actions = Actions()
        self.setup_commands()
        self.setup_rooms()
        self.setup_item()
        self.setup_player()
        self.setup_npcs()
    # L'affichage du DEBUG
    def debug_print(self, message):
        """Affiche un message de debug si le mode debug est activé."""
        if self.debug:
            print(f"DEBUG: {message}")
    def setup_commands(self):
        """
       Configure les commandes du jeu.

       Cette méthode initialise toutes les commandes disponibles dans
le jeu,
       en définissant leur nom, description, action associée et nombre
de paramètres.
       """
        # Setup commands
        self.commands["help"] = Command({"command_word":"help",
                                         "help_string":" : afficher
cette aide",
                                         "action": self.actions.help,
                                         "number_of_parameters": 0})
        self.commands["quit"] = Command({"command_word":"quit",
"help_string": " : quitter le jeu",
                                         "action":
self.actions.quit,"number_of_parameters": 0})
        self.commands["go"] = Command({"command_word":"go",
                                       "help_string": " <direction> :
se déplacer dans une "
                                      "direction cardinale (N, E, S,
O)",
```

```
                                        "action":
self.actions.go,"number_of_parameters": 1})
        self.commands["back"] = Command({"command_word":"back",
                                         "help_string": " : revenir à
la pièce précédente",
                                         "action":
self.actions.back,"number_of_parameters": 0})
        self.commands["look"] = Command({"command_word":"look",
                                         "help_string": ": observer
l'environnement",
                                         "action":
self.actions.look,"number_of_parameters": 0})
        self.commands["take"] = Command({"command_word":"take",
                                         "help_string": "nom_item :
Pour prendre un objet",
                                         "action":
self.actions.take,"number_of_parameters": 1})
        self.commands["drop"] = Command({"command_word":"drop",
                                         "help_string": "nom_item :
Pour déposé un objet",
                                         "action":
self.actions.drop,"number_of_parameters": 1})
        self.commands["check"] = Command({"command_word":"check",
                                          "help_string":": afficher
votre inventaire",
                                          "action":
self.actions.check,"number_of_parameters": 0})
        self.commands["talk"] = Command({"command_word":"talk",
                                         "help_string":" <personnage> :
parler à un personnage",
                                         "action":
self.actions.talk,"number_of_parameters": 1})
    def setup_rooms(self):
        """Configure le jeu en initialisant les pièces et les
sorties."""
        # Setup rooms
        forest = Room("Forest", "dans une forêt lugubre."
                      " Vous entendez une brise légère à travers la
cime des arbres.")
        self.rooms.append(forest)
        entry = Room("Entry", "dans l'entrée du manoir."
                     " Vous êtes dans un hall sombre et effrayant.")
        self.rooms.append(entry)
```

```
        kitchen = Room("Kitchen", "dans une cuisine macabre."
                       " La piece est couverte de sang et le frigo est
rempli d'organes humain.")
        self.rooms.append(kitchen)
        living_room = Room("Living room", "dans un salon maudit remplis
d'âmes errante.")
        self.rooms.append(living_room)
        stairs= Room("Stairs", "devant la cage d'escalier.")
        self.rooms.append(stairs)
        manor_exit = Room("Manor exit", "devant la sortie mais la porte
est fermée à clé."
                      " Vous voyez une voiture a l'extérieure qui"
                      " peut vous aider à sortir de la forêt.")
        self.rooms.append(manor_exit)
        office = Room("Office", "dans un bureau remplis d'armes."
                      " La pièce ressemble a une chambre de torture.")
        self.rooms.append(office)
        chapel = Room("Chapel", "dans une pièce ammenager come une
chapelle."
                      "Elle est completement detruite")
        self.rooms.append(chapel)
        bedroom = Room("Bedroom", "dans une chambre plein de tache de
sang."
                       " La chambre est completement rouge.")
        self.rooms.append(bedroom)
        self.rooms.extend([forest, entry, kitchen, living_room,
                           stairs, manor_exit, office, chapel,
bedroom])
        # Create exits for rooms
        forest.exits = {"N" :entry, "E" : None, "S" : None, "O" :
None,"U": None,"D": None}
        entry.exits = {"N" : None, "E" : living_room, "S" : None, "O" :
kitchen,"U": None,"D": None}
        living_room.exits = {"N" : manor_exit, "E" : None,
                             "S" : None, "O" : entry,"U": None,"D":
None}
        kitchen.exits = {"N" : stairs, "E" : entry, "S" : None, "O" :
None,"U": None,"D": None}
        stairs.exits = {"N" : None, "E" : None, "S" : kitchen, "O" :
None,"U" : office}
        office.exits = {"N" : None, "E" : bedroom, "S" : None, "O" :
None,"U": None,"D": stairs}
```

```
        bedroom.exits = {"N" : chapel, "E" : None, "S" : None, "O" :
office,"U": None,"D": None}
        chapel.exits = {"N" : None, "E" : None, "S" : bedroom, "O" :
None,"U": None,"D": None}
        manor_exit.exits = {"N" : None, "E" : None,
                            "S" : living_room, "O" : None,"U":
None,"D": None}
    def setup_item(self):
        """
        Configure les objets du jeu.

        Cette méthode crée tous les objets du jeu et les place
        dans leurs pièces respectives.
        """
        # Setup inventory
        exit_key= Item("exit_key", "les clés de la porte de sortie",
"1g")
        self.rooms[6].inventory_items["exit_key"] = exit_key
        flashlight = Item("flashlight", "une lampe pour tenter de
térasser l'obscurité", "70g")
        self.rooms[1].inventory_items["flashlight"] = flashlight
        knife = Item("Knife", "un couteau tranchant pour un massacre
amusant", "70g")
        self.rooms[2].inventory_items["Knife"] = knife
        gun = Item("Gun", "un fusil à pompe prêt à être dégainé à bout
portant", "3.5kg")
        self.rooms[3].inventory_items["Gun"] = gun
        car_key = Item("car_key", "les clés de la voiture", "1g")
        self.rooms[8].inventory_items["car_key"] =car_key
    def setup_player(self):
        """
        Configure le joueur.

        Cette méthode initialise le joueur, lui demande son nom,
        et définit la pièce de départ ainsi que son historique.
        """
        # Setup player and starting room
        self.player = Player(input("\nEntrez votre nom: "))
        self.player.current_room = self.rooms[0]
        self.player.history.append(self.player.current_room.name)
    def setup_npcs(self):
        """
        Configure le personnage non-joueur.
```

```
        Cette méthode crée le PNJ unique du jeu et
        l'ajoute à sa pièce de départ.
        """
        # Creating the NPC
        forest_room = next((room for room in self.rooms if room.name ==
"Forest"), None)
        if forest_room:
            self.npc = Character({"name": "Anthony",
                                  "description": "Un tueur en série à
l'apparence de Jeffrey Dahmer",
                                  "current_room": forest_room,
                                  "msgs": ["Je chasse ma prochaine victime
!"]},
                                 game=self, debug=self.debug)
        forest_room.characters[self.npc.name.lower()] = self.npc
    # Play the game
    def play(self):
        """
        Démarre la boucle principale du jeu.

        Cette méthode initialise le jeu, affiche un message de
bienvenue,
        et traite les commandes du joueur jusqu'à ce que le jeu soit
terminé.
        """
        self.setup()
        self.print_welcome()
        while not self.finished:
            self.npc.move()
            self.process_command(input("> "))
    # Process the command entered by the player
    def process_command(self, command_string) -> None:
        """
        Traite une commande entrée par le joueur.

        Args:
            command_string (str): La commande entrée par le joueur
            sous forme de chaîne de caractères.

        Cette méthode analyse la commande, vérifie si elle est valide,
        et exécute l'action correspondante.
        """
```

```
        # Split the command string into a list of words
        list_of_words = command_string.split(" ")
        command_word = list_of_words[0].lower()
        if command_word not in self.commands:
            print(f"\nDirection '{command_word}' non reconnue."
              " Entrez 'help' pour voir la liste des commandes
disponibles.\n")
        else:
            command = self.commands[command_word]
        if command_word == "go" and self.player.current_room.name ==
"manor_exit":
            #vérifie si les objest nécessaires sont dans l'inventaire
            print("Avez-vous les clés de la sortie et de la voiture ?")
            answer=input("Répondez par oui ou non ")
            if answer == 'oui':
                print('vous avez gagné')
            else:
                print("vous n'avez pas toutes les clés. Trouvez les et
revenez")
        else:
            command.action(self, list_of_words,
command.number_of_parameters)
    # Print the welcome message
    def print_welcome(self):
        """
        Affiche le message de bienvenue et l'introduction du jeu.

        Cette méthode imprime un message de bienvenue personnalisé pour
le joueur,
        décrit le contexte initial du jeu, et fournit des instructions
de base.
        Elle affiche également la description de la pièce de départ du
joueur.
        """
        print(f"\nBienvenue {self.player.name} dans ce jeu d'horreur et
de survie !")
        print("Votre voiture est tombée en panne au bord d'une forêt "
              "lors d'un voyage et vous apercevez un manoir au loin.")
        print("Vous entrez dans la forêt pour rejoindre le manoir.")
        print("vous arrivez devant le manoir qui a l'air d'être
abandoné,"
              "donc vous décidez de faire demi-tour")
        print("Malheureseument vous êtes perdue et decidez donc de "
```

```
                "retourner prendre refuge dans le manoir")
        print("Entrez 'help' si vous avez besoin d'aide.")
        print(self.player.current_room.get_long_description())
def main():
    """Fonction principale pour lancer le jeu."""
    Game().play()
if __name__ == "__main__":
    main()




from item import Item
from room import Room
from player import Player
from command import Command
from character import Character

class Game:
    """Classe principale du jeu."""
    # Constructor
    def __init__(self):
        """Initialise les attributs du jeu."""
        self.finished = False
        self.rooms = []
        self.commands = {}
        self.player = None
        self.debug = True
        self.npc = None
    # Setup the game
    def setup(self):
        from actions import Actions
        """Initialise le jeu en configurant les commandes,
          les pièces, les objets, le joueur et les PNJ."""
        self.actions = Actions()
        self.setup_commands()
        self.setup_rooms()
        self.setup_item()
        self.setup_player()
        self.setup_npcs()
    # L'affichage du DEBUG
    def debug_print(self, message):
        """Affiche un message de debug si le mode debug est activé."""
```

```
        if self.debug:
            print(f"DEBUG: {message}")
    def setup_commands(self):
        """
       Configure les commandes du jeu.

       Cette méthode initialise toutes les commandes disponibles dans
le jeu,
       en définissant leur nom, description, action associée et nombre
de paramètres.
       """
        # Setup commands
        self.commands["help"] = Command({"command_word":"help",
                                         "help_string":" : afficher
cette aide",
                                         "action": self.actions.help,
                                         "number_of_parameters": 0})
        self.commands["quit"] = Command({"command_word":"quit",
"help_string": " : quitter le jeu",
                                         "action":
self.actions.quit,"number_of_parameters": 0})
        self.commands["go"] = Command({"command_word":"go",
                                        "help_string": " <direction> :
se déplacer dans une "
                                       "direction cardinale (N, E, S,
O)",
                                       "action":
self.actions.go,"number_of_parameters": 1})
        self.commands["back"] = Command({"command_word":"back",
                                         "help_string": " : revenir à
la pièce précédente",
                                         "action":
self.actions.back,"number_of_parameters": 0})
        self.commands["look"] = Command({"command_word":"look",
                                         "help_string": ": observer
l'environnement",
                                         "action":
self.actions.look,"number_of_parameters": 0})
        self.commands["take"] = Command({"command_word":"take",
                                         "help_string": "nom_item :
Pour prendre un objet",
                                         "action":
self.actions.take,"number_of_parameters": 1})
```

```
        self.commands["drop"] = Command({"command_word":"drop",
                                         "help_string": "nom_item :
Pour déposé un objet",
                                         "action":
self.actions.drop,"number_of_parameters": 1})
        self.commands["check"] = Command({"command_word":"check",
                                          "help_string":": afficher
votre inventaire",
                                          "action":
self.actions.check,"number_of_parameters": 0})
        self.commands["talk"] = Command({"command_word":"talk",
                                         "help_string":" <personnage> :
parler à un personnage",
                                         "action":
self.actions.talk,"number_of_parameters": 1})
    def setup_rooms(self):
        """Configure le jeu en initialisant les pièces et les
sorties."""
        # Setup rooms
        forest = Room("Forest", "dans une forêt lugubre."
                      " Vous entendez une brise légère à travers la
cime des arbres.")
        self.rooms.append(forest)
        entry = Room("Entry", "dans l'entrée du manoir."
                     " Vous êtes dans un hall sombre et effrayant.")
        self.rooms.append(entry)
        kitchen = Room("Kitchen", "dans une cuisine macabre."
                       " La piece est couverte de sang et le frigo est
rempli d'organes humain.")
        self.rooms.append(kitchen)
        living_room = Room("Living room", "dans un salon maudit remplis
d'âmes errante.")
        self.rooms.append(living_room)
        stairs= Room("Stairs", "devant la cage d'escalier.")
        self.rooms.append(stairs)
        manor_exit = Room("Manor exit", "devant la sortie mais la porte
est fermée à clé."
                      " Vous voyez une voiture a l'extérieure qui"
                      " peut vous aider à sortir de la forêt.")
        self.rooms.append(manor_exit)
        office = Room("Office", "dans un bureau remplis d'armes."
                      " La pièce ressemble a une chambre de torture.")
        self.rooms.append(office)
```

```
        chapel = Room("Chapel", "dans une pièce ammenager come une
chapelle."
                      "Elle est completement detruite")
        self.rooms.append(chapel)
        bedroom = Room("Bedroom", "dans une chambre plein de tache de
sang."
                       " La chambre est completement rouge.")
        self.rooms.append(bedroom)
        self.rooms.extend([forest, entry, kitchen, living_room,
                           stairs, manor_exit, office, chapel,
bedroom])
        # Create exits for rooms
        forest.exits = {"N" :entry, "E" : None, "S" : None, "O" :
None,"U": None,"D": None}
        entry.exits = {"N" : None, "E" : living_room, "S" : None, "O" :
kitchen,"U": None,"D": None}
        living_room.exits = {"N" : manor_exit, "E" : None,
                             "S" : None, "O" : entry,"U": None,"D":
None}
        kitchen.exits = {"N" : stairs, "E" : entry, "S" : None, "O" :
None,"U": None,"D": None}
        stairs.exits = {"N" : None, "E" : None, "S" : kitchen, "O" :
None,"U" : office}
        office.exits = {"N" : None, "E" : bedroom, "S" : None, "O" :
None,"U": None,"D": stairs}
        bedroom.exits = {"N" : chapel, "E" : None, "S" : None, "O" :
office,"U": None,"D": None}
        chapel.exits = {"N" : None, "E" : None, "S" : bedroom, "O" :
None,"U": None,"D": None}
        manor_exit.exits = {"N" : None, "E" : None,
                            "S" : living_room, "O" : None,"U":
None,"D": None}
    def setup_item(self):
        """
        Configure les objets du jeu.

        Cette méthode crée tous les objets du jeu et les place
        dans leurs pièces respectives.
        """
        # Setup inventory
        exit_key= Item("exit_key", "les clés de la porte de sortie",
"1g")
        self.rooms[6].inventory_items["exit_key"] = exit_key
```

```
        flashlight = Item("flashlight", "une lampe pour tenter de
térasser l'obscurité", "70g")
        self.rooms[1].inventory_items["flashlight"] = flashlight
        knife = Item("Knife", "un couteau tranchant pour un massacre
amusant", "70g")
        self.rooms[2].inventory_items["Knife"] = knife
        gun = Item("Gun", "un fusil à pompe prêt à être dégainé à bout
portant", "3.5kg")
        self.rooms[3].inventory_items["Gun"] = gun
        car_key = Item("car_key", "les clés de la voiture", "1g")
        self.rooms[8].inventory_items["car_key"] =car_key
    def setup_player(self):
        """
        Configure le joueur.

        Cette méthode initialise le joueur, lui demande son nom,
        et définit la pièce de départ ainsi que son historique.
        """
        # Setup player and starting room
        self.player = Player(input("\nEntrez votre nom: "))
        self.player.current_room = self.rooms[0]
        self.player.history.append(self.player.current_room.name)
    def setup_npcs(self):
        """
        Configure le personnage non-joueur.

        Cette méthode crée le PNJ unique du jeu et
        l'ajoute à sa pièce de départ.
        """
        # Creating the NPC
        forest_room = next((room for room in self.rooms if room.name ==
"Forest"), None)
        if forest_room:
            self.npc = Character({"name": "Anthony",
                                  "description": "Un tueur en série à
l'apparence de Jeffrey Dahmer",
                                  "current_room": forest_room,
                                  "msgs": ["Je chasse ma prochaine victime
!"]},
                                 game=self, debug=self.debug)
        forest_room.characters[self.npc.name.lower()] = self.npc
    # Play the game
    def play(self):
```

```
        """
        Démarre la boucle principale du jeu.

        Cette méthode initialise le jeu, affiche un message de
bienvenue,
        et traite les commandes du joueur jusqu'à ce que le jeu soit
terminé.
        """
        self.setup()
        self.print_welcome()
        while not self.finished:
            self.npc.move()
            self.process_command(input("> "))
    # Process the command entered by the player
    def process_command(self, command_string) -> None:
        """
        Traite une commande entrée par le joueur.

        Args:
            command_string (str): La commande entrée par le joueur
            sous forme de chaîne de caractères.

        Cette méthode analyse la commande, vérifie si elle est valide,
        et exécute l'action correspondante.
        """
        # Split the command string into a list of words
        list_of_words = command_string.split(" ")
        command_word = list_of_words[0].lower()
        if command_word not in self.commands:
            print(f"\nDirection '{command_word}' non reconnue."
              " Entrez 'help' pour voir la liste des commandes
disponibles.\n")
        else:
            command = self.commands[command_word]
        if command_word == "go" and self.player.current_room.name ==
"manor_exit":
            #vérifie si les objest nécessaires sont dans l'inventaire
            print("Avez-vous les clés de la sortie et de la voiture ?")
            answer=input("Répondez par oui ou non ")
            if answer == 'oui':
                print('vous avez gagné')
            else:
```

```
                print("vous n'avez pas toutes les clés. Trouvez les et
revenez")
        else:
            command.action(self, list_of_words,
command.number_of_parameters)
    # Print the welcome message
    def print_welcome(self):
        """
        Affiche le message de bienvenue et l'introduction du jeu.

        Cette méthode imprime un message de bienvenue personnalisé pour
le joueur,
        décrit le contexte initial du jeu, et fournit des instructions
de base.
        Elle affiche également la description de la pièce de départ du
joueur.
        """
        print(f"\nBienvenue {self.player.name} dans ce jeu d'horreur et
de survie !")
        print("Votre voiture est tombée en panne au bord d'une forêt "
              "lors d'un voyage et vous apercevez un manoir au loin.")
        print("Vous entrez dans la forêt pour rejoindre le manoir.")
        print("vous arrivez devant le manoir qui a l'air d'être
abandoné,"
              "donc vous décidez de faire demi-tour")
        print("Malheureseument vous êtes perdue et decidez donc de "
              "retourner prendre refuge dans le manoir")
        print("Entrez 'help' si vous avez besoin d'aide.")
        print(self.player.current_room.get_long_description())
def main():
    """Fonction principale pour lancer le jeu."""
    Game().play()
if __name__ == "__main__":
    main()
```

```
"""Module définissant la classe Character pour le jeu."""

# Define the Character class.
```

```python
class Character():
    """Représente un personnage dans le jeu."""
    # Define the constructor.
    def __init__(self, character_info, game, debug=False):
        """
        Initialise un personnage.

        :param character_info: dict contenant name, description,
current_room, msgs
        :param game: instance du jeu
        :param debug: mode debug (par défaut False)
        """
        self.name = character_info['name']
        self.description = character_info['description']
        self.current_room = character_info['current_room']
        self.msgs = character_info['msgs']
        self.debug = debug
        self.game = game


    def move(self):
        """Déplace le personnage si nécessaire."""
        player = self.game.player
        if player.current_room.name == "Chapel":
            entry_room = next((room for room in self.game.rooms if
room.name == "Entry"), None)
            if entry_room and self.current_room != entry_room:
                old_room = self.current_room
                self.current_room.characters.pop(self.name.lower(),
None)
                self.current_room = entry_room
                entry_room.characters[self.name.lower()] = self
                if self.debug:
                    print(f"DEBUG: {self.name} s'est déplacé de
{old_room.name} "
                          f"à {entry_room.name}")
                return True
        if self.debug:
            print(f"DEBUG: {self.name} est resté dans
{self.current_room.name}")
        return False

    def get_msg(self):
```

```
        """Retourne le message actuel du personnage."""
        if not self.msgs:
            return "Ce personnage n'a rien à dire."
        if (self.game.player.current_room.name == "Chapel" and
            self.current_room.name == "Entry"):
            return ("Vous ne devriez pas être ici. "
                    "Mais vous en avez trop vu donc vous devez mourir")
        msg = self.msgs[self.msg_index]
        self.msg_index = (self.msg_index + 1) % len(self.msgs)
        return msg
def __str__(self):
    return f"{self.name} : {self.description}"
```

```
"""Module contenant la classe Item pour représenter les objets du
jeu."""
class Item:
    """classe représentant un objet dans le jeu."""
    def __init__(self,name,description,weight):
        """initialise un nouvel objet item

        Args:
        name (str): le nom de l'objet.
        description (str): la description de l'objet.
        weight (str): le poids de l'objet.
        """
        self.name= name
        self.description= description
        self.weight= weight
    def __str__(self):
        """Retourne les informations détaillées de l'objet."""
        return f"Nom: {self.name},\nDescription:
{self.description},\npoids: {self.weight}"
    def get_info(self):
        """Retourne les informations détaillées de l'objet"""
        return self.__str__()
    def is_heavy(self):
        """Détermine si l'objet est considéré comme lourd (plus de 1
kg)."""
        weight_value = float(self.weight.replace('kg', '').replace('g',
''))
```

```
        return weight_value > 1 if 'kg' in self.weight else
weight_value > 1000
    def is_usable(self):
         """Détermine si l'objet peut être utilisé par le joueur."""
         return True  # Par défaut, tous les objets sont utilisables
#Exemple d'objet
gun = Item("gun","un fusil à pompe prêt à être dégainer à bout
portant","3.5kg")
knife = Item("knife","un couteau tranchant pour un massacre
amusant","70g")
exit_key = Item("exit_key","des clés! par ici la sortie","1g")
flashlight = Item("flashlight", "pour tenter de térasser l'obscurité",
"70g")
car_key = Item("car_key", "les clés de la voiture", "1g")
```

```
"""
Ce module contient les actions pour un jeu d'aventure textuel. Il
définit les actions
que le joueur peut effectuer, telles que se déplacer, parler, regarder
et interagir avec des objets.
"""
# Description: The actions module.
# The actions module contains the functions that are called when a
command is executed.
# Each function takes 3 parameters:
# - game: the game object
# - list_of_words: the list of words in the command
# - number_of_parameters: the number of parameters expected by the
command
# The functions return True if the command was executed successfully,
False otherwise.
# The functions print an error message if the number of parameters is
incorrect.
# The error message is different depending on the number of parameters
expected by the command.
# The error message is stored in the MSG0 and MSG1 variables and
formatted
# with the command_word variable, the first word in the command.
# The MSG0 variable is used when the command does not take any
parameter.
import game
MSG0 = "\nLa commande '{command_word}' ne prend pas de paramètre.\n"
```

```
# The MSG1 variable is used when the command takes 1 parameter.
MSG1 = "\nLa commande '{command_word}' prend 1 seul paramètre.\n"
class Actions:
    """
    Gère les actions qu'un joueur peut effectuer dans le jeu.
    Cela inclut des actions comme déplacer le joueur,
    interagir avec l'environnement et gérer l'état du jeu.
    """
    def go(self,game, list_of_words, number_of_parameters):
        """
        Move the player in the direction specified by the parameter.
        The parameter must be a cardinal direction (N, E, S, O, U, D,
or alias ).

        Args:
            game (Game): The game object.
            list_of_words (list): The list of words in the command.
            number_of_parameters (int): The number of parameters
expected by the command.

        Returns:
            bool: True if the command was executed successfully, False
otherwise.

        Examples:

        >>> from game import Game
        >>> game = Game()
        >>> game.setup()
        >>> go(game, ["go", "N"], 1)
        True
        >>> go(game, ["go", "N", "E"], 1)
        False
        >>> go(game, ["go"], 1)
        False
        """
        player = game.player
        l = len(list_of_words)
        # If the number of parameters is incorrect, print an error
message and return False.
        if l != number_of_parameters + 1:
            command_word = list_of_words[0]
            print(MSG1.format(command_word=command_word))
```

```
            return False
        # Liste des directions valides
        direction = {"n": ["n", "nord"],
                     "e": ["e", "est"],
                     "s": ["s", "sud"],
                     "o": ["o", "ouest"],
                     "u": ["u", "haut"],
                     "d": ["d", "bas"] }
        # Get the direction from the list of words.
        direction_input = list_of_words[1].lower()
        # Check if the direction is valid (by checking the aliases).
        valid_direction = None
        for main_direction, alias in direction.items():
            if direction_input in alias:
                valid_direction = main_direction
                break
        # If the direction is not recognized, print an error message
        if valid_direction is None:
            print(f"\nDirection: '{direction_input}' non reconnue.\n")
            return False
        # Move the player in the direction specified by the parameter.
        player.move(valid_direction)
        print(player.current_room.get_long_description())
        print("\n" + player.get_history())
         # Vérifier si le joueur est dans la pièce de sortie
        if player.current_room.name == "Manor_exit":
        # Vérifier l'inventaire du joueur pour les deux clés
nécessaires.
            if "exit_key" in player.inventory and "car_key" in
player.inventory:
                print("\nFélicitations ! Vous avez les deux clés et
vous pouvez quitter le manoir.")
                print("Vous avez gagné le jeu !")
                game.finished = True  # Fin du jeu si les deux clés
sont présentes.
                return True
            else:
                print("\nVous n'avez pas les deux clés nécessaires pour
quitter le manoir.")
                print("Vous avez été tué... Le jeu est terminé.")
                game.finished = True  # Fin du jeu si les clés sont
manquantes.
                return False
```

```
        return True
    def quit(self,game, list_of_words, number_of_parameters):
        """
        Quit the game.

        Args:
            game (Game): The game object.
            list_of_words (list): The list of words in the command.
            number_of_parameters (int): The number of parameters
expected by the command.

        Returns:
            bool: True if the command was executed successfully, False
otherwise.

        Examples:

        >>> from game import Game
        >>> game = Game()
        >>> game.setup()
        >>> quit(game, ["quit"], 0)
        True
        >>> quit(game, ["quit", "N"], 0)
        False
        >>> quit(game, ["quit", "N", "E"], 0)
        False

        """
        l = len(list_of_words)
        # If the number of parameters is incorrect, print an error
message and return False.
        if l != number_of_parameters + 1:
            command_word = list_of_words[0]
            print(MSG0.format(command_word=command_word))
            return False
        # Set the finished attribute of the game object to True.
        player = game.player
        msg = f"\nMerci {player.name} d'avoir joué. Au revoir.\n"
        print(msg)
        game.finished = True
        return True
        pass
    def help(self,game, list_of_words, number_of_parameters):
```

```
        """
        Print the list of available commands.

        Args:
            game (Game): The game object.
            list_of_words (list): The list of words in the command.
            number_of_parameters (int): The number of parameters
expected by the command.

        Returns:
            bool: True if the command was executed successfully, False
otherwise.

        Examples:

        >>> from game import Game
        >>> game = Game()
        >>> game.setup()
        >>> help(game, ["help"], 0)
        True
        >>> help(game, ["help", "N"], 0)
        False
        >>> help(game, ["help", "N", "E"], 0)
        False

        """
        # If the number of parameters is incorrect, print an error
message and return False.
        l = len(list_of_words)
        if l != number_of_parameters + 1:
            command_word = list_of_words[0]
            print(MSG0.format(command_word=command_word))
            return False
        # Print the list of available commands.
        print("\nVoici les commandes disponibles:")
        for command in game.commands.values():
            print("\t- " + str(command))
        print()
        return True
    def back(self,game, list_of_words, number_of_parameters):
        """
    Move the player back to the previous room.
```

```
    Args:
        game (Game): The game object.
        list_of_words (list): The list of words in the command.
        number_of_parameters (int): The number of parameters expected
by the command.

    Returns:
        bool: True if the command was executed successfully, False
otherwise.

    Examples:

        >>> from game import Game
        >>> game = Game()
        >>> game.setup()
        >>> back(game, ["back"], 1)
        True
        >>> back(game, ["back", "N", "E"], 1)
        False
        >>> back , (game, ["back"], 1)
        False

        """
        l = len(list_of_words)
        if l != number_of_parameters + 1:
            command_word = list_of_words[0]
            print(MSG0.format(command_word=command_word))
            return False
        player=game.player
        # Verifying if it is possible to go back
        if len(player.history) < 2:
            print("\nImpossible de revenir en arrière,"
                  " vous n'avez visité qu'une seule pièce ou
aucune.\n")
            return False
        #Removing the room from the history and going back to the room
before
        player.history.pop()
        previous_room_name = player.history[-1]
        previous_room = next(room for room in game.rooms if room.name
== previous_room_name)
        player.current_room = previous_room
        print(previous_room.get_long_description())
```

```
        print("\n" + player.get_history())
        return True
    def look(self,game, list_of_words, number_of_parameters):
        """
        Affiche les items présents dans la pièce actuelle,
        les objets et les personnages présents.
        """
        current_room = game.player.current_room
        if len(current_room.inventory_items) == 0:
            print("Il n'y a rien d'intéressant dans cette pièce.")
        else:
            print("La pièce contient les objets suivants :")
            for item in current_room.inventory_items.values():
                print(f"    - {item.name} : {item.description}
({item.weight}g)")
        if current_room.characters:
            print("\nPersonnages présents :")
            for character in current_room.characters.values():
                print(f"    - {character.name} :
{character.description}")
        else:
            print("Il n'y a personne ici.")
            return True
    def take(self, game, list_of_words, number_of_parameters):
         """Permet au joueur de prendre un item."""
         if len(list_of_words) > number_of_parameters:
            item_name = list_of_words[1]  # Premier paramètre : nom de
l'objet
            current_room = game.player.current_room
         if item_name in current_room.inventory_items:
            item = current_room.inventory_items.pop(item_name)
            game.player.add_item_to_inventory(item)
            print(f"\nVous avez pris {item_name}.")
         else:
            print("\nCet objet n'est pas présent dans cette pièce.")

    def drop(self,game, list_of_words, number_of_parameters):
        """Permet au joueur de déposer un item dans la pièce
actuelle."""
        if len(list_of_words) != number_of_parameters + 1:
            print("Erreur dans le nombre de paramètres.")
            return False
        name_item = list_of_words[1].lower()
```

```
        name_item = list_of_words[1].lower()
        player = game.player
        current_room = player.current_room
        if name_item in player.inventory:
            item = player.inventory[name_item]
            current_room.inventory_items[name_item] = item  # Ajoute
l'item à la pièce
            del player.inventory[name_item]  # Retire l'item de
l'inventaire du joueur
            print(f"Vous avez déposé {item.name}.")
            return True
        else:
            print(f"Vous avez retiré {name_item} de votre inventaire.")
            return False
    def check(self,game, list_of_words, number_of_parameters):
        """Affiche l'inventaire du joueur."""
        player = game.player
        if len(player.inventory) == 0:
            print("Il n'y a rien dans votre inventaire.")
        else:
            print("Vous avez dans votre inventaire :")
            for item in player.inventory.values():
                print(f"    - {item.name} : {item.description}
({item.weight})")
    def talk(self,game, list_of_words, number_of_parameters):
        if len(list_of_words) != number_of_parameters + 1:
            print("Erreur dans le nombre de paramètres.")
            return False
        character_name = list_of_words[1].lower()
        current_room = game.player.current_room
        if character_name in current_room.characters:
            character = current_room.characters[character_name]
            print(f"{character.name} dit : {character.get_msg(game)}")
        else:
            print(f"Il n'y a pas de personnage nommé {character_name}
ici.")
        return True
```

```
"""
Ce module contient la classe Player, qui représente le joueur dans le
jeu.
```

```
La classe Player gère les éléments suivants :
- Le nom du joueur.
- La pièce actuelle du joueur.
- L'historique des pièces visitées par le joueur.
- L'inventaire des objets du joueur.

La classe permet au joueur de se déplacer, de consulter son historique,
 de gérer son inventaire, etc.
"""
class Player():
    """
    Classe représentant le joueur dans le jeu.
    Le joueur a un nom, une pièce actuelle,
    un historique de pièces visitées et un inventaire d'objets.
    """
    def __init__(self, name):
        """
        Initialise un nouveau joueur.

        Args:
            name (str): Le nom du joueur.
        """
        self.name = name
        self.current_room = None
        self.history = []
        self.inventory={}
    # Define the move method.
    def add_item_to_inventory(self, item):
        """Ajoute un objet à l'inventaire du joueur."""
        self.inventory[item.name] = item
        print(f"Vous avez ajouté {item.name} à votre inventaire.")
    def move(self, direction):
        """
        Déplace le joueur dans la direction spécifiée.

        Args:
        direction (str): La direction dans laquelle le joueur doit se
déplacer.

        Returns:
        bool: True si le déplacement a réussi, False sinon.
        """
```

```
        direction = direction.upper()
        if direction in self.current_room.exits:
            next_room = self.current_room.exits[direction]
            if next_room is not None:
                self.current_room = next_room
                self.history.append(self.current_room.name)
                return True
            print("\nAucune porte dans cette direction !\n")
            return False
        print(f"\nDirection '{direction}' non valide dans cette
pièce.\n")
        return False
    def get_history(self):
        """
        Retourne l'historique des pièces visitées par le joueur.

        Returns:
        str: Une chaîne représentant l'historique des pièces visitées.
        """
        history_str = "Vous avez déjà visité les pièces suivantes:\n"
        if self.history:
            history_str += "\n".join(f"     - {room_name}" for
room_name in self.history[:-1])
        return history_str.strip()
    def get_inventory(self):
        """
        Affiche l'inventaire du joueur.

        Returns:
        None
        """
        if len(self.inventory) == 0:
            print("Il n'y a rien dans votre inventaire.")
        else:
            print("Vous avez dans votre inventaire :")
            for item in self.inventory.values():
                print(f"    - {item.name} : {item.description}
({item.weight})")
```

```
"""
```

```
Ce module contient la classe Room, représentant une pièce dans le jeu.
Chaque pièce a un nom, une description, des objets et des personnages.
"""
class Room:
    """
    Représente une pièce dans le jeu, avec des sorties, des objets et
des personnages.

    Attributs :
        name (str): Le nom de la pièce.
        description (str): La description de la pièce.
        exits (dict): Un dictionnaire des sorties de la pièce
(direction -> Room).
        inventory_items (dict): Un dictionnaire des objets présents
dans la pièce.
        characters (dict): Un dictionnaire des personnages présents
dans la pièce.
    """
    def __init__(self, name, description):
        """
        Initialise une nouvelle instance de Room.

        Args:
            name (str): Le nom de la pièce.
            description (str): La description de la pièce.
        """
        self.name = name
        self.description = description
        self.exits = {}
        self.inventory_items={}
        self.characters = {}
    def get_exit(self, direction):
        """
        Retourne la pièce dans la direction donnée si elle existe.

        Args:
            direction (str): La direction vers laquelle le joueur veut
aller.

        Returns:
            Room: La pièce de destination si elle existe, sinon None.
        """
        if direction in self.exits:
```

```
            return self.exits[direction]
        return None
    def get_exit_string(self):
        """
        Retourne une chaîne décrivant les sorties de la pièce.

        Returns:
            str: Une chaîne listant les directions disponibles.
        """
        exit_string = "Sorties: "
        for direction,room in self.exits.items():
            if room:
                exit_string += direction + ", "
        exit_string = exit_string.strip(", ")
        return exit_string
    def get_long_description(self):
        """
        Retourne une description détaillée de la pièce, incluant les
sorties.

        Returns:
            str: Une chaîne contenant la description de la pièce et des
sorties.
        """
        return f"\nVous êtes
{self.description}\n\n{self.get_exit_string()}\n"
    def get_inventory_items(self):
        """
        Affiche les objets présents dans la pièce.

        Si la pièce ne contient aucun objet, un message l'indique.
        """
        if len(self.inventory_items) == 0:
            print("Il n'y a rien ici dans cette pièce")
        else:
            print("La pièce contient :")
            for item in self.inventory_items.values():
                print(f"    - {item.name} : {item.description}
({item.weight})")
```

```
"""
```

```
Ce module définit la classe Command, qui représente une commande dans
le jeu,
et les méthodes associées à l'exécution des commandes par
l'utilisateur.
"""
class Command:
    """
    This class represents a command. A command is composed of a command
word,
     a help string, an action and a number of parameters.

    Attributes:
        command_word (str): The command word.
        help_string (str): The help string.
        action (function): The action to execute when the command is
called.
        number_of_parameters (int): The number of parameters expected
by the command.
        alias (list[str]): A list of alternative words for the command.

    Methods:
        __init__(self, command_word, help_string, action,
number_of_parameters, alias)
         : The constructor.
        __str__(self) : The string representation of the command.
        matches(self, input_word): Checks if the input matches the
command word or its alias.

    Examples:

    >>> from actions import go
    >>> command = Command("go", "Permet de se déplacer dans une
direction.", go, 1)
    >>> command.command_word
    'go'
    >>> command.help_string
    'Permet de se déplacer dans une direction.'
    >>> type(command.action)
    <class 'function'>
    >>> command.number_of_parameters
    1
    """
    def __init__(self, command_details):
```

```
        self.command_word = command_details.get("command_word")
        self.help_string = command_details.get("help_string")
        self.action = command_details.get("action")
        self.number_of_parameters =
command_details.get("number_of_parameters")
        self.alias = command_details.get("alias", None)
    def matches(self, input_word):
        """
        Vérifie si le mot saisi correspond au mot de commande ou à l'un
de ses alias.

        Args:
            input_word (str): Le mot saisi par l'utilisateur.

        Returns:
            bool: Retourne True si le mot saisi correspond au mot de
            commande ou à l'un de ses alias, sinon False.
        """
        if self.alias is None:
            return input_word.lower() == self.command_word.lower()
        return input_word.lower() == (self.command_word.lower() or
                                      input_word.lower()
                                      in map(str.lower, self.alias))
    def __str__(self):
        return  self.command_word \
                + self.help_string
```