Haunted Hill

Guide Utilisateur

Il s'agit d'un jeu textuel horrifique dans lequel on incarne Evelyn Westwood (donc on ne choisit pas son nom). Le joueur arrive dans un manoir et a la possibilité d'explorer le rez-de-chaussée du manoir.
Le joueur peut explorer les pièces dans chaque étage puis fouiller des endroits dans ces pièces où il peut trouver des objets utiles pour progresser dans l'histoire.
En lançant le programme, le joueur doit entrer start pour commencer puis il a accès aux commandes suivantes :
- help (afficher les commandes disponibles)
- quit (quitter le jeu)
- go (aller dans une pièce)
- look (fouiller un endroit dans la pièce)
- take (prendre un objet)

Le jeu se termine lorsqu'on arrive à l'entrée des escaliers menant au sous-sol ou lorsqu'Evelyn assiste à assez de phénomènes surnaturels pour faire une crise cardiaque.

N.B : Ceci est une description de la démo du jeu. Le jeu complet devrait avoir 5 étages et le paramètres des phénomènes surnaturels ne ferait pas perdre mais plutôt débloquer une « mauvaise » fin.

Guide Développeur

Classes (prenant pour base le document « Architecture du projet » donné pour le QCM)

La classe Game

C'est la classe principale du jeu. L'environnement (les lieux ete leurs connexions, les items et leur localisation, etc) est décrit par la méthode setup(). La méthode play() implémente un mécanisme itératif de jeu au tour par tour. La méthode process_command() transforme la chaîne de caractères entrée au clavier en un objet Python capable de communiquer avec structurer de données qui gardent la mémoire de l'état de jeu.

Les attributs

- isdead : bool
- rooms : list[Rooms]
- commands : dict[str, Command]
- player : Player

Les méthodes

- __init__(self) → None
- setup() → None
- play(self) → None
- process_command(self, command_string : str) → None
- print_welcome(self) → None

La classe Actions

La classe Actions n'est pas instanciable. Elle regroupe les fonctions qui permettent au joueur d'interagir avec le jeu. Toutes les fonctions ont la même signature. Les fonctions :

- go(), start(), quit(), help(), look(), take()

Les paramètres (identiques pour chacune des fonctions) :

- game : Game
- list_of_words : list[str]
- number_of_parameters : int

La valeur de retour (identique pour chacune des fonctions)

- bool

La classe Command

La classe Command associe une fonction de la classe Actions à une chaîne de caractères.

Les attributs

- command_word : str
- help_string : str
- action : Action
- number_of_words : int

Les méthodes

- __init__(self, command_word, help_string, action, number_of_parameters) → None
- __str__(self) → str

La classe Item

La classe Item décrit les caractéristiques des objets utilisables.

Les attributs

- self.name : str
- self.description : str

Les méthodes

- __init_(self, name, description) → None

La classe Room

La classse Room décrit les lieux du jeu.

Les attributs

- name → str
- description →str
- exits_room → dict(str, Room)
- exits_ssroom → dict(str, Ssroom)
- ssrooms → dict(str, Ssroom)

Les méthodes

- __init_(self, name, description) → None
- get_long_description(self) → str

La classe Ssroom

La classe Ssroom décrit les « sous-pièces », les endroits à fouiller, les ssrooms qui se trouvent dans les lieux/pièces du jeu.

Les attributs

- name →str
- description →str
- objects → dict(str, Item)

Les méthodes

- __init_(self, name, description) → None
- get_long_description(self) → str

La classe Player

La classe Player décrit le joueur.

Les attributs

- name → str
- current_room : Room | None
- current_ssroom : Ssroom | None
- history : list[Room]
- inventory : set()
- isdead : bool
- state : int

Les méthodes

- __init_(self, name) → None
- move(self, direction : str) → bool
- get_exit(self, direction : str) → Room | None
- get_exit_string(self) → str

Perspectives de développement

Améliorations globales

- Plusieurs étages du manoir accessibles après certaines actions (5 étages)
> L'objectif final est d'implémenter une structure avec tous les étages du manoir (sous-sol, rdc, 1er et 2e étages, toit). Chaque étage est accessible lorsque l'énigme de la salle précédente a été résolue, c'est-à-dire qu'un objet particulier (comme une clé) a été ajouté à l'inventaire après avoir accompli des actions dans un certain ordre.
Par exemple, dans les toilettes du rdc, on peut faire en sorte de faire apparaître une lampe-torche dans la cuvette des toilettes après avoir examiné la tâche rouge sur le miroir (élément surnaturel). On ajoute l'item lampe dans la ssroom « Cuvette » lorsque le joueur « récupère » l'objet « Tâche rouge ».
Dans la « démo », le parcours gagnant donne accès au sous-sol seulement après avoir trouvé la clé sur le canapé du salon.
L'organisation se fera avec trois dictionnaires imbriqués avec des sections dédiées aux floors/rooms/ssrooms dans la méthode move pour passer de l'un à l'autre pour les trois niveaux. Dans le code actuel, il y a 2 sections dans la méthode move (une pour changer de current_room et une autre pour changer de current_ssroom).
- On propose deux fins en fonction de l'état psychologique (attribut « state » de player) d'Evelyn. A chaque élément surnaturel rencontré par Evelyn, le compteur state est incrémenté de 1 et à partir d'un certain palier, il n'est plus possible d'avoir une des deux fins.

Améliorations de confort ou pour aller plus loin

- Implémentation d'une fonction « combiner » pour fusionner deux objets et pouvoir les utiliser dans une énigme
- Faire en sorte que les noms des pièces/sous-pièces/objets puissent être écrits avec des espaces dans la fonction go/look/take
- Ajout d'un historique qui apparaît chaque fois qu'on entre dans une pièce où tous les objets ont été récupérés
- A partir d'un certain palier de l'attribut state, l'écran de l'ordinateur affiche des « hallucinations » (changement de la langue, inversion des mots, randomisation etc), le clavier pourrait passer en Qwerty…
> Le niveau de folie d'Evelyn influence des paramètres extra-diégétiques.