DungeonCrawler

Documentation

1 EXECUTION DU JEU

Le jeu est écrit en langage Python 3.11 et utilise les librairies suivantes (et leurs dépendances)

`Pygame 2.5.0, random, math, csv 1.0`

Le jeu a été programmé en utilisant l'environnement Pycharm, mais devrait fonctionner avec tout autre environnement. En cas de problème lié à l'exécution du jeu, contacter l'équipe professorale dans le canal Général de l'équipe Teams.

Pour les questions concernant l'utilisation du jeu et les fonctions utiles, utiliser le canal **Question Jeu DungeonCrawler**.

2 DESCRIPTION ET OBJECTIFS DU JEU

Le jeu DungeonCrawler et un labyrinthe basé sur une grille dans lequel vous trouverez des artefacts (coins), des trésors et des ennemis. Il y a aussi des obstacles distribués sur les passages et une sortie. L'objectif du jeu est de sortir du labyrinthe. L'entrée et point de départ est indiqué par une tuile bleue, et la sortie par une tuile verte. Les murs sont formés de tuiles grises et les passages par des tuiles noires.

Les artefacts, trésors et obstacles sont distribués aléatoirement sur une tuile de passage. Vous pouvez ramasser les artefacts et les trésors simplement en les touchant. Il est impossible de traverser un obstacle.

Les ennemis, ou monstres, occupent une tuile complète – il est impossible de les contourner (quoiqu'il puisse y avoir des passages alternatifs). Lorsque votre joueur entre en contact avec un monstre, le combat est instantanément déclenché. Selon le résultat du combat, le monstre disparaitra, ou votre joueur meure et le jeu termine.

Si vous atteignez la sortie, le jeu termine avec des félicitations. Le score correspond au total d'artefacts ramassé. Chaque jeton vaut 1, et les trésors valent 10, un combat réussi contre un monstre vaut 100. Un chronomètre de calcul également le temps pris pour compléter le jeu.

À la fin d'un partie, vous pouvez presser sur la barre d'espace pour recommencer le jeu.

Au démarrage du jeu, le joueur est créé dans la tuile de départ. Vous pouvez déplacer le joueur en utilisant les flèches du clavier ou les touches « wasd ». Il existe aussi des fonctionnalités supplémentaires pour vous aider au développement de votre joueur. La touche «`p`» permet d'accéder à la fonction `make_perception_list` et la touche «`m`» permet d'accéder à la fonction `mock_fight`. De plus, les touches «`space`» et «`u`» permettent respectivement de voir l'état de la porte et de tenter de déverrouiller la porte en envoyant une clé.

Toutes les touches clavier seront remplacées par une interaction avec votre joueur intelligent. Une fonction `on_AI_input(instruction)` permet d'envoyer des instructions au jeu. Vous pouvez modifier cette fonction selon vos besoins. De même, votre joueur n'est pas confiné à des mouvements dans les 4 directions principales (haut, bas, droite gauche) et vous pouvez définir d'autres mouvements dans cette fonction.

Cependant, **si vous jugez que vous devez modifier d'autres fonctions du jeu, le demander à l'équipe professorale avec votre justification.**

Nous recommandons de développer votre joueur dans une classe et un fichier isolé du jeu Games2D afin d'éviter de surcharger le fichier.

3 LABYRINTHE

Le labyrinthe est généré à partir d'un fichier .csv contenu dans le dossier *assets*. La distribution initiale contient deux labyrinthes avec quatre variations augmentant en niveau de difficulté :

- mazeSmall/Medium_0 : Labyrinthe avec artefacts et trésors seulement
- maze Small/Medium_1 : Labyrinthe avec artefacts, trésors et obstacles
- maze Small/Medium_2 : Labyrinthe avec artefacts, trésors, obstacles, porte et ennemis non-bloquant
- maze Small/Medium_3 : Labyrinthe avec artefacts, trésors, obstacles, portes, et ennemis bloquants.

Votre joueur a accès au fichier csv au démarrage du jeu. Attention ! Vous ne pouvez pas « préanalyser » le fichier en dehors du jeu. **Toute l'analyse, la planification, la navigation et l'optimisation doivent se faire pendant le déroulement du jeu.**

Dans le fichier csv, le 1 représente des murs et les 0 des passages vides. Les tuiles contenant des artefacts, trésors, obstacles, portes et monstres sont indiquées par les lettres, C, T, O, D et M. La position exacte d'un item (obstacles, artefacts et trésors) dans la tuile est générée aléatoirement au démarrage du jeu. Les monstres et les portes occupent leur tuile entière.

4 CLASSES

4.1 GAMES2D

Classe principale qui gère le jeu, s'occupe de l'initialisation, des contrôles et de l'affichage graphique.

`On_execute` : Boucle principale du jeu. Récupère les évènements (horloge, quitter le jeu, clavier), modifie le statut du jeu et génère l'affichage. En cas de victoire ou de défaite, deux boucles vont faire un dernier affichage avec un message et attendre que l'utilisateur quitte le jeu ou redémarre une partie.

À modifier :

```
pygame.event.pump()
keys = pygame.key.get_pressed()
```

```
self.on_keyboard_input(keys)
# self.on_AI_input(instruction)
```

Vous devez modifier ces lignes pour passer d'une interface clavier à une interface avec votre joueur intelligent. Votre agent peut envoyer une seule instruction à la fois.

4.2 PLAYER

Classe qui gère la création du joueur, ses déplacements et ses attributs de combat.

Vous avez accès aux fonctions `get_position`, `get_attributes` et `set_attributes` de cette classe. À l'initialisation du joueur, les attributs sont générés de manière aléatoire. C'est à vous de les modifier pour gagner vos combats (voir classe Monster). Vous pouvez changer vos attributs entre les combats.

NOTE : En cours de développement, vous pouvez changer les constantes (`constants.py`) et la vitesse du joueur (`player.speed`) afin de faciliter certains tests ou valider différentes approches. Cependant, la validation doit se faire avec les valeurs par défaut.

4.3 MAZE

Classe qui gère la création, l'exécution et la préparation de l'affichage du labyrinthe.

À L'initialisation la classe lit le fichier csv contenant le labyrinthe. Les murs, items, obstacles, portes et monstres sont consignés dans des listes qui permettent la vérification de collisions dans la classe Games2D.

La fonction `make_perception_list` est dans cette classe et retourne les éléments visibles par le joueur pour chacune des listes. La distance de perception est déterminée par la constante PERCEPTION_RADIUS.

Une option de debug est disponible dans cette fonction. En activant le code commenté, le rectangle de vision sera affiché sur le jeu en appuyant sur la touche « p ».

Note : Si vous choisissez une autre méthode pour la perception de votre joueur, vous ne pouvez pas avoir un champ de vision plus grand – **vous devez vous limiter à un rayon de vision équivalent à 1.2 tuiles.** Un non-respect de cette consigne pourrait invalider votre solution et annuler les points accordés pour la navigation.

4.4 MONSTER

Classe qui gère la création du monstre et ses combats. Vous n'avez pas accès au détail de cette classe.

Lors d'une collision avec le monstre, la fonction `fight` est appelée et retourne le résultat du combat : victoire ou défaite.

Vous avez accès à la fonction `mock_fight` qui calcule le **nombre de rondes que vous avez gagné** contre le monstre ainsi qu'une valeur cumulative qui indique le **niveau de succès du combat**. Vous devez gagner les quatre rondes pour vaincre le monstre.

La valeur cumulative permet de déterminer si le combat était serré ou pas et en faveur de qui. Pour chaque ronde, une valeur négative indique un échec lamentable contre le monstre. Au contraire, une valeur s'approchant de 1 indique que votre joueur a gagné sans même avoir eu une égratignure. Les valeurs de fonction d'adéquation de chaque ronde sont additionnées pour retourner la valeur cumulative d'adéquation.

4.5 DOOR

Classe qui gère la création des portes et leur déverrouillage.

Chaque porte contient 3 à 6 cristaux de couleurs variées dans une serrure de couleur bronze, argent ou or. Pour ouvrir la porte, il faut retirer un des cristaux. Attention, retirer le mauvais cristal réinitialisera la porte avec un nouvel agencement de cristaux.

Vous pouvez obtenir l'état de la porte avec la fonction `look_at_door` de la classe Maze. L'état sera retourné sous la forme d'une liste énumérant d'abord la couleur de la serrure, ensuite les couleurs des cristaux dans l'ordre où ils apparaissent :

```
['METAL', 'FIRST COLOR', 'SECOND COLOR', 'THIRD COLOR', NULL, NULL, NULL]
```

Par exemple :

```
['silver', 'blue', 'red', 'red', 'yellow', 'black', NULL]
```

Afin de déterminer le bon cristal à retirer, il faut respecter les règles énumérées ci-dessous. Une fois le bon cristal identifié, vous devez envoyer la clé sous la forme suivante : `'POSITION'` où `POSITION` prend la valeur `first`, `second`, `third`, etc.

4.5.1 Règles pour déverrouiller la porte[1].

Trois cristaux :

S'il n'y a pas de cristal rouge, retirer le deuxième cristal.
Sinon, si le dernier cristal est blanc, retirer le dernier cristal.
Sinon, s'il y a plus d'un cristal bleu, retirer le dernier cristal bleu.
Sinon, retirer le premier cristal.

Quatre cristaux :

S'il y a plus d'un cristal rouge et si la couleur de la serrure est argent, retirer le dernier cristal rouge.
Sinon, si le dernier cristal est jaune et s'il n'y a pas de cristal rouge, retirer le premier cristal.
Sinon, s'il y a exactement un cristal bleu, retirer le premier cristal.
Sinon, s'il y a plus d'un cristal jaune, retirer le dernier cristal.
Sinon, retirer le deuxième cristal.

Cinq cristaux :

Si le dernier cristal est noir et si la couleur de la serrure est or, retirer le quatrième cristal.
Sinon, s'il y a exactement un cristal rouge et plus d'un cristal jaune, retirer le premier cristal.
Sinon, s'il n'y a pas de cristal noir, retirer le deuxième cristal.
Sinon, retirer le premier cristal.

Six cristaux :

S'il n'y a pas de cristal jaune et si la couleur de la serrure est bronze, retirer le troisième cristal.
Sinon, s'il y a exactement un cristal jaune et plus d'un cristal blanc, retirer le quatrième cristal.
Sinon, s'il n'y a pas de cristal rouge, retirer le dernier cristal.
Sinon, retirer le quatrième cristal.

[1] Inspiré du jeu *Keep talking and nobody explodes*: https://keeptalkinggame.com/