DungeonCrawler

Documentation

1 EXECUTION DU JEU

Le jeu est écrit en langage Python 3.10 et utilise les librairies suivantes (et leurs dépendances)

`pygame, random, math, csv`

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 Bug Report de l'équipe Teams.

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

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 le 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 (coins) ramassé. Chaque artefact vaut 1, et les trésors valent 10. Un chronomètre de calcul également le temps pris pour compléter le jeu.

Il n'y a pas présentement de menu pour redémarrer une partie – il faut redémarrer le jeu complet.

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 ». La touche « p » permet d'accéder à la fonction `make_perception_list` et la touche « m » permet d'accéder à la fonction `mock_fight`.

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

3 LABYRINTHE

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

- mazeMedium_0 : Labyrinthe avec artefacts et trésors seulement
- mazeMedium_1 : Labyrinthe avec artefacts, trésors et obstacles
- mazeMedium_2 : Labyrinthe avec artefacts, trésors, obstacles et un ennemi non-bloquant
- mazeMedium_3 : Labyrinthe avec artefacts, trésors, obstacles et trois ennemis dont deux bloquants.

Votre joueur a accès au fichier csv au démarrage du jeu. Attention ! Vous ne pouvez pas « pré-analyser » le fichier en dehors du jeu. **Toute l'analyse, planification, navigation, 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 et monstres sont indiquées par les lettres, C, T, O et M. La position exacte d'un item dans la tuile est générée aléatoirement au démarrage du jeu.

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.

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 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 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 tuile.** Un non-respect de cette consigne pourrait invalider votre solution et annuler les points accordés pour la navigation.

4.3 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 se rapprochant de 0 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. Une valeur de 0.5 par combat indique un match quasi nul. Les valeurs de chaque ronde sont additionnées de manière que la valeur totale soit entre 0 et 4.

4.4 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.

Vous pouvez changer vos attributs entre les combats.