Parcours développeur d'application – Python

Projet 3 : Aidez McGyver à s'échapper.

I. Description du projet

Le projet 3 du parcours Python consiste à élaborer un jeu en python dans lequel un personnage (McGyver) doit trouvez la sortie d'un Labyrinthe (le gardien), après avoir ramassé des objets répartis aléatoirement dans celui-ci. Le jeu doit être créé en utilisant le module PyGame. Les ressources graphiques sont fournies. Le programme doit être un standalone.

II. Réalisation du projet

Afin de réaliser ce programme j'ai procédé en différentes étapes :

A. Préparer les ressources

J'ai tout d'abord créé un répertoire dans lequel j'ai mis toutes les ressources graphiques dont j'allais avoir besoin (joueur, murs, objets…). J'ai ensuite redimensionné ces images pour qu'elles aient toutes une taille de 40x40 pixels, ce qui correspond à la taille que j'ai choisi pour les sprites du jeu.

B. Création du Labyrinthe

Pour le design du Labyrinthe, j'ai d'abord dessiné celui-ci à la main. Puis je l'ai recopié dans un fichier CSV en utilisant des 1 pour les cases comportant des murs et des 0 pour les vides. L'utilisation d'un tel format permet de modifier le niveau très facilement à l'aide d'un tableur. J'utilise par la suite le module « pandas » pour ouvrir ce fichier et le parcourir dans une boucle pour disposer les Sprites « Wall » sur la fenêtre.

C. Les classes

Les classes utilisées pour le programme sont stockées dans un fichier à part. J'ai choisi de créer une classe pour chaque type d'élément présent sur la fenêtre de jeu. A savoir : Les murs, le héro, le gardien et les objets. Chaque classe hérite d'une classe mère « Sprite » prenant au moins une image et une position en paramètres. J'ai choisi de stocker le path des images dans des constantes au sein du fichier « class_game.py » pour ne jamais avoir à toucher au code principal pour apporter des changements graphique au jeu. Ainsi, il suffit de mettre de nouvelle images dans le dossier « pictures » et de changer le path dans la constante du fichier « class_game.py » pour renouveler le visuel du jeu.

Deux classes comportes des ajouts majeurs par rapport à la classe « Sprite » dont elles héritent :

- La classe « Player » : Cette classe contient une fonction nommée « move » qui va gérer le déplacement du héros en prenant en compte les obstacles. Elle prend en paramètre la touche directionnelle pressée par le joueur et la liste des position des murs via un groupe de sprites « Wall » (voir plus bas).

- La Classe « Stuff » : Elle représente les objets à ramasser. Elle contient une constante de classe fixée à 7, qui est le nombre d'objets qui seront disposés sur le Labyrinthe. Cette constante sert de compteur dans la boucle de disposition des objets. Cette classe contient également une fonction pour le positionnement des objets. Elle va générer une position aléatoire pour un objet jusqu'à ce que ce nouvel objet ne soit pas en collision avec un mur, un objet déjà disposé ou avec un personnage (joueur ou sortie). La constante permet également de compléter le path vers l'image du sprite, pour que chaque objet ait une image différente.

D. Le cœur du programme : « game.py »

Outre les imports nécessaires au fonctionnement du jeu, on peut découper le fichier en 4 parties :

1. Création de la fenêtre de jeu et déclaration des variables :

La première partie du code a pour but d'initialiser une fenêtre de jeu via pygame et de déclarer des variables qui contiendront les groupes de sprites. La fenêtre permet d'afficher 17x17 sprites de 40x40 pixels.

2. Les fonctions :

Le code contient 3 fonctions :
- data_from_csv : Elle permet d'ouvrir le fichier csv contenant le labyrinthe qui sera parcouru par la suite.
- update_screen : Cette fonction sera appelée à chaque action du joueur pour réafficher les éléments du jeu en prenant en compte la nouvelle position du joueur et la disparition éventuelle des objets. Elle vérifie au passage si jamais le joueur rempli ou non la condition de victoire (à savoir être sur la case d'arrivée avec les 7 objets) ou de défaite (idem sans les 7 objets), et affiche l'écran de victoire ou de défaite le cas échéant.
- create-border : Cette fonction crée un cadre de mur permettant de matérialiser les bord du jeu. Dans un premier temps j'ai voulu gérer les collisions entre le joueur et les bords de la fenêtre de jeu. Cela m'a paru très difficile et je ne suis pas arrivé à un résultat satisfaisant, j'ai donc choisi cette option. Créer un cadre directement dans le code permet de modifier le fichier csv du labyrinthe sans se préoccuper des bordures.

3. La disposition des Sprites sur la fenêtre :

Comme indiqué précédemment, la disposition des sprites se fait grâce à une boucle qui parcours le fichier labyrinthe et crée les murs, le héros et le gardien. Ensuite, une deuxième boucle crée les objet grâce à la fonction de la classe.

4. Boucle principale :

Elle attend les entrées du joueur et appel les fonction de déplacement si une touche directionnelle est pressée. Elle vérifie si les conditions sont requises pour passer la variable de condition de victoire à « vrai »et enfin, elle appelle la fonction « update_screen » pour afficher les changements à l'écran.

III. Un Standalone

Une des consignes du projet était de fournir le projet en tant que standalone. Cette étape est celle qui m'a posé le plus de difficultés. J'ai envisagé plusieurs possibilité :
- L'utilisation d'un module appelé cx_freeze permettant de créer des exécutables du programme.
Mais l'exécutable créé n'est pas crossplateforme. Il dépend de l'OS sur lequel il à été créé.
-L'utilisation de « pypi » pour distribuer mon jeu sous forme de paquet stocké sur le site pypi et installable via la commande « pip install ». Cette méthode a également un inconvénient, elle nécessite que l'utilisateur ait déjà python installé sur sa machine. Cependant, j'ai choisi cette dernière car elle me paraît être une méthode plus standard pour distribuer des modules python et donc être une meilleure habitude à prendre pour la suite.

IV. Difficultés rencontrées

La principale difficulté rencontrée concernait les collisions entre les Sprites et le déplacement du joueur, pour éviter que le personnage ne traverse les murs... Pour résoudre ce souci, j'ai finalement utilisé une fonction du module « pygame » permettant de tester si un sprite est en collision avec un groupe de sprites. C'est ce qui m'a amené à créer un groupe contenant tous les murs pour gérer les collisions. De plus, cela m'a permis de réaliser l'intérêt des groupes de sprites et d'en utiliser pour d'autres cas, comme pour afficher tous les sprites du jeu en une seule fois par exemple.

V. Liens vers GitHub et installation

- Le code source de ce projet est disponible à l'adresse suivante :
https://github.com/NanroYahel/P3_Labyrinthe

- L'installation du jeu se fait via « pip » avec la commande suivante : **« pip install McGyverLabyrinth »**