P11 / Améliorez un projet

Lien : https://github.com/CoLoDot/PyGame_V2

Le projet 11 du parcours développeur d'application Python/Django a pour but d'améliorer un projet pré-existant. Il s'inscrit dans une mise en situation professionnelle donnant lieu à des échanges avec un client demandant des corrections et des ajouts fonctionnels.

I - Choix du projet

A la lecture de l'énoncée du projet 11, j'ai fait le choix d'améliorer le projet 3 du parcours développeur d'application Python/Django. Le projet 3 était le premier projet permettant de découvrir et d'apprendre le langage Python. Il m'a donc paru pertinent de l'améliorer afin d'établir une véritable différence dans les améliorations à apporter. Par ailleurs, l'amélioration de ce « premier » projet m'a permis de mettre en pratique et d'exposer ma montée en compétences depuis le début du parcours OpenClassrooms.

II - Choix des améliorations

A) FONCTIONNALITÉS

En examinant le projet 3, j'ai constaté plusieurs améliorations potentielles, notamment du point de vue fonctionnel. En premier lieu, il m'a paru plus pertinent d'adopter une approche orientée objet plus forte dans ce projet. Deuxièmement, j'ai constaté le manque d'utilisation des différentes fonctionnalités de la librairie Pygame. Troisièmement, j'ai fait le choix d'ajouter une fonctionnalité permettant de rejouer une partie à la fin d'une autre, ce qui permet à l'utilisateur de ne pas relancer le script à chaque fois qu'il souhaite jouer, ce point s'inscrit donc dans l'amélioration de l'expérience utilisateur. Quatrièmement, l'ajout d'un second labyrinthe m'a paru nécessaire afin, une fois de plus, d'améliorer l'expérience utilisateur. Enfin, l'ajout de sons supplémentaires s'est révélé nécessaire afin de rendre le gameplay plus agréable.

B) TESTS

Les améliorations du projet 3 se révélant potentiellement nombreuses à la vue des scripts initiaux, j'ai fait le choix d'ajouter des tests, absents dans la première version. J'ai donc ajouté une quinzaine de tests permettant de vérifier que les variables globales du projet ne sont pas corrompues. Les tests permettent également de vérifier l'intégrité de la création des sprites (surfaces visibles par l'utilisateurs.)

III - Améliorations

A) FONCTIONNELLES

Le projet initial avait déjà une approche orientée objet, cependant, étant donné qu'il s'agissait du premier projet python réalisé, on y constate une POO assez superficielle par rapport aux possibilités qu'offre la POO. Ainsi, j'ai amélioré la POO en créant une classe MazeCreation plus succincte (une seule méthode de classe pour générer le labyrinthe) et basée sur l'utilisation de la classe Sprite de la librairie Pygame. Il

en va de même pour la création des classes PickSafe, End, Floor, PickPotion, PickCoin et Walls qui visent à générer les surfaces liées au éléments présents dans la variable maze (script constants.py).

Par ailleurs, dans la première version, les scripts n'exploitaient pas réellement la librairie Pygame. De fait, le recours à la classe Sprite était absent alors qu'il permet de générer en deux lignes un élément Sprite visible à l'écran. De plus, l'usage des fonctionnalités liées aux collisions entre Sprite n'étaient pas utilisé. J'ai donc implémenté cette fonctionnalité, notamment dans la boucle principale du jeu (mainloop.py) afin de gérer les collisions entre le joueur et, par exemple, les objets à collecter pour gagner la partie.

Egalement absente de la première version, la fonctionnalité permettant de rejouer une partie à la fin d'une première a été ajouté, elle déclenche l'affichage d'un bouton (winner ou looser) selon le score du joueur et relance la boucle principale du jeu lorsqu'un event MousseKeyDown de Pygame est détecté.

Toujours dans une optique d'amélioration de l'expérience utilisateur, j'ai fait le choix d'ajouter un second labyrinthe, générer de manière aléatoire par le biais du module random de python. Cette fonctionnalité peut être améliorée, notamment en créant un système de niveau lors d'une troisième version du jeu.

Enfin, l'ajout de sons supplémentaires s'est révélé intéressant afin de rendre ce petit jeu en deux dimensions plus attractif. Ainsi, la musique de fond du jeu de la première version a été conservé mais le son en a été abaissé afin de ne pas gêner l'utilisateur. Un son lorsqu'un objet est ramassé par le joueur a également été ajouté afin de valoriser la récupération des items présents dans le labyrinthe. Pour finir, deux sons ont été ajouté en fonction du résultat de la partie, lorsque le joueur gagne, un son de fanfare résonne et lorsqu'il perd, une voix désespérée permet d'identifier que la partie est perdue en plus du bouton « looser , play again ? ».

B) TESTS UNITAIRES

Les tests unitaires ont été réalisé avec unittest, la librairie de tests, embarquée, de Python. Ces tests viennent vérifier, entre autre, que les fichiers de création du labyrinthe ne sont pas corrompus, autrement dit, qu'ils contiennent bien les items permettant de créer les sprites. Ainsi, la dernière classe de tests vérifie que les sprites sont biens crées en fonction d'un item présent dans la string de la variable maze_1 (par exemple).

IV - Difficultés

En ce qui concerne les difficultés, une en particulier est apparue lors de l'amélioration de ce projet : les collisions entre sprites. De fait, la librairie Pygame permet de gérer les-dites collisions mais étant totalement néophyte dans ce domaine, il me fut nécessaire de réaliser plusieurs tutoriels avant de parvenir au résultat que je désirais. Cela m'a également permis de découvrir et d'utiliser les groupes de sprites que Pygame peut créer afin de gérer des collisions entre plusieurs items d'un groupe avec un Sprite en particulier (celui du joueur par exemple).