Projet 3 : Save MacGyver from the Maze!

1. Introduction du projet :

Dans le parcours développeur Python sur Open Classroom, il nous est demandé dans le Projet 3, de développer un jeu vidéo 2D mettant en scène Mac Gyver, perdu dans un labyrinthe. Il doit s'en échapper et pour ce faire, Mac Gyver doit récupérer trois objets, (un tube, une aiguille et de l'éther) disposés de manière aléatoire dans le jeu, pour fabriquer une seringue en vue d'endormir le garde et de pouvoir s'évader. Ce document relate mes périples lors de la réalisation de ce jeu.

2. L'algorithme :

1. Introduction

Pour la création du code, je me suis appuyé sur les modules Pygame et Random, je me suis servit dans mon projet, de l'éditeur de texte sublime. J'ais commencé à coder sur l'OS Windows, mais j'ai rapidement rencontré des difficultés et l'aspect peu pratique et fluide d'IDLE ne m'ont pas vraiment aidé, malgré tout j'ai persisté.

2. Le code du jeu

Sous les conseils de mon mentor, j'ai fais le choix de divisé mon code en trois fichiers .py distincts : Un fichier Main qui comporte les fonctions principales (Maze.py) qui initialise la fenêtre du jeu, un fichier (Class.py) qui regroupe les classes et enfin un fichier (Constantes.py) qui définie les variables liées principalement aux éléments graphiques et visuels du jeu. En répartissant ainsi de manière rationnelle et logique mon code en trois parties, il devient plus simple à lire et à modifier pour le futur.

3. Importation de la librairie PYGAME et des fichiers :

Au départ de mon fichier (Maze.py), j'importe la librairie «Pygame», les modules «Constantes.py, Classes.py». Puis ensuite viens l'initialisation du module «PyGame», et de la fenêtre du jeu. La taille de celle-ci découle d'une variable «WINDOW_SIZE», pour chacun de ses cotés que je définie dans le fichier «Constantes.py» comme le résultat de multiplication du nombre de sprites (15) et de leurs tailles en pixels (30). Je reviendrais plus tard sur cette formule pour fixé une valeur fixe en Int de 480 pour la hauteur de la fenêtre afin de laisser une marge noire au dessus.

Dans la fenêtre je me sers ensuite de la fonction «blit», pour afficher un fond visuel de couleur sable (fourni dans le DK-laby), zone que j'étire sur l'ensemble de la zone de jeu.

Au même niveau je situe l'initialisation des visuels comme les murs, personnages et objet avec l'aide de la fonction display. L'emplacement de ces images étant défini dans des variables du fichier. «Constantes.py»

- Les Classes:

J'ai créé 3 classes pour la réalisation de mon jeu :

1. La classe Char :

La classe du personnage de Mac Gyver dispose d'une unique fonction «mooving» qui gère les déplacement du personnage de case en case avec les touches du clavier à l'aide de la fonction «pygame.event.get()». Elle à l'apparence d'une structure conditionnelle en «If» qui vérifie que plusieurs conditions sont remplies avant d'autoriser le mouvement. Elle compare les données en X et Y (self.case x, self.case.y) de Mac Gyver avec les coordonnées des éléments de level.structure pour voir si les nouvelles coordonnées de Mac Gyver, correspondent avec une zone libre (soit un o dans le fichier Level.txt) et non un mur (m). Elle dispose également d'une condition relative à la direction voulue interdisant le déplacement hors de la zone de jeu. Exemple Pour la gauche : if self.case_x < (Nbr_Sprite_Side – 1)

2. La classe Level :

Cette classe a pour fonction principale de définir le parcours du labyrinthe. Pour ce faire j'ai créé deux fonctions «generate» et «display», la fonction generate avec la fonction «with open» va aller puiser dans un fichier «Level.text» dans lequel se trouve le plan du labyrinthe, sous la forme de caractère accès représentants. Les éléments du jeu comme les murs (m), les vides (0) et l'emplacement initial de MacGyver (d) et de Murdoc (a). Je demande à Python de lire ce fichier ligne par ligne, sprite par sprite tout en lui indiquant de procéder à la ligne suivante en bout de ligne (15 caractère par ligne, 16 lignes). Le programme stockera sous forme de liste les éléments puisés dans le fichier Level.txt dans une variable. «structure» La fonction display va prendre le relais pour remplacer les éléments de la variable structure créé précédemment par leur équivalent visuel. Le script va ensuite attribuer des coordonnées X et Y aux éléments en fonction de l'ordre d'apparition des caractères dans la liste «structure» et affichera les images des murs et du gardien dans la fenêtre du jeu sur à ces même coordonnées. C'est la classe qui m'a posé le plus de difficultés à définir. Je me suis aidé d'un tutoriel d'Open Classroom sur le module Pygame (DK Labyrinthe) dont j'ai réussis à décortiqué le code après plusieurs heures d'observations et de tâtonnements.

3. La classe Loot :

*La classe des objets à une fonction display qui va répartir les objets au sein du labyrinthe dans les zones libres de level.structure (les 0), l'aspect aléatoire du process est gérer par la fonction randint du module random qui génère un chiffre entre 0 et 14 pour définir les variable case_x et case_y qui seront transformés en coordonnées dans les variables self.y et self.x avec la formule self.case_y * Sprite_Size. La méthode de ma classe Loot contient également un boléens self.loaded qui est True au début de la boucle est qui devient False lorsque que les coordonnées auto-générées par randint équivalent à celles d'un espace vide dans le niveau.*