Projet 3 : Aidez MacGyver à s'échapper !

Lien du projet sur guthub

- Introduction

Dans le parcours développeru d'application Python il nous est demandé de développer un jeu mettant en difficulté MacGyver. Le jeu se déroule dans un labyrinthe ou le but pour MacGyver est de récupérer les objets nécessaires à la fabrication d'un sédatif permettant d'endormir le gardien du labyrinthe et de se sauver. Les lignes qui suivent vous présentent mon projet.

Choix de l'algorithme

Fichier constants.py

Dans ce fichier je définis les données qui nous seront utiles pour les fichiers **"macgyver.py"** et **"game.py"**. Ce fichier sera appelé en tant que module dans ces fichiers. Je définis notamment la taille des cases et de la bannière en pixel, le nombre de lignes et de colonnes du labyrinthe et la taille de la fenêtre qui sera affichée en fonction de ces données. Je définis aussi la représentation ASCII des différents objets à récupérer par MacGyver. Je crée aussi des variables pour définir les chemins des différents fichiers externes du jeu notamment les images et les musiques.

Fichier macgyver.py : Les classes

classe "Level"

La but de cette classe est de structurer le labyrinthe en fonction des éléments qui lui sont associé. Pour ce faire je lui ai intégré deux méthodes: **"generate"** et **"display"**. Le fichier sera appelé en tant que module dans le fichier **"game.py"**.

La **méthode "generate"** va aller lire dans un fichier externe (lab1) les différents éléments du jeu ligne par ligne puis caractères par caractères qui sont MacGyver (m), les murs (w) les cases vides (o) et le gardien (g) qui seront représentés par 15 lignes de 15 caractères.
Par la suite je demande à la méthode de générer les objets de façon aléatoire grâce à la librairie **"random"** définis au préalables dans le fichiers "constants.py" représentés par "S", "A" et "P" en fonction du caractère que la méthode va lire dans la liste. Si c'est un "o" alors elle peut générer l'objet. J'ai défini un nombre maximum de 3 objets à générer. La variable "self.structure" permet de sauvegarder cette configuration de labyrinthe pour la rappeler plus tard.

Par la suite, la **méthode "display"** va remplacer les caractères par leurs équivalents graphique que je définis dans "constants.py" et directement dans la méthode. La méthode va parcourir chaque lignes et chaque caractères de la variable "structure_map", qui est en fait la méthode "generate", pour la convertir en case possédant un nombre de pixel défini la encore dans "constants.py", puis son équivalent graphique.

Classe "Character"

Dans cette classe je définis les mouvements de MacGyver et l'incidence de ces mouvements sur l'inventaire ainsi que son affichage.

Dans un 1er temps, la méthode "move" va définir l'incidence sur la structure du labyrinthe en fonction du choix de l'utilisateur d'aller en haut, à gauche, en bas ou à droite en l'occurrence l'affichage de MacGyver sur telle case de telle ligne représentée par les variables "x" et "y".

La **méthode "reset"** va quand à elle remettre à zéro (ou plutôt "o") une case. La méthode sera appelée lorsque macgyver se déplacera sur un objet.

La **méthode "collect_inventory"** permettra d'augmenter l'inventaire de MacGyver. La méthode sera elle aussi appelée lorsque MacGyver se déplacera sur un objet.

Enfin, la **méthode "display_inventory"** permettra d'afficher le nombre d'objet restant à récupérer par MacGyver.

Fichier game.py : Les boucles du jeu

Dans ce fichier je commence par importer la librairie du module P**ygame** nécessaire à l'affichage de la fenêtre du jeu, des images, des musiques et la prise en compte des actions des touches du clavier. Puis les les fichiers **"constants.py"** et **"macgyver.py"**

Je commence par définir la taille de la fenêtre, puis les variables pour l'affichage des images du jeu et les musiques. Puis je commence ma première boucle : la boucle "main_screen" pour l'écran principal du jeu. Dans cette boucle le programme lance la musique et l'image principale du jeu. La boucle "load_game" se lance si l'utilisateur appuie sur la touche requise et cela lance l'affichage du labyrinthe. La boucle "continue_game" prend le relaie et permet le déplacement du joueur en fonction de la touche de déplacement choisie par l'utilisateur. Pour le ramassage des objets, le programme va comparer les données du labyrinthe avec celle de la position de MacGyver et si le déplacement se fait sur un objet, il appellera les méthodes **reset** et **collect_inventory** déjà citées précédemment.
Pour déterminer si le joueur gagne ou perd, nous utilisons des booléens. Si le joueur possède les 3 objets en se déplaçant sur le gardien alors c'est la boucle "win" qui se lance et affiche la fenêtre de la victoire, sinon c'est la fenêtre de défaite.

- Difficultés rencontrées et les solutions trouvées

J'ai commencé à coder pour sur ce projet en utilisant **Sublime Text**, puis, sur les conseils de mon mentor, avec Atom. Cela m'a permis de lancer mon code directement dans **Atom** grâce à l'une de ses extensions. Étant novice dans la programmation informatique, j'ai passé un temps très important à me documenter et tester mon code qui était loin d'être fonctionnelle. Heureusement, on trouve beaucoup de tutoriels sur internet notamment le DK labyrinthe sur Openclassrooms qui m'a beaucoup aidé.
J'ai fais le choix au début du projet de ne pas utiliser l'interface graphique **Pygame** et de coder de façon "brut" mon labyrinthe, mais je n'avançait pas assez vite à mon goût et je commençais à me décourager. Les idées étaient là, mais j'avais du mal à les mettre en place. Incorporer **Pygame** m'a aidé à mieux comprendre comment coder.
Je pense avoir manqué d'organisation au début de projet notamment au niveau du pseudo code qui n'était pas très clair ou mal organisé dans la page. Je pense c'est un point ou je peux largement m'organiser.