Serein MEIDOM

PROJET 3 : AIDER MACGYVER A S'ECHAPPER

Le projet 3 intitulé "MAVGYVER" permet d’apprendre et de maitriser le langage python.

I- Définititon de l'algorithme du jeu

La réalisation de ce projet s'est déroulée en trois étapes. Nous avons rencontré avant le mois d'août quelques difficultés dues d'une part à mon emploi salarié et d'autre part à la dégradation de mon premier ordinateur.

1) La définition de l'algorithme du jeu et des interactions entre les personnages

Etape la plus longue du processus, celle ci m'a permis d'optimiser ma maîtrise du langage python. Plus précisément des boucles, des conditions et le développement orienté objet.

Quelques difficultés ont été rencontrées lors de cette étape, notamment au niveau de l'édition du labyrinthe et de la méthode permettant le déplacement du personnage.

2) La définition des méthodes de lancement du jeu et l'implémentation du module pygame

Etape intermédiaire du projet, celle ci m'a permis de maîtriser l'installation et l'utilisation des modules notamment **Pygame**.
Quelques difficultés se sont présentées lors de l'adaptation du module au code.

3) L'adaptation du code selon les règles de la PEP 8 et son exécution

Aucune difficulté particulière ne s'est présentée durant cette phase.
Quelques irrégularités au niveau des normes ont toutefois été noté dans l'utilisation de **Pylint**.
Concernant l'exécution du jeu, je n'ai pu rendre le jeu exécutable grâce à l'installation de **cx freeze**.
Cependant, en ayant recours à la fonction **find_data_file()** stockée dans le document unfroz.py j'ai pu exécuter le jeu via cx_freeze.

II- Aspects techniques du jeu

1- Le fichier constants.py:

Le fichier regroupe les différentes contantes utilisées par le jeu.

2- Le fichier labyrinth.py:

Ce fichier permet de créer la structure générale du labyrinthe. Il abrite la classe **Level** dans laquelle nous créons un dictionnaire de coordonnées représentant les différents éléments du labyrinthe que nous souhaitons afficher. Grâce à la fonction **read_coordinates()**, nous définissons un dictionnaire qui permet de récupérer les valeurs du labyrinthe.
Nous créons une ligne et pour chaque ligne dans le labyrinthe nous ajoutons une sprite, ainsi de suite.

Ensuite grâce à la fonction **randomize_obj()**, nous disposons aléatoirement trois objects dans le labyrinthe. Nous créons une liste renfermant les objets, en attribuant une valeur aléatoire à leur valeur(pos_x, pos_y) en utilisant la fonction **randint()** du module random.

Nous établissons une boucle dans laquelle on récupère la position des objets.

Enfin nous affichons le labyrinthe grâce à la fonction **display()** en implémentant pygame, notamment la fonction blit() et les images stockées dans le fichier constants.py.

On crée une liste laby_map pour récupérer le labyrinthe qui est représenté par une liste. Ensuite on récupère les différents niveaux du labyrinthe grace à level("").

Une boucle 'for' nous permet ensuite d'afficher les personnages et objets du labyrinthe en multipliant la taille de chaque éléments par la taille de la sprite (coord * SPRITE_SIDE).

Par ailleurs, nous affichons le score grâce à display().

3- Le fichier macgyver:

Le fichier **macgyver.py** nous permet de déplacer le personnage principal dans le labyrinthe. On initialise la position de **Macgyver** dans le labyrinthe. On crée un panier (bucket) dans lequel **Macgyver** collectera les objets. Ce panier permet d'obtenir le score grâce à la fonction **len()**.
Self.finish représente la fin du jeu et **self.result** représente la victoire ou la défaite du personnage.

La fonction **move()** permet d'appliquer le mouvement du personnage. Elle définit la condition selon laquelle le personnage pourra se déplacer au sein du labyrinthe.

On définit la direction future du personnage dans un dictionnaire par une clé, qui prend en valeur un sous dictionnaire qui comprend en clé la position future du personnage et la condition qui nous permet de nous assurer que le personnage reste dans le labyrinthe ("is_out_laby" > 0 si la direction est gauche ("q") ou en haut("z"), "is_ou_laby" < 14 si la direction est à droite ("d") ou en bas("s")).

Si la direction du personnage in ("q", "z", "d", "s") on exécute le mouvement du personnage(apply_move).

Le mouvement du personnage est définit par la fonction **apply_move()** qui prend en argument, la direction, le labyrinthe, la position future et l'indication s'il est dans le labyrinthe ou non("is_out_laby"). Si la position future de Macgyver n'est pas égale à celle du mur il peut avancer. S'il passe sur un objets il prend la place de l'objet. Chaque objet est stocké dans **bucket** grâce à la fonction **stock_quest_item()**.

Pour chaque objet stocké on augmente le score de **+1**.

Dès lors que le personnage prend la place du guardian, nous exécutons la fonction **check_victory_or_defeat()**.
Cette dernière permet de terminer le jeu avec **self.finish = True** et de dire si le personnage a gagné (self.result = True) ou perdu (self.result = False).

Si le nombre d'objet dans bucket est égale à la longueur de la liste d'objet, le personnage gagne sinon il perd la partie.
Grâce au module Font de pygame et aux images définis dans constants.py on affiche le message de victoire ou de défaite.

4- Le fichier main:

Le fichier **main.py** permet d'exécuter l'algorithme du jeu avec pygame.

Elle affiche le labyrinthe grâce à la fonction **display()** et exécute le mouvement du personnage en implémentant pygame.

Si l'utilisateur clique sur la croix ou appuie sur la touche "esc" il quitte le jeu.

S'il appuie sur les touches left(à gauche), right (à droite), up(en haut) ou down (en bas) le personnage bouge conformément à la direction indiquée grâce à la fonction **move()**.

Si Macgyver finit le jeu **(mcgyver.finish)**, on vérifie s'il a gagné ou non, dès lors il peut recommencer ou quitter le jeu.

5- Le fichier LaunchGame:

Le fichier **launchgame.py** permet de lancer le jeu.
Grâce au module Pygame nous définissons les éléments permettant de configurer la fenêtre de jeu.

Nous créons une boucle principale infinie qui permet de démarrer le jeu et d'afficher le menu. Ensuite nous générons une sous boucle à l'intérieur de la boucle principale afin de permettre à l'utilisateur de sélectionner le labyrinthe dans lequel il souhaite évoluer.
Les différents niveaux de labyrinthes sont stockés dans un dictionnaire.

Une deuxième sous boucle permet à l'utilisateur de rejouer lorsqu'il achève un niveau ou de quitter le jeu grâce à main() et la condition mcgyver.finish.