Démarche du Projet 3: Aidez MacGyver à s'échapper !

Ce document à pour but d'expliquer la démarche que j'ai utilisé pour mettre en œuvre le projet 3 de la formation OpenClassRooms développeur d'application PYTHON.
Le lien redirigeant vers mon code source est : https://github.com/micktymoon/oc_projet3.git

Après avoir lu la mission, j'ai décidé de créer une ébauche des classes que je voulais utiliser pour le labyrinthe, ce qu'elles pouvaient faire et ce qu'elles contenaient.
Puis j'ai été lire les documentations de Pygame pour comprendre l'interface graphique et sont fonctionnement. J'ai commencé par essayer d'afficher une fenêtre, puis d'afficher une image, de la faire bouger, d'en afficher plusieurs les unes à côté des autres.

J'ai ensuite réfléchis à comment créer un labyrinthe avec. Dans un premier temps il me fallait un écran, et qu'il soit un carré de 15 sprites de large. Mais qu'est ce qu'un sprite? Internet m'a aidé à trouver la réponse. J'ai essayé d'afficher uniquement le sprite du mur que je voulais. Car la map des murs proposée est composée de plusieurs sprites de sols différents, de 20x20 de dimension. Après avoir réussis, j'ai affiché les contours du labyrinthe. J'ai répété l'action pour créer les murs du labyrinthe. J'ai ensuite positionné le Départ et l'Arrivée ainsi que le héro et le gardien. Mais les images des personnages et des objets étaient plus grandes que 20x20, alors me voilà repartis pour des recherches sur internet. Celles-ci m'ont permit de trouver le modul dont j'avais besoin. Puis j'ai trouvé un moyen d'afficher les objets de manière aléatoire, sans qu'ils ne soient sur les murs. Pour cela j'ai utilisé une liste des murs, et si l'image de l'objet se positionnait sur un mur de la liste, l'objet devait alors se repositionner ailleurs et redessiner le mur effacé.
J'avais donc mon labyrinthe, il ne me restait plus qu'à faire bouger mon héro. J'ai donc créer dans ma fonction 'main', une gestion des événements, si on clique sur 'QUIT', si on presse une des flèches du clavier cela bouge le héro. Mais celui-ci traversait les murs (en les effaçant au passage) et une image du héro restait au point de départ.

Après avoir montré tout cela à mon mentor, il m'a expliqué qu'il fallait que je crée un labyrinthe manuellement avec une liste, représentant un tableau, contenant des listes correspondant aux lignes du tableau, chaque éléments de ces listes correspondent aux colonnes. Et si possible qu'il soit contenue dans un fichier à part, pour pouvoir le générer à partir du fichier. Il m'a aussi expliqué qu'il fallait, qu' à chaque évènements, je redessine le labyrinthe.

J'ai donc décidé de dessiner mon labyrinthe d'abord sur le papier, pour savoir comment l'écrire ensuite dans un fichier. Une lettre étant égale à un sprite, il me fallait un carré de 15 lettres de large. Une fois finis, je l'ai réécris dans un fichier. Chaque lettre de la configuration du labyrinthe correspond à un élément, 'm' pour un mur, 'x' pour un vide, 'D' pour le départ, 'A' pour l'arrivée, 'P' pour le héro et 'G' pour le gardien.
Puis j'ai créé les classes que j'avais écris dans ma première ébauche :

- La classe 'Lab' : ayant pour attributs l'écran et le fichier a préciser en paramètre, une liste des murs, une liste des places vides, la configuration du labyrinthe et l'image de la map des murs. Cette classe contient deux fonctions :
  - 'generate_lab' : génère dans une liste, la configuration du labyrinthe à partir du fichier et qui ajoute les rectangles des positions des murs à la liste de mur, de même pour la liste des positions vides.
  - 'display_lab' : affiche le labyrinthe complet. En affichant les sprites, en fonction des lettres auxquels ils correspondent, sur l'écran.
- La classe 'Player' : héritée de la classe 'pygame.sprite.Sprite', ayant pour attributs l'écran, x et y a préciser en paramètre, l'image du héro redimensionnée, le rectangle de cette image, la position du héro et l'état des deux objets.

Cette classe contient plusieurs fonctions :
    - ‘move_right’, ‘move_left’, ‘move_up’ et ‘move_down’: modifient la position du héro.
    - ‘draw_me’ : affiche l’image sur l’écran.
- La classe ‘Labobject’ : ayant le même héritage que ‘Player’, et en attributs, l’écran, l’image de l’objet et la liste des positions vides a préciser en paramètre, l’image redimensionnée, son rectangle et la position, qui est choisit aléatoirement dans la liste. Cette classe ne possède qu’une fonction :
    - ‘draw_me’ : affiche l’image sur l’écran.
- La classe ‘Guardian’ : ayant le même héritage que ‘Player’, et en attributs, l’écran, x et y a préciser en paramètre, l’image redimensionnée, son rectangle et sa position. Cette classe contient deux fonctions :
    - ‘draw_me’ : affiche l’image sur l’écran.
    - ‘my_rect’ : retourne le rectangle.

Une fois mes classes créées, je les ai disposées dans deux fichiers différents, et je me suis intéressée à mon fichier ‘main’ pour tout mettre dans l’ordre.

D’abord l’initialisation du monde. Création de l’écran et du labyrinthe. Puis, pour le héro et le gardien, j’ai créé une fonction ‘create_character’ qui prend en paramètre la configuration du labyrinthe, la lettre du personnage et sa classe, de manière à ce que je puisse créer mes deux personnages de façon simple. Cette fonction, parcourt la configuration du labyrinthe à la recherche de la lettre demandée et crée un personnage de la classe demandée, à cette position multipliée par 20 pour obtenir la bonne position sur l’écran.
Puis j’ai créé une autre fonction ‘erase_pos_character’ qui remplace la lettre dans la configuration, tout deux ajoutés en paramètre, par un ‘x’ . Cela permet d’éviter que l’image du héro ne reste à sa position de départ, malgré son déplacement.
A la suite de cela j’ai créé l’ether, puis j’ai supprimé sa position de la liste des positions vide, pour évité que les deux objets ne choisissent la même position.
Et j’ai utilisé ‘erase_pos_character’ juste après pour évité que la position du héro au départ ne se retrouve dans la liste des positions vides.

Ensuite vient la fonction ‘main’, qui lance le programme. Elle est composée d’une boucle, qui contient une gestion des évènements. D’abord la gestion de la fermeture du jeux, au cas où le joueur clique sur ‘QUIT’. Puis la gestion des contrôles grâce aux flèches du clavier. La gestion des collisions du héro avec les murs, si il y a collision avec les positions de la liste des murs, alors le héro revient à sa position précédente. Les collisions avec les objets, si le héro entre en collision avec l’ether alors l’attribut ‘obj1’ du héro passe de ‘False’ à ‘True’ , de même que si il entre en collision avec l’aiguille l’état de l’attribut ‘obj2’ devient ‘True’. Et enfin la collision avec le gardien, si le héro entre en collision avec le gardien et que ces deux attributs ‘obj1’ et ‘obj2’ ont pour état ‘True’ alors le héro endort le gardien, le message ‘YOU WIN !’ s’affiche sur la console et le jeu se ferme. Sinon il perd, le message ‘YOU LOOSE’ s’affiche et le jeu se ferme.

Après la gestion des évènements nous limitons l’affichage des images par seconde grâce au module ‘time.sleep’ qui prend en paramètre le nombre de seconde. Nous voulons 60 images par seconde, donc le nombre de seconde mis en paramètre est un soixantième.
Ensuite nous affichons le labyrinthe avec la fonction ‘display_lab’. Puis l’ether si ‘obj1’ est ‘False et l’aiguille si ‘obj2’ est ‘False’, grâce aux fonctions ‘draw_me’. Puis le gardien et le héro grâce à leurs fonctions ‘draw_me’ également.

Et pour finir nous donnons comme condition que si l’attribut ‘__name__’ de notre fonction est égale à ‘__main__’ alors nous appelons la fonction ‘main()’.