Rapport du projet P3 - Labyrinthe

Ce projet consitait à réaliser une interface graphique représentant un labyrinthe. Dans lequel Mac Gyver devait collecter des objets répartis à travers le niveau. Afin de vaincre le gardien et s'échapper du labyrinthe.
Le programme se contrôlait au clavier, il devait être standalone et ne comportait qu'un seul niveau de jeu.

Difficultées rencontrées

Ayant toujours programmé de façon procédurale, avoir cette approche "objet" était loin d'être évident. Je m'y suis pris à plusieurs reprises afin d'aboutir un résultat satisfaisant. Au début, j'ai commencé simplement à déplacer le code et le mettre dans une classe principale qui faisait tout. Mais ce n'était pas du tout le principe de la POO. Alors j'ai regroupé, non sans mal, certaines fonctions du programme par famille (labyrinte, interface graphique, gameplay)

La réalisation de la version standalone m'a posé le plus problème. En suivant les cours d'OpenClassroom, cx_freeze était l'outil proposé. Mais après de nombreuses tentatives, je n'ai jamais réussi à obtenir un programme fonctionnel. A la place il y a de nombreux plantages et erreurs. Un mentor m'a conseillé d'utiliser le système de wheel, beaucoup plus standard et qui a l'avantage d'être indépendant du système d'exploitation.

Réalisation du projet

Afficher les élements du labyrinthe
La première chose à faire était d'afficher les éléments dans la fenêtre de jeu. On appelle aussi ces éléments des sprites. Pour afficher un sprite à l'écran, on charge dans un premier temps l'image en appelant la méthode load('mon_image') qui va seulement charger l'image en mémoire. Ensuite, l'affichera en faisant via la méthode blit() qui va transférer l'image et la "coller" dans la fenêtre à l'endroit indiqué.

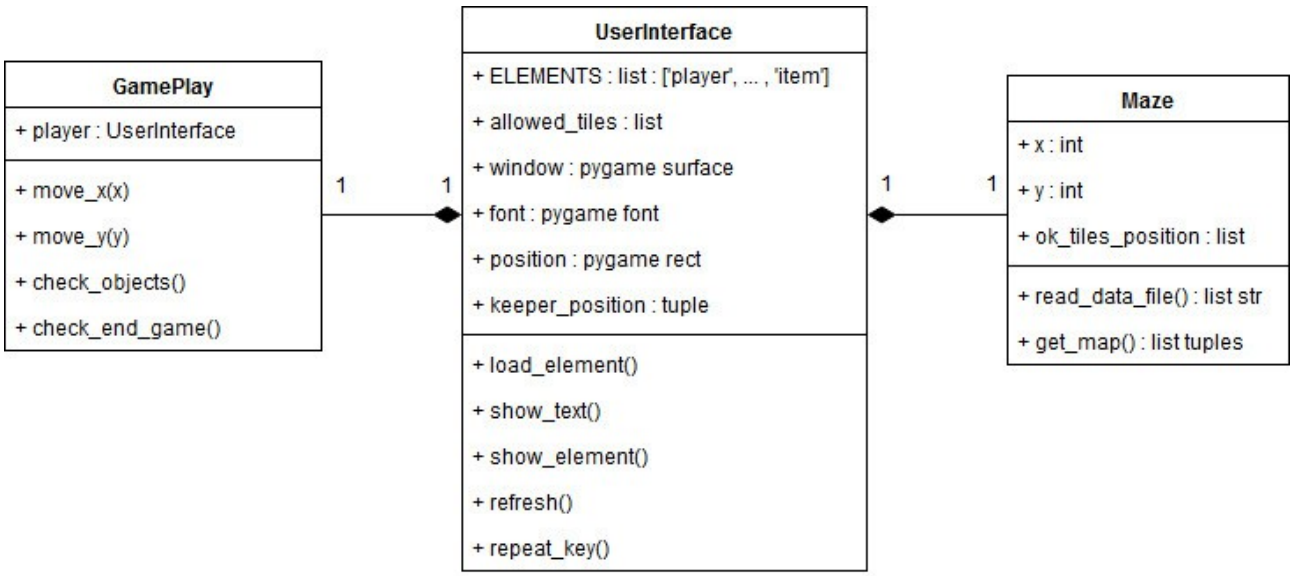
Déplacement du joueur
La deuxième étape consistait à déplacer le sprite du joueur, seul élément mobile du jeu. Le sprite se déplacait soit trop loin ou soit pas assez. Pour qu'il y ait un affichage correct, j'ai défini que pour chaque 'case', elles avaient pour dimension 40 pixel par 40. Ainsi lorsque le joueur se déplace, le sprite se déplace de 40 pixels en ordonnée ou en abscisse.

Système de collisions
Certainement la partie la plus intéressante à mes yeux, il fallait faire en sorte que le joueur ne passe pas à travers les murs, ni même ne sort hors de la fenêtre. Pour répondre à cette problématique, le programme va définir une liste de cases, dans lesquelles le joueur est autorisé à se déplacer.
Lors de son déplacement, le programme va vérifier que la case sur laquelle il veut aller est bien présente dans la liste. Si ce n'est pas le cas, alors le programme modifie la valeur du déplacement à 0 pixel. Donnant ainsi, l'illusion que le joueur reste immobile.

Afficher les murs
Une fois le système de collision mis en place, j'ai défini le niveau du jeu dans un simple fichier texte. Un X représentant un mur, et un O représentant une case où le joueur est autorisée à se déplacer dessus. Lors du lancement du programme, il va lire le fichier afin de récupérer les cases autorisées puis les stocker sous forme de tuples dans liste.

Génération des objets
Pour ce qui est des éléments à récupérer par le joueur, le programme va simplement piocher au hasard trois cases dans la liste des cases autorisées. Ensuite le programme se chargera de les afficher à l'écran.

Diagramme des classes



La classe UserInterface gère l'affichage des éléments sur l'interface. On remarquera qu'elle possède en attribut une constante éléments qui contient la liste des images à charger au démarrage du programme.

La classe Gameplay gère tout le mécanisme du jeu, à savoir le déplacement du joueur, ainsi que la vérification des objets à récupérer. Lorsque le joueur se présente devant le gardien, la méthode check_end_game() est appelée pour contrôler si le joueur a tous les objets. Si oui, alors il gagne la partie, autrement il perd la partie.

La classe Maze se contente simplement de récupérer le niveau du labyrinthe à partir du fichier data. Puis fournira la liste des cases autorisées sous forme d'une liste de tuples à la classe UserInterface via la méthode get_map()

Liens

Retrouvez l'intégralité du projet via ce lien : https://github.com/Hyperyon/p3-labyrinthe
Vous pouvez l'obtenir le projet en faisant `pip install nz-labyrinth`