Projet 3 : Aidez MacGyver à s'échapper :

Lien vers le code source sur Github : https://github.com/elwaze/MacGyver

Pour ce projet, j'ai commencé par établir les besoins et les contraintes qu'il contenait, et par réaliser une architecture générale de mon programme :

1. Mettre en place un cadre de départ :

   - Mise en place du labyrinthe de 15 * 15 cases à partir d'un fichier extérieur.
   Dans ce fichier extérieur, représentation de la case de départ, des cases mur, des cases couloir, de la case de sortie (gardien). Lecture du fichier extérieur pour créer une liste de listes, ligne par ligne.

   - Génération de MacGyver sur la case de départ.

   - Génération de 3 objets aléatoirement sur des cases couloir.

2. Animation :

   - Créer une boucle qui gère les événements (quitter, appui sur les touches directionnelles), et qui se termine soit quand le joueur quitte, soit quand MacGyver arrive sur la case arrivée.

   - Gérer le déplacement avec les flèches directionnelles du clavier (en restant dans la fenêtre et en restant dans les couloirs).

3. Récupérer les objets :

   - Créer un compteur dobjets initialisé à zéro.

   - Si la position de MacGyver est celle d'un objet, effacer l'objet et ajouter 1 au compteur.

4. Gérer l'issue du jeu :

   - Si la position de MacGyver est celle du gardien (arrivée), si le compteur est à 3, victoire. Sinon, défaite.

Réalisation :

J'ai commencé à écrire mon code "à la volée" en suivant le cours OpenClassrooms sur Pygame (DK Labirynthe) avant de me rendre compte que ce n'était pas efficace, que je me répétais beaucoup. J'ai donc réorganisé tout cela de manière orientée objet.

J'ai d'abord créé une classe pour le labyrinthe, une classe pour le joueur (MacGyver), et une classe pour les objets à collecter. Je me suis rendu compte que les deux dernières classes présentaient beaucoup de similitudes. J'en ai donc fait deux sous-classes héritant d'une super-classe items. J'ai enfin créé une classe Runner pour le déroulement du jeu, qui initialise les différents composants et contient une méthode run qui contient la boucle de déroulement du jeu, une méthode handle_moove qui gère les déplacements de MacGyver, et une méthode arrival qui gère l'issue du jeu.

Une fois mon programme terminé, j'ai vérifié en ligne qu'il respectait bien la PEP8.

En dernier lieu, j'ai utilisé cx_Freeze pour rendre mon programme standalone, et j'ai proposé à quelques amis de le tester, pour vérifier qu'il fonctionnait bien en standalone et bénéficier de leurs avis et remarques.

Difficultés rencontrées :

- Dans mon compte des objets, à chaque fois que MacGyver repassait sur la position d'un objet déjà ramassé, le compteur s'incrémentait. Solution : Plutôt que de compter les objets, j'ai décidé de vérifier s'ils étaient toujours présents dans le labyrinthe.

- Je n'avais pas pensé à la possibilité de générer deux objets sur la même case. Solution : après génération d'un objet, modifier le "0" en "x" dans la liste représentant le labyrinthe, à l'emplacement de l'objet.

- Commits Git : n'ayant pas créé le fichier .gitignore dès le démarrage, je continuais à commiter des fichiers précédemment commités malgré leur inscription dans .gitignore. Solution : pour les supprimer du repository, effectuer une commande git rm.

- Gestion standalone du programme : au départ, en suivant des tutoriels en ligne, je n'arrivais pas à faire fonctionner mon programme à l'aide du build créé avec cx_Freeze. Il a fallu que je comprenne que je devais ajouter la liste de tous les fichiers contenus dans mon projet et pas seulement le fichier principal (fichier contenant la modélisation du labyrinthe, images...). J'ai aussi du télécharger une police d'écriture pour le message de victoire ou défaite à l'issue du jeu.

Propositions d'amélioration :

- Afficher la seringue après avoir récolté les 3 objets, pour indiquer au joueur qu'il peut se rendre vers la sortie.

- Proposition à l'issue du jeu de rejouer ou de quitter.

- Création de plusieurs niveaux.

- Enregistrement des scores.

- Ajouer du son.