Openclassrooms DA Python Projet 11

Améliorez un projet existant en Python

Elève : Baptiste Fina

Lien GitHub : https://github.com/Palombredun/Projet11

Introduction :

Ce projet consiste à reprendre un projet précédent de la formation, ou un projet personnel existant, et à le modifier de manière à ce qu'il comporte des tests unitaires pour s'assurer de son bon fonctionnement, ainsi qu'une fonctionnalité supplémentaire demandée par un pseudo-client.

Mon choix s'est porté sur le projet 3 « Aidez MacGyver à s'échapper ». C'est le seul projet à ne pas comporter de tests unitaires allié au fait qu'il ne soit pas lié au développement web. C'est une composante déjà très forte de la formation, et j'ai donc voulu profiter de ce projet pour travailler un autre aspect du développement avec Python.

J'ai donc choisi de générer de manière aléatoire un labyrinthe, grâce à l'algorithme « recursive backtracker » détaillé ici :
https://en.wikipedia.org/wiki/Maze_generation_algorithm#Recursive_backtracker

Première partie : choix de la méthode de travail

Pour ce projet, j'ai choisi de travailler en TDD. Le projet 7 de la formation l'imposant déjà, mais étant donné la quantité d'informations à emmagasiner, j'avais trouvé la tâche assez difficile. C'était donc un premier pas, mais peu concluant.

Ce projet-ci m'a permis, maîtrisant bien mieux le sujet, de correctement faire de la TDD, d'autant plus que les demandes du pseudo-client d'avoir des tests unitaires s'y prêtaient bien.

Ensuite, j'ai voulu adopter le pattern MVC (Model View Controller) afin de bien séparer chaque fonctionnalité du programme.

Deuxième partie : choix de la structure de données

Afin de stocker en mémoire la structure du labyrinthe généré par l'algorithme « recursive backtracker », j'ai utilisé un arbre composé de nodes à quatre branches. J'ai fait ce choix pour plusieurs raisons. La première étant que sur hackerrank j'avais fait quelques exercices sur les arbres, représenter un labyrinthe par un arbre me semblait assez naturel. Ensuite, l'algorithme utilisé ne compte pas les murs comme des cases mais comme une simple séparation entre les différentes cases de chemin.

J'ai donc trouvé plus simple, au début, de choisir cette structure de donnée.

Les difficultés sont apparues au moment de l'affichage du labyrinthe. Les nodes composant l'arbre étant munies de quatre branches, un parcours classique de l'arbre (par récursion), posait vite problème. En effet, Python, pour éviter les erreurs de stack overflow limite la profondeur des récursions. J'aurais pu augmenter cette limite jusqu'à ce que le parcours de l'arbre puisse se faire,

mais je n'ai pas choisi cette solution. D'une part car cela peut poser des problèmes de mémoire consommée (et donc faire planter l'ordinateur), et ensuite car si on souhaite augmenter la taille du labyrinthe, il faut aussi conséquemment augmenter manuellement la valeur de la profondeur de la récursion.

J'ai donc choisi une autre solution. Au moment de générer le labyrinthe, j'ai stocké dans une liste les chemins des feuilles de l'arbre sous forme de chaîne de caractère. Ainsi : « lldrud » signifie : depuis la racine, emprunter les branches left, left, down, right, up, et down. Mimant ainsi le déplacement dans l'arbre et sur l'écran du personnage. Je me suis inspiré de l'algorithme https://en.wikipedia.org/wiki/Huffman_coding qui consiste à encoder une chaîne de caractère dans un arbre binaire selon leur position. On peut ensuite facilement reconstruire le mot en suivant la chaîne de caractère produite après l'encodage.

Cette méthode a l'avantage d'éviter toute récursion, mais est un peu plus coûteuse en temps de calcul. Cela ne se ressent évidemment pas sur un arbre aussi petit.

Troisième partie : l'affichage du jeu

L'affichage du labyrinthe, comme évoqué dans la partie précédente, se fait en parcourant la liste des chemins des feuilles de l'arbre. Le programme suit simplement chaque chemin dans l'arbre et renseigne pour chaque case l'image à afficher.

Le héros est systématiquement placé à la racine de l'arbre, l'ennemi et les objets sont placés sur des feuilles, afin d'assurer que toutes les parties puissent être gagnées.

Conclusion :

Ce projet a pour moi été l'occasion de refaire de la TDD mais dans cette fois dans de très bonnes conditions, et bien que cela nécessite un peu plus de réflexion et un peu de moins flexibilité (ou de laxisme) durant la rédaction du code, cela permet aussi de s'assurer que les fonctions écrites fonctionnement comme attendues.

J'ai beaucoup apprécié de pouvoir coder un projet n'impliquant aucun framework, pour simplement me concentrer sur l'algorithmique et la qualité du code (certainement perfectible, mais aussi certainement améliorée par rapport au projet 3).