Programme Mac Gyver Maze, document explicatif

SÉBASTIEN JOURDAN

Lien vers le projet Mac Gyver :

https://github.com/sebajou/MacGyver_labyrinthe

Choix de l'algorithme

L'algorithme peu se scindé en deux parties, la partie console, où l'on peut jouer aux jeux dans la console, et la partie graphique, où l'on joue aux jeux dans un environnement graphique et sonore grâce à PyGame.

Initialement, l'algorithme ne contenait pas de mode graphique. Il a d'abord été développé de manière procédurale, puis refactoré avec des fonctions et enfin les fonctions ont été regroupées et réarrangées pour fonctionner dans des classes, avec quelquefois un héritage entre les classes. Une couche PyGame, a alors été réalisée en rajoutant des modules contenant des classes utilisant PyGame.

Pour la partie console, le labyrinthe est chargé dans une table à partir d'un fichier texte où des lettres et des symboles représentent le labyrinthe. Un autre autre labyrinthe peut être réalisé en remplaçant le fichier texte, ou en modifiant le chemin vers le fichier texte dans le code.

Le déplacement de Mac Gyver consiste à enregistrer la commande de déplacement de Mac Gyver et prédire si le déplacement est possible (si Mac Gyver ne va pas dans le mur).

Les objets (que j'ai nommé artefacts dans le jeu), sont effacés du labyrinthe une fois que Mac Gyver passe dessus. Un conteur enregistre alors le nombre d'artefacts collecté.

La victoire, ou la mort (atroce), de Mac Gyver survient lorsqu'il se trouve à la position du gardien. Le gardien n'est pas considéré comme un objet. Dans ce programme c'est juste une position dans le labyrinthe où est affiché son (affreux) visage.

Le module « build_maze » utilise pygame pour charger le labyrinthe en mode graphique. Pour chaque symbole dans la table contenant le labyrinthe, le module, créez une surface et charge une image correspondante. Chaque case contient une image (de mur, de sol, d'artefacts ou de personnages).

Le module « sound_music_and_message » se charge de jouer des sons et envoyer des messages à l'écran lorsqu'il est appelé par d'autres modules.

Difficulté rencontrée et solutions trouvées

Déplacer Mac Gyver m’a demandé de la réflexion. Ma solution consiste à prédire si le déplacement doit être exécuté ou non. Cette solution a le défaut de ne pas être évolutive. En effet, si je veux rendre le déplacement plus rapide, cette partie de l’algorithme devra être entièrement revue. Un déplacement à l’aide du module PyGame est peut-être plus souple.

Pendant le développement de ce programme, j’ai été amené à réaliser des héritages. L’utilisation de l’héritage que j’ai fait dans ce jeu est bien en dessous du potentiel de ce paradigme.

Pour chercher l’inspiration, j’ai été amené à lire des jeux faits sous PyGame par d’autres codeurs. Il m’a paru très difficile de lire et comprendre le code fait par un autre, surtout s’il manque des commentaires…

Dans ma version de labyrinthe, le gardien est juste une position. C’est la solution la plus simple, et elle est adéquate, car le jeu n’est pas appelé à évoluer (par exemple avec le garde qui se déplace).

La documentation de PyGame est difficile d’accès. Elle est utile, surtout si on a déjà une compréhension du fonctionnement de PyGame et pour voir dans le détail le fonctionnement de chaque objet Pygame.

Enfin ma dernière difficulté a été de freezer le programme. J’ai voulu utiliser cx_Freeze. Je n’ai jamais réussi… Finalement, un requirements.txt suffit.