Parcours Python

Projet 3: Sauvez MacGyver

Auteur: Guillaume Pont
Date: 7 janvier 2018

Github
https://github.com/GuiThink/opc_projet3

Découpage du projet

1. Installation de l'environnement de travail

Afin de préparer le projet j'ai dû commencer par installer mon environnement de développement. J'ai donc eu recours à l'installation de python et d'un environnement virtuel afin de développer le projet.

Un fichier requirements.txt contient les éléments préalables à installer pour faire fonctionner le programme: pygame en version ≥ 1.9.3

Pour coder le jeu, j'ai eu recours à PyCharm puis à Atom.

2. Préparation du projet: les images

Pour préparer le projet, j'ai commencé par récupérer les images du jeu dont j'avais besoin. J'ai donc récupéré les éléments de jeux déjà proposés : MacGyver, méchant et je me suis procuré d'autres images pour constituer le font, les murs, les objets à récupérer et l'image de départ.

J'ai dû convertir ces images en PNG8 car j'ai rencontré lors du développement un problème de compatibilité entre Pygame et des versions PNG non-8 des images : problème d'entrelacement.

3. Réflexion sur la structure du jeu

J'ai décidé de créer deux classes:

- Level = contient les fonctions de génération du niveau, de génération aléatoire des objets, de récupération des objets et de mise à jour des images générées par pygame lors de l'affichage du niveau.

- Hero = contient les fonctions qui permettent de générer le héro, de gérer le déplacement ainsi que l'autorisation où non d'aller vers haut, bas, gauche, droite en fonction des éléments du niveau (mur / voie libre). Pour terminer, la classe Hero contient la fonction 'fight' qui permet de déterminer si le jeu est gagné ou perdu.

Ainsi qu'un fichier Main :

- Main = permet de récupérer les touches tapées par l'utilisateur pour diriger le héro et enfin permet d'avoir la boucle de jeu et de lancer les fonctions nécessaires au démarrage du jeu.

4. Classe Level : Création du niveau

Pour générer le niveau de mon jeu, j'ai dû créer un fichier « level_1 » qui permet de constituer mon niveau avec de simples caractères.

L'avantage est que ce fichier est facilement éditable et permet de rapidement visualiser la structure du niveau.

Dans le fichier, les lettres correspondent à des éléments du niveau :

s = image de départ
# = mur
‘ ‘ = voie libre pour se déplacer
e = méchant (fin du niveau)
; = fin d’une ligne

4. Génération du niveau

La première étape est de générer le niveau avec Pygame.
Pour cela j’ai dû créer une fonction qui récupère le fichier « level_1 », et génère en python une liste qui contient d’autres listes (1 par ligne) avec les caractères contenus dans « level_1 ».
C’est grâce à cette liste qu’il est ensuite possible de générer visuellement le niveau grâce à Pygame.

Après la génération de base du niveau, on génère aussi les 3 objets à récupérer par le héro. Ces objets sont générés de façon aléatoire dans le niveau grâce à l’emploi du module random. Par ailleurs, il y a une vérification des cellules disponibles (cellules vides) afin de déterminer sur quelles positions les objets peuvent potentiellement s’afficher.

A la suite de cela, on gère aussi l’affichage du héro dans le niveau avec comme position par défaut le coin supérieur gauche du jeu: x = 0; y = 0.

5. Gestion des déplacements

En fonction des touches tapées par le joueur, une fonction vérifie que le héro peut être déplacer dans la direction indiquée par le joueur. On vérifie les nouvelles coordonnées x et y et on déclenche une fonction de vérification pour vérifier que la case est vide.
Si la case est bien vide, alors on déclenche une fonction qui va enfin attribuer les nouvelles coordonnées x et y au héro afin que celui-ci soit bien affiché sur la nouvelle case du niveau.
A la suite de cela, on déclenche une fonction clear cell qui va permettre d’effacer l’image du héro de la case précédente. Cette fonction clear cell à permis de régler un problème de persistance de l’image de MacGyver sur toutes les cases où il se déplaçait. On se retrouvait donc avec des dizaines de MacGyver à l’écran, ce qui n’était pas normal car MacGyver est unique.

6. Gestion des loots

Lorsque le héro est sur une case, on déclenche systématiquement une fonction qui va vérifier si ses coordonnées x et y sont égales aux coordonnées x et y d’un loot. Si c’est le cas alors on incrémente un compteur (qui est en fait une liste) et on efface l’image du loot afin qu’il disparaisse du niveau.
Cette liste est importante car pour gagner, MacGyver doit avoir 3 loots. Lorsque la longueur de la liste est de 3, alors MacGyver pourra gagner. Si la longueur est inférieure à 3, alors il perdra s’il se présente devant le gardien.

7. Gestion du combat

La fonction fight vérifie que le héro se trouve sur la dernière case du jeu, celle où se trouve le méchant (‘e‘). Si c’est le cas, alors on déclenche le combat. Le combat consiste à vérifier que la liste qui contient les loot est d’une longueur de 3 (ce qui veut dire que les 3 objets sont détenus par

le héro). Si c’est le cas, le jeu s’arrête et MacGyver est vainqueur. Sinon, il meurt, perd et le jeu s’arrête.

8. Validation PEP8 avec Pylint

J’ai dû procéder à la validation PEP8 du projet avec des partis pris : en effet certains problèmes remontés comme non-compatibles PEP8 n’étaient pas forcément pertinents. Par exemple, les coordonnées x et y ne sont pas des variables dites ‘valides PEP8’ sauf que ces variables étant des coordonnées elles sont tout de même suffisamment parlantes et compréhensibles. J’ai donc décidé de les laisser en l’état.