PARCOURS OPENCLASSROOMS DÉVELOPPEUR D'APPLICATIONS PYTHON

QUENTIN LATHIÈRE

Projet 3: Aidez MacGyver à s'échapper !

https://github.com/Synkied/OC_Projet-3

Le présent document résume la réalisation du 3e projet (Aidez MacGyver à s'échapper !) dans le cadre de mon parcours Développeur d'application - Python.
Ce document recense les grandes lignes de la réalisation du projet. La réflexion pré-projet, la réalisation du projet et les réflexions post-projet.

Définition du projet

Description

McGyver est coincé dans un labyrinthe, dont la sortie est gardée ! Il doit récupérer 3 objets et les assembler afin d'endormir le garde et s'échapper !

Fonctionnalités

- Utiliser Python 3 ;
- Créer un labyrinthe sur lequel sont présents des objets placés aléatoirement à chaque lancement du jeu, un départ (MacGyver) et une arrivée (Gardien) ;
- Utiliser la librairie Pygame pour afficher les éléments graphiques du jeu ;
- Fenêtre de jeu de 15 sprites sur la longueur ; Déplacement du héros de case en case ;
- Le programme s'arrête si le héros a trouvé et récupéré tous les objets, et qu'il a pu rejoindre la sortie ou s'il rejoint la sortie sans tous les objets, et meurt ;
- Programme standalone, exécutable sur n'importe quel ordinateur.

Contraintes

- Versionner son code en utilisant Git et le publier sur Github ;
- Développer dans un environnement virtuel en utilisant Python 3 ;
- Respecter les bonnes pratiques de la PEP 8 ;
- Code écrit en anglais : nom des variables, commentaires, fonctions...

Pré-projet

Environnement de développement

- Machine sous Windows 10. Environnement virtuel via virtualenv (initialement via Conda) ;
- Sublime Text 3 avec plusieurs plugins (Principaux : Anaconda, GitGutter) ;
- Tests sur un MacBook Pro 2017 sous MacOS X Sierra. Virtualenv
- Tests sur Raspberry Pi 3B ;

Anticipation des problèmes

- Apprendre à utiliser la librairie Pygame (Python + SDL) ;
- Structurer mes dossiers et fichiers de projet ;
- Structurer les classes et leurs interactions ;
- Comment gérer les objets et l'inventaire.

Difficulté rencontrée
- Installer Pygame sur MacOS Sierra via un environnement virtuel **conda**.

Solution apportée
- Installer Pygame sur MacOS X dans un environnement virtuel via **virtualenv**.

Mise en place du projet

Première approche

Pour m'initier à l'apprentissage de Pygame, j'ai suivi le cours disponible sur OpenClassrooms (https://openclassrooms.com/courses/interface-graphique-pygame-pour-python).
J'ai réalisé le premier TP, qui, par chance, était aussi un labyrinthe, mais plus simple.

J'ai aussi lu la documentation de Pygame pour certaines fonctionnalités, telle que la capture d'événement avec « get_pressed » (qui m'a servi pour afficher l'inventaire et la page des contrôles).
Ensuite, sur la base du TP réalisé avec le tuto Pygame d'OC, je me suis lancé dans la réalisation du projet, avec une petite idée en tête de la structure du programme.

Remaniement

Après avoir réalisé un premier jet du jeu calqué sur le modèle du TP PG, je me suis orienté vers une architecture un peu différente, en séparant notamment au mieux les modèles et les vues (MVC).
Étant novice en la matière, je pense qu'il est possible de faire bien mieux. Cependant, je suis assez satisfait par cette dernière structure qui permet d'abstraire beaucoup plus mon programme que lors de mon premier jet.

Choix de format du fichier de labyrinthe et création du labyrinthe:

J'ai choisi le format de fichier « .txt » car c'est un format lisible sur toutes les machines.
Pour créer le labyrinthe, je me suis renseigné sur la façon de créer des algorithmes de création aléatoire avec toujours un départ et une arrivée joignables. Cela m'a paru un peu complexe et être un projet à part entière. J'ai donc choisi de créer le labyrinthe « à la main », en définissant un départ, et la position des murs/sols, en faisant en sorte de ne laisser aucun îlot inaccessible.

Analyse d'algorithme

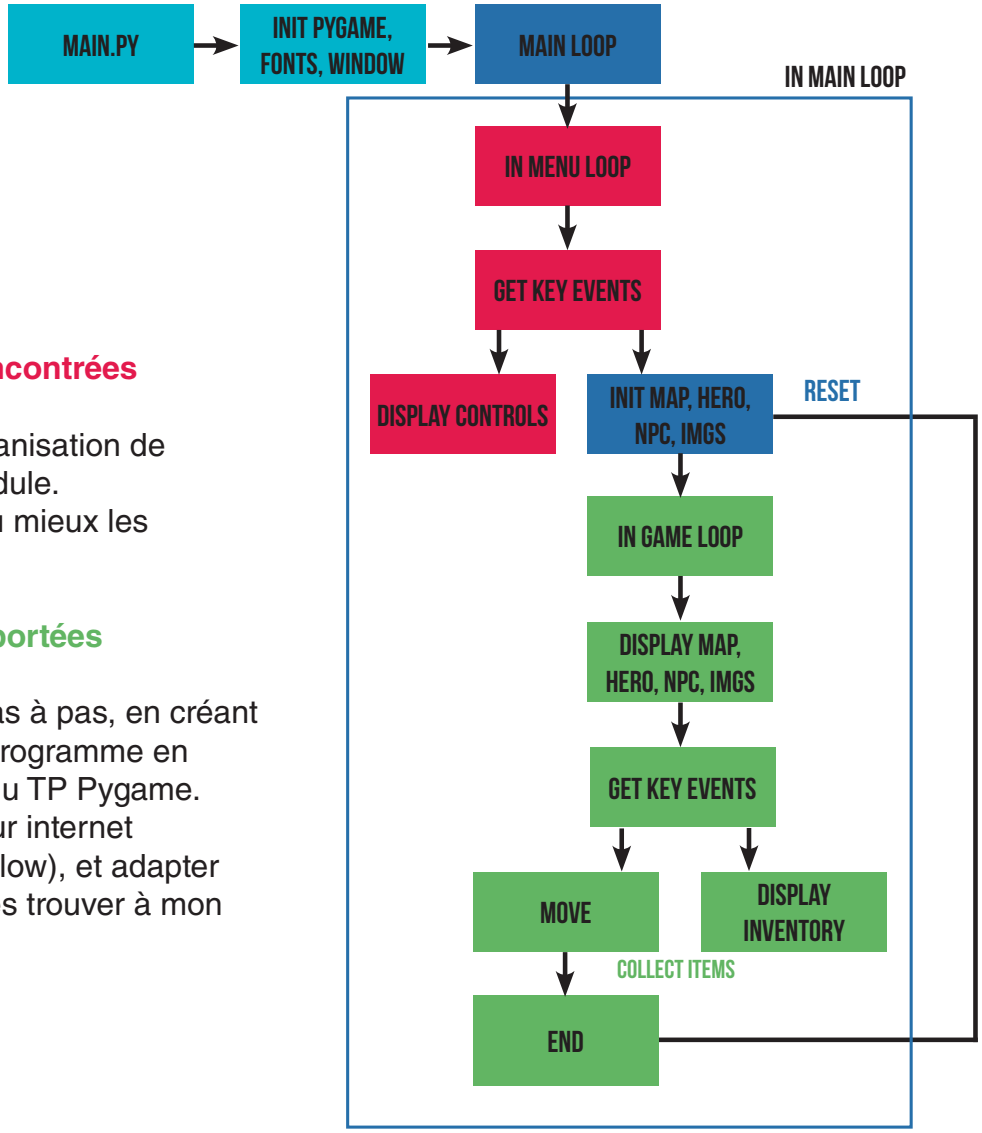
Ramassage des items
L'algorithme de la méthode *collect_items* vérifie à chaque mouvement du héros si celui-ci arrive sur une case avec un item associé affiché, via le booléen « displaying » initialisé à True.
Si c'est effectivement le cas, alors l'objet est ajouté à l'inventaire du héros et le booléen « displaying » passe à False, afin de cacher l'objet récupéré sur la map.
Un message est ensuite passé dans la console afin de dire quel objet a été récupéré et montrer l'inventaire du héros sous forme textuelle.
Difficultés
J'ai eu un peu de mal à bien mettre en place cet algorithme, car au début je n'avais pas rattaché les items au labyrinthe, et je gérais donc chaque instance des items dans le module *main*.
Après avoir rattaché les items à la classe Level, j'ai pu abstraire le ramassage des items et arriver à cet algorithme très concis, qui peut gérer n'importe quel nombre d'items sur la map.

Position random des items et du gardien
L'algorithme consiste en une boucle while qui assigne un nombre aléatoire
entre 0 et NB_SPRITES - 1 à pos_x et pos_y. Tant que la cordonnée formée par ces 2 positions n'est pas un sol, alors la boucle continue. Une fois que la boucle a trouvé une bonne cordonnée, le nom de l'item ou du gardien est placé à l'endroit de la cordonnée dans la map. Cela assure le fait que les items ou le gardien ne se superposent.
Difficultés
Trouver comment mettre le nom de chaque item sur la map, je n'y suis pas arrivé et ai donc défini une constante ITEMS_MAP_NAME.

Modules du projet

core : classes des éléments du jeu (Level, Item, GamePersona, NPC, Character Inventory, Images)
game_settings : constantes du jeu
views : affichage des éléments à l'écran
exceptions : gérer des exceptions personnalisées
main : boucle principale du programme

Exécution du programme



Difficultés rencontrées

- Définir l'organisation de chaque module.
- Abstraire au mieux les classes.

Solutions apportées

- Procéder pas à pas, en créant d'abord le programme en s'inspirant du TP Pygame.
- Chercher sur internet (StackOverflow), et adapter les exemples trouver à mon projet.

Le programme est découpé en plusieurs modules, contenant des classes/méthodes/fonctions.

- Le module *main* contient la fonction principale du jeu (launch_game()), qui permet... de lancer le jeu :). Cette fonction contient la **boucle principale** du jeu, et deux autres boucles, une pour le **menu**, et une pour le **lancement d'une partie**.
- Le module *core* regroupe la quasi-totalité des classes créées pour le jeu. Il y a une classe pour le **niveau**, les **personnages jouables**, les **personnages non jouables**, les **items**, l'**inventaire**, et les **images des éléments** du jeu.
- Le module *view* contient une fonction pour **afficher tous les éléments du niveau**, et une fonction pour **afficher un inventaire.**
- Le module *exceptions* contient une classe (optionnelle) pour une **exception**, afin de lever une exception si le nombre d'items à afficher est plus grand que la taille de l'inventaire.
- Le fichier *game_settings* comprend toutes les **constantes** du jeu.

Conformité PEP 8

Afin de fournir un code propre et conforme aux PEP 8, j'utilise le package « Anaconda » pour Sublime Text, qui fourni un linter PEP 8 et permet de savoir à la volée si le code qu'on écrit est conforme ou non. S'il n'est pas conforme, une indication de la violation est indiquée, et le code PEP 8 associé est affiché. Il suffit alors de réaliser les modifications demandées (et parfois chercher sur internet car le problème n'est pas toujours évident, comme pour les conditions multi-lignes)

Post-projet

Ce que j'ai appris

- Découper des planches de sprite dans Photoshop (oui, oui) ;
- Utiliser la librairie Pygame et par conséquent la SDL ;
- Structurer un projet ;
- Créer un fichier de License pour GitHub ;
- Créer un makefile ;

Les acquis renforcés

- Utiliser GitHub quotidiennement, cela m'a permis de me re-familiariser avec les notions de branche et me rappeler que le versioning, c'est la vie.
- Travailler tous les jours ma logique ;
- Utiliser le langage markdown ;
- Utiliser pip freeze > requirements.txt ;
- Utiliser .gitignore ;
- Utiliser les compréhensions de liste et les 2D arrays.

Améliorations

- Faire un plan UML en pré-projet ;
- Utiliser un design pattern et choisir une architecture en pré-projet pour accélérer le développement ;
- Abstraire + les classes afin de penser à une évolution du jeu ;
- Construction automatique et aléatoire d'un labyrinthe solvable (algorithme + poussé) ;
- Ajouter des tests unitaires.