Soutenance du projet 3: Aidez MacGyver à s'échapper !

Jérémy Chmelkin
(alias 'irmi' sur OC, alias 'irmiaw' sur github)

mentor de projet: Nicolas Resin
mentor de soutenance: Stéphane Nédélec
date de soutenance: 22/09/2017

Code source:
https://github.com/irmiaw/MacGyver

Introduction

Ce document résumé le 3ème projet réalisé dans le cadre de ma formation de développeur d'application Python d'OpenClassrooms. Maintenant que le programme est fini, on peut avoir une vision d'ensemble et réfléchir sur les choix faits durant son développement.
Ce document est en complément du README.md qui détaille les points suivants: contraintes, difficultés rencontrés, choix techniques, modules et historique des versions.

Structure du programme

Ci-dessous un aperçu de l'organisation du programme ainsi que son déroulement.

Modules et leurs Classes ou fonctions:
game: Lvl, Item, Character
display: draw
load: Images, map_from_file, config_from_file
constants: constantes
main: main
config: chemin et nom des images du jeu, objets et dialogues, en format json

Déroulement du programme:
main → init pygame → load config, images, map → init lvl, mac_gyver → display →
main loop → get keyboard event → move character → display → end of game → exit
↑ ↓ ↓
\---------- ← reset ← replay

Problèmes rencontrés:
Trouver comment organiser le programme était facile (et a été anticipé dés le départ), le problème se trouvait dans la mise en pratique. J'aurais du implémenter la structure immédiatement au lieu de commencer a tout écrire dans le main (et en utilisant des variables globales en plus).
C'était une source de nombreux bugs et relectures du code lors du déplacement des fonctions dans leurs modules respectifs.

Fonctions du programme

Le programme est logiquement découpé en sous programmes – les fonctions.

Les grands classiques:
On les rencontre pratiquement dans chaque programme et ils sont faciles a reconnaître.
Les voici regroupées en catégories:

- Chargement des données: load_image, map_from_file, config_from_file
- Parcours d'un tableau 2D avec une double boucle: draw, _initial_position
- Calcul des collisions (4 directions ou dans la position actuelle): free_path, _collect_items
- Parcours des événements: main
- Conversion/interprétation/initialisation des données: pixel_position, status, reset, __init__

Analyse de la fonction random_position:
La fonction random_position a pour but de déterminer aléatoirement une position d'un objet avec comme contrainte de ne pas le placer sur une case non vide (mur, personnage, ou objet déjà placé).
Il y avait plusieurs choix techniques sur comment implémenter le fonction:

- Un algorithme déterministe:
  L'algorithme ne sera pas détaillé ici, mais il impliquait de parcourir la map une fois pour compter le nombre de cases vides et puis une seconde fois pour placer un objet.

- Un algorithme probabiliste:
  C'est l'algorithme choisi. Il y a moins de lignes de code et il ne faut pas parcourir le tableau, juste de la chance de tomber sur une case vide. Pour éviter de placer un objet sur un autre objet, on note dans la case correspondante une étoile.
  Problème anticipé: si on ne supprime pas les étoiles des parties passées, après 37 parties le programme se fige dans une boucle infinie car il n'y a plus aucune place pour les objets ! (car avec 3 objets par partie et avec environ 112 cases vides (15*15/2), il faut 112/3=37 parties pour tout remplir)

Gestion du projet

Le projet a été développé avec Atom, sous Ubuntu 16.04 et en versionnant le code avec git/github.
Une bonne partie du temps était passé a se documenter, a lire des tutos et a chercher des réponses sur différents sites (notamment stackoverflow).
L'avancement du projet était facilité/retardé par les points suivants:

Points positifs:

- Divers connaissances techniques/geek.
- Je sais déjà programmer en C.
- Je suis familier avec la sdl.
- Je suis a l'aise en ligne de commande (pratique pour se faire des alias et utiliser des clefs ssh avec github).
- Projet bien organisé (découpage en versions et anticipation des dépendances des diverses fonctionnalités).
- Manipulations graphiques de base sous Gimp.

Points à améliorer:

- Penser et mettre en place la structure du programme dés le départ.
- Travailler plus régulièrement sur le projet.
- Pusher sur github plus souvent.