– Projet – 3 –

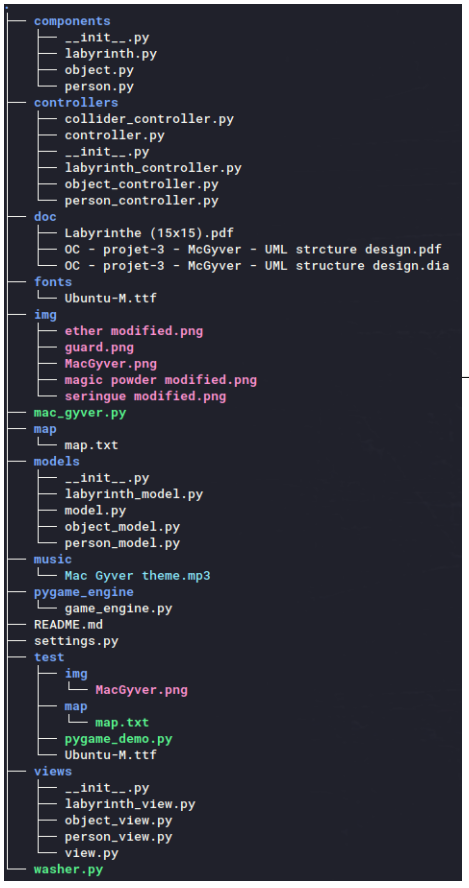
Mac Gyver labyrinth game

Explications du code

OPENCLASSROOMS

Introduction

Cahier des charges



Concevoir un jeudont le héro (“MacGyver”) doit parvenir à sortir d’un labyrinthe et que ce code puisse tourner sur n’importe quel opérateur système.. Le héro doit trouver la sortie de ce labyrinthe (de 15 x 15). Devant la sortie se trouve un garde qui ne bouge pas mais qui est mortel si le hero rentre en contact. Cependant, trois objets à receuillir sont disséminés dans le labyrinthe. L’obtention de ces trois objets par le héro va lui permettre de changer son rapport avec le guarde à son contact (celui-ci va s’endormir et le héro pourra sortir).

Le principe général du code

Le code adopte un structure MVC (Model-View-Controller). Le principe du code c’est de récupérer un fichier de carte (map.txt) qui ne contient que des caractères dont la représentation est du code hexadécimal par demi-octet (1 caractère pour ½ octet) représentant pour chacun d’eux une cellule du labyrinth avec ses murs d’enceinte.

Ensuite, pour gérer le jeux, je me suis servi d’une librairie existante dédiée aux jeux 2D: “pygame”. Puis j’ai créé un fichier game_engine.py qui contient la classe GameEngine qui correspond au moteur du jeux.

Puis j’ai dessiné une structure objet, un plan UML du projet à construire (lien du plan UML) qui m’a permis de dicerner les différents modèles, vues et controlleurs avec leurs méthodes, les méthodes communes qui peuvent être séparés dans des classes de base communes, etc…

Il y a donc un répertoire contenant les modèls, un autre pour les vues, un pour les controlleurs, un pour les composants et un pour le moteur du jeux. Ensuite, d’autres répertoires contiennent la carte, les images à afficher, une musique, de la documentation… les noms de ceux-ci parlent d’eux même.

EXPLICATIONS DU CODE

LA CARTE

Elle est constitué de carcatères qui, chacun d'eux, représente un demi-octets et reflète l'existance et la disposition des murs de chaque cellule du labyrinth (donc 15 par ligne, et 15 lignes pour un labyrinthe de 15 x 15).
Par exemple le demi-octet 0 est une cellul sans mur, alors que F est une cellule avec 4 murs (F => 111 en binaire). Pour déchiffrer le code en partant de sa représentation hexadécimale, il faut simplement convertir l'hexadécimal en binaire, puis le bit de poid faible représente le mur du haut (puisà suivre dans le sens des aiguilles d'une montre). Par exemple le B => 1011 correspond à une cellule avec 3 murs, et il n'y a pas de mur à gauche. (lien de schéma de carte)

PYGAME ET LE CODE MVC

Pygame a sa propre structure mais n'est pas MVC. Des appels à pygame existent donc en dehors du moteur de jeux, mais de façon restreinte de façon à conserver une structure de controleurs, vues, modèles. PyGame c'est assez simple, vous initialisez une fenêtre, un titre, une horloge, et ensuite vous démarrer une boucle infinie (qui va s'arrêter à vôtre demande, à la fin du jeux). Ensuite à chaque tour de boucle, des actions à mener vont consister à saisir une entrée, afficher quelque chose, cacluler des collisions, etc… Puis pour les collisions, pygame utilise des "sprites" qui sont en fait des images avec une position, qui peuvent être insérées dans un ou des groupes… ceux-ci pouvant analyser à la demande des collisions entre chaque groupes et chaques sprites créées.

MON CODE

mac_gyver.py est le fichier à exécuter pour démarrer le jeux. Il démarre le moteur de jeux, puis créé des objets qui sont tous des controleurs: Le labyrinth (qui va gérer l'affichage de la carte, mais aussi lenvoi du son et la disposition des murs en créant pour chaque mur, un objet: "Wall"), Le héro, le garde et les objets. Il y a donc 4 grands controlleurs.
Un controlleur spécial s'occupe de la gestion des collisions (classe Collider) dont va hérité le héro (puisque seulement lui peut bouger, c'estd onc bien lui qui va créer des collisions…. à lui de les gérer de facto).
Les évennements en provenance du clavier sont gérés par le controleur principal (la classe de base des contrôleurs), Tout ce qui est à affiché est géré par les vues
Ensuite, chaque controlleur va créer ses modèles, et ses vues (qui vont tenir compte des modèles). Les modèles comportent une classe de base qui est un objet composant commun propre à l'identité de se fonctions et, lorsque elle peuvent hériter de celle-ci, de l'objet qu'il représente (par exemple, une classe HeroModel héttie de Hero et de Model), parfois au lieu d'éhrité de l'objet représenté, celui-ci est en attributs parce que le modèle représente plusieurs de ces objets, comme par exemple la classe ObjetModel, qui n'hérite que de Model mais dont les "Object" sont liés dans une attribut liste. Il est important de regader le plan UML pour voir les liaisons logiques qui relient les classes. Ça reste très simple: les modèle gèrent les données propres à chaque entité, les vues gèrent ce qui doit être afficher ou ce qui est directement en rapport, et les controlleurs décident des ations à mener et interagissent avec l'utilisateur, une classe commune partage les ressources de codes et les attributs communs.
Pour afficher des textes avec le moteur de jeux (comme celui-ci gère peu cela), j'ai créé un petit mécanisme de stockage de fonctions simples (avec leurs arguments) dans une structure de dictionnaire, poussés dans un simple liste (_background_jobs) qui ensuite est exécuté à chaque tour de boucle du moteur de jeux (je n'ai pas ajouté de threads).
Le code étant généreusement commenté, pour entrer plus dans le détail du fonctionnement d'un jeux comportant 41 fichiers et 25 classes (en deux pages, ça n'est raisonnablement pas possible), rentrez llire le code avec le plan UML sous le coude.. (lien github)