Présentation projet 3
Aidez Macgyver à s'échapper !

Lien vers le projet : *https://github.com/Anca6889/OC_Projet_3*

Introduction :

Cette présentation a pour objectif de décrire les différentes étapes pour la réalisation d'un jeu où le joueur incarne MacGyver et doit s'échapper d'un labyrinthe en ramassant une série d'objet avant de pouvoir éliminer un gardien pour s'échapper.

SOMMAIRE :

A) Description de l'architecture :

1) Un dossier pour la partie algorithme et fonctionnement « *classes* » :
    a. Trois classes pour la partie algorithme. Une classe qui aura pour but d'instancier le joueur, une seconde le gardien et enfin une dernière qui aura pour but de gérer le labyrinthe avec l'interaction du joueur et des éléments à l'intérieur. Un module de config pour les constantes.
2) Un dossier pour la partie graphique Pygame « *display* » :
    a. Cinq classes, dont une pour l'interface du menu principal, deux autres pour l'interface du jeu et l'inventaire puis enfin deux dernières pour les écrans de victoire ou de défaite. Un module de config pour les constantes.
3) Un dossier pour la partie audio « audio » :
    a. Une classe pour l'audio regroupant tous les bruitages et musiques utilisées dans l'application. Un module de config pour les constantes.
4) Un dossier « *game* » :
    a. Une seule classe Game qui gère tout le fonctionnement du jeu et dans laquelle la plupart des classes précédentes s'entrecroisent. Un module de config pour les constantes.
5) Racine du projet :
    a. Un module « main.py » qui lance le module « game.py. »

B) Cadre de départ et fonctionnement de l'algorithme :

1) Création d'une grille virtuelle de jeu :
    a. En premier lieu, j'ai réalisé un fichier txt « level.txt » ou le labyrinthe de base est dessiné et où chaque éléments chemins(-), murs(#), positions des personnages(M & G) et point d'arrivé(1) sont représentés sur une grille de 15x15.
    b. Création des classes « *Level* », « Player » et « Guardian ».
    c. A l'aide d'une fonction « *grid_gen* » dans la classe « *Level* », qui lit chaque élément de « level.txt » , on répertorie dans un dictionnaire « *self.coord* » toutes les coordonnées x et y de chaque élément de « level.txt » et dans une liste uniquement les chemins « *self.path* ».
    d. Dans cette même fonction « *grid_gen* » on instancie Macgyver et le gardien sur la position attribué par la lettre M et G.

2) Positionnement aléatoire des objets :
    a. Je place en premier lieu le nom des objets dans une liste « OBJECTS » du module « *config_classes* » sous forme de chaine de caractère.
    b. A l'aide de la fonction « *set_objects* » on pioche trois clés (positions) non identique au hasard dans la liste des chemins « *self.path* » généré précédemment. On va récupérer chacun des objets dans le module « *config_classes* », puis on va attribuer à la valeur des clés piochées (obligatoirement un chemin et se trouvant forcément également dans le dictionnaire « *self.coord* »), une nouvelle valeur qui correspond à l'objet dans le dictionnaire « *self.coord* » .

3) Déplacement et contrôle des positions :
   a. A l'aide de la fonction « *move* », on déplace simplement Macgyver d'une case x ou y en fonction de la direction où on veut aller.
   b. Pour contrôler le contenu de la case de destination, on utilise des conditions : si la prochaine coordonnée contient un chemin, alors on peut avancer et mettre à jours les coordonnés de Macgyver sans oublier de remettre un chemin à sa position précédente. Si la cellule suivante contient un objet, alors l'objet doit être rajouté à l'attribut « *self.inventory* ». Si la cellule suivante contient le gardien et que « *self.inventory* » est égal à 3 (on a tous les objets) alors on peut avancer. Sinon on perd le jeu (grâce a une fonction détaillée ci-après), si la cellule suivante contient le chiffre 1 (la sortie atteignable uniquement en passant le gardien) on gagne le jeu.

C) Partie graphique et audio avec Pygame :

1) Cadre général :
   a. L'ensemble du jeu se déroulera dans une fenêtre de 640x600px. (Différents menus et le jeu principal)
   b. Les cellules de la grilles sont des carrés de 40px de coté
   c. L'inventaire se situera à gauche de la fenêtre et fera 40x600px

2) Classes display et audio :
   a. Les classes « *_display » ne sont que l'ensemble des attributions d'images aux éléments du jeu.
   b. La classe « Audio » n'est que l'ensemble de l'attribution des fichiers audios aux éléments du jeu.

3) Classe Game :
   a. Cette classe a pour objectif de faire tourner tout le jeu.

   b. Création graphique du labyrinthe : A chaque élément de « level.txt » est associé une image de 40px. Pour placer les images à la bonne place, on multiplie simplement leur positions x et y par 40px. Ces instructions sont contenues dans la fonction « *game_gen* » , sur le même principe, les fonctions « *main_menu_gen* », « *gameover_menu_gen* » et « *win_menu_gen* » génèrent la partie graphique des différents menu. Elles sont cependant moins complexes, ne nécessitant qu'un fond d'écran et un bouton. En dernier lieu, il y'a encore la fonction « *inventory_gen* » qui génère l'inventaire sur le coté gauche de l'écran et qui met à jour les images en fonction des objets ramassés. Une fonction « *mini_sound* » viendra compléter ce ramassage d'objet en jouant un petit son lorsqu'un objet est ramassé, mais également lorsque le gardien est vaincu.

   c. Le jeu comportant 4 écrans différents, cette classe comporte 4 fonctions avec des boucles « *while* ». Chacune de ces fonctions génère un écran de jeu (1 pour chaque menu et 1 pour le jeu principal). Ce sont les fonctions « *lauch_game* » et « *run_** »

   d. Afin d'éviter certaines répétitions dans les fonctions de boucles ci-dessus, plusieurs fonctions ont été créées, une fonction qui instancie toutes les classes nécessaires « *data_gen* » , une autre qui génère à chaque fois la fenêtre de jeu 640x600px « *display_gen* ». Les quatre variables booléennes nécessaires au fonctionnement des boucles dans la fonction « *loop_conditions* » Et enfin comme ces trois dernières fonctions sont à chaque fois nécessaires pour chaque boucle, une fonction « *all_set_fonctions* » se chargera de les regrouper.

4) Module main.py :
   a. Ce module instancie la classe « *Game* » de game.py et joue la fonction « *launch_game* » qui lancera le jeu.

D) Mise en conformité PEP8 :

1) Afin de vérifier la conformité avec la PEP8, j'ai utilisé deux outils « linter » pour repérer les non conformités par rapport à la PEP8 : Pylint et Flake8. Il s'agit simplement de prendre erreur par erreur et d'y apporter les modifications nécessaires.

Jean-Maxime GUTH
Etudiant OPENCLASSROOMS DA PYTHON Le
21/11/2020