1 - Pré-requis

- **Python 3.6.x**.
- pygame

Installation

- Télécharger le repository.
- Utiliser un **environnement virtuel** est recommandé.
    - Exécuter la ligne de commande : `python3 -m venv <path/to/new/virtual/environment>`
      puis `source <path/to/venv>/bin/activate` depuis MacOS
      ou `<path/to/venv>\Scripts\activate.bat` depuis Windows
- Installer les dépendances : `pip install -r requirements.txt`

2 - Lancer le programme

- Exécuter : `python starter.py` depuis la console

Projet MacGyver

Elaboration du programme “Aider MacGyver à s’échapper”

dépôt Github: https://github.com/ivan-fr/oc_projet_3

Programme écrit en Python3 grâce au module Pygame.

Problèmes rencontrés

J’ai rencontré des difficultés avec la gestion de l’affichage de la fenêtre avec les fonctions fournie par pygame.

Fonctionnalités

- Il n’y a qu’un seul niveau. La structure (départ, emplacement des murs, arrivée), est enregistrée dans un fichier pour la modifier facilement au besoin.

- MacGyver est contrôlé par les touches directionnelles du clavier.

- Les objets sont répartis aléatoirement dans le labyrinthe et change d’emplacement si l’utilisateur ferme le jeu et le relance.

- La fenêtre du jeu sera un carré pouvant afficher 15 sprites sur la longueur.

- MacGyver peut se déplacer de case en case.

- Il récupère un objet simplement en se déplaçant dessus.

- Le programme s’arrête uniquement si MacGyver a bien récupéré tous les objets et trouvé la sortie du labyrinthe. S’il n’a pas tous les objets et qu’il se présente devant le garde, il meurt.

- Le programme est standalone, c’est-à-dire qu’il pourra être exécuté sur n’importe quel ordinateur.

- Respecter les bonnes pratiques de la PEP 8

- Code écrit en anglais : nom des variables, commentaires, fonctions…

Présentation du projet :

Mon code est écris avec le design pattern MVC (modèle - vue - contrôleur).

Les modèles, dans **models.py**, permetttent de stocker les informations « brutes » utilisées par la logique du jeu.
Chaque modèle représente une entité utile au jeu:

- **ModelScreen** stock les informations liées à la fenêtre d’affichage du jeu.
- **ModelMaze** stock les informations liées aux labyrinthes.
- **ModelTile** stock les informations liées à chaque case d’un labyrinthe.
- **ModelCharacter** stock les informations liées au personnage (le joueur).

Les managers, dans **managers.py**, gèrent la logique du code, ils vont demander aux modèles les données et les analyser pour prendre des

décisions.
Chaque manager s'occupe d'un ensemble de tâches:

- **GameManager** s'occupe du déroulement des différents états du jeu. (Affichage du menu, affichage du jeu, affichage du résultat).
    - Chaque état de jeu possède des données qui leurs sont propres, ses données sont stockers dans **states.py**, on y trouves LevelState et WinScreenState, LoseScreenState.
- **InputManager** s'occupe de la gestion des touches du clavier disponible pour chaque état du jeu.
- **MotionManager** s'occupe du déplacement du personnage dans un labyrinthe.
- **GraphicManager** s'occupe d'afficher les cellules d'un labyrinthe sur l'écran.
- **LogManager** s'occupe d'afficher tous les messages sur l'écran.

GraphicManager et **LogManager** peuvent être considérés comme des vues.

Structure du labyrinthe:

- Quand le jeux est initialisé depuis **starter.py**, la méthode "fill_structures" de la classe ModelMaze dans **models.py** récupère la structure des labyrinthes à partir de **levels.txt**.
  Un labyrinthe contient 15 lignes de 15 caractères:
    - "j" = joueur
    - "m" = mur
    - "0" = case vide

- “a” = gardien