

1 - Pré-requis

- Python 3.6.x.
- pygame

Installation

- Télécharger le repository.
 - `git clone https://github.com/ivan-fr/oc_projet_3`
- 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 main.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.

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 écrit avec le design pattern MVC (modèle - vue - contrôleur).

Tout d'abord le programme se décompose en cinq fichiers principaux, on a : **models.py**, **states.py**, **managers.py**, **views.py** et **store.py**.

Les modèles, dans **models.py**, permettent 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 states, dans **states.py**, représentent les données associées à chaque état de jeux.

Les états de jeux sont l'écran du labyrinthe, l'écran de partie gagnée et l'écran de partie perdu.

Chaque état est relié à différents modèles selon le contexte :

- **LevelScreenState** est relié à **ModelMaze** et à **ModelScreen**
- **LoseScreenState** est relié à **ModelScreen**
- **WinScreenState** est relié à **ModelScreen**

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

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

- **GameManager** s'occupe du bon déroulement entre les différents états (les states) du jeu (l'écran du labyrinthe, l'écran de partie gagnée et l'écran de partie perdu).
- **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.

Les vues, dans **views.py**, s'occupent de lire les modèles via les states, dans **state.py**, et d'afficher du contenu (résultant de la lecture) à l'écran.

Chaque vue s'occupe d'afficher leur propre interface:

- **GraphicView** s'occupe d'afficher les cellules d'un labyrinthe sur l'écran.
- **LogView** s'occupe d'afficher tous les messages sur l'écran.

Enfin, la classe **Store**, dans **store.py**. Cette classe permet de donner à un manager ou à une vue, s'il le souhaite, l'état courant (le **state** courant) afin que ledit manager ou ladite vue puisse manipuler (ou lire) les données provenant de l'état (du **state**).

Je précise que seuls les états (les **states**) sont reliés à des modèles.

Structure du labyrinthe:

- Quand le jeu est initialisé depuis **main.py**, la méthode "fill_structures" de la classe ModelMaze dans **models.py**

récupère la structure des labyrinthes à partir de **file/levels.txt**.

Un labyrinthe contient 15 lignes de 15 caractères:

- "j" = joueur
- "m" = mur
- "0" = case vide
- "a" = gardien

Problèmes rencontrés

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

Pour résoudre ces difficultés j'ai regardé la documentation de pygame de manière plus approfondie, et, j'ai cherché des solutions à mes problèmes sur des sites comme stackoverflow.