>> DOCUMENTATION

I. Prérequis :

PycMyn nécessite Python 3.10 au moins avec la librairie pygame. Pour cela utiliser les commandes :

```
>>> pip install pygame
```

Le jeu nécessite aussi un système audio opérationnel (peut être désactivé).

Pour respecter l'idée du jeu, nous avons utilisé le fandom de PacMan : https://pacman.fandom.com/wiki/Pac-Man_(game)#:~:text=Split%2Dscreen%20level,-Main%20Article%3A%20Map&text=If%20one%20uses%20a%20hack,known%20as%20a%20kill%20screen.

Le jeu nécessite au moins 40 images par seconde. Autrement, des bugs de navigation apparaissent.

Pour jouer, lancez le fichier main.py

II. Analyse :

1. Le fonctionnement graphique

L'ensemble du code est écrit en Python, avec un style **orienté objet** principalement et **fonctionnel** de façon secondaire. La seule librairie extérieur est pygame. L'ensemble du sur-code graphique est donc géré dans le module `graphics.py`.

Le choix des structures de données s'est porté principalement sur les listes car les données stockées ne sont pas de grande taille (par exemple pour le stockage des éléments graphiques d'une interface). On peut ainsi profiter de la simplicité des listes.

Le fonctionnement de ce module peut être rapporté à ceci :

- Une **interface** est un ensemble d'**éléments** graphiques liés entre eux. Chaque élément graphique est indépendant des autres dans la gestion de son positionnement et de son affichage personnel.

- L'interface regroupe ces affichages individuels en une liste de surfaces. Si l'interface est la plus élémentaire, elle donne son résultat d'affichage à l'écran. Sinon, elle fait partie d'une autre interface et remonte à celle-ci sa liste d'affichage

Tous les autres éléments graphiques sont construits à partir de ces briques élémentaires. Le bouton par exemple est doté d'un élément, qui est responsable de son affichage, et d'une méthode spécifique à l'interaction avec l'utilisateur qui ignore tout de l'affichage.

2. Le fonctionnement du jeu :

Par dessus ce système, le reste du jeu est construit. Tous les objets présents dans le jeu (fantômes, joueur, fruits) ont pour racine la classe **Entity** du module `entite.py`. Cette classe permet la gestion des collisions entre toutes les entités et avec les murs pour les entités mouvantes grâce aux méthodes *`collide_with`* et *`collide_wall`*. Les collisions sont faites à partir de l'intersection des masques des entités. Ceux-ci sont construits lors de l'instanciation des entités grâce à la fonction *`forme_mask`*, puis ajustés lors de l'évolution de la surface de l'élément.

Le module `collectable.py` rassemble toutes les entités récupérables par le joueur (pouvoirs, pièces, fruits) ainsi que les fonctions associées à leur apparition. À chaque début de partie, le jeu génère quatre fruits et quatre pouvoirs de façon aléatoire sur le terrain.

Le terrain est défini par le module `plateau.py`. Ce module crée un fond à partir de l'image **map.png** et génère un masque associé qui sera utilisé dans les collisions avec les murs grâce à la fonction *`forme_mask`*. Pour le plateau, on utilise une meilleure précision afin de distinguer la zone de sortie des fantômes qui serait autrement absorbée par deux carrés. En effet, la fonction *`forme_mask`* parcourt la surface donnée subdivisée en carrés de taille size x size. Si un pixel de la zone parcourue est actif, toute la zone du masque est activée.

La résolution donnée à la fonction permet donc d'avoir un masque d'autant plus précis que la valeur est petite, jusqu'à retrouver le comportement de la fonction fournie de pygame *`pygame.mask.from_surface`* quand size vaut 1.

Les modules `player.py` et `fantome.py` permettent de contrôler respectivement le joueur et les fantômes. Le contrôle du joueur est un peu plus complexe que ce qu'on pourrait penser car il faut mémoriser la direction souhaitée par l'utilisateur jusqu'à ce qu'elle soit effectuée. Tout le contrôle du joueur passe par la méthode *`controle`*. Toutes les interactions sont gérées par les méthodes *`collect`* et *`interact`*. Cette seconde méthode repose sur l'identifiant des entités qui permet de les répartir

selon leur classe. L'identifiant 0 revient aux collectables, 1 aux fantômes, 2 au joueur et 3 à la porte.

Le contrôle des fantômes passe par une méthode assez similaire. Toutefois, comme il n'y a pas d'humain pour donner les directions à suivre, on doit faire appel à une fonction qui va reproduire ce choix. Ces fonctions sont `follow`, `evite`, `aleatoire` et `piege`.

Toutes les secondes, les fantômes analysent la situation avec leur fonction respective et recalculent la meilleure direction. Si cette direction ne convient pas (il y a un mur), ils vont alors chercher dans leur mémoire leur dernière direction valide. Si celle-ci non plus ne fonctionne pas, les fantômes analysent plus en détail la situation.

S'ils se trouvent à une intersection, ils recalculent parmi les directions disponibles leur direction préférée et avancent dans cette direction.

Les fonctions de choix citées plus haut permettent de calculer cette direction préférée. `aleatoire` renvoie une valeur au hasard parmi celle qu'on lui a fournie. `follow`, `piege` et `evite` sont assez similaires :

follow cherche à maximiser le produit scalaire de la direction avec le vecteur de la différence de la position du fantôme et celle du joueur :

```
vector = player.pos.xy - fantome.pos.xy

direct = vector.normalize() if vector.length() != 0 else vector

produits_scalaires = [utl.gen_vector(direction).dot(direct) for direction in directions]

return directions[produits_scalaires.index(max(produits_scalaires))]
```

De façon similaire, `piege` va seulement prendre la position 5 cases plus loin dans la direction du joueur afin d'anticiper un peu.

`evite` cherche au contraire à minimiser ce produit scalaire afin d'éviter le joueur.

3. Sauvegarde des données

La sauvegarde des données est faite dans un fichier json afin que cela puisse être facilement éditable par un humain, d'autant plus que le nombre de données à sauvegarder entre chaque partie est faible (seulement le meilleur niveau et le niveau actuel). Lors du lancement du jeu, le fichier json est importé grâce au module json et est chargé en tant que dictionnaire. Ses valeurs sont changées pendant les parties, avant d'être de nouveau stockées quand le jeu est fermé.

Les paramètres de niveau sont enregistrés dans un fichier csv chargé grâce au module python de même nom.