Aidez *MacGyver* à s'échapper!

Résumé

Développez un jeu de labyrinthe en 2D dans lequel un personnage:

- est déplacé à l'aide des touches directionnelles
- collecte des objets placé au hasard

Les consignes détaillées sont disponible sur la page du projet.

Le code est hébergé sur GitHub.

Environnement

Le développement à été réalisé avec:

- `python 3.6.4`
- `pygame 1.9.3`
- `LMDE2`

Démarche

Pour la construction du script, j'ai suivi une démarche séquentielle: commencer par les fonctionnalités basiques puis ajouter les fonctionnalités plus spécifiques:

1. labyrinthe jouable en console, commandes événementielles
2. interception des événements clavier avec `pygame`
3. affichage graphique (GUI) du labyrinthe avec `pygame`
4. déplacement du joueur dans la GUI jusqu'à la sortie
5. placement des objets *collectable* (items) dans le labyrinthe
6. ramassage des items par le joueur
7. messages contextuels par tour
8. vérification des objets ramassé pour déterminer si la partie est gagnée/perdue

Structure

Tout en gardant en tête les principes du zen of Python, les critères retenus pour la conception de l'algorithme ont été les suivants: modularité et évolutivité.

Le code est répartit dans les fichiers suivants:

- **`main.py`** : script principal à lancer
  1. importation les différents modules du script
  2. si la création de l'objet `Maze` est réussie, les autres objets (`GraphicUI & Player`) sont alors créés
  3. démarrage de la boucle de jeu: traitement primaire des événements clavier et on appel les méthodes correspondantes
  4. boucle du dernier message, pour laisser le dernier message afficher
- **`conf.py`** : données de configuration du script
  1. la liste `ELEMENTS` contient toutes les données des éléments présents dans le labyrinthe, on accède à son contenu par la fonction `elmt_val()` en fin de fichier
  2. les dimensions, messages, chemins, etc.
  3. les données *calculables* sont calculée en fin de fichier
- **`maze.py`** : classe `Maze`: gestion du labyrinthe
  1. vérifie que le fichier labyrinthe est utilisable
  2. renseigne les données spécifique du labyrinthe utilisé
  3. place les items au hasard
- **`player.py`**: classe `Player`: gestion du joueur, traitement secondaire des événements du clavier et les conséquences de ceux ci (déplacement, collecte d'items, fin de partie, messages)

- **`gui.py`** : classe `GraphUI`: gestion de l'interface graphique
  1. *wrap* la partie graphique de `pygame display` & `font`
  2. affiche l'intégralité du labyrinthe, ou seulement les tuiles modifiés chaque tour
  3. affiche le *header* (messages)
- **`01.maze`** : fichier définissant le labyrinthe
  1. une case pour un caractère
  2. les caractères sont ceux de la liste d'éléments contenus dans `ELEMENTS[x]['symbol']`
  3. ses dimensions (H & L) doivent correspondre à la valeur de `MAZE_SIZE`
- **`img/`** : répertoire des images : leurs dimensions (H & L) doivent correspondre à la valeure de conf `CELL_SIZE`
- **`requirement.txt`** : fichier pip des dépendances du script, pour configurer un environnement virtuel permettant une exécution correcte du script
- **`README.md`** : mode d'emploi

Difficultés rencontrées

Le modèle objet

Ce script aura été pour moi l'occasion de comprendre véritablement comment construire un modèle objet. Ma principale difficulté aura été de constituer les 3 objets utilisé. Pour y parvenir j'ai consacré beaucoup de temps à étudier du code et le *déclic* c'est finalement fait quand j'ai compris que les objets pouvaient être passé en paramètre des un et des autres.

La définitions des éléments

Tant que je n'étais pas à l'aise avec mon modèle objet, je n'étais pas à l'aise non plus avec la définitions des éléments. Finalement j'ai préféré ne pas répartir cette définition dans divers objets afin de garder la centralisation de la configuration.

Rester dans le périmètre du projet

L'énoncé était pour moi assez flou: pas facile de savoir clairement quelles fonctionnalités doivent être implémentée. Certaines m'ont semblé indispensable sans avoir été demandées explicitement, par exemple:

- Les messages du *header*
- `GraphUI.update()`
- `Maze.check_file()`

Navigation *à l'index*

Le choix évident pour naviguer dans le labyrinthe est d'utiliser un stockage dans une variable avec des coordonnées sur 2 axes. J'ai trouvé l'usage d'une *string* plus intéressant à développer du fait du contexte *fini* du labyrinthe. Le revers de la médaille à été la gestion du caractère de fin de ligne. L'usage d'une chaîne *splittée* (`Maze.string`) ainsi que d'une méthode de conversion *index => coordonnées sur 2 axes* (`coord_from_index()`) aura permis de cerner cette problématique.