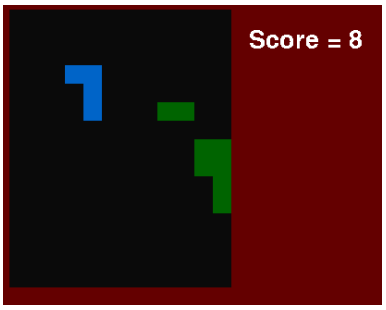
Projet python : Jeu de blocs

Dans ce projet, vous allez créer un petit jeu de « blocs ». Le joueur déplace un bloc sur l'écran à l'aide des flèches, et valide la position d'un bloc avec la touche entrée. Quand certains alignements sont réalisés, des blocs disparaîssent. Quand on ne peut plus jouer, on a perdu.



Le score augmente en fonction de ce que fait le joueur (alignements, pose des blocs, etc).

Vous avez une certaine liberté dans les règles du jeu.

0 Consignes de rendu

Ce projet est à faire *seul(e) ou à deux*, et il est à rendre pour le 12 février au plus tard (avant les vacances).

Les séances en classe feront avancer la principale partie du projet, mais vous avez le droit d'avancer à la maison.

Vous le mettrez dans un dossier à vos noms, dossier qui contiendra tous les fichiers de votre jeu, dans le dossier Ramassage sur une machine, et il sera ramassé automatiquement.

En cas d'absence ou d'oubli, vous pouvez envoyer le projet par Pronote ou le nuage, avant le soir à minuit. Pour cela, il faudra mettre votre dossier de travail dans une archive au format zip.

Vous rendrez le travail sous forme de dossier à vos noms, contenant votre fichier (ou vos fichiers s'il y en a plus) ainsi que le fichier `graphique_jeu.py`.

Toutes les fonctions devront être documentées par une « docstring », indiquant quels sont les arguments en entrée, ce qui est renvoyé (si il y a quelque chose de renvoyé) et une brève description de la fonction (« mode d'emploi » de la fonction).

Notamment, la docstring de la fonction `jeu` devra contenir vos règles du jeu (voir plus bas).

La lisibilité du code est, comme d'habitude, très importante : mettez des commentaires, utilisez des boucles et fonctions là où c'est pertinent, et mettez des commentaires. Et n'oubliez pas de commenter votre code, même si vous trouvez que votre prof' radote.

1 Principe

1.1 La grille

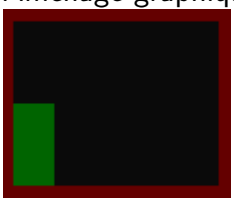
La grille de jeu peut être de taille variable, mais reste un rectangle. Elle est représentée sous forme de *matrice* ou tableau double. Par défaut, elle contient des 0 et des 1. 0 signifie que la case est « libre » et 1 que la case est occupée. Dans l'affichage, les cases libres sont noires et les occupées en vert.

L'affichage graphique de la grille correspond à l'affichage du tableau en python. Exemple :

Matrice python :

```
g = [[0, 0, 0, 0, 0],
     [0, 0, 0, 0, 0],
     [1, 0, 0, 0, 0],
     [1, 0, 0, 0, 0]]
```

Affichage graphique :



Explications :

Ici les dimensions de la grille sont 5 (largeur) et 4 (hauteur). Deux cases sont occupées, `g[2][0]` et `g[3][0]`

1.2 Le bloc « mobile »

Le bloc mobile est celui que le joueur peut déplacer avec les flèches. Il est représenté par un *tableau* de *coordonnées*, donc un tableau de tuples. Chaque tuple du tableau correspond aux coordonnées (dans la grille) de là où se situe le bloc. Sur l'affichage graphique, ce bloc sera représenté en bleu.

Exemple :

Le bloc en python :

```
b=[(0, 0), (0, 1),
(1, 1), (2, 1)]
```

Affichage graphique :

Explications :

Le premier élément de la liste, `b[0]` représente le bloc en haut à gauche : $(0,0)$. `b[1]` représente le bloc dans le « coin » du « L » : $(0,1)$, etc.

2 Ce qui vous est fourni

On utilise pour ce projet la librairie `pygame`. Cette librairie permet l'affichage graphique. Il vous faut installer cette librairie pour pouvoir visualiser le jeu. Vous n'avez pas besoin de maîtriser cette librairie.

- Si vous avez installé spyder, vous pouvez (probablement) installer pygame en tapant dans la console de spyder
  `pip install pygame`
- Si vous avez installé edupython, la librairie est (probablement) déjà installée
- Sinon, allez voir la documentation ici : `https://www.pygame.org/wiki/GettingStarted`

Pour tester si la librairie est installée, il suffit de taper dans la console (ou dans un programme vide) `import pygame`

Si vous n'avez pas d'erreur, c'est bon !

Si vous avez des difficultés à installer pygame, sachez qu'il est possible de travailler sur une grande partie du projet sans (vous visualiserez votre jeu sous forme de matrice et ce n'est pas très joli mais ça marche).

Le fichier `graphique_jeu.py` contient des fonctions et instructions d'affichage avec la librairie pygame. Il est commenté et expliqué, mais vous n'avez pas besoin de comprendre tout ce qui s'y passe.

La seule chose à savoir est qu'il y a dedans une fonction **affiche_jeu** qui prend en argument une grille de jeu, un bloc mobile, et un texte (chaîne de caractère) qui s'affichera à droite du jeu (par exemple le score).

Le fichier `jeu_eleves.py` est le fichier qui va contenir l'essentiel de votre code. Il y a dedans l'instruction pour importer les fonctions de `graphique_jeu.py`, un début de première fonction (`cree_grille`)

et la fonction principale du jeu (`jeu`) partiellement écrite. Vous devrez les compléter et y ajouter les fonctions nécessaires pour faire tourner votre jeu.

Il suffira d'ajouter à la fin de votre programme `jeu(10,5)` par exemple, pour lancer le jeu sur une grille 10×5.

3 Ce qu'il vous reste à faire

Prenez le temps de réfléchir aux règles de votre jeu. Voici quelques questions pour vous guider :

- Peut-on déplacer le bloc mobile en le superposant aux blocs fixes, ou non ? Que se passe-t-il si on le fait, est-ce que ça "bloque" simplement ou y a-t-il un comportement particulier (malus de points, un morceau de bloc disparaît, ...) ?
- Idem pour le déplacement qui sortirait de l'écran.
- Quand on valide la position d'un bloc, a-t-il le droit de se superposer à un bloc existant ? Est-ce interdit par le déplacement (selon question précédente), ou cela fait perdre ? Ou un simple malus sur le score ?
- Que se passe-t-il quand des blocs (fixes) forment une ligne, une colonne ? Vous pouvez également avoir d'autres combinaisons intéressantes de votre choix.
- Comment le score augmente-t-il ? Le score a-t-il une influence sur le jeu ?
- Le plus simple, pour "perdre" est qu'il soit impossible de créer un nouveau bloc, ou de le valider pour diverses raisons. Est-ce bien votre choix ? Comment cela se manifeste dans le code ?

Vous expliquerez les règles du jeu dans la "docstring" de la fonction `jeu`.

Vous devez écrire les fonctions suivantes. Leur spécification peut être modifiée selon vos besoins, indiquer clairement cela dans leur documentation.

1) `cree_grille(longueur, largeur)` crée la grille vide de dimensions longueur et largeur. Selon vos règles du jeu, il pourra être pertinent d'y mettre quelques asserts pour éviter les dimensions aberrantes.

   Par exemple, `cree_grille(5,4)` va renvoyer

```
[[0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0]]
```

2) `nouveau_bloc()` est la fonction qui crée un nouveau bloc mobile, aléatoire. Par exemple elle pourrait renvoyer `b=[(0, 0), (0, 1), (1, 1), (2, 1)]` qui est le bloc proposé en exemple plus haut. Prévoyez assez de blocs pour que votre jeu soit intéressant !

   Selon vos règles du jeu, cette fonction pourrait prendre des arguments.

3) `valide(grille, bloc)` vérifie si le bloc est "valide" dans la grille (s'il a la place d'être là). Par exemple, s'il ne sort pas du cadre de jeu et s'il ne se « superpose » pas aux blocs fixes.

   A priori, cette fonction sera utilisée pour vérifier si un déplacement est valide. Selon les règles du jeu on pourra peut-être en coder une autre pour vérifier qu'on peut bien « ancrer » le bloc ici.

4) `deplacebloc(grille, bloc, direction)` où `grille` est la grille de jeu, `bloc` est un bloc mobile et `direction` est par exemple "H", "G", "D", "B" (quatre directions). Vous pourrez décider de vos conventions pour indiquer la direction.

   Cette fnoction *renvoie* le nouveau bloc déplacé dans la direction donnée (et peut renvoyer le bloc d'origine si jamais le bloc n'a pas pu bouger).

Elle pourra faire appel aux fonctions précédentes, comme par exemple `valide` afin de s'assurer que le déplacement est possible (ne fonce pas dans les bords ou dans un bloc fixe, si c'est dans vos règles).

Indication : un déplacement à droite correspond à quoi pour les coordonnées [i][j] dans une matrice ?

5) `validation(grille, bloc, score)` va « ancrer » le bloc mobile pour en faire un bloc fixe (et donc passer au bloc suivant). Le score sera stocké sous la forme d'un tableau à une seule valeur (ce qui permettra de le modifier dans cette fonction).

   Cette fonction effectue :
   — Vérifie si on peut bien mettre le bloc là (selon vos règles c'est automatique ou pas).
   — Si oui, on « ancre » le bloc dans la grille et ça devient du bloc fixe.
   — Ensuite, on doit vérifier les conditions de type ligne complète, colonne complète, et autres combinaisons spéciales. S'il y a disparition de blocs, le faire.
   — Mettre à jour le score selon ce qui s'est passé (selon vos règles).
   — Si on ne peut pas mettre le bloc là, voir ce qui se passe selon vos règles (rien, ou autre chose ?).

   Elle renverra `True` si le bloc a bien été validé (et qu'on peut passer au bloc suivant), et `False` sinon.

6) La fonction `jeu` prend en argument les dimensions de la grille voulue. Elle est déjà partiellement écrite, à vous de la compléter.

N'oubliez pas de commenter votre code... et de bien écrire les spécifications des fonctions !

4 Pour aller plus loin

Le rendu graphique utilise la bibliothèque `pygame`. Vous pouvez, si vous le souhaitez, aller modifier `graphique_jeu.py` selon vos goûts et choix de règles. Vous pourrez prendre exemple sur le code existant, et/ou aller regarder la documentation de pygame ici : `https://www.pygame.org/docs/` (en anglais).

Des bonus pourront être accordés si ce qui est en plus est intéressant.

Quelques idées : Des blocs de « statut » différent, ce qui peut se manifester par des nombres différents dans la grille (autres que 1), et s'affiche d'une couleur différente. Une nouvelle touche qui a un effet spécial (regarder la documentation de pygame pour avoir le « code » de la touche). Des nouveaux blocs qui deviennent plus « durs » à placer au fur et à mesure qu'on avance dans le jeu (nombre de coups, score, ...). Des événements spéciaux quand on passe des paliers sur le score. Et à votre imagination !