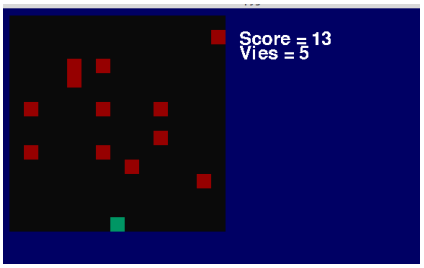
Projet python : Jeu de blocs

Dans ce projet, vous allez créer un petit jeu. Le joueur déplace un petit carré bleu-vert sur l'écran à l'aide des flèches directionnelles, et doit éviter (ou pas ?) les blocs rouges qui bougent sur l'écran.



Vous avez une certaine liberté dans les règles du jeu : que se passe-t-il quand on touche un bloc rouge (score, points de vie, game over ?), comment ces blocs bougent, à quelle vitesse, ...

De nombreuses adaptations sont possibles...

0 Consignes de rendu

Ce projet est à faire *seul(e)*, et il est à rendre pour le .....

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

Vous archiverez votre projet (qui contiendra probablement deux fichiers python, le fichier avec votre code ainsi que le fichier `graphique_jeu.py`) dans un fichier **zip** avant de le mettre en ligne sur pronote, section "devoirs à rendre".

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 Les coordonnées

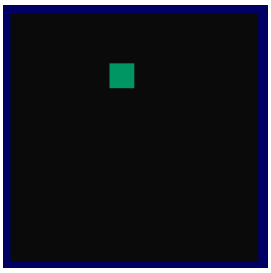
On a une « grille » rectangulaire de dimensions `taillex` × `tailley`.

Les coordonnées sont des entiers entre `0` et `taillex-1` pour les abscisses et `0` et `tailley - 1` pour les ordonnées.

Attention, l'origine $(0,0)$ est en haut à gauche (et donc le sens « positif » pour y est vers le bas).

Les coordonnées du joueur (bloc bleu-vert) sont désignées par ses coordonnées entières `x` et `y`.

Exemple :

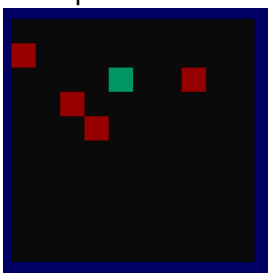


Voici un placement pour une grille 10×10, avec `x=4` et `y=2`.

1.2 Les carrés rouges qui bougent

Les carrés qui bougent seront codés sous forme de deux tableaux : `lx` et `ly` qui contiennent les coordonnées en x et en y de tous les blocs.

Exemple :



Ici `lx = [2, 7, 3, 0]` et `ly = [3, 2, 4, 1]` cela signifie qu'il y a des blocs rouges en coordonnées $(2,3), (7,2), (3,4)$ et $(0,1)$. Le bloc joueur est placé en `x=4` et `y=2`, comme plus haut.

On pourra envisager ou non d'ajouter petit à petit des carrés rouges au fil de la partie.

De temps en temps, un bloc pourra disparaître (parce qu'il touche le joueur ou sort de l'écran, à vous de décider), il pourra être « réinitialisé » (réapparaître à un endroit donné, ou aléatoire, ...) en changeant ses coordonnées au lieu d'être supprimé.

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 !

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(taillex, tailley, lx, ly, x, y, textes) :

- `taillex`, `tailley` sont les dimensions du jeu.
- `lx`, `ly` les deux listes de coordonnées des carrés mobiles
- `x`, `y` les coordonnées du carré perso

— `textes` : un tableau de chaînes de caractères à afficher à droite du jeu. Si on veut plusieurs lignes, on fait plusieurs éléments du tableau. Par exemple si on met ["Bonjour", "Au revoir"] le texte se met sur deux lignes. Si on met ["Bonjour au revoir"] le texte se met sur une ligne. Pratique pour afficher diverses informations (score, points de vie, niveau, ou autre message !).

Exemple :

Voici un exemple d'affichage de
`affiche_jeu(10, 10, [2, 7, 3, 0], [3, 2, 4, 1], 4, 2, ["Bienvenue", "dans le jeu"])`

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_listes`) 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(15, 15)` par exemple, pour lancer le jeu sur une grille 15×15.

3 Ce qu'il vous reste à faire

Prenez le temps de réfléchir aux règles de votre jeu. Voici quelques questions pour vous guider, même si vous pourrez adapter au fur et à mesure :

— Où démarre le joueur ?
— Combien de blocs rouges seront créés au début de la partie ? Où sont-ils répartis ?
— Comment se déplacent-ils à intervalle régulier ?
— Quel est cet intervalle ? Change-t-il avec le temps ou un autre événement ?
— Que se passe-t-il lorsqu'on touche un bloc rouge ? Score qui change ? Points de vie ? Autre ? Que devient le bloc rouge ? Que devient le bloc perso ?
— Comment le score évolue-t-il ? Et d'autres paramètres éventuels ? Y a-t-il une augmentation du nombre de blocs rouges ?

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_listes(taillex, tailley, nbre)` prend en argument les dimensions du jeu ainsi que le nombre de blocs rouges voulus. Elle renvoie les listes `lx` et `ly` composés des premiers blocs rouges créés au début. Note : vous pourrez enlever le `pass` qui signifie "ne rien faire".

2) `bouge(taillex, tailley, lx, ly)` prend en argument les dimensions du jeu, ainsi que les listes `lx` et `ly`. Fait bouger les blocs rouges dans le jeu, et donc modifie les listes `lx` et `ly`.

3) `collision(lx, ly, x, y)` va tester si les blocs rouges sont en collision avec le bloc perso. Si oui, elle renverra un des indices des tableaux où cette collision a lieu, sinon elle renverra -1.
   Exemple : si `x = 3` et `y = 4`,
   si on a `lx = [2, 4, 3, 1]` et `ly = [5, 4, 4, 2]` alors il y a collision en position 2 (car `lx[2] = 3` et `ly[2] = 4`, égaux tous deux aux coordonnées du joueur). Il n'y a pas collision en 1 car seule le `ly[1]` est égal à `y` ! Ici il faut donc renvoyer 2.

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

Quelques explications sur la fonction jeu

La docstring doit être complétée par les règles de votre jeu.

Il y a déjà beaucoup de commentaires expliquant qui fait quoi, mais en voici le fonctionnement global :

Il y a une première partie initialisation du jeu, où on crée tout ce qu'il faut pour démarrer (coordonnées du joueur, blocs mobiles, etc). `continuer` est une variable qui vaut `True` au début et sert à rester dans la boucle principale de jeu (voir plus bas). À tout moment dans la boucle de jeu, si vous écrivez `continuer = False` la boucle pourra s'interrompre (par exemple quand on a perdu).

La boucle principale de jeu est un `while continuer` donc tant que la variable `continuer` est à `True` (vrai), la boucle s'exécute en continu. À la toute fin de cette boucle, on peut voir un `clock.tick(fps)`. Vous n'avez pas à le modifier, cela permet de faire en sorte que la boucle (qui pourrait s'exécuter en quelques millièmes de secondes car les machines sont rapides !) ne va pas aller plus vite que `fps` tours de boucle par seconde. Cela permet de garder une vitesse raisonnable pour le jeu.

À chaque tour de boucle (donc très vite), une fois que vous aurez tout complété, il se passera les choses suivantes :

— gérer les touches appuyées, modifier la position du joueur si besoin
— gérer les collisions entre le joueur et les blocs s'il y en a,
— gérer les blocs qui bougent si ils doivent bouger,
— afficher l'état du jeu avec `affiche_jeu`

Il est bien sûr conseillé d'écrire autant de fonctions annexes qui peuvent vous servir, afin de rendre votre code le plus clair et lisible possible !

Encore une fois, n'oubliez pas de commenter votre code... et de bien écrire les spécifications (docstring) des fonctions, ce sont deux choses différentes et importantes (qui seront prises en compte dans la notre finale) !

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.

Idées : différents types de blocs qui bougent avec des effets différents (certains positifs, d'autres négatifs ? ou différents effets), ...