LE GOUVERNEMENT
DU GRAND-DUCHÉ DE LUXEMBOURG
Ministère de l'Éducation nationale,
de l'Enfance et de la Jeunesse

EXAMEN DE FIN D'ÉTUDES SECONDAIRES CLASSIQUES
2020

BRANCHE	SECTION(S)	ÉPREUVE ÉCRITE
Informatique	**B**	*Durée de l'épreuve :* 3h *Date de l'épreuve :*

Un puzzle avec des tétrominos

Créez sur votre bureau un dossier nommé suivant les instructions du professeur. Le programme à écrire est appelé **`tetrominoes_puzzle.py`** et sera stocké dans ce dossier. En haut du code vous indiquerez en tant que commentaire votre lycée et votre numéro de candidat. A la fin de l'épreuve vous imprimerez et signerez le programme.

I. Initialisations (8 p.)

(1) Le programme **`tetrominoes_puzzle.py`** nécessite le chargement de la bibliothèque **`pygame`**. Effectuez les importations nécessaires. Les seules importations permises sont **`pygame, pygame.locals`** et **`sys`**. (1 p.)

(2) La fenêtre d'application mesure 601 x 571 pixels et porte l'entête :

`Tetrominoes Puzzle – LYCEE – NUMERO`.

`LYCEE` et **`NUMERO`** sont à remplacer respectivement par votre lycée et votre numéro de candidat. La surface de dessin **`screen`** devra être rafraîchie 15 fois par seconde. (1 p.)

(3) Ecrivez la fonction **`draw_puzzle_grid()`** qui permet de représenter sur fond blanc la grille de la figure 1. Les lignes verticales et horizontales, d'épaisseur 1 pixel, sont dessinées en couleur **`'darkgrey'`** aux pixels dont les abscisses ou ordonnées sont des multiples de 30. Le rectangle central, formé des 8x7=56 cellules centrales de la grille, est rempli avec la couleur **`'lightgrey'`**. (4 p.)

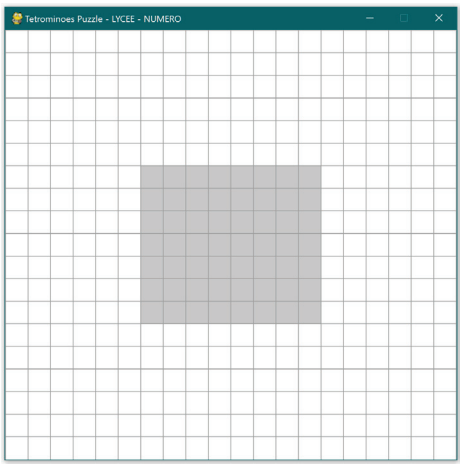
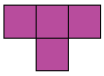
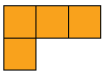
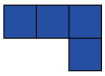
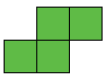
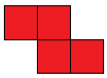


fig.1

(4) Un tétromino (terme combiné du préfixe « tetra » signifiant « quatre » en grec et du mot « domino ») est une figure géométrique composée de quatre carrés, chacun ayant au moins un côté complètement partagé avec un autre. A une rotation près, il y a sept types possibles, dont la forme et les propriétés sont présentées dans le tableau de la page suivante. (Les tétrominos sont utilisés par exemple dans le fameux jeu Tetris.)

Au cours de la programmation, on utilisera deux dictionnaires globaux et constants **`COLORS`** et **`SHAPES`**, contenant respectivement les couleurs et les matrices associées aux différents types de tétrominos. Définissez ces deux dictionnaires : **`COLORS`** contient les sept éléments de la forme **`kind:color`**, par exemple **`'I':Color('cyan')`** et **`SHAPES`** contient les sept éléments de la forme **`kind:shape`**, par exemple **`'I':[[1,1,1,1]]`**. (Ces dictionnaires ne devront pas être modifiés par la suite.) (2 p.)

Tétromino	Type (**kind**)	Couleur (**color**)	Matrice (**shape**)
	`'I'`	`Color('cyan')`	`[[1,1,1,1]]`
	`'O'`	`Color('yellow')`	`[[1,1], [1,1]]`
	`'T'`	`Color('magenta')`	`[[1,1,1], [0,1,0]]`
	`'L'`	`Color('orange')`	`[[1,1,1], [1,0,0]]`
	`'J'`	`Color('blue')`	`[[1,1,1], [0,0,1]]`
	`'S'`	`Color('green')`	`[[0,1,1], [1,1,0]]`
	`'Z'`	`Color('red')`	`[[1,1,0], [0,1,1]]`

Rappel : Une matrice **`m`** est implémentée dans Python sous la forme d'une liste dont les éléments sont également des listes, représentant les différentes lignes de la matrice. La ligne d'indice **`i`** de la matrice est donc **`m[i]`**. L'élément **`(i,j)`** de la matrice **`m`** (élément à l'intersection de la ligne d'indice **`i`** et de la colonne d'indice **`j`**) est **`m[i][j]`**.

Le ***puzzle*** qu'on se propose de résoudre consiste à remplir toutes les cellules du rectangle gris en utilisant exactement 2 tétrominos de chaque type (qui évidemment ne devront pas se recouper). Une solution possible est visible sur la figure 4.

II. La classe Tetromino (18 p.)

La classe **`Tetromino`** permet de créer, de manipuler et de représenter graphiquement des tétrominos.

```
Tetromino
    kind : str
    x : int
    y : int
    shape : list
    color : pygame.Color
__init__(self, kind, x, y)
get_cells(self)
move(self, dx, dy)
rotate(self)
draw(self)
```

(1) Chaque tétromino représenté sur la surface de dessin **`screen`** recouvre exactement quatre cellules de la grille de la figure 1. Les attributs **`x`** et **`y`** déterminent la position du tétromino sur cette grille : ce sont les coordonnées de la cellule en haut à gauche du tétromino (même pour un tétromino de type **`'S'`**, où cette cellule n'est pas occupée.)

Pour le tétromino de la figure 2 ci-dessous, on a par exemple : **`x=2`** et **`y=1`**.

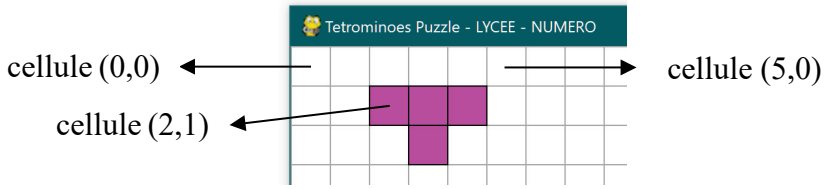


fig. 2

Le constructeur initialise les attributs **`kind`**, **`x`**, et **`y`** sur les valeurs passées aux paramètres correspondants. Les valeurs par défaut pour les coordonnées de la cellule en haut à gauche sont **`x=0`** et **`y=0`**. L'attribut **`kind`** est la lettre majuscule précisant le type du tétromino (cf. tableau p. 2) et doit être obligatoirement précisé à l'appel du constructeur. Les attributs **`shape`** (matrice associée) et **`color`** (couleur) sont initialisés grâce aux dictionnaires **`COLORS`** et **`SHAPES`**. (2 p.)

(2) La méthode **`get_cells(self)`** retourne la liste de tous les tuples de coordonnées des cellules effectivement occupées par le tétromino. Dans le cas du tétromino de la figure 2, cette méthode devra par exemple retourner **`[(2,1),(3,1),(4,1),(3,2)]`**. (5 p.)

(3) La méthode **`move(self,dx,dy)`** permet de déplacer le tétromino de **`dx`** cellules horizontalement et de **`dy`** cellules verticalement. **`dx`** et **`dy`** sont donc des paramètres de type **`int`** et leur signe détermine si le déplacement se fait vers la droite ou vers la gauche, respectivement vers le bas ou vers le haut. ***Remarque*** : Le tétromino est autorisé à quitter la partie visible de la surface de dessin. (2 p.)

(4) La méthode **`rotate(self)`** modifie la matrice **`shape`** de telle sorte qu'elle tourne de 90° dans le sens contraire des aiguilles d'une montre (sens positif).

Par exemple : si initialement on a

```
shape =    [
            [a, b, c],
            [d, e, f]
           ],
```

alors l'appel de la méthode modifie **`shape`** en :

```
shape =    [
            [c, f],
            [b, e],
            [a, d]
           ]
```

Indication : La 1re ligne de la matrice tournée (en rouge) est la dernière colonne de la matrice initiale, la 2e ligne de la matrice tournée (en bleu) est l'avant-dernière colonne de la matrice initiale, etc.

N.B. : L'algorithme utilisé devra fonctionner pour toute matrice bien définie, c.-à-d. ayant au moins une ligne et une colonne non vides. (6 p.)

(5) La méthode **`draw(self)`** permet de représenter graphiquement les quatre cellules occupées par le tétromino sur la grille de la surface de dessin. Chaque cellule du tétromino est remplie avec la couleur **`color`**. Le bord d'une cellule, de couleur noire et d'épaisseur 1 pixel, doit coïncider avec le bord de la cellule correspondante de la grille. ***Indication*** : Utilisez la méthode **`get_cells()`**. (3 p.)

III. La classe TetrominoesList (12 p.)

La classe **`TetrominoesList`** permet de gérer et de représenter une liste de tétrominos.

`TetrominoesList`
`items : list` **`selected_item : Tetromino`** ou **`NoneType`** **`counts : dict`**
`__init__(self)` **`add(self, t)`** **`remove(self, t)`** **`get_item_containing(self, x, y)`** **`draw(self)`**

(1) Les attributs de la classe doivent être initialisés par le constructeur :

- **`items`** est la liste de tous les tétrominos (de type **`Tetromino`**) créés et utilisés durant l'exécution. Initialement cette liste est vide.
- **`selected_item`** est l'élément de la liste **`items`** que le joueur est en train de manipuler. Initialement cet élément n'est pas encore défini, donc on lui affectera la valeur **`None`**.
- **`counts`** est un dictionnaire permettant de compter combien de tétrominos de chaque type sont actuellement dans la liste **`items`**. Les clés de ce dictionnaire sont à nouveau les sept majuscules correspondant aux différents types de tétrominos. Les valeurs associées, initialement 0, comptent le nombre de tétrominos existants du type correspondant. Pensez à actualiser ce dictionnaire si nécessaire. (2 p.)

(2) La méthode **`add(self,t)`** ajoute le tétromino **`t`** à la liste **`items`**. De plus **`t`** sera affecté à l'attribut **`selected_item`**. (2 p.)

(3) La méthode **`remove(self,t)`** enlève le tétromino **`t`** de la liste **`items`**, s'il y est présent. Dans ce cas, **`selected_item`** est à nouveau indéfini. (2 p.)

(4) La méthode **`get_item_containing(self,x,y)`** détermine le premier tétromino de la liste **`items`** qui contient la cellule de coordonnées **`x`** et **`y`**, si un tel tétromino existe. Ce tétromino sera alors affecté à l'attribut **`selected_item`** et retourné par la méthode. Sinon la méthode ne retourne rien. (3 p.)

(5) La méthode **`draw(self)`** permet de dessiner tous les tétrominos de la liste **`items`**. Le tétromino **`selected_item`**, s'il existe, doit être dessiné en dernier lieu. ***Remarque*** : Un tétromino peut glisser sur un autre lorsqu'il se déplace. Comme le tétromino **`selected_item`** est dessiné en dernier, il glisse au-dessus des autres tétrominos (voir figure 3). (3 p.)

IV. Le programme principal (22 p.)

(1) **Préparation** : Créez une instance **`tetrominoes`** de la classe **`TetrominoesList`**. Initialisez éventuellement d'autres variables utilisées dans la suite. (1 p.)

(2) **Boucle principale – Réactions aux événements :**

a) Lorsqu'on clique sur la croix de fermeture, le programme se termine correctement. (1 p.)

b) Lorsqu'on enfonce l'une des touches **`'i'`**, **`'o'`**, **`'t'`**, **`'l'`**, **`'j'`**, **`'s'`**, ou **`'z'`** du clavier, un nouveau tétromino de ce type est créé avec les coordonnées par défaut (c.-à-d. dans le coin supérieur gauche de la grille). Ce tétromino doit être ajouté à la liste de **`tetrominoes`**. ***Attention*** : Il est fort probable que la constante **`K_z`** n'a pas l'effet désiré, remplacez-la dans ce cas par **`K_y`**. (2 p.)

c) Un tétromino créé ou sélectionné (c'est toujours l'élément **`selected_item`** de la liste des tétrominos) peut être déplacé sur l'écran grâce aux touches de direction. Si l'une des quatre touches est enfoncée, le tétromino se déplace d'une cellule dans la direction correspondante. Le mouvement de translation doit se poursuivre aussi longtemps que la touche correspondante reste enfoncée. (4 p.)

d) Lorsqu'on appuie sur la touche **`'SPACE'`**, le tétromino est tourné de 90° dans le sens positif. Pour tourner le tétromino de 180° (resp. de 270° etc.), il faudra appuyer 2 fois (resp. 3 fois etc.) sur **`'SPACE'`**. Maintenir cette touche enfoncée ne permet pas de répéter le mouvement de rotation. (2 p.)

e) Lorsqu'on clique avec le bouton gauche de la souris à l'intérieur d'un tétromino, ce tétromino est sélectionné. (2 p.)

f) Lorsqu'on clique avec le bouton droit de la souris à l'intérieur d'un tétromino, ce tétromino est effacé de la liste de **`tetrominoes`** et disparaît de l'écran. (2 p.)

Par exemple, sur la figure 3, le joueur a déjà créé, déplacé et tourné un certain nombre de tétrominos. Le **`'J'`** bleu qui recouvre en partie un **`'I'`** vient d'être déplacé en dernier lieu. La figure 4 montre une solution possible du puzzle.

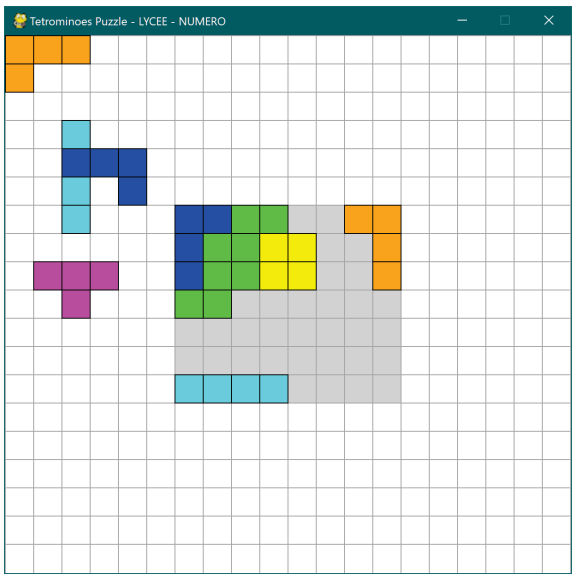


fig. 3

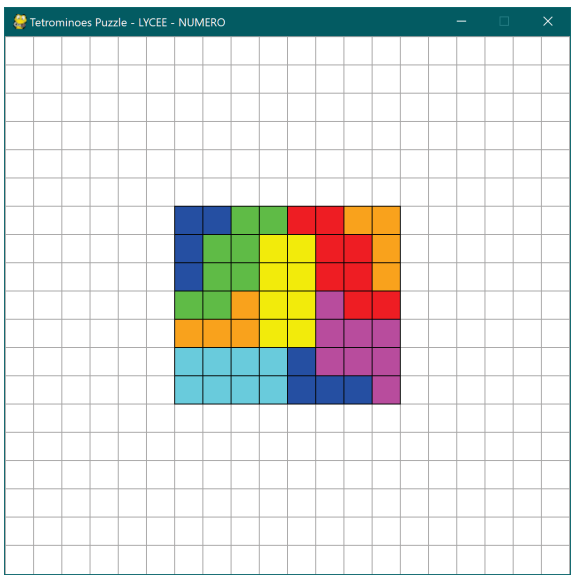


fig. 4

g) Lorsqu'on enfonce la touche **`'c'`** du clavier, le programme vérifie si l'utilisateur a correctement résolu le puzzle. Si c'est le cas, le message **`'CORRECT !'`** est affiché dans la console, sinon le message **`'WRONG !'`** est affiché.

Pour cela, on demande d'écrire et d'utiliser une fonction **`check_puzzle()`** qui retourne **`True`** si le puzzle est correctement résolu, **`False`** sinon. La fonction doit tester d'abord si le nombre de tétrominos dessinés de chaque type est égal à 2. Sinon elle retourne déjà **`False`**. Ensuite il faudra tester si l'ensemble des cellules formant les tétrominos dessinés occupent bien toutes les cellules du rectangle central (et pas d'autres cellules). (6 p.)

(3) **Boucle principale – Actions répétées à chaque itération :**
A chaque passage dans la boucle principale, la grille et tous les tétrominos sont dessinés et l'horloge avance d'un tic. (2 p.)