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

CORRIGÉ – BARÈME

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

```python
# I. Initialisations (8 p.)

import pygame, sys  # 1 p.
from pygame.locals import *

pygame.init()  # 1 p.
screen = pygame.display.set_mode((601, 571))
pygame.display.set_caption("Tetrominoes Puzzle - LYCEE - NUMERO")
clock = pygame.time.Clock()
FPS = 15


def draw_puzzle_grid():  # 4 p.
    screen.fill(Color('white'))
    pygame.draw.rect(screen, Color('lightgrey'), (180, 180, 240, 210))
    for x in range(21):
        pygame.draw.line(screen, Color('darkgrey'), (x * 30, 0), (x * 30, 570), 1)
    for y in range(19):
        pygame.draw.line(screen, Color('darkgrey'), (0, y * 30), (600, y * 30), 1)


# 2 p.
COLORS = {
    'I': Color('cyan'),
    'O': Color('yellow'),
    'T': Color('magenta'),
    'L': Color('orange'),
    'J': Color('blue'),
    'S': Color('green'),
    'Z': Color('red')
}

SHAPES = {
    'I': [[1, 1, 1, 1]],
    'O': [[1, 1], [1, 1]],
    'T': [[1, 1, 1], [0, 1, 0]],
    'L': [[1, 1, 1], [1, 0, 0]],
    'J': [[1, 1, 1], [0, 0, 1]],
    'S': [[0, 1, 1], [1, 1, 0]],
    'Z': [[1, 1, 0], [0, 1, 1]]
}


# II. La classe Tetromino (18 p.)

class Tetromino:
    def __init__(self, kind, x=0, y=0):  # 2 p.
        self.x, self.y = x, y
        self.kind = kind
        self.shape = SHAPES[kind]
        self.color = COLORS[kind]
```

```
    def get_cells(self):  # 5 p.
        row_count = len(self.shape)
        col_count = len(self.shape[0])
        cells = []
        for i in range(row_count):
            for j in range(col_count):
                if self.shape[i][j] == 1:
                    cells.append((self.x + j, self.y + i))
        return cells

    def move(self, dx, dy):  # 2 p.
        self.x += dx
        self.y += dy

    def rotate(self):  # 6 p.
        row_count = len(self.shape)
        col_count = len(self.shape[0])
        columns = []
        for j in range(col_count):
            columns.append([self.shape[i][j] for i in range(row_count)])
        columns.reverse()
        self.shape = columns

    def draw(self):  # 3 p.
        cells = self.get_cells()
        for x, y in cells:
            pygame.draw.rect(screen, self.color,(x * 30, y * 30, 31, 31))
            pygame.draw.rect(screen, Color('black'),(x * 30, y * 30, 31, 31), 1)


# III. La classe TetrominoesList (12 p.)

class TetrominoesList:
    def __init__(self):  # 2 p.
        self.items = []
        self.selected_item = None
        self.counts = {kind: 0 for kind in 'IOTLJSZ'}

    def add(self, t):  # 2 p.
        self.items.append(t)
        self.selected_item = t
        self.counts[t.kind] += 1

    def remove(self, t):  # 2 p.
        if t in self.items:
            self.items.remove(t)
            self.selected_item = None
            self.counts[t.kind] -= 1

    def get_item_containing(self, x, y):  # 3 p.
        for t in self.items:
            if (x, y) in t.get_cells():
                self.selected_item = t
                return t

    def draw(self):  # 3 p.
        for t in self.items:
            if self.selected_item != t:
                t.draw()
        if self.selected_item != None:
            self.selected_item.draw()
```

```
# IV Le programme principal (22 p.)

# g) fonction puzzle_check 5 p.
# (1 p. supplémentaire sur l'appel)

def puzzle_check():
    for kind in 'IOTLJSZ':
        if tetrominoes.counts[kind] != 2:
            return False
    puzzle_cells = {(x, y): False for x in range(6, 14) for y in range(6, 13)}
    for t in tetrominoes.items:
        for c in t.get_cells():
            if c in puzzle_cells:
                puzzle_cells[c] = True
            else:
                return False
    return not (False in puzzle_cells.values())


tetrominoes = TetrominoesList()  # 1 p.
key_pressed = KMOD_NONE
done = False
while not done:
    for event in pygame.event.get():
        if event.type == QUIT:  # 1 p.
            done = True

        elif event.type == KEYDOWN:  # 2 p.
            if event.key == K_i:
                tetrominoes.add(Tetromino('I'))
            elif event.key == K_o:
                tetrominoes.add(Tetromino('O'))
            elif event.key == K_t:
                tetrominoes.add(Tetromino('T'))
            elif event.key == K_l:
                tetrominoes.add(Tetromino('L'))
            elif event.key == K_j:
                tetrominoes.add(Tetromino('J'))
            elif event.key == K_s:
                tetrominoes.add(Tetromino('S'))
            elif event.key == K_y:  # si K_z ne fonctionne pas
                tetrominoes.add(Tetromino('Z'))
            elif event.key == K_c:  # 1 p.
                if puzzle_check():
                    print('CORRECT !')
                else:
                    print('WRONG !')
            else:
                key_pressed = event.key

        elif event.type == KEYUP:
            key_pressed = KMOD_NONE

        elif event.type == MOUSEBUTTONDOWN:  # 2+2 p.
            x, y = pygame.mouse.get_pos()
            t = tetrominoes.get_item_containing(x // 30, y // 30)
            if pygame.mouse.get_pressed() == (False, False, True) and t != None:
                tetrominoes.remove(t)

    t = tetrominoes.selected_item  # 4 p. (1 p. pour le "key repeat")
    if t != None:
        if key_pressed == K_RIGHT:
            t.move(1, 0)
        if key_pressed == K_LEFT:
```

```
            t.move(-1, 0)
        if key_pressed == K_DOWN:
            t.move(0, 1)
        if key_pressed == K_UP:
            t.move(0, -1)
        if key_pressed == K_SPACE:  # 2 p.
            t.rotate()
            key_pressed = KMOD_NONE

    draw_puzzle_grid()  # 2 p.
    tetrominoes.draw()
    pygame.display.update()
    clock.tick(FPS)

pygame.quit()
sys.exit()
```