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 :* 180 minutes *Date de l'épreuve :*

1 Programme `vector3.py`

```
from math import sqrt

class Vec3: # C: 15 p.
    def __init__(self, x = 0, y = 0, z = 0): # 2 p.
        self.x = x
        self.y = y
        self.z = z
    def __str__(self): # 2 p.
        return f"( {self.x} {self.y} {self.z} )"
    def dot(self, v): # 2 p.
        return self.x * v.x + self.y * v.y + self.z * v.z
    def norm(self): # 2 p.
        return sqrt(self.dot(self))
    def cross(self, v): # 3 p.
        vc = Vec3()
        vc.x = self.y * v.z - self.z * v.y
        vc.y = self.z * v.x - self.x * v.z
        vc.z = self.x * v.y - self.y * v.x
        return vc
    def colin(self, v): # 2 p.
        return self.cross(v).norm() < 1e-9
    def det(self, v, w): # 2 p.
        return self.cross(v).dot(w) # le déterminant de 3 vecteurs est le produit mixte de ces vecteurs

# programme principal, C: 5 p.
v1 = Vec3(3, 2, 1)
v2 = Vec3(-4, 5, 6)
v3 = Vec3(0.3, 0.2, 0.1)
print(f"v1 = {v1}, v2 = {v2}, v3 = {v3}")
print("Produit scalaire : v1 . v2 =", v1.dot(v2))
print("Norme de v1 :", v1.norm())
print("Produit vectoriel : v1 x v2 =", v1.cross(v2))
print("v1 et v3 sont-ils colinéaires ?", v1.colin(v3))
print("det(v1, v2, v3) =", v1.det(v2, v3))
```

2 Programme `knight.py`

```
1  import pygame, sys
2  from pygame.locals import *
3  from random import shuffle
4
5  def draw_board(s): # C: 4 p.
6      s.fill(Color("white")) # 1 p.
7      pygame.draw.rect(s, Color("black"), (9, 9, 402, 402), 1) # 1 p. (si l'emplacement est exact)
8      for x in range(8):
9          for y in range(8):
10             if (x + y) % 2 == 1: # 1 p. si l'alternance des couleurs est correcte
11                 pygame.draw.rect(s, Color("black"), (10+50*x, 10+50*y, 50, 50), 0) # 1 p. pour le
                       dessin d'un carré
12
13 def draw_disc(s, x, y, c): # C: 2 p.
14     pygame.draw.circle(s, c, (35+50*x, 35+50*y), 25, 0)
15
16 def draw_square(s, x, y): # C: 2 p.
17     pygame.draw.rect(s, Color("yellow"), (28+50*x, 28+50*y, 15, 15), 0)
18
19 def knight_moves(p): # C: 5 p.
20     (x, y) = p
21     moves = []
22     for (dx, dy) in [(-1,-2), (1,-2), (-2,-1), (2,-1), (-2,1), (2,1), (-1,2), (1,2)]: # 3 p. pour décrire
           les 8 directions
23         if 0 <= x+dx < 8 and 0 <= y+dy < 8: # 1 p. pour le contrôle des bords
24             moves.append((x+dx, y+dy))
25     shuffle(moves) # pour permuter au hasard les directions et obtenir ainsi tous les chemins possibles
26     return moves
27
28 def shortest_path(p1, p2): # C: 7 p.
29     # 2 p. si l'élève écrit une version simplifiée qui trouve uniquement les chemins de longueur 1 (1 seul
           saut)
30     # 5 p. si l'élève écrit une version simplifiée qui trouve uniquement les chemins de longueur 1 et 2
31     # bien sûr ces deux crédits partiels ne sont pas cumulables
32     visited = { p1:None } # case de départ
33     while p2 not in visited: # 5 p. pour la recherche du chemin
34         for k in visited.copy():
35             for m in knight_moves(k):
36                 if m not in visited:
37                     visited[m] = k # le cavalier peut sauter de k à m
38     path = [p2]
39     while path[0] != p1: # 2 p. pour la construction de la liste
40         path.insert(0, visited[path[0]]) # reconstitution de la fin au début
41     return path
42
43 def draw_path(s, p1, p2): # C: 7 p.
44     if p1 == None:
45         return
46     (x1, y1) = p1
47     draw_disc(s, x1, y1, Color("blue")) # 1 p.
48     if p2 == None:
```

```
49          return
50      (x, y) = p2
51      draw_disc(s, x, y, Color("red")) # 1 p.
52      path = shortest_path(p1, p2)
53      for (x, y) in path[1:-1]:
54          draw_square(s, x, y) # 2 p.
55      for (x2, y2) in path[1:]:
56          pygame.draw.line(s, Color("green"), (35+50*x1, 35+50*y1), (35+50*x2, 35+50*y2), 3) # 3 p.
57          (x1, y1) = (x2, y2)
58
59  # programme principal, C: 13 p.
60  start_pos = None
61  end_pos = None
62  pygame.init()
63  size = (420, 420)
64  screen = pygame.display.set_mode(size)
65  draw_board(screen)
66  game_over = False # 2 p. pour toutes les initialisations
67  pygame.display.set_caption("Sauts de cavalier") # en-tête: 1 p.
68
69  while not game_over: # 1 p. pour le mécanisme de la boucle principale (y compris la fin du programme)
70      pygame.display.update()
71      for event in pygame.event.get():
72          if event.type == QUIT:
73              game_over = True # 1 p. pour le mécanisme de sortie
74          if event.type == MOUSEBUTTONDOWN: # 2 p. pour l'étude des coordonnées de la souris
75              (mx, my) = pygame.mouse.get_pos()
76              x = (mx - 10) // 50
77              y = (my - 10) // 50
78              if 0 <= x < 8 and 0 <= y < 8: # 6 p. pour les réactions du programme
79                  if start_pos == None:
80                      start_pos = (x, y)
81                  elif start_pos == (x, y):
82                      start_pos = None
83                      end_pos = None
84                  else:
85                      end_pos = (x, y)
86                  draw_board(screen) # méthode brute : on redessine tout
87                  draw_path(screen, start_pos, end_pos)
88
89  pygame.quit()
90  sys.exit()
```