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 :*	*180 minutes*
		Date de l'épreuve :	

Ajouter le nom du lycée et le numéro personnel sous forme de commentaire tout au début de chaque code-source ! Enregistrer les deux fichiers `.py` demandés dans votre répertoire de travail.

Les seules importations permises sont celles de `pygame`, `pygame.locals`, `sys`, `math` et `random`.

Exercice 1 : Vecteurs à trois dimensions (20 points)

Écrire un programme `vector3.py` qui implémente une classe `Vec3` représentant des vecteurs dans l'espace à 3 dimensions, muni d'une base orthonormée directe :

1. Le constructeur permet de construire un vecteur $\vec{u}\begin{pmatrix} x \\ y \\ z \end{pmatrix}$ à partir de ses composantes réelles x, y, z. La valeur par défaut pour chacune des composantes est 0. **[2 p.]**

2. La méthode `__str__` renvoie le vecteur sous la forme (x y z). Les parenthèses et les coordonnées sont séparées par une espace. **[2 p.]**

3. La méthode `dot` renvoie le *produit scalaire* du vecteur en question et d'un 2e vecteur passé comme argument. **[2 p.]**

   Rappel : $\vec{u}\begin{pmatrix} x \\ y \\ z \end{pmatrix} \cdot \vec{v}\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = xx' + yy' + zz'$

4. La méthode `norm` renvoie la *norme* du vecteur en question. **[2 p.]**

   Rappel : $\|\vec{u}\| = \sqrt{\vec{u} \cdot \vec{u}}$

5. La méthode `cross` renvoie le *produit vectoriel* du vecteur en question et d'un 2e vecteur passé comme argument. **[3 p.]**

   Rappel : $\vec{u}\begin{pmatrix} x \\ y \\ z \end{pmatrix} \times \vec{v}\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = (\vec{u} \times \vec{v})\begin{pmatrix} yz'-y'z \\ zx'-z'x \\ xy'-x'y \end{pmatrix}$

6. La méthode `colin` renvoie `True` si le vecteur en question et un 2e vecteur passé comme argument sont *colinéaires*, et `False` sinon. **[2 p.]**

   Rappel : deux vecteurs sont colinéaires si et seulement si l'un s'écrit comme multiple de l'autre.

7. La méthode `det` renvoie le *déterminant* du vecteur en question et de deux vecteurs supplémentaires passés comme arguments. **[2 p.]**

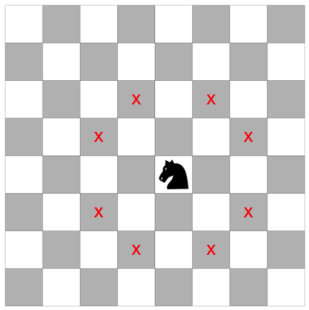
8. Écrire un programme principal qui appelle chacune des méthodes de la classe une fois (au moins) et affiche les résultats obtenus ; voir l'affichage suivant. Voici les vecteurs à utiliser : $\vec{v}_1\begin{pmatrix}3\\2\\1\end{pmatrix}$, $\vec{v}_2\begin{pmatrix}-4\\5\\6\end{pmatrix}$, $\vec{v}_3\begin{pmatrix}0{,}3\\0{,}2\\0{,}1\end{pmatrix}$. **[5 p.]**

Voici l'affichage après l'exécution du programme :

```
v1 = ( 3 2 1 ), v2 = ( -4 5 6 ), v3 = ( 0.3 0.2 0.1 )
Produit scalaire : v1 . v2 = 4
Norme de v1 : 3.7416573867739413
Produit vectoriel : v1 x v2 = ( 7 -22 23 )
v1 et v3 sont-ils colinéaires ? True
det(v1, v2, v3) =  0.0
```

Exercice 2 : Sauts de cavalier sur un échiquier (40 points)

Au jeu d'échecs, le **cavalier** (allemand : *Springer* ; anglais : *knight*) se déplace en L, c'est-à-dire de deux cases dans une direction, suivies d'une case dans la direction perpendiculaire, sans pouvoir quitter l'échiquier :



Le but du programme est de déterminer le(s) chemin(s) le(s) plus court(s) reliant deux cases de l'échiquier choisies par l'utilisateur, par des sauts de cavalier.

L'**écran** ouvert par `pygame` a comme dimensions 420×420 pixels. En son centre, un échiquier de 400×400 pixels est représenté, dont chacune des 8×8 cases mesure 50×50 pixels. L'échiquier est entouré d'un cadre (carré) noir d'épaisseur 1 pixel. L'arrière-fond extérieur à l'échiquier est blanc. L'en-tête de la fenêtre contient à tout moment le texte « Sauts de cavalier ».

Écrire un programme `knight.py`, comme suit :

1. La fonction `draw_board` reçoit comme argument l'écran. Elle dessine un échiquier vide, semblable à la 1re capture d'écran, avec 32 cases noires et autant de cases blanches. La case en haut à gauche est obligatoirement blanche. **[4 p.]**

2. La fonction `draw_disc` reçoit comme arguments l'écran, les numéros de colonne et de ligne d'une case de l'échiquier et une couleur. Elle dessine, en utilisant la couleur indiquée, un disque de diamètre 50 pixels, centré dans la case indiquée. **[2 p.]**

3. La fonction `draw_square` reçoit comme arguments l'écran et les numéros de colonne et de ligne d'une case de l'échiquier. Elle dessine en couleur jaune un carré rempli de côté 15 pixels, centré dans la case indiquée. **[2 p.]**

4. La fonction `knight_moves` reçoit comme argument un couple de coordonnées représentant la case de départ du cavalier. Elle génère et renvoie une liste avec les couples de coordonnées des cases vers lesquelles le cavalier peut sauter de sa case de départ en un seul coup. La liste contient donc entre 2 et 8 couples ; ces couples doivent y figurer en **ordre aléatoire**, c'est-à-dire des appels successifs de la fonction avec le même argument donnent des listes ordonnées différemment, en fonction du hasard. **[5 p.]**

   Indication : Il est permis d'établir d'abord la liste de façon déterministe et de la permuter à la fin pour obtenir un ordre aléatoire.

5. La fonction `shortest_path` reçoit comme arguments deux couples p_1 et p_2 de coordonnées représentant deux cases distinctes de l'échiquier. Elle génère et renvoie une liste contenant les couples de coordonnées d'un des chemins les plus courts, de sauts de cavalier menant de p_1 vers p_2. Le 1$^{\text{er}}$ élément de cette liste sera p_1, et le dernier élément sera p_2. Lorsqu'il existe plusieurs chemins qui sont les plus courts, la fonction renvoie l'un d'eux au hasard. **[7 p.]**

   Indication : Utiliser une variable (une liste ou un dictionnaire) qui, pour chaque case de l'échiquier, contiendra la case précédente du chemin le plus court qui la relie à la position initiale p_1. Remplir la variable successivement à l'aide d'appels de `knight_moves`, en considérant successivement des chemins de longueur $1, 2, 3, \ldots$ jusqu'à ce que la case p_2 y soit incluse. La variable permet alors de reconstruire le chemin cherché.

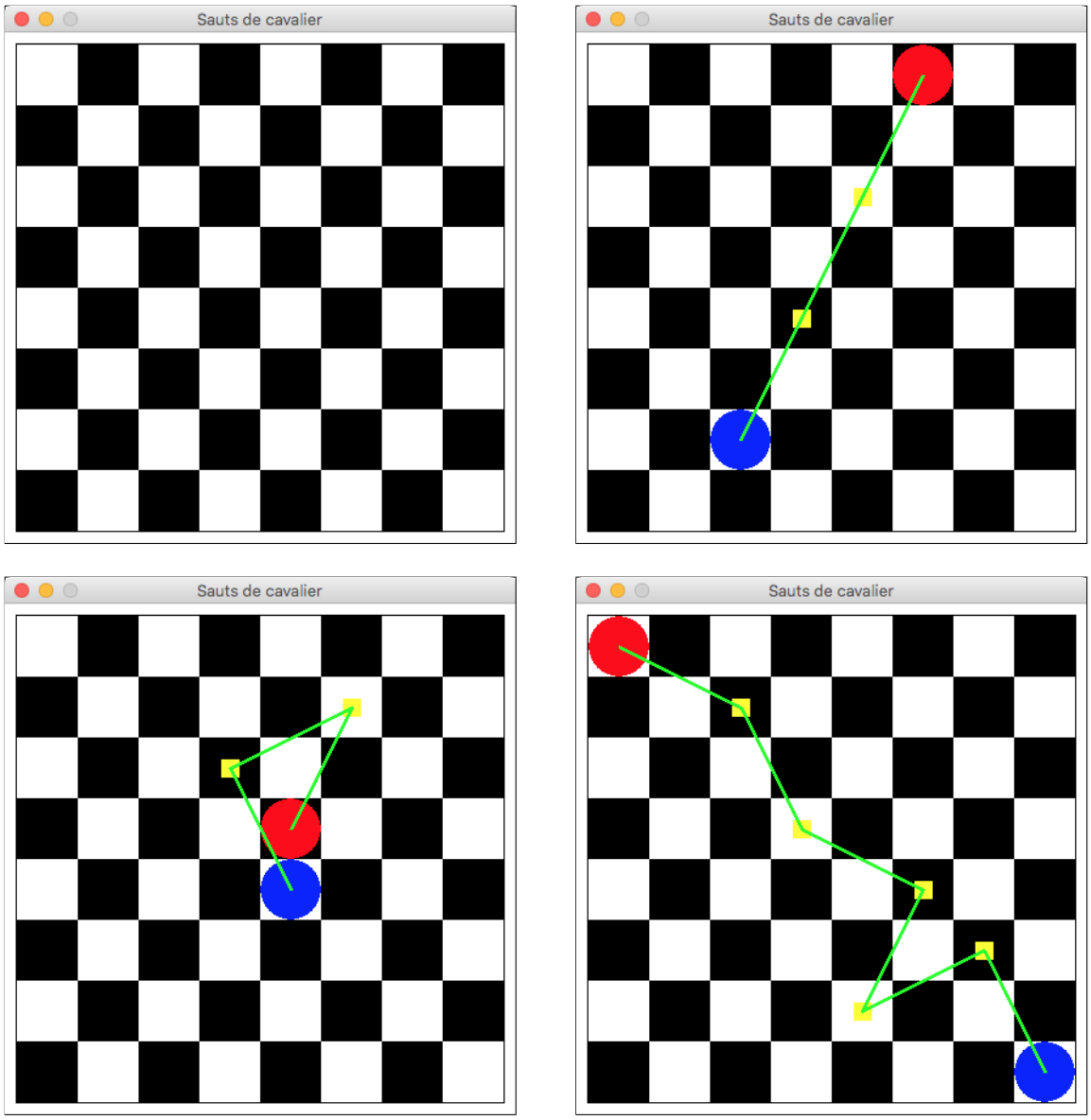
   Crédit partiel accordé : **2 p.** si les chemins de longueur 1 sont tous correctement construits ; **5 p.** (au total) si les chemins de longueur 1 ou 2 sont tous correctement construits.

6. La fonction `draw_path` reçoit comme arguments l'écran, les couples de coordonnées éventuelles de la case de départ et les couples de coordonnées éventuelles de la case d'arrivée. La fonction appelle les fonctions précédentes pour dessiner un disque bleu dans la case de départ (si elle existe), un disque rouge dans la case d'arrivée et l'un des chemins les plus courts entre ces cases (lorsque la case d'arrivée existe). Le chemin est représenté par des segments droits reliant les étapes successives, de couleur verte et d'épaisseur 3 pixels. Les cases intermédiaires, visitées par le cavalier, sont marquées par des carrés de couleur jaune. **[7 p.]**

7. Boucle principale (pouvant être quittée par l'événement `QUIT`) : **[13 p.]**

   - ⋆ Si l'utilisateur clique (événement `MOUSEBUTTONDOWN`) dans une case de l'**échiquier vide**, la case de départ est ainsi définie.
   - ⋆ Si l'utilisateur clique dans une case **différente** de la case de départ déjà définie, la case d'arrivée est ainsi définie ou redéfinie : l'ancienne case d'arrivée (si elle existait déjà) est remplacée par la nouvelle.
   - ⋆ Si l'utilisateur clique à nouveau dans la **case d'arrivée déjà définie**, la case d'arrivée reste inchangée, mais comme le chemin le plus court est choisi au hasard, souvent un autre chemin est affiché.
   - ⋆ Si l'utilisateur clique dans la **case de départ déjà définie**, l'échiquier est vidé (réinitialisé) et une nouvelle case de départ pourra être choisie par le clic suivant.

   Après chaque clic dans l'échiquier, la fonction `draw_path` redessine tout ce qui existe au moment de son appel.



- ⋆ En haut à gauche : après le démarrage du programme (ou après un clic dans la case de départ déjà définie auparavant).
- ⋆ En haut à droite : l'unique chemin le plus court (3 sauts) entre les cases C2 et F8.
- ⋆ En bas à gauche : l'un des 12 chemins les plus courts (3 sauts) entre les cases E4 et E5.
- ⋆ En bas à droite : l'un des 108 chemins les plus courts (6 sauts) entre les cases H1 et A8.