DM de vacances

I. Introduction

Ce DM d'informatique a pour objectif de vous préparer à la rentrée. Il est conseillé de réalisé ce DM petit à petit plutôt qu'en une seule fois pour avoir le temps de réfléchir aux questions. Ce DM devra être rendu au premier cours d'informatique. Vous pouvez choisir l'une des trois pistes suivantes (de la moins rémunératrice à la plus rémunératrice) :

- Piste verte : Partie III, VI et VII pour les élèves en difficultés.
- Piste bleue : Partie III, IV, VI et VII pour les élèves de niveau standard.
- Piste rouge : en entier, pour les élèves à l'aise en informatique ou souhaitant se confronter à la difficulté

Il vous faudra importer les modules suivants.

```
1 from collections import deque
2 import matplotlib.pyplot as plt
3 import random as rd
4 import networkx as nx
5 import numpy as np
```

On rappelle qu'il faut toujours donner la signature d'une fonction. Par exemple, si l'on demande de coder une fonction `maFonction` qui prend en argument un entier `n` et une liste `L` et qui renvoie une chaîne de caractères `c`, on écrira l'en-tête suivante.

```
1 def maFonction(n, L) :
2     ''maFonction(n : int, L : list) -> c : str''
```

Il est fortement recommandé d'utiliser les fonctions mises en place précédemment pour répondre à certaines questions.
Toutes les fonctions et structures de données demandées sont supposées viables pour la suite des questions. Il n'est donc pas du tout nécessaire de les avoir codées pour pouvoir les utiliser.

Il suit du paragraphe précédent que la quasi-totalité des questions sont indépendantes (les seules exceptions étant les questions portant sur la complexité temporelle des fonctions).

Une Annexe est présente à la fin du sujet pour vous aider (documentations, complexités, définitions, etc).

II. Présentation du jeu

Le jeu *Serpents et échelles* est un jeu de société où on espère monter les échelles en évitant de trébucher sur les serpents. Il provient d'Inde et est utilisé pour illustrer l'influence des vices et des vertues sur une vie.

Le plateau

- Le plateau comporte 100 cases numérotées de 1 à 100 en boustrophédon (1) : le 1 est en bas à gauche et le 100 est en haut à gauche ;
- des serpents et des échelles sont présentes sur le plateau : les serpents font descendre un joueur de sa tête à sa queue, les échelles font monter un joueur du bas de l'échelle vers le haut.

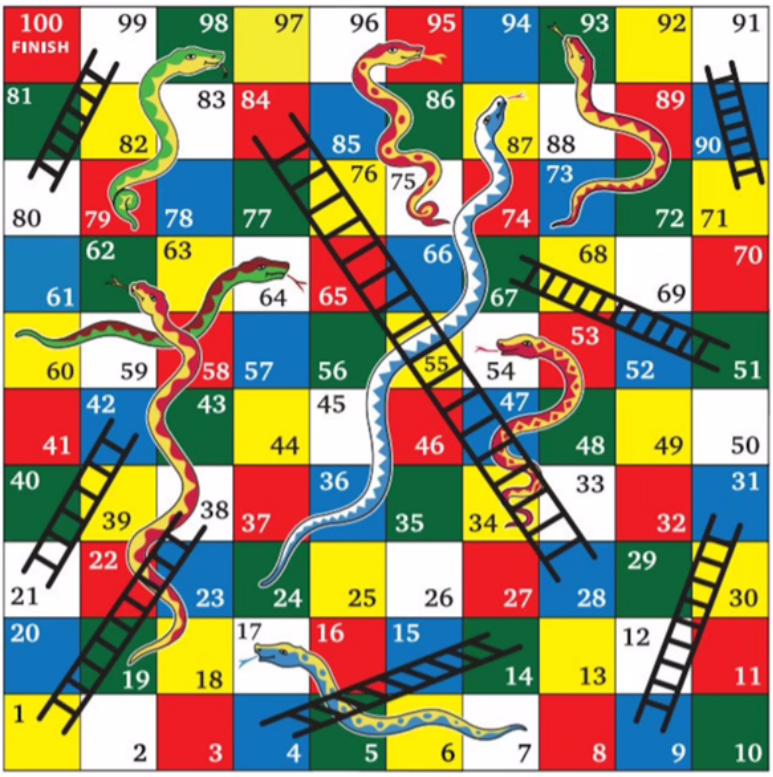

Fig. 1 Exemple d'un plateau de serpents et d'échelles.

Déroulement d'une partie

- Chaque joueur a un pion sur le plateau. Plusieurs pions peuvent être sur une même case. Les joueurs lancent un dé à tour de rôle et ils avancent du nombre de cases marqués sur le dé. S'ils aterrissent sur un bas d'échelle ou sur une tête de serpent, ils vont directement à l'autre bout ;
- les joueurs commencent sur une case 0 hors du plateau. La première case où mettre leur pion correspond donc au numéro obtenu au premier lancer de dé ;
- les premier joueur à arriver sur la case 100 a gagné ;
- il existe 3 variantes quand la somme de la case actuelle et du dé dépasse 100 :
  - × le rebond : on recule d'autant de cases qu'on dépasse ;
  - × l'immobilisme : on n'avance pas du tout si on dépasse ;
  - × la fin rapide : on va à la case 100 quoiqu'il arrive.

On utilisera les notations suivantes pour les complexités : N_{cases}, le nombre de cases du plateau (100), et N_{SE} la somme du nombre de serpents et du nombre d'échelles (16 dans notre exemple). Ces variables ne sont pas déclarées dans le script.

(1). À la manière du boeuf traçant des sillons, avec alternance gauche-droite et droite-gauche.

III. Simulation d'une partie

1. Écrire une fonction `lancerDe` qui ne prend pas d'argument et renvoie un nombre entier compris entre 1 et 6 en utilisant une fonction du module `random`.
On pourra utiliser l'Annexe.

Les serpents et les échelles seront, dans un premier temps, représentés par une liste `LSE` de 2 listes. Plus précisément, pour un indice `i` donné :

× `LSE[0][i]` contient le numéro de la case de départ d'un symbole (tête du serpent ou bas de l'échelle) ;

× `LSE[1][i]` contient le numéro de la case d'arrivée du symbole (queue du serpent ou haut de l'échelle).

Par exemple, pour la Figure **??**, on obtient :

```
LSE  =  [[ 1, 4, 9, 17, 21, 28, 51, 54, 62, 64, 71, 80, 87, 93, 95, 98],
         [38, 14, 31, 7, 42, 84, 67, 34, 19, 60, 91, 99, 24, 73, 75, 79]]
```

On suppose que cette variable est accessible depuis toutes les futures fonctions, c'est-à-dire qu'elle est déjà déclarée dans le script.

2. Écrire une fonction `caseFuture1` qui prend en argument un entier `case` représentant le numéro de la case où atterit le joueur après son lancer de dé, et qui renvoie le numéro de la case où il va se trouver à la fin de son tour.
Par exemple :

× `caseFuture1(5)` doit renvoyer 5, car c'est un numéro de case stable ;

× `caseFuture1(1)` doit renvoyer 38, car le joueur atterrit alors au pied d'une échelle ;

× `caseFuture1(17)` doit renvoyer 7, car le joueur atterrit alors à la tête d'un serpent.

3. Quelle est la complexité de la fonction `caseFuture1` ?

Les serpents et les échelles sont maintenant, et pour toute la suite, représentés par un dictionnaire `dSE`. Plus précisément, pour chaque numéro de case de départ numérotée `i`, l'élément `dSE[i]` contient le numéro de la case d'arrivée.

Par exemple, avec la Figure **??**, on obtient :

```
dSE  =  { 1 : 38, 4 : 14, 9 : 31, 17 : 7, 21 : 42, 28 : 84, 51 : 67, 54 : 34,
          62 : 19, 64 : 60, 71 : 91, 80 : 99, 87 : 24, 93 : 73, 95 : 75, 98 : 79 }
```

On suppose que cette variable est accessible depuis toutes les futures fonctions, c'est-à-dire qu'elle est déjà déclarée dans le script.

4. Réécrire la fonction `caseFuture2` avec le dictionnaire `dSE` au lieu de la liste de listes `LSE`.

5. Quelle est la complexité de cette nouvelle fonction `caseFuture2` ?

6. Écrire une fonction `avanceCase` qui prend en argument un entier `case` correspondant à la case d'un joueur au début de son tour, un entier `de` correspondant à la valeur obtenue par le joueur à son lancer de dé, une chaîne de caractères `choix` correspondant à la stratégie de fin différente : `'r'` pour le rebond, `'i'` pour l'immobilisme et `'q'` pour une fin rapide. Cette fonction doit renvoyer le numéro de la case d'arrivée du joueur lorsqu'il part de la case `case`, que son résultat au lancer de dé est `de` et que la stratégie de fin choisie est `choix`.

7. Écrire une fonction `partie` qui prend en argument une chaîne de caractères `choix` correspondant à la stratégie de fin différente choisie, et qui renvoie la liste successive des cases visitées sur le plateau jusqu'à la fin de la partie.
La liste renvoyée commencera donc forcément par 0 et finira par 100.

IV. Plus court chemin

On souhaite dans cette partie, utiliser un algorithme glouton pour trouver la partie la plus courte.

8. Écrire une fonction `casesAccessibles` qui prend en argument une variable `case` contenant le numéro d'une case du plateau, et qui renvoie la liste des 6 cases accessibles depuis celle-ci.
On utilisera la fonction `avanceCase` *de la question* ??.
La liste renvoyée sera stockée dans une variable `cases` et, pour tout indice `i`, l'élément `cases[i]` doit contenir le numéro de la case d'arrivée après avoir obtenu la valeur `i+1` au lancer de dé. Cette liste doit donc toujours être de longueur 6.
On prendra l'option de fin rapide.

9. Écrire une fonction `meilleurChoix` qui prend en argument une variable `case` contenant le numéro d'une case du plateau, et qui renvoie le numéro de la meilleure case accessible depuis `case`.
Il est interdit d'utiliser la fonction `max` dans cette question.

L'algorithme glouton consistera à choisir la valeur du dé permettant de maximiser son déplacement à chaque coup.

10. Écrire une fonction `partieGloutonne` qui ne possède pas d'argument, et qui renvoie la liste des cases par lesquelles passe le pion dnas l'algorithme glouton.

Cette dernière fonction nous renvoie la liste `[0, 38, 44, 50, 67, 91, 97, 100]`.

11. Construire un exemple de plateau, contenant par exemple 2 échelles et pas de serpent, pour lequel notre algorithme ne trouve pas le chemin le plus court en nombre de coups.
Vous préciserez le résultat de l'algorithme glouton et un exemple d'une partie strictement plus rapide.

V. Étude du graphe correspondant

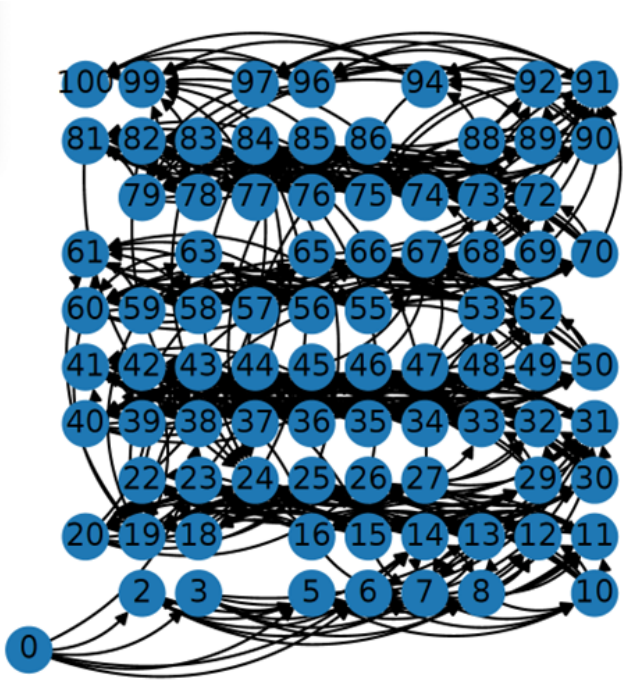

Fig. 2 Graphe correspondant au plateau de la Figure ??.

12. Écrire une fonction `eliminationDoublon1` qui prend en argument une liste `L` et qui renvoie la liste comportant les mêmes éléments que la liste `L` mais sans répétition.
On n'utilisera ni dictionnaire, ni tri.

13. Prouver que la fonction `eliminationDoublon1` est de complexité quadratique en la longueur de la liste `L` dans le pire des cas.

14. Écrire une fonction `eliminationDoublon2` qui prend en argument une liste `L` et qui renvoie la liste comportant les mêmes éléments que la liste `L` mais sans répétition. On impose à la fonction `eliminationDoublon2` d'avoir une complexité en $\underset{n\to+\infty}{O}\big(n\ln(n)\big)$, où n est la longueur de la liste `L`.
On effectuera un tri sur `L`.

15. Proposer une fonction `eliminationDoublon3` qui prend en argument une liste `L` et qui renvoie la liste comportant les mêmes éléments que la liste `L` mais sans répétition. On impose à la fonction `eliminationDoublon3` d'avoir une complexité linéaire en la longueur de la liste `L`.
On utilisera un dictionnaire.

On souhaite créer un graphe à partir du jeu : chaque case stable est un sommet et il existe un arc de la case `case1` à la case `case2` si un lancer de dé permet d'aller de `case1` à `case2`. On notera que la case 100 n'a pas de successeurs (puisque le jeu s'arrête). De plus, une case 0 sera utilisée pour prendre en compte le départ.
Une représentation de ce graphe pour le plateau de la Figure **??** est donnée par la Figure **??**.

16. En vous aidant des fonctions `casesAccessibles` et `eliminationDoublon3`, construire un graphe orienté `G` sous la forme d'un dictionnaire d'adjacence (dictionnaire de listes).

17. Pourquoi a-t-on besoin d'éliminer les doublons ? Prenez un exemple pour illustrer.

18. Quel est le nombre de sommets de ce graphe en fonction de N_{cases} et N_{SE} ?

Afin de résoudre le problème du parcours à nombre de coups minimum, on a mis au point une fonction qui trouve si un chemin existe entre un sommet de départ et un sommet d'arrivée.

```
1  def chemin(G, depart, arrivee) :
2      ''chemin(G : dict, depart : int, arrivee : int) -> bool''
3      predecesseur = {}
4      file = deque([depart])
5      predecesseur[depart] = None
6      while file :
7          sommet = file.popleft()
8          for successeurDeSommet in G[sommet] :
9              if successeurDeSommet not in predecesseur.keys() :
10                 file.append(successeurDeSommet)
11                 predecesseur[successeurDeSommet] = sommet
12     return arrivee in predecesseur
```

19. Sur quel type de parcours est basée la fonction `chemin` ? Justifiez cette réponse.
Précisez l'intérêt d'utiliser ce parcours.

Pour l'instant, la fonction renvoie un booléen, mais on souhaiterait qu'elle renvoie l'ensemble des cases par lesquelles on est passé pour passer de `depart` à `arrivee` avec, comme premier élément de la liste retournée `depart`, et comme dernier élément de la liste retournée `arrivee`. La fonction renverra une liste vide si aucun chemin n'a été trouvé.

20. Remplacer la ligne 11 par un ensemble de lignes permettant de répondre à cette exigence.

21. Écrire une fonction `partieOptimale` qui ne prend aucun argument et qui renvoie une liste de cases de longueur minimum partant de 0 et arrivant à 100 en vous aidant de la fonction `chemin` mise en place précédemment.

Cette dernière fonction nous renvoie `[0, 38, 39, 45, 67, 91, 94, 100]` pour le plateau de la Figure ??. Pour ce plateau, elle ne nous donne pas une partie plus courte que l'algorithme glouton, et un chemin qui n'est pas fondamentalement différent.

VI. Statistiques

On souhaite mettre en place des outils pour s'assurer que le jeu vérifie des standards quant à la longueur d'une partie en fonction de la stratégie de fin de partie.

Le code pour compiler obtenir la longueur des parties est donné ici :

```
1  longR, longI, longQ = [], [], []
2  for i in range(5000) :
3      longR.append(len(partie('r'))-1)
4      longI.append(len(partie('i'))-1)
5      longQ.append(len(partie('q'))-1)
```

22. Expliquer en quoi la soustraction par 1 est nécessaire pour connaître la longueur de la partie. On rappelle que la longueur d'une partie est définie par le nombre de lancers de dé.

On va vouloir s'intéresser à différentes statistiques sur ces longueurs de parties, comme la moyenne , l'écart-type ou encore la médiane. L'argument d'entrée pour les 4 questions suivantes est une liste d'entiers (`longR`, `longI` ou `longQ`).

23. Écrire une fonction `moyenne` qui prend en argument une liste `L`, et qui renvoie la moyenne des éléments de cette liste. Autrement dit, si on note n la longueur de la liste `L`, cette fonction renvoie la valeur : $\overline{x}_n = \dfrac{1}{n}\displaystyle\sum_{i=0}^{n-1} \texttt{L[i]}$.

24. Écrire une fonction `variance` qui prend en argument une liste `L`, et qui renvoie la variance des éléments de cette liste. Autrement dit, si on note n la longueur de la liste `L`, cette fonction renvoie la valeur :

$$s_n^2 = \frac{1}{n}\sum_{i=1}^{n} \left(\texttt{L[i]} - \overline{x}_n\right)^2$$

25. En déduire une fonction `ecartType` qui prend en argument une liste `L`, et qui renvoie l'écart-type de ses éléments.

La fonction `mediane` qui prend en argument une liste `L` et renvoie la valeur médiane du tableau des valeurs de `L` est définie ci-dessous. On notera que la recherche de la médiane est basée sur un algorithme de tri.

```
1  def mediane(L) :
2      n = len(L)
3      for i in range(n) :
4          cle = L[i]
5          j = i
6          while 0 < j and cle < L[j-i] :
7              # ligne à compléter
8              # ligne à compléter
9          L[j] = cle
10     return L[n//2]
```

26. Choisir l'une des 4 propositions données pour compléter les 2 lignes manquantes.

a) Proposition 1 :

```
7  L[j-1] = L[j]
8  j = j - 1
```

b) Proposition 2 :

```
7  L[j] = L[j-1]
8  j = j - 1
```

c) Proposition 3 :

```
7  L[j+1] = L[j]
8  j = j + 1
```

d) Proposition 4 :

```
7  L[j] = L[j+1]
8  j = j + 1
```

27. Donner le nom, puis la complexité de l'algorithme de tri employé :

a) dans le meilleur des cas : le cas d'une liste `L` déjà triée ;

b) dans le pire des cas : le cas d'une liste `L` triée à l'envers.

VII. Dessiner le plateau

On souhaite faire une représentation schématique du plateau (voir Figure ??) avec le code suivant :

```
1  for case in range(1, 101) : # pour les cases 1 à 100
2      i, j = position(case)
3      plt.text(i, j, str(case),
4        horizontalalignment = 'center', verticalalignment = 'center')
5
6  for caseD, caseA, in dSE.items() :
7      iD, jD = position(caseD)
8      i1, j1 = position(caseA)
9      if caseA > caseD :
10         couleur = 'b'
11     else :
12         couleur = 'r'
13     plt.plot(i1, j1, '-', color = couleur)
14     plt.plot(iD, jD, 'o', color = couleur)
15     plt.plot([iA, iD], [j1, jD], color = couleur)
16 plt.axis('equal')
17 plt.show()
```

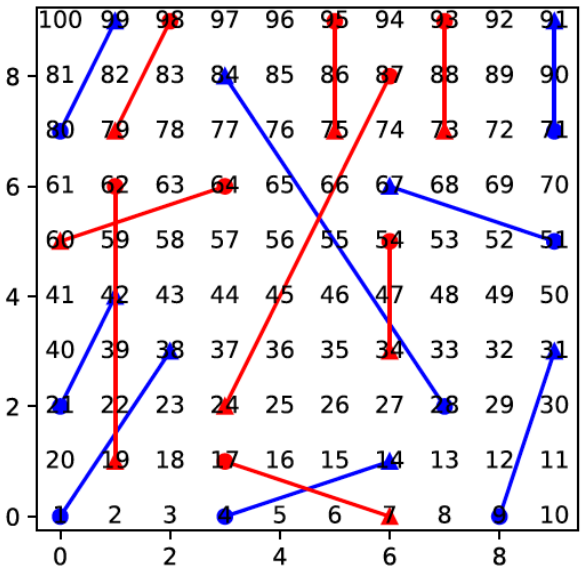

Fig. 3 Représentation schématisée du plateau de la Figure ??.

28. Écrire une fonction `position` qui prend en argument un entier `case` contenant le numéro d'une case, et qui renvoie les coordonées de `case` dans le plan sous la forme d'un couple d'entiers. Par exemple :

- `position(1)` renvoie $(0, 0)$: le coin en bas à gauche ;
- `position(54)` renvoie $(6, 5)$;
- `position(91)` renvoie $(9, 9)$: le coin en haut à droite.

29. En commentant la ligne 6 du code précédent, dites à quoi correspondent les variables `caseD` et `caseA` par rapport au dictionnaire `dSE`.

30. Commenter les lignes 9 à 12 du code fourni dans cette section.

ANNEXE

Utilisation du module `random`

On donne les docstrings correspondant à deux fonctions du module `random` :

```
randint(a, b) method of random.Random instance
        Return random integer in range [a, b], including both end points.
```

```
choices(seq) method of random.Random instance
        Choose a random element from a non-empty sequence.
```

Complexité des opérations sur les listes et dictionnaires

Principales opérations sur les listes.

On note n la longueur d'une liste `L` et `k` un indice valide d'un élément de cette liste. On note de même n_1 la longueur d'une liste `L1` et n_2 la longueur d'une liste `L2`.

Opération	**Complexité**
Longueur (`len(L)`)	$O(1)$
Accès en lecture d'un élément	$O(1)$
Accès en écriture d'un élément	$O(1)$
Copie (`L.copy()` ou `L[:]`)	$O(n)$
Ajout (`L.append(elt)` ou `L += [elt]`)	$O(1)$
Extension (`L1.extend(L2)` ou `L1 += L2`)	$O(n_2)$
Concaténation (`L1 + L2`)	$O(n_1 + n_2)$
Test de présence (`elt in L`)	$O(n)$
Désempiler le dernier élément (`L.pop()`)	$O(1)$
Désempliler un autre élément (`L.pop(-k)`)	$O(k)$
Maximum ou minimum (`max(L)` ou `min(L)`)	$O(n)$
Tri (`L.sort()` ou `sorted(L)`)	$O\big(n \ln(n)\big)$

Principales opérations sur les dictionnaires.
On note n la longueur d'un dictionnaire `d` et `cle` une clé du dictionnaire.

Opération	Complexité
Longueur (`len(d)`)	$O(1)$
Accès en lecture d'un élément (`v = d[cle]`)	$O(1)$
Accès en écriture d'un élément (`d[cle] = v`)	$O(1)$
Copie (`d.copy()`)	$O(n)$
Ajout (`d[cle] = v` la première fois)	$O(1)$
Test de présence (`cle in d`)	$O(1)$
Retrait d'un élément (`del d[cle]` ou `d.pop(cle)`)	$O(1)$

Utilisation du module `matplotlib`

- `plt.text(x, y, s)` permet de placer la chaîne de caractères `s` aux coordonnées `(x, y)`. Des arguments optionnels permettent de préciser la méthode de placement horizontal (`horizontalalignment` peut prendre les valeurs `'center'`, `'right'` ou `'left'`), et vertical (`verticalalignement` peut prendre les valeurs `'center'`, `'top'`, `'bottom'`, `'baseline'`, `'center_baseline'`).
- `plt.plot(x, y, '^')` permet de placer un point aux coordonnées `(x, y)` sous la forme d'un triangle vers le haut. Beaucoup d'autres symboles existent, en plus du triangle vers le haut, dont (liste non exhaustive) : `'v'`, `'<'` et `'>'` pour des triangles orientés différemment ; et `'.'` et `'o'` pour des disques plus ou moins gros ; `'s'`, `'p'` et `'h'` pour des figures régulières à 4, 5 ou 6 côtés.
- `plt.plot(Sx, Sy)` permet de tracer une courbe en trait plein en reliant les points dans l'ordre des 2 séquences données en entrées (le point de coordonnées `(Sx[i], Sy[i])` est lié au point de coordonnées `(Sx[i+1], Sy[i+1])`.

  Un argument de couleur peut être utilisé avec `plt.plot` : des raccourcis existent pour les couleurs les plus fréquentes : `'b'` pour bleu, `'r'` pour rouge, `'g'` pour vert, `'c'` pour cyan, `'m'` pour magenta, `'y'` pour jaune, `'k'` pour noir et `'w'` pour blanc.
- `plt.axis('equal')` permet de contraindre un ratio de 1 entre l'échelle en abscisses et en ordonnées dans un repère orthonormé.
- `plt.show()` crée la figure en cours dans une nouvelle fenêtre.