TP C# : Puissance 4 Clicker

Table des matières

1 Introduction

Aujourd'hui, nous vous proposons de choisir entre construire un Puissance 4 (niveau facile) ou réaliser un Pac-Man (niveau moyen : uniquement conseillé aux personnes ayant déjà programmé).

Ici, c'est le sujet du Puissance 4. Je suppose que vous y avez déjà tous joué plus petit. C'est un jeu composé d'une grille 7x6 et de jetons de deux couleurs (rouge et jaune). Il se joue à 2 et le but est d'aligner 4 jetons consécutifs de sa couleur sur la même ligne, colonne ou diagonale. Chacun leur tour, les joueurs déposent un jeton dans une colonne et celui ci tombe dans la dernière case vide de la colonne (la plus basse). Si la grille est remplie mais qu'aucun joueur ne peut gagner, il y a match nul.

On vous propose de réaliser ce jeu dans un langage nommé Python, un langage simple à comprendre, qui a peu de restriction, et qui possède beaucoup de bibliothèques disponibles pour renforcer les applications du langage. Aujourd'hui nous allons utiliser la bibliothèque Tkinter pour créer l'interface de notre jeu.

2 Cours

2.1 Python

Le python est un langage de programmation basique.

On y trouve par exemple des commentaires, ce sont des phrases (généralement écrites en anglais) pour expliquer une partie du code.
On peut faire un commentaire sur une ligne avec '#' :

```
# Ceci est un commentaire
```

Ou sur plusieurs lignes avec '"""' (triple guillemets) qui marque le début et la fin du commentaire :

```
""" Ceci
est un commentaire
sur plusieurs lignes """
```

On y trouve également des instructions/fonctions, par exemple on peut afficher du texte :

```
print("texte")
```

La fonction d'affichage est 'print' et '"texte"' est une chaîne de caractères, on le dénote grâce aux guillemets au début et à la fin, c'est ce qu'on appelle une string.

Une fonction est une liste d'instructions qui peut prendre des paramètres et renvoyer un résultat. Par exemple, on peut créer la fonction 'plus' qui jouera le rôle d'addition, en écrivant :

```
def plus(a, b):
    return a + b
```

Le mot clé 'def' signifie que nous définissons une fonction, 'plus' est le nom de cette fonction, 'a' et 'b' sont les 2 paramètres, 'return' indique que ce que la fonction renvoie, ici le résultat de l'addition entre a et b.
Pour appeler cette fonction il suffira d'écrire :

```
plus(2, 3) # qui renvoie 5 (= 2 + 3)
```

On peut également définir des variables, ce sont des identifiants/noms qui permettent de stocker une valeur, pour après s'en servir : y accéder plus tard, la modifier, etc.
Pour définir une variable il suffit d'écrire :

```
nom_de_la_variable = valeur
```

L'intérêt d'un programme est de faire des actions différentes en fonction des entrées. Pour pouvoir choisir, on utilise les conditions, avec le mot clé 'if'. Par exemple, si on veut tester le signe d'un nombre, on peut écrire :

```
if nb >= 0:
    # do something 1
# do something 2
```

Donc ici nous faisons quelque chose de spécifique si nb est positif ou nul.
Si nous voulons faire autre chose si nb est négatif, if suffit d'écrire :

```
if nb >= 0:
    # do something
else:
    # do something else
# do something
```

Il existe également un moyen de séparer plusieurs cas grâce au 'elif' :

```
if nb == 0:
    # do something 1
elif nb > 0:
    # do something 2
else:
    # do something 3
# do something
```

Nous pouvons ajouter autant de 'elif' que l'on veut.

Depuis tout a l'heure, lorsqu'on veut effectuer une action en fonction d'une autre, on effectue une tabulation. C'est le moyen de Python pour définir des 'scopes'. En fait, si on reprend le premier exemple du 'if', l'action 'do something 1' s'effectue uniquement si la condition du if est respectée, si elle était sur plusieurs lignes, elle s'écrirait :

```
if nb >= 0:
    # do something 1.1
    # do something 1.2
    # do something 1.3
# do something 2
```

Si on déroule ce bout de code, on a : "Si nb est positif ou nulle alors on effectue les actions 1.1, 1.2, 1.3, puis on exécute l'action 2. Sinon, on exécute seulement l'action 2."
On retrouve les tabulations dans chaque définition (d'une fonction, d'une classe, etc).

Si on veut répéter une action sans les écrire a la suite avec des if, on utilise les boucles.
Il existe 2 types de boucles : les boucles 'for' et les boucles 'while'.
Une boucle 'while' se répète tant qu'une condition est vérifiée, elle s'écrit :

```
while condition:
    # do something
# do something
```

Attention à ne pas faire de boucle infinie, si la condition est toujours vérifiée.

La boucle 'for' ne répète un nombre donné de fois, elle s'écrit :

```
for i in ensemble:
    # do something
# do something
```

Pour lui donner une plage d'entiers, il suffit d'écrire :

```
for i in range(nb):
    # action 1
# action 2
```

L'action 1 va s'exécuter nb fois. C'est équivalent à écrire :

```
for i in range(0, nb, 1):
    # action 1
# action 2
```

La fonction range(a, b, c) créer un ensemble compris entre [a, b[dont le pas est c.
Par exemple, le programme suivant :

```
s = ""
for i in range(1, 10, 2):
    s = s + 'a'
print(s)
```

Ce programme va afficher 'aaaaa' car la fonction range va renvoyer l'ensemble 1, 3, 5, 7, 9 et s est une chaîne de caractères, on lui concatène le caractère a.

2.2 Listes

En python, une liste est dynamique, c'est à dire que sa taille n'a pas besoin d'être renseigné au moment de l'initialisation de la liste, et elle n'est pas fixe. On peut ajouter et supprimer des éléments de la liste librement.

Pour initialiser une liste vide :

```
nom_de_la_liste = []
```

Pour initialiser une liste qui contient n éléments :

```
nom_de_la_liste = [element1, element2, element3, ..., elementn]
```

Pour ajouter un élément à la fin d'une liste :

```
nom_de_la_liste.append(element)
```

Pour ajouter un élément à une position précise dans la liste (en décalant tout ce qui suit :

```
nom_de_la_liste.insert(position, element)
# position étant compris entre 0 et n (n = longueur de la liste)
```

Pour supprimer un élément de la liste et le retourner :

```
nom_de_la_liste.pop(position)
# position étant comprise entre 0 et n - 1 (n = longueur de la liste)
```

Le paramètre position n'est pas obligatoire, s'il est omis, l'élément qui sera supprimer est le dernier de la liste.

Il existe également la fonction 'remove' qui cherche un élément dans une liste et le supprime dès qu'il l'a trouvé (uniquement la première occurrence) :

```
nom_de_la_liste.remove(element)
```

2.3 Tableau

Un tableau peut être défini de 2 manières différentes :

— une liste de listes (donc de dimension 2) ;
— une simple liste (donc de dimension 1).

Une liste de listes est de taille n * n : n listes de n éléments
Une simple liste est de taille n * n : une liste de n * n éléments

Pour accéder à la valeur à la position x, y (x = n° colonne et y = n° ligne) :
Une liste de listes :

```
liste[y][x]
```

Une simple liste :

```
liste[y * n + x]
```

On représente un tableau de cette manière :

```
      x = 2
        v
   0  1  2  3  4  5
0  .  .  -  .  .  .     <- y = 0
1  .  -  .  -  .  .
2  -  .  .  .  -  .
3  +  .  .  .  +  .
4  +  .  |  .  +  .     <- y = 4
5  +  .  |  .  +  .
```

Par exemple, à la position x = 2, y = 0,

```
tab[0][2] = '-'
tab[4][2] = '|'
tab[3][0] = '+'
```

Mais attention, tab[6][x] est invalide quelque soit x, pour y = 6 ou plus.
De même pour tab[y][6], pour x = 6 ou plus.

Pour initialiser une liste de listes, on utilise :

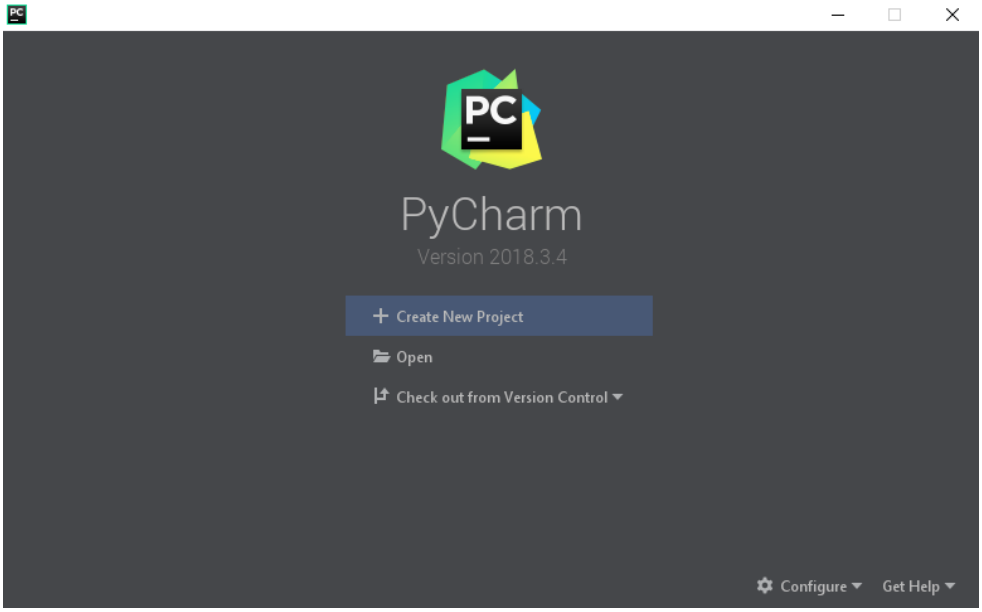
```
list = []
for i in range(n):
    list.append([])
    for j in range(n):
        list[i].append(element)
```

Pour initialiser une simple liste, on utilise :

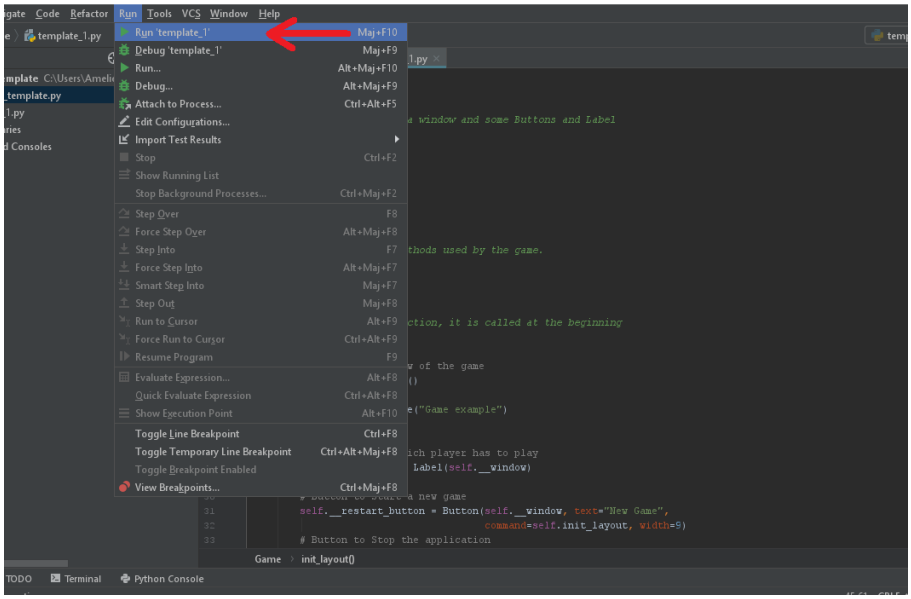
```
list = []
for i in range(n * n):
    list.append(element)
```

3 Exercices

Pour commencer, nous vous fournissons une template (un fichier déjà codé) que vous allez compléter.
Commencez par lancer l'IDE Python fourni, PyCharm.



Ouvrez le fichier template_1.py et lancez l'exécution (en appuyant sur Run).



Tkinter vous ouvre ainsi une fenêtre avec les éléments qu'on vous a fourni.
Vous pouvez voir en regardant le code que cette fenêtre est un objet Tk, qui possède un Title (le nom de la fenêtre). Nous avons également ajouté des boutons (objets Button) ainsi que du texte (objet Label).
Nous allons pas vraiment rentrer dans les détails mais pour faire simple, il est possible de changer le texte ("text="), les dimensions ("width="), la position de chacun de ces objets ("row=,

column=") et il est aussi possible de définir une fonction comme étant celle à exécuter si un certain bouton est actionné ("command=").

Bien, donc ceci pose les bases d'un programme utilisant Tkinter. N'hésitez pas à lire entièrement ce programme ainsi que les commentaires et n'hésitez pas à poser des questions aux assistants présents pour vous aider.

Ouvrez maintenant le second fichier. Il contient le code précédent ainsi que quelques ajouts comme la création de la grille (grid), et les boutons des colonnes stockés dans une liste (buttons). Cependant, il n'est pas complet, vous devez écrire les fonctions :

- put_coin : qui insère un jeton dans une colonne (la case la plus basse : qui est la plus proche de zéro)
- check_line : qui vérifie si un joueur a gagné en alignant 4 jetons consécutifs sur une ligne
- check_column : même chose que check_line mais sur les colonnes

Vous pouvez également gérer les diagonales mais ceci est un bonus pour ceux qui ont compris et réussi les étapes précédentes.

Enfin, vous pouvez personnaliser votre interface en changeant couleur, texte, dimensions de chaque éléments, ou même les dimensions de la grille !

Si vraiment vous êtes hyper doué et avez déjà tout fini, d'abord gg à vous ! ! et pour éviter de vous ennuyer, créez votre propre jeu à partir du premier template ;)

(Pour plus d'informations sur tkinter et ses nombreuses possibilités, on vous propose d'aller sur : http://www.xavierdupre.fr/app/teachpyx/helpsphinx/c_gui/tkinter.html, https://docs.python.org/3/library/tkinter.ttk.html)

I see no point in coding if it can't be beautiful.