CHAPITRE 11 UN EXEMPLE DE PROGRAMME

Programmation en Python

Un exemple récapitulatif

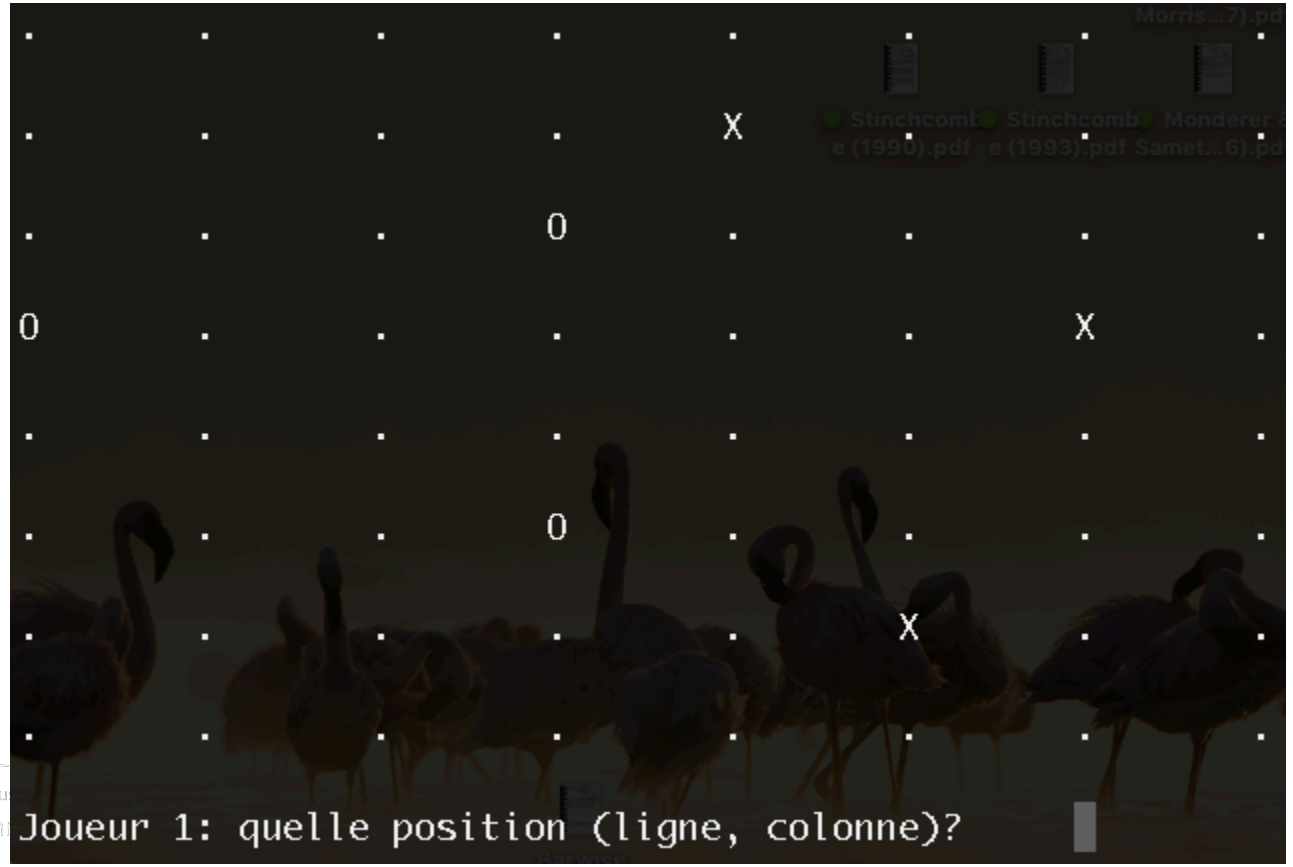
- Nous allons présenter un programme un peu plus conséquent qui permet de récapituler toutes les notions abordées jusqu'à maintenant.
- Le but est de présenter une implémentation du jeu « Puissance 4 » sur une grille dont la taille N x N est choisie au départ.

Puissance 4

- Puissance 4 est un jeu bien connu…
- Le jeu se joue à deux joueurs et sur une grille de taille N x N.
- Chacun des joueurs place à tour de rôle un pion dans une grille.
- Le but chaque joueur est de réussir à aligner en ligne, en colonne, ou en diagonale quatre de ses pions.
- Le premier qui réussit à aligner quatre de ses pions gagne le jeu.

Puissance 4

- Faire une démonstration du programme.
- (fichier cours/code/Ch10.py)



Puissance 4

- Nous allons créer petit à petit un certains nombre de fonctions dont nous avons besoin pour l'implémentation de ce jeu.
- Au final, le jeu lui-même sera implémenté comme une procédure globale. Celle-ci faisant appel aux fonctions définies préalablement.

Puissance 4

- Cette manière de procéder est courante en programmation : on crée des fonctions ou procédures qui appellent d'autres fonctions ou procédures, qui elles même appellent d'autres fonctions ou procédures, etc.
- Ainsi, à partir de briques de bases, et en se basant sur ce qui a déjà été implémenté, on crée fonctionnalités de plus en plus complexes.

Puissance 4

- On programme une fonction `generer_grille(N = 4)` qui génère une grille d'une certaine taille `N` valant 4 par défaut.
- Chaque ligne de la grille est une liste de taille N dont les éléments sont des chaînes `"."`. La grille est alors représentée comme la liste de ses lignes, comme illustré ci-dessous.
- Exemple : une grille de taille 3 x 3
  - `[[".", ".", "."], [".", ".", "."], [".", ".", "."]]`

Puissance 4


```
def generer_grille(taille = 4):
    """genere une grille d'une certaine taille"""
    grille = []
    for i in range(taille):
        grille.append(["."]*taille)
    return grille
```

Puissance 4


```
# we create a grid of size 5
ma_grille = generer_grille(5)
print ma_grille
```

```
[['.', '.', '.', '.', '.'], ['.', '.', '.', '.', '.'],
['.', '.', '.', '.', '.'], ['.', '.', '.', '.', '.'],
['.', '.', '.', '.', '.']]
```

Puissance 4

- On programme une fonction `placer_pion(joueur, grille, position)` qui permet de placer le pion du joueur `joueur` à la position `position` dans la grille `grille`.
- L'argument `joueur` vaut 1 ou 2. S'il vaut, 1, le pion placé sera un rond (O majuscule) "O" ; s'il vaut, 2, le pion placé sera une croix (X majuscule) "X".
- L'argument `grille` est une liste de listes.
- L'argument `position` est une liste ou un tuple de la forme (i,j).

Puissance 4


```
def placer_pion(joueur, grille, position):
    """place un pion dans une grille"""
    if joueur == 1:
        pion = "O"
    elif joueur == 2:
        pion = "X"
    grille[position[0]-1][position[1]-1] = pion
    return grille
```

Puissance 4


```
# On place quelques pions
ma_grille = placer_pion(1, ma_grille, (1,3))
print ma_grille
ma_grille = placer_pion(2, ma_grille, (3,5))
print ma_grille
ma_grille = placer_pion(1, ma_grille, (2,4))
print ma_grille
ma_grille = placer_pion(2, ma_grille, (4,3))
print ma_grille
```

Puissance 4

- On programme une fonction `chercher_alignement(joueur, liste, n)` qui cherche si un alignement de `n` pions du joueur `joueur` existe dans la liste `liste` (cette liste représentera plus tard, une ligne, une colonne ou une diagonale).
- Suivant que l'argument `joueur` vaut 1 ou 2, on cherchera un alignement de n "O" ou n "X", respectivement.

Puissance 4


```
def chercher_alignement(joueur, liste, n):
    """cherche un alignement de n symboles dans une
       liste"""
    compteur = 1
    pions = ["O", "X"]
    for i in range(len(liste)-1):
        if liste[i] == pions[joueur-1] \
           and liste[i] == liste[i+1]:
            compteur += 1
            if compteur >= n:
                return True
                break
        else:
            compteur = 1
    return False
```

Puissance 4


```
# On teste la fonction
l1 = ["O", "O", "O", "X", "X"]
print chercher_alignement(1, l1, 3)
True

l2 = ["O", "X", "O", "X", "X"]
print chercher_alignement(2, l2, 3)
False
```

Puissance 4

- On programme une fonction `generer_lignes(grille)` qui retourne la liste de toutes les lignes de la grille `grille`.
- Remarquez que dans ce cas, il suffit de retourner la grille elle même, puisque celle-ci est donnée comme la liste de ces lignes.

Puissance 4


```
def generer_lignes(grille):
    """retourne la liste des lignes d'une grille"""
    return grille


# Exemple
print generer_lignes(ma_grille)
```

```
[['.', '.', '.', '.', '.'], ['.', '.', '.', '.', '.'],
['.', '.', '.', '.', '.'], ['.', '.', '.', '.', '.'],
['.', '.', '.', '.', '.']]
```

Puissance 4

- On programme une fonction `generer_colonnes(grille)` qui retourne la liste de toutes les colonnes de la grille `grille`.
- L'idée générale est de transposer la « matrice » `grille`…
- C'est un peu plus compliqué en pratique.

Puissance 4


```
def generer_colonnes(grille):
    """retourne la liste des colonnes d'une grille"""
    liste_colonnes = []
    # we update them
    for i in range(len(grille)):
        liste_colonnes.append([])
        for j in range(len(grille)):
            liste_colonnes[-1].append(grille[j][i])
    return liste_colonnes


# Exemple
print generer_colonnes(ma_grille)
```

Puissance 4

- On programme une fonction `generer_diagonales1(grille)` qui retourne la liste de toutes les diagonales « montantes » de la grille `grille`.
- Pour cela, remarquez que chaque diagonale de ce type est constituée des éléments de la grille dont les indices donnent une même somme.
- Par exemple, la diagonale `[(3,1),(2,2),(1,3)]` est formée de tous les éléments dont les indices ont une somme de 4.

Puissance 4


```
def generer_diagonales1(grille):
    """retourne la liste des diagonales d'une grille"""
    dico_diag = {} # on utilise un dico cette fois
    # les sommes des indices vont de 2 à N*2
    for k in range(2, len(grille)*2+1):
        dico_diag[k] = []
    for i in range(len(grille)):
        for j in range(len(grille)):
            n = (i+1)+(j+1) # somme des indices
            dico_diag[n].append(grille[i][j])
    return dico_diag


# Exemple
print generer_diagonales1(ma_grille)
```

Puissance 4

- On programme une fonction `generer_diagonales2(grille)` qui retourne la liste de toutes les diagonales « descendantes » de la grille `grille`.
- Remarquez que chaque diagonale de ce type est constituée des éléments de la grille dont les indices donnent une même différence.
- Par exemple, la diagonale la diagonale `[(1,3),(2,4),(3,5)]` d'une grille de taille 5 est formée de tous les éléments dont les indices ont une différence de -2.

Puissance 4


```
def generer_diagonales2(grille):
    """liste des autres diagonales d'une grille"""
    dico_diag = {} # on utilise un dico
    # les sommes des indices vont de -N+1 à N-1
    for k in range(-len(grille)+1, len(grille)):
        dico_diag[k] = []
    for i in range(len(grille)):
        for j in range(len(grille)):
            n = (i+1)-(j+1) # différence des indices
            dico_diag[n].append(grille[i][j])
    return dico_diag


# Exemple
print generer_diagonales2(ma_grille)
```

Puissance 4

- On programme une fonction `imprimer_grille(grille)` qui imprime la grille grille dans votre terminal de manière conviviale, comme illustré précédemment.
- On rappelle que les chaînes `"\t"` et `"\n"` codent un espace de tabulation et un retour de ligne, respectivement.

Puissance 4


```
def imprimer_grille(grille):
    for i in range(len(grille)):
        for j in range(len(grille)):
            print grille[i][j], "\t",
        print "\n"

# Exemple
imprimer_grille(ma_grille)
```

Puissance 4

- On définit maintenant une fonction `jouer(n)` qui permet de jouer à Puissance 4 sur une grille de taille n.
- On utilise toutes les fonctions que l'on a programmées précédemment.
- Utilisez une grande boucle « while » qui demande à chaque joueur de placer un pion à tour de rôle, jusqu'à ce qu'un alignement de 4 pions soit trouvé dans une ligne, une colonne ou une diagonale.

Puissance 4


```
def jouer(n):
    """simule un jeu de taille n"""
    fin_du_jeu = False
    # on génère la grille
    G = generer_grille(n)
    imprimer_grille(G)
    # chaque joueur doit jouer à son tour
    while fin_du_jeu == False:
        ...(cf. slide suivant)...
```

Puissance 4


```
# Player 1 joue...
# il doit entrer une position valide...
# repose la question tant que position non valide
placement = False
while placement == False:
    pos = input("Joueur 1: quelle position \
                (ligne, colonne)? ")
    if G[pos[0]-1][pos[1]-1] == ".":
        G = placer_pion(1,G,pos)
        placement = True
# on imprime la grille mise à jour
imprimer_grille(G)
```

Puissance 4


```
# On génère les lignes, colonnes, diagonales
lignes = generer_lignes(G)
colonnes = generer_colonnes(G)
diagonales1 = generer_diagonales1(G)
diagonales2 = generer_diagonales2(G)
```

Puissance 4


```
# On examine si Player 1 gagne
# D'abord dans les lignes...
for l in lignes:
    if chercher_alignement(1, l, 4):
        print "Player 1 wins!"
        fin_du_jeu = True
        break # casse la boucle for
```

Puissance 4


```
# s'il n'y a pas de ligne gagnante,
# on examine les colonnes
if fin_du_jeu == False:
    for c in colonnes:
        if chercher_alignement(1, c, 4):
            print "Player 1 wins!"
            fin_du_jeu = True
            break # breaks the for loop
```

Puissance 4


```
# s'il n'y a pas de colonne gagnante,
# on examine les diagonales de type 1
if fin_du_jeu == False:
    for d in diagonales1.values():
        if chercher_alignement(1, d, 4):
            print "Player 1 wins! »
            fin_du_jeu = True
            break
```

Puissance 4


```
# s'il n'y a pas de diagonales de type 1 gagnante,
# on examine les diagonales de type 2
if fin_du_jeu == False:
    for d in diagonales2.values():
        if chercher_alignement(1, d, 4):
            print "Player 1 wins! »
            fin_du_jeu = True
            break
```

Puissance 4


```
# Si le joueur 1 n'a pas gagné,
# le joueur 2 peut jouer
if fin_du_jeu == False:
    # il doit entrer une position valide...
    # repose la question tant que nécéssaire
    placement = False
    while placement == False:
        pos = input("Joueur 2: quelle position \
                    (ligne, colonne)? ")
        if G[pos[0]-1][pos[1]-1] == ".":
            G = placer_pion(2,G,pos)
            placement = True
    # on imprime la grille mise à jour
    imprimer_grille(G)
```

Puissance 4


```
# On génère les lignes, colonnes, diagonales
lignes = generer_lignes(G)
colonnes = generer_colonnes(G)
diagonales1 = generer_diagonales1(G)
diagonales2 = generer_diagonales2(G)
```

Puissance 4


```
# On examine si Player 2 gagne
# D'abord dans les lignes...
for l in lignes:
    if chercher_alignement(2, l, 4):
        print "Player 2 wins!"
        fin_du_jeu = True
        break
```

Puissance 4


```
# s'il n'y a pas de ligne gagnante,
# on examine les colonnes
if fin_du_jeu == False:
    for c in colonnes:
        if chercher_alignement(2, c, 4):
            print "Player 2 wins!"
            fin_du_jeu = True
            break
```

Puissance 4


```
# s'il n'y a pas de colonne gagnante,
# on examine les diagonales de type 1
if fin_du_jeu == False:
    for d in diagonales1.values():
        if chercher_alignement(2, d, 4):
            print "Player 2 wins!"
            fin_du_jeu = True
            break
```

Puissance 4


```
# si pas de diagonales de type 1 gagnante,
# on examine les diagonales de type 2
if fin_du_jeu == False:
    for d in diagonales2.values():
        if chercher_alignement(2, d, 4):
            print "Player 2 wins!"
            fin_du_jeu = True
            break
```

Puissance 4


```
# Finalement, pour lancer le jeu sur une grille de taille
N x N, il suffit d'appeler la fonction jouer(N).

# Par exemple (faire démo):

jouer(6)
```