Phase 2 - Créer des jeux

Pour vérifier si votre code Python adhère aux recomendations PEP8 allez sur ce site:
http://pep8online.com/ (http://pep8online.com/)

Utiliser le notebook Jupyter

Les cellules code dans un notebook Jupyter doivent être executable. Si vous voulez simplement montrer un bout de code pour en parler sans l'executer, placez-le entre triple-backquotes.

```
bout de code
(exemple)
```

Toutes les cellules code doivent être executable et produire un résultat :

- Pour executer une cellule appuyez sur `SHIFT-Return`
- Avant de créer un PDF, recalculez toutes les cellules code
- Choisissez le menu `Kernel > Restart & Run all`
- Ceci renumérote automatiquement toutes les cellules à partir de 1
- Si une variables ou fonctions ne sont pas définis, vous en êtes avertit
- Selon la langue du système (Raspberry Pi settings), l'étiquette `In` ou `Entrée` est affichée

In [1]:

```
from sense_hat import SenseHat
from gamelib import *
from random import randint, choice
from time import time, sleep
import numpy as np

sense = SenseHat()
sense.clear()
```

```
module name = gamelib
```

Le module `gamelib` contient les couleurs.

In [2]:

```
colors = [BLACK, RED, GREEN, YELLOW, BLUE, CYAN, MAGENTA]
```

Variables gobales

Nous déclarons quelques variables commes globales. Ces variables seront accessible depuis partout dans Python.

In [3]:

```
x, y = 0, 0    # cursor position
n = 8          # board size (n x n)
T = np.zeros((n, n), int)    # Matrix of the board (table)
colors = (BLACK, RED, GREEN, BLUE)  # color list
p = 1          # current player
score = [0, 0] # current score
```

Le module NumPy

Pour cette deuxième phase, nous allons remplacer les matrices (listes 2D Python) par des matrices 2D NumPy. Voici les avantages:

- les calculs sont beaucoup plus rapide
- la matrice 2D peut être affiché avec `print`
- il existent beaucoup de fonctions
- Utiliser le module NumPy pour le calcul matriciel.

C'est standard d'importer le module **NumPy** avec le raccourci `np`.

In [4]:

```
import numpy as np
```

La fonction `np.array()` transforme une liste Python en matrice NumPy.

In [5]:

```
L = np.array([[0, 1], [1, 1]])
```

Regardons le type de cet objet. C'est un `numpy.ndarray` (numerical data array).

In [6]:

```
type(L)
```

Out[6]:

```
numpy.ndarray
```

Une matrice 2D peut être affiché avec `print`.

In [7]:

```
print(L)
```

```
[[0 1]
 [1 1]]
```

Créer des matrices

Il existent des fonctions pour créer des matrices

In [8]:

```
A = np.zeros((8, 8), int)
A
```

Out[8]:

```
array([[0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0]])
```

In [9]:

```
B = np.ones((3, 8), int)
B
```

Out[9]:

```
array([[1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1]])
```

In [10]:

```
print(L*2)
```

```
[[0 2]
 [2 2]]
```

Afficher une matrice sur senshat
https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.randint.html (https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.randint.html)

In [11]:

```
m = len(colors)
R = np.random.randint(0, m-1, (8, 8))
R
```

Out[11]:

```
array([[1, 1, 1, 1, 2, 1, 0, 1],
       [0, 1, 2, 2, 2, 1, 2, 1],
       [0, 1, 2, 0, 2, 2, 2, 2],
       [1, 2, 2, 2, 2, 1, 1, 2],
       [1, 1, 2, 2, 2, 1, 1, 2],
       [0, 1, 0, 1, 0, 2, 2, 1],
       [1, 2, 0, 2, 1, 2, 1, 2],
       [2, 1, 1, 0, 0, 1, 0, 0]])
```

Afficher sur le SenseHAT

Pour afficher sur les LEDs du SenseHAT nous utilisons une matrice avec des entiers. Cest entiers sont également l'indice pour la liste `colors`

In [12]:

```
def show_matrix(M):
    global colors
    n, m = M.shape
    for x in range(n):
        for y in range(m):
            col = colors[M[y, x]] # lines & columns are inversed
            sense.set_pixel(x, y, col)
```

In [13]:

```
show_matrix(R)
```

In [14]:

```
def show_cursor(x, y, p):
    """Show cursor for player p (1, 2)."""
    sense.set_pixel(x, y, colors[p])
```

In [15]:

```
show_cursor(x, y, p)
```

Indices

Dans une matrice 2D nous utilisons les indices i, j. D'habitude on utilise i pour les lignes et j pour les colonnes. Donc les indices sont invertit:

- i correspond à l'axe y
- j correspond à l'axe x

In [16]:

```
T[1, 4] = 3
T
```

Out[16]:

```
array([[0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 3, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0]])
```

In [17]:

```
T[2:4, 2:4] = L * 3
T
```

Out[17]:

```
array([[0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 3, 0, 0, 0],
       [0, 0, 0, 3, 0, 0, 0, 0],
       [0, 0, 3, 3, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0]])
```

Fonctions callback

Des fonctions peuvent être associé aux 5 directions du joystick. Les fonctions prennent comme argument l'événement `event` .

- direction_left
- direction_right
- direction_up
- direction_down
- direction_middle
- direction_any

Si vous êtes intéressés dans les détails, ces fonctions utilisent un mécanismes qu'on appelle des **threads**. Vous pouvez aller regarder le code source de l'API sur GitHub:

https://github.com/RPi-Distro/python-sense-hat/blob/master/sense_hat/stick.py (https://github.com/RPi-Distro/python-sense-hat/blob/master/sense_hat/stick.py)

In [18]:

```
# dir(sense.stick)
```

In [19]:

```
def right(event):
    """Make display red."""
    sense.clear(RED)

def left(event):
    """Make display blue."""
    sense.clear(BLUE)

def up(event):
    """Make display green."""
    sense.clear(GREEN)

def down(event):
    """Make display yellow."""
    sense.clear(YELLOW)

def middle(event):
    """Clear display."""
    sense.clear()

sense.stick.direction_left = left
sense.stick.direction_right = right
sense.stick.direction_up = up
sense.stick.direction_down = down
sense.stick.direction_middle = middle
```

Exercice

Redéfinir les fonctions `up` et `down` pour incrementer ou décrémenter un nombre qui change de couleur selon l'index dans `colors` .

In [20]:

```
i = 0

def up(event):
    global i, n
    if event.action == 'pressed':
        i = (i+1) % len(colors)
        sense.show_letter(str(i), text_colour=colors[i])

def middle(event):
    pass

def down(event):
    global i, n
    if event.action == 'pressed':
        i = (i-1) % len(colors)
        sense.show_letter(str(i), text_colour=colors[i])
```

Trouver des info sur une fonction callback

Verifier si une fonction callback et définie.

In [21]:

```
print(sense.stick.direction_left)
```

```
<function left at 0x6ffa1e40>
```

Afficher le docstring de cette fonction.

In [22]:

```
print(sense.stick.direction_left.__doc__)
```

```
Make display blue.
```

Afficher le nom de cette fonction.

In [23]:

```
print(sense.stick.direction_left.__name__)
```

```
left
```

In [24]:

```
callbacks = (sense.stick.direction_up, sense.stick.direction_down,
             sense.stick.direction_left, sense.stick.direction_right,
             sense.stick.direction_middle, sense.stick.direction_any)

for cb in callbacks:
    if cb != None:
        print(cb.__name__, cb.__doc__, sep='\t')
```

```
up      Make display green.
down    Make display yellow.
left    Make display blue.
right   Make display red.
middle  Clear display.
```

Déplacer un curseur

Beaucoup de jeux necessitent le déplacement d'un curseur. Nous utilisons les 4 directions pour déplacer le curseur, et le boutons central pour faire la sélection.

In [25]:

```
def move_cursor(event):
    """Deplace le curseur (x, y) sur un board (n x n)."""
    global x, y, n
    if event.direction == 'left':
        x = max(x-1, 0)
    elif event.direction == 'up':
        y = max(y-1, 0)
    elif event.direction == 'right':
        x = min(x+1, n-1)
    elif event.direction == 'down':
        y = min(y+1, n-1)

def play(event):
    if event.action == 'pressed':
        move_cursor(event)
        sense.clear()
        show_cursor(x, y, 1)

sense.stick.__init__()
sense.stick.direction_any = play
```

En posant $n = 3$ nous pouvons restreindre le mouvement du curseur à une région 3x3.

In [26]:

```
n=3
```

Afficher le gagnant

Après chaque match nous affichons le gagnant (1 ou 2) dans sa couleur. Si c'est match nul nous affichons 0 en blanc.

Pour donner d'importance, nous flashons 3 fois.

In [27]:

```
def show_winner(p, n=3):
    """Show the winner (1, 2) in the player's color."""
    col = WHITE if p==0 else colors[p]
    for i in range(3):
        sense.show_letter(str(p), text_colour=col)
        sleep(0.2)
        sense.clear()
        sleep(0.2)

for i in range(3):
    show_winner(i)
```

Afficher le score

Après chaque match nous affichons le score des deux jouer, ou le score simple.

In [28]:

```
def show_score():
    """Display the score as scrolling text."""
    text = str(score[0]) + ':' + str(score[1])
    sense.show_message(text)

score = [2, 3]
show_score()
```

Si le score est entre 0 et 9 on peut l'afficher comme lettre, un après l'autre, dans la couleur respective du joueur.

In [29]:

```
def show_score2():
    """Display score as two numbers in different colors."""
    for i in range(2):
        sense.show_letter(str(score[i]),text_colour=colors[i+1])
        sleep(1)
    sense.clear()

show_score2()
```

Si le score est entre 0 et 8 on peut aussi l'afficher de façon graphique avec une barre.

In [30]:

```
def show_score3():
    """Display score as two colored bars."""
    M = np.zeros((8, 8), int)
    M[0:score[0], 3] = 1
    M[0:score[1], 4] = 2
    show_matrix(M)
    sleep(1)

show_score3()
```

Cellules conséqutifs identiques

Dans Morpion et Puissance 4 nous devons tester si une séquence de 3 ou 4 cellules sont identiques. Nous définissons une fonctions qui compare 3 ou 4 cellules à partir d'une position initiale et une direction.

In [31]:

```
def is_equal(T, x, y, dx, dy, p, n):
    """Check if n cells starting at (x, y),
    in direction (dx, dy) are all equal to p"""
    for i in range(n):
        if T[y, x] != p:
            return False
        x += dx
        y += dy

    return True
```

Testons notre function dans les 4 directions.

In [32]:

```
T = np.zeros((8, 8), int)
T[1:5, 3] = 1
T[7, 4:8] = 2
T[2, 0] = T[3, 1] = T[4, 2] = T[5, 3] = 3
T
```

Out[32]:

```
array([[0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 0, 0, 0, 0],
       [3, 0, 0, 1, 0, 0, 0, 0],
       [0, 3, 0, 1, 0, 0, 0, 0],
       [0, 0, 3, 1, 0, 0, 0, 0],
       [0, 0, 0, 3, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 2, 2, 2, 2]])
```

In [33]:

```
print(is_equal(T, 3, 1, 0, 1, 1, 4))
print(is_equal(T, 4, 7, 1, 0, 2, 4))
print(is_equal(T, 0, 2, 1, 1, 3, 4))
```

```
True
True
True
```

In [34]:

```
def in_a_row(T, p, k):
    """Check if player p has k stones in a row."""
    n, m = T.shape
    # columns
    for x in range(m):
        for y in range(n-k+1):
            if is_equal(T, x, y, 0, 1, p, k):
                return True
    # lines
    for y in range(n):
        for x in range(m-k+1):
            if is_equal(T, x, y, 1, 0, p, k):
                return True
    # diagonals
    for y in range(n-k+1):
        for x in range(m-k+1):
            if  is_equal(T, x, y, 1, 1, p, k):
                return True
            if  is_equal(T, x+k-1, y, -1, 1, p, k):
                return True
    return False
```

In [35]:

```
T = np.array([[1, 2, 2], [2, 2, 1], [1, 2, 1]])
print(T)
print(in_a_row(T, 2, 3))
```

```
[[1 2 2]
 [2 2 1]
 [1 2 1]]
True
```

Fin du jeu

Le jeu est terminé quand toutes les places sont occupées. La méthode `all()` est vrai si toutes les éléments sont différent de 0.

In [36]:

```
def is_finished(T):
    """Returns True if all cells are occupied."""
    return T.all()
```

On va tester la fonction avec deux cas.

In [37]:

```
T = np.ones((3, 3), int)
is_finished(T)
```

Out[37]:

```
True
```

In [38]:

```
T[0, 0] = 0
is_finished(T)
```

Out[38]:

```
False
```

Morpion

Dans le jeu morpion les joueurs dovent placer en alternance une pièce sur un plateau de 3x3 cellules. Celui qui a placé en premier 3 pièces sur une ligne, colonne ou diagonale gagne.

In [39]:

```
n = 3
T = np.zeros((n, n), int)
print(T)
```

```
[[0 0 0]
 [0 0 0]
 [0 0 0]]
```

La fonction `init` prépare le jeu.

In [40]:

```
def init(reset=False):
    global x, y, p, n, score, colors, T
    x, y = 1, 1
    n = 3
    T = np.zeros((n, n), int)
    colors = [BLACK, BLUE, YELLOW]
    if reset:
        score = [0, 0]
        p = 1
    show_board()
    show_cursor3()
```

La fonction `show_board` affiche le tableau.

In [41]:

```
def show_board():
    """Show a 3x3 matrix on the 8x8 board."""
    for x in range(8):
        for y in range(8):
            if x % 3 == 2 or y % 3 == 2:
                sense.set_pixel(x, y, GRAY)
            else:
                val = T[y//3, x//3]
                sense.set_pixel(x, y, colors[val])
```

In [42]:

```
def show_cursor3():
    """Scale the cursor from 3x3 matrix to 8x8 board."""
    global x, y, p
    col = colors[p] if T[y, x] != p else BLACK
    sense.set_pixel(3*x, 3*y, col)
```

The following implements the `play` function.

In [43]:

```
def play(event):
    """Place a stone on the board if the position is empty."""
    global x, y, p
    if event.action == 'pressed':
        move_cursor(event)
        if event.direction == 'middle':
            if T[y, x] == 0:
                T[y, x] = p
                if in_a_row(T, p, 3):
                    show_winner(p)
                    score[p-1] += 1
                    show_score3()
                    init()
                elif is_finished(T):
                    show_winner(0)
                    show_score3()
                    init()
                p = 3 - p
        show_board()
        show_cursor3()

sense.stick.__init__()
sense.stick.direction_any = play
init(True)
```

m, n, k game

Le m, n, k game est joué sur un tableau m x n, par deux joueurs qui placent une pièce en alternance. Le but est de placer k pièces en ligne, colonne ou en diagonale.

https://en.wikipedia.org/wiki/M,n,k-game (https://en.wikipedia.org/wiki/M,n,k-game)

In [44]:

```
def init(reset=False, size=8):
    global x, y, p, n, score, T
    n = size
    T = np.zeros((n, n), int)
    x, y = 0, 0
    if reset:
        score = [0, 0]
        p = 1
    show_matrix(T)
    show_cursor(x, y, p)
```

In [45]:

```
def play(event):
    """Use 4 directions keys to move cursor, middle to play."""
    global x, y, p
    if event.action == 'pressed':
        move_cursor(event)
        if event.direction == 'middle':
            if T[y, x] == 0:
                T[y, x] = p
                if is_finished(T):
                    show_winner(0)
                    show_score3()
                elif in_a_row(T, p, 4):
                    show_winner(p)
                    score[p-1] += 1
                    show_score3()
                    init()
                p = 3 - p

        show_matrix(T)
        show_cursor(x, y, p)

sense.stick.__init__()
sense.stick.direction_any = play
init(True)
```

Puissance 4

Le but du jeu est d'aligner une suite de 4 pions de même couleur sur u ne grille comptant 7 rangées et 8 colonnes.

Tour à tour les deux joueurs placent un pion dans la colonne de leur choix, le pion coulisse alors jusqu'à la position la plus basse possible dans la dite colonne à la suite de quoi c'est à l'adversaire de jouer.

Le vainqueur est le joueur qui réalise le premier un alignement (horizontal, vertical ou diagonal) consécutif d'au moins quatre pions de sa couleur. Si, alors que toutes les cases de la grille de jeu sont remplies, aucun des deux joueurs n'a réalisé un tel alignement, la partie est déclarée nulle.

In [46]:

```
def init(reset=False):
    """Initialize global parameters for Connect4."""
    global x, y, p, dt, colors, T
    x, y = 3, 0
    if reset:
        score = [0, 0]
        p = 1
    dt = 0.2
    colors = [BLUE, RED, YELLOW, BLACK]
    T = np.zeros((8, 8), int)
    T[0] = 3

    show_matrix(T)
    show_cursor(x, y, p)
```

In [47]:

```
init()
T
```

Out[47]:

```
array([[3, 3, 3, 3, 3, 3, 3, 3],
       [0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0]])
```

Ceci sont les fonctions pour le jeu Puissance 4.

In [48]:

```
def show(event):
    if event.action == 'pressed':
        show_matrix(T)
        show_cursor(x, y, p)

def left(event):
    """Move cursor to the left."""
    global x
    if event.action == 'pressed':
        x = max(x-1, 0)

def right(event):
    """Move cursor to the right."""
    global x
    if event.action == 'pressed':
        x = min(x+1, 7)

def down(event):
    """Move disc downwards."""
    global x, y, p
    if event.action == 'pressed':
        while (y < 7 and T[y+1, x] == 0):
            y += 1
            show_matrix(T)
            show_cursor(x, y, p)
            sleep(.1)
        T[y, x] = p
        if in_a_row(T[1:], p, 4):
            show_winner(p)
            show_score3()
            init()
        elif is_finished(T):
            show_winner(0)
            show_scores3()
            init()

        y = 0
        p = 3 - p

sense.stick.__init__()
sense.stick.direction_left = left
sense.stick.direction_right = right
sense.stick.direction_down = down
sense.stick.direction_up = init
sense.stick.direction_middle = show_score()
sense.stick.direction_any = show
init()
```

Tetris

Définissons les formes des base: L, O, I.

In [49]:

```
L = np.array([[1, 0], [1, 1]], int)
print(L)
```

```
[[1 0]
 [1 1]]
```

In [50]:

```
O = np.ones((2, 2), int)
print(O)
```

```
[[1 1]
 [1 1]]
```

In [51]:

```
I = np.ones((3, 1), int)
print(I)
```

```
[[1]
 [1]
 [1]]
```

Rotation des matrices

In [52]:

```
np.rot90(L)
```

Out[52]:

```
array([[0, 1],
       [1, 1]])
```

In [53]:

```
np.rot90(L, 2)
```

Out[53]:

```
array([[1, 1],
       [0, 1]])
```

In [54]:

```
np.rot90(L, 3)
```

Out[54]:

```
array([[1, 1],
       [1, 0]])
```

In [55]:

```
np.rot90(I)
```

Out[55]:

```
array([[1, 1, 1]])
```

Check if space is empty

In [56]:

```
def overlap(T, S, x, y):
    pass
```

Add a shape

In [57]:

```
def add(T, S, x, y):
    """Add shape S to T at (x, y)."""
    n, m = S.shape
    T[y:y+n, x:x+m] = S
```

In [58]:

```
add(T, L, 0, 0)
add(T, O*2, 5, 0)
add(T, L*3, 6, 0)
print(T)
show_matrix(T)
```

```
[[1 0 3 3 3 2 3 0]
 [1 1 0 0 0 2 3 3]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]]
```

In [59]:

```
show_matrix(T)
```

In [60]:

```
def rem(T, S, x, y):
    """Remove shape S at (x, y)."""
    n, m = S.shape
    for i in range(n):
        for j in range(m):
            T[y+i, x+j] = 0
```

In [61]:

```
rem(T, O, 0, 0)
add(T, O, 1, 0)
show_matrix(T)
```