Projet Morpion

par *Hélène et Victoria* ; Bugnon Ours, oc.info 2018/2019

Morpion est un jeux simple qui se joue sur un cadrillage 3x3. Le but est d'aligner 3 jetons en colonne, ligne au en diagonale.

Le jeux est joué sur la plateforme SenseHAT pour le Raspberry Pi. Dans ce notebook, des fragment de code sont expliqué. Parfois le résultat apparait sur le SenseHAT et parfois dans la cellule Out.

Pour utiliser SenseHAT, il faut importer le module SensHAT pour avoir accès aux fonctions. Il faut créer une instance SenseHat() pour accéder aux méthodes.

In [1]:
```
from sense_hat import SenseHat
from time import sleep, time
sense = SenseHat()
```

Ensuite nous définissons les variables : les couleurs qui représentent le player 1 par exemple.

In [2]:
```
X = (255, 255, 255)
O = (0, 0, 0)
P1 = (0, 0, 255)
P2 = (255, 255, 0)
colors = (O, P1, P2)
score1 = 0
score2 = 0
```

In [3]:
```
print(colors)
```

```
((0, 0, 0), (0, 0, 255), (255, 255, 0))
```

Définition init()

- Nous devons faire une défnition initation, où nous définissons l'état initial du jeu qui sera modifié par les joueurs.

- Pour que nos arguments appelés *state*, *board*, *state_to_board* soit utilisables dans tous le code, il fat utiliser le mot-clé **global**. Ensuite nous créeons la grille sur lequel a lieux le jeu qui est *board*. Puis nous créeons une matrice dont 4 pixels représentent un élément de la matrice *state* 3x3, sans oublier de bien faire les sauts qui correspondent au cadrillage de la grille.

In [4]:
```
def init():
```

In [4]:
```
def init():
    global state
    global board
    global state_to_board
    board = [
        O, O, X, O, O, X, O, O,
        O, O, X, O, O, X, O, O,
        X, X, X, X, X, X, X, X,
        O, O, X, O, O, X, O, O,
        O, O, X, O, O, X, O, O,
        X, X, X, X, X, X, X, X,
        O, O, X, O, O, X, O, O,
        O, O, X, O, O, X, O, O,
        ]
    state_to_board = [[(0, 1, 8, 9), (3, 4, 11, 12), (6, 7, 14, 15)],
                      [(24, 25, 32, 33), (27, 28, 35, 36), (30, 31, 3
8, 39)],
                      [(48, 49, 56, 57), (51, 52, 59, 60), (54, 55, 6
2, 63)]]

    state =  [[0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

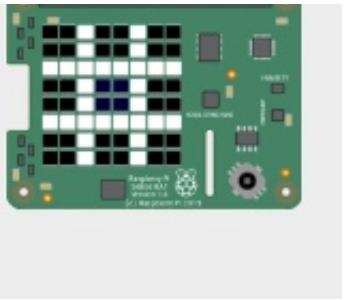
```
---------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
last)
<ipython-input-4-21e81b536c8e> in <module>
     20
     21
---> 22 state

NameError: name 'state' is not defined
```

In [24]:
```
  board = [
        O, O, X, O, O, X, O, O,
        O, O, X, O, O, X, O, O,
        X, X, X, X, X, X, X, X,
        O, O, X, O, O, X, O, O,
        O, O, X, O, O, X, O, O,
        X, X, X, X, X, X, X, X,
        O, O, X, O, O, X, O, O,
        O, O, X, O, O, X, O, O,
        ]

board[:3]
```

Out[24]:
```
[(0, 0, 0), (0, 0, 0), (255, 255, 255)]
```

Nous voyons qu'il imprime que les trois premieres cases du *board*.

Voici à quoi ressemble notre *board* à l'état initial.



Définition Show_board(board, state)

Cette définition nous permet d'afficher les différents états du jeu. Nous parcourons, les liste *state* et *state_to_board* pour aquérir à l'état de nos quatres pixels qui forment une case (soit les coordonées). Puis nous faisons correspondre son état, qui est 0,1,2 au position de la liste *colors* qui permettront d'afficher l'état du jeu par exemple : jaune qui indique que c'est le player 2 qui a selectionné la case.

In [5]:
```
def show_board(board, state):
    for y in range(len(state)):
        for x, s in enumerate(state[y]):
            c = colors[s]
            for index in state_to_board[y][x]:
                board[index] = c
    sense.set_pixels(board)
```

In [6]:
```
show_board(board, state)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call
last)
<ipython-input-6-794df6fbf837> in <module>
----> 1 show_board(board, state)

NameError: name 'board' is not defined
```

In [7]:
```
board = [
        1, 1, X, O, O, X, O, O,
        1, 1, X, O, O, X, O, O,
        X, X, X, X, X, X, X, X,
        O, O, X, O, O, X, O, O,
        O, O, X, O, O, X, O, O,
        X, X, X, X, X, X, X, X,
        O, O, X, O, O, X, O, O,
        O, O, X, O, O, X, O, O,
        ]

state =  [[1, 0, 0], [0, 0, 0], [0, 0, 0]]
def show_board(board, state):
    for y in range(len(state)):
        for x, s in enumerate(state[y]):
            c = colors[s]
            for index in state_to_board[y][x]:
```

```
                board[index] = c
    sense.set_pixels(board)
show_board(board, state)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call
last)
<ipython-input-7-1cd070a64714> in <module>
     18                 board[index] = c
     19     sense.set_pixels(board)
---> 20 show_board(board, state)

<ipython-input-7-1cd070a64714> in show_board(board, state)
     15         for x, s in enumerate(state[y]):
     16             c = colors[s]
---> 17             for index in state_to_board[y][x]:
     18                 board[index] = c
     19     sense.set_pixels(board)

NameError: name 'state_to_board' is not defined
```

Définition is_winning(p,state)

Nous définissons tous les cas ou un joueur est gagant. Pour cela il faut que l'état de trois cases selon les règles du jeu de morpion est le même état. Si c'est le cas la fonction retoune **True**, autrement **False**.

In [8]:
```
def is_winning(p, state):
    return  state[0][0] == state[0][1] == state[0][2] == p or \
            state[1][0] == state[1][1] == state[1][2] == p or \
            state[2][0] == state[2][1] == state[2][2] == p or \
            state[0][0] == state[1][0] == state[2][0] == p or \
            state[0][1] == state[1][1] == state[2][1] == p or \
            state[0][2] == state[1][2] == state[2][2] == p or \
            state[0][0] == state[1][1] == state[2][2] == p or \
            state[0][2] == state[1][1] == state[2][0] == p
```

In [9]:
```
is_winning(1, state)
```

Out[9]:
```
False
```

In [4]:
```
state =  [[1, 1, 1], [0, 0, 0], [0, 0, 0]]
show_board(board, state)
is_winning(2, state)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call
last)
<ipython-input-4-3337739bff9d> in <module>
      1 state =  [[1, 1, 1], [0, 0, 0], [0, 0, 0]]
----> 2 show_board(board, state)
      3 is_winning(2, state)
```

```
NameError: name 'show_board' is not defined
```

Définitions is_draw(state)

Si les cas de is_winnig(p,state) n'apparaissent pas dans le jeu, et que toutes les cases sont à l'état 1 ou 2 alors c'est un match nul. Si une case est à l'état 0 la fonction retourne **False**, autrement elle retourne **True**.

In [11]:
```
def is_draw(state):
    for i in state:
        for s in i:
            if s == 0:
                return False
    return True
```

In [12]:
```
state = [[1,2,1],[1,2,2],[2,1,1]]
is_draw(state)
```

Out[12]:
```
True
```

Définition play(p, board, state)

- Grâce à cette fonction nous définissons les paramètres du joueur.

- D'abord nous définissons les paramètres du curseurs à l'état initial ((x,y) = (1,1)). Mais à l'état initial aucun joueur n'a encore joué donc la couleur du curseur est divisé par deux.Nous faisons cela en parcourant la liste state_to_board qui va donc prendre chaque état des quatres pixels pour ensuite reparcourir la liste colors pour afficher l'état correspondant mais en divisant la couleur par deux.

- Ensuite nous lançons une boucle qui attend un évènement de la bibliothèque *directions* pour modifier les coordonnées du curseur. Lorsqu'un évènment est préssé il ajoute la coordonées à celles initinales, mais les coordonées ne doivent pas etre plus "grandes" que notre matrice nous faisons donc un modulo. Les coordonées sont donc modifiées mais pas forcément selectionnées pour de bon donc la case apparait encore d'une intensité de couleur diminuée.

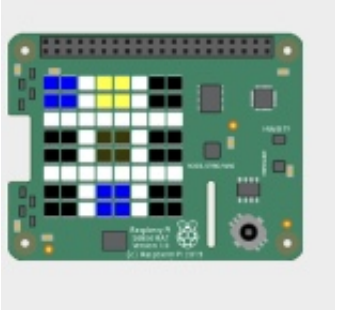
- Il ne reste plus qu'a ajouter un paramètre : celui qui ne permet pas de modifier un état losque celui ci n'est pas égal à 0 .

In [13]:
```
def play(p,board, state):
    (x, y) = (1, 1)
    dirs = {'up':(0, -1), 'down':(0, 1)
```

```
    dirs = {'up':(0, -1), 'down':(0, 1),
            'right':(1, 0), 'left':(-1, 0)}

    c = tuple(int(x/2) for x in colors[p])
    for index in state_to_board[y][x]:
        board[index] = c
    sense.set_pixels(board)

    while True :
        event = sense.stick.wait_for_event()
        if event.action == 'pressed':
            if event.direction in dirs:
                (dx, dy) = dirs[event.direction]

                x = (x + dx) % len(state)
                y = (y + dy) % len(state)

                show_board(board, state)   # eviter de colorier le ch
emin

                c = tuple(int(x/2) for x in colors[p])
                for index in state_to_board[y][x]:
                    board[index] = c
                sense.set_pixels(board)
            else:
                if state[y][x] == 0:
                    state[y][x] = p
                    show_board(board, state)
                    return

play(p, board, state)
```

```
---------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
last)
<ipython-input-13-42dc4f8cae50> in <module>
     30                     return
     31
---> 32 play(p, board, state)

NameError: name 'p' is not defined
```

Voici ce qui se passe visuellement losrqu'une partie est en cours.Ici le joeur *bleu* a finit de jouer, c'est pour cela que la case central apparait en jaune claire : le joueur n'a pas encore changer les coordonées et n'a pas préssé sur le joystick.



Dénition show_score(p)

Si les player 1 ou 2 ont gagné leur score est incrémenté de 1 et s'affiche à la fin de la partie.

In [22]:
```
def show_score(p):
    global score1, score2
    if p == 1:
        score1 += 1
    elif p == 2:
        score2 += 1
    msg = 'player1=' + str(score1) + ' player2=' + str(score2)
    sense.show_message(msg)
```

In [23]:
```
p = 1
score1 = 0
show_score(p)
```

Définition end_game(p)

Lorsque aucun joueur gagne, on affiche le message *draw* autrement c'est le numéro du joueur gagant qui est affiché, puis le score s'affiche. Pour relancer la partie le message *?* apparait pendant 3 secondes, il faut donc appuyer sur le joystick pour recommencer la partie.

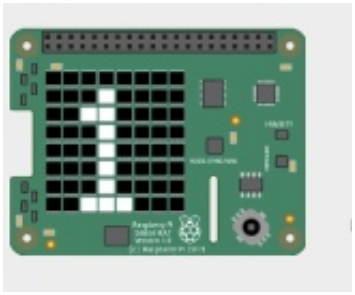
In [20]:
```

def end_game(p):
    if p == 0:
        sense.show_message("draw")
    else:
        sense.show_letter(str(p))
    sleep(3)
    show_score(p)
    return continue_game()
```

In [21]:
```
p = 1
end_game(p)
```

```
---------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
last)
<ipython-input-21-7e79c64accf0> in <module>
      1 p = 1
----> 2 end_game(p)

<ipython-input-20-f4134aff3446> in end_game(p)
      7     sleep(3)
      8     show_score(p)
----> 9     return continue_game()
```

```
NameError: name 'continue_game' is not defined
```

Voici ce qui se passe visuellement lorsqu'il affiche le gagnant à la fin du jeu :

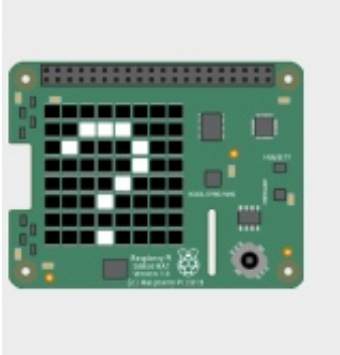


Définition continue_game()

Si pendant 3 secondes aucune action est faite avcec le joystick, alors le jeu s'éteind (**False**), autrement il rejoue une partie (**True**).

In [ ]:
```
def continue_game():
    sense.show.letter('?')
    sense.stick.get.events()
    t0 = time()
    while time() < t0 + 3:
        for event in sense.stick.get_events():
            init()
            show_board(board, sate)
            print('continue')
            return True
        print('timeout')
        return False
```

Voici ce qui se passe visuellement losrque La fonction demande pour continuer le jeu :



Définition Main()

Tout d'abord il regarde si l'état du jeu correspond à un état gagnant ou a un match nul. Puis il regarde si la réponse de la définition continue_game est True ou False pour refaire une partie. Pour intervertir les

la réponse de la définition continue_game est True ou False pour relaire une partie. Pour intervertir les joueurs entre 1 et 2 nous utilisons l'astuce `player = 3 - player` .

In [ ]:
```
def main():
    init()
    show_board(board, state)
    player = 1
    playing = True

    while playing:
        play(player, board, state)
        if is_winning(player, state):
            playing = end_game(player)

        elif is_draw(state):
            playing = end_game(0)

    player = 3 - player
```