Introduction

Tout d'abord il est nécessaire d'importer du module `sense_hat` la classe `SenseHat`. Ceci permet d'instantier un objet `sense` qui va être utilisé pour commuiquer avec la carte. Egalement, nous allons nettoyer le senshat afin d'éviter des problèmes.

In [15]:

```
# Importation des modules requis
from sense_hat import SenseHat
from random import randint
from time import sleep
from time import time

sense = SenseHat()
sense.clear(0, 0, 0)
size = 8 # Taille par defaut
```

Couleurs

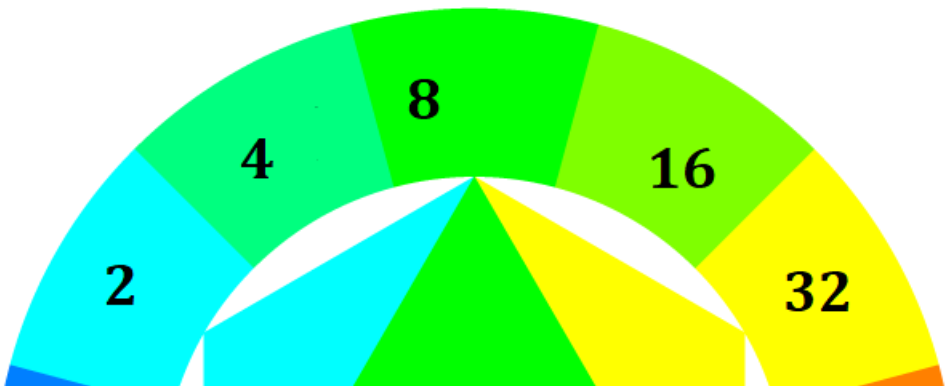
On définit les couleurs et nous créons une liste `colors` dans laquelle nous allons mettre tous les couleurs possible de notre jeu. Chaque couleur donne une valeur différent à chaque pixel.

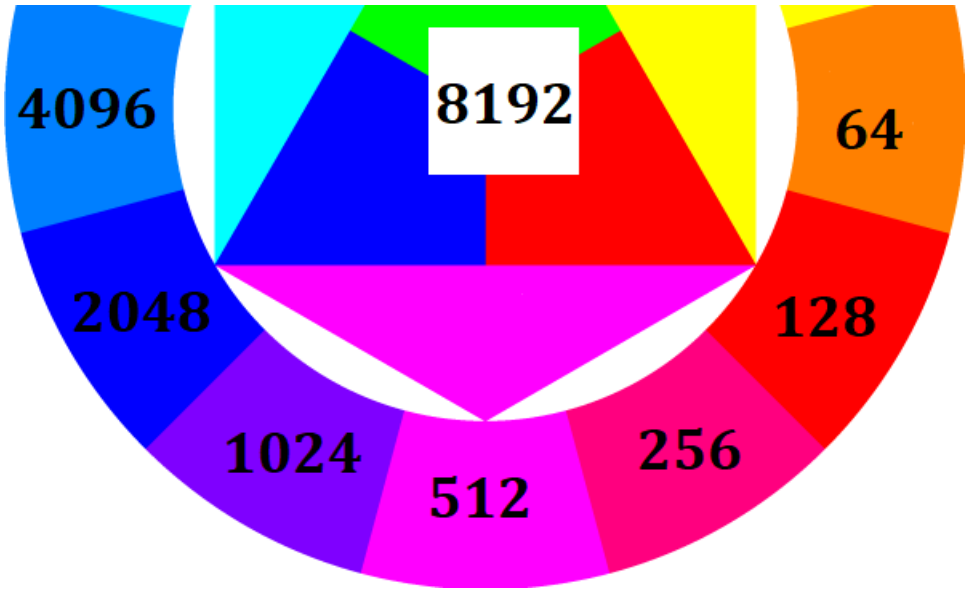
In [16]:

```
#-----Définition des couleurs-----

MESSAGE = (128, 124, 128)
BLACK_0 = (0, 0, 0)
BLUE_1 = (0, 255, 255)
GREEN_2 = (0, 255, 127)
GREEN_3 = (0, 255, 0)
GREEN_4 = (127, 255, 0)
YELLOW_5 = (255, 255, 0)
ORANGE_6 = (255, 127, 0)
RED_7 = (255, 0, 0)
PINK_8 = (255, 0, 127)
PINK_9 = (255, 0, 255)
PINK_10 = (127, 0, 255)
BLUE_11 = (0, 0, 255)
BLUE_12 = (0, 127, 255)
WHITE_13 = (255, 255, 255)
r = RED_7
o = BLACK_0
y = YELLOW_5
end = True # On définit une variable "end" comme True

colors = [BLACK_0, BLUE_1, GREEN_2, GREEN_3, GREEN_4, YELLOW_5, ORANGE_6, RED_7,\
          PINK_8, PINK_9, PINK_10, BLUE_11, BLUE_12, WHITE_13,]
```

In [ ]:

```
print(colors)
```





Matrices

Nous allons définir 3 matrices: 2 qui permettrons que notre jeu s'affiche en 4x4 et l'autre pour le mode 8x8. La matrice `L_cross` s'affiche seulement lorsque le jeu est fini et que le joueur a perdu. et `L_WIN` lorsqu'elle aura gagné.

In [17]:

```
L4 = [[0, 0, 0, 0],
      [0, 0, 0, 0],
      [0, 0, 0, 0],
      [0, 0, 0, 0],
     ]
L8 = [[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]
     ]

L_cross = [r, o, o, o, o, o, o, r,
           o, r, o, o, o, o, r, o,
           o, o, r, o, o, r, o, o,
           o, o, o, r, r, o, o, o,
           o, o, o, r, r, o, o, o,
           o, o, r, o, o, r, o, o,
           o, r, o, o, o, o, r, o,
           r, o, o, o, o, o, o, r
          ]
L_WIN = [ o, o, o, o, o, o, o, o,
          o, y, y, y, y, y, y, o,
          o, y, y, y, y, y, y, o,
          o, y, y, y, y, y, y, o,
          o, o, y, y, y, y, o, o,
          o, o, o, y, y, o, o, o,
          o, o, o, y, y, o, o, o,
          o, y, y, y, y, y, y, o,
        ]
```

In [ ]:

```
print(L4)
print(L8)
print(L_cross)
print(L_WIN)
```

Startup

Fonction qui nous permettra lancer le jeu. Elle définit également les messages affichés dans `selection_startup`

In [18]:

```
def startup():
    """Start the game"""
    global size #globalise la vairable size
    set_matrices_0()
    sense.clear()
    sense.show_message('Choose your mode:',0.1, MESSAGE)
    modes = ['4X4', '8X8'] #Modes (messages) affichés
    mode = [4, 8] #Les modes dans une liste
    sleep(0.2)
    selecting = True
    i = 0
    selection_startup(selecting, modes, mode, i)
    new_block(size)
```

In [46]:

```
print(sense.show_message)
```

```
<bound method SenseHat.show_message of <sense_hat.sense_hat.SenseHat object at 0x72031bb0>>
```

Matrice

Elle definit la taille de notre jeu.

In [19]:

```
def set_matrices_0():
    """Setting matrixes"""
    for x in range(4):
        for y in range(4):
            L4[x][y] = 0
    for x in range(8):
        for y in range(8):
            L8[x][y] = 0
```

Sélection du mode

Cette fonction permet la navigation dans les différents modes disponibles du jeu (4x4 ou 8x8).

In [20]:

```
def selection_startup(selecting, modes, mode, i):
    """Navigation to select the mode"""
    global size
    while selecting:
        sense.show_message(modes[i], 0.1, MESSAGE)
        for event in sense.stick.get_events():
            if event.action == 'pressed':
                if event.direction == 'right' or event.direction == 'left': #Gauche et droite perme
ttent de naviguer
                    i = (i + 1) % 2
                    sense.show_message(modes[i], 0.1, MESSAGE)
                elif event.direction == 'middle': #Avec le milieu nous pouvons donc choisir le mode
souhaité
                    selecting = False
                    size = mode[i]
```

Pixels

On définit la partie visuelle de notre jeu.

In [21]:

```
def set_pixels(n):
    if n == 4: #Si n = 4, alors le jeu est affiché en 4x4 sinon en 8x8
        set_pixels_4()
    else:
        for x in range(8):
            for y in range(8):
                sense.set_pixel(x, y, colors[L8[x][y]])

def set_pixels_4():
        L8_affichage = [
                        [L4[0][0], L4[0][0], L4[0][1], L4[0][1], L4[0][2], L4[0][2], L4[0][3], L4[0]
]],
                        [L4[0][0], L4[0][0], L4[0][1], L4[0][1], L4[0][2], L4[0][2], L4[0][3], L4[0]
]],
                        [L4[1][0], L4[1][0], L4[1][1], L4[1][1], L4[1][2], L4[1][2], L4[1][3], L4[1]
]],
                        [L4[1][0], L4[1][0], L4[1][1], L4[1][1], L4[1][2], L4[1][2], L4[1][3], L4[1]
]],
                        [L4[2][0], L4[2][0], L4[2][1], L4[2][1], L4[2][2], L4[2][2], L4[2][3], L4[2]
]],
                        [L4[2][0], L4[2][0], L4[2][1], L4[2][1], L4[2][2], L4[2][2], L4[2][3], L4[2]
]],
                        [L4[3][0], L4[3][0], L4[3][1], L4[3][1], L4[3][2], L4[3][2], L4[3][3], L4[3]
]],
                        [L4[3][0], L4[3][0], L4[3][1], L4[3][1], L4[3][2], L4[3][2], L4[3][3], L4[3]
]]
                        ]
        #En 4x4 le jeu est affiché de manière dans laquelle un pixel est représenté par 4 pixels.
        for x in range(8):
            for y in range(8):
                sense.set_pixel(x,y, colors[L8_affichage[x][y]])
```

Blocks

On regarde si on peut ajouter un ou deux blocks et s'il est possible de le faire. Car il peut exister aucune place disponible donc on appelle `control_end`

In [22]:

```
def new_block(n):
    """Create a new block"""
    sleep(0.25)
    i = number_empty_block(n)
    print (i)
    if i > 1:
        two_new_blocks(n)
    elif i == 1:
        one_new_block(n)
    control_end(n)
    set_pixels(n)

def number_empty_block(n):
    """Number of empty block"""
    L = L4 if n == 4 else L8
    i = 0
    for x in range(n):
        for y in range(n):
            if L[x][y] == 0:
                i = i + 1
    return i

def two_new_blocks(n):
    """Add two new blocks"""
    r = randint(0,1)
    L = L4 if n == 4 else L8
    while r < 2: #tant qu'on en a pas créé 2
        x = randint(0, (n - 1))#
        y = randint(0, (n - 1))
        # On choisis aléatoirement une ligne et une colonne
        if L[x][y] == 0:# On controle si ce pixel est vide
           L[x][y] = 1 # On défini un bloc de couleur correspondant au chiffre 2
           r = r + 1# Si le bloc est créé on indente pour créé exactement 2 nouveaux pixels

def one_new_block(n):
    """Add only one block"""
```

```
    Add only one block
    r = randint(0, 1)
    L = L4 if n == 4 else L8
    while r < 1: #tant qu'on en a pas créé 2
        x = randint(0, (n - 1))#
        y = randint(0, (n - 1))# On choisis aléatoirement une ligne et une colonne
        if L[x][y] == 0:# On controle si ce pixel est vide
            L[x][y] = 1 # On défini un bloc de couleur correspondant au chiffre 2
            r = r + 1
```

Joystick

Il faut finalement définir les réactions de notre joystick afin de pouvoir intéreagir avec le jeu.

In [23]:

```
def moved_up(n):
    """Reacts to the joystick pushed up."""
    print(L4)
    L = L4 if n == 4 else L8
    for x in range(n):
        for y in range(n):# Sur chaque pixel en prenantles pixels en ligne puis en colonne
            if L[x][y] > 0 and y >= 1:# On controle que le pixel ne soit pas une case vide
                move_pixel_up(x, y, n)
    set_pixels(n)
    print(L4)
    new_block(n)

def move_pixel_up(x, y, n):
    """Move up the pixel in the matrix"""
    L = L4 if n == 4 else L8
    while L[x][y - 1] == 0 and y >= 1:# Si la case est vide
        L[x][y - 1] = L[x][y]
        L[x][y] = 0
        y = y - 1
    if L[x][y - 1] == L[x][y]:
        L[x][y - 1] = L[x][y - 1] + 1
        L[x][y] = 0

def moved_down(n):
    """Reacts to the joystick pushed down."""
    L = L4 if n == 4 else L8
    for x in range(n):
        for z in range(n - 1):
            y = n - 2 - z
            if L[x][y] > 0 and y <= (n - 2):# On controle que le pixel ne soit pas une case vide
                move_pixel_down(x, y, n)
    set_pixels(n)
    new_block(n)

def move_pixel_down(x, y, n):
    """Move down the pixel in the matrix"""
    L = L4 if n == 4 else L8
    while y <= (n - 2) and L[x][y + 1] == 0:# Si la case est vide
        L[x][y + 1] = L[x][y]
        L[x][y] = 0
        y = y + 1
    if y < (n - 1) and L[x][y + 1] == L[x][y]:
        L[x][y + 1] = L[x][y + 1] + 1
        L[x][y] = 0

def moved_left(n):
    """Reacts to the joystick pushed left."""
    L = L4 if n == 4 else L8
    for y in range(n):
        for x in range(n):
            if L[x][y] > 0:# On controle que le pixel ne soit pas une case vide
                move_pixel_left(x, y, n)
    set_pixels(n)
    new_block(n)

def move_pixel_left(x, y, n):
    """Move left the pixel in the matrix"""
    L = L4 if n == 4 else L8
    while x > 0 and L[x - 1][y] == 0:# Si la case est vide
```

```
            L[x - 1][y] = L[x][y]
            L[x][y] = 0
            x = x - 1
        if L[x - 1][y] == L[x][y]:
            L[x - 1][y] = L[x - 1][y] + 1
            L[x][y] = 0

def moved_right(n):
    """Reacts to the joystick pushed right."""
    L = L4 if n == 4 else L8
    for y in range(n):
        for z in range(n - 1):
            x = n - 2 - z
            if L[x][y] > 0 and x < (n - 1):
                move_pixel_right(x, y, n)
    set_pixels(n)
    new_block(n)

def move_pixel_right(x, y, n):
    """Move right the pixel in the matrix"""
    L = L4 if n == 4 else L8
    while x < (n - 1) and L[x + 1][y] == 0:
        L[x + 1][y] = L[x][y]
        L[x][y] = 0
        x = x + 1
    if x < (n - 1) and L[x + 1][y] == L[x][y]:
        L[x + 1][y] = L[x + 1][y] + 1
        L[x][y] = 0
```

Contrôle

On doit également ajouter des fonctions permettant de définir l'état du jeu. `control_end` nous dira s'il est encore possible d'ajouter des nouveaux blocks. `check_empty_cells` regardera si une cellule est libre ou non. Finalement, `check_neigbors` va définir l'état des cellules au centre et au bord du SenseHat

In [24]:

```
def control_end(n):
    """Returns True when the game is finished."""
    global end
    end = True
    L = L4 if n == 4 else L8
    check_empty_cells(n)
    check_neigbors_cells_for_center(n)
    check_neigbors_cells_for_border(n)
    if end == True:
        end_animation(n)
    else:
        control_victory(n)

def check_empty_cells(n):
    global end
    """Check if a cell is empty or not"""
    L = L4 if n == 4 else L8
    for x in range(n):
        for y in range(n):
            if L[x][y] == 0:
                end = False

def check_neigbors_cells_for_center(n):
    global end
    """Check the state of neighbours cells (only cells in the center)"""
    L = L4 if n == 4 else L8
    if end == True:
        for x in range(1, n - 1):
            for y in range(1, n - 1):
                if L[x][y] == L[x][y + 1] or L[x][y] == L[x + 1][y] \
                    or L[x][y] == L[x - 1][y] or L[x][y] == L[x][y - 1]:
                    end = False

def check_neigbors_cells_for_border(n):
    global end
    """Check the state of neighbours cells (only cells in the border)"""
    L = L4 if n == 4 else L8
```

```
    if end == True:
        for y in range(n - 1):
            for x in range(n - 1):
                if L[0][x] == L[0][x + 1] or L[x][0] == L[x + 1][0] \
                    or L[n - 1][x] == L[n - 1][x + 1] or L[x][n - 1] == L[x + 1][n - 1]:
                    end = False
```

Fin du jeu

Pour conclure, on doit définir les différentes fins possibles de notre jeu. `end_nimation` s'affichera lorsque le joueur aura perdu (et le score avec). `losser_animation_part(1 ou 2)` . `score_calculator` calcule le score à montrer grâce à `show_score`. `control_victory` nous aide à voir si le 14ème bloc a été atteint. Si c'est le cas `victory` fait son apparition en montrant un message de félicitation. Et `exit` nous permet de sortir du jeu lorsqu'on le souhaite en appuyant deux fois sur le joystick direction `middle`.

In [25]:

```
def end_animation(n):
    """Show a message when the player loses the game and show the score"""
    loser_animation_part_1(n)
    score_calculator(n)
    sense.show_message('You lose... Your score is:', 0.075, MESSAGE)
    show = True
    show_score()
    main()

def loser_animation_part_1(n):
    """First animation of a game lost"""
    set_pixels(n)
    sleep(3)
    r = RED_7
    o = BLACK_0
    sense.clear()
    loser_animation_part_2(n)

def loser_animation_part_2(n):
    """Animation of a red cross when the game is over"""
    for i in range(5):
        sense.set_pixels(L_CROSS)
        sleep(0.1)
        sense.clear()
        sleep(0.1)
    sense.set_pixels(L_CROSS)
    sleep(1)
    set_pixels(n)
    sleep(2)

def score_calculator(n):
    """Calculate the score shown"""
    L = L4 if n == 4 else L8
    score = 0
    for x in range(n):
        for y in range(n):
            if L[x][y] != 0:
                score = score + 2 ** L[x][y]

def show_score():
    """Show the score"""
    while show:
        score = str(score)
        string = score + 'pts'
        sense.show_message(string, 0.1, MESSAGE)
        sense.show_message('Press to end', 0.075, MESSAGE)
        for event in sense.stick.get_events():
            if event.action == 'pressed':
                show = False

def exit():
    """Use to exit the game"""
    t0 = time()
    while time() < t0 + 1:
        for event in sense.stick.get_events():
            if event.action == 'pressed' and event.direction == 'middle':
```

```
                show_message = True
                while show_message:
                    sense.show_message('Press to return to the menu', 0.075, MESSAGE)
                    for event in sense.stick.get_events():
                        if event.action ==  'pressed':
                            show_message = False
                            main()

def control_victory(n):
    """Control if the maximum is reached (14th block)"""
    L = L4 if n == 4 else L8
    for x in range(n):
        for y in range(n):
            if L[x][y] == 14:
                sense.set_pixels(L_WIN)
                victory(n)
    set_pixels(n)


def victory(n):
    """Show the message when the player wins"""
    sleep(9)
    score_calculator(n)
    sense.show_message('Congratulations, you just reached the highest block. Your score is :', 0.0
75, MESSAGE)
    show_score
    main()
```

In [ ]:

```
victory()
```

Assignation des commandes

Pour finir, nous devons assigner à chaque mouvement de joystick, sa fonction correspondante.

In [26]:

```
def joystick_direction():
    """Definition of direction"""
    if event.direction == 'up':
        moved_up(size)
    elif event.direction == 'down':
        moved_down(size)
    elif event.direction == 'right':
        moved_right(size)
    elif event.direction == 'left':
        moved_left(size)
    elif event.direction == 'middle':
        exit()
```

Main menu

Il faut finalement ajouter une dernière fonction permettant l'utilisation du jeu 2048 dans un module qui rassemble tous les jeux de la classe.

In [ ]:

```
def main():
    """Main menu"""
    startup()
    running = True
    while running:
        for event in sense.stick.get_events():
            if event.action == 'pressed':
                if event.direction == 'up':
                    moved_up(size)
                elif event.direction == 'down':
                    moved_down(size)
                elif event.direction == 'right':
                    moved_right(size)
```

```
                elif event.direction == 'left':
                    moved_left(size)
                elif event.direction == 'middle':
                    exit()


if __name__ == '__main__':
    main()
```

In [ ]:

```

```