Jeu du démineur

But :

Le but de ce mini-jeu est de découvrir toutes les cases sauf celles qui contiennent des bombes.

Principes :

Lorsque l'on active une case qui n'est pas une bombe, une fonction compte le nombre de bombes présentes sur les huit cases adjacentes. Ce nombre est indiqué sur la case, dans notre cas, chaque chiffre correspond à une couleur prédéfinie. Cela permet de détecter la place des bombes. Tandis que lorsqu'on active une case contenant une bombe cette dernière explose et met fin à la partie.

Notre programme :

Notre programme est écrit en python3 et se joue principalement sur un raspberry pi muni d'un sense hat mais peut aussi être jouer sur n'importe quel ordinateur grâce au module sense emu, qui permet de crée une réplique d'un sense hat.

Nos modules :

Pour commencer la première chose à faire est d'importer les modules nécessaires à notre jeu.

In [1]:

```
##########################################
#      We import the different modules      #
##########################################

# We import the library "Sensehat"
from sense_emu import SenseHat

# We import the module "random"
from random import randint, choice

# We import the module "time"
from time import sleep, time
```

Nos variables :

Après nous devons initialiser les variables du jeu comme par exemple les couleurs, la valeur de x et y qui permettent de se déplacer sur le plateau de jeu ainsi que la variable sense qui fait le lien avec le SenseHat.

In [2]:

```
##########################################
#     Definition of the game variables      #
##########################################

# We define the colors of the game:
green = (0,255,0)        # Flag
grey = (90,94,107)       # Box undiscovered
white = (255,255,255)    # Box without bomb
yellow = (247,255,60)    # 1 bomb around
orange = (255,127,0)     # 2 bombs around
corail = (231,62,1)      # 3 bombs around
red = (248,0,0)          # 4 bombs around
bordeau = (91,60,19)     # 5 bombs around
black = (0,0,0)          # Bomb

# We initialize the module "sense hat" in the variable "sense":
sense = SenseHat()
sense.clear()

##########################################
```

```
#    Initialization of golbal variables     #
#############################################

# The variable containing the flags list:
list_flag = []

# The variable containing the bombs list:
List_bomb = []

# The variable containing the checked boxes:
list_checked = []

# The variable containing the number of bombs:
nb_bombs = 7

# Displacement variables:
game = True
running = True
x = 0
y = 0

# Color of the board:
old_color = grey
```

```
/usr/lib/python3/dist-packages/sense_emu/sense_hat.py:91: Warning: No emulator detected; spawning
sense_emu_gui
  warnings.warn(Warning('No emulator detected; spawning sense_emu_gui'))
```

Nos fonctions :

Ensuite, un programme est très régulièrement divisé en fonctions. C'est une suite d'instructions qui s'effectue lorsque la fonction est appelée. Cela permet de répéter plusieurs les mêmes actions sans écrire le même code plusieurs fois. Pour notre jeu, nous avons écrit une dizaine de fonctions.

Fonction qui place les bombes :

Une des fonctions les plus importante pour notre jeu est celle qui place les bombes de manière aléatoire sur le plateau de jeu.

In [4]:

```
# Function that choose the places of the bombs:
def choose_bomb(nb_bomb):

    list_bomb = []

    i = 0
    while i < nb_bomb :
        x_bomb = randint(0,7)
        y_bomb = randint(0,7)
        xy_bomb = 10 * x_bomb + y_bomb
        if(xy_bomb in list_bomb):
            i = i
        else:
            i += 1
            list_bomb.append(xy_bomb)

    return list_bomb;
```

Cette fonction prend comme paramètre le nombre de bombe qu'elle doit placer sur le plateau. Lorsque deux bombes ont la même position, elle choisit une autre case pour l'une d'entre elles.

Fonctions qui compte les bombes :

Il y a une autre fonction indispensable pour notre jeu, il s'agit de celle qui compte le nombre de bombes qui se trouvent autour d'une certaine case. Cette case est transmise à la fonction grâce à deux paramètres x et y.

In [3]:

```
#############################################
```

```
#     Definition of the game functions      #
##############################################

# Functions that count the number of bomb around:
def count_bombs_around(x,y):

    global List_bomb
    nb = 0

    x_check = x+1
    y_check = y
    xy_ckeck = 10*x_check + y_check

    if xy_ckeck in List_bomb:
        nb += 1

    x_check = x
    y_check = y+1
    xy_ckeck = 10*x_check + y_check

    if xy_ckeck in List_bomb:
        nb += 1

    x_check = x+1
    y_check = y+1
    xy_ckeck = 10*x_check + y_check

    if xy_ckeck in List_bomb:
        nb += 1

    x_check = x-1
    y_check = y
    xy_ckeck = 10*x_check + y_check

    if xy_ckeck in List_bomb:
        nb += 1

    x_check = x
    y_check = y-1
    xy_ckeck = 10*x_check + y_check

    if xy_ckeck in List_bomb:
        nb += 1

    x_check = x-1
    y_check = y-1
    xy_ckeck = 10*x_check + y_check

    if xy_ckeck in List_bomb:
        nb += 1

    x_check = x+1
    y_check = y-1
    xy_ckeck = 10*x_check + y_check

    if xy_ckeck in List_bomb:
        nb += 1

    x_check = x-1
    y_check = y+1
    xy_ckeck = 10*x_check + y_check

    if xy_ckeck in List_bomb:
        nb += 1

    return nb;
```

Cette fonction retourne le nombre de bombes qu'elle a repéré.

Fonction qui initialise la couleur du plateau :

Cette fonction doit, lorsque l'on commence ou recommence le jeu, remettre toutesles cases avec la couleur d'origine. Dans notre cas, nous avons choisi la couleur grise.

In [5]:

```
# Initialization of game board:
def set_all_in_gray():
    G = grey

    set_color_grey = [
    G,G,G,G,G,G,G,G,
    G,G,G,G,G,G,G,G,
    G,G,G,G,G,G,G,G,
    G,G,G,G,G,G,G,G,
    G,G,G,G,G,G,G,G,
    G,G,G,G,G,G,G,G,
    G,G,G,G,G,G,G,G,
    G,G,G,G,G,G,G,G,
    G,G,G,G,G,G,G,G]

    sense.set_pixels(set_color_grey)
```

Fonctions de propagation :

Nous arrivons maintenant à une des partie les plus complexes que nous avons dû programmer. Il s'agit de la propagation des cases. Lorsque l'on utilise une case vierge, autrement dit sans bombes à proximité, une fonction va décacher toutes les cases vierges autour et va s'arreter à chaque fois qu'une case sera colorée. Pour cela, nous avons créé deux fonctions dont une qui est récursive, c'est à dire qu'elle s'appelle elle-même

La première va créer la liste des cases concernées par la propagation :

In [ ]:

```
def check_case(x,y):
    global list_checked

    if ((x+1)*10 + y) not in list_checked:
        if count_bombs_around(x+1,y) == 0 and -1<(x+1)<8 and -1<y<8:
            list_checked.append((x+1)*10+y)
            check_case(x+1,y)
        else:
            list_checked.append((x+1)*10+y)


    if ((x+1)*10 + y+1) not in list_checked:
        if count_bombs_around(x+1,y+1) == 0 and -1<(x+1)<8 and -1<(y+1)<8:
            list_checked.append((x+1)*10+(y+1))
            check_case(x+1,y+1)
        else:
            list_checked.append((x+1)*10+(y+1))

    if ((x)*10 + y+1) not in list_checked:
        if count_bombs_around(x,y+1) == 0 and -1<x<8 and -1<(y+1)<8:
            list_checked.append(x*10+(y+1))
            check_case(x,y+1)
        else:
            list_checked.append(x*10+(y+1))

    if ((x-1)*10 + y+1) not in list_checked:
        if count_bombs_around(x-1,y+1) == 0 and -1<(x-1)<8 and -1<(y+1)<8:
            list_checked.append((x-1)*10+(y+1))
            check_case(x-1,y+1)
        else:
            list_checked.append((x-1)*10+(y+1))

    if ((x-1)*10 + y) not in list_checked:
        if count_bombs_around(x-1,y) == 0 and -1<(x-1)<8 and -1<y<8:
            list_checked.append((x-1)*10+y)
            check_case(x-1,y)
        else:
            list_checked.append((x-1)*10+y)

    if ((x-1)*10 + y-1) not in list_checked:
        if count_bombs_around(x-1,y-1) == 0 and -1<(x-1)<8 and -1<(y-1)<8:
            list_checked.append((x-1)*10+(y-1))
            check_case(x-1,y-1)
        else:
            list_checked.append((x-1)*10+(y-1))
```

```
            list_checked.append((x-1)*10+(y-1))

    if ((x)*10 + y-1) not in list_checked:
        if count_bombs_around(x,y-1) == 0 and -1<x<8 and -1<(y-1)<8:
            list_checked.append(x*10+(y-1))
            check_case(x,y-1)
        else:
            list_checked.append(x*10+(y-1))

    if ((x+1)*10 + y-1) not in list_checked:
        if count_bombs_around(x+1,y-1) == 0 and -1<(x+1)<8 and -1<(y-1)<8:
            list_checked.append((x+1)*10+(y-1))
            check_case(x+1,y-1)
        else:
            list_checked.append((x+1)*10+(y-1))

    return list_checked;
```

Et la seconde va simplement colorer de la bonne couleur (par rapport au nombre de bombes à proximité) les cases extraites de la liste crée par la première fonction :

In [ ]:

```
# Function that color the box during the spread
def color_case(list):

    i = 0
    while i < len(list):
        y = list[i] % 10
        x = int((list[i] - y) / 10)
        i += 1
        print(x)
        print(y)

        nb = count_bombs_around(x,y)

        if -1 < x < 8 and -1 < y < 8:
            if nb == 0:
                sense.set_pixel(x,y,white)
            elif nb == 1:
                sense.set_pixel(x,y,yellow)
            elif nb == 2:
                sense.set_pixel(x,y,orange)
            elif nb == 3:
                sense.set_pixel(x,y,corail)
            elif nb == 4:
                sense.set_pixel(x,y,red)
            elif nb == 5:
                sense.set_pixel(x,y,bordeau)
            else:
                sense.set_pixel(x,y,black)
```

Fonction qui termine le jeu :

Cette fonction a pour but de vérifier que le jeu n'est pas fini. Elle compte toutes les cases encores cachées. Tant que ce nombre est plus grand que le nombre de bombes, le jeu n'est pas fini tandis que si ce nombre est égal au nombre de bombes, la partie est terminée et la fonction affiche le message "You won" puis elle relance une partie. Cette fonction est appelée après chaque action.

In [ ]:

```
# Function that check for every action if the game is over:
def end_game():

    x = -1
    y = -1
    nb = 0

    for i in range(8):
        x += 1
        for j in range(8):
            y += 1
            if sense.get_pixel(x,y) == [88,92,104]:
                nb += 1
```

```
        y = -1

    if nb == nb_bombs:
        sense.show_message("You won")
        # We restart the game:
        new_game(nb_bombs)
```

Fonction qui lance le jeu :

La fonction new_game a comme objectif de tout initialiser pour le début du jeu. Elle génère une nouvelle liste de bombes, remet les variables à zéro et réinitialise le plateau de jeu. Elle utilise des fonctions définie précédement.

In [ ]:

```
def new_game(nb):

    # We select the external variable that will contain the bombs list
    global List_bomb

    # Random placement of bombs
    List_bomb = choose_bomb(nb)

    print(List_bomb)

    # We initialize the game board
    set_all_in_gray()


    # Initialization of deplacement variables
    global x
    global y

    x = 0
    y = 0
```

Fonction qui utlise une case :

Pour finir, cette dernière fonction est utlisée lorsque l'on appuie sur le bouton du milieu du joystick. Elle sert à sélectionner une case à propos de laquelle le joueur ne pense pas qu'il y a une bombe. Elle va regarder si c'est une bombe, si c'est le cas elle termine le jeu et relance une nouvelle partie, sinon, elle propage les cases si besoin puis elle colore la ou les case(s) avec la couleur correspondant au nombre de bombes à proximité.

In [ ]:

```
# Function that performs the action on the selected cell:
def use_case(x,y):

    global List_bomb

    xy = x * 10 + y

    if sense.get_pixel(x,y) == [88,92,104]:
        # We check if it is a bomb :
        if xy in List_bomb and xy not in list_flag:
            sense.clear(255,0,0)
            sleep(5)
            # We restart the game:
            new_game(nb_bombs)
        else:
            # If there is no bomb around, we start the spread :
            if count_bombs_around(x,y) == 0:
                list_case_checked = check_case(x,y)
            else:
                xy = x * 10 + y
                list_case_checked = [xy]
            # After the end of the spread, we set a color to all the boxes
            color_case(list_case_checked)
            # At the end of use_case we reset the list for the next use :
            list_checked = []

            # At the end of use_case we check if the game is over
            end_game()
```

Gestion du joystick :

Cette partie est très importante car il s'agit de celle qui va gérer le déplacement sur le plateau ainsi que le fait d'effectuer des actions. Les mouvements *"haut"*, *"bas"*, *"droit"* et *"gauche"* permettent de déplacer la case sélectionnée. Le bouton *milieu"* quant à lui permet d'utiliser la case.

In [ ]:

```
##########################################
#            Start of the game           #
##########################################

# We use the function of game start:
new_game(nb_bombs)

# Part that manages the deplacement with joystick:
while game:
    while running:
        for event in sense.stick.get_events():
            if event.action == 'held' and event.direction == 'middle':
                xy_flag = x * 10 + y
                if xy_flag not in list_flag:
                    list_flag.append(xy_flag)
                    sense.set_pixel(x,y,green)
                else:
                     list_flag.remove(xy_flag)

            if event.action == 'pressed':
                sense.set_pixel(x,y,old_color)
                if event.direction == 'down' and y < 7:
                    y = y + 1
                    old_color = sense.get_pixel(x,y)
                elif event.direction == 'up' and y > 0:
                    y = y - 1
                    old_color = sense.get_pixel(x,y)
                elif event.direction == 'right' and x < 7:
                    x = x + 1
                    old_color = sense.get_pixel(x,y)
                elif event.direction == 'left' and x > 0:
                    x = x - 1
                    old_color = sense.get_pixel(x,y)
                elif event.direction == 'middle':
                    use_case(x,y)
                    running = False

        if running == True:
            # On crée le clignotement :
            t = int(3 * time())
            if t % 2 == 0:
                sense.set_pixel(x,y,old_color)
            else:
                sense.set_pixel(x,y,green)

    if running == False:
        for event in sense.stick.get_events():
            if event.action == 'pressed':
                if event.direction == 'down' or event.direction == 'up' or event.direction == 'righ
t' or event.direction == 'left':

                    running = True
                    if event.direction == 'down' and y < 7:
                        y = y + 1
                    elif event.direction == 'up' and y > 0:
                        y = y - 1
                    elif event.direction == 'right' and x < 7:
                        x = x + 1
                    elif event.direction == 'left' and x > 0:
                        x = x - 1
```

Améliorations à apporter :

Les points suivants pourraient être améliorer ou ajouter à notre jeu :

- Possibilité de choisir le nombre de bombes en début de partie.
- Système de niveau de difficulté avec progression.
- Simplifier et compacter le code.