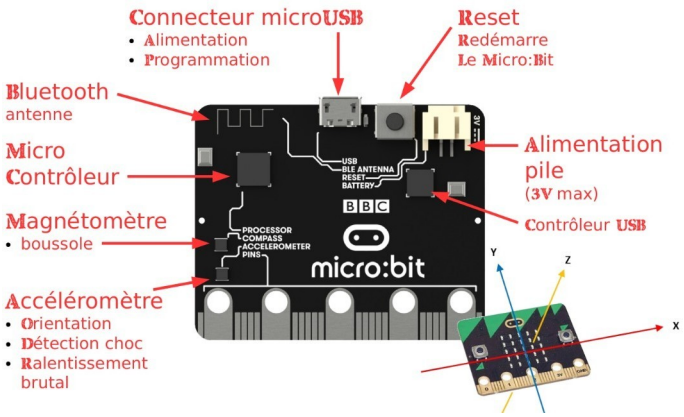
GOUGUEULE AGENDA

PIN Tom, BILBAUT Tom, SEPHANH Cédric

Description du projet

Le GOUGUEULE AGENDA est un prototype d'agenda permettant de voir rapidement et clairement les jours lesquels nous sommes disponibles. On utilise des fichiers stockant les différents événements pour détérminer la couleurs des différentes LEDs du bandeau.



Liste du matériel utilisé

- 1 carte Micro:bit V1.5
- 1 shield Grove pour Micro:bit
- 1 module Grove ruban 30 leds

Fichiers

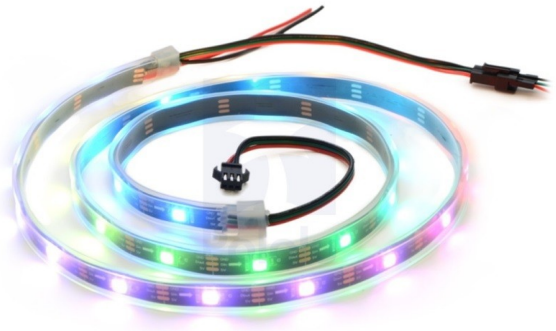
Le projet comporte deux fichiers différents :

- projet_nsi_fichier_fonctions.py
- afficher_leds_bandeau.py

Le premier fichier contient toute les fonctions nécessaires au bon fonctionnement de l'agenda et le second est le programme à exécuter, il contient la fonction show_leds() qui permet d'activer les leds du ruban.

Ruban leds

→ description fonctionnelle



Le ruban leds Neopixel dispose de 30 leds adressables. La couleur de chaque led peut être contrôlée individuellement par trois valeurs correspondants aux 3 couleurs primaires : rouge, vert et bleu.

→ Pilotage du ruban de leds :

En début de programme on crée une occurrence de l'objet Neopixel et on lui spécifie ses paramètres soit le port sur lequel il est connecté (pin) et le nombre de leds du ruban.

```
>>> np=neopixel.NeoPixel(pin1, 30)
```

Les principales méthodes de la librairie Neopixel sont :

```
>>> np.clear()          # pour éteindre toutes les leds
>>> np.clear(n)         # pour éteindre la led n
>>> np.show()           # pour allumer les leds,
>>> np.show(n)          # pour allumer la led n
```

Pour assigner une couleur à une led n on attribuer le tuple (rouge, vert, bleu) à notre objet np (qui se comporte ici comme une liste de tuples)

```
>>> np[n]=(50,0,255)
```

Pour le projet, on récupère les trois valeurs x, y et z de l'accéléromètre (comprises entre -2048 et +2048) et on transforme ces valeurs pour qu'elles soient comprises entre 0 et 255.

La gestion du temps

Pour gérer le temps, on utilise la librairie *time*, elle permet de manipuler tout ce qui touche au temps et de récupérer la date actuelle par exemple.

→ Principales fonctions:

```
>>> localtime() #Récupère la date actuelle
>>> localtime(ts) #Récupère la date actuelle à partir d'un timestamp
>>> mktime(date) #Transforme une date en timestamp
```

La librairie *time* contient un partie concernant le timestamp, c'est le nombre des secondes écoulées depuis le 1er Janvier 1970 à minuit précise. Il est utilisé dans toutes les machines qui ont besoin d'une gestion du temps. La valeur renvoyée par localtime(ts) doit être stockée dans une variable :

```
>>> var = localtime().tm_mday #Récupère le numéro du jour de la date actuelle
```

Il existe aussi des methodes pour récupérer l'année le mois etc. : tm_sec, tm_min, tm_mon, tm_year

On peut donc manipuler les dates en ajoutant le nombre de secondes que l'on veut. Par exemple, pour avancer d'une heure, on ajoute 3600 secondes.

```
>>> var += 3600 #avance le timestand de 1h
```

Code Python du projet

projet_nsi_fichier_fonctions.py

```
import microbit
from time import *

class day():
    """classe créant un objet correspondant
    à une journée : un jour, ce que l'on prévoit de faire/où/quelle h"""

    def __init__(self, jour, activity=[]): # je creer un objet de la class day avec
                                    # les paramètres jour (string) et activity(liste de 4 objets)
        self.date = jour                # l'objet créé a pour date jour
        self.event = activity[0]          # pour event le premier objet de la liste activity
        self.place = activity[1]          # pour place le deuxieme objet de la liste activity
        self.time = activity[2]           # pour time le troisieme objet de la liste activity
        self.period = activity[3]         # pour period le quatrieme objet de la liste activity


def get_events(date):
    """fonction prenant en parametre la date(str) et retournant les événements du jour
date(list)"""
    with open("{}".format(date+".event"), "rb") as info_n:      # on ouvre le fichier
correspondant a la date
        paper = info_n.read()                                   # on recupere le contenu de ce
fichier dans la variable paper
    return paper.split(":")


def get_event(date, n):
  """fonction prenant en parametre la date(str) et le numero de l'événement n(int)
  retournant les caractéristiques de l'événement numero n"""
  with open("{}".format(date+".event"), "rb") as info_n:
        paper = info_n.read()
```

```
  return paper.split(":")[n]


def add_event(date, event):
    """fonction prenant en parametres une date(str) et un evenement(list)
        créant un fichier avec la date contenant le nouvel evenement si aucun
        evenement n'est prevu à cette
        date et ajoutant l'evenement au fichier du jour date si un evenement existe déjà
        pour celui-ci"""
    try:                                            # a l'aide du try
        get_events(date)     # on teste l'instruction get_event(date) (on teste si il y a deja
                                                    # un fichier correspondant a ce jour)
    except:                                          # si elle leve une exception (une erreur)
                                                    # on execute alors le code intendé en
dessous du except
        ev = event.split("/")                               # on stocke les caracteristiques de
l'evenement event dans la liste ev
        jour = day(date, ev)                                # on cree un objet de la classe day
        with open("{}".format(date+".event"), "wb") as day_n:       # on creer(comme il
n'existe pas encore) le fichier correspondant a la date
            day_n.write(date + ":")                          # on ecrit la date dans le fichier
            day_n.write(event)                              # on ecrit ensuite les
caracteristiques de l'evenement
    else:                                               # sinon(si un fichier correspondant deja
au jour date) on execute le code intende
        ev = event.split("/")
        with open("{}".format(date+".event"), "ab") as day_n:       # on ouvre(comme il
existe deja) le fichier correspondant a la date
            day_n.write(":" + event)                         # on rajoute (ab ligne 48) le
nouvel evenement a la suite du fichier


def modifier_event(date, modif, new_data):
    """fonction prenant en parametres une date(str), le numero de l'evenement à
        modifier(int)
    et la nouvelle valeur de remplacement(str) et remplacant la valeur à modifier par
    la nouvele dans le fichier date"""
    list_1 = get_events(date)                           # on stocke les evenements du
jour date dans la liste list_1
    list_1[modif] = new_data                            # on remplace l'evenement
numero modif dans la liste par newdata
    with open(date+".event", "wb") as nv_fichier:
        nv_fichier.write(":".join(list_1))                  # on ecrit la liste avec la
modification a la place de la precedentes (wb ligne 58)
```

```
def supp_event(date, elemsupp):
    """fonction prenant en parametre un date(str) et le numero d'un evenement a
        supprimer(int)
    puisle supprime du fichier date"""
    list_1 = get_events(date)
    del list_1[elemsupp]                    # on supprime l'evenement souhaite de la liste
    #j'écris le nouveau contenu
    with open(date+".event", "wb") as nv_fichier:
        nv_fichier.write(":".join(list_1))                    # on ecrit dans le fichier la
nouvelle liste (sans l'element supprime)


local_date = mktime(localtime())
show_date = mktime(localtime())
for i in range(localtime(show_date).tm_mon, 30):
    show_date -= 86400


def test_event(date):
    """Fonction prenant en parametre une date (jj-mm-aaaa) et renvoyant 0 si il n'y a
        pas d'évenement ou 1 si il y a des evenements."""
    try:                                                # instruction try deja expliquee
        get_events(date)
    except:
        return 0
    else:
        return 1


def dayInMon(ts):
    """Cette fonction prend en paramètre le timstamp d'un jour et renvoie le nombre de
jours dans son mois en timestamp."""
    mon = localtime(ts).tm_mon
    day = localtime(ts).tm_mday
    for i in range(day, 31):
        ts += 86400
    if localtime(ts).tm_mday == 31:
        return 2678400
    elif localtime(ts).tm_mday == 1:
        return 2592000
    else:
        return 2419200
```

afficher_leds_bandeau.py

```
from microbit import *
import time
from neopixel import NeoPixel
from projet_nsi_fichier_fonctions import *

nb_led = 31
bandeau_led = NeoPixel(microbit.pin0, nb_led)   # Cree une instance pour piloter 10 WS2812b

dix = Image("90999:""90909:""90909:""90909:""90999:")
onze = Image("90090:""90090:""90090:""90090:""90090:")
douze = Image(  "90090:""90909:""90009:""90090:""90999:")

def showLeds(ts_date):
    """Fonction prenant en parametre le timestamp d'un mois
    et allume les leds en rouge si il n'y a pas d'évenements,
    en vert si il y a un ou des évenements et ne s'allume pas si il n'y a pas de jour"""
    mois = localtime(ts_date).tm_mon
    year = localtime(ts_date).tm_year
    if dayInMon(ts_date) == 2678400:
        nb_led=31
    elif dayInMon(ts_date) == 2592000:
        nb_led=30
    else:
        nb_led=28

    for i in range(1, nb_led+1):
        if test_event(str(i) + "-" + str(mois) + "-" + str(year)) == 0:
            bandeau_led[i-1] = (255, 0, 0)  # Configure les Leds en rouge
        else:
            bandeau_led[i-1] = (0, 255, 0)  # Configure les Leds en vert
        if dayInMon(ts_date) == 2592000:
            bandeau_led[30] = (0, 0, 0) # Eteint la dernière led
        elif dayInMon(ts_date) == 2419200:
            for i in range(28, 31):
```

```
            bandeau_led.clear(i)  # Eteint les 3 dernières led

    bandeau_led.show()  # Affiche toutes les Leds

showLeds(show_date)

while True:
    if microbit.button_a.was_pressed():                #si nous pressons le bouton a (celui de gauche)
        show_date -= dayInMon(show_date)               #recule l'information d'un mois
        showLeds(show_date)                            #change la bande led pour le bon mois
    if microbit.button_b.was_pressed():                #si nous pressons le bouton b (celui de droite)
        show_date += dayInMon(show_date-dayInMon(show_date))   #avance l'information d'un mois
        showLeds(show_date)                                    #change la bande led pour le bon mois

    if localtime(show_date).tm_mon < 10:                       #si le mois est inferieur a 10
        display.show(str(localtime(show_date).tm_mon))         #affiche les paterne par defaut
    elif localtime(show_date).tm_mon == 10:                    #si le mois est egale a 10
        display.show(dix)                                      #affiche le paterne "dix"
    elif localtime(show_date).tm_mon == 11:                    #si le mois est egale a 11
        display.show(onze)                                     #affiche le paterne "onze"
    elif localtime(show_date).tm_mon == 12:                    #si le mois est egale a 12
        display.show(douze)                                    #affiche le paterne "douze"
```

Extensions ou développements possibles

On peut afficher le contenu d'une journée choisie sur un écran LCD
On peut ajouter des données comme la température en ajoutant d'autres capteurs
On peut ajouter des caractéritiques aux évenement comme la durée et faire en sorte qu'à l'entrée d'un événement on vérifie si le nouvel événement ne se chevauche pas avec un autre déjà existant.

Ce dispositif pourrait être intégré dans une application agenda sur smartphone et ordinateur.