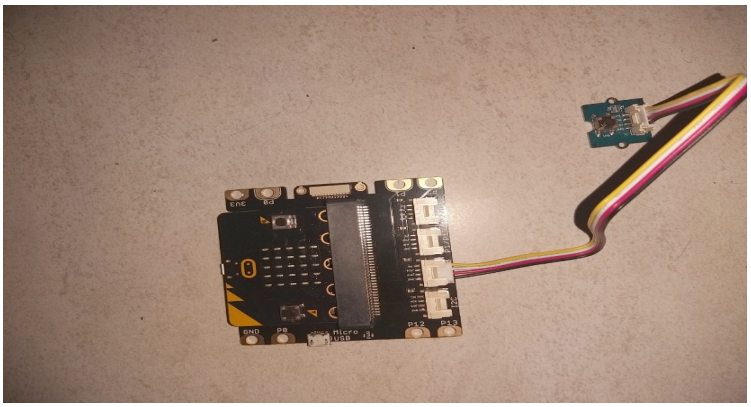
Micro-Tetris

Sénéchal Tom, Alivon Pierre, Dénoue Adrien

Description du projet

Micro-Tetris est un projet où nous avons tenté de reproduire au maximum le fameux jeu Tetris sur carte micro:bit et coder en micro-python. Nous utilisons la matrice de led pour afficher les différents blocs et pour jouer et nous pouvons déplacer les blocs qui tombent sur la droite et la gauche grâce aux boutons de la carte. Un troisième bouton est ici pour permettre de tourner les blocs.



Liste du matériel utilisé

- 1 carte Micro:bit V1.5
- 1 shield Grove pour Micro:bit
- 1 bouton v1.2

Fonctionnement des capteurs et actionneurs utilisés

• La matrice de leds

C'une matrice de 5 x 5 leds, chaque led est considérée comme un pixel dont l'intensité peut varier de 0 à 9 (0 étant éteint, et 9 le plus fort possible). Chaque led possède une position x allant de 0 à 4 et une position y allant de 0 à 4 aussi.

Il y a plusiseurs manières d'afficher ou de modifier l'intensité des leds mais tout ce fait par l'intermédiaire de la fonction display qui dispose de plusieurs utilisations :

display.set_pixel(x, y, val) :
Sert à allumer une led de coordonnées x (entre 0 et 4) et y (entre 0 et 4) avec une intensité val (valeur entre 0 et 9).

	0	1	2	3	4
0	9	7	5	3	1
1	8	6	4	2	0
2	6	4	2	0	0
3	8	6	4	2	0
4	9	7	5	3	1

display.clear()
Sert à éteindre toutes les leds.

display.get_pixel(x, y)

Permet de récuperer (en la renvoyant) l'intensité de la led de coordonnées x et y.

display.show(var,delay=t, ...)

Permet d'afficher une image qui est enregistrée dans une variable var prédéfinie, chaque ensemble de 5 chiffres représentant les 5 leds d'une même ligne et où le chiffre désigne l'intensité de la led (ex: var=Image("00000:09090:00500:70007:07770"). Ensuite t correspond à une durée en millisecondes. On peut encore modifier cela avec par exemple clear=True qui est censé effacer le dessin une fois le temps t écoulé.

• Les boutons de la carte micro:bit

Les boutons sont de simples actionneurs qui créent des événements qui permette de faire réagir Python à certaines conditions ou évènements.
3 méthodes existent pour récupérer une information losqu'on appuie sur un bouton :

button_a.is_pressed()

Renvoie True si le bouton est pressé au moment de l'exécution de l'instruction.

button_a.was_pressed()

Renvoie True si le bouton a été pressé depuis la dernière fois que ça a été utilisé.

button_a.get_presses()

Renvoie le nombre de fois que le bouton a été pressé depuis le dernier appel de l'instruction.

• Le bouton supplémentaire

Ce bouton suppplémentaire que nous allons nommer "c" permet de renvoyer en plus une valeur 1 ou 0 en fonction de si il est pressé ou non.

Pour pouvoir cependant lire cette donnée et détecter le bouton il faut indiquer à python où il se trouve (sur quel pin).
Exemple si le bouton c est sur P0 :
button_c=pin0.read_digital()

Code Python du projet

from microbit import * from random import choice	#import de la librairie microbit #import de la fonction choice de la librairie random
grid=[[1,0,0,0,0,0,1],[1,0,0,0,0,0,1], [1,0,0,0,0,0,1],[1,0,0,0,0,0,1],[1,0,0,0,0,0,1], [1,1,1,1,1,1,1]]	#Configuration de la grille Tetris, pour créer une limite
bricks = [[9,9],[9,0]],[[9,9],[0,9]],[[9,0],[0,0]], [[9,9],[0,0]]	#Création d'une liste de 4 briques; chaque brique est contenu dans une grille 2x2
brick = choice(bricks) x=3 y=0 frameCount=0	#sélectionner une brique au hasard et la positionner au centre/haut de la grille (y=0,x=3)
def max(a,b):	#Pour retourner le maximum de deux valeurs

```
    if a>=b:
        return a
    else:
        return b

def hideBrick():                                    #fonction pour masquer la brique 1x1 sur l'écran LED
    if x>0:
        display.set_pixel(x-1,y,grid[y][x])
    if x<5:
        display.set_pixel(x+1-1,y,grid[y][x+1])
    if x>0 and y<4:
        display.set_pixel(x-1,y+1,grid[y+1][x])
    if x<5 and y<4:
        display.set_pixel(x+1-1,y+1,grid[y+1]
[x+1])

def showBrick():                                    #fonction pour faire montrer la brique 1x1 sur l'écran LED
    if x>0:
        display.set_pixel(x-1,y,max(brick[0]
[0],grid[y][x]))
    if x<5:
        display.set_pixel(x+1-1,y,max(brick[0]
[1],grid[y][x+1]))
    if x>0 and y<4:
        display.set_pixel(x-1,y+1,max(brick[1]
[0],grid[y+1][x]))
    if x<5 and y<4:
       display.set_pixel(x+1-1,y+1,max(brick[1]
[1],grid[y+1][x+1]))

    pixel00 = brick[0][0]                           #Une fonction pour faire pivoter la brique 1x1
    pixel01 = brick[0][1]                           def rotateBrick():
    pixel10 = brick[1][0]
    pixel11 = brick[1][1]
    if not ((grid[y][x]>0 and pixel00>0) or         #Vérification si la rotation est possible
(grid[y+1][x]>0 and pixel10>0) or (grid[y]
[x+1]>0 and pixel01>0) or (grid[y+1][x+1]>0
and pixel11>0)):
        hideBrick()
        brick[0][0] = pixel10
        brick[1][0] = pixel11
        brick[1][1] = pixel01
        brick[0][1] = pixel00
        showBrick()

def moveBrick(delta_x,delta_y):                     #Une fonction pour déplacer la brique
    global x,y
    move=False
    if delta_x==-1 and x>0:
        if not ((grid[y][x-1]>0 and brick[0][0]>0)  #Vérifier si le déplacement est possible : pas de collision avec d'autres blocs ou bordures de la grille
or (grid[y][x+1-1]>0 and brick[0][1]>0) or
(grid[y+1][x-1]>0 and brick[1][0]>0) or
```

```
(grid[y+1][x+1-1]>0 and brick[1][1]>0)):
            move=True
    elif delta_x==1 and x<5:
        if not ((grid[y][x+1]>0 and brick[0][0]>0)
or (grid[y][x+1+1]>0 and brick[0][1]>0) or
(grid[y+1][x+1]>0 and brick[1][0]>0) or
(grid[y+1][x+1+1]>0 and brick[1][1]>0)):
            move=True
    elif delta_y==1 and y<4:
        if not ((grid[y+1][x]>0 and brick[0][0]>0)
or (grid[y+1][x+1]>0 and brick[0][1]>0) or
(grid[y+1+1][x]>0 and brick[1][0]>0) or
(grid[y+1+1][x+1]>0 and brick[1][1]>0)):
            move=True
    if move:                                    #Si le déplacement est possible, mettre à jour les
        hideBrick()                             coordonnées x,y de la brique
        x+=delta_x
        y+=delta_y
        showBrick()

                                                 #Retourner Vrai ou Faux pour confirmer si le
    return move                                 deplacement a eu lieu

def checkLines():                               #Une fonction pour vérifier et supprimer les
    global score                                lignes terminées
    removeLine=False
    for i in range(0, 5):                        #vérifier chaque ligne une à une
        if (grid[i][1]+grid[i][2]+grid[i][3]+grid[i]  #Si 5 blocs sont remplis (9 d'intensité), une
[4]+grid[i][5])==45:                            ligne est terminée (9*5=45)
            removeLine = True
            score+=10                           #Augmenter le score (10 points par ligne)
            for j in range(i,0,-1):             #Retirer la ligne et faire tomber toutes les lignes
                grid[j] = grid[j-1]             au-dessus de 1 :
            grid[0]=[1,0,0,0,0,0,1]
    if removeLine:
        for i in range(0, 5):                   #Actualiser l'écran LED
            for j in range(0, 5):
                display.set_pixel(i,j,grid[j][i+1])
    return removeLine

gameOn=True
score=0
showBrick()

while gameOn:                                   #Boucle de programme principale - répetition
    sleep(50)                                   toutes les 50ms
    frameCount+=1
    button_c=pin0.read_digital() #lecture de
l'entrée de P0

    if button_c==1 :                             #les commande de l'utilisateur
        rotateBrick()
        sleep(100)
```

```
    elif button_a.is_pressed():
        moveBrick(-1,0)
        sleep(50)
    elif button_b.is_pressed():
        moveBrick(1,0)
        sleep(50)

    if frameCount==15 and moveBrick(0,1) ==
False:                                          #Tous les 15 cadres essayer de déplacer la
        frameCount=0                            brique vers le bas
        grid[y][x]=max(brick[0][0],grid[y][x])  #si le déplacement n'est pas possible, la brique
        grid[y][x+1]=max(brick[0][1],grid[y][x+1]) reste en place
        grid[y+1][x]=max(brick[1][0],grid[y+1][x])
        grid[y+1][x+1]=max(brick[1][1],grid[y+1]
[x+1])

        if checkLines()==False and y==0:
            gameOn=False                        #La brique a atteint le haut de la grille - Game
        else:                                   Over
            x=3                                 #sélectionner une nouvelle brique au hasard
            y=0
            brick = choice(bricks)
            showBrick()

    if frameCount==15:
       frameCount=0

                                                #fin de la Game
sleep(2000)
display.scroll("GAME OVER: Score: " +           #affiche le scrore
str(score))
```

Extensions ou développements possibles

On pourrait compléter et améliorer le programme avec de nouvelles insctructions ou autre capteurs / actionneurs :

- Rajouter des blocs ou différentes conditions
- Rajouter de la musique à condition d'avoir la place sur la carte car nus avions pensé à rajouter de la musique avec un haut-parleur mais par faute de place sur la carte nous n'avons pas pû et n'avons pas pû non plus tester si notre programme était bon mais le voilà :

```
from music import*
while True :
        music.playTone(330, music.beat(BeatFraction.Whole))
        music.playTone(247, music.beat(BeatFraction.Half))
        music.playTone(262, music.beat(BeatFraction.Half))
        music.playTone(294, music.beat(BeatFraction.Whole))
        music.playTone(262, music.beat(BeatFraction.Half))
        music.playTone(247, music.beat(BeatFraction.Half))
        music.playTone(220, music.beat(BeatFraction.Whole))
        music.playTone(220, music.beat(BeatFraction.Half))
```

```
        music.playTone(262, music.beat(BeatFraction.Half))
        music.playTone(330, music.beat(BeatFraction.Whole))
        music.playTone(294, music.beat(BeatFraction.Half))
        music.playTone(262, music.beat(BeatFraction.Half))
        music.playTone(247, music.beat(BeatFraction.Whole))
        music.playTone(247, music.beat(BeatFraction.Half))
        music.playTone(262, music.beat(BeatFraction.Half))
        music.playTone(294, music.beat(BeatFraction.Whole))
        music.playTone(330, music.beat(BeatFraction.Whole))
        music.playTone(262, music.beat(BeatFraction.Whole))
        music.playTone(220, music.beat(BeatFraction.Whole))
        music.playTone(220, music.beat(BeatFraction.Whole))
        basic.pause(400)
        music.playTone(294, music.beat(BeatFraction.Whole))
        music.playTone(349, music.beat(BeatFraction.Half))
        music.playTone(440, music.beat(BeatFraction.Half))
        music.playTone(440, music.beat(BeatFraction.Half))
        music.playTone(392, music.beat(BeatFraction.Half))
        music.playTone(349, music.beat(BeatFraction.Half))
        music.playTone(330, music.beat(BeatFraction.Whole))
        music.playTone(262, music.beat(BeatFraction.Whole))
        music.playTone(330, music.beat(BeatFraction.Whole))
        music.playTone(294, music.beat(BeatFraction.Half))
        music.playTone(262, music.beat(BeatFraction.Half))
        music.playTone(247, music.beat(BeatFraction.Whole))
        music.playTone(247, music.beat(BeatFraction.Half))
```

Avec plus de matériel et plus de moyen on pourrait aussi imaginer :

- connecter la carte avec une batterie efficace, pas trop encombrente et recargeable pour pouvoir éviter d'avoir à forcément être brancher pour faire foonctionner
- créer une sorte de socle comme un game-boy pour pouvoir utiliser les différents boutons plus facilement et que ce soit moins encombrant
- créer un grand socle pour pouvoir recréer une sorte de borne d'arcade.
- connecter un écran plus grand à la plce de la matrice de led pour des parties plus longues et intéressantes à condition de rajouter de nouveaux blocs.