Guide CircuitPython.

Documentation officielle Python sur Gamebuino Meta V1.2

Table des matières

Spécifications

Spécifications techniques

Gamebuino META est une console portable fabriquée en France, et qui permet de jouer et de développer soi-même des jeux au look rétro. La Gamebuino META tient dans une poche, et contient de nombreux jeux exclusifs gratuits. La carte microSD intégrée peut contenir des centaines de jeux !

Microcontrôleur : ATSAMD21, 32bit ARM Cortex M0+, 256KB flash, 32KB RAM, identique à l'Arduino Zero

Ecran : 1.8", 80x64 en mode SD, 160x128 pixels en mode HD

Batterie : LiPo de 1000mAh, chargement par le port micro USB B.

Son : 10 bits, lecture multi canal de fichiers WAV 8bit, hautparleur 1W et prise casque.

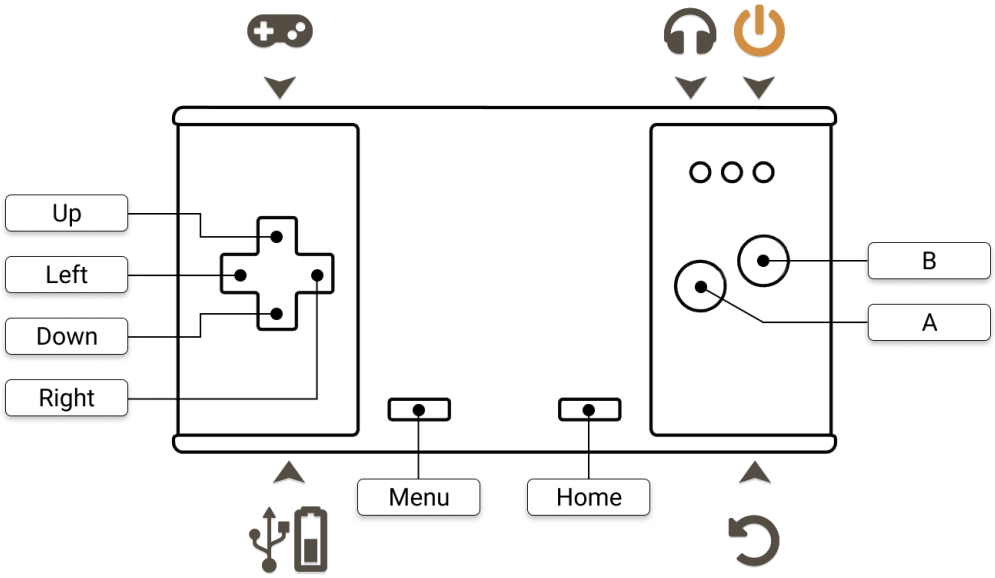
LEDs arrières : 8 LED indépendantes RGB pour les effets de lumière.

Touches : D-pad, A, B, Menu, Home.

Connecteur d'extension : les entrées sorties du microcontrôleur sont accessibles via le connecteur placé à l'arrière.

ATTENTION : la Gamebuino META fonctionne en 3.3V.
Ne pas brancher d'éléments prévus pour une alimentation de 5V.

Comment utiliser votre Gamebuino



- **Interrupteur marche arrêt** : pour mettre en route et arrêter votre Gamebuino.
- **Carte SD** : utiliser le lecteur de carte fourni pour charger plus de jeux !
- **Jack audio** : pour brancher un casque ou un haut-parleur externe.
- **Connecteur USB** : connecteur micro-USB pour charger la console et développer vos propres jeux. La plupart des câbles microUSB du commerce sont compatibles.
- **Reset** : une pression pour redémarrer la console, double pression pour accéder au bootloader.
- **Bouton A** : sélection / Action principale
- **Bouton B** : annuler / Action secondaire
- **Menu** : touche de menu dépendant du jeu.
- **Home** : sortir des jeux, ajuster les réglages, effectuer une capture d'écran.

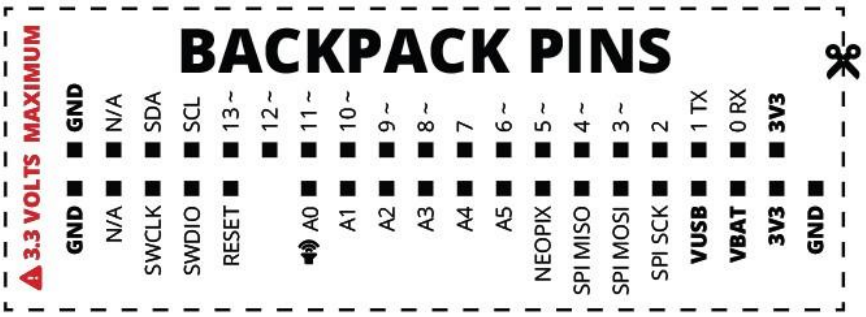
Votre console n'a pas seulement été conçue comme une console de jeu : nous avons pris le temps de transformer ce produit en outil de création. Vous pouvez créer vos propres jeux en C++ ou en Python. La configuration de l'environnement est simple et les possibilités de jeux sont infinies. Faire des jeux sur Gamebuino s'adresse à tous les niveaux, débutants comme experts.

Backpacks

Toutes les broches Arduino sont accessibles au dos de votre Gamebuino.

Vous pouvez ainsi faire de l'électronique avec votre console !

Les broches du connecteur :



Pour commencer.

waitForUpdate()

Pour avoir un affichage fluide, on ne dessine pas directement sur l'écran mais dans la mémoire interne de la console. Ceci permet de ne pas voir les éléments du jeu s'afficher un par un et d'éviter des effets de clignotement si plusieurs éléments graphiques sont superposés. Quand une image est prête à être affichée, il suffit d'appeler la waitForUpdate() pour transférer l'image sur l'écran de la console.

```
from gamebuino_meta import waitForUpdate, display

while True:
    waitForUpdate()
    display.clear()
    display.print("hello")
```

display.clear()

display.clear() efface tout le contenu de l'écran.

```
from gamebuino_meta import waitForUpdate, display

while True:
    waitForUpdate()
    display.clear()
    display.print("hello")
```

display.print (*texte*)

Affiche un texte à l'écran à la position courante du curseur.

NOTE : display.clear() ramène la position courante du curseur en haut à gauche de l'écran.

NOTE : le texte doit être placé entre guillemets.

```
from gamebuino_meta import waitForUpdate, display

while True:
    waitForUpdate()
    display.clear()
    display.print("hello")
```

NOTE : avant d'aller plus loin, essayez d'afficher plusieurs textes pour voir le résultat 😉

Les fonctions d'affichage : « display »

Les fonctions qui commencent par **display** regroupent en ensemble de commandes permettant d'afficher des formes géométriques et du texte.

Pour les utiliser, il est nécessaire d'intégrer le module display avec en utilisant la commande **from** … **import**

```
from gamebuino_meta import display
```

display.clear() et display.clear(*couleur*)

La fonction display.clear() permet d'effacer l'écran, mais on peut aussi préciser un paramètre de couleur pour changer la couleur du fond.

```
from gamebuino_meta import waitForUpdate, display, color

while True:
    waitForUpdate()
    display.clear(color.WHITE)
    display.setColor( color.BLUE )
    display.print("hello")
```

display.setColor(*couleur*)

Sélectionne la valeur de pinceau courante. La couleur du pinceau sera utilisée pour les textes et graphiques.

```
from gamebuino_meta import waitForUpdate, display, color

while True:
    waitForUpdate()
    display.clear(color.WHITE)
    display.setColor( color.BLUE )
    display.print("hello")
```

La palette de couleurs Gamebuino

Nom Python	Couleur
color.WHITE	Blanc
color.GRAY	Gris
color.DARKGRAY	Gris foncé
color.BLACK	Noir
color.PURPLE	Violet
color.PINK	Rose
color.RED	Rouge
color.ORANGE	Orange

Nom Python	Couleur
color.BROWN	Marron
color.BEIGE	Beige
color.YELLOW	Jaune
color.LIGHTGREEN	Vert clair
color.GREEN	Vert
color.DARKBLUE	Bleu marine
color.BLUE	Bleu
color.LIGHTBLUE	Bleu ciel

display.print(*texte*) et display.print(*x, y, texte*)

print() permet d'afficher un texte sur l'écran. On peut préciser les coordonnées (x,y).

```
from gamebuino_meta import waitForUpdate, display, color

while True:
    waitForUpdate()
    display.clear(color.WHITE)
    display.setColor( color.BLUE )
    display.print( 10, 20, "hello")
```

display. setFontSize(*taille*)

Par défaut, la Gamebuino META affiche le texte avec une police de 6 x 8 points. **setFontSize**() permet de changer la taille de la police. Le paramètre taille multiplie la taille du texte par un facteur entier (X1, X2, X5…)

```
from gamebuino_meta import waitForUpdate, display, color

while True:
    waitForUpdate()
    display.clear()
    display.setFontSize(1)
    display.setColor( color.RED )
    display.print( 20, 10, "petit" )
    display.setFontSize(2)
    display.setColor( color.GREEN )
    display.print( 20, 20, "moyen" )
    display.setFontSize(3)
    display.setColor( color.BLUE )
    display.print( 20, 30, "grand" )
```

display.drawPixel (x, y) et display.drawPixel (x, y, *couleur*)

Peint un pixel avec la valeur courante du pinceau, ou la couleur spécifiée « couleur ».

```
while True:
    waitForUpdate()
    display.clear()
    display.setColor( color.BROWN )
    display.drawPixel ( 40, 32 )
    display.drawPixel ( 40, 34, color.BLUE )
```

display.drawLine(x1, y1, x2, y2)

Trace une ligne avec la valeur courante du pinceau entre les deux points aux coordonnées (x1, y1) et (x2, y2)

```
while True:
    waitForUpdate()
    display.clear()
    display.setColor( color.YELLOW )
    display.drawLine ( 40, 0, 40, 63 )
```

display.drawRect (x, y, w, h) et display.fillRect(x, y, w, h)

drawRect() trace le contour d'un rectangle de largeur w et de hauteur h avec le bord en haut à gauche aux coordonnées (x, y).

fillRect() dessine un rectangle plein.

```
from gamebuino_meta import waitForUpdate, display, color

while True:
    waitForUpdate()
    display.clear()
    display.setColor( color.PURPLE )
    display.drawRect( 10, 10, 30, 40 )
    display.setColor( color.LIGHTGREEN)
    display.fillRect( 10, 10, 30, 40 )
```

display.drawRoundRect(x, y, w, h, r) et display.fillRoundRect (x, y, w, h, r)

drawRoundRect() trace le contour d'un rectangle aux bords arrondis de largeur w et de hauteur h avec le bord en haut à gauche aux coordonnées (x, y). Les angles du rectangle ont un bord de rayon r. **fillRoundRect()** dessine d'un rectangle aux bords arrondis plein.

```
from gamebuino_meta import waitForUpdate, display, color

while True:
    waitForUpdate()
    display.clear()
    display.setColor( color.PURPLE )
    display.drawRoundRect( 10, 10, 30, 40, 5 )
    display.setColor( color.LIGHTGREEN)
    display.fillRoundRect( 15, 15, 20, 30, 5 )
```

display.drawCircle(x, y, r) et display.fillCircle(x, y, r)

drawCircle() trace le contour d'un cercle de rayon r avec le centre aux coordonnées (x, y).

fillCircle() dessine un cercle plein.

```
from gamebuino_meta import waitForUpdate, display, color

while True:
    waitForUpdate()
    display.clear()
    display.setColor( color.ORANGE )
    display.drawCircle ( 30, 30, 10 )
    display.setColor( color.GREEN )
    display.fillCircle ( 60, 30, 10 )
```

display.drawTriangle(x1, y1, x2, y2, x3, y3) et display.fillTriangle (x1, y1, x2, y2, x3, y3)

drawTriangle() trace un triangle entre les coordonnées des trois points (x1, y1), (x2, y2) et (x3, y3).

fillTriangle() dessine un triangle plein.

```
from gamebuino_meta import waitForUpdate, display, color

while True:
    waitForUpdate()
    display.clear()
    display.setColor( color.LIGHTGREEN)
    display.fillTriangle( 70, 10, 70, 54, 30, 32)
    display.setColor( color.DARKBLUE)
    display.drawTriangle ( 10, 10, 10, 54, 50, 32)
```

Les fonctions de collision

Le module **collide** propose 4 fonctions pour vous aider dans la gestion des collisions entre les objets : c'est-à-dire déterminer si un objet en touche un autre.

Pour les utiliser, il est nécessaire d'intégrer le module **collide** avec en utilisant la commande **from** … **import**

```
from gamebuino_meta import collide
```

collide.rectRect(x1, y1, w1, h1, x2, y2, w2, h2)
collide.pointRect(pointX, pointY, rectX, rectY, rectW, rectH)
collide.circleCircle(centerX1, centerY1, r1, centerX2, centerY2, r2)
collide.pointCircle(pointX, pointY, centerX, centerY, r)

collide.rectRect() permet de déterminer si il y a collision entre deux rectangles.

collide.pointRect() permet de déterminer si il y a collision entre un point et un rectangle.

collide.circleCircle() permet de déterminer si il y a collision entre deux cercles.

collide.pointCircle() permet de déterminer si il y a collision un point et un cercle.

```
from gamebuino_meta import waitForUpdate, display, buttons, color, collide

pos_x = 40
pos_y = 32
rayon = 5
rect_x = 30
rect_y = 20
rect_w = 20
rect_h = 20

while True:
    waitForUpdate()
    display.clear()
    if buttons.timeHeld( buttons.UP ) and pos_y > 0 :
        pos_y = pos_y - 1
    if buttons.timeHeld( buttons.DOWN ) and pos_y < 63 :
        pos_y = pos_y + 1
    if buttons.timeHeld( buttons.LEFT ) and pos_x > 0 :
        pos_x = pos_x - 1
    if buttons.timeHeld( buttons.RIGHT ) and pos_x < 79 :
        pos_x = pos_x + 1

    display.drawRect( rect_x, rect_y, rect_w, rect_h )
    display.fillCircle( pos_x, pos_y, rayon )
    if collide.pointRect( pos_x, pos_y, rect_x, rect_y, rect_w, rect_h ) :
        display.print( "COLLISION");
```

Les boutons de la console

Le module buttons permet de connaitre l'état des boutons.

Pour les utiliser, il est nécessaire d'intégrer le module **buttons** en utilisant la commande **from** … **import**

```
from gamebuino_meta import buttons
```

buttons.DOWN	D-Pad BAS	buttons.A	Touche action A
buttons.LEFT	D-Pad GAUCHE	buttons.B	Touche action B
gb.buttons.RIGHT	D-Pad DROITE	buttons.MENU	Touche menu MENU
buttons.UP	D-Pad HAUT	buttons.HOME	Touche menu HOME

buttons.pressed(*bouton*) et buttons.released(*bouton*)

Les fonctions **pressed()** et **released()** permettent de savoir si une touche viens d'être appuyée ou relâchée.

Ces fonctions retournent **vrai** dans ce cas.

NOTE : si l'utilisateur presse une touche, **pressed()** ne retourne vrai qu'une fois.

```
from gamebuino_meta import waitForUpdate, display, buttons
etat = "rien"

while True:
    waitForUpdate()
    display.clear()
    if buttons.pressed( buttons.DOWN ) :
        etat = "appui"
        display.print( "DOWN" )
    if buttons.released( buttons.DOWN ) :
        etat = "relache"
        display.print( "NO DOWN" )

    display.print(0, 5, "La touche BAS est")
    display.print(0, 10, etat )
```

buttons.timeHeld(*bouton*)

La fonctions **timeHeld()** permet de savoir depuis combien de temps une touche est appuyée.

Si la touche n'est pas appuyée, **timeHeld()** retourne la valeur 0.

NOTE : le temps est indiqué en nombre d'images.

```
from gamebuino_meta import waitForUpdate, display, buttons

while True:
    waitForUpdate()
    display.clear()
    duree = buttons.timeHeld( buttons.A )

    display.print("La touche A est \n")
    display.print("appuyee depuis\n")
    display.print( duree )
    display.print(" images ")
```

buttons.held(bouton, durée) / buttons.repeat(bouton, durée)

La fonction **held()** retourne **true** une fois que la touche est restée appuyée un certain nombre d'images : déclenchez vos actions à retardement...

La fonction **repeat()** retourne **true** périodiquement tant que la touche reste appuyée : c'est la fonction idéale pour gérer un tir de missile à répétition !

```
from gamebuino_meta import waitForUpdate, display, buttons, color

x = 0;
while True:
    waitForUpdate()
    display.clear()
    display.print(0, 0,  "Pressez A ou B" )

    display.drawLine( 0, 32, 79, 32 )
    if buttons.repeat( buttons.A, 8 ) or buttons.held( buttons.B, 25 ):
        display.drawCircle( x, 32-10, 5 )
    else :
        display.drawCircle( x, 32-5, 5 )

    x = x + 1
    if x > 80 :
        x = 0
```

Les LEDs

Votre Gamebuino comporte 8 LEDS multicolores à l'arrière : ces LEDS sont utilisées come un mini affichage de 2 pixels par 8.

Les fonctions qui commencent par **lights** regroupent en ensemble de commandes permettant de modifier ces 8 LEDs.

Pour les utiliser, il est nécessaire d'intégrer le module **lights** avec en utilisant la commande **from** … **import**

```
from gamebuino_meta import lights
```

lights.clear() et lights.fill(couleur)

clear() éteint toutes les LEDs.

fill() allume toutes les LEDs de la couleur désirée. (voir les codes couleurs Gamebuino)

```
from gamebuino_meta import waitForUpdate, display, buttons, lights, color

while True:
    waitForUpdate()
    display.clear()
    if buttons.timeHeld( buttons.A ) :
        lights.fill( color.RED )
        display.print( "Touche A" )
    else :
        if buttons.timeHeld( buttons.B ) :
            lights.fill( color.GREEN )
            display.print( "Touche B" )
        else :
            lights.clear()
```

lights.drawPixel(x, y, couleur)
lights.setColor(couleur)
lights.fillRect(x, y, w, h)

drawPixel() permet de choisir individuellement la couleur d'une des LEDs.

setColor() sélectionne la couleur de pinceau active.

fillRect() alllume avec la couleur courant une zone de LEDS : rappel, les 8 LEDS sont similaires à un petit écran de 2 pixels de large sur 8 de hauteur.

```
from gamebuino_meta import waitForUpdate, display, buttons, lights, color

while True:
    waitForUpdate()
    display.clear()
    if buttons.timeHeld( buttons.UP ) :
        lights.drawPixel( 0, 0, color.RED )
        lights.drawPixel( 1, 0, color.RED )

    if buttons.timeHeld( buttons.DOWN ) :
        lights.drawPixel( 0, 3, color.RED )
        lights.drawPixel( 1, 3, color.RED )

    if buttons.timeHeld( buttons.LEFT ) :
        lights.setColor( color.RED )
        lights.fillRect( 0, 0, 1, 4 )

    if buttons.timeHeld( buttons.RIGHT ) :
        lights.setColor( color.RED )
        lights.fillRect( 1, 0, 1, 4 )

    if buttons.timeHeld( buttons.A ) :
        lights.clear();
```

L'interface GUI

gui.keyboard(texte) et gui.keyboard(texte, texte par défaut)

gui.keyboard() affiche un clavier virtuel sur l'écran pour permettre à l'utilisateur de saisir un texte. Un texte par défaut peut être précisé.

```
from gamebuino_meta import waitForUpdate, display, gui

texte = gui.keyboard( "votre texte", "rien" )

while True :
    waitForUpdate()
    display.clear()
    display.print( 20, 32, "votre texte : " )
    display.print( 20, 40, texte )
```

gui.menu (*texte*, [*valeur1*, *valeur2*])

gui.menu() permet d'afficher un menu de choix à l'utilisateur. Le second argument est un tableau de chaines.

```
from gamebuino_meta import waitForUpdate, display, gui

choix = gui.menu( "fruit :", ["pomme", "poire", "banane"] )

while True :
    waitForUpdate()
    display.clear()
    display.print( 20, 32, "votre choix : " )
    display.print( 20, 40, choix )
```

gui.popup(*texte* , *images*)

Affiche le message « texte » pendant un nombre d'images.

NOTE : la Gamebuino META affiche 25 images par secondes. Une durée de 50 correspond à 2 secondes.

```
from gamebuino_meta import waitForUpdate, display, gui


while True:
    waitForUpdate()
    display.clear()
    gui.popup( "hello", 30 )
```

Fonctions avancées

setFrameRate(*fréquence de rafraichissement*)

Par défaut, la Gamebuino META affiche 25 images par secondes. La fonction **setFrameRate()** permet d'augmenter ou de diminuer cette fréquence de rafraichissement.

```
from gamebuino_meta import waitForUpdate, display, setFrameRate

nombre = 0
setFrameRate(10)

while True:
    waitForUpdate()
    display.clear()
    display.print(nombre)
    nombre = nombre+1
```

Afficher des sprites

```
from gamebuino_meta import  waitForUpdate, display, color

buffer_bitmap = ( b"\x08\x08"                                  # taille : 8 x 8 points
                  b"\x3C\x42\x81\x81\xa5\x99\x42\x3c" )     # bitmap : 8 octets de 8 points

SCALE = 2
dx = 1
dy = 1
px = 40
py = 32
while True:
    waitForUpdate()
    display.clear()
    display.setColor( color.GREEN )
    display.drawBitmap( px, py, buffer_bitmap, SCALE )
    px = px + dx
    py = py + dy
    if px > 80-8*SCALE or px < 0 :
        dx = -dx
    if py > 64-8*SCALE or py < 0 :
        dy = -dy
```

Jouer avec les couleurs des LEDs

Essayez ce petit exemple pour profiter des belles couleurs des LEDS de la Gamebuino META.

```
from gamebuino_meta import waitForUpdate, display, color, lights
from random import randrange

color_val = [0, 0, 0]
color_idx = 0
color_dir = 1

while True :
    waitForUpdate()
    display.clear()
    color_val[color_idx] = color_val[color_idx] + color_dir
    if (color_dir>0 and color_val[color_idx]==31) or (color_dir<0 and color_val[color_idx]==0) :
        color_idx = randrange(3)
        if color_val[color_idx] > 0 :
            color_dir = -1
        else :
            color_dir = +1
    lights.fill( (color_val[0]<<0) + (color_val[1]<<6) + (color_val[2]<<11)  )
    display.setColor( color.BLUE )
    display.print( 20, 32, color_val[0] )
    display.setColor( color.GREEN )
    display.print( 40, 32, color_val[1] )
    display.setColor( color.RED )
    display.print( 60, 32, color_val[2] )
```