CircuitPython Guide.

Official Gamebuino Meta Python documentation V1.3

Table des matières

Specifications

Technical specifications

Gamebuino META is a portable console made in France, which allows you to play and develop retro games yourself. The Gamebuino META fits in your pocket, and contains many exclusive free games. The built-in microSD card can hold hundreds of games !

Microcontroller : ATSAMD21, 32bit ARM Cortex M0+, 256KB flash, 32KB RAM, similar to Arduino Zero

Screen : 1.8", 80x64 in SD mode, up to 160x128 in HD mode

Battery : 1000mAh LiPo, charging via the micro USB B port.

Sound: 10-bit, multi-channel playback of 8-bit WAV files, 1W speaker and headphone jack.

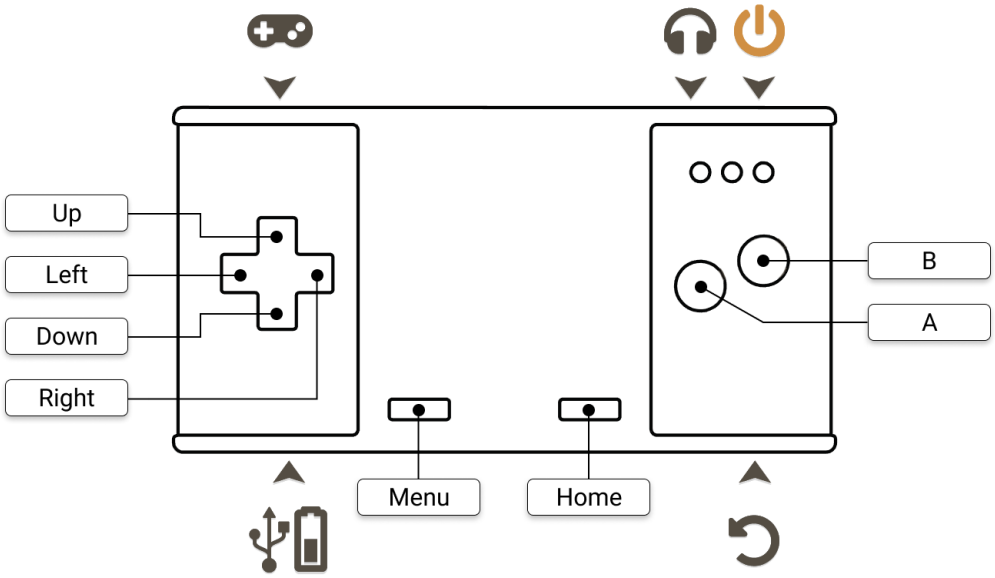
Back LEDs : 8 independent RGB LEDs for lighting effects.

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

Extension connector : the inputs and outputs of the microcontroller are accessible via the connector placed at the back.

WARNING: the Gamebuino META operates at 3.3V.
Do not connect elements designed for a 5V supply.

How to use your Gamebuino ?



- **On-off switch** : to start and stop your Gamebuino.
- **SD Card** : use the provided card reader to load more games !
- **Audio Jack** : to connect headphones or an external speaker.
- **USB Connector** : micro-USB connector to charge the console and develop your own games. Most commercial microUSB cables are compatible.
- **Reset** : one press to restart the console,
  double press to access the bootloader.
- **A button** : selection / Main Action.
- **B button** : cancel / secondary action.
- **Menu** : game-dependent menu key.
- **Home** : exit games, adjust settings, take a screenshot.

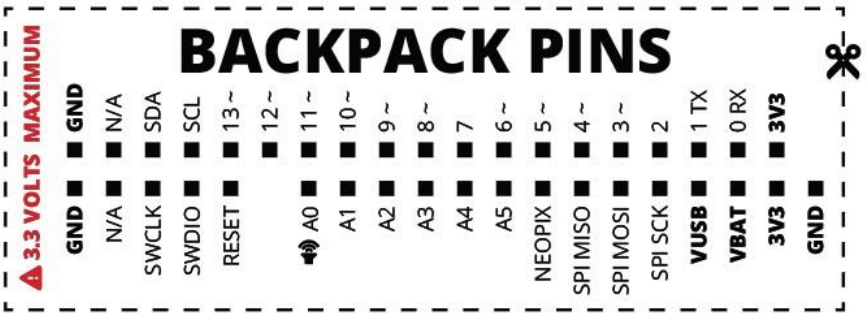
Your console wasn't just designed as a game console : we took the time to turn this product into a creative tool. You can create your own games in C++ or Python. Setting up the environment is simple and the play possibilities are endless. Making games on Gamebuino is for all levels, beginners to experts.

Backpacks

All Arduino pins are accessible on the back of your Gamebuino.

So you can do electronics with your console!

Connector pinout :



To start.

waitForUpdate()

To have a fluid display, we do not draw directly on the screen but in the internal memory of the console. This prevents game elements from being displayed one by one and avoids flashing effects if several graphic elements are superimposed. When an image is ready to be displayed, just call the waitForUpdate() to transfer the image to the console screen.

```
from gamebuino_meta import waitForUpdate, display

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

display.clear()

display.clear() erase all content from the screen.

```
from gamebuino_meta import waitForUpdate, display

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

display.print (*text*)

Displays text on the screen at the current cursor position.

NOTE : display.clear() returns the current cursor position to the top left of the screen.

NOTE : the text must be enclosed in quotes.

```
from gamebuino_meta import waitForUpdate, display

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

NOTE : before going further, try to display several texts to see the result 😉

Display functions: « display »

Functions that begin with **display** group together a set of commands for displaying geometric shapes and text.

To use them, it is necessary to integrate the display module with using the command **from** … **import**

```
from gamebuino_meta import display
```

display.clear() and display.clear(*color*)

The display.clear() function clears the screen, but you can also specify a color parameter to change the background color.

```
from gamebuino_meta import waitForUpdate, display, color

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

display.setColor(*color*)

Selects the current brush value. The brush color will be used for text and graphics.

```
from gamebuino_meta import waitForUpdate, display, color

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

The Gamebuino color palette

Python color name	Color
color.WHITE	White
color.GRAY	Gray
color.DARKGRAY	Dark gray
color.BLACK	Black
color.PURPLE	Purple
color.PINK	Pink
color.RED	Red
color.ORANGE	Orange

Python color name	Color
color.BROWN	Grown
color.BEIGE	Beige
color.YELLOW	Yellow
color.LIGHTGREEN	Light green
color.GREEN	Green
color.DARKBLUE	Dark blue
color.BLUE	Blue
color.LIGHTBLUE	Light blue

display.print(*text*) and display.print(*x, y, text*)

print() displays text on the screen. You can specify the (x,y) coordinates.

```
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(*text size*)

By default, the Gamebuino META displays text in a 6 x 8 point font. **setFontSize()** allows to change the font size. The size parameter multiplies the size of the text by an integer factor (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) and display.drawPixel (x, y, *color*)

Draw a pixel with the current brush value, or the specified color « color ».

```
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)

Draws a line with the current brush value between the two points at coordinates (x1, y1) and (x2, y2).

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

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

drawRect() draws the outline of a rectangle of width w and height h with the upper left edge at coordinates (x, y).

fillRect() draws a filled rectangle.

```
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) and display.fillRoundRect (x, y, w, h, r)

drawRoundRect() draws the outline of a rectangle with rounded edges of width w and height h with the upper left edge at coordinates (x, y). The corners of the rectangle have an edge of radius r.

fillRoundRect() draws a filled rounded rectangle.

```
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) and display.fillCircle(x, y, r)

drawCircle() draws the outline of a circle of radius r with the center at coordinates (x,y).

fillCircle() draws a filled circle.

```
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) and display.fillTriangle (x1, y1, x2, y2, x3, y3)

drawTriangle() draws a triangle between the coordinates of the three points (x1, y1), (x2, y2) and (x3, y3).

fillTriangle() draws a filled triangle.

```
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)
```

Collision functions

The **collide** module offers 4 functions to help you manage collisions between objects: i.e. determine if an object touches another.

To use them, it is necessary to integrate the **collide** module with using the command **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() determines if there is a collision between two rectangles.

collide.pointRect() determines if there is a collision between a point and a rectangle.

collide.circleCircle() determines if there is a collision between two circles.

collide.pointCircle() determines if there is a collision between a point and a circle.

```
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");
```

Console buttons

The buttons module allows to know the state of the buttons.

To use them, it is necessary to integrate the **buttons** module with using the command **from** … **import**

```
from gamebuino_meta import buttons
```

buttons.DOWN	D-Pad DOWN	buttons.A	A action key
buttons.LEFT	D-Pad LEFT	buttons.B	B action key
gb.buttons.RIGHT	D-Pad RIGHT	buttons.MENU	MENU key
buttons.UP	D-Pad UP	buttons.HOME	HOME key

buttons.pressed(*button*) and buttons.released(*button*)

The **pressed()** and **released()** functions let you know if a key has just been pressed or released.

These functions return **true** in this case.

NOTE : if the user presses a key, **pressed()** returns true only once.

```
from gamebuino_meta import waitForUpdate, display, buttons
buttonstatus = "none"

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

    display.print(0, 5, "The DOWN key is ")
    display.print(0, 10, buttonstatus)
```

buttons.timeHeld(*button*)

The **timeHeld()** function allows you to know how long a key has been pressed.

If the key is not pressed, **timeHeld()** returns the value 0.

NOTE : time is returned in number of frames.

```
from gamebuino_meta import waitForUpdate, display, buttons

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

    display.print("The A key is \n")
    display.print("press since \n")
    display.print( duration )
    display.print(" frames ")
```

buttons.held(button, duration) / buttons.repeat(button, duration)

The **held()** function returns true once the key has been pressed for a certain number of images: trigger your delayed actions...

The **repeat()** function returns true periodically as long as the key is pressed: it is the ideal function to manage a repeated missile launch!

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

x = 0;
while True:
    waitForUpdate()
    display.clear()
    display.print(0, 0,  "Press A or 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
```

Rear LEDs

Your Gamebuino has 8 multicolored LEDs on the back: these LEDs are used as a mini display of 2 pixels by 4.

The functions that start with **lights** group together a set of commands to modify these 8 LEDs.

To use them, it is necessary to integrate the **lights** module with using the command **from** … **import**

```
from gamebuino_meta import lights
```

lights.clear() and lights.fill(color)

clear() turn off all LEDs.

fill() lights up all the LEDs of the desired color. (see Gamebuino color codes)

```
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( "Key : A" )
    else :
        if buttons.timeHeld( buttons.B ) :
            lights.fill( color.GREEN )
            display.print( "Key : B" )
        else :
            lights.clear()
```

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

drawPixel() allows you to individually choose the color of one of the LEDs.

setColor() selects the active brush color.

fillRect() lights up with the current color an area of LEDS: reminder, the 8 LEDS are similar to a small screen 2 pixels wide by 8 high.

```
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();
```

The GUI interface

gui.keyboard(text) and gui.keyboard(text, default text)

gui.keyboard() displays a virtual keyboard on the screen to allow the user to enter text. A default text can be specified.

```
from gamebuino_meta import waitForUpdate, display, gui

usrtext = gui.keyboard( "your text", "nope" )

while True :
    waitForUpdate()
    display.clear()
    display.print( 20, 32, "your test is : " )
    display.print( 20, 40, usrtext )
```

gui.menu (*text*, [*value1*, *value2*])

gui.menu() Displays a menu of choices to the user. The second argument is an array of strings.

```
from gamebuino_meta import waitForUpdate, display, gui

userchoice = gui.menu( "fruit :", ["apple", "Pear", "banana"] )

while True :
    waitForUpdate()
    display.clear()
    display.print( 20, 32, "your choise : " )
    display.print( 20, 40, userchoice)
```

gui.popup(*text* , framecount)

Displays the "text" message for a number of frames.

NOTE : the Gamebuino META displays 25 frames per second. A duration of 50 corresponds to 2 seconds.

```
from gamebuino_meta import waitForUpdate, display, gui


while True:
    waitForUpdate()
    display.clear()
    gui.popup( "hello", 50 )
```

Advanced functions

setFrameRate(*refresh rate*)

By default, the Gamebuino META displays 25 frames per second. The **setFrameRate()** function allows you to increase or decrease this refresh rate.

```
from gamebuino_meta import waitForUpdate, display, setFrameRate

nombre = 0
setFrameRate(10)

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

Draw sprites

```
from gamebuino_meta import  waitForUpdate, display, color

buffer_bitmap = ( b"\x08\x08"                                  # size : 8 x 8 points
                  b"\x3C\x42\x81\x81\xa5\x99\x42\x3c" )     # bitmap : 8 bytes of 8 dots

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
```

Play with LED colors

Try this small example to enjoy the beautiful LED colors of the 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] )
```