Create interactive games using electronics and MicroPython

with BBC micro:bit

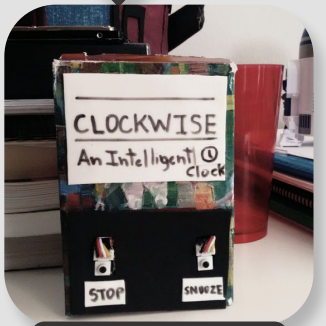
Juliana Karoline de Sousa (@julianaklulo)

Juliana Karoline de Sousa

@julianaklulo

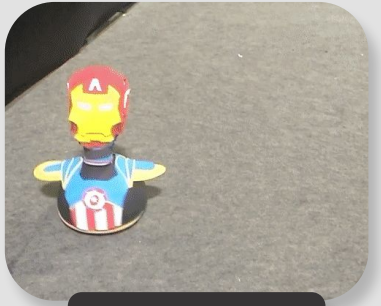
- Bachelor in Computer Science (**UFSCar**)
- **PyLadies São Carlos** | co-founder & organizer
- **grupy-sanca** and **sancaLUG** | co-founder & organizer
- **Omnivector** | software engineer
- **IoT, robotics and kitties enthusiast**

DEVELOPMENT BOARDS I`VE USED



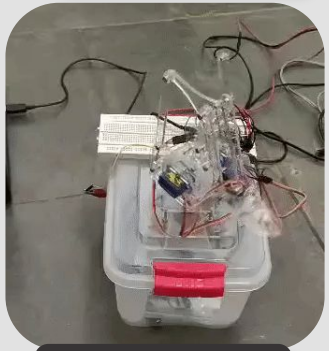
INTEL EDISON

INTEL GALILEO

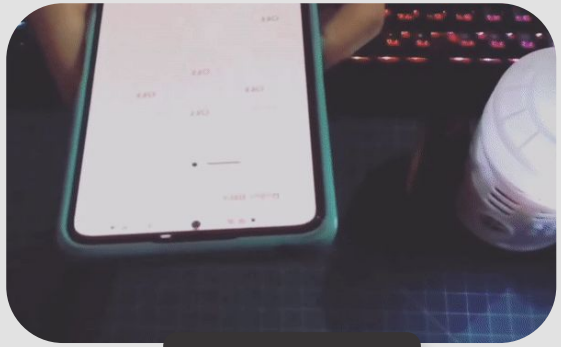


ARDUINO MEGA

ARDUINO UNO

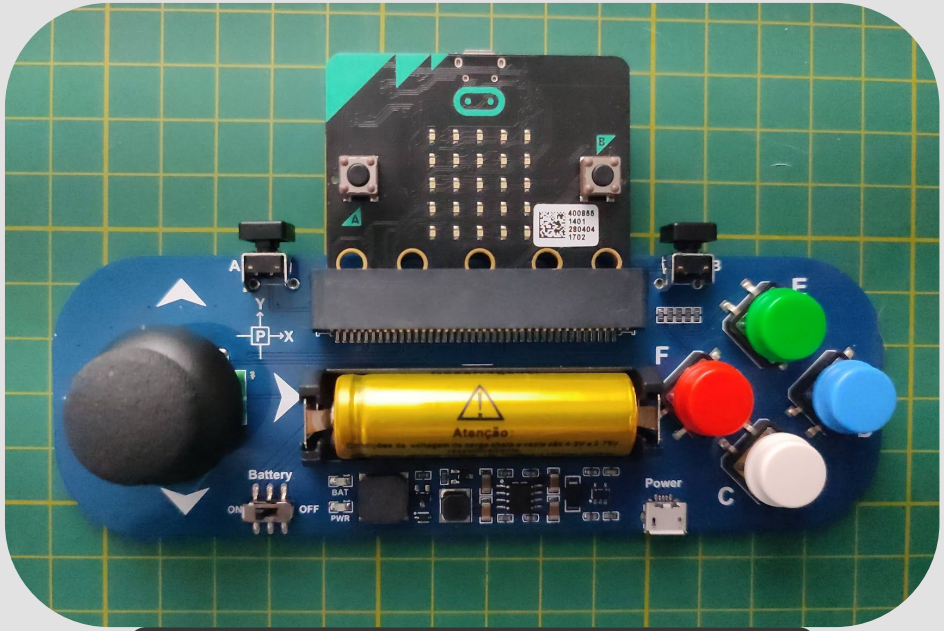


NODE MCU



RASP PI PICO

WHAT I WILL USE FOR THIS PRESENTATION



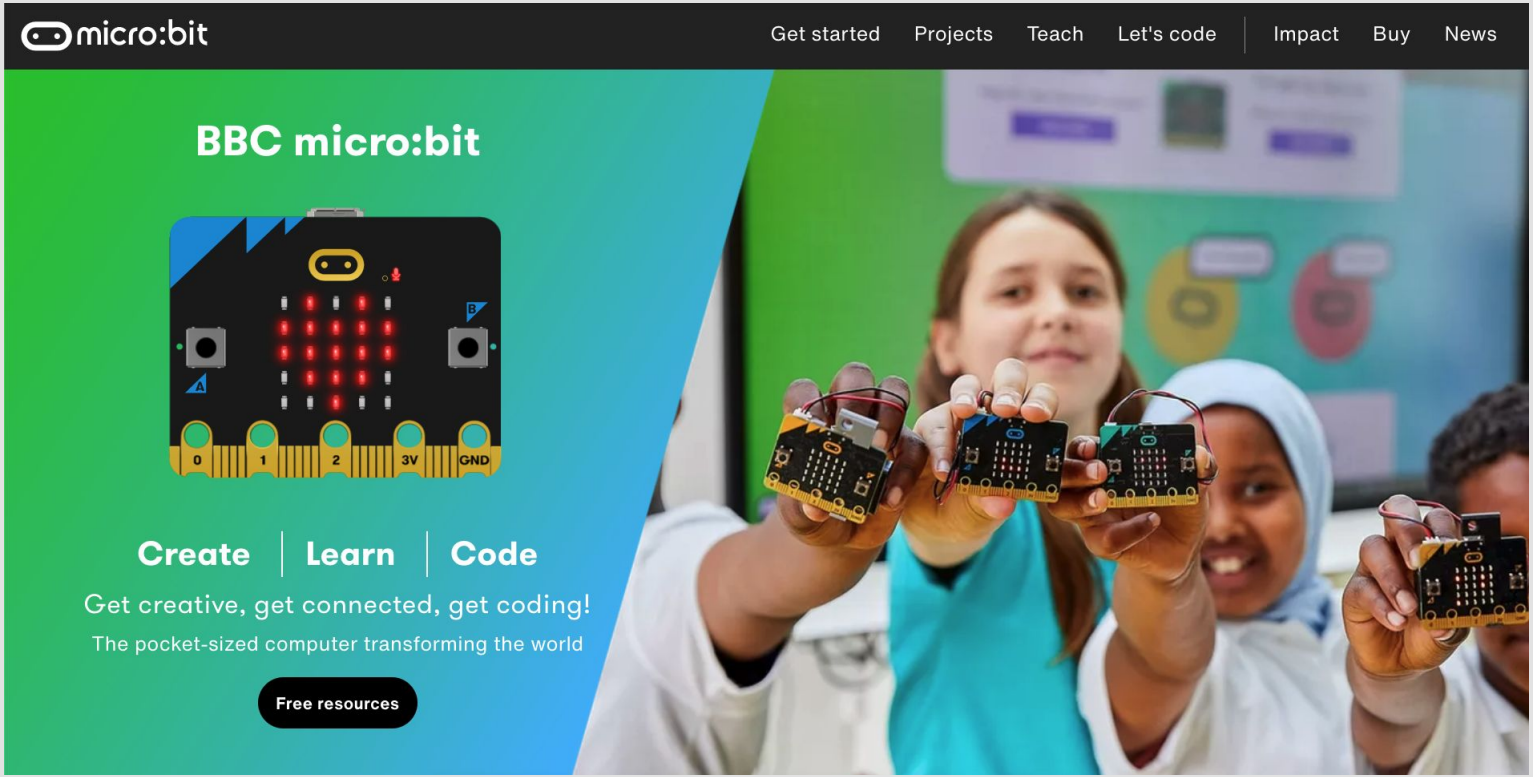
BBC MICRO:BIT + GAMEPAD

AGENDA

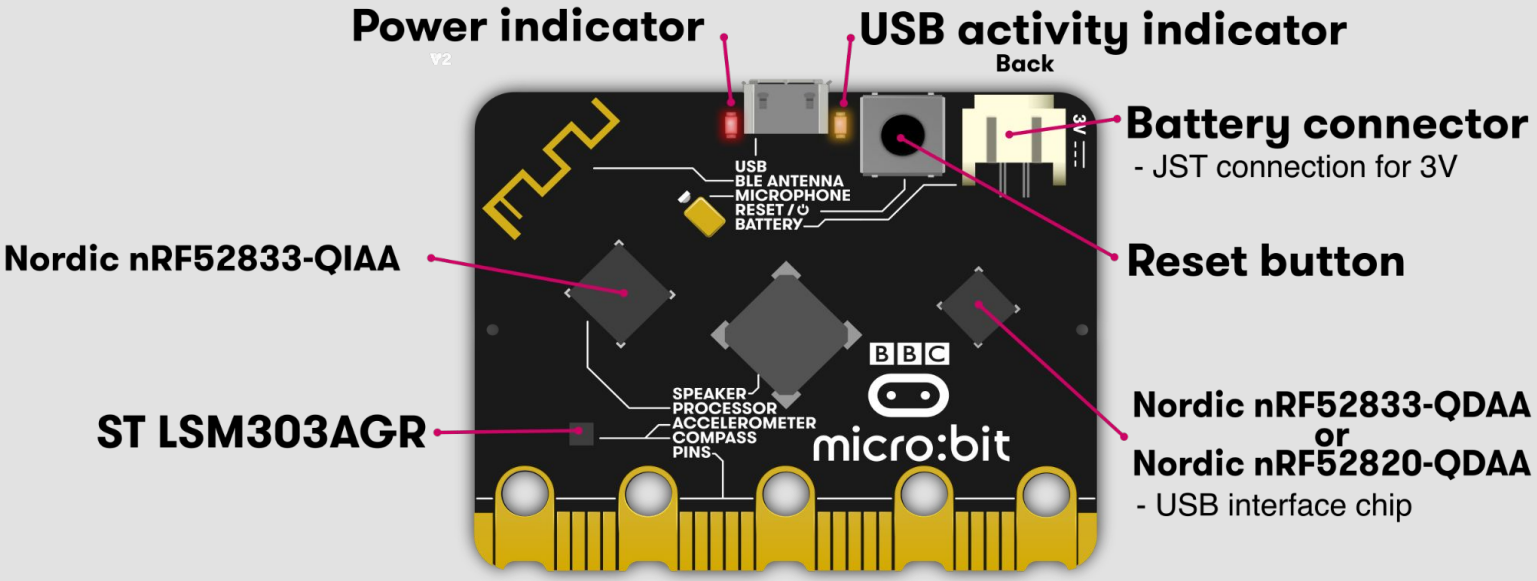
- BBC micro:bit project overview
- Gamepad overview
- GPIO pins, digital and analog IO
- Programming the micro:bit
- Code examples of interactive games:
  - Genius
  - Chase the Dot
  - Car Crash
- QA

ABOUT THE BBC MICRO:BIT PROJECT

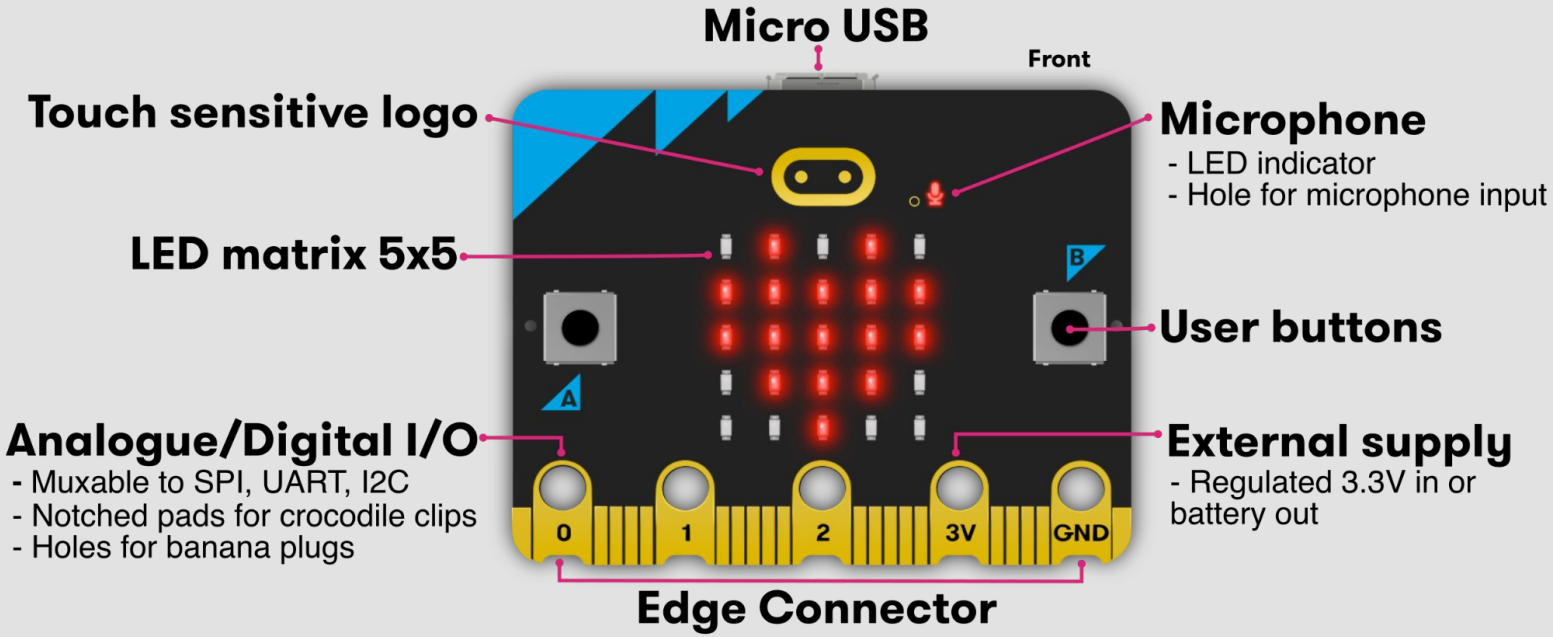
BBC MICRO:BIT PROJECT OVERVIEW



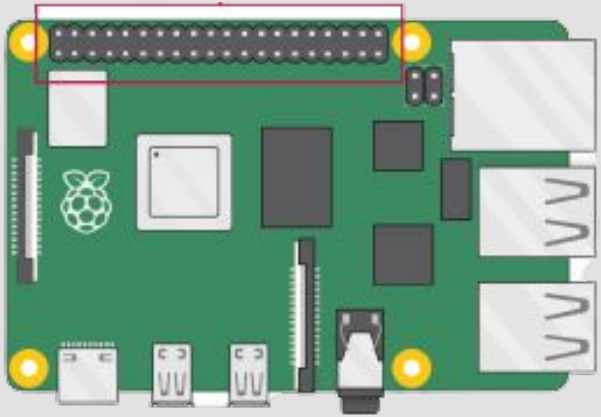
BBC MICRO:BIT V2 SPECIFICATIONS

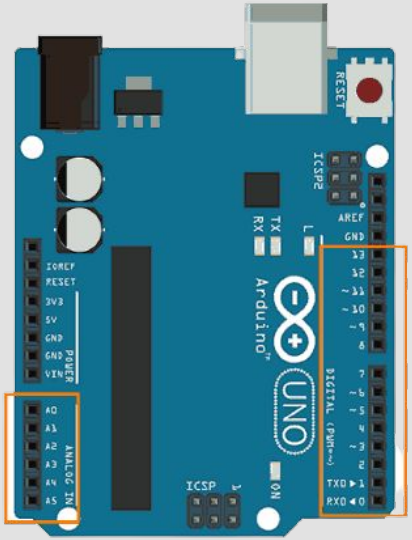


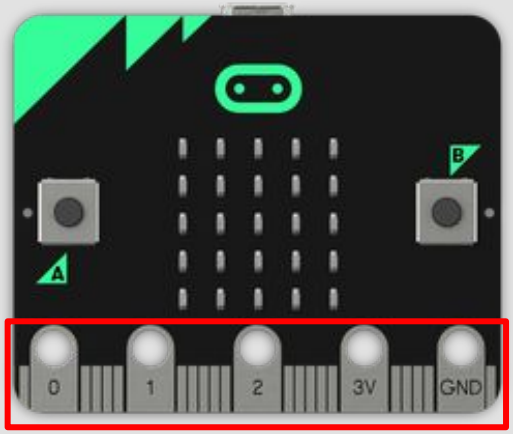
BBC MICRO:BIT V2 SPECIFICATIONS



GP10: GENERAL PURPOSE INPUT/OUTPUT







ANALOG AND DIGITAL SIGNALS

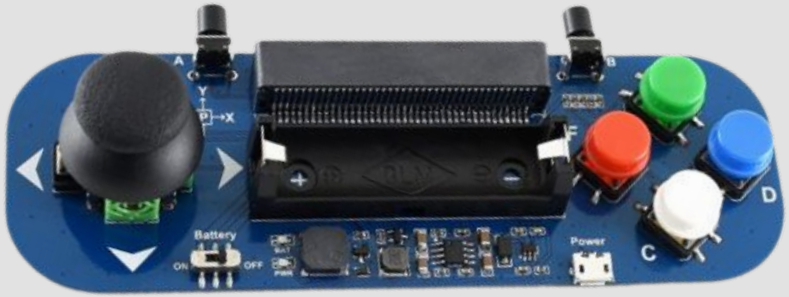
Digital Signal

Analog Signal

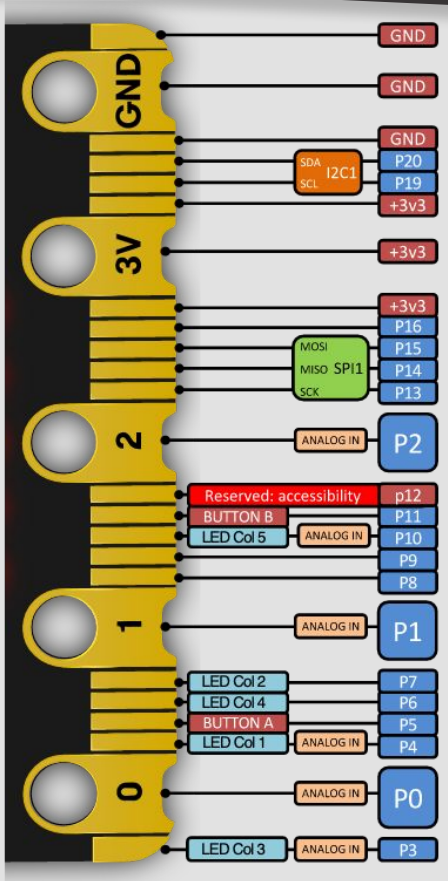
ABOUT THE GAMEPAD FOR BBC MICRO:BIT

GAMEPAD FOR MICRO:BIT SPECIFICATIONS

FEATURES
BBC micro:bit edge connector
Dual channel joystick
Onboard battery and charger circuit
Battery indicator
4 Push buttons
2 Side buttons
1 Buzzer

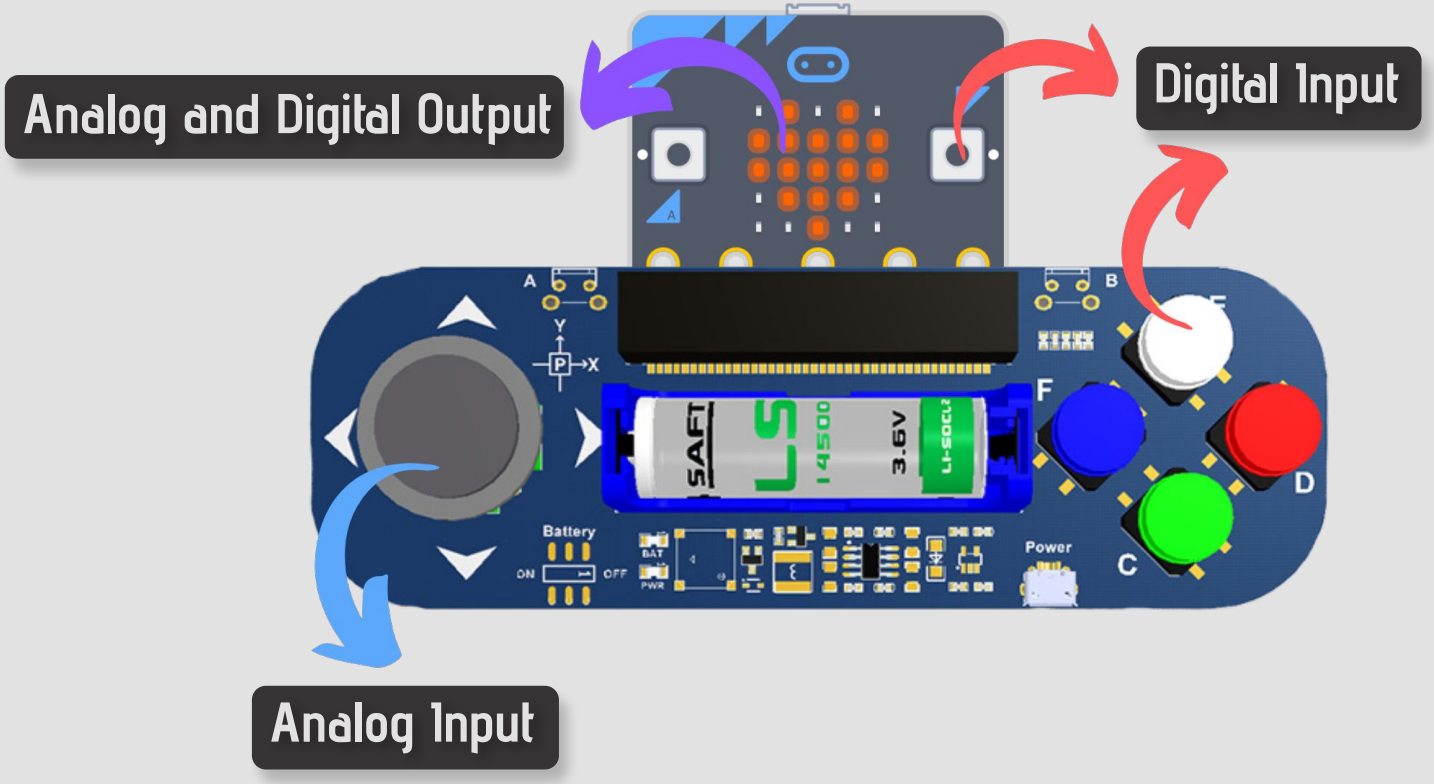


GAMEPAD PINOUT



SYMBOL	PIN	DESCRIPTION
A	5	Button A
B	11	Button B
C	15	Button C
D	14	Button D
E	13	Button E
F	12	Button F
X	1	Joystick X axis
Y	2	Joystick Y axis
P	8	Joystick press
Buzzer	0	Buzzer

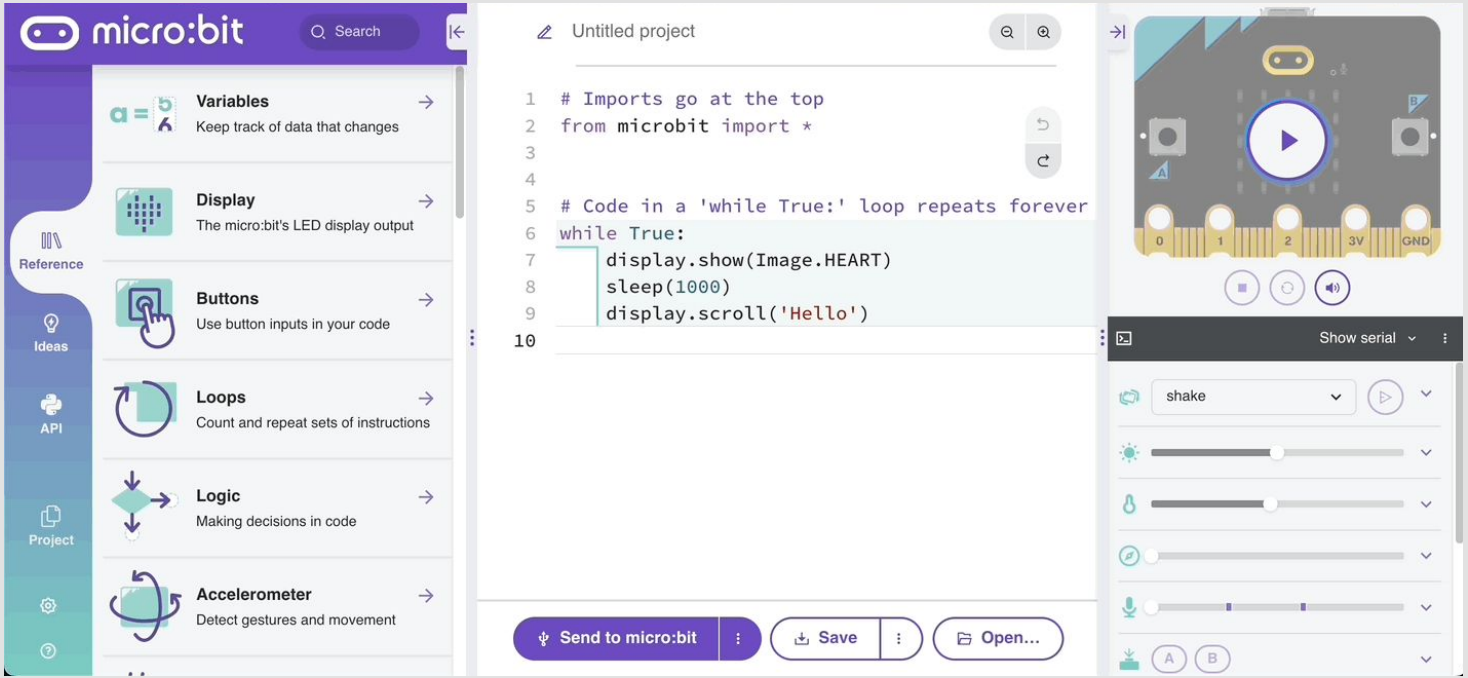
DIGITAL AND ANALOG INPUT/OUTPUT



PROGRAMMING THE MICRO:BIT

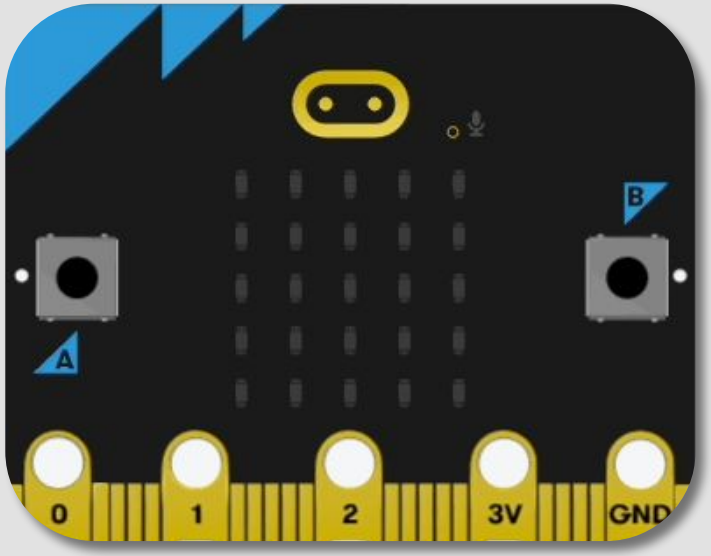
PROGRAMMING THE MICRO:BIT

MICRO:BIT PYTHON TEXT EDITOR



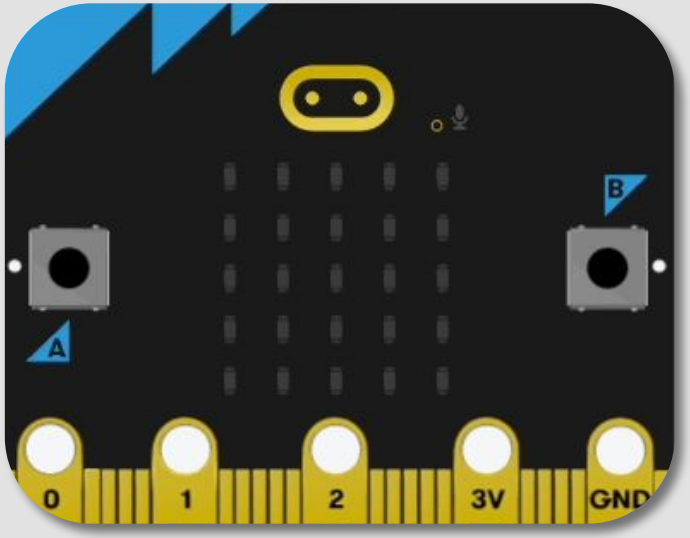
DISPLAY EXAMPLE

```
from microbit import *

display.show(Image.HAPPY)
display.set_pixel(2, 2, 7)
display.show(Image(
    "99099:"
    "90509:"
    "05550:"
    "90509:"
    "99099"
))
display.scroll("PyCon US")
display.clear()
```



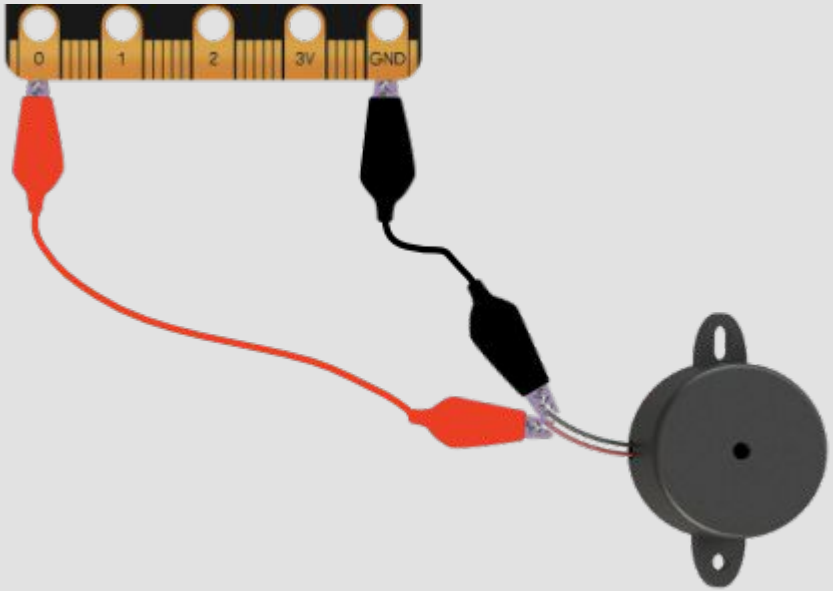
BUTTON EXAMPLE

```
from microbit import *

while True:
    if button_a.is_pressed():
        display.show(Image.ARROW_W)
    if button_b.is_pressed():
        display.show(Image.ARROW_E)
    sleep(100)
    display.clear()
```



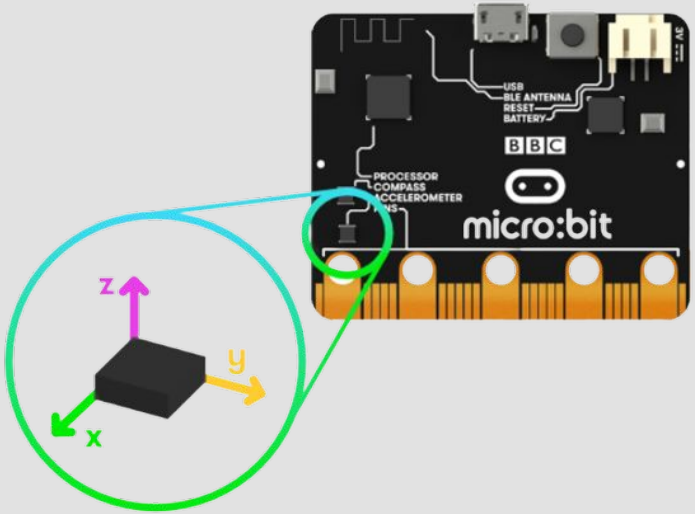
BUZZER EXAMPLE

```
import music

music.play(music.POWER_UP)
music.pitch(350, 100)

music.play([
    "C4:4", "D4:4", "E4:4",
    "C4:4", "C4:4", "D4:4",
    "E4:4", "C4:4", "E4:4",
    "F4:4", "G4:8", "E4:4",
    "F4:4", "G4:8"
])
```



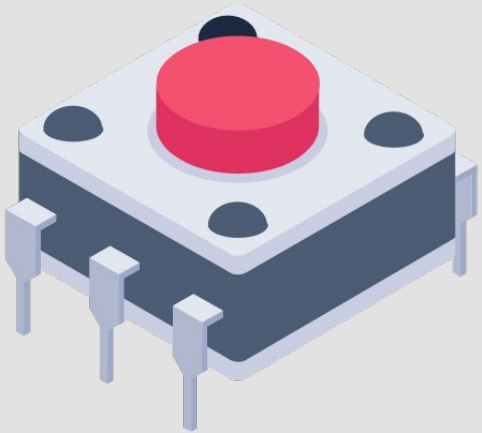
ACCELEROMETER EXAMPLE

```
from microbit import *

x = accelerometer.get_x()
y = accelerometer.get_y()
z = accelerometer.get_z()

accelerometer.was_gesture("shake")
accelerometer.is_gesture("face up")
```



GP10 EXAMPLE

```
"""
Digital read the pin15.
If the button is pressed:
    read_digital returns 0
If the button is not pressed:
    read_digital returns 1
"""
from microbit import *

button = pin15
is_pressed = button.read_digital() == 0
```

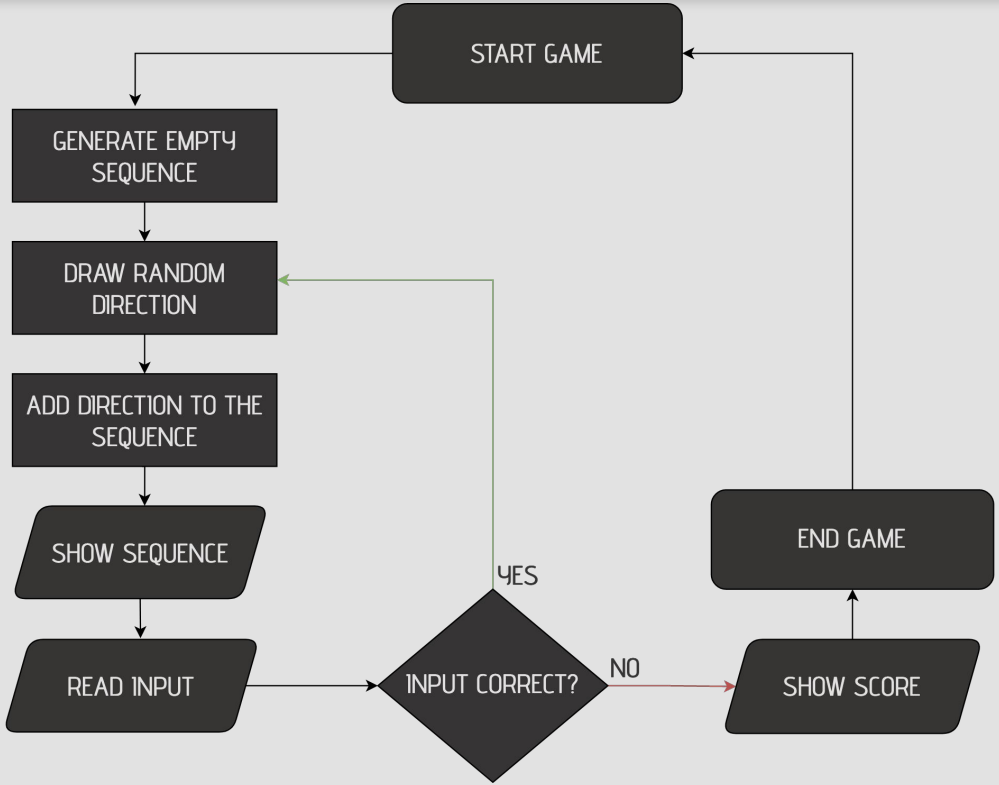


JOYSTICK EXAMPLE

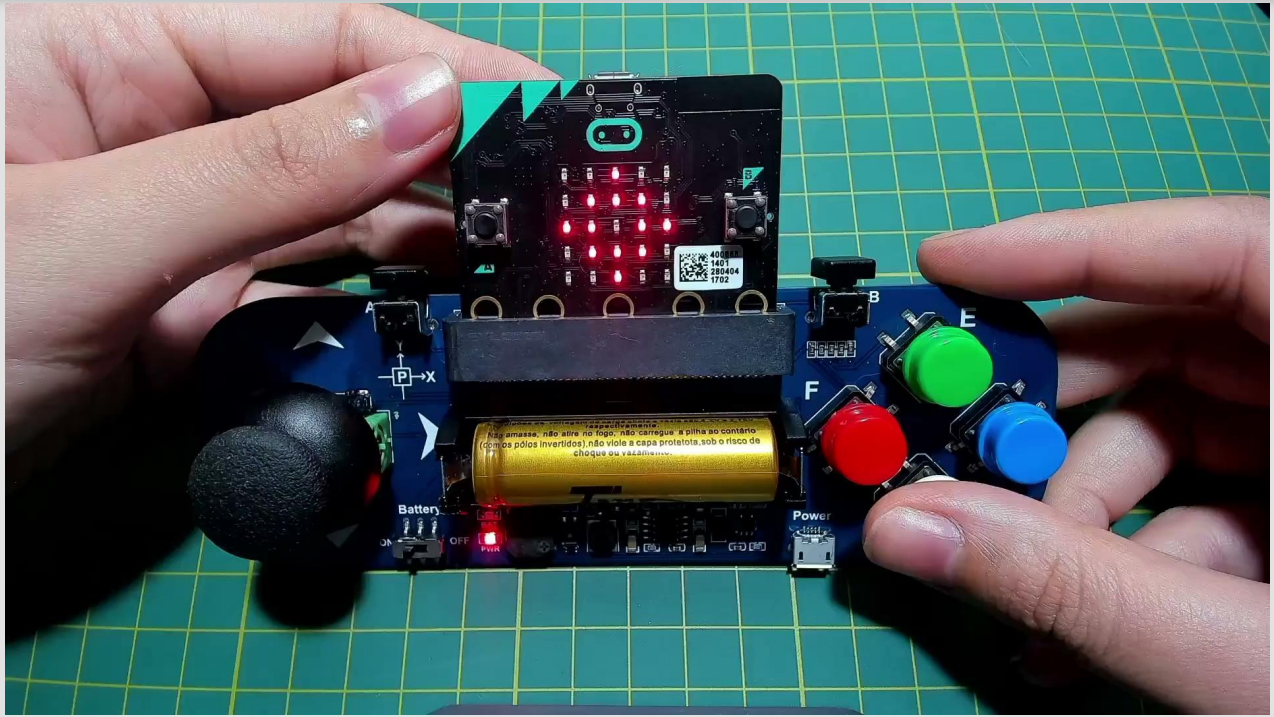
```
"""
Analog read the joystick X and Y axis' pins.
If x < 400, shifted left
If x > 600, shifted right

If y < 400, shifted down
If y > 600, shifted up
"""
from microbit import *

joystick_x = pin1
joystick_y = pin2

x = joystick_x.read_analog()
y = joystick_y.read_analog()
```

GAME EXAMPLES

GAME #1: GENIUS



GAME #1: GENIUS



https://www.youtube.com/watch?v=YnDZ1uw0zpU

#1 GENIUS: START GAME

```
class Game:
    def __init__(self):
        self.sequence = Sequence()
        self.input = Input()

    def start(self):
        display.show(Image.TARGET)
        music.play(music.ENTERTAINER)
        while not (button_a.is_pressed() and button_b.is_pressed()):
            display.clear()
            sleep(350)
            display.show(Image.TARGET)
            sleep(350)

        for number in "321":
            display.show(number)
            music.play(music.BA_DING)
            sleep(1000)
        music.play(music.JUMP_UP)
        display.clear()
```

#1 GENIUS: GENERATE THE SEQUENCE

```
class Sequence:
    def __init__(self):
        self.sequence = []
        self.directions = [
            DIRECTIONS["N"],
            DIRECTIONS["S"],
            DIRECTIONS["W"],
            DIRECTIONS["E"],
        ]

    def __len__(self):
        return len(self.sequence)

    def __getitem__(self, key):
        return self.sequence[key]
```

```
def show(self):
    self.sequence.append(
        random.choice(self.directions)
    )

    for direction in self.sequence:
        sleep(350)
        display.show(direction["image"])
        music.pitch(direction["freq"], PITCH_DURATION)
        sleep(350)
        display.clear()

    display.show(
        Image(
            "99999:99999:99999:99999:99999"
        )
    )
    sleep(350)
    display.clear()
```

#1 GENIUS: READ THE USER INPUT

```
class Button:
    def __init__(self, pin):
        self.pin = pin

    def is_pressed(self):
        return self.pin.read_digital() == 0


class Input:
    def __init__(self):
        self.input = []

        self.button_N = Button(pin13)
        self.button_S = Button(pin15)
        self.button_W = Button(pin12)
        self.button_E = Button(pin14)
```

```
    def read(self):
        if self.button_N.is_pressed():
            direction_pressed = DIRECTIONS["N"]
        elif self.button_S.is_pressed():
            direction_pressed = DIRECTIONS["S"]
        elif self.button_W.is_pressed():
            direction_pressed = DIRECTIONS["W"]
        elif self.button_E.is_pressed():
            direction_pressed = DIRECTIONS["E"]
        else:
            direction_pressed = None

        if direction_pressed:
            self.input.append(direction_pressed)
            display.show(direction_pressed["image"])
            music.pitch(
                direction_pressed["freq"],
                PITCH_DURATION
            )
            sleep(200)
            display.clear()
```

#1 GENIUS: VALIDATE THE USER INPUT

```
def check(self, sequence):
    while len(self.input) < len(sequence):
        self.read()
        if sequence[: len(self.input)] != self.input:
            return False
    return True

def clear(self):
    self.input.clear()
```

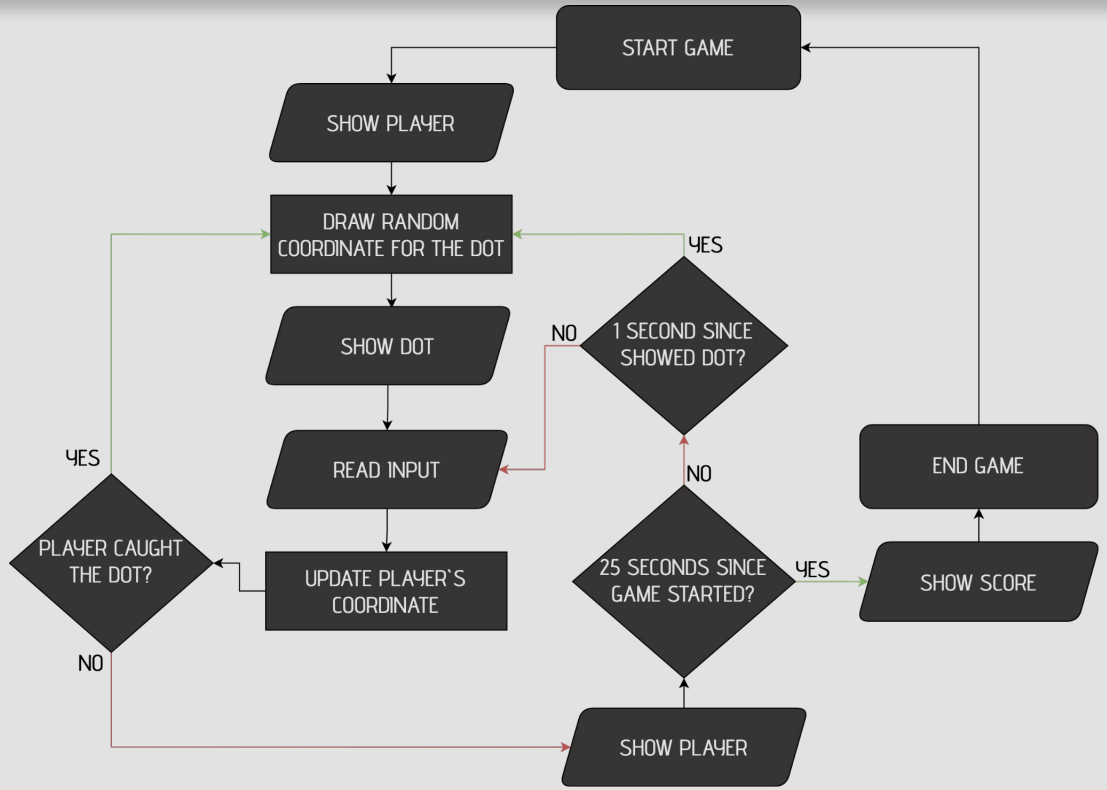
#1 GENIUS: END GAME

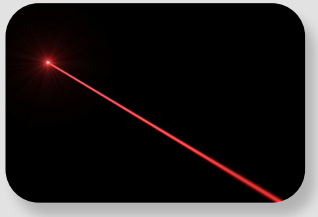
```
def end(self):
    display.show(Image.SAD)
    music.play(music.POWER_DOWN)
    display.scroll(len(self.sequence) - 1)
    sleep(1500)
    self.input.clear()
    self.sequence.clear()
```

#1 GENIUS: RUN THE GAME

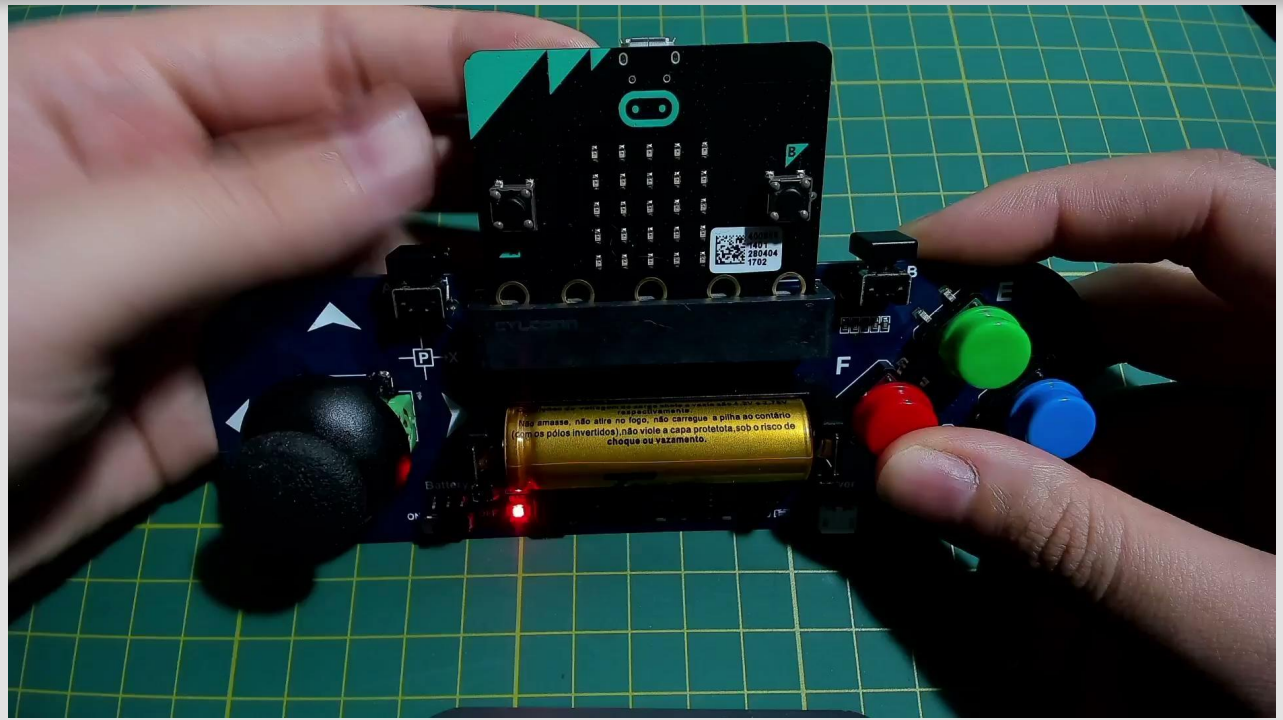
```
def check(self):
    if self.input.check(self.sequence):
        display.show(Image.HAPPY)
        music.play(music.POWER_UP)
        sleep(1500)
        display.clear()
        self.input.clear()
        return True
    return False

def run(self):
    self.start()
    while True:
        self.sequence.show()
        correct_answer = self.check()
        if not correct_answer:
            self.end()
            self.start()
```

GAME #2: CHASE THE DOT





GAME #2: CHASE THE DOT



https://www.youtube.com/watch?v=NVJH6F-3XuU

#2 CHASE THE DOT: GENERATE THE DOT

```
class Dot:
    def __init__(self):
        self.x = 0
        self.y = 0
        self.brightness = BRIGHTNESS_DOT
        self.time_showed = time.ticks_ms()

    def get_coordinates(self):
        return (self.x, self.y)

    def update_coordinates(self):
        self.x = random.randint(
            DISPLAY_MIN, DISPLAY_MAX
        )
        self.y = random.randint(
            DISPLAY_MIN, DISPLAY_MAX
        )
```

```
    def flash(self):
        self.clear()
        sleep(50)
        self.show()
        sleep(50)
        self.clear()
        sleep(200)

    def clear(self):
        display.set_pixel(self.x, self.y, 0)

    def show(self):
        self.time_showed = time.ticks_ms()
        display.set_pixel(
            self.x, self.y, self.brightness
        )
```

#2 CHASE THE DOT: READ THE USER INPUT

```
class Input:
    def __init__(self):
        self.toggle_button = pin15
        self.joystick_x = pin1
        self.joystick_y = pin2

    def get_input(self):
        if self.toggle_button.read_digital() == 0:
            x = accelerometer.get_x() + 512
            y = 512 - accelerometer.get_y()
        else:
            x = self.joystick_x.read_analog()
            y = self.joystick_y.read_analog()
        return x, y
```

#2 CHASE THE DOT: MOVE THE PLAYER

```
class Player:
    def __init__(self):
        self.x = 2
        self.y = 4
        self.brightness = BRIGHTNESS_PLAYER
        self.input = Input()

    def update_coordinates(self):
        x, y = self.input.get_input()
        if x > 600:  # right
            self.x = self.x + 1 if self.x < DISPLAY_MAX else DISPLAY_MAX
        elif x < 400:  # left
            self.x = self.x - 1 if self.x > DISPLAY_MIN else DISPLAY_MIN
        if y < 400:  # down
            self.y = self.y + 1 if self.y < DISPLAY_MAX else DISPLAY_MAX
        elif y > 600:  # up
            self.y = self.y - 1 if self.y > DISPLAY_MIN else DISPLAY_MIN

    def clear(self):
        display.set_pixel(self.x, self.y, 0)

    def show(self):
        display.set_pixel(self.x, self.y, self.brightness)
```

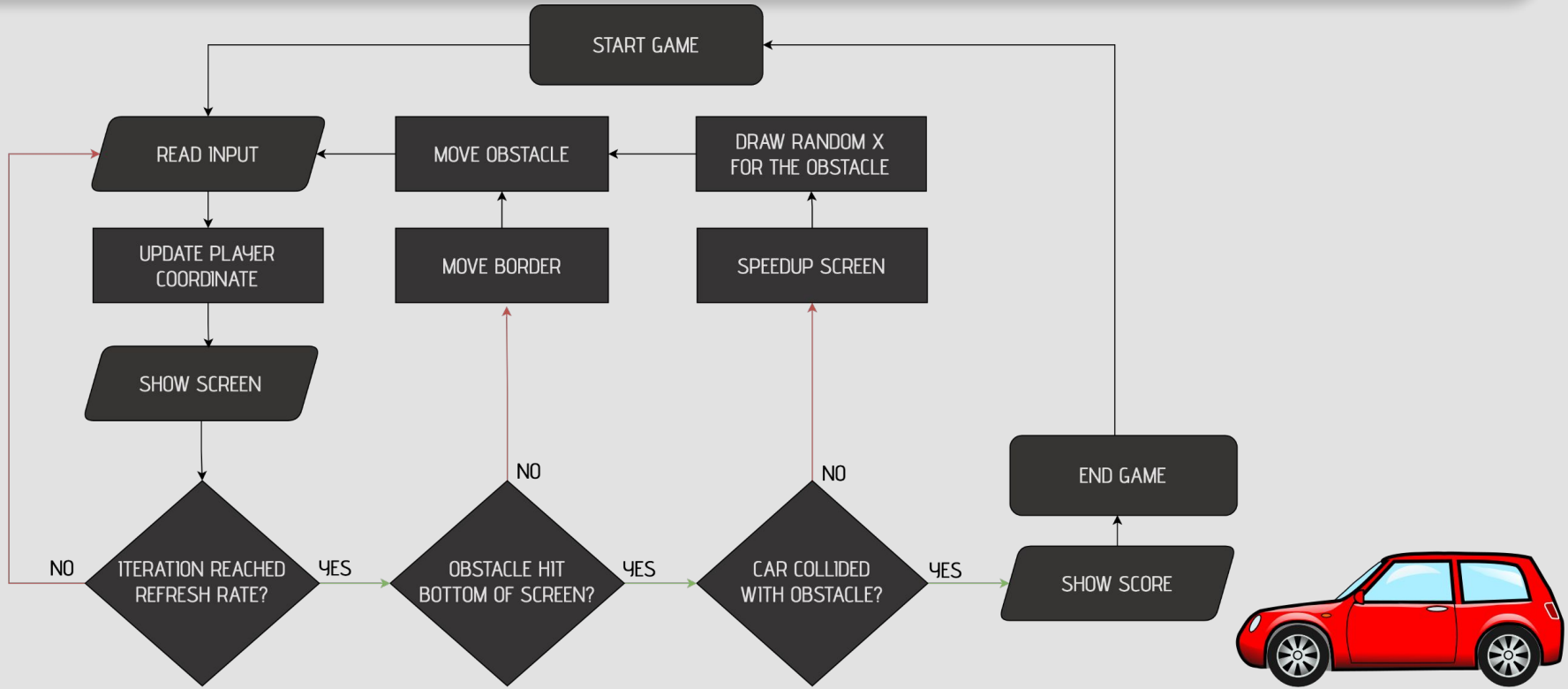
#2 CHASE THE DOT: CHECK COLLISION

```
class Game:
    def __init__(self):
        self.player = Player()
        self.dot = Dot()
        self.score = 0

    def check_collision(self):
        if self.player.x == self.dot.x and self.player.y == self.dot.y:
            self.score += 1
            return True
        return False
```

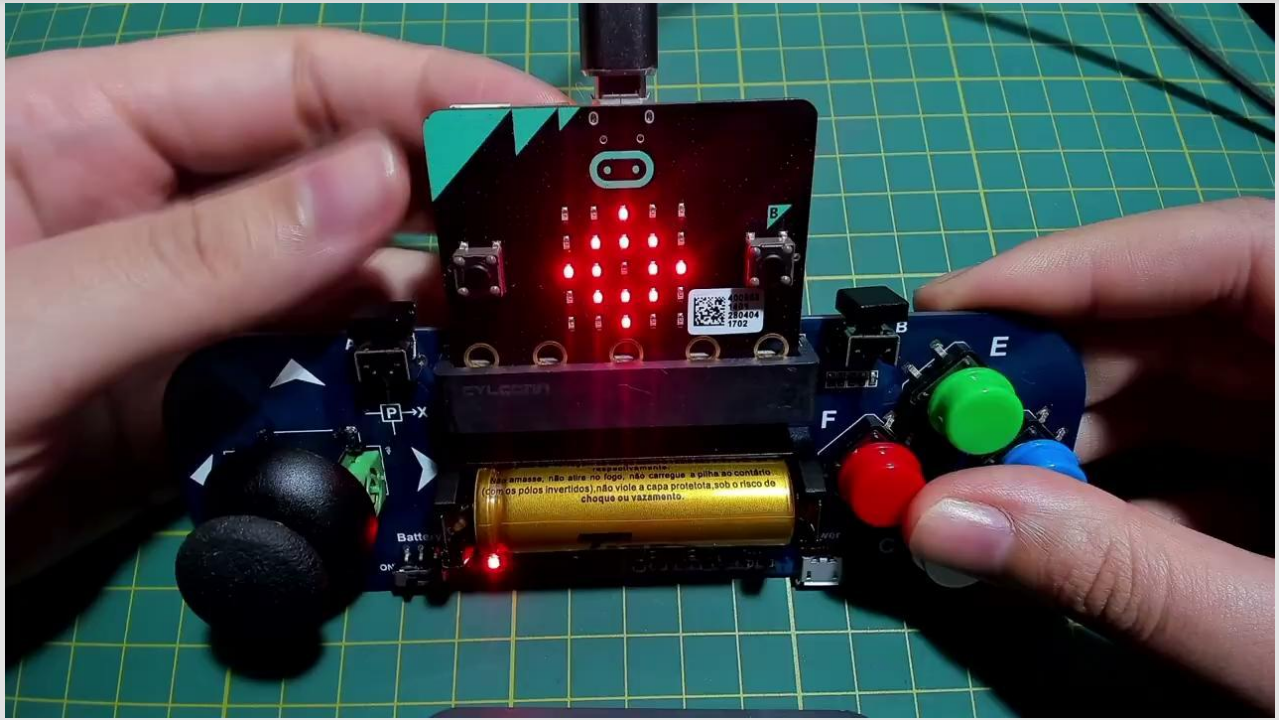
#2 CHASE THE DOT: RUN THE GAME

```
def run(self):
    self.start()
    while True:
        display.clear()
        self.player.show()
        self.dot.show()

        start = time.ticks_ms()
        now = time.ticks_ms()

        while time.ticks_diff(now, start) < 25000:
            if time.ticks_diff(
                now, self.dot.time_showed
            ) > 1000:
                self.dot.clear()
                self.dot.update_coordinates()
                self.dot.show()

            self.player.clear()
            self.player.update_coordinates()
            if self.check_collision():
                music.play(music.POWER_UP)
                self.dot.flash()
                while
                self.dot.x == self.player.x and
                self.dot.y == self.player.y:
                    self.dot.update_coordinates()
                self.dot.show()

            self.player.show()
            now = time.ticks_ms()
            sleep(150)

    self.end()
    self.start()
```

GAME #3: CAR CRASH



GAME #3: CAR CRASH



https://www.youtube.com/watch?v=vGMr4VFS-Q8

#3 CAR CRASH: PLAYER AND OBSTACLE

```
class Player:
    def __init__(self):
        self.x = 2
        self.y = 4
        self.brightness = BRIGHTNESS_PLAYER
        self.joystick_x = pin1

    def update_coordinates(self):
        x = self.joystick_x.read_analog()

        if x < 400: # left
            if self.x > LEFT_BORDER:
                self.x -= 1
        elif x > 600: # right
            if self.x < RIGHT_BORDER:
                self.x += 1

    def reset(self):
        self.x = 2
        self.y = 4
```

```
class Obstacle:
    def __init__(self):
        self.x = 2
        self.y = 0
        self.brightness = BRIGHTNESS_OBSTACLE

    def update_coordinates(self):
        if self.y < 5:
            self.y += 1
        if self.y == 5:
            self.x = random.randint(
                LEFT_BORDER, RIGHT_BORDER
            )
            self.y = 0

    def reset(self):
        self.x = 2
        self.y = 0
```

#3 CAR CRASH: BORDER

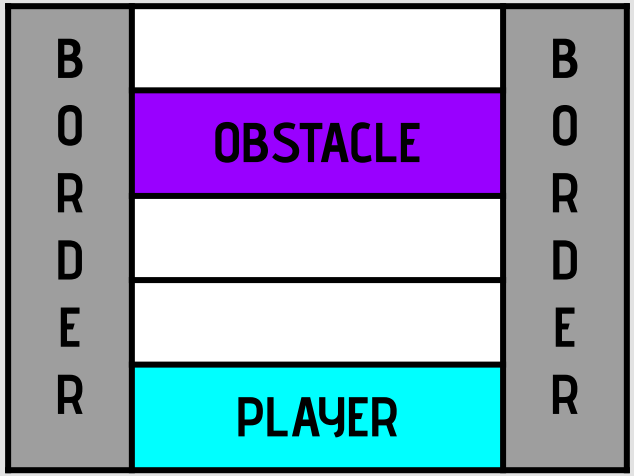
```
class Border:
    def __init__(self):
        self.top = 1
        self.bottom = 2
        self.brightness = BRIGHTNESS_BORDER

    def move(self):
        self.bottom = self.bottom + 1 if self.bottom < 4 else 0
        self.top = self.bottom + 1 if self.top < 4 else 0
```

#3 CAR CRASH: SCREEN

```
class Screen:
    def __init__(self):
        self.buffer = "00000:00000:00000:00000:00000"
        self.refresh = MAX_REFRESH_RATE

     def show(self):
        display.show(Image(self.buffer))

    def reset(self):
        self.refresh = MAX_REFRESH_RATE

    def speedup(self):
        if self.refresh >= MIN_REFRESH_RATE:
            self.refresh -= 1
```

#3 CAR CRASH: SCREEN BUFFER

```
def update_buffer(self, border, player, obstacle):
    border_space = ["0", "0", "0", "0", "0"]
    border_space[border.top] = str(border.brightness)
    border_space[border.bottom] = str(border.brightness)

    obstacle_space = ["0", "0", "0"]
    obstacle_space[obstacle.x - 1] = str(obstacle.brightness)

    buffer = ["00000", "00000", "00000", "00000", "00000"]

    for row in range(5):
        buffer[row] = border_space[row] + "000" + border_space[row]

    buffer[obstacle.y] = (
        buffer[obstacle.y][0] + "".join(obstacle_space) + buffer[obstacle.y][4]
    )

    buffer[player.y] = (
        buffer[player.y][: player.x]
        + str(player.brightness)
        + buffer[player.y][player.x + 1 :]
    )

    self.buffer = ":".join(buffer)
```



#3 CAR CRASH: RUN THE GAME

```
def check_collision(self):
    return self.player.x == self.obstacle.x and self.player.y == self.obstacle.y

def run(self):
    self.start()

    iteration = 0
    while True:
        self.player.update_coordinates()
        self.screen.update_buffer(self.border, self.player, self.obstacle)
        self.screen.show()
        iteration += 1

        if iteration >= self.screen.refresh:
            if self.obstacle.y == 4: # hit bottom
                if self.check_collision():
                    self.end()
                    self.start()
                    continue
                music.pitch(300, 100, wait=False)
                self.score += 1
                self.screen.speedup()
            self.border.move()
            self.obstacle.update_coordinates()
            iteration = 0

        sleep(100)
```

CONCLUSION

micro:bit + MicroPython == FUN

Thank you!

 julianaklulo/pycon-interactive-games

 julianaklulo