

INTRODUCTION TO MICROPYTHON

GETTING STARTED WITH
BBC MICRO:BIT



Juliana Karoline de Sousa | @julianaklulo

JULIANA KAROLINE DE SOUSA | @JULIANAKLULO



- Bachelor of Computer Science| UFSCar
- PyLadies São Carlos | co-founder and organizer
- Grupy-Sanca | co-founder e organizer
- Omnivector Solutions | software engineer
- IoT, robotics, 3D printing e kitties :3

Disclaimers

- 1 **No sponsors**
This tutorial is not sponsored
- 2 **Be careful**
Please handle the equipment
with care
- 3 **Return**
Don't forget to return the devices
before leaving the room
- 4 **Have fun!**

Agenda

01.

MicroPython

Overview about
MicroPython

02.

BBC micro:bit

Overview of the device
and the specifications

03.

Python editor

Using the Python editor
and built-in simulator

04.

Exercises

Experimenting with the
device's components

05.

Battleship

Build the single player
version of the game

06.

Challenge

Make the game
multiplayer

01. MicroPython

Python for microcontrollers



MicroPython

What is MicroPython?

Reimplementation of Python 3
targeted to microcontrollers



But what is a
microcontroller?

Microcontroller

Integrated Circuit

Processor, memory and peripherals all on a single chip



Programmable

Can store persistently user-defined code in memory



I/O Capabilities

Built-in input/output ports to interact with sensors and actuators



Low power consumption

Suitable for battery powered applications



MicroPython



Key points:

- Re-implementation of Python 3, written in C
- Contains a small subset of the standard library
- Optimized for constrained environments

Features:

- Interactive prompt (REPL)
- List comprehensions, generators, closures
- Exception handling
- Access to GPIO, PWM, ADC, UART and more

Requirements:

- 256k code space
- 16k RAM



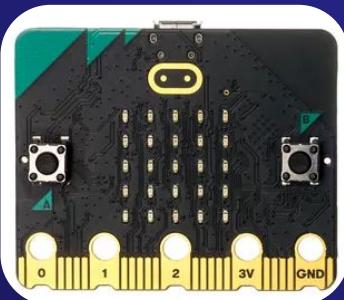
Microcontrollers with MicroPython



PyBoard



ESP8266 &
ESP32



BBC
micro:bit



Raspberry
Pi Pico

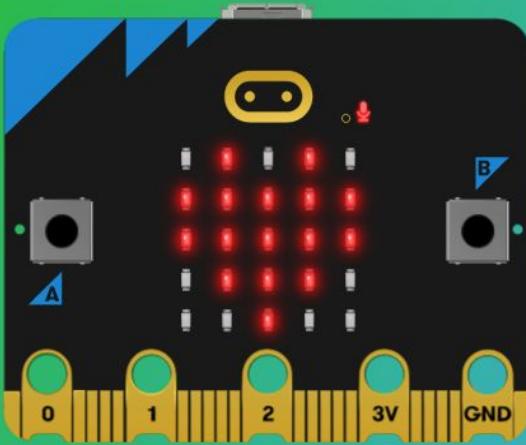


Arduino Nano
RP2040



02. BBC micro:bit

Pocket-sized computer for physical computing



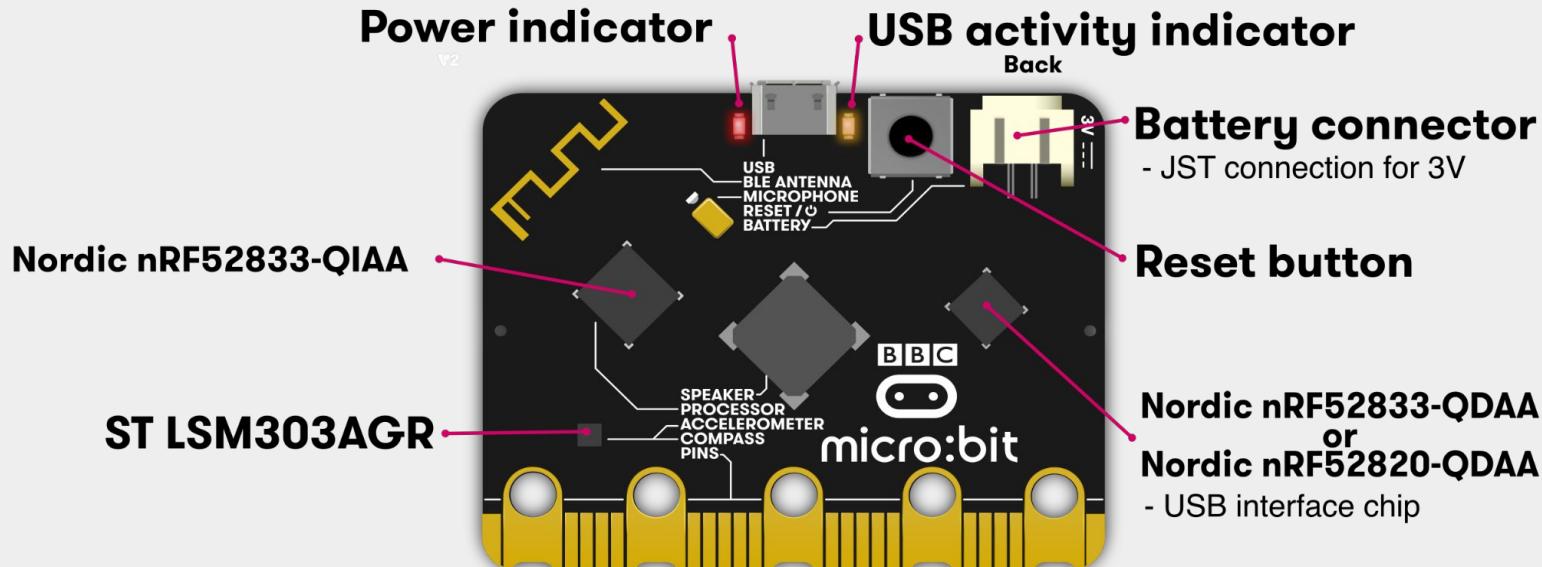
Create | **Learn** | **Code**

Get creative, get connected, get coding!

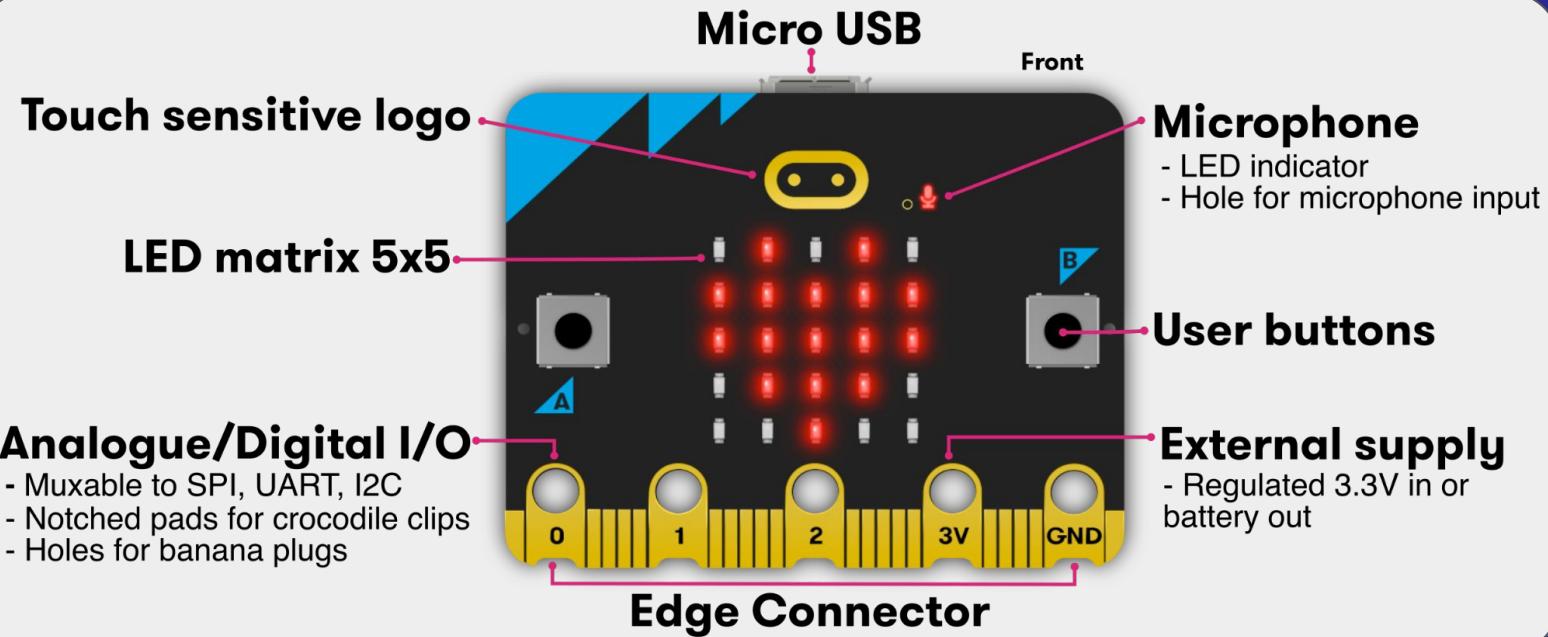
What is the BBC micro:bit?

Pocket-sized computer that introduces you to how software and hardware work together

Which components are included?



Which components are included?



03. Python editor

Easiest way to program the micro:bit



Online Python editor

The screenshot shows the micro:bit online Python editor interface. On the left, a sidebar titled "Reference" lists various categories: Variables, Display, Buttons, Loops, Logic, and Accelerometer. Below these are "Ideas" and "API" sections. The main workspace is titled "Untitled project" and contains the following Python code:

```
1 # Imports go at the top
2 from microbit import *
3
4
5 # Code in a 'while True:' loop repeats forever
6 while True:
7     display.show(Image.HEART)
8     sleep(1000)
9     display.scroll('Hello')
10
```

Below the code are buttons for "Send to micro:bit", "Save", and "Open...". To the right of the code is a graphical representation of the micro:bit board with pins labeled 0, 1, 2, 3V, and GND. A central circular button is highlighted. At the bottom right, there is a "Serial" port window showing sensor values for shake, light, temperature, and sound.

<https://python.microbit.org>



Workflow

Supported browsers (Chrome)

Plug micro:bit
into computer

Write **.py** file
in the Python editor

Click on
“save to micro:bit”
and select the micro:bit

Automatically flash
the **.hex** file
to micro:bit

Other browsers (Firefox)

Plug micro:bit
into computer

Write **.py** file
in the Python editor

Click on
“save to micro:bit”
to download **.hex** file
to your machine

Copy **.hex** file
to micro:bit
flash drive

Using the resources

Open the file **cheatsheet.py** in the repository

<https://github.com/julianaklulo/pycon2024-intro-to-micropython>

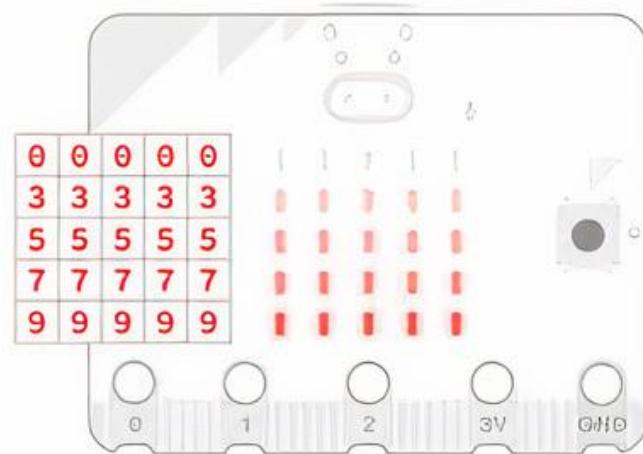




Using the Display

```
from microbit import *\n\n# Display an image\ndisplay.show(Image.HEART)\n\n# Display custom image\ndisplay.show(Image('00990:\n    \"00900:\n    \"00900:\n    \"99999:\n    \"09990'))\n\n# Scroll text\ndisplay.scroll("PyCon US")\n\n# Set pixels\ndisplay.set_pixel(0, 0, 9)\n\n# Clear display\ndisplay.clear()
```

Image('00000:\n '33333:\n '55555:\n '77777:\n '99999:')

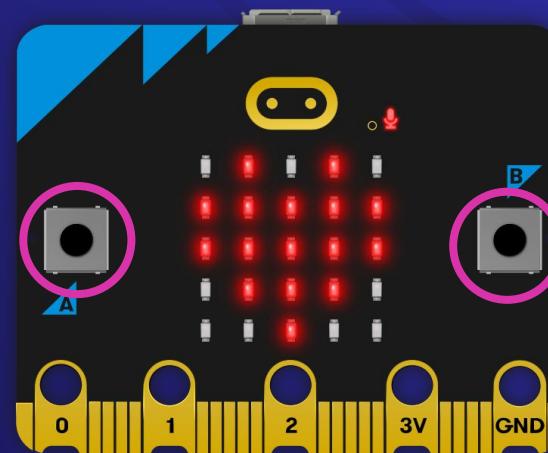


0-9: Brightness intensity



Using the Push Buttons

```
from microbit import *\n\n# Check if button A was pressed\nif button_a.was_pressed():\n    pass\n\n# Check if button A is pressed\nif button_a.is_pressed():\n    pass\n\n# Get how many times button A was pressed\nquantity = button_a.get_presses()
```



Button A (left) and Button B (right)



Using the Speaker

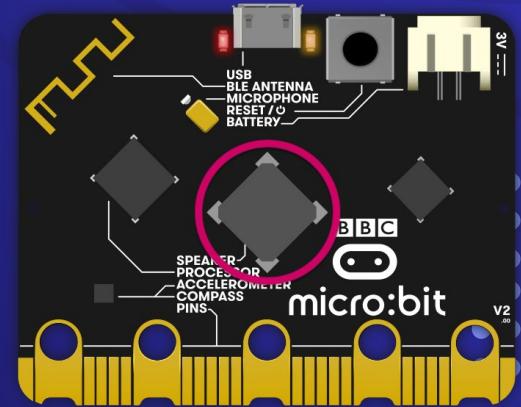
```
from microbit import *
import music, speech, audio

# Play pre-defined music
music.play(music.POWER_UP)

# Create your own music - Format: '{note}{octave}:{duration}'
music.play([
    "G4:2", "A4:2", "B4:4", "D5:4", "D5:4", "B4:4", "C5:4", "C5:4",
    "r:2", "G4:2", "A4:2", "B4:4", "D5:4", "D5:4", "C5:4", "B4:8"
])

# Text to speech
speech.say("Hello, PyCon!")

# Play expressive sounds
audio.play(sound.YAWN)
```





Using the Microphone

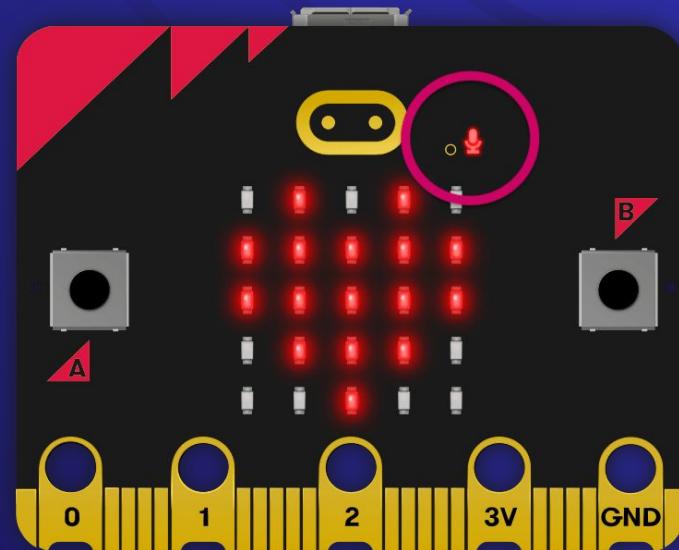
```
from microbit import *

# SoundEvent can be LOUD or QUIET

# Read current event
event = microphone.current_event()

# Get the sound history as a tuple
history = microphone.get_events()

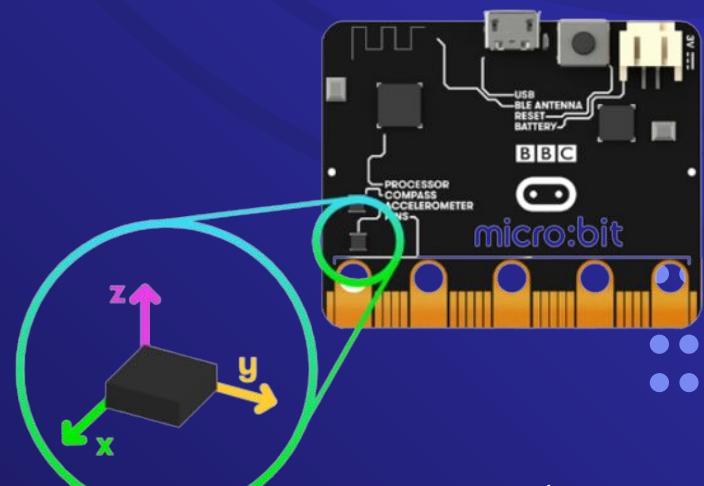
# Get sound level (0-255)
level = microphone.sound_level()
```





Using the Accelerometer

```
from microbit import *\n\n# Read acceleration in each axis\nx = accelerometer.get_x()\ny = accelerometer.get_y()\nz = accelerometer.get_z()\n\n# Read pre-defined current gesture\ngesture = accelerometer.is_gesture("face up")\n\n# Read pre-defined past gesture\ngesture = accelerometer.was_gesture("shake")
```





Using the Radio

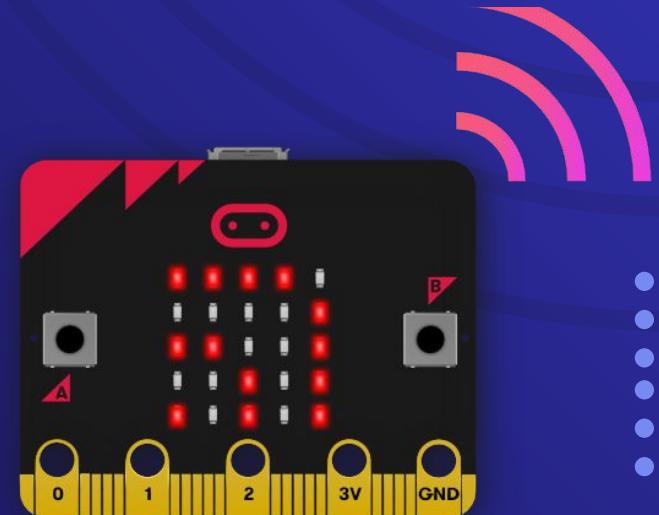
```
import radio

# Turn on the radio
radio.on()

# Set the radio group and power
# Group: 0-255, Power: 0-7 (signal intensity)
radio.config(group=23, power=7)

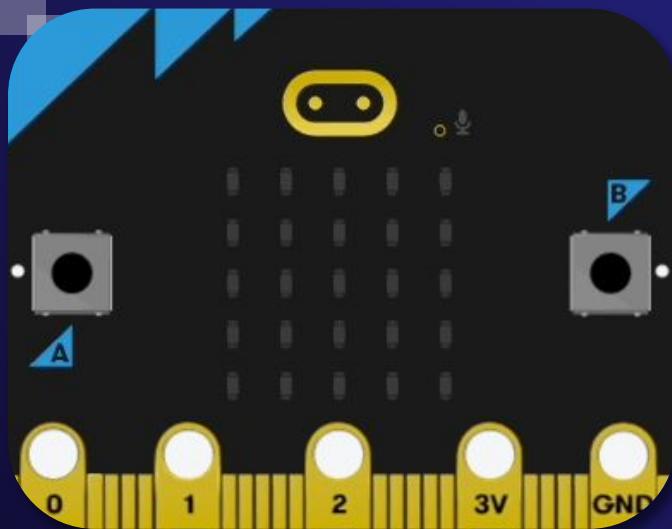
# Receive a message from the group
message = radio.receive()

# Send a message to the group
radio.send("PyCon US")
```



04. Exercises

Let's use each component available in the device



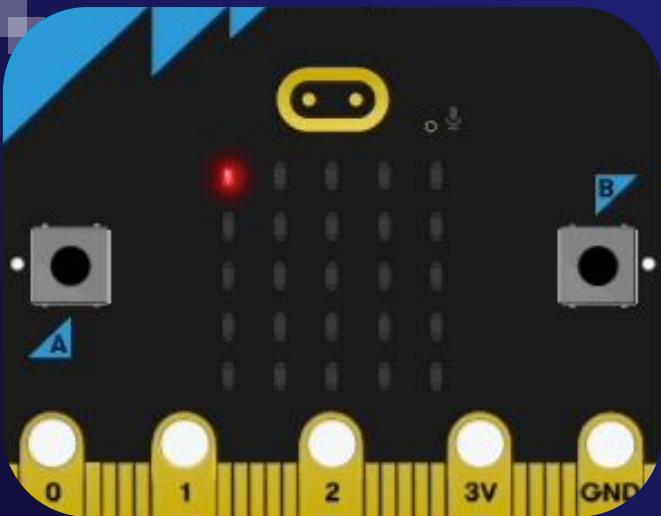
Exercise #1

- Show an arrow pointing to the direction of the button that is pressed:
 - Left if the A button is pressed
 - Right if the B button is pressed

Exercise #1

```
from microbit import *

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



Exercise #2

- Display a pixel at coordinate (2, 2) and make it move based on the accelerometer values:
 - Read x and y axis
 - Increase or decrease the coordinates based on the the values read
 - Show the pixel at the new coordinates
- Tip: **clear the pixel** before showing it at the new coordinates.



Exercise #2

```
from microbit import *

x, y = 2, 2

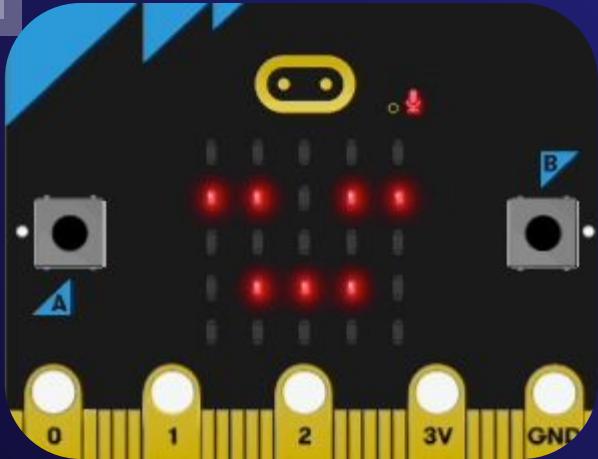
while True:
    display.set_pixel(x, y, 0)

    if accelerometer.get_x() > 0:
        x = min(x + 1, 4)
    else:
        x = max(x - 1, 0)

    if accelerometer.get_y() > 0:
        y = min(y + 1, 4)
    else:
        y = max(y - 1, 0)

    display.set_pixel(x, y, 9)
    sleep(100)
```





Exercise #3

- Get the current sound event and display:
 - Happy face if the noise is “loud”
 - Asleep face if noise is “quiet”



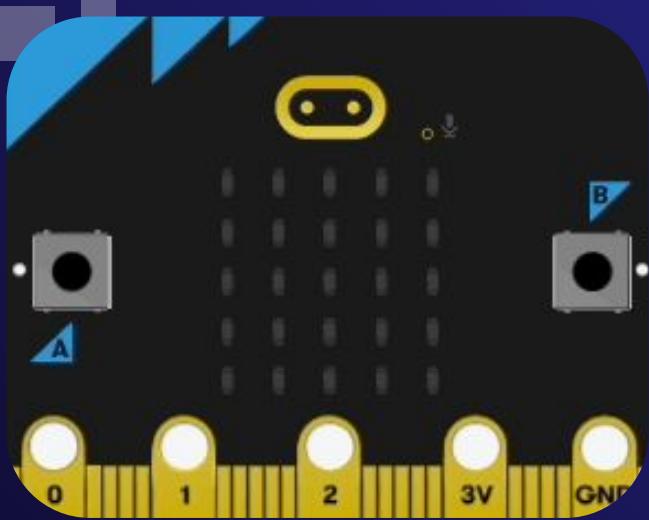
Exercise #3

```
from microbit import *

while True:
    if microphone.current_event() == SoundEvent.LOUD:
        display.show(Image.HAPPY)
    elif microphone.current_event() == SoundEvent.QUIET:
        display.show(Image.ASLEEP)

    sleep(100)
    display.clear()
```





Exercise #4

- Create a teleporting duck:
 - Choose a partner and define a radio group
 - Send “duck” when the device is shaken
 - Display a duck when “duck” is received
- Tip: **clear the display** before sending the duck

Exercise #4

```
from microbit import *
import radio

GROUP = 23

radio.on()
radio.config(group=GROUP)

while True:
    if accelerometer.was_gesture("shake"):
        display.clear()
        radio.send("duck")
    if radio.receive() == "duck":
        display.show(Image.DUCK)
        sleep(100)
```



05. Battleship

Single-player version of the game

Battleship Breakdown



Sea

Stores the automatically generated ships and checks if a shot hit a ship



Player

Handles the logic to fire a shoot at the sea and stores the shots fired



Game

Runs the game until the player hits all the ships



Sea

How to populate the sea with ships?



Rules to place the ships



1

Size matters

Place the biggest ships first

2

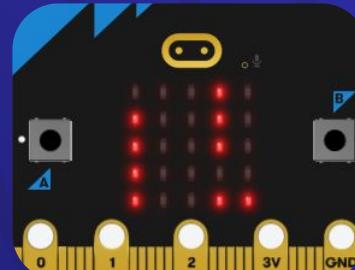
Location matters

The ship can't overflow the board

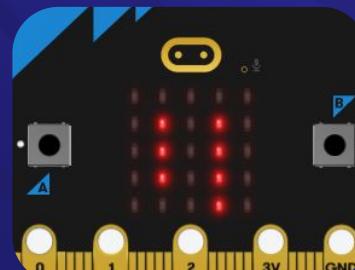
3

No touching

Each boat must be surrounded
only by water



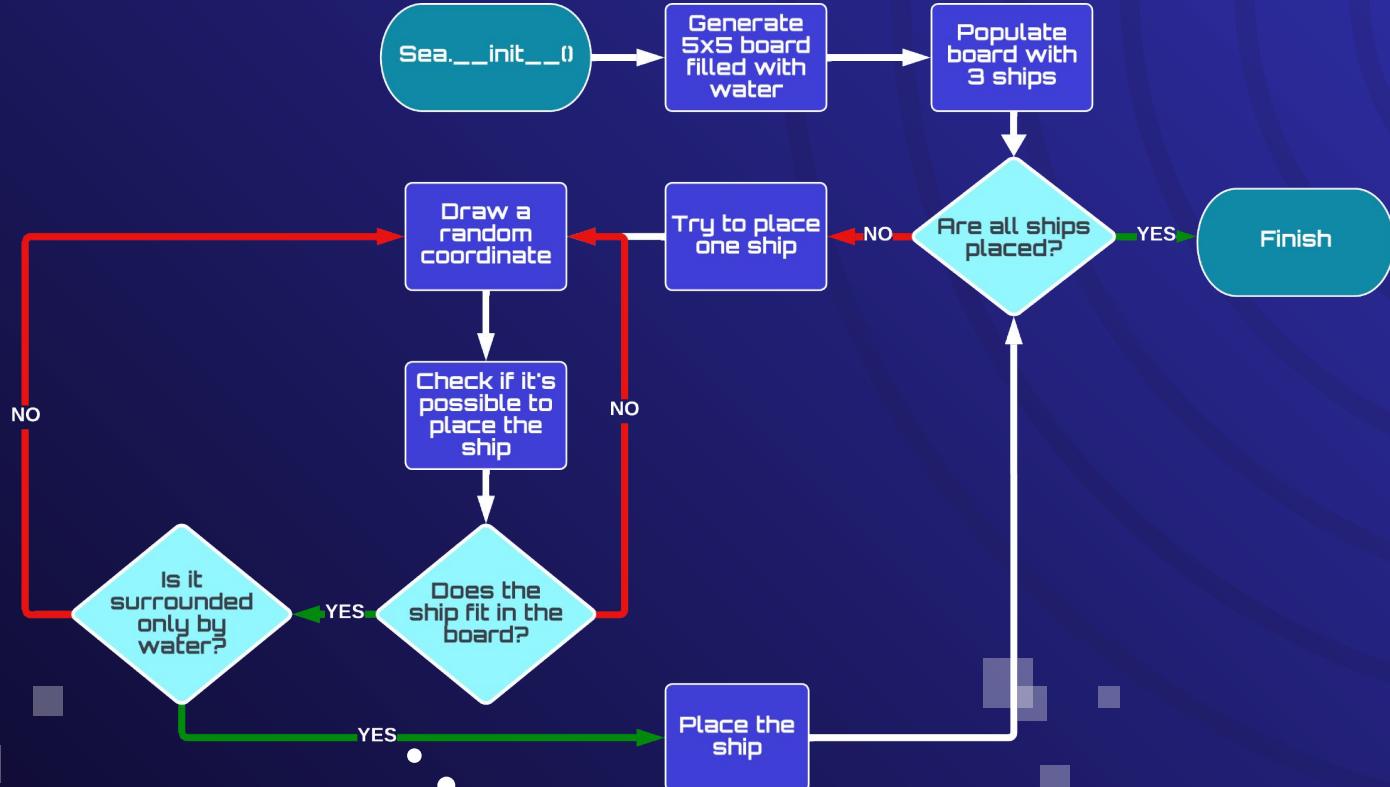
Valid placement



Impossible to place
the third ship



Sea Breakdown





Analyzing the code

Open file **battleship-single-player/template_1.py** in the repository

Sea - Generate board



```
class Sea:  
    """  
        Holds the state of the game board.  
    """  
    ship sizes  
    def __init__(self, ships=[4, 3, 2]):  
        """  
            Initialize the game board with the given ships.  
        """  
        self.board = [[WATER] * 5 for _ in range(5)]  
        self.populate_board(ships)
```

SHIP = 9
WATER = 2

brightness

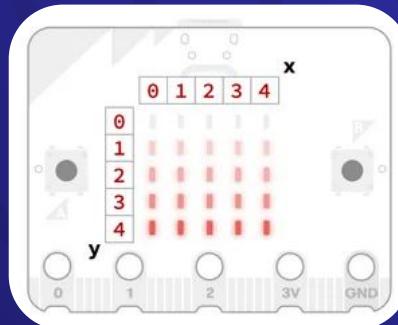
Sea - Generate board



```
board = [  
    [WATER, WATER, WATER, WATER, WATER],  
    [WATER, WATER, WATER, WATER, WATER],  
]
```

Index: [ROW] [COL]

VS



Index: [COL] [ROW]

Sea - Populate board



```
def populate_board(self, ships):
    """
    Place all the ships on the board.

    If it is not possible to place all the ships, try again.
    """

    while True:
        placed_ships = [self.place_ship(ship) for ship in ships]

        if all(placed_ships):
            break

    self.board = [[WATER] * 5 for _ in range(5)]
    reset the board if the ships don't fit
```

Sea - Place Ship (part 1)



```
def place_ship(self, size):
    """
    Place a ship of the given size in the board.

    The ship will be placed in a random candidate coordinate.
    The orientation can be horizontal or vertical.
    """

    candidates_coordinates = [
        (row, col) for row in range(5)
        for col in range(5) if self.board[row][col] == WATER
    ]
    select among the candidates coordinates
    while candidates_coordinates:
        row, col = random.choice(candidates_coordinates)
        candidates_coordinates.remove((row, col))

        orientation = random.choice(["H", "V"])

        if self.possible(row, col, size, orientation):
            break
    else:
        return False
fails if none of the coordinates work
```



Each water coordinate is a candidate

Sea - Possible



```
def possible(self, row, col, size, orientation):
    """
    Check if it is possible to place a ship.

    The ship must not be near other ships.
    """

    for i in range(size): check for each part of the boat
        if orientation == "H":
            if not self.near_ships(row, col + i):
                return False
        elif orientation == "V":
            if not self.near_ships(row + i, col):
                return False
    return True
```

H: hold the row and increase the col

V: increase the row and hold the col

	WATER	WATER	WATER	WATER
	WATER	SHIP	SHIP	SHIP
	WATER	WATER	WATER	WATER

desired placement

Sea - Near Ships



```
def near_ships(self, row, col):
    """
    Check if there are ships surrounding the given coordinates.

    If the coordinates are out of bounds, return False.
    """
    if row > 4 or col > 4:
        return False
    non-inclusive
    for i in range(row - 1, row + 2):
        for j in range(col - 1, col + 2):
            exclude the coordinate
            if 0 <= i < 5 and 0 <= j < 5 and (i != row or j != col):
                if self.board[i][j] == SHIP:
                    return False
    return True
```



Example: (2, 2)



Sea - Place Ship (part 2)

```
for i in range(size):
    if orientation == "H":
        self.board[row][col + i] = SHIP
    elif orientation == "V":
        self.board[row + i][col] = SHIP

return True
```

H: hold the row and increase the col

V: increase the row and hold the col

Sea - Hit



```
def hit(self, row, col):
    """
    Check if there is a ship in the given coordinates.
    """
    return self.board[row][col] == SHIP
```

will be used once the player starts shooting

Sea - Show Board



```
def show(self):
    """
    Show the game board.
    """
    display.show(
        Image(
            ":".join(
                ["".join(str(point) for point in line) for line in self.board]
            )
        )
    )
```

create one string for each row
then concatenate each string using :

Sea - Show Board

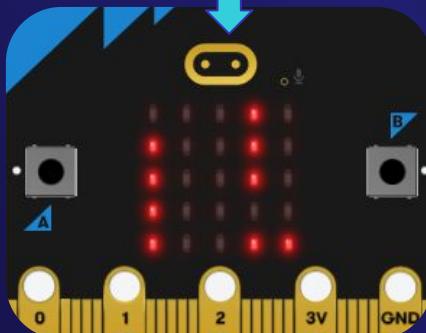


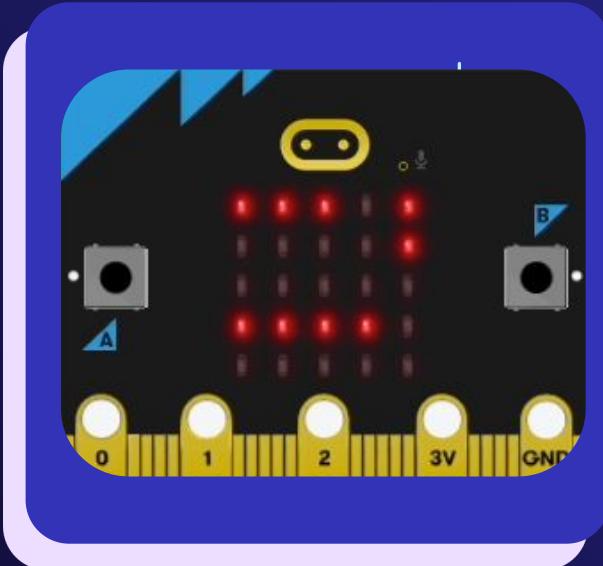
```
board = [  
    [WATER, WATER, WATER, SHIP, WATER],  
    [SHIP, WATER, WATER, SHIP, WATER],  
    [SHIP, WATER, WATER, SHIP, WATER],  
    [SHIP, WATER, WATER, WATER, WATER],  
    [SHIP, WATER, WATER, SHIP, SHIP],  
]
```

```
board = Image("22292:92292:92292:92222:92299")
```

SHIP = 9
WATER = 2

represents the brightness





Try it in the simulator

Open **template_1.py**, paste in the simulator and execute the code

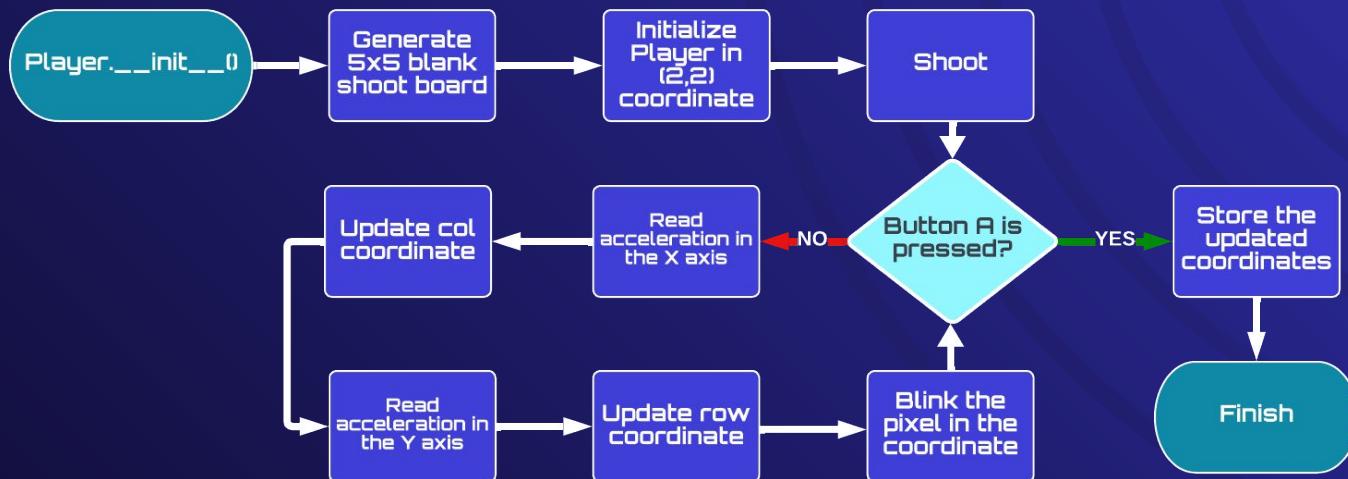
<https://python.microbit.org>



Player

How to shoot the ships?

Player Breakdown





Analyzing the code

Open file **battleship-single-player/template_2.py** in the repository

Player - Generate Shots Board



```
class Player:  
    """  
        Holds the state of the player board.  
    """  
  
    def __init__(self):  
        """  
            Initialize the player board.  
        """  
  
        self.shots = [[0] * 5 for _ in range(5)]  
  
        self.row = 2  
        self.col = 2  
  
start player in the center  
of the board
```

same structure as the
sea board

Player - Mark



```
def mark(self, row, col, hit):
    """
    Mark the player board with the result of the shot.
    """
    if hit:
        self.shots[row][col] = SHIP
    else:
        self.shots[row][col] = WATER
```

similar to the hit method

Player - Shoot



1

While loop

Run until Button A is pressed

2

Show the board

Show the shots board so the user knows where they've already shot

3

Threshold

Value to filter out small movements or noise in the accelerometer readings

4

Update row

Use accelerometer axis Y to update the row

5

Update col

Use accelerometer axis X to update the col

6

Blink

Alternate between the coordinate in the shot board and the current coordinate

Writing the shoot method

Edit file **battleship-single-player/template_2.py** in the repository



Player - Shoot

```
def shoot(self):
    """
    Shoot the board.

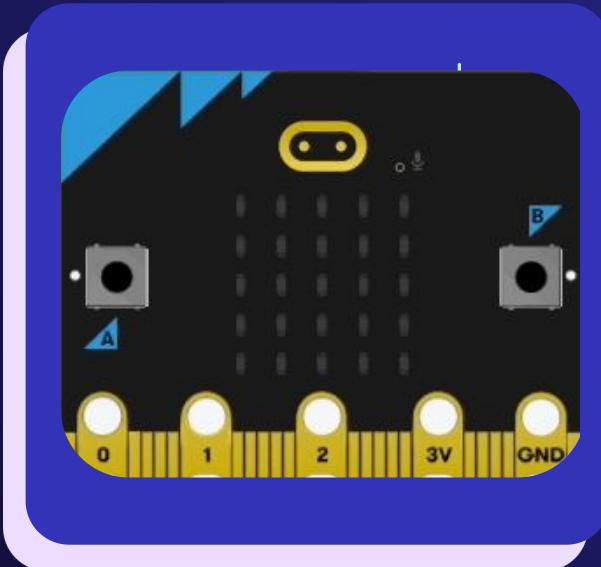
    The player can move the cursor with the accelerometer.
    The player can shoot with the A button.

    The coordinates are stored for future reference.
    """
    while not button_a.is_pressed():
        self.show()

        if accelerometer.get_x() > THRESHOLD:
            self.col = min(self.col + 1, 4)
        elif accelerometer.get_x() < -THRESHOLD:
            self.col = max(self.col - 1, 0)

        if accelerometer.get_y() > THRESHOLD:
            self.row = min(self.row + 1, 4)
        elif accelerometer.get_y() < -THRESHOLD:
            self.row = max(self.row - 1, 0)

        self.blink(self.row, self.col)
```



Try it in
the micro:bit

Upload your version of the
template_2.py to the micro:bit

<https://python.microbit.org>



Game

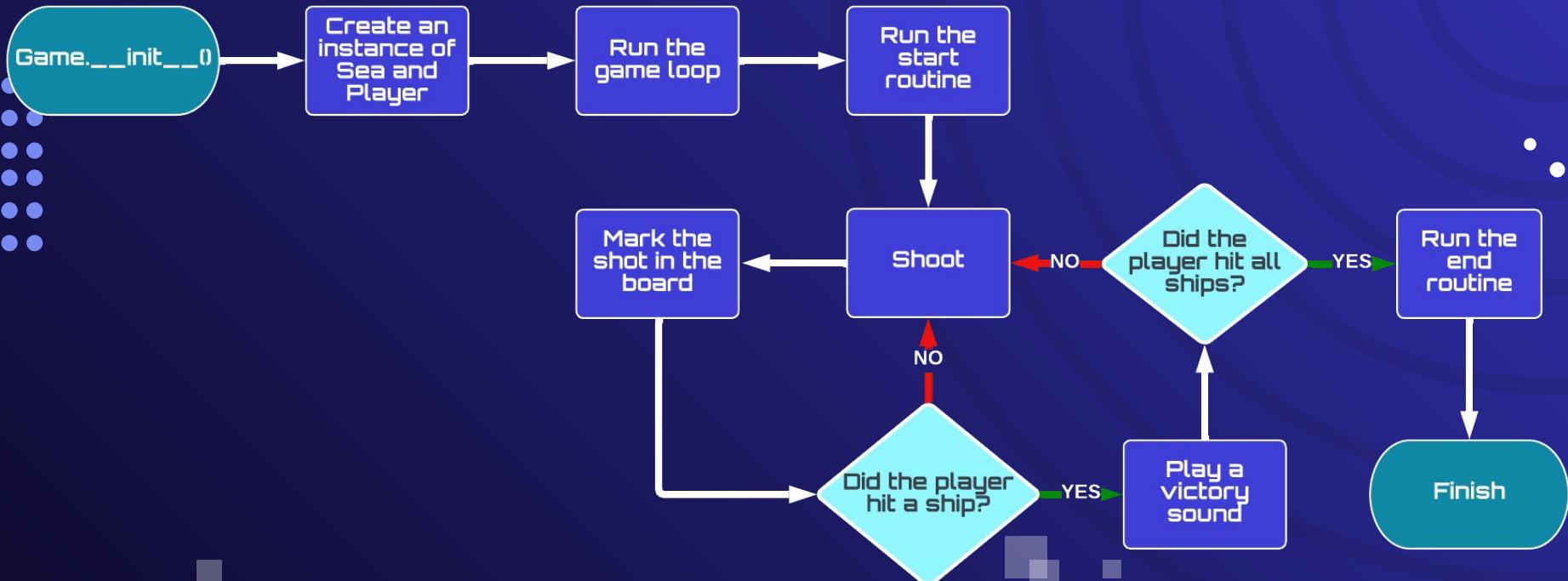
Control the logic of the game



Analyzing the code

Open file **template_3.py** in the repository

Game Breakdown



Game - Init Sea and Player



```
class Game:  
    """  
        Holds the state of the game.  
    """  
  
    def __init__(self):  
        """  
            Initialize the game.  
  
            The game board is initialized with the ships.  
            The player board is initialized.  
  
            The player starts at the top-left corner.  
        """  
  
        self.sea = Sea()  
        self.player = Player()
```

Game - Win



```
def win(self):
    """
    Check if the player has won the game.

    The player wins the game if they hit all the ships.
    """

    for i in range(5):
        for j in range(5):
            if self.sea.board[i][j] == SHIP and self.player.shots[i][j] != SHIP:
                return False
    return True
```

Game - Start and End



1

Greet the player

Play sound and display image

4

Only one end

The game always ends in victory

2

Wait for input

Keep blinking the image until the player press both buttons

5

Display a positive image

Happy face is a good choice

3

Countdown

Display a countdown to indicate that the game is about to start

6

Play an ending sound

Choose an appropriate sound



Writing start and end methods

Edit file **battleship-single-player/template_3.py** in the repository

Game - Start and End



```
def start(self):
    """
    Routine to start the game.
    """
    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()
```

```
def end(self):
    """
    Routine to end the game.
    """
    display.show(Image.HAPPY)
    music.play(music.CHASE)
```

Game - Run



1

Start the game

Run the start routine

2

Loop until end

Run the loop until the game ends with the player winning

3

Shoot the board

Read the player's shot

4

Check hit and mark

Check in the Sea instance if the shot hit any ships

5

Check win

Check if the player has already hit all the ships and run the end routine if so

6

Display the result

Let the player know if the shot has hit or missed the ships



Writing the Game run method

Continue editing file **battleship-single-player/template_3.py** in the repository

Game - Run



```
def run(self):
    """
    Run the logic of the game.

    The player can shoot in the board.

    If the player hits a ship, the player board is marked with a ship.
    If the player misses, the player board is marked with water.

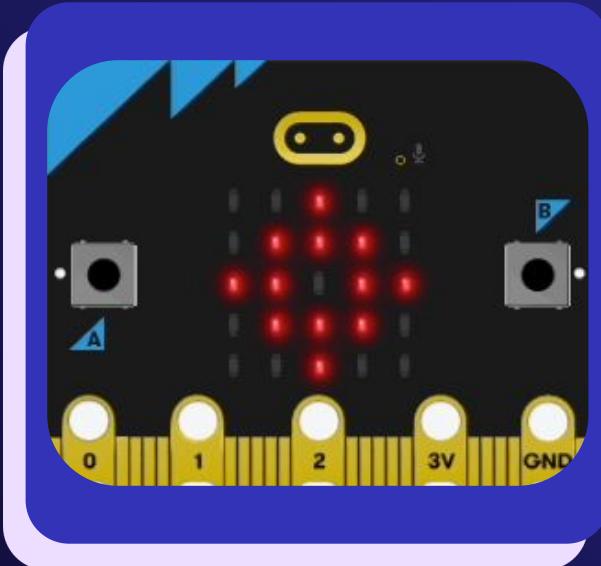
    If the player hits all the ships, the game ends.
    """
    self.start()

    while True:
        self.player.shoot()

        hit = self.sea.hit(self.player.row, self.player.col)
        self.player.mark(self.player.row, self.player.col, hit)

        if hit:
            display.show(Image.TARGET)
            music.play(music.POWER_UP)

            if self.win():
                self.end()
                break
            else:
                display.show(Image.NO)
                music.play(music.POWER_DOWN)
```



Try it in the micro:bit

Upload your version of the
template_3.py to the micro:bit

<https://python.microbit.org>

06. Challenge

Multiplayer version of the game

Multiplayer Breakdown



Sea

No changes needed



Player

No changes needed



Game

Handles the publisher
and subscriber logic



Game

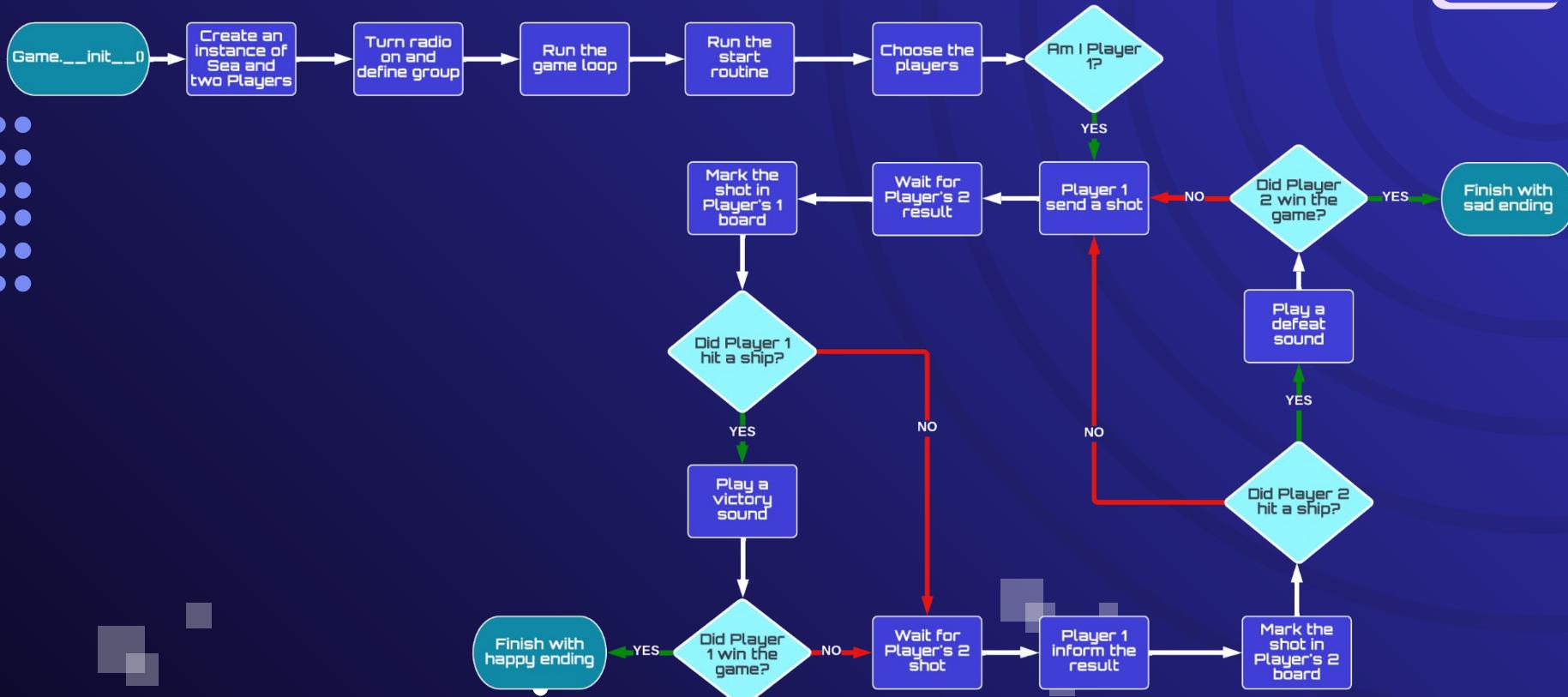
Control the logic of the game



Analyzing the code

Open file **template_3.py** in the repository

Game Breakdown



Game - Init Sea and Players



```
class Game:  
    """  
        Holds the state of the game.  
    """  
  
    def __init__(self):  
        """  
            Initialize the game.  
  
            Create the game board and the player boards.  
            Each player will have a board to place the ships and a board to make the shots.  
        """  
  
        self.sea = Sea()  
        self.me = Player() same Player class for both players  
        self.opponent = Player()  
  
        self.winner = ""  
  
        radio.on()  
        radio.config(group=GROUP) define the group with a partner and set the constant
```

Game - Start



same as the single player version

```
def start(self):
    """
    Routine to start the game.
    """

    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()
```

Game - Lost



```
def lost(self):
    """
    Check if the player has lost the game.

    The player loses the game if the opponent hits all the ships.
    """
    for i in range(5):
        for j in range(5):
            if self.sea.board[i][j] == SHIP and self.opponent.shots[i][j] != SHIP:
                return False
    return True
```

only the player receiving the shot can check if they've lost the game

Game - End

different endings depending on which player won the game

```
def end(self, win):
    """
    Routine to end the game.

    If the player wins, show a happy face.
    If the player loses, show a sad face.
    """

    if win:
        display.show(Image.HAPPY)
        music.play(music.CHASE)
    else:
        display.show(Image.SAD)
        music.play(music.WAWAWAWAA)
```



Game - Choose Players



1

Display boat image

Show an image while waiting for the input from the players

2

Loop until press

Wait for both players to be ready to start the game

3

Player 1 - Button A

Send a message after Player 1 pressed Button A, wait for response from Player 2

4

Player 2 - Button B

After Player 2 pressed Button B, wait for message from Player 1 and reply with acknowledgment message

5

Assign the players

Store the player number to define who starts the turn



Writing the Choose Players method

Edit file **battleship-multiplayer/template_1.py** in the repository

Game - Choose Players



```
def choose_players(self):
    """
    Routine to choose the players.

    The first player to press the A button will be PLAYER_1.
    The second player to press the B button will be PLAYER_2.
    """

    display.show(Image("00990:00900:00900:99999:09990"))

    while True:
        if button_a.is_pressed():
            radio.send("PLAYER_1 READY")
            while True:
                message = radio.receive()
                sleep(100)
                if message and message == "PLAYER_2 READY":
                    self.me.player_number = "PLAYER_1"
                    self.opponent.player_number = "PLAYER_2"
                    display.show("1")
                    break
            break
        break
```

```
if button_b.is_pressed():

    while True:
        message = radio.receive()
        sleep(100)
        if message and message == "PLAYER_1 READY":
            radio.send("PLAYER_2 READY")
            self.me.player_number = "PLAYER_2"
            self.opponent.player_number = "PLAYER_1"
            display.show("2")
            break
        sleep(1000)
```

define the group with a partner and set the constant

Game - Send Shot



1

Run Shoot method

Update the coordinates of the current player

2

Send message

Send to the opponent the coordinates of the shot

3

Wait for response

Wait for the opponent to reply with the result of the shot

4

Mark result

Mark in the current player's board the result of the shot

5

Check hit

Celebrate if the current player's shot hit a ship

6

Return value

Returns True if the current player won





Writing the Send Shot method

Continue editing the file **battleship-multiplayer/template_1.py** in the repository

Game - Send Shot

```
def send_shot(self):
    """
    Send a shot to the opponent.

    The player shoots in the opponent's game board.
    The opponent will respond with the result of the shot
    and if the player has won the game.
    The player will mark the result of the shot in the player board.

    Returns True if the current player has won the game.
    """
    self.me.shoot()

    message = str(self.me.row) + " " + str(self.me.col)
    radio.send(message)

    response = None
    while not response:
        response = radio.receive()
        sleep(100)
```

```
hit, won = response.split(" ")
hit = hit == "True"
won = won == "True"

self.me.mark(self.me.row, self.me.col, hit)

if hit:
    display.show(Image.TARGET)
    music.play(music.POWER_UP)
else:
    display.show(Image.NO)
    music.play(music.POWER_DOWN)

return won
```



Game - Receive Shot



1

Show the board

Display the sea board to the current user

2

Wait for message

Wait for the opponent to make the shot

3

Blink the coordinate

Display in the board the coordinate that the opponent shot

4

Check hit and mark

Check if the opponent hit a ship in the sea board and mark the shot in the opponent's board

5

Check lose

Check if the opponent hit the last ship in the current player's board

6

Send response

Send message with result of the shot and if the opponent won the game

7

Show shot result

Display the result of the shot to the current player

8

Return value

Returns True if the current player lost





Writing the Receive Shot method

Continue editing the file **battleship-multiplayer/template_1.py** in the repository

Game - Receive Shot



```
def receive_shot(self):
    """
    Receive a shot from the opponent.

    The player will receive a shot from the opponent.
    The player will mark the result of the shot in the opponent board.
    The player will respond with the result of the shot
    and if the opponent has won the game.

    Returns True if the current player has lost the game.
    """
    self.sea.show()

    response = None
    while not response:
        response = radio.receive()
        sleep(100)

    row, col = response.split(" ")
    row = int(row)
    col = int(col)
```

```
    self.sea.blink(row, col)
    hit = self.sea.hit(row, col)
    self.opponent.mark(row, col, hit)

    lost = self.lost()

    message = str(hit) + " " + str(lost)
    radio.send(message)

    if hit:
        display.show(Image.TARGET)
        music.play(music.POWER_DOWN)
    else:
        display.show(Image.NO)
        music.play(music.POWER_UP)

    return lost
```

Game - Run



1

Start the game

Run the start routine

4

Player 2

Receives the first shot and then shoots

2

Choose players

Decide who's starting the game

5

Current player win?

The current player wins when
send_shot returns True

3

Player 1

Shoots first then receives a shot

6

Opponent win?

The current player loses when
receive_shot returns True





Writing the Run method

Continue editing the file **battleship-multiplayer/template_1.py** in the repository

Game - Run



```
def run(self):
    """
    Run the logic of the game.

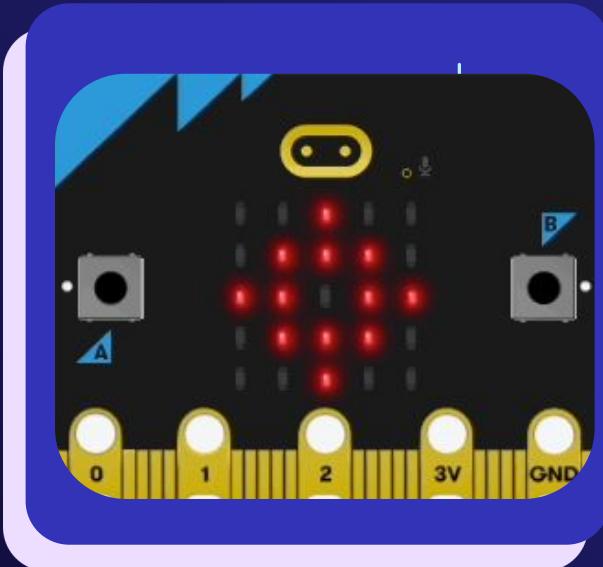
    Player 1 starts the game by sending a shot and receiving a shot from Player 2.
    Player 2 sends a shot and receives a shot from Player 1.

    The game ends when a player wins.

    The player wins by hitting all the opponent's ships before the opponent.
    """
    self.start()
    self.choose_players()

    while True:
        if self.me.player_number == "PLAYER_1":
            if self.send_shot():
                self.winner = self.me.player_number
                break
            if self.receive_shot():
                self.winner = self.opponent.player_number
                break
        elif self.me.player_number == "PLAYER_2":
            if self.receive_shot():
                self.winner = self.opponent.player_number
                break
            if self.send_shot():
                self.winner = self.me.player_number
                break

    self.end(self.winner == self.me.player_number)
```



Try it in the micro:bit

Choose a partner, decide the group and
upload your version of the **template_1.py**
to the micro:bit



<https://python.microbit.org>

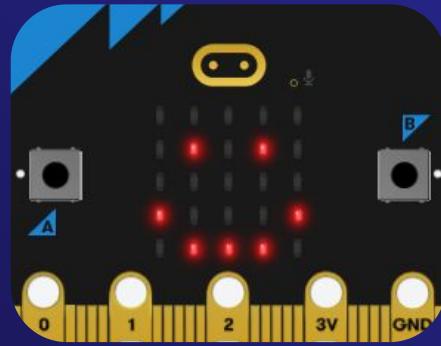
Improvements

- Track the score
- Let the player place the ships manually
- Mark in the player board the sunk ships
- ...
- **What improvements would you make?**

+ ::

Thank you for attending!

▶ :



I hope you enjoyed it :)

+ ▶

:::::

▼ ::

■ ■ ■

For more games, watch my talk at
PyCon US 2023



<https://www.youtube.com/watch?v=teALLngESw0>



@julianaklulo