THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

DEPARTMENT OF
COMPUTER SCIENCE
Te Tari Rorohiko

2024

Contributors

J. Turner

V. Moxham-Bettridge

B. Jones

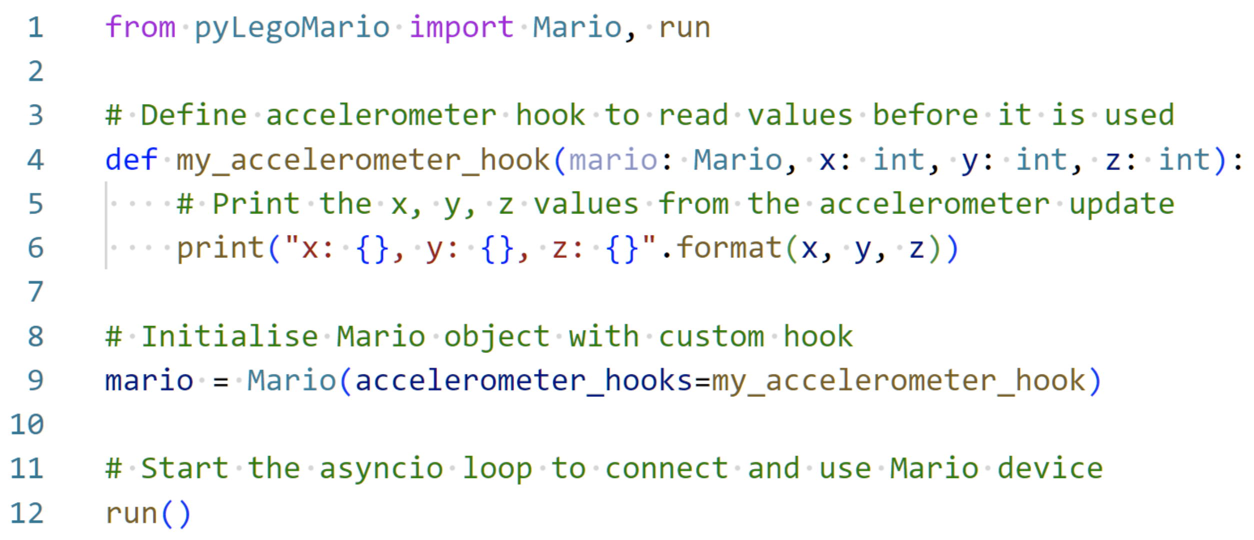
LEGO MARIO: TANGIBLE INTERACTIONS

The purpose of the previous 2 sessions in this series have been to familiarise you with LEGO® Super Mario™ and how it works. In this session we take what you have learnt about the sensors and the data they generate and apply it to a small project!

Refresher

As a reminder of how hooks work, as well as the accelerometer data available, we will create a simple program that converts accelerometer values into a direction.

With the PyLegoMario module, hooks can be added when the Mario object is created, or at a later time. To get started, create a new file and type in the code shown in figure 43. The hook format is defined by the module; it will have the Mario object that trigged the hook, and the accelerometer's x, y, and z values, which we can then use.

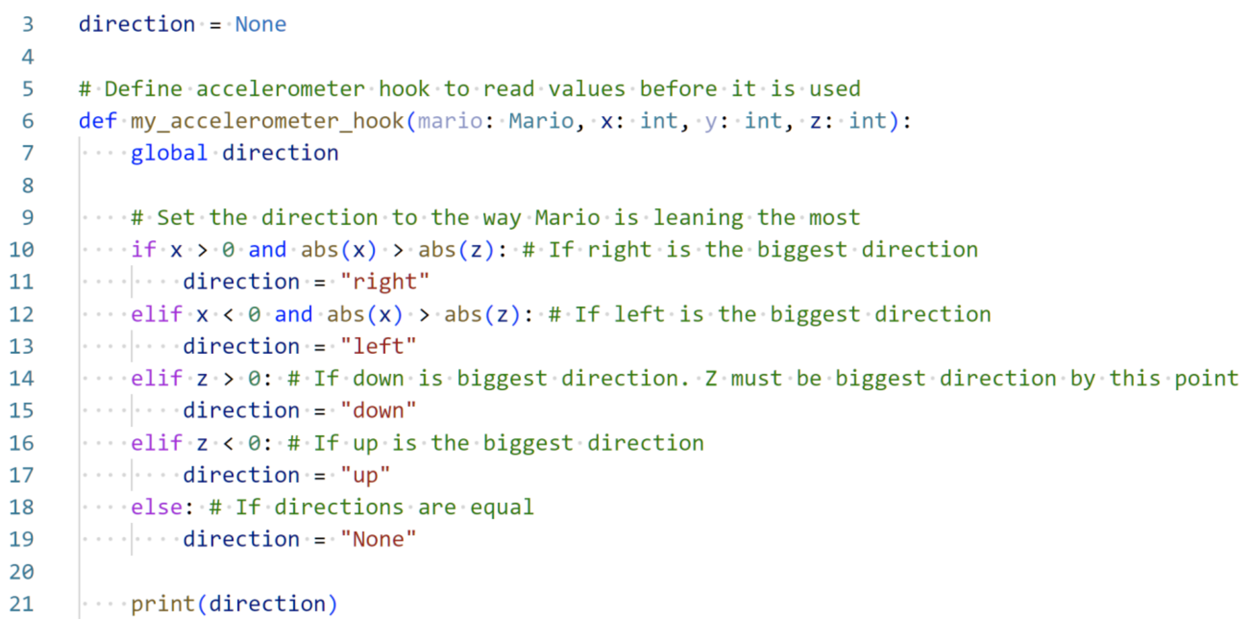


```python
from pyLegoMario import Mario, run

# Define accelerometer hook to read values before it is used
def my_accelerometer_hook(mario: Mario, x: int, y: int, z: int):
    # Print the x, y, z values from the accelerometer update
    print("x: {}, y: {}, z: {}".format(x, y, z))

# Initialise Mario object with custom hook
mario = Mario(accelerometer_hooks=my_accelerometer_hook)

# Start the asyncio loop to connect and use Mario device
run()
```

Figure 43: The Code for Adding a Hook

Now that we have a hook added to a Mario object, we can run this Python file. Remember that every change in the accelerometer's values will cause the hook to run, which can result in many things being printed to your terminal.

This isn't very useful at the moment, so let's turn the x, y, and z values into a direction. Remember the directions from figure 40. If x is negative, the Mario is leaning to the left, and if x is positive, the Mario is leaning to the right. If z is negative, the Mario is leaning forwards, and if z is positive, the Mario is leaning backwards. The y direction is the orientation, so we will ignore this for now. Change the accelerometer hook to match figure 44.



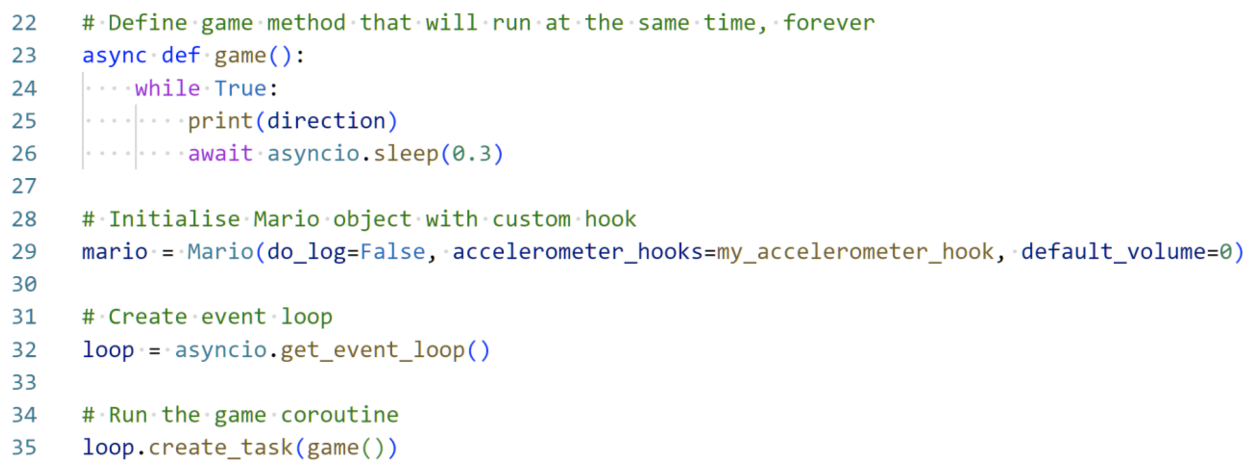
```python
direction = None

# Define accelerometer hook to read values before it is used
def my_accelerometer_hook(mario: Mario, x: int, y: int, z: int):
    global direction

    # Set the direction to the way Mario is leaning the most
    if x > 0 and abs(x) > abs(z): # If right is the biggest direction
        direction = "right"
    elif x < 0 and abs(x) > abs(z): # If left is the biggest direction
        direction = "left"
    elif z > 0: # If down is biggest direction. Z must be biggest direction by this point
        direction = "down"
    elif z < 0: # If up is the biggest direction
        direction = "up"
    else: # If directions are equal
        direction = "None"

    print(direction)
```

Figure 44: Convert Accelerometer Data to Direction

Snake Jr.

We will now create a very basic version of snake, where we will just move a character around the console window.

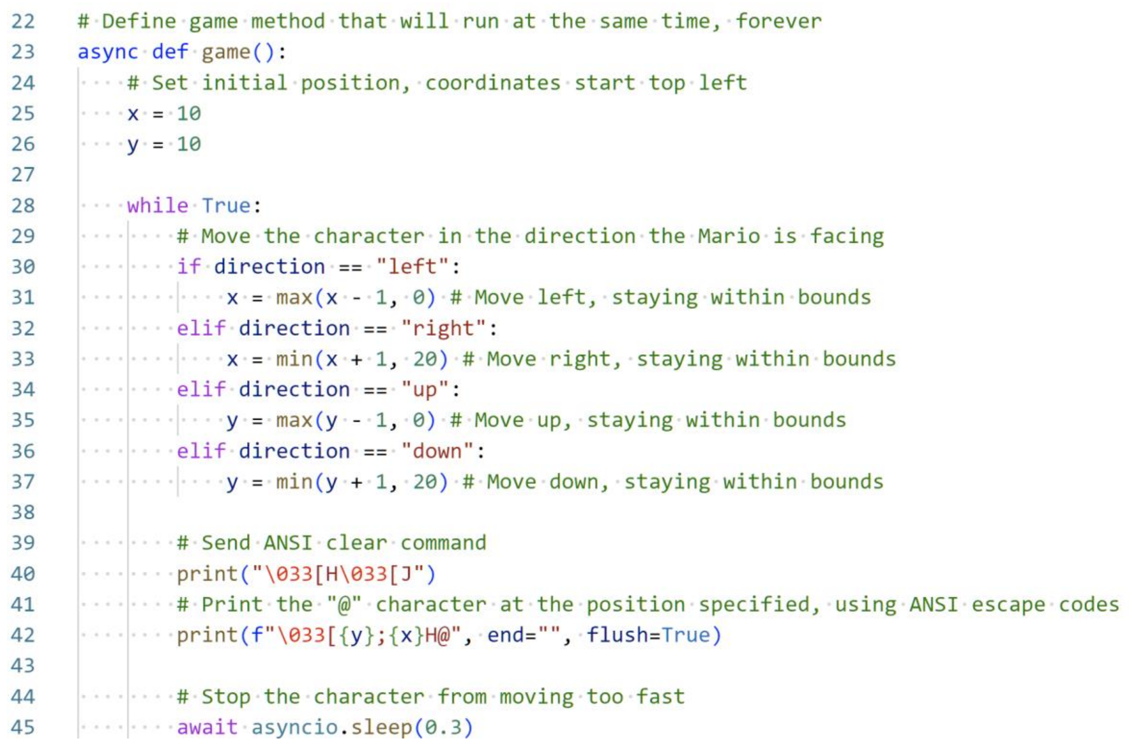
To create a function that runs asynchronously along with the run() loop, we need to define an async game function, and set that as a task to run (see figure 45). We will move the print statement from the hook into our new game function. This is why the direction variable was made globally accessible. You will also notice that direction gets set to the string value "None" instead of the NoneType, and this is because if we try to use it in our game, it can cause issues with it not being able to be used. Don't forget to import the asyncio module!



```python
# Define game method that will run at the same time, forever
async def game():
    while True:
        print(direction)
        await asyncio.sleep(0.3)

# Initialise Mario object with custom hook
mario = Mario(do_log=False, accelerometer_hooks=my_accelerometer_hook, default_volume=0)

# Create event loop
loop = asyncio.get_event_loop()

# Run the game coroutine
loop.create_task(game())
```

Figure 45: Create Asynchronous Loop

Now we are going to move a character around the terminal window. First, we need variables for the x and y position, declared outside of the loop so they retain their value. You could also use an array for this. The next part is updating the position based on the direction but ensuring it does not go out of bounds (stays within the window, otherwise it will break). Then finally, we can print a character at a certain position using ANSI escape codes. These are special commands terminal windows use to do things, like clear the screen, set characters to a certain colour, and more. The complete code is shown in figure 46.



```python
# Define game method that will run at the same time, forever
async def game():
    # Set initial position, coordinates start top left
    x = 10
    y = 10

    while True:
        # Move the character in the direction the Mario is facing
        if direction == "left":
            x = max(x - 1, 0) # Move left, staying within bounds
        elif direction == "right":
            x = min(x + 1, 20) # Move right, staying within bounds
        elif direction == "up":
            y = max(y - 1, 0) # Move up, staying within bounds
        elif direction == "down":
            y = min(y + 1, 20) # Move down, staying within bounds

        # Send ANSI clear command
        print("\033[H\033[J")
        # Print the "@" character at the position specified, using ANSI escape codes
        print(f"\033[{y};{x}H@", end="", flush=True)

        # Stop the character from moving too fast
        await asyncio.sleep(0.3)
```

Figure 46: Character Movement

Today’s Exercise

You now have the core of a simple Snake game! Your goal today is to make any kind of single-player game you like using the knowledge you have gained on using the LEGO® Super Mario™ sensors and hooks. You are free to create whatever you want. Here are some ideas:

- Continue the code to make a full Snake game. You may wish to use the Curses module for printing to the console.
- Create a Pong game.
- Create a Breakout game.
- Create a two-player game that connects 2 Mario devices.
- Use the PyGame library to create a game with a UI.

On the GitHub repository for the PyLegoMario module, there is an example of turning the Mario into a controller for the Mario64 video game, by making it appear as an Xbox 360 controller. There is also an example of using the Mario a controller for the Lego EV3 vehicles. You can also find inspiration on using PyGame in the repository too.

Summary

In this series of sessions, you have learnt about embedded electronics, and programming with sensors asynchronously. Today you put all that knowledge together to create a game, using the Mario devices as controllers. You are now ready to get into the world of IoT, robotics, low power devices, and game consoles!

Useful Resources

∉ PyLegoMario GitHub Repository: https://github.com/Jackomatrus/pyLegoMario
∉ ANSI escape codes: https://gist.github.com/ConnerWill/d4b6c776b509add763e17f9f113fd25b
∉ Curses library: https://docs.python.org/3/howto/curses.html
∉ PyGame libary: https://www.pygame.org/wiki/tutorials