THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

DEPARTMENT OF
COMPUTER SCIENCE
Te Tari Rorohiko

2024

Contributors

J. Turner

V. Moxham-Bettridge

LEGO MARIO: DATA ANALYSIS

Last week we introduced you to LEGO® Super Mario™ and tasked you with building interesting levels to then traverse using the interactive figurine. We continue on with this in today's session but look at how we can communicate with the figurine using the Bluetooth connection.

A Closer Look at the Data Generated

As mentioned in the previous session, the interactive figure (which we may also refer to as 'Lego Mario' from now on) has multiple sensors (see figure 29 in the previous session for a labelled diagram of these): an accelerometer for detecting movement in 3 axes, an optical sensor which can detect the colour of a surface and read the tile barcodes and a method that recognises what pants Lego Mario is currently wearing. Each of these sensors and/or methods generates data which we can analyse to get a better understanding of how Lego Mario works.

PyLegoMario Python Library

In the Useful Resources section of last week's session was a link to a YouTube video titled 'Using Code to Unleash Lego Mario'. The content in the video was inspiration for this week's session so if you haven't watched it already, please watch it at this link: https://www.youtube.com/watch?v=Zi-3scHOR1Q

As explained in the video, Jamin Kauf (the YouTuber) found a project online that allows you to connect to a figurine via Bluetooth using a Python script. Kauf developed this further and created a GUI that displays all of the available data (see figure 35). They also created other projects such as using Lego Mario as a controller for the Super Mario 64 game, a custom soundboard (using the tile barcodes as triggers) and integrating Lego Mario with the PyGame Python library.

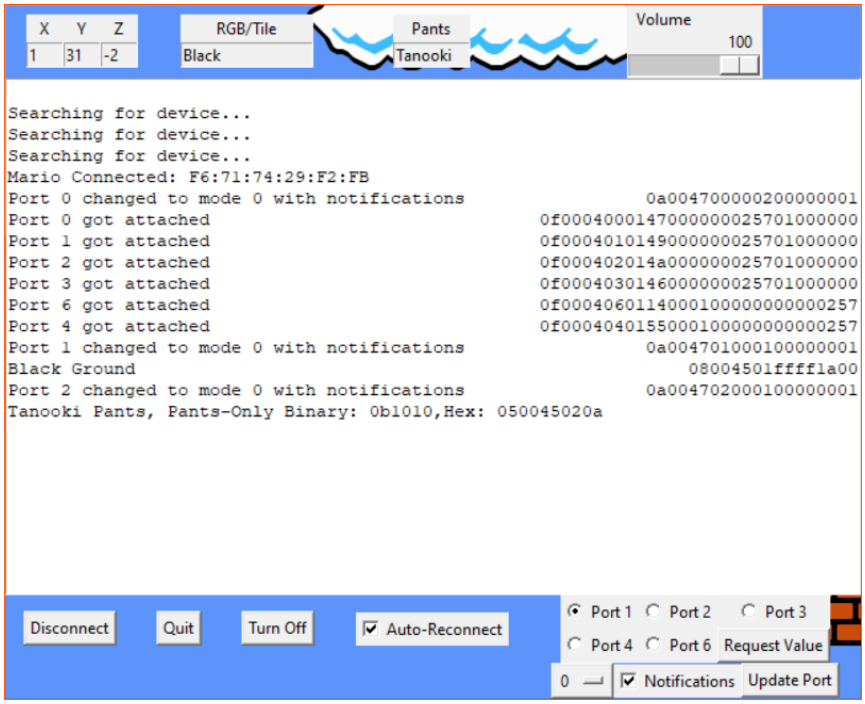

Figure 35: The GUI Included in the pyLegoMario Python Library

Before we can start attempting any of those projects, we first must cover the basics of the pyLegoMario library to see how it works and how to use it.

Open Visual Studio Code and create a new Python file called `lego_mario_basics.py`. Now type in the code shown in figure 36 below.

lego_mario_basics.py > ...

```
1   from pyLegoMario import Mario, MarioWindow, run
2   # Initialize Mario
3   mario = Mario()
4   # Create GUI
5   MarioWindow(mario)
6   # call run() at the end of your program to keep the asyncio loop running
7   run()
```

Figure 36: The Code for Displaying the GUI

We will be using the CLI to run the code so open a terminal now; do you remember the command for running Python scripts?

Once the terminal is open and ready (often denoted by the path displaying followed by a '>'), type `python lego_mario_basics.py` in and press enter (remember to save the file first!). You should then be presented with the screen shown in figure 37.

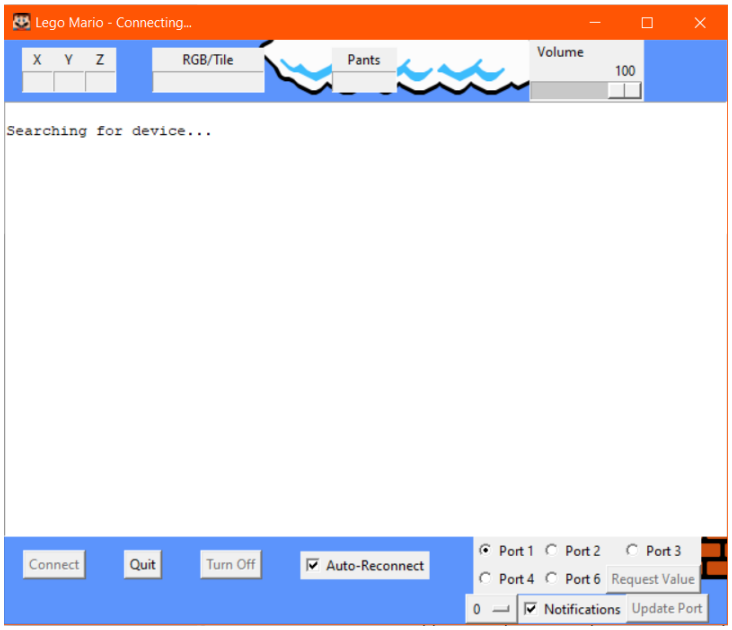

Figure 37: The GUI When First Opened

Notice how the title of the window is 'Lego Mario – Connecting...', this is because we need to pair the figurine with the computer. To do this, turn Lego Mario ON and then press the Bluetooth pair button (you will see dots appear on their belly screen, see figure 38).

Figure 38: What Displays While Bluetooth Pairing

It may take a few attempts so don't worry if it takes a while to connect. When it does connect, you should see the screen shown in figure 39.

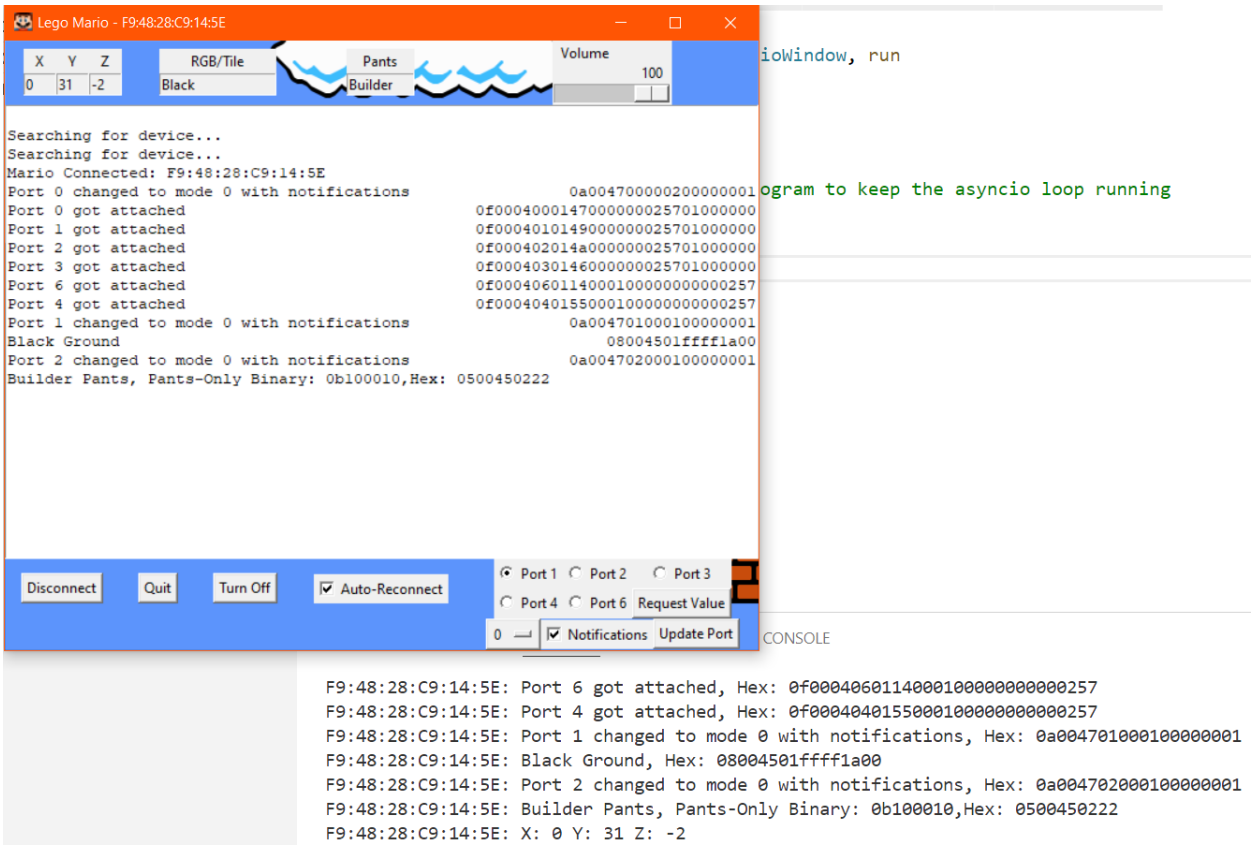

Figure 39: What the GUI and Terminal Display When Paired

Using the worksheet provided, complete the following exercises.

Exercises:

1. Keep Lego Mario (or Luigi or Peach etc.) standing on the table and move him left and right, what happens?
2. Now place Lego Mario in the centre of the worksheet (the centre is denoted by the picture of Lego Mario).
3. Tilt Lego Mario left and then right, what happens? What values change? What axis (or axes) does this movement correspond to? *Write your answers on the worksheet next to the **blue** arrows.*
4. Repeat exercise 3 but tilt forward and backward. Again, what values change? What axis (or axes) does this movement correspond to? *Write your answers on the worksheet next to the **green** arrows.*
5. Complete the remaining arrows on the worksheet.
6. The Y-axis value is currently positive, how would you orient Lego Mario so that the value is negative?
7. Place Lego Mario on 4 different coloured surfaces, what happens? Are all colours recognised?

8. Place Lego Mario on 4 different tile barcodes, what happens? Again, are they all recognised?

Hopefully you noticed that as you attempted the exercises, the values displayed on the GUI and CLI updated. For exercises 3 – 5, their answers can be seen in figure 40 below (note that for the max values, yours may be slightly different – especially for the left and right sides). To get the Y axis value to be negative, Lego Mario needs to be tipped upside down so that his head is closer to the ground than his feet. For the last 2 exercises, you should have noticed that most colours and codes were recognised but not all. This isn't actually a bad thing as it means we can create custom codes and assign custom actions to them that are executed (computer side) when scanned.

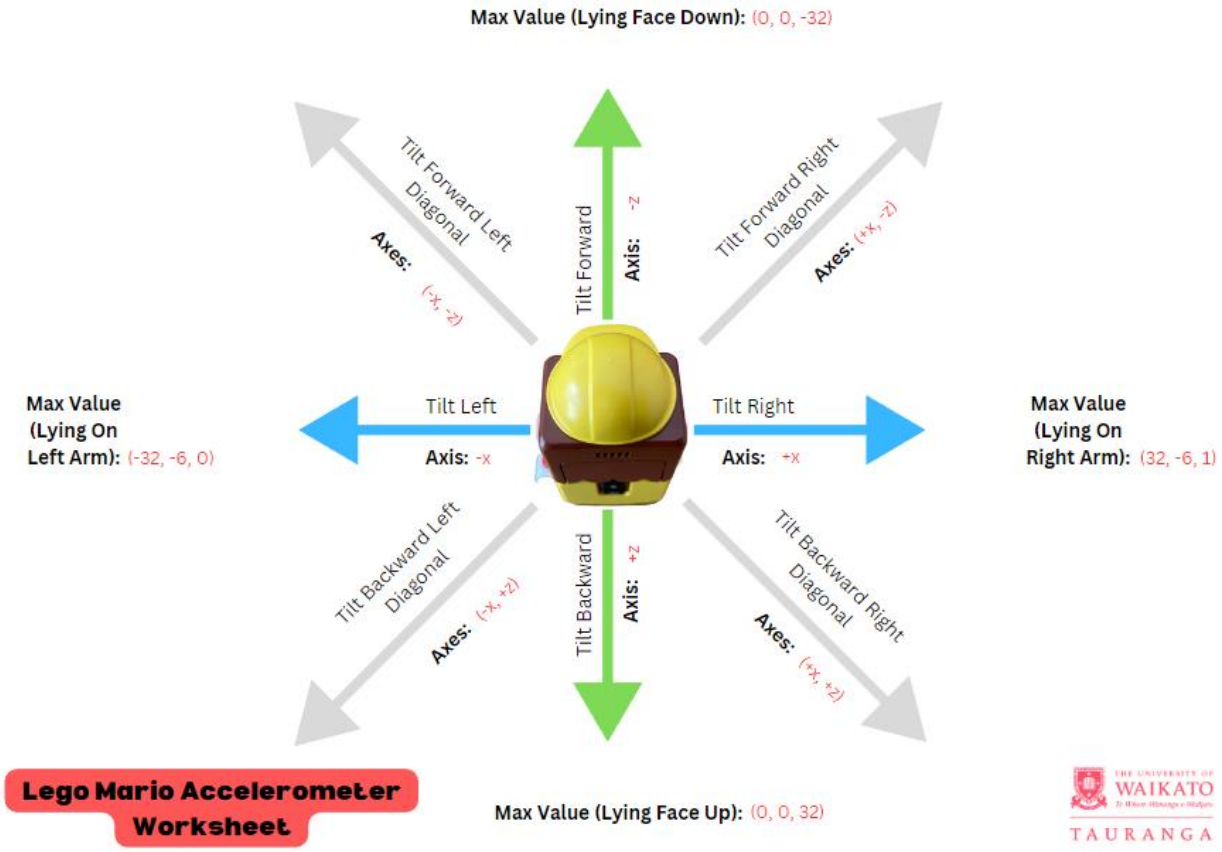

Figure 40: Lego Mario Accelerometer Worksheet Answers

Understanding the Code

The pyLegoMario code has been written asynchronously which means the code doesn't necessarily execute in sequential order (e.g. line 40 after line 3). This is because of the asynchronous nature of the figurine, as, would you be able to predict when an accelerometer

event would occur (i.e. Lego Mario being moved) or when a tile would be scanned? No, well it's very unlikely which is why the code has to be written so that it can accommodate spontaneous occurrences. See figure 41 for the difference between synchronous and asynchronous execution.

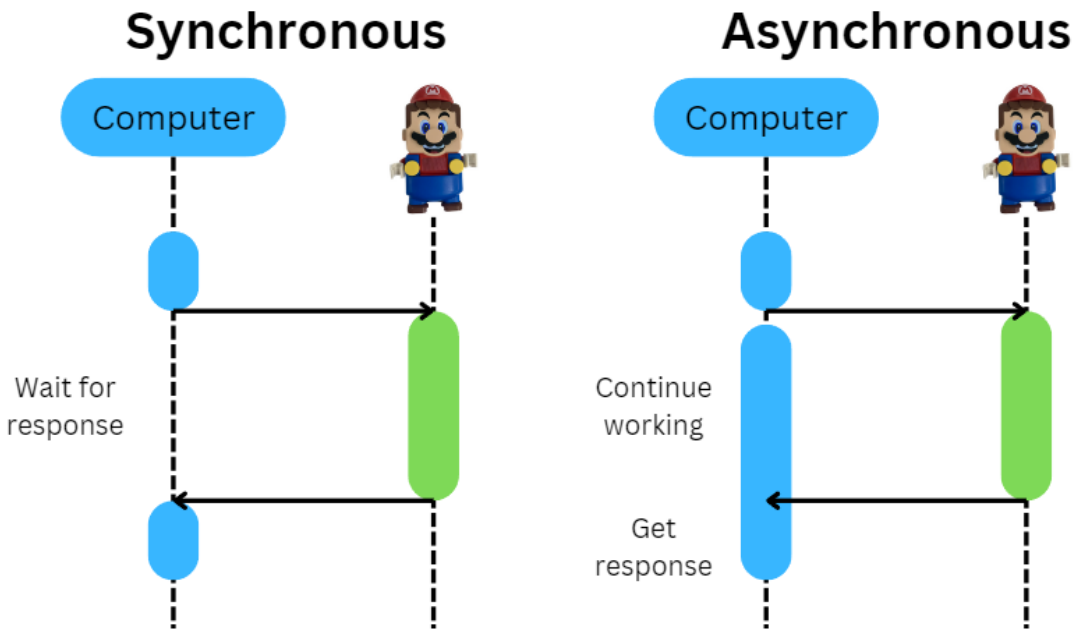

Figure 41: Synchronous vs. Asynchronous Execution Diagram

To see how this is implemented, let's look through the code which can be found at this link: https://github.com/Jackomatrus/pyLegoMario.

Once the page loads, click on the 'pyLegoMario' folder (the one with a blue folder icon) to see the code files. This code has been installed on the computer you're currently using, however where the files are located requires admin permission to access which is why we're viewing them in the GitHub repository.

Attempt the following exercises while looking through the repository. Don't worry if the following leaves you feeling a little unsure about how it works, that's okay, it's more about the underlying concepts (like asynchronous execution) so feel free to ask staff for help if you would like any clarification.

Exercises:

1. Scan through the `ALL_RGB_CODES.json` file, what pieces of data are stored for each one?
2. Read (quickly) through the `mario.py` file, what do you think this file is for? What does it do?

3. Read (quickly) through the `lego_mario_data.py` file, do you see what the constants used in the previous file are for? (i.e. the variables with names in capital letters such as `HEX_TO_COLOUR_TILE` and `HEX_TO_RGB_TILE`).
4. Scan through the `mario_GUI.py` file (this is what creates the GUI window in the program you just run).

For the first exercise, you would have seen that 4 strings and 1 integer constitute an entry in the JSON array. Of those 4 strings, 1 is a hex code which is used along with the integer to determine what colour surface Lego Mario is currently on (see figure 42).

```
298         def _handle_events(self, sender: int, data: bytearray) -> None:
302
303             Args:
304                 sender (int): Only necessary for bleak compatibility
305                 data (bytearray): The data of the notification
306             """
307             hex_data = data.hex()
308             # Port Value
309             if data[2] == 0x45:
310                 # Camera Sensor Data
311                 if data[3] == 0x01:
312                     if data[4] == data[5] == data[6] == data[7] == 0xff:
313                         self.log(f"IDLE?, Hex: {hex_data}")
314                     elif data[4] == data[5] == 0xff:
315                         # Ground Colors
316                         color = HEX_TO_COLOR_TILE.get(
317                             data[6],
318                             f"Unkown Color: {hex(data[6])}")
319                         self.log(f"{color} Ground, Hex: {hex_data}")
320                         self._call_tile_hooks(color)
321                     else:
```

Figure 42: The _handle_events() Function in the mario.py file Showing Surface Colour Determination

The `mario.py` file creates a Mario class which houses the logic for connecting via Bluetooth, retrieving data, calling the method hooks and basic Lego Mario control (such as volume level and turning the figurine off). This is a very important file as most of the other files require a Mario object to function.

With the third exercise you can see the definitions of the constants used in the `mario.py` file. Most of them contain a dictionary where you need a key in order to access the value which is what lines 316 and 317 shown in figure 42 relate to.

We asked you to scan through the `mario_GUI.py` file instead of reading it because we won't be modifying it or using it as a basis in this series.

Adding Hooks

While reading through the contents of the `mario.py` file, it's likely you would have seen code referring to or mentioning 'hooks'. A hook is a mechanism which allows developers to add their own functionality to code that already exists without modifying the original code (Startup-House, n.d.). In our case, it means that we don't have to modify the `mario.py` file to add extra functionality as we can just add to the hook lists (i.e. add to the `_accelerometer_hooks`, `tile_event_hooks`, `_pants_event_hooks` and `_log_event_hooks` lists).

Let's try this by adding the following code to the `lego_mario_basics.py` file we created earlier.

```
04   ...
05   MarioWindow(mario)
06
07   def display_pants_hook(mario: Mario, powerup: str) ->
     None:
08        print(f"I'm wearing {powerup} pants!")
09
10   mario.add_pants_hooks(display_pants_hook)
11
12   #call run() at the end of your program...
13   ...
```

Now save and run your code to see what happens! Note: you will need to take off the pants to trigger the hook (and remember to look in the terminal window).

How this code works is by first creating a function called `display_pants_hook` (on line 7) that takes 2 arguments: the Mario object and a string (the name of the pants). Each argument is followed by a `:` which means the argument is expected to be of the type included after the `:`. For example, 'mario' is an object passed in and is expected to be of type `Mario`, and the argument, 'powerup' is expected to be of type `string`. These are called 'Annotations' which aren't actually enforced by the interpreter as they are primarily used for documentation purposes. The '`-> None`' is the same in that it is an annotation which means that the expected return type for

the function is `None` (i.e. nothing is expected to be returned). On line 8 we have a formatted print statement which prints the string to the terminal and on line 9 we add the function to the `_pants_event_hooks` list which means whenever a pants event occurs, this method (along with the others in the list) will execute.

Also while reading through the code you may have noticed the keywords: `async` and `await`. These inform the interpreter of the code's asynchronous behaviour with `async` primarily used for function definitions (can be applied to `for` and `with` statements too) and `await` used to suspend execution of objects that are 'awaitable' (i.e. can be an async function or an object with an await function).

Today's Exercise

Now that you know how to create a hook, let's make more through the exercises listed below. Feel free to ask staff for assistance if you get stuck.

Exercises:

1. Can you figure out how to make a hook which prints out the surface that Lego Mario is currently standing on? Hint: looking at the `_call_tile_hooks` function in the `mario.py` file may prove to be helpful.
2. Can you make a hook which prints out the values returned from the accelerometer in the form "`(X, Y, Z): (<x>, <y>, <z>)`"? Hint: looking at the `_call_accelerometer_hooks` function in the `mario.py` file may prove to be helpful.
3. Expand the hook you just wrote by including how Lego Mario is moving (i.e. is Lego Mario face down? Face up? Upside down? tilting? And in which direction? Use the worksheet you filled in earlier as a reference).

It's important you understand how Lego Mario works as in the next session (the last one of this series) you will be tasked with a small project which makes use of the various sensors.

Summary

In this session we continued with the LEGO® Super Mario™ series and took a closer look at the data the interactive figurine generates. We used the pyLegoMario Python library to connect via Bluetooth to Lego Mario so that we could read the generated data. Once we knew how to access the data we created hooks which allow us to extend the functionality of code without modifying the original. The hooks we created meant every time a pants, tile or accelerometer event occurred, the corresponding function would be executed. Next week, we take what we have learnt here and use it in a few small (but fun!) projects.

Useful Resources

- W3Schools Python Tutorials: https://www.w3schools.com/python/
- PyLegoMario GitHub Repository: https://github.com/Jackomatrus/pyLegoMario
- Asynchronous Programming: A Beginner's Guide: https://www.bmc.com/blogs/asynchronous-programming/
- What is Async/Await in Python: https://superfastpython.com/async-await-python/
- What is Hook (Programming): https://startup-house.com/inventory/hook
- How annotations are used in Python: https://www.educative.io/answers/how-annotations-are-used-in-python