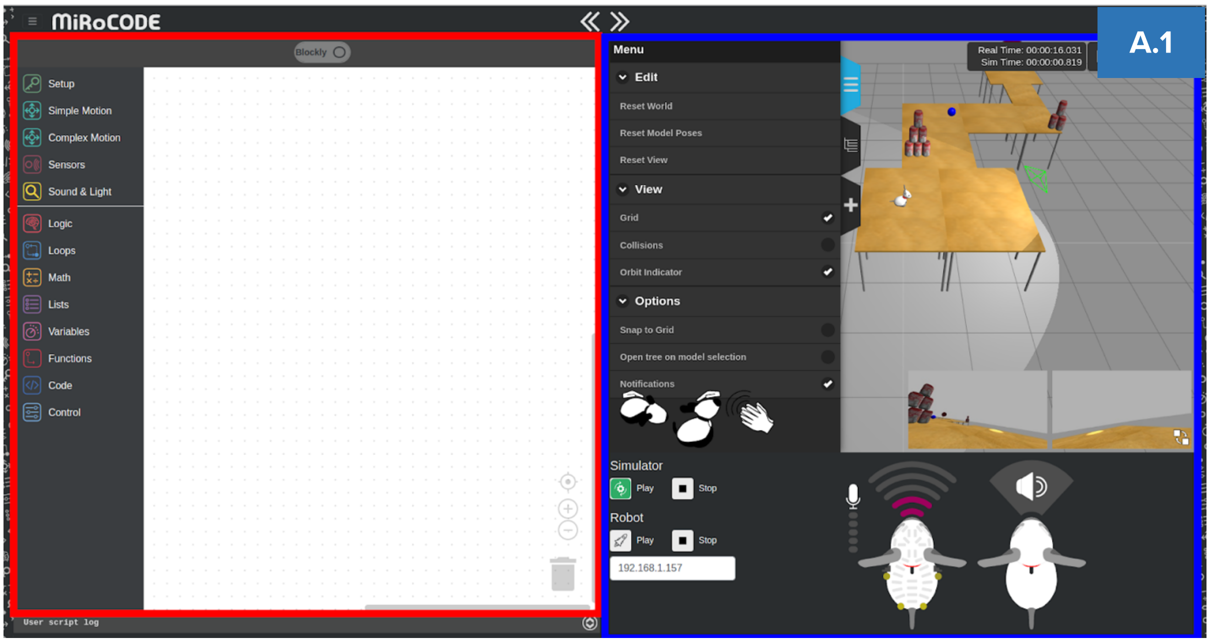
My First MiRoCODE Program

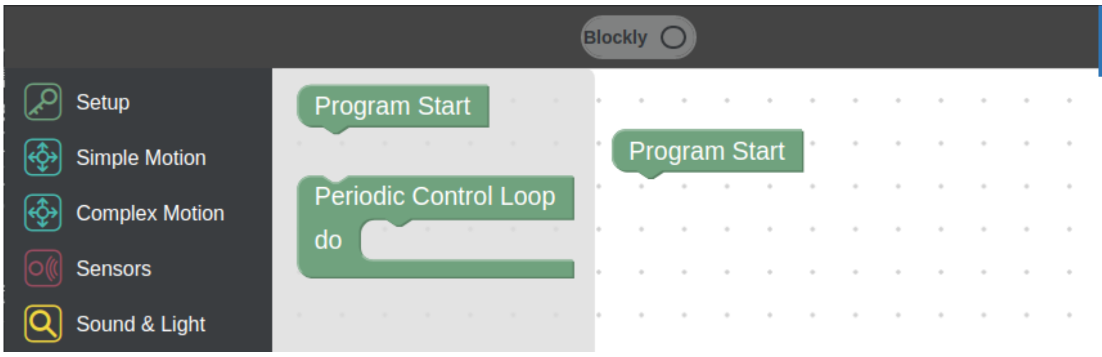
This tutorial will take you through the basics of MiRoCODE. In it we will:

A CREATE A SMALL PROGRAM

1. When you first open MiRoCODE in your browser you will see the following screen. The left-hand side (highlighted in red) is where we will create and edit our programs, the right-hand side (highlighted in blue) is where we can view our simulated MiRo and outputs from the simulator and robot.

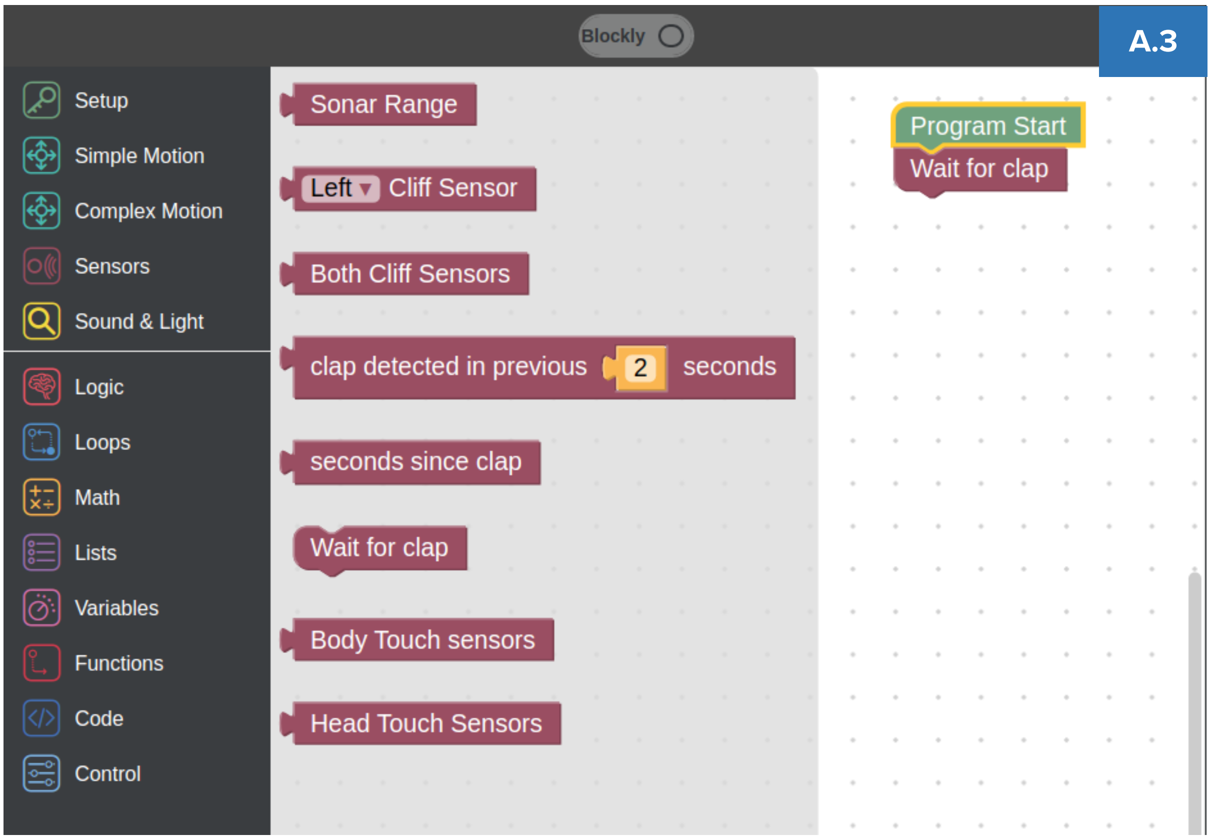

A.1

2. All code must start with a “**Program Start**” block. All blocks are listed in categories on the left-hand side of the screen, they can be dragged and dropped into the workspace. The “**Program Start**” block exists in the *“Setup”* category.



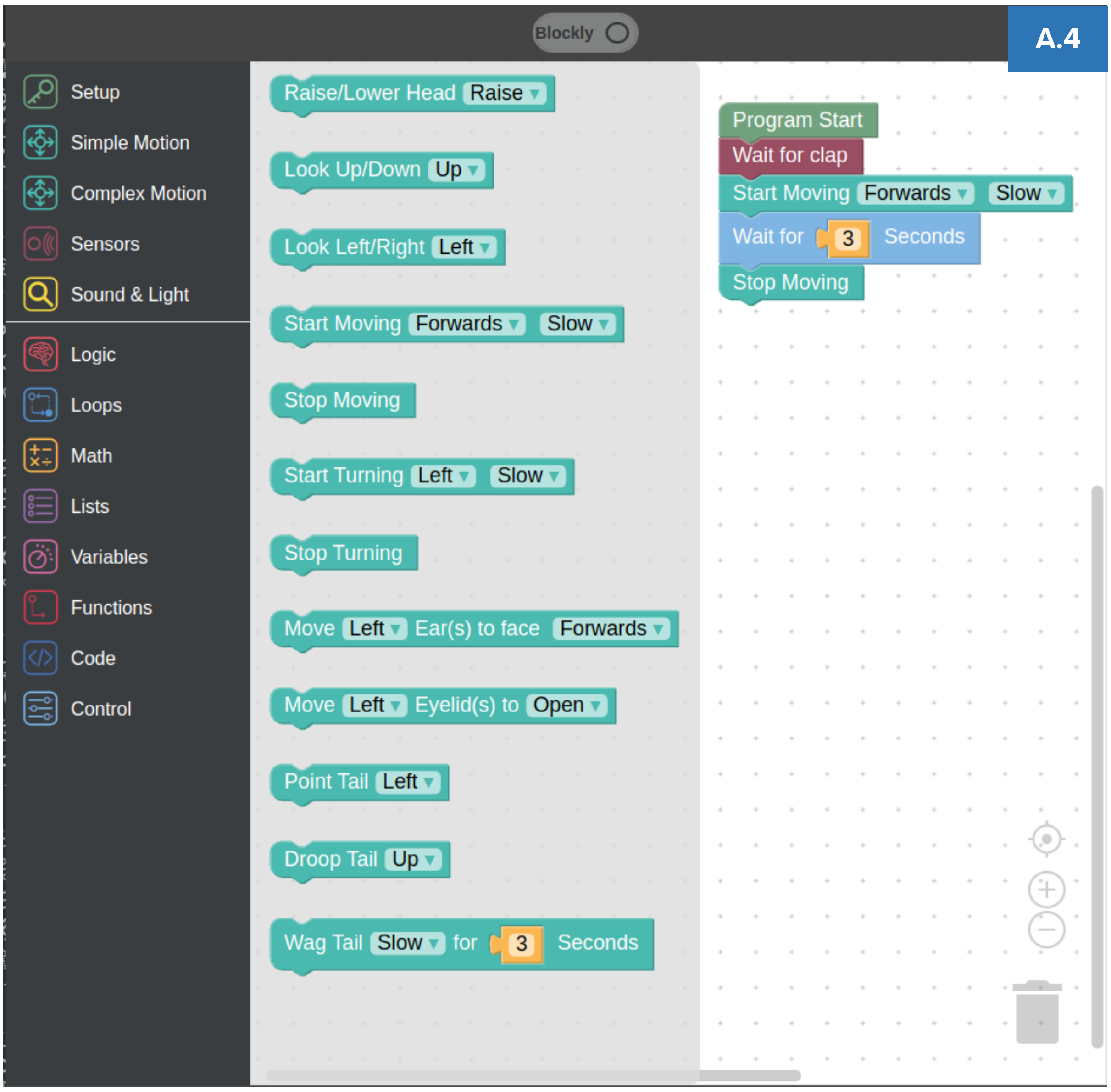
A.2

3. The next block we will add is a “**Wait for clap**”, which is located in the *“Sensors”* category. This is an example of a block which will require interaction through the simulator. Drag and drop it to connect with the bottom of the first block.



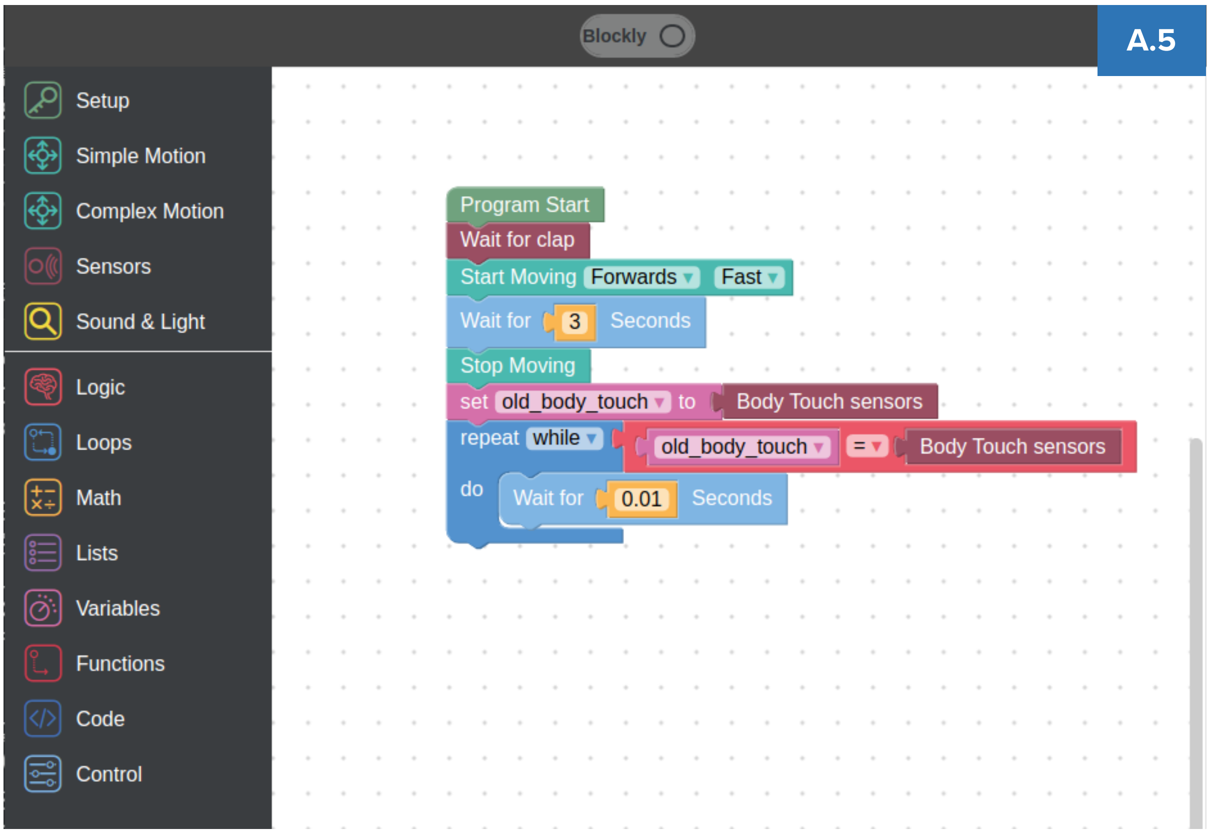
A.3

4. The next thing we will do is add some movement blocks. In order to make MiRo move forwards for 3 seconds we will first add a “**Start moving**” block from the “*Simple Motion*” category, followed by a “**Wait**” block from the “*Control*” category and finally a “**Stop Moving**” block from the “*Simple Motion*” category. The speed and direction in the “**Start Moving**” block can be configured using the dropdowns. The duration of the wait can be altered by editing the value in the number block (click the value and type).



5. Next we will add some more blocks which require interaction through the simulator. The following arrangement of blocks is used to wait for activity on the body touch sensors. We store the value of the sensors using a “**set *variable_name* to**” block from the “*Variables*” category. We then use a “repeat while” block from the Loops category to repeatedly compare the stored value with the latest sensor readings. Inside the loop we use a “wait” block set to 0.01 seconds so that loop rests between iterations. Once the values don’t match, the code will progress out of the loop.

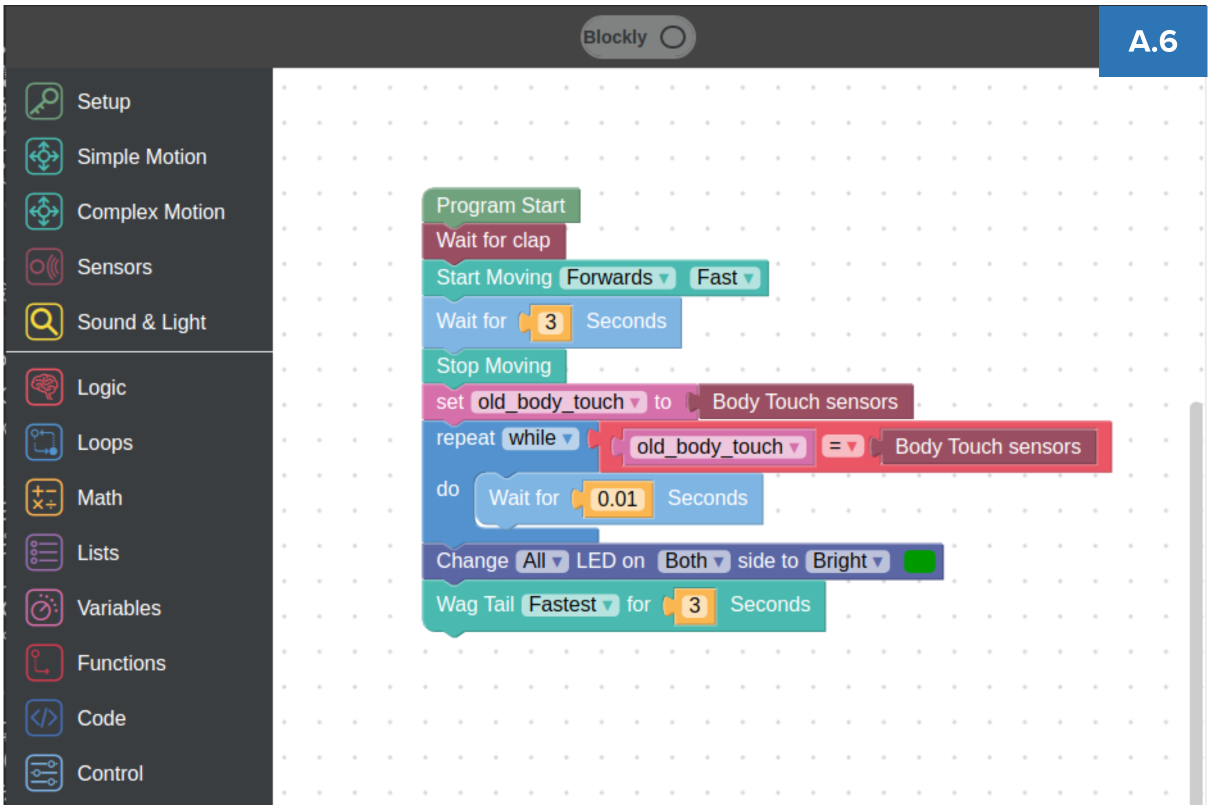
A.5



```
Program Start
Wait for clap
Start Moving Forwards Fast
Wait for 3 Seconds
Stop Moving
set old_body_touch to Body Touch sensors
repeat while old_body_touch = Body Touch sensors
do
    Wait for 0.01 Seconds
```

6. Next we will add an output block which changes the colour of the LEDs from the “*Sound and Light*” category. The colour can be selected by clicking on the coloured square and selecting from the picker. Finally, we will add a “**Wag Tail**” block from the “*Simple Motion*” category.

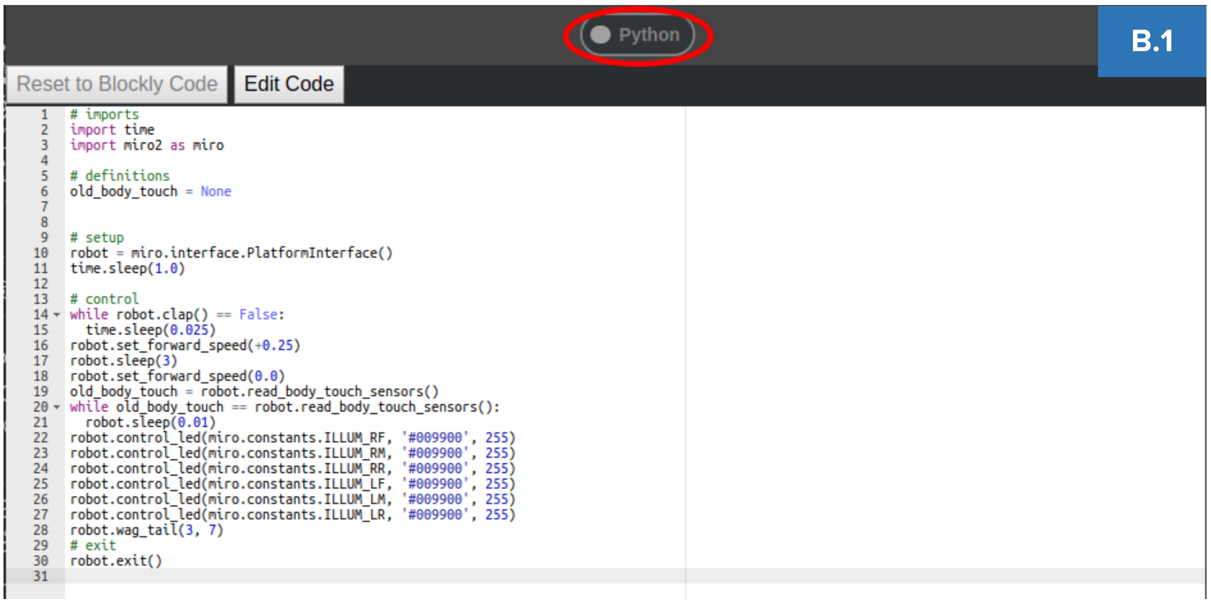
A.6



```
Program Start
Wait for clap
Start Moving Forwards Fast
Wait for 3 Seconds
Stop Moving
set old_body_touch to Body Touch sensors
repeat while old_body_touch = Body Touch sensors
do
    Wait for 0.01 Seconds
Change All LED on Both side to Bright 🟩
Wag Tail Fastest for 3 Seconds
```

B LOOK AT THE GENERATED PYTHON CODE

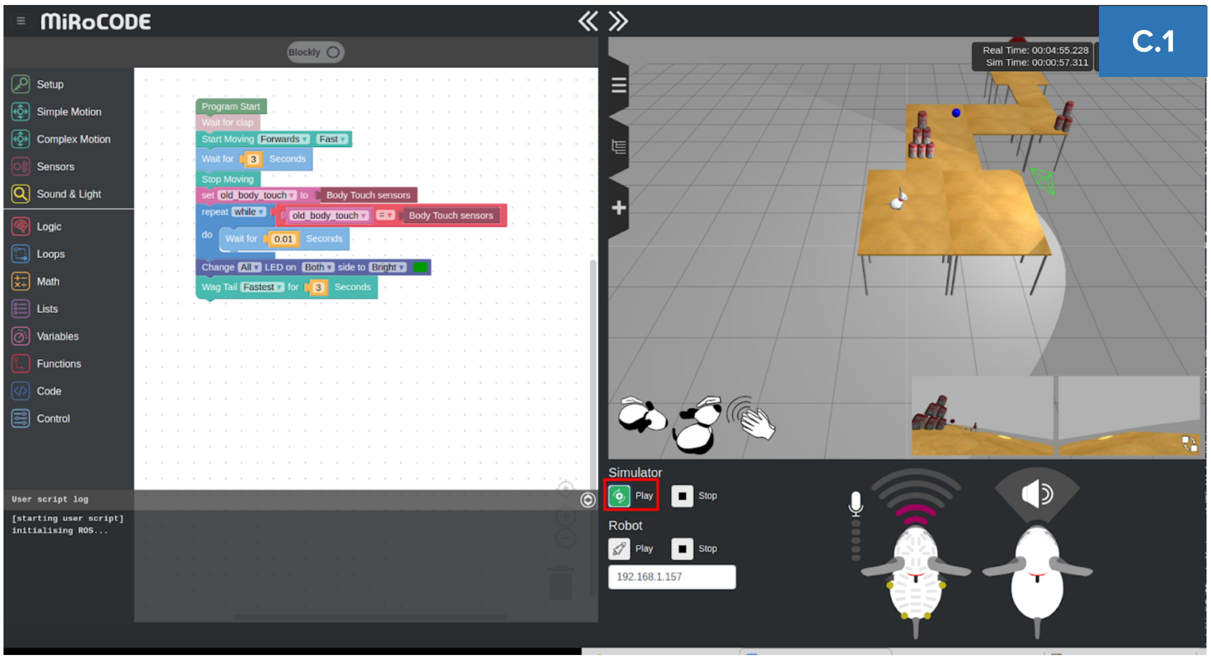
1. We have now created our first runnable program, in order to view the code which will be generated by these blocks you can click the Blockly/Python switch at the top of the editor.



```python
# imports
import time
import miro2 as miro

# definitions
old_body_touch = None


# setup
robot = miro.interface.PlatformInterface()
time.sleep(1.0)

# control
while robot.clap() == False:
  time.sleep(0.025)
robot.set_forward_speed(+0.25)
robot.sleep(3)
robot.set_forward_speed(0.0)
old_body_touch = robot.read_body_touch_sensors()
while old_body_touch == robot.read_body_touch_sensors():
  robot.sleep(0.01)
robot.control_led(miro.constants.ILLUM_RF, '#009900', 255)
robot.control_led(miro.constants.ILLUM_RM, '#009900', 255)
robot.control_led(miro.constants.ILLUM_RR, '#009900', 255)
robot.control_led(miro.constants.ILLUM_LF, '#009900', 255)
robot.control_led(miro.constants.ILLUM_LM, '#009900', 255)
robot.control_led(miro.constants.ILLUM_LR, '#009900', 255)
robot.wag_tail(3, 7)
# exit
robot.exit()
```

C RUN THE PROGRAM IN THE SIMULATOR

1. We are now ready to run our program. First, we will run it in the simulator to ensure that the program behaves as we intended. To do this we simply need to press the "**Play**" button under the Simulator header. Once pressed, we will see the blocks in the program begin to light up as they are executed.

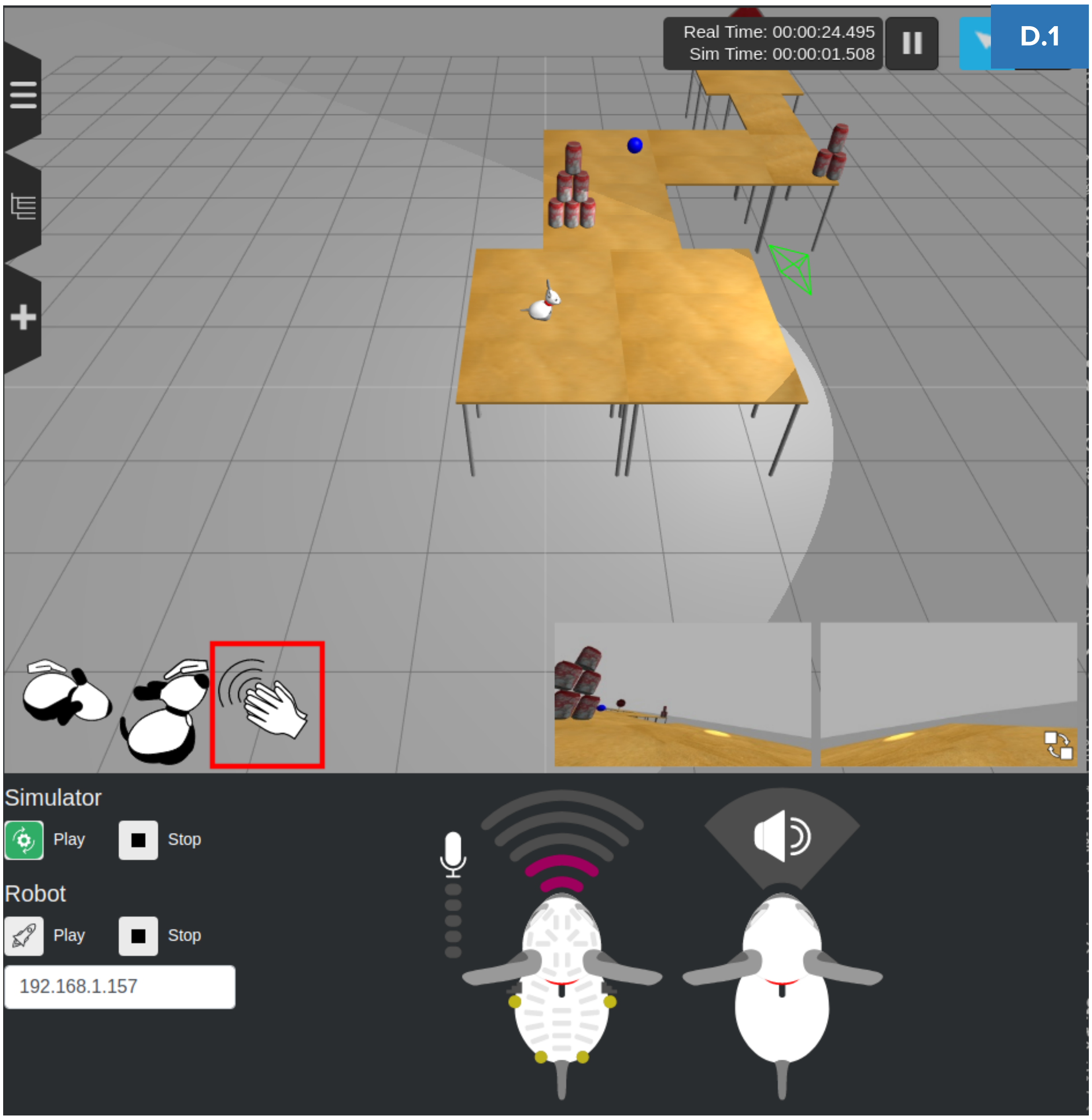


```
User script log
[starting user script]
initialising ROS...
```

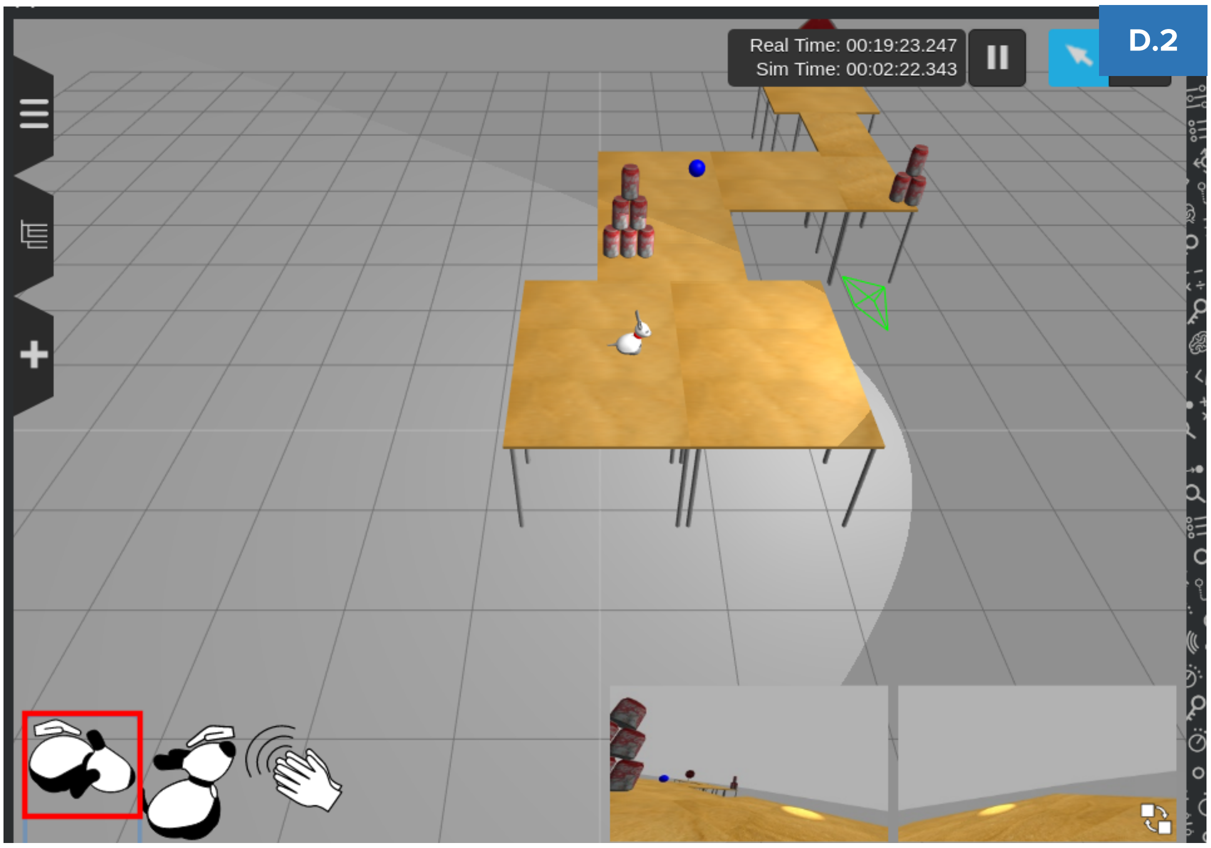
D INTERACT WITH MIRO THROUGH THE SIMULATOR

1. The program is now running on the simulated MiRo. We can see from the highlighting that the code is stuck at the "**Waiting for Clap**" block. The simulated world is silent, but we can simulate the sound of a clap using the interaction buttons located in the bottom left corner of the simulator. Click the **Clap icon** (highlighted in red in D.1). You will see a dialog appear informing you that the button press was successful. After a short while we will see the code progress and MiRo begin to move forward according to the blocks.

D.1

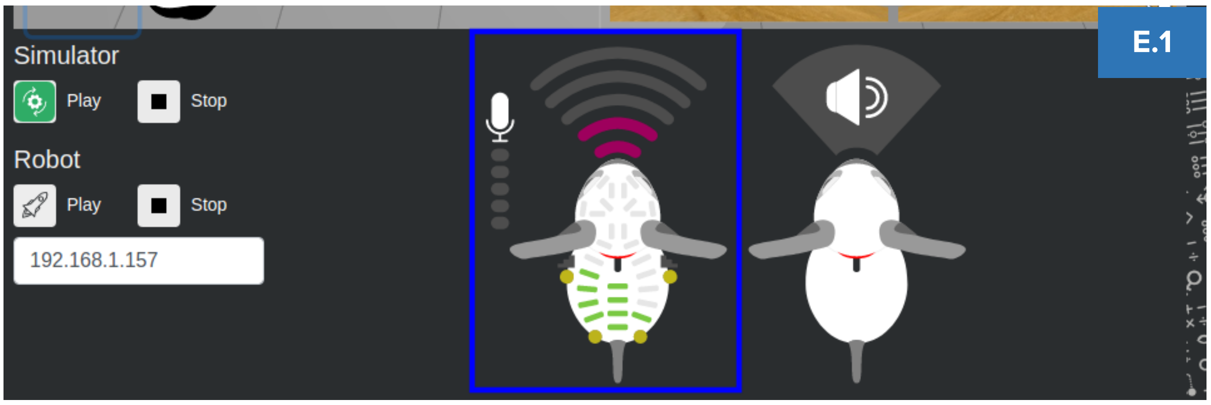


2. Once MiRo has finished the movement, the code will enter the while loop where it is waiting for a change in the state of MiRo's body touch sensors. As we cannot physically touch the simulated MiRo, we need to use another interaction button. Press the **Body Stroke icon** (highlighted in red in D.2).



E LOOK AT MIRO'S INPUTS AND OUTPUTS

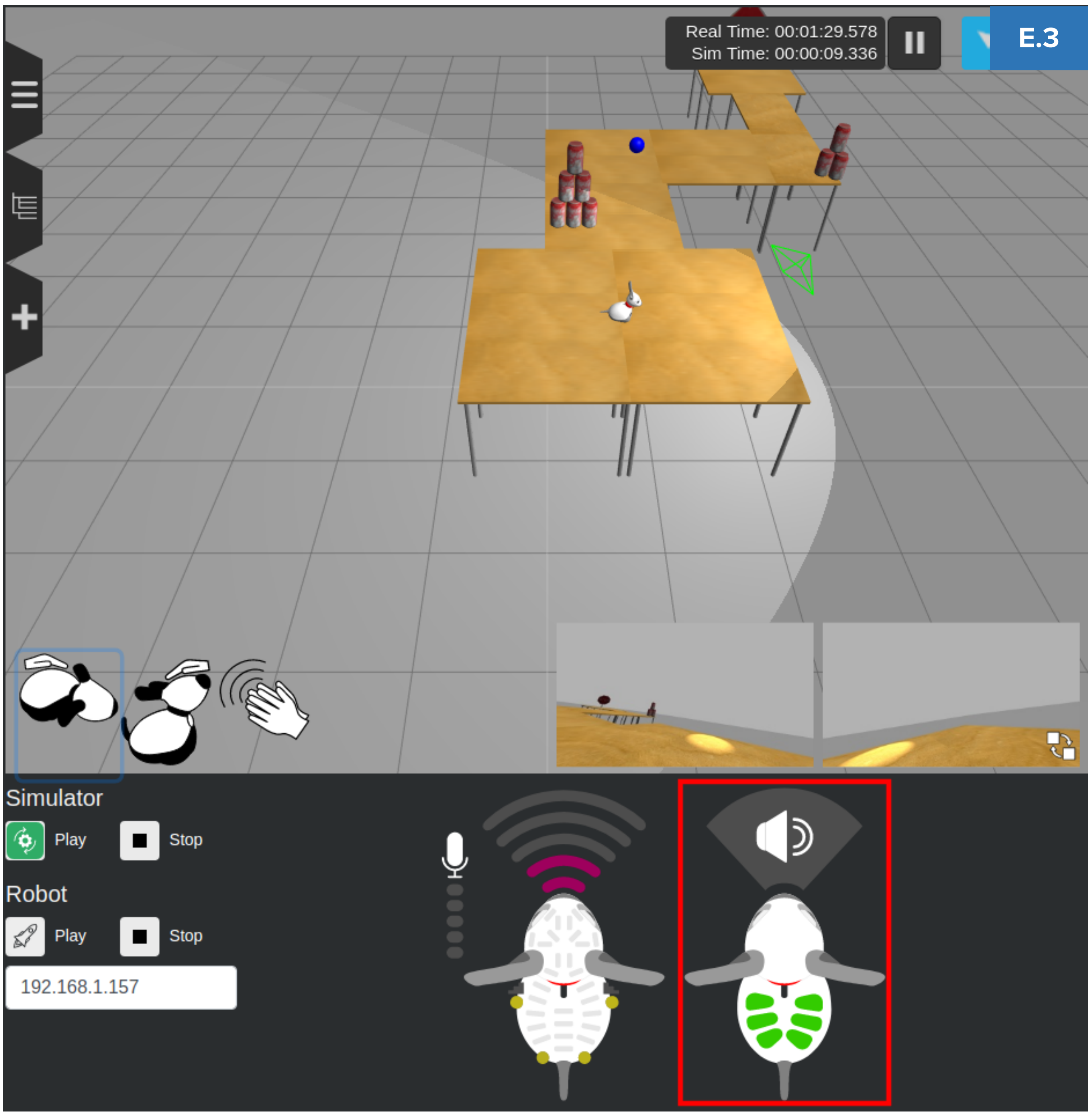
1. In order to visualise what MiRo's sensors are reading, there are two small graphics underneath the simulator. The left graphic represents MiRo's **Sensor Input** (highlighted in blue in E.1) and the right represents MiRo's physical output. After pressing the button, we will see the sensors on the graphical MiRo's back begin to light up green, this tells us that the touch sensors are being triggered, and so the code will progress out of the while loop.



2. Within the graphic, there are several sensory inputs shown. The microphone is on the left and highlights in orange depending on the volume level of any sound detected. The sonar is shown in front of MiRo's nose and highlights in pink, at steps of 0.2m (up to 1m), how far away an object is – in this case, an object or surface is detected 0.4m away from the MiRo. There are 28 touch sensors shown over the head and body which are highlighted in green when they are activated/touched. The four light sensors are shown

as circles and these will increase from grey to bright yellow depending on the light input. The two cliff sensors are shown as steps and highlight in blue if they are activated.

3. The next block will command the colour and brightness of MiRo's LEDs. The simulator does not contain MiRo's lights so these are represented on the right hand graphic below the simulator. The right graphic represents MiRo's **Physical Output** (highlighted in red in E.2). In this case we can see that the LEDs have changed to a bright green colour.



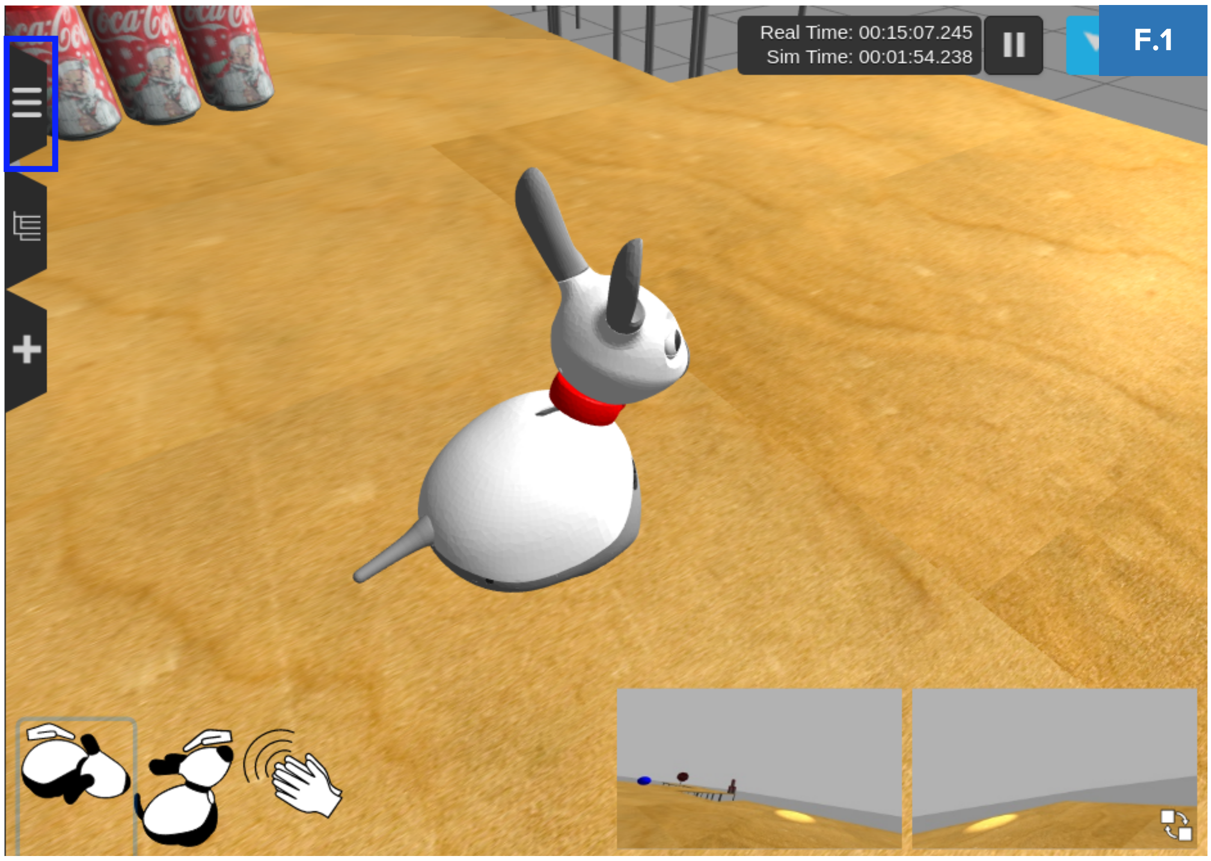
4. Apart from the six individual LED colour outputs shown, the other physical output shown here is the speaker which will be highlighted in orange if the speaker is being utilised.

5. When a physical MiRo robot is connected to the program, these graphics will display the physical MiRo's sensory inputs and physical outputs.

F

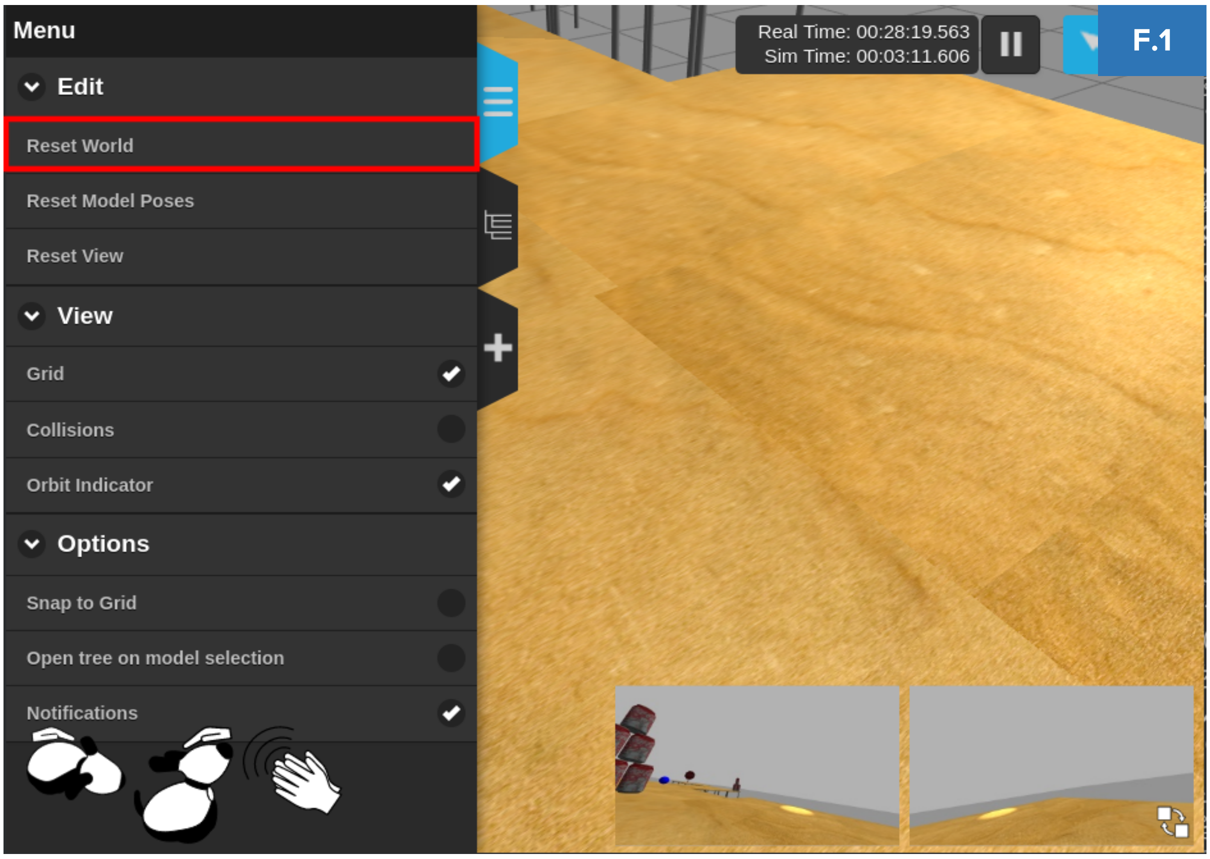
ADJUST SIMULATOR VIEW

1. Finally, the wag tail block is executed. The standard view on MiRoSIM is wide enough that the whole world can be viewed, this may make it difficult to see the movement of the tail. It is possible to move, zoom, and rotate our view of the world using the mouse. Scroll the mouse wheel to zoom, click and drag the mouse wheel to rotate, and left click and drag to pan. We will use the functions to get a better view of MiRo's tail.



F.1

2. It may also be the case that we want to add further code to our program and rerun it, this can be achieved by simply adding more blocks and clicking play again. Before running our new code, we should reset the simulator so that MiRo returns to the start position. To do this first open the **Menu** by clicking on the menu tab in the top left of the simulator (highlighted in blue in F.1), then press the "**Reset World**" button (highlighted in red in F.2).



G EDIT THE GENERATED PYTHON CODE

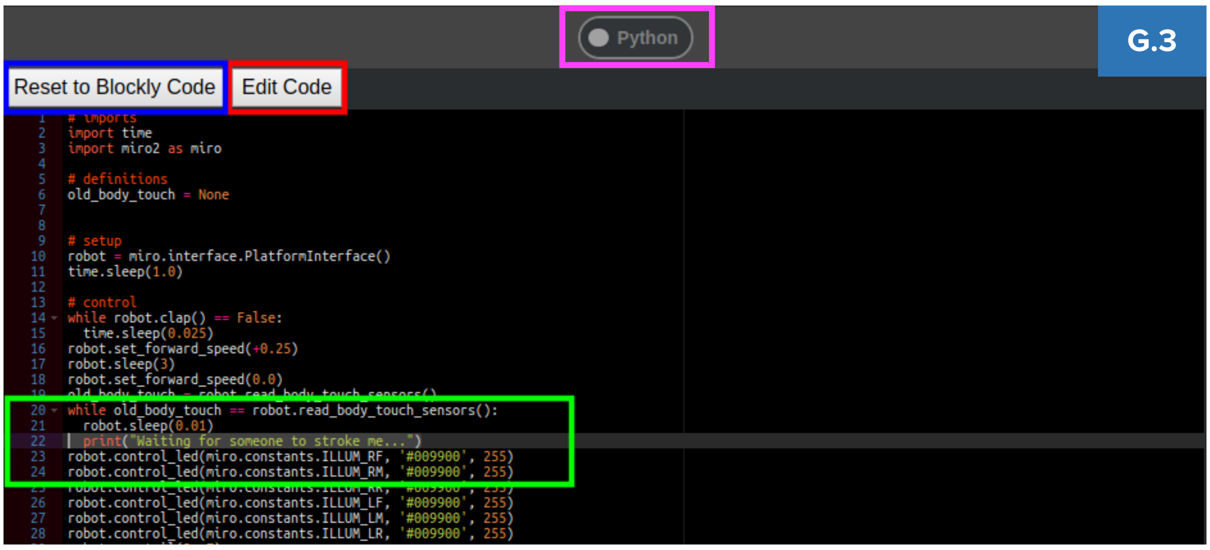
1. In some cases, it may be that we want to add some code which isn't represented by the blocks (or just test ourselves). For this reason, it is also possible to add your own code.

2. In this example we will use the edit code feature to add an extra line of code to our program. First, switch the editor back to **Python view** (highlighted in pink in G.3) and then click the "**Edit Code**" (highlighted in red in G.3) button in the toolbar, this will bring up a blank document, if you are feeling brave you could use this space to write your own program from scratch. In this case we want to edit our existing code.

3. Clicking the "**Reset to Blockly Code**" button (highlighted in blue in G.3) will fill in our document with the code from our blocks which we are now able to edit. In this case we will add some feedback to our while loop so that we can see why MiRo isn't doing anything. On Line 22 we will add the following code (highlighted in green in G.3):

   print("Waiting for someone to stroke me…")

   This will give us some feedback in the User Script Log when we come to run the program.

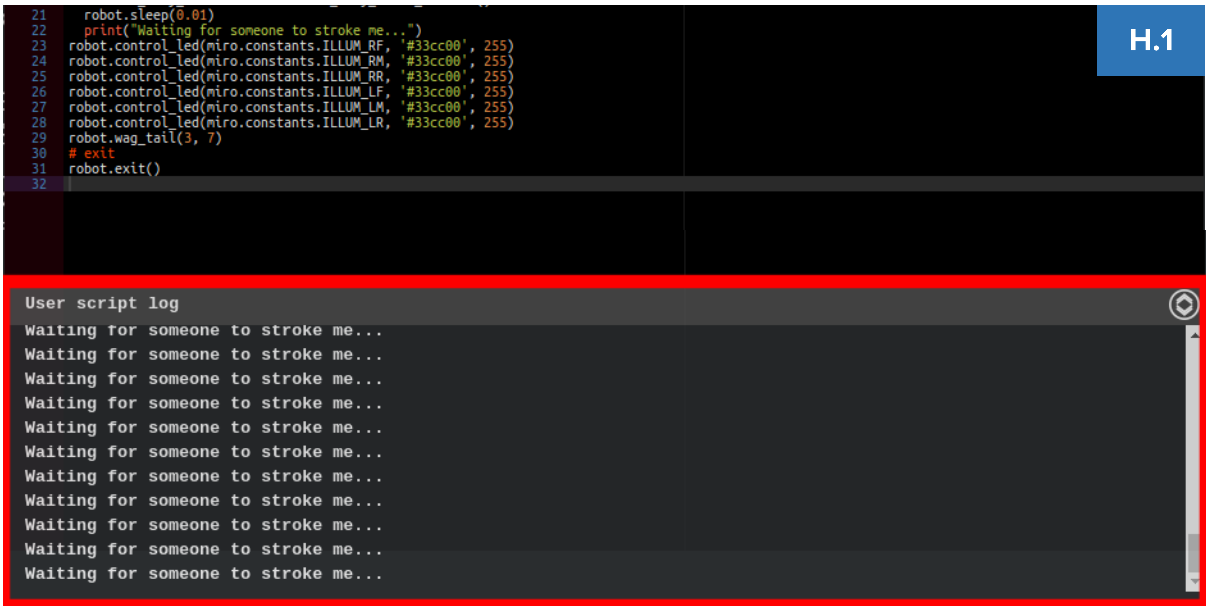


```
# imports
import time
import miro2 as miro

# definitions
old_body_touch = None


# setup
robot = miro.interface.PlatformInterface()
time.sleep(1.0)

# control
while robot.clap() == False:
  time.sleep(0.025)
robot.set_forward_speed(+0.25)
robot.sleep(3)
robot.set_forward_speed(0.0)
old_body_touch = robot.read_body_touch_sensors()
while old_body_touch == robot.read_body_touch_sensors():
  robot.sleep(0.01)
  print("Waiting for someone to stroke me...")
robot.control_led(miro.constants.ILLUM_RF, '#009900', 255)
robot.control_led(miro.constants.ILLUM_RM, '#009900', 255)
robot.control_led(miro.constants.ILLUM_RR, '#009900', 255)
robot.control_led(miro.constants.ILLUM_LF, '#009900', 255)
robot.control_led(miro.constants.ILLUM_LM, '#009900', 255)
robot.control_led(miro.constants.ILLUM_LR, '#009900', 255)
```

4. To run our new edited code, we simply need to press the simulator play button again. It is important to note that the code which runs is dependent on what code you are viewing in the simulator. If you are in the **Blockly** view or standard **Python** view, this is the code that will be run. If you are in the **Edited Code** view, it will run this code.

H LOOK AT THE USER SCRIPT LOG

1. When any code runs, the User Script Log which sits at the bottom of the editor will expand. The User Script Log is used to relay information about the execution of your code, it will also display error messages and in this case our print statements. Once the code reaches the while loop, we will see our messages printed repeatedly until we exit the loop. The User Script Log will appear in either **Blockly** view or **Python** view.



```
  robot.sleep(0.01)
  print("Waiting for someone to stroke me...")
robot.control_led(miro.constants.ILLUM_RF, '#33cc00', 255)
robot.control_led(miro.constants.ILLUM_RM, '#33cc00', 255)
robot.control_led(miro.constants.ILLUM_RR, '#33cc00', 255)
robot.control_led(miro.constants.ILLUM_LF, '#33cc00', 255)
robot.control_led(miro.constants.ILLUM_LM, '#33cc00', 255)
robot.control_led(miro.constants.ILLUM_LR, '#33cc00', 255)
robot.wag_tail(3, 7)
# exit
robot.exit()
```

```
User script log
Waiting for someone to stroke me...
Waiting for someone to stroke me...
Waiting for someone to stroke me...
Waiting for someone to stroke me...
Waiting for someone to stroke me...
Waiting for someone to stroke me...
Waiting for someone to stroke me...
Waiting for someone to stroke me...
Waiting for someone to stroke me...
Waiting for someone to stroke me...
Waiting for someone to stroke me...
```

2. Once we are happy that our code behaves as intended, we can move on to running it on a physical MiRo-E. For instructions on how to achieve this, please see the guide "***MiRoCODE Run on Physical MiRo-E***".