STEMKIT 4SCHOOLS

# MAKING A DETONATOR IN MINECRAFT

## LESSON PLAN 1

Co-funded by the Erasmus+ Programme of the European Union

This project has been funded with support from the European Commission.

# Table of Contents

# 1. Making a Detonator in Minecraft

## 1.1 General information

### 1.1.1 Short description

In this lesson plan, we will build a big red detonator button. When we press the button there will be a countdown on a 7-segment display so the player has time to run away from the explosion, and then a huge crater will appear at the point where our player was standing when we pressed the button. This will be quite handy to our adventure in Minecraft as a quick way to clear space as we move around and build constructions. This lesson plan introduces another type of GPIO function, called an input, that senses when a button is pressed.

### 1.1.2 Learning objectives

The main learning objectives of this lesson plan are:

- utilizing a 7-segment display to interact with Minecraft Pi.
- familiarizing with circuits and programs to interact with GPIO pins of Raspberry Pi
- understanding basic structures of Python programming language (namely, use in program the syntax, purpose and function of *while loop*, *if statements*, *definition of function*, etc.

### 1.1.3 Links to curriculum

The domains, subdomains, subjects/topics that this lesson plan can be linked to are:

- Computer Science/Informatics: processing unit and peripherals, interfaces, programming language and main structures, coding
- Technology: electronics, open-source hardware and software, sensors, digital signal, circuits, single board computers

### 1.1.4 Materials required

For this lesson plan (and for each student group) besides the STEMKIT console and its Raspberry Pi we'll need:

- 1 x 40P T-Cobbler Plus GPIO Breakout Board
- 1 x GPIO 40P Rainbow Ribbon Cable
- 1 x 7-segment display
- 11 x Male-to-Male jumper wires
- 1 x Breadboard
- 1 x Push button
- 1 x 10K Ohm resistor
- 8 x 220 Ohm resistors

### 1.1.5 Duration

The duration of this lesson plan is estimated to be about 80-100 mins, i.e., two classroom hours

## 1.2 Lesson plan

The lesson plan is divided in four phases, which are introduction, preparation, execution and conclusion.

### 1.2.1 Introduction

In this project, we will wire up a button that we can press to set off a detonator. For realistic purposes we choose a big red button, so it resembles an actual detonator button that we see in movies and video games.

The button we are going to use is a non-locking, push-to-make button. This means that the button does not normally make an electrical circuit and it is only when we press it that a completed circuit is created. The button does not stick down and so as soon as we remove our finger, the circuit is broken again.

To watch a tutorial video on how to build and play the detonator game, visit the following website https://www.wiley.com/WileyCDA/Section/id-823690.html and choose the Adventure 5 video.

### 1.2.2 Preparation

First thing we need to do is to make a circuit and connect our sensor to GPIO pins of our Raspberry Pi. Before we proceed, we turn off our Raspberry Pi and unplug it. For our circuit, we will need our 7-segment display, a breadboard, resistors, jumper wires and a push button. In addition, we use a 40P T-Cobbler GPIO Breakout Board and a 40P Rainbow Ribbon Cable to help us set up the circuit on the breadboard instead of connecting jumper wires directly on the Raspberry. The complete circuit is in the Figure 1 that follows.

Here are 4 steps to setup our circuit:

1. We push the button into some spare space on the breadboard. The button that we used has four pins on it, and it fits nicely across the midway point of the breadboard. Pressing the button joins the pins on the left-hand side together with pins on the right-hand side, as long as we have fitted the button with the pins showing at the top and bottom.

2. We need a resistor to pull up the button input to 3.3 volts. If we don't fit a resistor, our button will generate spurious presses and our detonator will keep going off all the time! Fit the 10K pull-up resistor (with the coloured bands: brown, black, orange) so that one leg is

connected to the left pin of the button, and the other leg is connected to the 3.3-volt power rail at the top of the breadboard.

3. We run a wire from the junction between the resistor leg and the button, and we connect the other end of the wire to GPIO number 4.

4. We run a wire from the bottom right pin of the button and connect the other end of the wire to the 0-volt power rail at the bottom of the breadboard. All buttons are normally wired up to computers in this way.

Here is some more information about the circuit: When we wired our button, we fitted a resistor to it. This resistor is called a pull-up resistor because it pulls the voltage on the pin of the button “up” to 3.3 volts. When the button is not pressed (and no connection made), the wire that connects the GPIO to this point in the circuit is now at 3.3 volts, which is the voltage the GPIO pin will interpret as a digital “1”. When we press the button, the internal mechanics of the button make a circuit between the left-hand and right-hand pins, and the GPIO wire is “pulled hard down” to 0 volts via the 0-volt power rail. In this condition, the GPIO pin will interpret this as a digital “0”. The resistor is quite a high value (10K ohms, which is 10,000 Ohms) because otherwise the button will short together the 3.3 Volt power rail and the 0-volt power rail, and our Raspberry might reboot! This is not recommended, as it can in some cases cause damage to our hardware. 10K is a short-hand form that engineers use to refer to a 10,000 Ohm resistor.

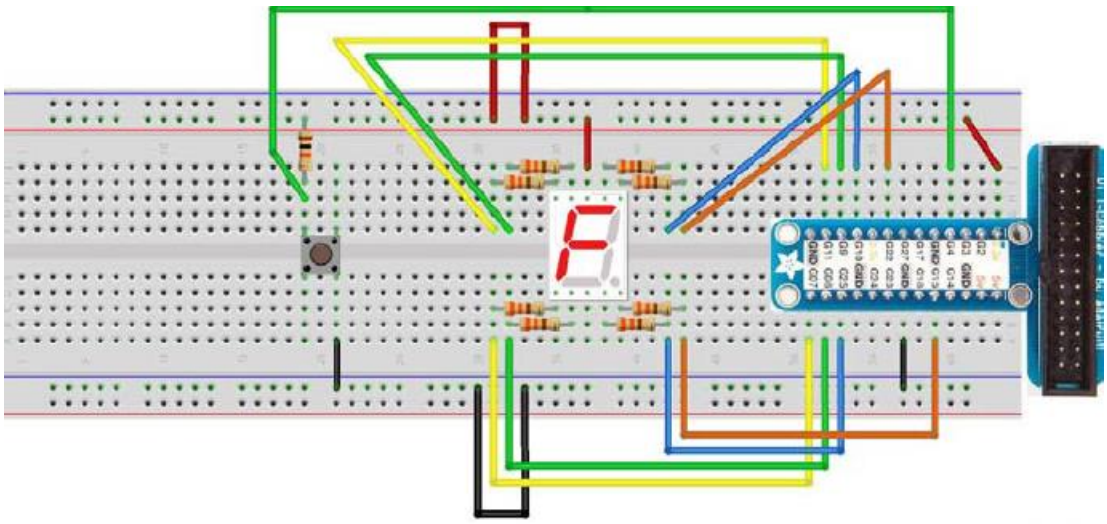

FIGURE 1 CIRCUIT DIAGRAM OF A BUTTON CONNECTED TO RASPBERRY PI

## 1.2.3 Execution

Now that we have wires our button, we are ready to write the detonator program in Python. This program will monitor the state of the button each time round the game loop, and when it senses a digital “0” on the GPIO pin (in other words, when the button is pressed), it will count from 5 down to 0 on the 7-segment display and then blow a massive crate in the Minecraft world.

We start a new program by choosing File → New File and save it as detonator.py.

Then we start writing our program by importing the necessary modules:

```
import mcpi.minecraft as minecraft
import mcpi.block as block
import time
import anyio.seg7 as display
```

Then we configure the GPIO setting for our computer:

```
import RPi.GPIO as GPIO
BUTTON = 4
LED_PINS = [10, 22, 25, 8, 7, 9, 11, 15] #the order is important
```

We setup the GPIO for the button so it is as an input, and configure the display GPIOs:

```
GPIO.setmode(GPIO.BCM)
GPIO.setup(BUTTON, GPIO.IN)
ON = False # False=common-anode, True=common-cathode
display.setup(GPIO, LED_PINS, ON)
```

We then connect to the Minecraft game:

```
mc = minecraft.Minecraft.create()
```

We then write a function that drops a block of TNT to mark where the bomb will go off, counts down to zero, then blows a big crater where the TNT block was. The following code builds the TNT block slightly to the side of the player, so it does not land on top of him! Note that the crater is 20 blocks in size (10 to the left and 10 to the right of your player) and this size is set by the calculations done in `setBlocks()` here:

```
def bomb(x, y, z):
      mc.setBlock(x+1, y, z+1, block.TNT.id)
      for t in range(6):
            display.write(str(5-t))
            time.sleep(1)

mc.postToChat("BANG!")
mc.setBlocks(x-10, y-5, z-10, x+10, y+10, z+10, block.AIR.id)
```

Finally, we write the main game loop so that is waits for a button press, then set off a bomb. We should remember that our button connects to 0 Volts when it is presses, which is why we have to compare the GPIO against `False` in order to detect a button press:

```
try:
      while True:
            time.sleep(0.1)
            if GPIO.input(BUTTON) == False:
                  pos = mc.player.getTilePos()
                  bomb(pos.x, pos.y, pos.z)
finally:
      GPIO.cleanup()
```

We save our program and we run it. Then in the Minecraft world, we go somewhere, we press the button and we run for our lives. Figure 2 shows the aftermath of the explosion.

**FIGURE 2 A CRATER BLOWN INTO THE MINECRAFT WORLD**

The size of the crater is set by the numbers inside the `setBlocks ()` statement. By tweaking the numbers in that statement, we can expand or shrink the size of the explosion. Try doing so and see what happens!

## 1.2.4 Conclusion

We successfully programmed a button to simulate an explosion in our Minecraft world. We also programmed a display to show the countdown until the bomb goes.

Why not doing even more?

Add three more buttons to your circuit, in the same way that you added the first button. Then write some more Python code to make those buttons do different things, like post random messages to the Minecraft chat, teleport your player to a secret location, build blocks of different types where your player is standing, or even build a whole house where you are standing. The following GPIO numbers are safe to use for your three extra buttons.

| Button | Raspberry Pi Pin |
|---|---|
| BUTTON2 | 14 |
| BUTTON3 | 23 |
| BUTTON4 | 24 |

## 1.3 References

O'Hanlon M. & Whale D., 2015, Adventures in Minecraft, Wiley Publications.

## 1.4 Appendix

Quick Reference Table

| Configuring GPIO pins | Reading and writing GPIO pins |
|---|---|
| `import RPi.GPIO as GPIO # RaspberryPi`<br>`import anyio.GPIO as GPIO # Arduino`<br>`GPIO.setmode(GPIO.BCM)`<br>`GPIO.setup(5, GPIO.OUT) # output`<br>`GPIO.setup(5, GPIO.IN) # input` | `GPIO.output(5, True) # set on (3.3V)`<br>`GPIO.output(5, False) # set off (0V)`<br>`if GPIO.input(6) == False:`<br>`  print("pressed")` |
| **Safely cleaning yp after using the GPIO** | **Using the 7-segment display module** |
| `try:`<br>`  do_something() # put code here`<br>`finally:`<br>`  GPIO.cleanup()` | `import anyio.seg7 as display`<br>`LED_PINS = [10,22,25,8,7,9,11,15]`<br>`ON = False # common-Anode`<br>`ON = True # common-Cathode`<br>`display.setup(GPIO, LED_PINS, ON)` |
| **Writing character to the 7-segment display** | **Other 7-segment display function** |
| `display.write("3")`<br>`display.write("A")`<br>`display.write("up")` | `display.setdp(True) # point on`<br>`display.setdp(False) # point off`<br>`display.clear()`<br>`display.pattern([1,1,1,1,1,1,1,1])` |