Implement SmartLight

GATT Server

Rename the `gatt-server` script to `smartlight-gatt-server`.

Using `nano` or a suitable editor, open the `smartlight-gatt-server` file. You will see that it contains 3 python classes, `DemoService`, `DemoCharacteristic`, and `CharacteristicUserDescriptionDescriptor`.

Include the SmartHAT

You will need a way to access the SenseHAT LEDs and temperature sensor from the SmartLight service. At the top of the `smartlight-gatt-server` script, add the following to import the SenseHAT package and create the `sense` object.

```
from sense_hat import SenseHat
sense = SenseHat()
```

SmartLight Service

Change the DemoService class to match the Smart Light design as follows:

- change the class name to "SmartLightService"
- change TEST_SVC_UUID to SVC_UUID
- change the UUID to match the proposed design
- change the characteristic name to LightColourCharacteristic

```
class SmartLightService(Service):
    SVC_UUID = 'FF10'

    def __init__(self, bus, index):
        Service.__init__(self, bus, index, self.SVC_UUID, True)
        self.add_characteristic(LightColourCharacteristic(bus, 0, self))
```

LightColourCharacteristic

Update the `DemoCharacteristic` class as follows

- change the class name to "LightColourCharacteristic"
- change the UUID to match the proposed design(FF11)
- in the `__init__` function (i.e. the constructor):
  - initialise the light colour value to 0,0,0 (representing RGB)
  - clear the SenseHAT LED matrix

```
class LightColourCharacteristic(Characteristic):
    CHRC_UUID = 'FF11'

    def __init__(self, bus, index, service):
        Characteristic.__init__(
                self, bus, index,
                self.CHRC_UUID,
                ['read', 'write', 'writable-auxiliaries'],
                service)
        self.value = [0,0,0]
        sense.clear()
        self.add_descriptor(
                LightColourUserDescription(bus, 0, self))
```

Whenever a value is received from a connected client, change the colour of the LED matrix to the received value. Find the `WriteValue` function in the script and change it to the following:

```
    def WriteValue(self, value, options):
        print('light Colour WriteValue called')
        if len(value)!=3:
            raise InvalidValueLengthException()
        sense.clear([int(value[0]),int(value[1]),int(value[2])])
        self.value = value
        print('Finished changing colour!')
```

Light Colour User Description

Finally, change the name of the `CharacteristicUserDescriptionDescriptor` class and update the `value` field to a more accurate description of the characteristic:

```
class LightColourUserDescription(Descriptor):
    CUD_UUID = '2901'

    def __init__(self, bus, index, characteristic):
        self.writable = 'writable-auxiliaries' in characteristic.flags
        self.value = array.array('B', b'Smart Light Colour(RGB)')
        self.value = self.value.tolist()
        Descriptor.__init__(
                self, bus, index,
                self.CUD_UUID,
                ['read', 'write'],
                characteristic)
...
...
```

Update main() function

Find the `main()` function in the script. Replace the line that adds the Demo Service (`app.add_service(DemoService(bus, 1))`) to the following:

```
app.add_service(SmartLightService(bus, 0))
```

Run it!

If the example service is still running, stop it using `ctrl+c` . As before, run the service by entering `sudo ./smartlight-gatt-server` at the RPi command prompt and connect using LightBlue on your device. Write "FFFFFF" to the light colour characteristic and you should see the SenseHAT LED matrix light up. Try some other colours to make sure it's working.

Any Problems?

Bluetooth can be tricky. When you make changes to your service you may need to stop and start Bluetooth on your device/smartphone. If you cannot see the service on your device, restart Bluetooth on the RPi and make sure Low Energy Advertising is allowed:

```
sudo hciconfig hci0 down
sudo hciconfig hci0 up
sudo hciconfig hci0 leadv 0
```