IoT simulator

Documentation

Made by: Illia Takhtamyshev

Contents

Task

Develop a Python-based IoT simulator for a smart home automation system. The simulator should emulate the behavior of various IoT devices commonly found in a smart home, such as smart lights, thermostats, and security cameras. You will also create a central automation system that manages these devices and build a monitoring dashboard to visualize and control the smart home. Apply Python programming skills, including OOP, data handling, real-time data monitoring, and graphical user interfaces (GUIs).

1. Part 1: IoT Device Emulation (25 %)

- **Device Classes:**
  Create Python classes for each type of IoT device you want to simulate, such as SmartLight, Thermostat, and SecurityCamera. Each class should have attributes like device ID, status (on/off), and relevant properties (e.g., temperature for thermostats, brightness for lights, and security status for cameras).

- **Device Behavior:**
  Implement methods for each device class that allows for turning devices on/off and changing their properties. Simulate realistic behavior, such as gradual dimming for lights or setting temperature ranges for thermostats.

2. Part 2: Central Automation System (25 %)

- **Automation System Class:**
  Create a central automation system class, e.g., AutomationSystem, responsible for managing and controlling all devices. It should provide methods for discovering devices, adding them to the system, and executing automation tasks, such as activating lights automatically when motion is detected.

- **Randomization mechanism:**
  Include a randomization mechanism to simulate changing device states and properties.

- **Gathering sensor data:**
  Gathering sensor data(temp, brightness, and/or camera data) and storing it in a file for future analysis.

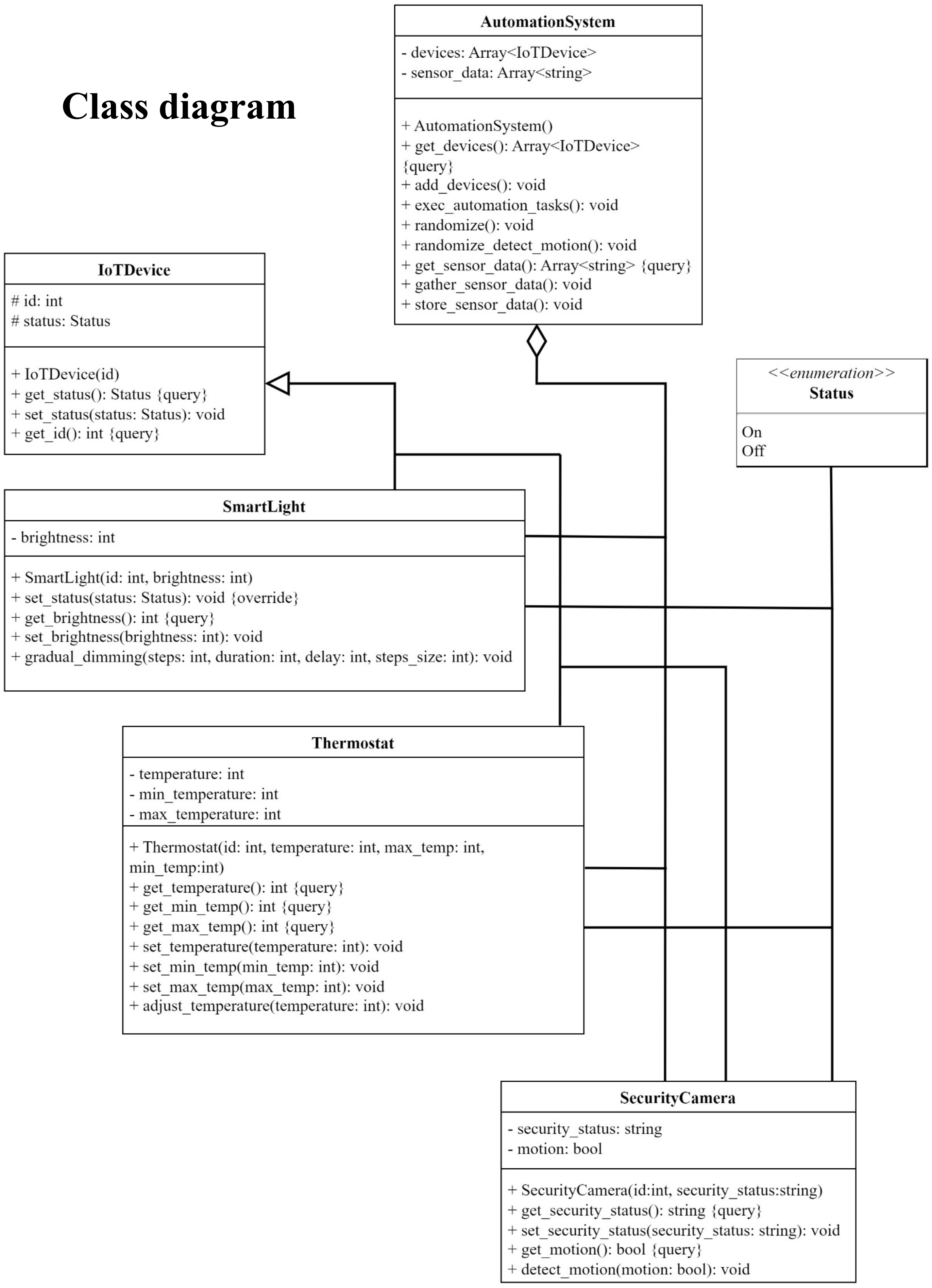
3. Part 3: Monitoring Dashboard (20%)

- **Graphical User Interface (GUI):**
  Create a GUI for monitoring and controlling the smart home system. You can use Python GUI libraries like Tkinter. The GUI should display the status and properties of each device, provide controls to interact with them and visualize data.
- **Real-time Data Monitoring:**
  Display real-time data from the simulated devices on the dashboard. This may include temperature for thermostats, motion detection status for cameras, and brightness levels for lights. User Interaction: Allow users to interact with devices through the GUI, such as toggling lights on/off, adjusting thermostat settings, and arming/disarming security cameras.
- **Automation Loop:**
  Implement a automation loop that runs periodically (e.g., every few seconds) to trigger automation rules, update device states.

4. Part 4: Documentation (30%)

- **Documentation:**
  Provide clear documentation for your code, including class descriptions, method explanations, and instructions on how to run `the simulation and use the dashboard.
- **Test cases:**
  Develop test cases to ensure that the simulator and automation system behave as expected. Test various scenarios, such as different automation rules and user interactions.

Class diagram


AutomationSystem
- devices: Array<IoTDevice>
- sensor_data: Array<string>
+ AutomationSystem()
+ get_devices(): Array<IoTDevice> {query}
+ add_devices(): void
+ exec_automation_tasks(): void
+ randomize(): void
+ randomize_detect_motion(): void
+ get_sensor_data(): Array<string> {query}
+ gather_sensor_data(): void
+ store_sensor_data(): void
IoTDevice
# id: int
# status: Status
+ IoTDevice(id)
+ get_status(): Status {query}
+ set_status(status: Status): void
+ get_id(): int {query}
<<enumeration>>
Status
On
Off
SmartLight
- brightness: int
+ SmartLight(id: int, brightness: int)
+ set_status(status: Status): void {override}
+ get_brightness(): int {query}
+ set_brightness(brightness: int): void
+ gradual_dimming(steps: int, duration: int, delay: int, steps_size: int): void
Thermostat
- temperature: int
- min_temperature: int
- max_temperature: int
+ Thermostat(id: int, temperature: int, max_temp: int, min_temp:int)
+ get_temperature(): int {query}
+ get_min_temp(): int {query}
+ get_max_temp(): int {query}
+ set_temperature(temperature: int): void
+ set_min_temp(min_temp: int): void
+ set_max_temp(max_temp: int): void
+ adjust_temperature(temperature: int): void
SecurityCamera
- security_status: string
- motion: bool
+ SecurityCamera(id:int, security_status:string)
+ get_security_status(): string {query}
+ set_security_status(security_status: string): void
+ get_motion(): bool {query}
+ detect_motion(motion: bool): void

Short description of classes and explanations of methods

“IoTDevice” class

The class represents an IoT device with an id and a status. It provides methods to get and set the status and get the device's ID.

1. **Constructor “__init__(self, id)”**
- Initializes an IoT device with a given id and set its status to "Off."

2. **Method “get_status(self)”**
- Returns the status of the IoT device ("On" or "Off").

3. **Method “set_status(self, status)”**
- Sets the status of the IoT device ("On" or "Off").

4. **get_id(self)**
- Returns the id of the IoT device

“SmartLight” class

The class represents an IoT device for controlling light. It extends the IoTDevice class and adds attributes for brightness control and a method for gradual dimming.

1. **Constructor “__init__(self, id, brightness)”**
- Initializes a smart light with a given id and initial brightness.

2. **Method “set_status(self, status)”**
- Sets the status of the smart light. It ensures that the brightness is adjusted when switching between "On" and "Off" states.

3. **Method “get_brightness(self)”**
- Returns the brightness of the smart light.

4. **Method “set_brightness(self, brightness)”**
- Sets the brightness of the smart light, ensuring that the status is adjusted when the brightness changes.

5. **Method “gradual_dimming(self, steps, duration, delay, step_size)”**
- Method to gradually dim the light by reducing its brightness in a series of steps over a specified duration. It also turns off the light after dimming to zero.

“Thermostat” class

The class represents an IoT device for controlling temperature. It extends the IoTDevice class and adds attributes for temperature control.

1. **Constructor “__init__(self, id, temperature, min_temp, max_temp)”**
- Initializes a thermostat with a given id, initial temperature, and temperature range.

2. **Method “get_temperature(self)”**
- Returns the current temperature set on the thermostat.

3. **Method “get_min_temp(self)”**
- Returns the minimum temperature allowed by the thermostat.

4. **Method “get_max_temp(self)”**
- Returns the maximum temperature allowed by the thermostat.

5. **Method “set_temperature(self, temperature)”**
- Sets the temperature on the thermostat within the specified range and ensures it is in the "On" state.

6. **Method “set_min_temp(self, min_temp)”**
- Sets the minimum temperature allowed by the thermostat.

7. **Method “set_max_temp(self, min_temp)”**
- Sets the maximum temperature allowed by the thermostat.

8. **Method “adjust_temperature(self, temperature)”**
- Adjusts the thermostat's temperature within the specified range while ensuring it is in the "On" state.

“SecurityCamera” class

The class represents an IoT device for security and motion detection. It extends the IoTDevice class and adds attributes for security status and motion detection.

1. **Constructor “__init__(self, id, security_status)”**
- Initializes a security camera with a given id and security status.

2. **Method “get_security_status(self)”**
- Returns the security status of the camera.

3. **Method “set_security_status(self, security_status)”**
- Sets the security status of the camera.

4. **Method “get_motion(self)”**
- If the camera is in the "On" state, checks if motion is detected.

5. **Method “detect_motion(self, motion)”**
- Simulates motion detection by setting the motion attribute of the camera based on the provided argument, ensuring the camera is in the "On" state.

“Status” enum

The Status enumeration defines two possible status values for IoT devices, "On" and "Off," using the Python enum module.

“AutomationSystem” class

The class represents an automation system that manages a collection of IoT devices. It allows adding devices, executing automation tasks, gathering sensor data, and storing the data to a file.

1. **Constructor “__init__(self)”**
- Initializes an empty list of devices and an empty sensor data list.

2. **Method “get_devices(self)”**
- Returns the list of IoT devices managed by the automation system.

3. **Method “add_devices(self)”**
- Adds specific types of IoT devices (SmartLight, Thermostat, SecurityCamera) to the system.

4. **Method “exec_automation_tasks(self)”**
- Checks if a security camera detects motion and, if so, turns on a connected smart light if it is off with a brightness greater than 0.

5. **Method “randomize(self)”**
- Randomly adjusts the brightness of a smart light, sets the temperature and temperature range of a thermostat, and simulates motion detection in a security camera.

6. **Method “randomize_detect_motion(self)”**
- Randomly simulates motion detection in a security camera.

7. **Method “get_sensor_data(self)”**
- Returns the sensor data collected by the automation system.

8. **Method “gather_sensor_data(self)”**
- Method that collects sensor data including light brightness, thermostat temperature, and camera motion detection, and appends it to the sensor data list with timestamps.

9. **Method “store_sensor_data(self)”**
- Method to store the sensor data to a text file named "out.txt".

Instructions

How to run the simulation (Windows)

One of the ways to run the simulation after downloading IoT simulator

1. **Open the folder in the console.**
- To do this you can copy the path of the folder in Windows Explorer, open console and type “cd <path you copied>”.

2. **Type “python main.py” to run the GUI**
- It is possible only if Python is installed on your computer

How to use the dashboard

GUI allows monitoring of the smart home system through several text fields which are implemented on the dashboard. For example, to see the current status ("ON" or "OFF") of all the devices it is enough to check the text box which goes right after the "randomize" button on the top. To check the current state of core feature of each device, for instance brightness of the light, it is enough to look at the text right after the button "Toggle ON/OFF".

For controlling the smart home system there are located different buttons and scales. For example, to turn on the automation you can tap on the first button on the top of the page called "Automation ON/OFF". To adjust different things to devices and change their statuses you can interact with buttons and scales which were created for each device. Hence, to change the brightness of the light you can pull the scale which is under the text "Living room light brightness".

Testing

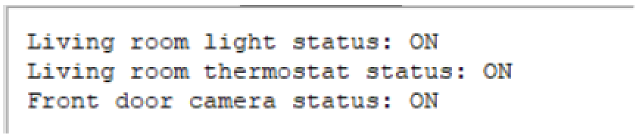
1. Test toggling on and off the devices

Description:

On the start all devices are turned off. Tap on three buttons "Toggle ON/OFF" for each device to turn on the devices. Tap on the buttons once again to turn off the devices.

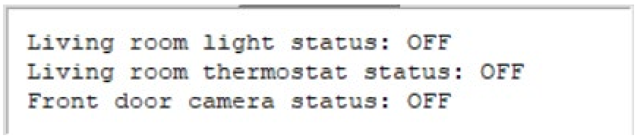
Result:

After the first taps all three devices are turned on:



```
Living room light status: ON
Living room thermostat status: ON
Front door camera status: ON
```

After the second taps all three devices are turned off:

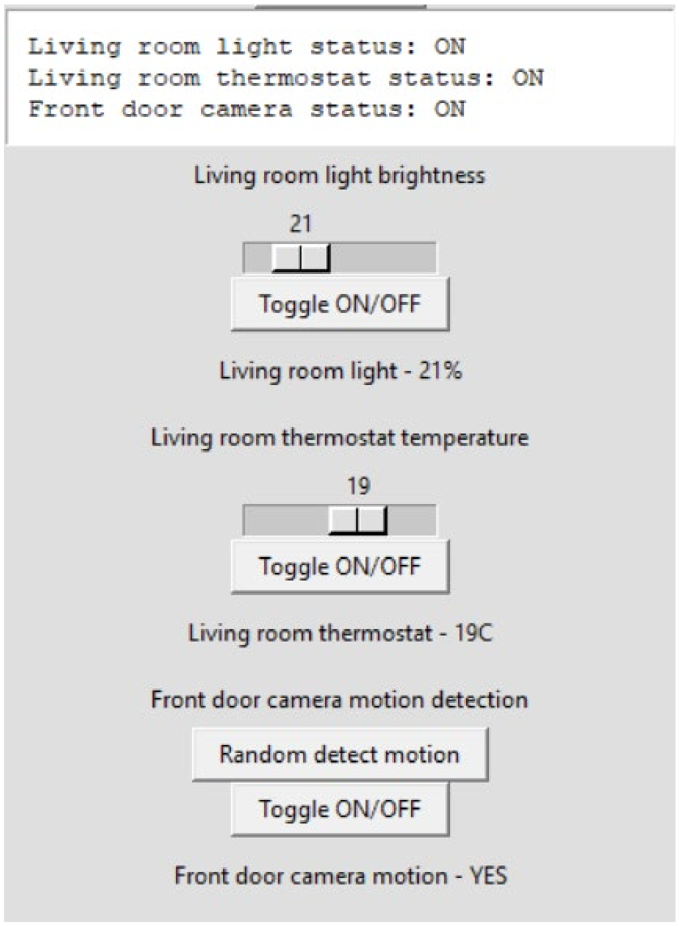


```
Living room light status: OFF
Living room thermostat status: OFF
Front door camera status: OFF
```

2. Test adjusting values and turning on caused by adjusting values

Description:
On the start all devices are turned off. Toggle brightness bigger than zero for light what has to cause adjusting new brightness and turning on the lights. Toggle thermostat temperature what has to cause adjusting new temperature and turning on the thermostat. Tap on “Random detect motion” what has to cause adjusting new motion detection status and turning on the camera.

Result:
All three devices are turned on. All values are adjusted.



```
Living room light status: ON
Living room thermostat status: ON
Front door camera status: ON
```

Living room light brightness

21

Toggle ON/OFF

Living room light - 21%

Living room thermostat temperature

19

Toggle ON/OFF

Living room thermostat - 19C

Front door camera motion detection

Random detect motion

Toggle ON/OFF

Front door camera motion - YES

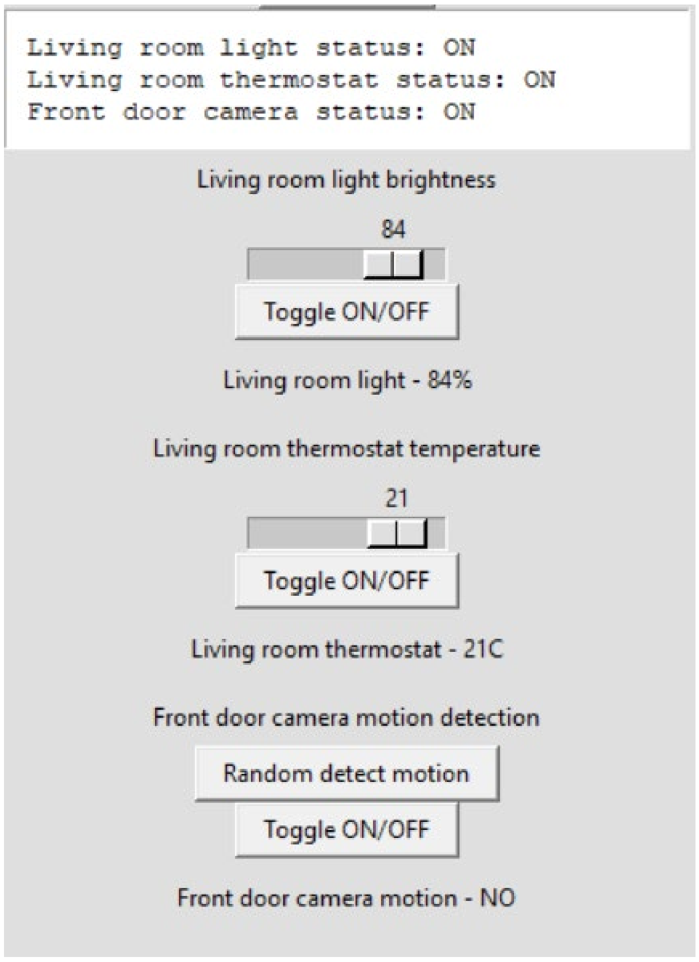
3. Test randomizing

Description:

On the start all devices are turned off. Tap on “Randomize” to randomize values for the devices what has to adjust random values and turn on the devices.

Result:

All three devices are turned on. Light received random brightness. Thermostat received random minimum temperature, maximum temperature and current temperature. Camera received random motion detection.



```
Living room light status: ON
Living room thermostat status: ON
Front door camera status: ON
```

Living room light brightness

84

Toggle ON/OFF

Living room light - 84%

Living room thermostat temperature

21

Toggle ON/OFF

Living room thermostat - 21C

Front door camera motion detection

Random detect motion

Toggle ON/OFF

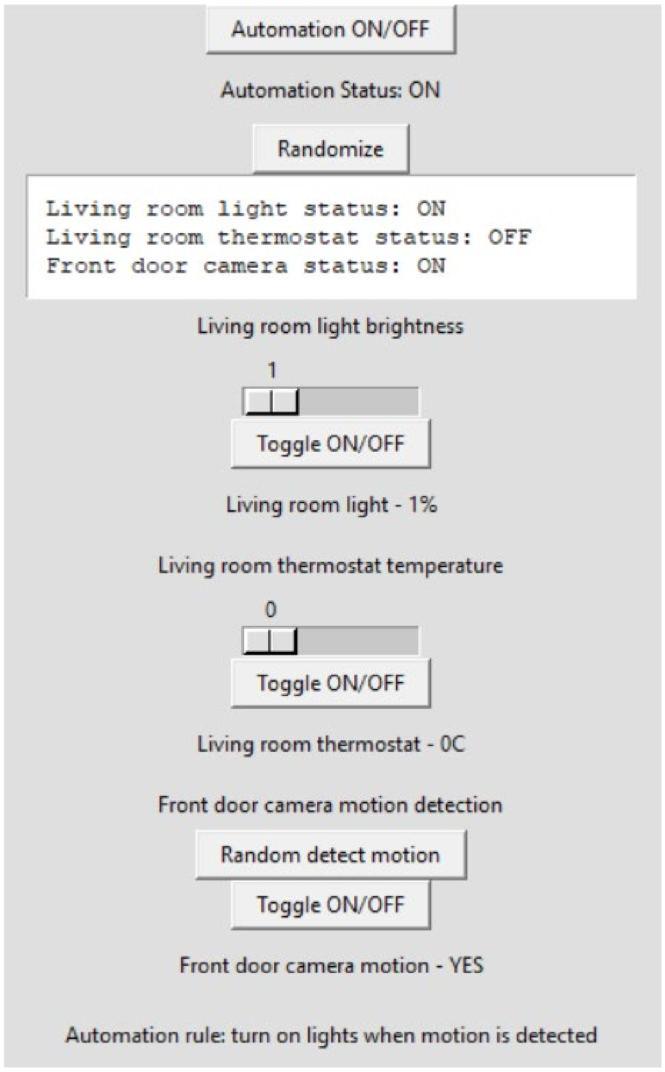
Front door camera motion - NO

4. Test automation

Description:
On the start all devices are turned off. Tap on “Automation ON/OFF” to turn on the automation which turns on the lights if the motion is detected by the camera. Tap multiple times on “Random detect motion” until the motion will be randomly detected what has to cause turning on the lights and adjusting 1% brightness.

Light cannot be turned off while motion is detected. Tap on “Toggle ON/OFF” for the light to try turning it off what has to do nothing.

Result:
Camera and light are turned on, light has 1% brightness. While motion is detected turning off the light is not possible.


Automation ON/OFF
Randomize
Living room light brightness
1
Toggle ON/OFF
Living room thermostat temperature
0
Toggle ON/OFF
Front door camera motion detection
Random detect motion
Toggle ON/OFF

Automation Status: ON

```
Living room light status: ON
Living room thermostat status: OFF
Front door camera status: ON
```

Living room light - 1%

Living room thermostat - 0C

Front door camera motion - YES

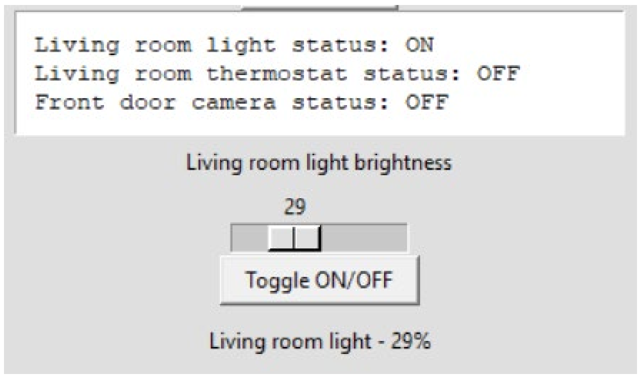
Automation rule: turn on lights when motion is detected

5. Test gradual dimming for the light brightness.

Description:
On the start all devices are turned off. Toggle brightness bigger than zero for light what has to cause adjusting new brightness and turning on the light. Tap on "Toggle ON/OFF". It has to cause gradual dimming for the light brightness what means that brightness will gradually fall during some time until hitting 0% what is supposed to cause turning off the light.

Result:
After toggling the light brightness bigger than zero light turned on. Tap on "Toggle ON/OFF" caused gradual dimming where the brightness felt to 0 and light was turned off.

After toggling the light brightness bigger than zero the light is turned on:



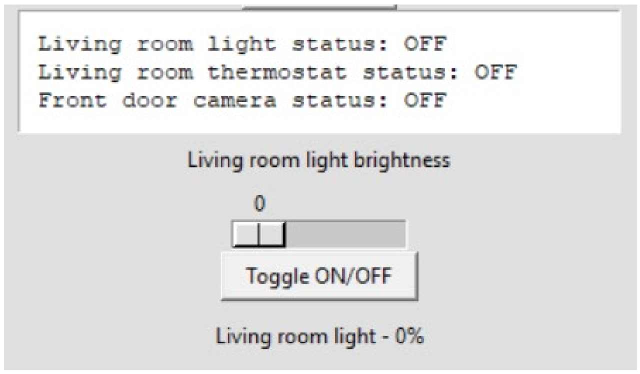
```
Living room light status: ON
Living room thermostat status: OFF
Front door camera status: OFF
```

Living room light brightness

29

Living room light - 29%

After the gradual dimming caused by tap on "Toggle ON/OFF" light is turned off:



```
Living room light status: OFF
Living room thermostat status: OFF
Front door camera status: OFF
```

Living room light brightness

0

Living room light - 0%