IoT Simulator

NAZIFA NAZRUL RODOSHI

W8CZNE

Contents

Task

In this assignment, you will need to develop a Python-based IoT simulator for a smart home automation system. The simulator should emulate the behaviour 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 visualise and control the smart home. This assignment will help you apply your Python programming skills, including OOP, data handling, real-time data monitoring, and graphical user interfaces (GUIs).

Part 1: IoT Device Emulation: 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 allow for turning devices on/off and changing their properties. Simulate realistic behavior, such as gradual dimming for lights or setting temperature ranges for thermostats. Randomization: Include a randomization mechanism to simulate changing device states and properties over time.

Part 2: Central Automation System: 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. Simulation Loop: Implement a simulation loop that runs periodically (e.g., every few seconds) to trigger automation rules, update device states, and simulate device behaviours.

Part 3: Documentation: Documentation: Provide clear documentation for your code, including class descriptions, method explanations, and instructions on how to run the simulation and use the dashboard. 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

Part 4: Monitoring Dashboard: 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 includes temperature graphs 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.

Description of the task

Classes created: SmartLight.py, Thermostat.py, SecurityCamera.py, AutomationSystem.py, GUISmartHome.py and style.py .

Description of the classes and methods

Class SmartLight:

- __init__(self, ID, Status, Brightness): This is the constructor for the SmartLight class. It initializes a new instance with an identifier (ID), a status (Status), and a brightness level (Brightness).

- turnOn(self): This method sets the light's status to "ON" and prints a message indicating the light has been turned on.

- turnOff(self): This method sets the light's status to "OFF", resets the brightness to 0, and prints a message indicating the light has been turned off.

- setBrightness(self, Brightness): If the light is on, this method adjusts the brightness to the value provided, ensuring it stays within the range of 0 to 100.

- controlBrightness(self): If the light is on and the brightness is above 0, this method decreases the brightness by 10 units (not allowing it to go below 0) and prints the new brightness level. If the light is off, it prints a message to indicate that the light is off.

Class Thermostat:

- __init__(self, ID, Status, Temperature): Constructor that initializes a new Thermostat instance with a unique identifier (ID), its operational status (Status), and the current temperature setting (Temperature).

- turnOn(self): Activates the thermostat, sets its status to "ON", and initializes a default temperature setting (though the code incorrectly sets self.SetTemperature instead of self.Temperature).

- turnOff(self): Deactivates the thermostat, sets its status to "OFF", and resets the temperature to a lower default value (similarly, the code incorrectly sets self.SetTemperature).

- setTemperature(self, Temperature): Adjusts the thermostat's temperature to a specified value if the thermostat is on, ensuring the temperature remains between 10 and 30 degrees.

- controlTemperature(self): Decreases the thermostat's temperature by 10 degrees if it is on and the temperature is above the minimum threshold, then prints the new temperature. If the thermostat is off, it prints a message indicating that.

Class SecurityCamera:

- __init__(self, ID, Status, Security): Constructor that initializes a new SecurityCamera instance with an identifier (ID), operational status (Status), and a default MotionDetected attribute set to False.

- turnOn(self): Activates the camera by setting its Status to "ON" and outputs a message indicating the camera is turned on.

- turnOff(self): Deactivates the camera, sets its Status to "OFF", resets MotionDetected to False, and outputs a message indicating the camera is turned off.

- detectMotion(self, light): Checks for motion if the camera is turned on, randomly setting MotionDetected to True or False. If motion is detected and the passed SmartLight instance is off, it turns on the light.

- automationRule(self, SmartLight): If the camera is on and detects motion, it triggers an automation rule that turns on the provided SmartLight instance and prints a message about the action taken. If no motion is detected, it prints a message indicating no action was taken.

Class GUISmartHome:

- Window Setup: A Tkinter window titled "Smart Home IoT Simulator" is created, and a custom style is applied.

- Device Initialization: Instances of SmartLight, Thermostat, and SecurityCamera are created with initial parameters.

- Toggle Functions: There are functions defined to toggle the state of each device (on/off).

- Status Update Function: updateStatus is responsible for updating the GUI labels with the current status of each device. It is set to run periodically every 1000 milliseconds.

- Brightness and Temperature Update Functions: These functions are linked to sliders that control the brightness of the light and the temperature of the thermostat.

- Motion Detection: There's a function to simulate motion detection by the security camera, which can trigger the light to turn on.

- GUI Components: Labels, sliders, and buttons are created for each device to display their status and allow user interaction.

- Grid Layout: The layout is organized into a grid where each component is placed in a specific row and column, with padding for visual spacing.

- Event Loop: The script enters an event loop to display the GUI and wait for user interactions.

- Repetitive Code: There is some repetitive code, especially in the creation of Tkinter variables and grid layout definitions, which could be refactored for conciseness without changing functionality.

Class style:

- Default Theme: It sets the ttk.Style which is used to change the appearance of the widgets.

- Custom Colors: It defines a set of custom colors for the background (bg_color), text (text_color), buttons (button_color), and sliders (slider_color).

- Default Font: It sets a default font (default_font) for all widgets in the application.

- Styles Configuration:

- Configures the TButton style to use the default_font, with specified padding and the defined button_color for the background and text_color for the text.
- Configures the TLabel style to use the default_font, with the bg_color for the background and text_color for the text.
- Configures the Horizontal.TScale style for horizontal sliders with the slider_color for the trough, a specified slider length, and a flat relief.
- Background Configuration: It sets the bg_color for the background of all TFrame and TLabel widgets.

- Active State for Buttons: It changes the button color when it's active (clicked on) by mapping the background and foreground properties to the button_color and text_color, respectively, only when the button is in an 'active' state.

How To Run

How To: Simulation

- Simulation Toggle Button: In the GUI, locate the buttons labeled "toggle_light", "toggle_thermostat", "toggle_camera ",   Clicking this button will either initiate or halt the simulation process. The simulation consists of all the automated interactions between the SmartLight, Thermostat, and SecurityCamera.
- Random Detect Motion Button: Find the button labeled "Random Detect Motion" within the GUI. Pressing this button will randomly simulate motion detection by the security camera. If motion is detected while the simulation is active, the camera may change its status to reflect the detection of motion, and the SmartLight may adjust its brightness level accordingly to simulate an automated response to the motion detected.

How To: Dashboard

- **Light Brightness Slider**: On the dashboard, locate the slider labeled "Brightness." This control allows you to adjust the smart light's brightness. Slide it to the right to increase brightness, or to the left to decrease it. The light's current brightness level will be displayed on the corresponding label.
- **Thermostat Temperature Slider**: Look for the slider labeled "Temperature (°C)" in the GUI. This control allows you to set the thermostat's temperature. Dragging the slider up will increase the temperature, while dragging it down will decrease it. The thermostat's current temperature setting will be shown on its label.
- **Toggle Camera Button**: The button marked "Toggle Security Camera" lets you switch the security camera on or off. Each click will change the camera's operational status, which is reflected in the camera's status label.
- **Toggle Light Button**: The "Toggle Light" button is used to power the smart light on or off. Clicking this button changes the light's status and updates the brightness level displayed on the slider and label.
- **Toggle Thermostat Button**: By clicking the "Toggle Thermostat" button, you can activate or deactivate the thermostat. This action will be reflected in the thermostat's status label and may affect the temperature setting shown on the slider.
- **Random Detect Motion Button**: This button is used to simulate a random motion detection event by the security camera. When clicked, it can potentially trigger the smart light to turn on if the simulation deems that motion has been detected.

- **Camera Motion Status Label**: The status label for the camera displays whether motion has been detected by the camera. If the camera perceives movement, the label will indicate "Motion: YES," and if no movement is perceived, it will show "Motion: NO."
- **Log Text Box**: Although not implemented in the provided code, a log text box could be used to display real-time logs of the simulation. It would show messages about device status changes, actions taken by the simulation (like turning on the light due to detected motion), and other relevant events as they occur within the dashboard.

Test Cases

Test Case 1: Turning On the Simulation and Detecting Motion

1. Click the "Toggle Simulation" button to start the simulation.
2. Observe the log text box for simulation updates.
3. Click the "Random Detect Motion" button.
4. Check the "Camera Motion Status" label to see if it displays "Motion: NO."
5. Observe any changes in device statuses (e.g., camera and light).

Test Case 2: Controlling the Smart Light

1. Start the simulation by clicking the "Toggle Simulation" button.
2. Use the "Brightness" slider to adjust the brightness of the smart light.
3. Observe the changes in the light's brightness on the GUI.
4. Toggle the light on and off using the "Toggle Light" button.
5. Observe the changes in the light's status on the GUI.

Test Case 3: Adjusting the Thermostat

1. Start the simulation by clicking the "Toggle Simulation" button.
2. Use the "Temperature (°C)" slider to set the desired temperature on the thermostat.
3. Observe the changes in the thermostat's temperature setting on the GUI.
4. Toggle the thermostat on and off using the "Toggle Thermostat" button.
5. Observe the changes in the thermostat's status on the GUI.

Test Case 4: Turning Off the Simulation

1. Start the simulation by clicking the "Toggle Simulation" button.
2. Observe the simulation running with device interactions.
3. Click the "Toggle Simulation" button again to stop the simulation.
4. Check if the simulation stops, and device interactions cease.
5. Observe any final status updates on the GUI.