Setting Up a Simple Puzzle with EscapeOS

* Sign up for EscapeOS access here: <https://forms.gle/eXfwY5Xg8VZPuWj46> or here ⬆
  + This will also be used to provision Google Drive access
* After you receive access you can use your credentials at <https://byupe.escapeos.com>
* Install the Arduino IDE (v2) <https://www.arduino.cc/en/software>
* Prepare Arduino IDE for use with ESP32s
  + Use esp32 by Espressif System v2.0.11
  + Optionally, follow this tutorial to do so <https://randomnerdtutorials.com/installing-esp32-arduino-ide-2-0/>
* EscapeOS uses MQTT to allow for simple communication between puzzles, sensors, event controllers, and servers. You don’t need to know the intricacies, but this provides a basic overview of the protocol <https://mqtt.org>
* Download **Example1Puzzle** folder. Rename the folder and the .ino file inside to be Group#Puzzle (e.g., **Group1Puzzle**)
* Duplicate the **example.credentials.h** file and name the duplicate **credentials.h**
* Fill **credentials.h** file. These credentials/parameters will be used but can be overridden on re-uploads or in real time using the appropriate serial command
  + WIFI\_SSID/WIFI\_PASSWORD
    - Valid WiFi credentials. If your local network involves a captive portal which requires interaction you can use a Personal Hotspot instead
  + MQTT\_SERVER
    - Comes from mqtt\_credentials.txt
  + MQTT\_USER
    - Comes from mqtt\_credentials.txt
  + MQTT\_PASSWORD
    - Comes from mqtt\_credentials.txt
  + BASE\_STATION\_CLIENT\_ID
    - *eOS-baseGroup#*
    - E.g., *eOS-baseGroup1*
    - This is defined based on the room your puzzle is in
  + PUZZLE\_CLIENT\_ID
    - *eOS-puzzleGroup#*
    - E.g., *eOS-puzzleGroup1*
    - This is the client name your puzzle will connect to MQTT with. It also defines the MQTT topic your puzzle will publish its current status on (will be <STATUS\_TOPIC>-status). This also functions as the value used as rfid in achieve checkpoint messages
  + BASE\_STATION\_TOPIC
    - *test-base\_station*
    - This is the MQTT topic that achieve checkpoint messages will be sent on
  + COMMAND\_TOPIC
    - *eOS-puzzleGroup#-command*
    - E.g., *eOS-puzzleGroup1-command*
    - This is the MQTT topic which your puzzle will subscribe to in order to handle any commands sent to it from EscapeOS
  + SHOW\_DEBUG
    - *true*
* Install all required libraries using Arduino IDE (see contents of **escapeos.h** to see which are necessary). Includes at least the following:
  + NTPClient by Fabrice Weinberg v3.2.1
  + PubSubClient by Nick O’Leary v2.6 OR <https://github.com/knolleary/pubsubclient/releases/tag/v2.8>
    - If installing from GitHub, after moving folder to Libraries, rename folder to *PubSubClient*
  + ArduinoJson by Benoit Blanchon v6.16.1
  + ESP\_WiFiManager by Khoi Hong v1.1.1
* Upload Project to your board as is. After uploading successfully:
  + Log in using your user/password to byupe.escapeos.com
  + Start the game in your group’s room by pressing the Start button (looks like a “play” icon ▶). Expand the room view by clicking on your room’s title
  + Once the game is running, reset your board by clicking the flash/EN button or unplugging it and plugging it back in
  + If everything was successful you should see that the Group # Puzzle checkpoint gets achieved. You should also see the onboard LED blink in groups of 10
* Once verifying that everything is working you can start making changes to the code for your puzzle and start connecting appropriate peripherals to it. The first change you should make is to remove the EXAMPLE CODE sections from the .ino file
* Throughout the semester make sure to periodically upload your whole project with an explanation of current functionality/peripherals/wiring to your group’s Google Drive folder. Use sub folders within your group folder as a primitive form of version control so that progress throughout the semester can be seen

Simple Button Puzzle

* After completing the setup steps above you can begin the first lab/assignment, which is to make a simple button puzzle. Puzzle requirements:
  + When the button is pressed:
    - Consider itself solved
    - Turn on the onboard and connected LEDs
    - Send the checkpoint achieved message like the simple original code
    - Using a delay, ignore further input for 10 seconds, then resume original state of waiting for input
* You can use the same checkpoint verification using a game in your group’s room

Complex Button Puzzle

* After completing the Simple Button Puzzle you can begin the second lab/assignment, which is to make a complex button puzzle. Puzzle requirements:
  + When the button is pressed (but the puzzle isn’t currently solved):
    - Turn on the onboard and connected LEDs
    - Update puzzle state
  + When the button is pressed in the correct sequence your puzzle should:
    - Consider itself solved
    - Send the checkpoint achieved message like the simple original code
    - Blink onboard and connected LEDs
    - Send signal to a pin that could be used to control a relay (depending on availability, could use another connected LED)
    - Ignore button input
    - After 10 seconds, stop blinking, stop signaling relay, and no longer consider itself solved. Accept new button input again
  + Whenever puzzle state changes (a long or short button press is detected or puzzle solved state changes):
    - Send puzzle state through the status topic. Puzzle state should be like the following JSON (but properly reflect current state:
      * {"solved": true, "tenRecentPresses": [0, 1, 1, 0, 0, 1, 1],"timestamp":1736935849}
      * 0 = short press, 1 = long press. tenRecentPresses should contain the ten most recent presses (if ten presses haven’t been made yet it should only show those made so far)
  + Button Presses
    - Make sure to properly debounce button signal to prevent confusion
    - Short Press: < 2 seconds
    - Long Press: >= 2 seconds
    - Do not use extensive delays to determine press length. Use currentMillis calculations instead
  + Store Correct Sequence in Flash Memory (so it won’t be lost on reset). Provide an initial value from the below but make it so that the Correct Sequence can be updated and saved to Flash Memory by sending a Serial Command
    - Update logic should be in your command callback. Serial command should be sent as JSON like the following:
      * {"topic":"eOS-puzzleGroup1-command","message":"storeNewCorrectSequence","newSequence":"0100111000"}
      * newSequence can be received as a String or Array based on your preferences. Document which option you choose
    - Helper functions *readFromMemory* and *writeToMemory* are available from **escapeos.h** for use in storing/reading the correct sequence
  + Sequence Options
    - 1001001011
    - 1001001001
    - 1110001110
    - 0001110001
    - 1010100100
    - 0110010110
  + At the end of setup print the correct sequence to the serial monitor for easy testing. After a button press has been registered, print the received button press and the next correct button press to the serial monitor