Università di Parma

Progetto PLSD

Random Number Generator

Davide Reverberi, Alessandro Galloni

Indice

1 Obiettivo Progetto

Si vuole realizzare un sistema nel quale il microcontrollore comunichi in modo bilaterale con un client mqtt tramite rete LoRa.
Il sistema genererà un numero casuale, basato sul risultato del lancio di n dadi. Il sistema gestirà la quantità dei dadi e il loro lancio. Il client provvederà un'interfaccia grafica interattiva, in cui sarà possibile visualizzare la quantità dei dadi, il risultato del loro lancio e cambiare lo stato del sistema in tempo reale, con dei pulsanti intuitivi.
Sono previste due modalità di funzionamento:

- una che si occuperà di gestire la quantità dei dadi: bisognerà inclinare la scheda verso destra o sinistra per incrementare o decrementare la quantità
- una che si occuperà del lancio dei dadi: per effettuare il lancio, bisognerà scuotere la scheda

Il uC leggerà la modalità dal cloud, e in base ad essa invierà dati diversi.
Il client riceverà i dati e in base alla modalità li processerà in modo diverso. Quando verrà premuto il tasto per cambiare modalità, il client invierà al cloud il valore della nuova modalità scelta.

2 Componenti utilizzati

Il progetto si basa su specifici componenti hardware:

- Microcontrollore STM32WL55JC, con protocollo multi-modulation radio LoRa
- Scheda Multi-Sensor X-NUCLEO-IKS01A3, per l'acqusizione dei sensori
  - Accelerometro e giroscopio con misurazione a 3 assi LSM6DSO

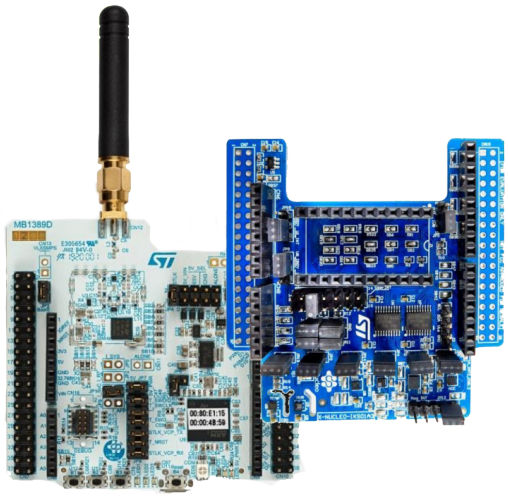

Figura 1: STM32WL55JC e X-NUCLEO-IKS01A3

Si utilizzano inoltre ulteriori componenti software:

- STM32CubeIDE per la scrittura del codice C riguardante il Microcontrollore
- Ambiente di sviluppo per il codice python riguradante il client
- Librerie di supporto pygame e mqtt per l'elaborazione grafica e per la connessione con il cloud

3 Software Microcontrollore

Il codice per la programmazione del microcontrollore si sviluppa dall'applicazione di base "*LoRaWAN_End_Node*" che fornisce le funzioni necessarie per l'inizializzazione dei componenti base per la comunicazione del microcontrollore attraverso il protocollo LoRa.

Per poter leggere i dati che si intendono acquisire dai sensori si è reso necessario organizzare e riformulare parte di codice per l'inizializzazione e il funzionamento dell'accelerometro e del giroscopio. Nelle sezioni 3.1 e 3.2 vengono rispettivamente presentate la strategia di collegamento e lettura dei sensori selezionati e le modifiche apportate al codice per poter comunicare correttamente con il cloud.

3.1 Lettura sensori

Per fare in modo che il microcontrollore riceva i dati acquisiti dai sensori, è stato effettuato il collegamento hardware tra i due componenti con i pin specifici della scheda (PA11 e PA12) adibiti alla comunicazione attraverso bus I2C. Una volta completato questo passaggio, con l'integrazione dei driver di basso livello del sensore X-CUBE-MEMS1 si è resa possibile l'effettiva comunicazione.

Per leggere i dati misurati dall'accelerometro e dal giroscopio, sono state sviluppate due funzioni in particolare (riportate in Figura 2). Queste due funzioni leggono i dati dell'accelerometro e del giroscopio rispettivamente e restituiscono un "exit code" uguale a zero se la lettura è stata completata con successo. I valori letti vengono scritti sulle tre variabili passate tramite puntatore.
Affinché queste funzioni operino correttamente, è necessario richiamare anche altre funzioni di inizializzazione:

- `LSM6DSO_USER_Init()` all'interno della funzione `LoRaWAN_Init` (dichiarata nel file *lora_app.c*)
- `BSP_I2C2_Init()` nel *main.c*

I dati che si intendono inviare al cloud sono racchiusi nella variabile `sensor_data` (presente in Figura 4) di tipo `sensor_t`. La struttura di `sensor_t`, riportata in Figura 3 è definita nel file *sys_sensor.h* ed è stata modificata aggiungendole i campi `acc_x`, `acc_y` e `gyr_y`, necessari al funzionamento del sistema.

La funzione `EnvSensorRead` (definita nel file *sys_sensor.c* e riportata in Figura 4) legge i dati dei sensori e li scrive nella variabile `sensor_data` (passata tramite puntatore). Questi dati saranno poi processati nella funzione `SendTxData` (vedi Sezione 3.2.1).

```c
int32_t LSM6DSO_USER_Acc_GetAxes(int32_t *px, int32_t *py, int32_t *pz)
{
    LSM6DSO16IS_Axes_t acc;
    int32_t ret;

    ret = LSM6DSO16IS_ACC_GetAxes(&LSM6DSO_OB_Handle, &acc);

    *px = acc.x;
    *py = acc.y;
    *pz = acc.z;

    return ret;

}

int32_t LSM6DSO_USER_Gyro_GetAxes(int32_t *px, int32_t *py, int32_t *pz)
{
    LSM6DSO16IS_Axes_t gyro;
    int32_t ret;

    ret = LSM6DSO16IS_GYRO_GetAxes(&LSM6DSO_OB_Handle, &gyro);

    *px = gyro.x;
    *py = gyro.y;
    *pz = gyro.z;

    return ret;
}
```

Figura 2: Funzioni di lettura dei sensori

```c
  */
typedef struct
{
  int32_t acc_x;
  int32_t acc_y;
  int32_t gyr_y;
  float temperature;
  float humidity;
  float pressure;
  int32_t latitude;       /*!< latitude converted to bina
  int32_t longitude;      /*!< longitude converted to bir
  int16_t altitudeGps;    /*!< in m */
  int16_t altitudeBar;    /*!< in m * 10 */
  /**more may be added*/
  /* USER CODE BEGIN sensor_t */

  /* USER CODE END sensor_t */
} sensor_t;
```

Figura 3: Struttura dati sensor_t

```
54 #endif /* USE_IKS01A3_ENV_SENSOR_LPS22HH_0 */
55 #else
56   TEMPERATURE_Value = (SYS_GetTemperatureLevel() >> 8);
57   LSM6DSO_USER_Gyro_GetAxes(&gyr_x, &gyr_y, &gyr_z);
58   LSM6DSO_USER_Acc_GetAxes(&acc_x, &acc_y, &acc_z);
59
60 #endif  /* SENSOR_ENABLED */
61
62   sensor_data->humidity    = HUMIDITY_Value;
63   sensor_data->temperature = TEMPERATURE_Value;
64   sensor_data->pressure    = PRESSURE_Value;
65
66   sensor_data->acc_x = acc_x;
67   sensor_data->acc_y = acc_y;
68   sensor_data->gyr_y = gyr_y;
```

Figura 4: Parte della funzione EnvSensorRead

3.2 LoRaWAN

Il uC, una volta connesso ad un gateway LoRa, invia i dati al cloud ogni 10 secondi (quantità definita dalla costante `APP_TX_DUTYCICLE` nel file *lora_app.h*) e legge i dati in entrata appena vengono caricati sul cloud dal client.
Un'idea per ottenere un interfaccia grafica e di conseguenza un programma più performante potrebbe essere quella di abbassare il dutycicle a pochi secondi.

3.2.1 Uplink scheda

Il uC invierà al cloud i valori presenti nella variabile `AppData.Buffer`, un array di tipo `uint8_t`, i cui valori degli elementi sono definiti nella funzione `SendTxData` (all'interno del file *lora_app.c*). Dato che i valori letti dai sensori (`acc_x`, `acc_y` e `gyr_y`) sono di tipo `int32_t` e gli elementi del buffer sono di tipo `uint8_t`, bisogna scrivere all'interno del buffer 8 bit alla volta partendo dai più significativi. Un esempio grafico di come i dati di tipo `int32_t` vengono suddivisi è riportato di seguito in Figura 5.

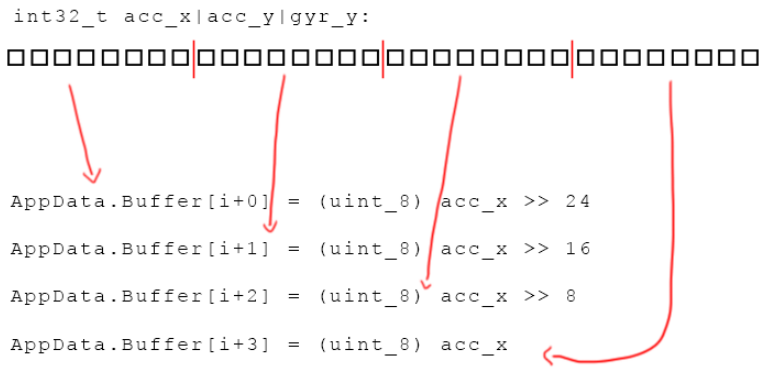

Figura 5: Come valori di tipo int32_t vengono suddivisi e scritti sul Buffer

Nella funzione `SendTxData` riportata in Figura 6, vengono invece definiti i valori degli elementi del buffer:

```
    acc_x = (int32_t)(sensor_data.acc_x);
    acc_y = (int32_t)(sensor_data.acc_y);
    gyr_y = (int32_t)(sensor_data.gyr_y);

    AppData.Buffer[i++] = AppLedStateOn;

    if (roll_state == 1)
    {
        AppData.Buffer[i++] = (uint8_t)((gyr_y >> 24) & 0xFF);
        AppData.Buffer[i++] = (uint8_t)((gyr_y >> 16) & 0xFF);
        AppData.Buffer[i++] = (uint8_t)((gyr_y >> 8) & 0xFF);
        AppData.Buffer[i++] = (uint8_t)(gyr_y & 0xFF);

    }else{

        AppData.Buffer[i++] = (uint8_t)((acc_x >> 24) & 0xFF);
        AppData.Buffer[i++] = (uint8_t)((acc_x >> 16) & 0xFF);
        AppData.Buffer[i++] = (uint8_t)((acc_x >> 8) & 0xFF);
        AppData.Buffer[i++] = (uint8_t)(acc_x & 0xFF);

        AppData.Buffer[i++] = (uint8_t)((acc_y >> 24) & 0xFF);
        AppData.Buffer[i++] = (uint8_t)((acc_y >> 16) & 0xFF);
        AppData.Buffer[i++] = (uint8_t)((acc_y >> 8) & 0xFF);
        AppData.Buffer[i++] = (uint8_t)(acc_y & 0xFF);
    }
```

Figura 6: Parte della funzione SendTxData

Il primo elemento del buffer (`AppData.Buffer[0]`) rappresenta lo stato del sistema e da esso dipenderanno i valori acquisiti dai sensori che verranno scritti sul buffer. Si utilizza la variabile `roll_state` per descrivere lo stato della modalità del sistema e il suo valore viene modificato quando l'uC riceve dati

dal cloud (Vedi Sezione 3.2.2):

- `roll_state` = 1 indica che la modalità attiva è quella del lancio dei dadi
  - il valore di `gyr_y` sarà scritto negli elementi di `AppData.Buffer[]` con indice tra 1 e 4.
- `roll_state` = 0 indica che la modalità attiva è quella di selezione della quantità dei dadi da lanciare
  - sul buffer verranno scritti i valori di `acc_x` e `acc_y` agli indici 1÷4 e 5÷8 di `AppData.Buffer[]` rispettivamente.

3.2.2 Downlink scheda

I dati provenienti dal cloud sono processati nella funzione `OnRxData` (all'interno del file *lora_app.c* e riportata in Figura 7).

```
if (appData->BufferSize == 1)
{
  AppLedStateOn = appData->Buffer[0] & 0x01;
  roll_state = appData->Buffer[0] & 0x01;
  if (AppLedStateOn == RESET)
  {
    APP_LOG(TS_OFF, VLEVEL_H, "LED OFF\r\n");
    HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, GPIO_PIN_RESET); /* LED_RED */
  }
  else //ROLL STATE FALSE
  {
    APP_LOG(TS_OFF, VLEVEL_H, "LED ON\r\n");
    HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, GPIO_PIN_SET); /* LED_RED */
  }
}
break;
```

Figura 7: Parte della funzione OnRxData

Qui viene letto l'elemento presente all'indice 0 del buffer, il quale indica la modalità di funzionamento del sistema, e viene scritto sulle variabili `roll_state` e `AppLedStateOn`. Quest'ultima determina l'accensione o lo spegnimento del LED_3 della scheda.

4 Client

L'applicazione lato client viene sviluppata con il linguaggio di programmazione python, utilizzando l'insieme di moduli Pygame per la realizzazione dell'interfaccia grafica. (Ulteriori informazioni su pygame al link https://www.pygame.org/)

Nel codice viene implementata la logica per garantire le funzionalità dell'applicazione, come l'interfaccia grafica, l'acquisizione dei dati dei sensori derivanti dal cloud, il cambiamento di stato e l'acquisizione degli input dell'utente.

Lo stato corrente viene mantenuto da specifiche variabili di stato, che vengono modificate nel momento in cui si verificano eventi gestiti dai moduli pygame (interazione del mouse con un pulsante presente sullo schermo). Inoltre, vengono mantenute ulteriori informazioni sui dati dell'applicazione in precise variabili:

```
roll_state = False      # variabile di stato corrente
dice_quantity = 1       # numero di dadi da lanciare
dice_state = []         # stato dei dadi usciti: es [6, 3, 1]
dice_generate = False   # lancia la funzione di generaz dadi
result = 0              # risultato finale
```

Nel complesso, il client realizzato permette di gestire la comunicazione bilaterale di dati tra il cloud e il microcontrollore, sfruttando il protocollo di comunicazione LoRa. Le funzioni che permettono la connessione del client con il cloud sono definite dal protocollo di messaggistica MQTT, progettato esclusivamente per fini legati all IoT. Di seguito si riportano brevemente le implementazioni software che riguardano la gestione della comunicazione **uplink** e **downlink**.

4.1 Ricezione dati: cloud-to-client

La gestione della comunicazione dei dati dal uC al cloud (e di conseguenza al client) lato software viene gestita attraverso la funzione di callback `on_message()`, richiamata ad ogni nuovo messaggio ricevuto dall'applicazione lato cloud. Partendo dal presupposto che il payload ricevuto dipende dallo stato corrente dell'applicazione, nella funzione di callback viene implementata la logica per l'elaborazione dei dati inviati dall' uC in base al valore della variabile di stato `roll_state`. Successivamente viene poi convertito il payload ricevuto in stringhe leggibili e stampato il valore della variabili, dopo i dovuti shift eseguiti sull'array contenente il payload, per ottenere correttamente i dati.

Di seguito si riporta l'esempio per l'acquisizione della coordinata y dell'accelerazione dal payload dati ricevuto in cloud e la sua successiva elaborazione, per cambiare lo stato corrente del sistema:

```
if roll_state == True:
    gyr_y = (((((message_bytes[1] << 24) | message_bytes[2] << 16)
     | message_bytes[3] << 8) | message_bytes[4] ) & 0xFFFFFFFF)

    if gyr_y & 0x80000000:
        gyr_y = gyr_y - 0x100000000
        print("gy: " + str(gyr_y) + "dice_generate: ", dice_generate)
        if abs(gyr_y) > 40000:
            dice_generate = True
```

I passaggi illustrati vengono analogamente replicati per l'acquisizione dei dati (`acc_x`, `acc_y`) relativi all'inclinazione della scheda quando `roll_state = False`, ovvero quando il sistema è nella modalità di scelta dei dadi.

Si può quindi affermare che la comunicazione uplink viene gestita interamente nella funzione asincrona di callback, ma ha influenza sullo stato corrente del sistema e quindi anche sulla visualizzazione grafica dell'interfaccia utente e sulla comunicazione downlink.

4.2 Invio dati: client-to-cloud

La gestione della comunicazione di dati dal client python al uC (passando per il cloud), è invece implementata nel ciclo infinito `while True` del programma, dove sono presenti anche le funzioni di pygame per la realizzazione grafica dell'interfaccia utente.

Lo stato corrente può essere cambiato in qualsialsi momento dall'utente che interagisce con il client, premendo con il mouse sul pulsante apposito contrassegnato con "Change mode". Nel momento in cui si rileva questo evento, viene generato il payload corrispondente allo stato che viene scelto dall'utente e successivamente inviato al cloud, in attesa che il microcontrollore lo riceva.

Di seguito è riportata la parte di codice in cui viene creato e inviato il payload al cloud per la comunicazione downlink:

```
if event.type == pygame.MOUSEBUTTONDOWN:
    if changeState_button.collidepoint(pygame.mouse.get_pos()):
        roll_state = not roll_state

        if roll_state == False:
            payload[0] = 0
            downlinks["downlinks"][0]["frm_payload"] =
```

```
            base64.b64encode(payload).decode('utf8')
    else:
        payload[0] = 1
        downlinks["downlinks"][0]["frm_payload"] =
            base64.b64encode(payload).decode('utf8')
    print("\r\n" + str(downlinks) + "\r\n")

    topic = "v3/" + CFG_APP_ID_AT_TTN + "/devices/" +
        CFG_DEVICE_ID + "/down/push"
    client.publish(topic, json.dumps(downlinks))
```

Come è possibile vedere dal codice, il payload da inviare al cloud viene organizzato nella struttura dati `downlinks` e successivamente trasformato in un json formattato in modo preciso per soddisfare la formattazione standard di un topic di tipo MQTT.

NOTA: Si è scelto di implementare la logica della comunicazione downlink in un ciclo temporizzato per 60 fotogrammi al secondo per esigenza di visualizzare in tempo reale i cambiamenti di stato nell'interfaccia grafica. Un'idea di ottimizzazione del progetto potrebbe essere quella di svincolare i messaggi verso il cloud relativi al cambiamento di stato dalla grafica e quindi ridurre la frequenza con cui questi possono essere potenzialmente inviati al cloud, dal momento che la frequenza di lettura dei dati del microcontrollore si aggira intorno a pochi secondi, non così frequente come il ciclo di aggiornamento della grafica.

4.3 Interfaccia grafica

L'interfaccia grafica viene implementata utilizzando i moduli per la generazione di elementi grafici messi a disposizione da pygame. La scelta stilistica di utilizzare pygame è giustificata dalla necessità di rendere l'esperienza dell'utente più intuitiva possibile. L'interfaccia grafica è realizzata in modo dinamico, per fare in modo che, al cambiamento di stato del sistema, si modifichi in tempo reale.

La grafica realizzata presenta un pulsante per poter passare dallo stato di scelta dei dadi allo stato di lancio e un pulsante per resettare il risultato e ritornare nella situazione iniziale, necessario per poter effettuare un nuovo lancio una volta ottenuto un risultato.

Di seguito è riportato un esempio dell'interfaccia grafica realizzata con le funzioni di pygame, in cui si mostra il risultato di un lancio dei dadi:

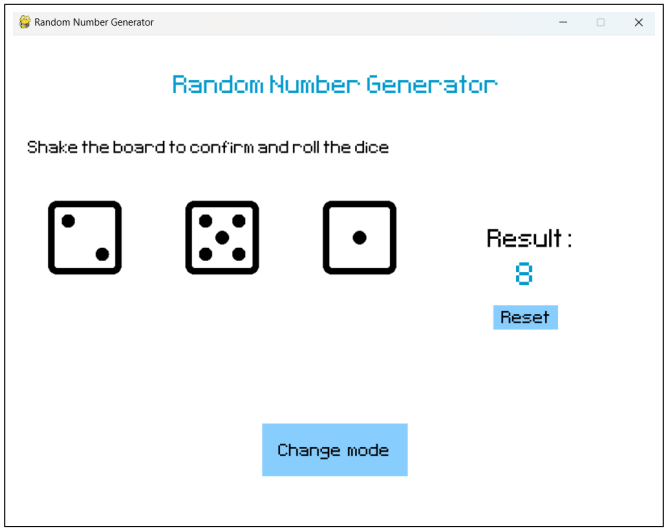

Figura 8: Interfaccia grafica dell'applicazione

Entrando nei dettagli della struttura del codice, l'interfaccia viene aggiornata con un massimo di 60 fotogrammi al secondo dalla funzione:

`clock.tick(60)`

In ciascuno di questi fotogrammi vengono generati gli elementi grafici attraverso le apposite funzioni di pygame:

- `pygame.draw.rect()` per disegnare sullo schermo gli oggetti di tipo Rect precedentemente creati (bottoni)
- `screen.blit()` per disporre sullo schermo i testi creati in precedenza. (screen è l'oggetto che identifica lo schermo)

Ad ogni fotogramma viene inoltre implementata una logica per la disposizione corretta dei dadi nello spazio, dipendente dal numero di dadi attualmente selezionati per il lancio espressi nella variabile `dice_quantity`. Per garantire una disposizione simmetrica dei dadi si immagina di suddividere lo spazio in una matrice 2 righe e 3 colonne da cui, con precise formule matematiche si può ricavare la coordinata precisa di generazione di ciascun dado:

```
for i in range(0, dice_quantity):
    n_cols = 3
    dice_x = 40 + (i % n_cols)*170 # 40 = x di partenza +
        (100 + 70 = dimensione dado + spazio tra i dadi)
    dice_y = 200 + (i // n_cols)*120
    if dice_state == []:
        screen.blit(dice_surfaces[0], (dice_x, dice_y))
    else:
        screen.blit(dice_surfaces[dice_state[i]], (dice_x, dice_y))
```

Nell'interfaccia grafica non vengono volutamente riportati i dettagli che riguardano la comunicazione tra il cloud e il client, ma solamente le funzionalità indispensabili per una buona esperienza utente.

5 Funzionamento

Per poter interagire con l'applicazione funzionante è necessario aver completato i seguenti step ordinatamente:

- compilazione ed esecuzione del codice relativo al microcontrollore
- attivazione di un gateway LoRa funzionante e disponibile per la connessione
- compilazione ed esecuzione del codice relativo al client utente python (RandomNumberGenerator.py)

Tutti i files con il codice necessario per il funzionamento dell'applicazione sono visualizzazbili e scaricabili dall'apposita repository GitHub al link `https://t.ly/G0qwD`

Una volta eseguito, il client python presenterà la seguente interfaccia utente:

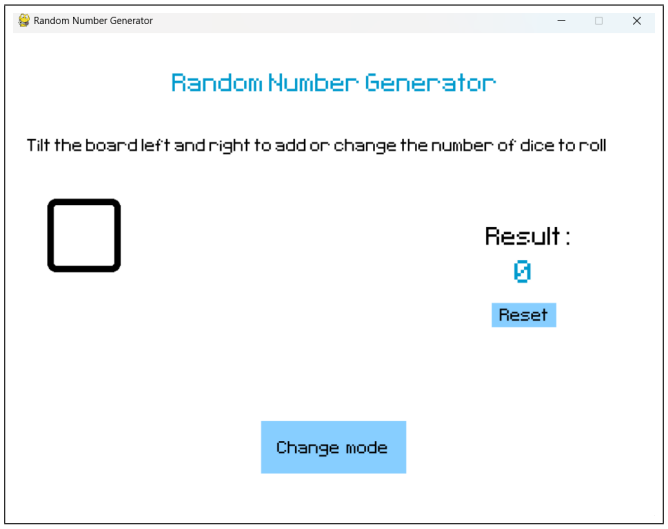

Figura 9: Schermata iniziale

La visualizzazione grafica cambia in base allo stato dell'applicazione, in continuo aggiornamento in base all'interazione dell'utente.
In particolare, all'avvio il sistema sincronizzerà lo stato iniziale della scheda con il client, lo stato di scelta del numero di dadi. In questo stato sarà possibile, così come da descrizione testuale, inclinare la scheda verso destra o verso sinistra per rispettivamente incrementare o decrementare il numero di dadi.
(NOTA: La velocità di risposta del client dipende dalla velocità di comunicazione impostata dal protocollo di comunicazione LoRa).

Interfaccia grafica nello **stato di scelta dei dadi**:

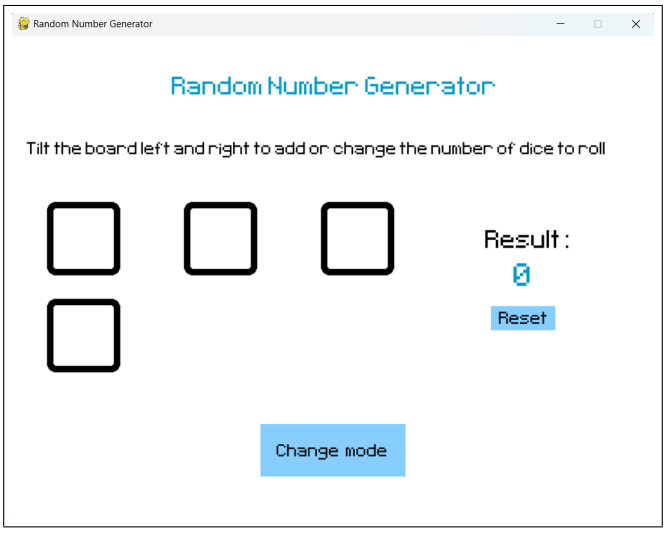

Figura 10: Schermata principale con 4 dadi selezionati

Il secondo stato assumibile dal sistema è quello di lancio dei dadi, in cui la descrizione testuale indica l'operazione da effettuare per ottenere il risultato randomico. Sarà quindi un movimento della scheda sufficientemente deciso che determinerà il valore di ciascun dado selezionato e la conseguente visualizzazione del risultato.

Interfaccia grafica nello **stato di lancio dei dadi**:

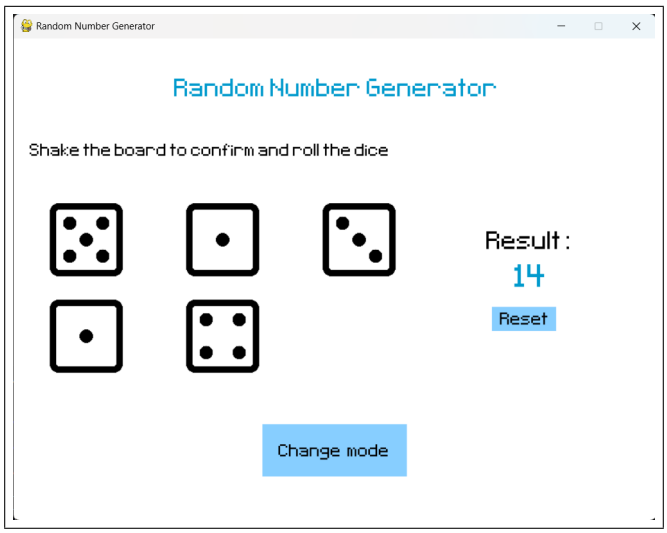

Figura 11: Schermata principale in seguito al lancio dei dadi

Nell'interfaccia grafica ci sono due bottoni con cui è possibile interagire:

- "Change mode" button, cambia lo stato del sistema
- "Reset" button, resetta il client e lo riporta alla schermata iniziale

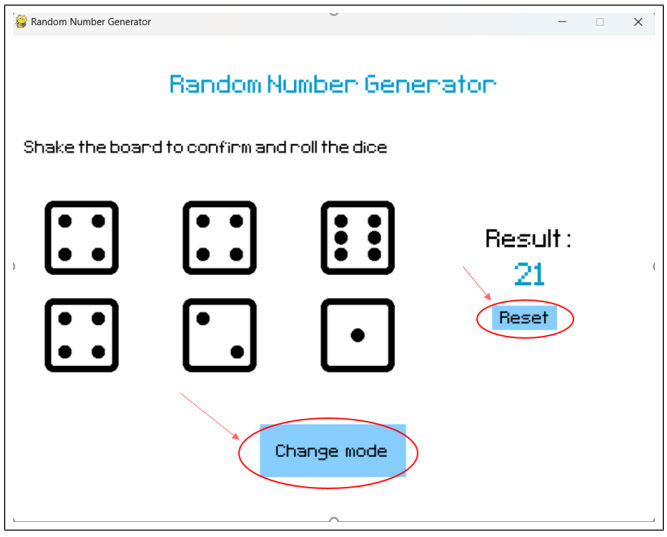

Figura 12: Bottoni nella schermata principale

La console del programma client python permette di visualizzare in tempo reale eventuali cambiamenti di stato dovuti alla pressione del bottone apposito, mostrando il payload inviato al cloud e i dettagli della comunicazione downlink tra il client, il cloud e il microcontrollore.

```
{'downlinks': [{'f_port': 2, 'priority': 'NORMAL', 'frm_payload': 'AA=='}]}

topic:
    v3/rev-gallo-application@ttn/devices/eui.......
payload:
    ...
```

Analogamente, la console mostra la corretta ricezione del payload inviato dal uC al client, ad ogni nuova notifica di comunicazione uplink, mostrando a video i dati che permettono l'elaborazione vera e propria da parte del programma, formattati in modo leggibile.

È possibile consultare lo stato del sistema anche direttamente sulla scheda. Se il LED contrassegnato con il numero 3 sulla scheda principale è acceso significa che il sistema è nello stato di selezione del numero di dati, se spento lo stato sarà quello del lancio dei dadi.