Manuale Tecnico

Bomba Logica

Università degli studi dell'Insubria:
Autori: Frigoli Matteo

Sommario

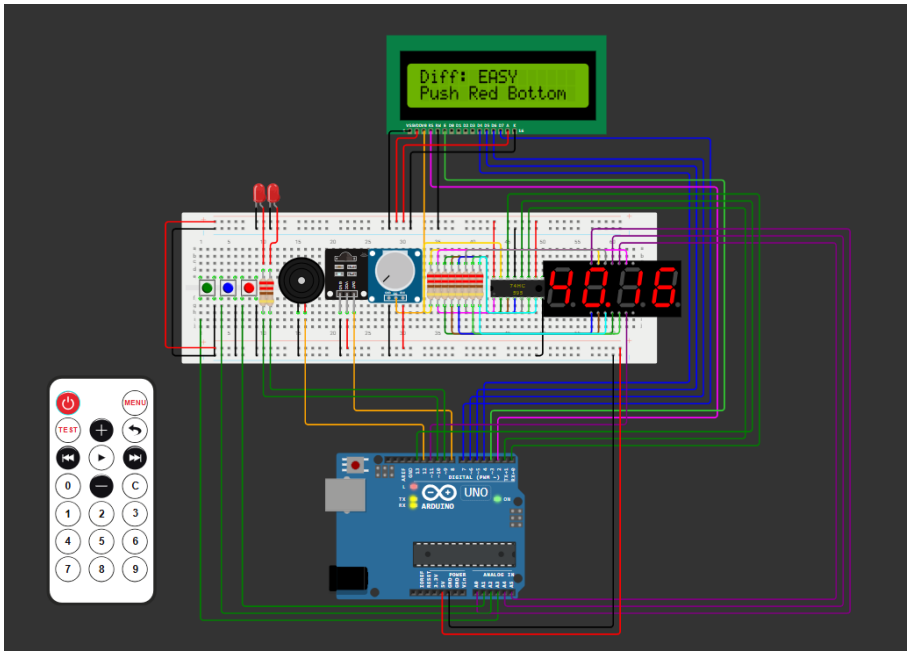
Scopo del documento

Questo manuale descrive la struttura hardware e la logica software del progetto “Bomba Logica con Arduino UNO”. Il documento è rivolto a docenti, valutatori e sviluppatori che desiderano comprendere o modificare il funzionamento interno del sistema.

Panoramica hardware

Componenti principali

- Arduino UNO R3
- LCD 16x2 in modalità 4-bit (pin RS, E, D4–D7)
- Display 7-segmenti 4 digit (catodo comune), pilotato tramite:
    - 1 IC 74HC595 per i segmenti
    - 4 pin digitali per la selezione delle cifre
- Ricevitore IR compatibile con libreria IRremote
- Telecomando IR con tasti numerici, frecce, *, #, OK
- Buzzer attivo
- 2 LED rossi per indicare gli errori disponibili
- 3 pulsanti per il taglio dei fili (rosso, blu, verde)
- Breadboard, resistenze, cablaggio vario



Architettura software

Struttura generale

Il programma è organizzato in:

- Configurazione hardware (pin, tipo di display, mapping segmenti).
- Strutture dati per rappresentare gli enigmi.
- Variabili di stato per il gameplay.
- Funzioni di utilità per:
    - beep,
    - gestione LCD,
    - gestione 7-segmenti,
    - interpretazione dei codici IR.
- Funzioni di logica di gioco:
    - applicazione difficoltà,
    - gestione errori,
    - gestione MISTERY,
    - calcolo filo corretto.
- Loop principale:
    - polling IR,
    - aggiornamento 7-seg in multiplex,
    - logica delle fasi,
    - gestione timer e turbo countdown.

Principali moduli logici

1. Modulo enigmi (Puzzle)
2. Modulo UI (LCD + 7-seg)
3. Modulo IR input
4. Modulo game state (phase, difficulty, errors)
5. Modulo timer e suoni
6. Modulo decisione filo (targetWireFromSum)

Strutture dati e stati

Struttura Puzzle

```
struct Puzzle {
  const char* expr;
  const char* solution;   // 1 o 2 cifre, senza zeri davanti
};
```

- expr: stringa da visualizzare nella prima riga LCD.
- solution: risultato corretto in formato stringa (usato sia per confronto che per conversione numerica).

Tre array globali:

- easyPool[], normalPool[], hardPool[].
- Ogni array ospita un certo numero di enigmi predefiniti.
- Le dimensioni sono calcolate con sizeof(...) / sizeof(Puzzle).

In ogni nuova partita:

- chooseRandomPuzzles() seleziona due enigmi casuali p1 e p2, senza ripetizioni.

Stati di gioco

```
enum Phase { PHASE_PUZZLE1, PHASE_PUZZLE2, PHASE_MISTERY, PHASE_WIRE };
Phase phase = PHASE_PUZZLE1;

enum Difficulty { EASY=1, NORMAL=2, HARD=3 };
Difficulty difficulty = NORMAL;
```

- phase indica quale sotto-logica è attiva nell'handler IR e nel loop.
- difficulty viene modificata in idle con le frecce SU/GIÙ.

Variabili chiave:

- errors, maxErrors
- gameActive, gameOver
- inputCode
- timeLeft (centesimi di secondo)
- speedFactor, speedStep

Fase MISTERY

Variabili specifiche:

- mysteryStartTime: per il timeout dei 5 secondi.
- lastMysterySeconds: per ridisegnare il countdown su LCD solo quando necessario.
- mysteryUsed: evita che il bonus/trap venga attivato più di una volta.
- mysteryBonusKey, mysteryTrapKey: mappano in modo dinamico, ad ogni partita, quale tasto IR corrisponde al bonus e quale alla penalità.

Gestione input IR

La libreria IRremote viene utilizzata tramite:

- IrReceiver.begin(IR_PIN, ENABLE_LED_FEEDBACK); in setup().
- IrReceiver.decode() nel loop().

Alla decodifica:

- si legge decodedRawData.
- lo si confronta con costanti IR_0 … IR_9, IR_HASH, IR_STAR, IR_UP, IR_DOWN, ecc.

La logica dei tasti è:

- IR_HASH (#):
    - in idle / game over → resetGame().
    - a partita in corso → goToIdleFromReset() (RUN AWAY MODE).
- IR_UP / IR_DOWN:
    - in idle → modificano difficulty e richiamano showIdleScreen().
- IR_STAR (*):
    - in fase enigmi → pulisce la risposta (inputCode = "").
- 0–9:
    - in fase enigmi → input numerico.
    - confronto automatico quando raggiunta la lunghezza della soluzione.
- IR_4 / IR_5:
    - in fase MISTERY → attivano BONUS o TRAP, a seconda della randomizzazione.

Timer, countdown e turbo beep

Rappresentazione del tempo

timeLeft è un long che rappresenta **centesimi di secondo**:

- es. 4500 = 45.00 secondi.

In applyDifficulty():

- per EASY: baseSeconds = 59 → timeLeft = 5900.
- per NORMAL: baseSeconds = 45 → 4500.
- per HARD: baseSeconds = 30 → 3000.

Aggiornamento del timer

Nel loop():

```
unsigned long interval = (unsigned long)(10.0 / speedFactor);
if (now - lastTick >= interval) {
  lastTick = now;
  timeLeft--;
  ...
}
```

- Si parte da un tick base di 10 ms (centi-secondo).
- Ad ogni errore speedFactor aumenta, riducendo interval e accelerando la discesa.

Gestione del beep (turbo countdown)

- beepCounter viene incrementato ad ogni decremento di timeLeft.
- Se timeLeft > 300 (3 secondi):
    - beep ogni 100 tick (~1 s).
- Se timeLeft <= 300:
    - beep ogni 10 tick (~0.1 s).

Questo crea un effetto tensione crescente nel finale.

Fase MISTERY – logica tecnica

Entrando in MISTERY (alla fine del secondo enigma corretto):

- phase = PHASE_MISTERY;
- mysteryUsed = false;
- mysteryStartTime = millis();
- lastMysterySeconds = 5;
- randomizzazione:

```
if (random(0, 2) == 0) {
  mysteryBonusKey = IR_4;
  mysteryTrapKey  = IR_5;
} else {
  mysteryBonusKey = IR_5;
  mysteryTrapKey  = IR_4;
}
```

- viene chiamata showMysteryScreen(lastMysterySeconds).

Nel loop():

1. Se l'utente preme mysteryBonusKey o mysteryTrapKey **prima** dei 5 secondi:
    - mysteryUsed = true;
    - blocchi if (code == mysteryBonusKey) { ... } else { ... } gestiscono:
        - animazioni LCD (flash, barre),
        - beep,
        - aggiornamento timeLeft (+1000 o −500 centesimi).
    - setDisplayFromTime(timeLeft);
    - reset di lastTick e beepCounter;
    - phase = PHASE_WIRE; e showWireScreen();.
2. Indipendentemente dall'input, un secondo blocco:

```
unsigned long elapsed = millis() - mysteryStartTime;
int remaining = 5 - (elapsed / 1000);
...
if (elapsed >= 5000) {
  phase = PHASE_WIRE;
  showWireScreen();
}
```

gestisce il countdown su LCD e il timeout.

Regola del filo e decisione finale

La funzione targetWireFromSum() implementa la regola globale:

- s1 = atoi(p1.solution);
- s2 = atoi(p2.solution);
- sum = s1 + s2;
- lastDigit = sum % 10;

Mappatura:

- lastDigit == 0 o 5 → return 2; (filo verde).
- lastDigit pari → return 0; (filo rosso).
- altrimenti → return 1; (filo blu).

Nel loop(), in PHASE_WIRE:

- si leggono i pulsanti con digitalRead(wirePins[b]) == LOW.
- se b == target → winGame();
- altrimenti → loseGame("WRONG WIRE!");.

Gestione errori, esplosione e disinnesco

registerError()

- Incrementa errors.
- Spegne uno dei LED errore.
- Aumenta speedFactor.
- Reset della risposta (inputCode = "") e dell'LCD di input.
- Se errors >= maxErrors → loseGame("TOO MANY ERRORS").

loseGame()

- Imposta gameOver = true, gameActive = false.
- Chiama strobeBoom():
    - lampeggio di "BOOOOM!!!" su LCD,
    - beep ripetuti.
- Mostra la schermata finale con:

    - BOOOOM!!! in riga 1,
    - motivo (es. TIME UP!, WRONG WIRE!, TOO MANY ERRORS) in riga 2.
- Spegne i LED errori.

winGame()

- Imposta gameOver = true, gameActive = false.
- Chiama animateDisarm() (animazione e beep).
- Mostra:
    - BOMB DISARMED
    - GOOD JOB!
- Spegne i LED errori.
- Emmette tre beep brevi.

Reset e “Run Away Mode”

Il tasto IR_HASH (#) gestisce:

- In idle / dopo fine partita:
    - resetGame() → nuova partita.
- In partita attiva:
    - goToIdleFromReset():
        - azzera gli stati,
        - spegne i LED,
        - mostra RUN AWAY MODE,
        - ritorna a showIdleScreen().

Questo permette di:

- testare rapidamente il gioco,
- uscire da situazioni senza via d’uscita,
- simulare un “abbandono campo” del giocatore.