FORTH MEMORY GAME

EMBEDEDD SYSTEMS

PROFESSORE
DANIELE PERI

STUDENTESSA
VALERIA RITA ORLANDO

Indice

1. Descrizione del progetto

Il progetto consiste nella realizzazione del gioco della memoria. Il sistema è stato realizzato a scopo didattico mediante l'uso del target Rapsberry Pi 4B e di diversi LED e pulsanti. Il linguaggio di programmazione usato è Forth, con pijFORTHos come interprete.

Il gioco inizia digitando il comando **"START_GAME"** nel terminale e si conclude quando il giocatore vince o perde.

Le dinamiche sono semplici:

1. **Avvio del gioco:**
    - All'inizio del gioco, si accendono **3 LED colorati** in una sequenza casuale.
2. **Input del giocatore:**
    - Il giocatore deve replicare la sequenza dei colori utilizzando **i tasti corrispondenti ai colori dei LED**.
    - Il sistema verifica l'input del giocatore.
3. **Condizioni di vittoria e sconfitta:**
    - Se il giocatore inserisce una **sequenza errata**, un segnale acustico (buzzer) indica la **sconfitta** ed il gioco termina.
    - Se invece la sequenza è **corretta**, il gioco prosegue, aumentando il livello di difficoltà.
4. **Avanzamento del gioco:**
    - Ad ogni round, la sequenza di LED viene incrementata di un elemento (ad esempio, da 3 a 4 LED), fino a raggiungere un massimo di **10 LED accesi**.
    - Se il giocatore replica correttamente anche l'ultima sequenza con 10 LED, il gioco termina con la **vittoria del giocatore**.

2. Componenti del progetto

2.1. Componenti Hardware

I componenti elettronici utilizzati per la realizzazione sono:

- Raspberry Pi 4B
- Modulo CP2102 USB-to-TTL per la comunicazione UART
- Buzzer acustico
- LED rosso
- LED blu
- LED giallo
- Nr. 3 Pulsanti con tappi colorati
- Cavi jumper
- Resistori
- Breadboard
- Estensione T-Cobbler

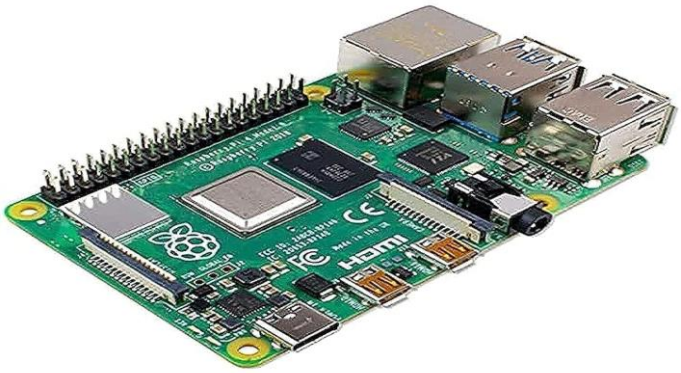
2.2. Componenti Software

Per la programmazione del dispositivo target è stato usato il sistema operativo con interprete pijFORTHos che permette l'interazione a basso livello con l'hardware del dispositivo mediante il linguaggio di programmazione Forth.

Per l'interazione con il dispositivo ed il caricamento del codice è stata instaurata una comunicazione seriale mediante il software ZOC8 Terminal da Windows.

3. Descrizione componenti

3.1. Raspberry Pi 4 B



Il **Raspberry Pi 4 B** è una scheda computer a singolo circuito altamente versatile, dotata di diverse interfacce che lo rendono ideale per una vasta gamma di applicazioni. La scheda è basata sul ***BCM2711***, un **system-on-a-chip (SoC)** prodotto da Broadcom, che integra un processore ARM ad alte prestazioni.

Tra le sue caratteristiche principali, il Raspberry Pi 4 B include un **header GPIO** con **40 pin** configurabili per diverse funzionalità:

- **Pin di alimentazione:** comprendono pin a **3.3V**, **5V** e **GND**.
- **GPIO generici:** pin di uso generale, configurabili ad esempio come **input** o **output** per il controllo e la lettura di dispositivi esterni.
- **GPIO specializzati:** pin dedicati a protocolli di comunicazione come **UART**, **I2C** e **SPI**, che consentono di collegare sensori, display ed altri moduli.

Nel nostro caso, il Raspberry Pi 4 B è stato utilizzato per eseguire il **codice di controllo del gioco della memoria**, sfruttando la sua capacità di gestire segnali digitali in modo rapido ed efficiente.

3.2 Modulo CP2102 USB-to-TTL per la comunicazione UART

Il modulo **CP2102 USB-to-TTL** si basa sul chip CP2102 di Silicon Labs. Questo modulo consente di stabilire una connessione UART (Universal Asynchronous Receiver/Transmitter) tramite una porta USB, presentandosi al computer come una porta COM standard.

Caratteristiche principali:

- Si collega al bus USB tramite un connettore USB femmina.
- Funziona come un'interfaccia per comunicare con il dispositivo di destinazione (target) tramite segnali seriali.

Per stabilire una connessione, è sufficiente:

1. Collegare il GND del modulo al GND del dispositivo target.
2. Collegare la linea di trasmissione (Tx) del modulo alla linea di ricezione (Rx) del target.
3. Collegare la linea di ricezione (Rx) del modulo alla linea di trasmissione (Tx) del target.

Questo modulo è particolarmente utile per il debug, la programmazione o la comunicazione con dispositivi che supportano la trasmissione seriale UART.

3.3 Buzzer acustico

Un **buzzer attivo** è un dispositivo elettronico che, quando alimentato, emette un suono ad una frequenza preimpostata grazie al suo **circuito oscillatore interno**. Questo circuito genera automaticamente la frequenza del suono, eliminando la necessità di segnali di controllo esterni.

Può essere quindi collegato direttamente ad una fonte di alimentazione o controllato con un interruttore o un pin digitale. Funziona a tensioni di alimentazione che variano da 3V a 12 V. Per proteggere il circuito e limitare la corrente viene utilizzata una resistenza da 220 Ohm.

Nel nostro caso è stato utilizzato per segnalare l'errore dell'utente e quindi il termine del gioco.

3.4 LED rosso, blu e giallo

Il sistema comprende **tre LED**: **rosso**, **blu** e **giallo**, ciascuno collegato in serie ad una **resistenza da 220 Ohm** per limitare la corrente ed evitare danni ai componenti. I LED rappresentano la **componente fondamentale** del gioco.

Si accendono in modo **casuale** (randomico) per mostrare al giocatore la **sequenza da replicare**. Si accendono inoltre quando il giocatore preme il **pulsante del colore corrispondente**, fornendo un feedback visivo immediato.

3.5 Pulsanti con tappi rosso, blu e giallo

I **pulsanti** di input sono stati collegati al Raspberry Pi 4 B attraverso delle resistenze di pull-up. Questo significa che quando il pulsante non viene premuto, la GPIO a cui è collegato il pulsante leggerà il valore di tensione Vcc, ovvero una logica 1. Quando il pulsante viene premuto, la GPIO leggerà il valore di tensione Gnd, ovvero una logica 0. In questo modo, il Raspberry Pi può leggere lo stato del pulsante e agire di conseguenza. L'utilizzo di resistenze di pull-up per interfacciare i pulsanti con il Raspberry Pi è una scelta comune in molte applicazioni poiché offre un modo semplice ed efficiente per leggere lo stato degli input.

Le resistenze di pull-up sono di solito di valore elevato, in modo da garantire che la tensione in ingresso sia sempre stabile e non dipenda dalla resistenza del circuito esterno. In questo modo, si garantisce una lettura precisa dello stato del pulsante.

3.6 Resistori

Sono stati utilizzati diversi ***resistori*** per LED, buzzer e tasti. Per quanto riguarda i LED ed il buzzer i resistori utilizzati sono da 220 Ohm. Per i tasti sono stati utilizzate resistenza da 10K Ohm.

3.7 Estensione T-Cobbler

E' una estensione Adafruit progettata appositamente per il Raspberry . Grazie al connettore a 40 pin può supportare GPIO, I2C e SPI su una breadboard senza saldatura.

4. Assegnazioni GPIO e Pinout

PINOUT RASPBERRY PI 4 B

	I2C SDA	*3v3 Power*	*1*	*2*	*5v Power*		
	I2C SCL	*GPIO 2*	*3*	*4*	*5v Power*		
	GPCLK0	*GPIO 3*	*5*	*6*	*GND*		**GND**
		GPIO 4	*7*	*8*	*GPIO 14*	UART Tx	**UART Rx**
		GND	*9*	*10*	*GPIO 15*	UART Rx	**UART Tx**
Buzzer		*GPIO 17*	*11*	*12*	*GPIO 18*	PCM CLK	**LED rosso**
		GPIO 27	*13*	*14*	*GND*		
Tasto blu		*GPIO 22*	*15*	*16*	*GPIO 23*		**LED giallo**
		3v3 Power	*17*	*18*	*GPIO 24*		
	SPI0 MOSI	*GPIO 10*	*19*	*20*	*GND*		
	SPI0 MISO	*GPIO 9*	*21*	*22*	*GPIO 25*		
	SPI0 SCLK	*GPIO 11*	*23*	*24*	*GPIO 8*	SPI0 CE0	
		GND	*25*	*26*	*GPIO 7*	SPI0 CE1	
	EEPROM SDA	*GPIO 0*	*27*	*28*	*GPIO 1*	EEPROM SCL	
Tasto giallo		*GPIO 5*	*29*	*30*	*GND*		
Tasto rosso		*GPIO 6*	*31*	*32*	*GPIO 12*	PWM0	
	PWM1	*GPIO 13*	*33*	*34*	*GND*		
	PCM FS	*GPIO 19*	*35*	*36*	*GPIO 16*		**LED blu**
		GPIO 26	*37*	*38*	*GPIO 20*	PCM DIN	
		GND	*39*	*40*	*GPIO 21*	PCM DOUT	

Sono state utilizzate le alimentazioni a 3.3 Volt per i tasti ed il buzzer mentre l'alimentazione 5 Volt per i LED .

	GPIO
LED ROSSO	GPIO 18
LED GIALLO	GPIO 23
LED BLU	GPIO 16
TASTO ROSSO	GPIO 6
TASTO GIALLO	GPIO 5
TASTO BLU	GPIO 22
BUZZER ATTIVO	GPIO 17
UART Tx e Rx	GND – GPIO 14 e GPIO 15

5. Procedura di installazione

5.1. Generalità

Prima di poter effettivamente utilizzare il Sistema sono necessari una serie di passaggi per la preparazione dell'ambiente di sviluppo sia del target che del PC che utilizzeremo per effettuare la programmazione Interattiva utilizzando il protocollo di comunicazione UART.

Avendo utilizzato un Raspberry Pi 4 B , è necessario formattare la micro SD che conterrà il sistema operativo pijFORTH. Per fare ciò è necessario usare il software Raspberry Pi Imager ed installare Raspberry OS Lite. Una volta terminata la procedura di formattazione, accedendo alla directory della micro SD, bisognerà eseguire i seguenti passaggi:

1. Eliminare tutti i file denominati come "kernelX.img" e copiare il file "kernel7.img", ridenominazione di pijForth OS.

2. Modificare il file "config.txt" aggiungendo in coda la stringa "enable uart=1" e salvare le modifiche. A questo punto il dispositivo di target è pronto per comunicare attraverso il protocollo UART.

L'Ambiente utilizzato in questo progetto è PijForthOs. PijForthOS è un ambiente di sviluppo Forth progettato appositamente per il Raspberry Pi. Forth è un linguaggio di programmazione compatto e veloce che si presta bene alla programmazione embedded, ovvero alla programmazione di sistemi a basso livello, come i microcontrollori e i dispositivi embedded. Utilizzando un ambiente di sviluppo come PijForthOS, si possono sfruttare appieno le potenzialità del Raspberry Pi, programmando il sistema a basso livello e interagendo direttamente con l'hardware.

Per fare in modo che Target e PC possano comunicare, è necessario installare un software che permette la comunicazione seriale tramite protocollo UART. Per le piattaforme Windows e MacOS è possibile utilizzare ZOC, mentre per quelle Linux è possibile utilizzare PicoCom-3.1. In questa documentazione verrà illustrata la procedura su piattaforma Windows.

5.2. ZOC 8

Il software ZOC8 è un emulatore di terminale seriale che permette di comunicare con l'interfaccia UART del Raspberry Pi. ZOC8 consente di stabilire una connessione seriale tra il computer ed il Raspberry Pi, utilizzando un cavo USB-to-TTL o un adattatore seriale. In questo modo, è possibile interagire con il Raspberry Pi come se si stesse utilizzando

un terminale seriale, inviando comandi e ricevendo risposte dal sistema. L'utilizzo di ZOC8 semplifica la programmazione del Raspberry Pi in quanto consente di interagire con il sistema in modo diretto e immediato. Ad esempio, è possibile accedere alla console del Raspberry Pi e interagire con il sistema operativo, eseguire comandi, visualizzare output e risolvere eventuali problemi di configurazione.

Per configurare la connessione tra il Raspberry Pi 4 B e l'interfaccia UART CP2102 utilizzando il software ZOC8, è necessario seguire i seguenti passaggi:

1. Collegare l'interfaccia UART al Raspberry Pi 4 B attraverso uno dei suoi connettori USB.
2. Accendere il Raspberry Pi 4B e avviare il software ZOC8 sul computer esterno.
3. Selezionare la porta seriale corretta all'interno delle impostazioni di connessione di ZOC8. Nel nostro caso, dovremo selezionare la porta seriale associata all'interfaccia UART CP2102.
4. Impostare la velocità di trasmissione dei dati sulla stessa velocità utilizzata dal Raspberry Pi 4 B. Di solito, la velocità di default è di 115200 bps.
5. Impostare il segnale RTS su "Off" e il segnale DTR su "Off" nelle impostazioni della porta seriale. Questo è necessario per garantire che il Raspberry Pi 4B riceva i comandi correttamente.
6. Confermare le impostazioni di connessione e premere il pulsante "Connetti" per stabilire la connessione tra il Raspberry Pi 4 B e il computer esterno.

Una volta stabilita la connessione, sarà possibile utilizzare il terminale di ZOC8 per interagire con il Raspberry Pi 4 B e caricare il codice Forth all'interno dell'interprete PijForthOS.

5.3. Setup del gioco

E’ necessario effettuare gli appositi collegamenti dell’hardware, i LED, i pulsanti ed il buzzer. Prima di procedere con la configurazione del software, è necessario installare l'interprete PijForthOS all'interno della microSD del Raspberry Pi e successivamente utilizzare il software ZOC8 per comunicare con il dispositivo.

Successivamente può essere caricato il codice che contiene le funzioni necessarie per il funzionamento del gioco inclusi i controlli GPIO per i pulsanti di input. É sufficiente, infine, digitare “START_GAME” per fare partire il gioco.

6. Word principali del gioco

Il codice è costituito principalmente da tre word e da un insieme di word di utilità. In particolare:

- ***BUTTON-CHECK, RESETBUTTON, BUZZER, BUTTON*** gestiscono il controllo del pulsante e l'interazione con l'utente.
- ***RANDOM*** genera un numero casuale che viene utilizzato per determinare quale LED accendere nel gioco. Per generare il numero casuale, la word RANDOM legge il valore del timer CLO e ne calcola il modulo 3 ottenendo sempre un numero compreso tra 0 e 2.
- ***LOOP_LED*** è definita da due cicli ed un costrutto CASE e gestisce l'accensione della sequenza di LED. Ad ogni round, genera una sequenza di LED da accendere in modo casuale (rosso, giallo, blu), con ogni LED che rimane acceso per 1 secondo prima di spegnersi e accendere il successivo. Il ciclo continua finchè la sequenza non è completa per il round corrente (che parte da 3 e aumenta ogni qualvolta l'utente replica correttamente la sequenza). Nella word LOOP_LED utilizziamo lo stack di ritorno per memorizzare il valore di RANDOM. Ciò è fondamentale per invertire l'ordine dei LED rispetto l'ordine di accensione. In pratica:
    1. Durante il primo ciclo i numeri vengono duplicati e memorizzati nello stack di ritorno nell'ordine in cui sono generati.
    2. Nella cima dello stack è presente il valore che genera l'accensione di un LED e viene consumato dal costrutto CASE. Si prosegue poi con gli altri LED che man mano vengono accesi. Alla fine del primo BEGIN-WHILE-REPEAT nello stack di ritorno saranno presenti nelle ultime N posizioni i valori corrispondenti ai LED accesi, mentre lo stack sarà libero.
    3. Nel secondo ciclo BEGIN-WHILE-REPEAT, dallo stack di ritorno vengono spostati gli ultimi M valori (con M=N). Di conseguenza, nello stack saranno presenti i valori corrispondenti ai LED accesi, in ordine inverso.

  Esempio: supponiamo che la sequenza di accensione dei LED sia: **rosso** (**0**), **blu** (**2**) e **giallo** (**1**). Dopo il secondo ciclo BEGIN-WHILE-REPEAT sullo stack saranno presenti i valori **1 2 0**.

  Quando l'utente, attraverso la word READ_PRESSED_BUTTON replicherà la sequenza dei LED accesi e premerà il pulsante *rosso,* esso lascerà nello stack il valore **0**. A questo punto, avviene il confronto tra gli ultimi due elementi in cima allo stack per verificare se il giocatore ha premuto il pulsante del colore del LED corretto. Il confronto consumerà entrambi i valori in cima allo stack. Se il confronto risulta essere positivo, quindi i due elementi sono uguali, il giocatore potrà inserire le restanti sequenze di LED da replicare attraverso i pulsanti. Altrimenti, il gioco termina.

Come si può notare da questo esempio semplice, l'utilizzo dello stack di ritorno è stato fondamentale per poter implementare un confronto tra valori in cima allo stack.

Sequenza generata da RANDOM 0-2-1

RANDOM-DUP

Stack *Stack di ritorno*

>R

Stack *Stack di ritorno*

Dopo il costrutto CASE

Stack *Stack di ritorno*

Ripeto per altri due LED

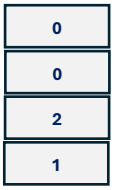
Stack *Stack di ritorno*

R> per tutti e tre i LED

Stack *Stack di ritorno*

Quando viene premuto il pulsante rosso

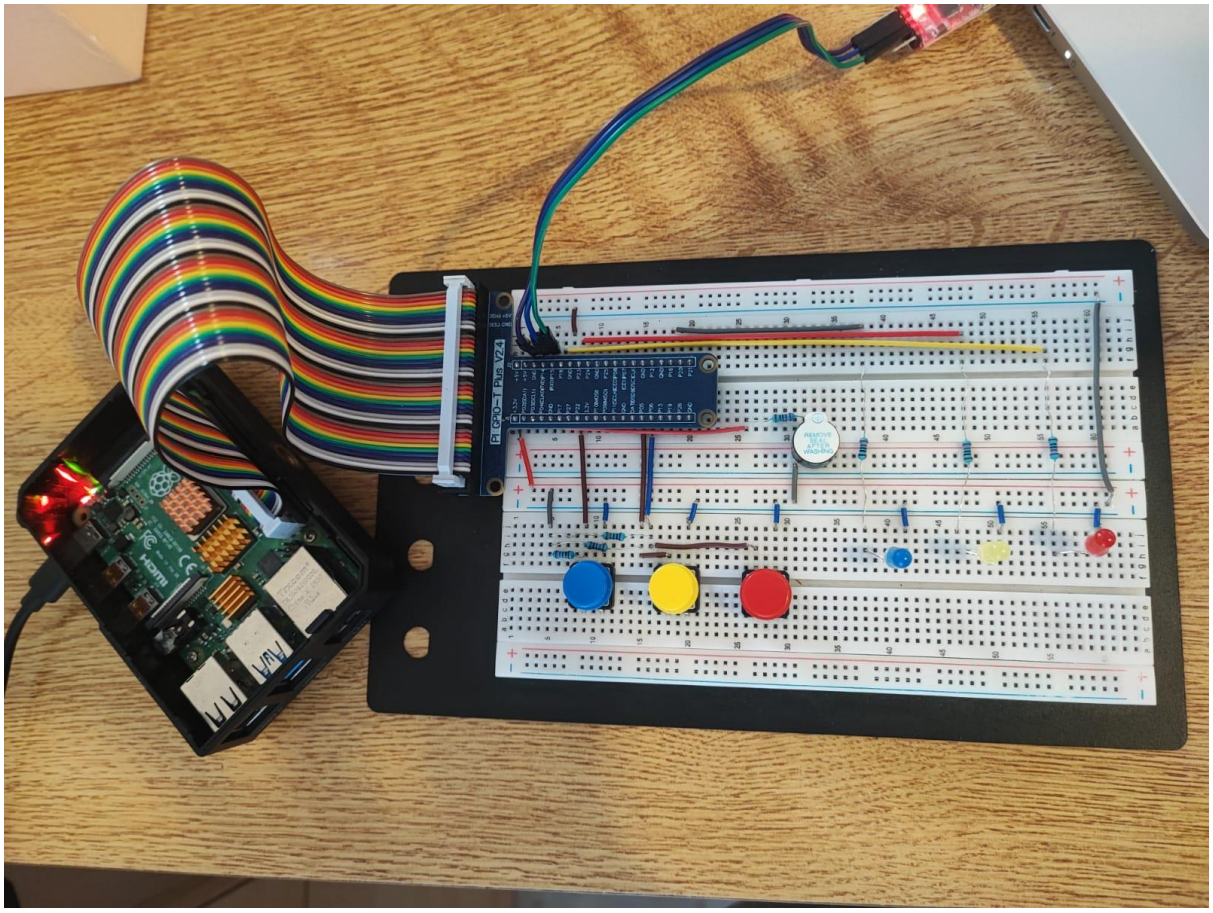
Stack

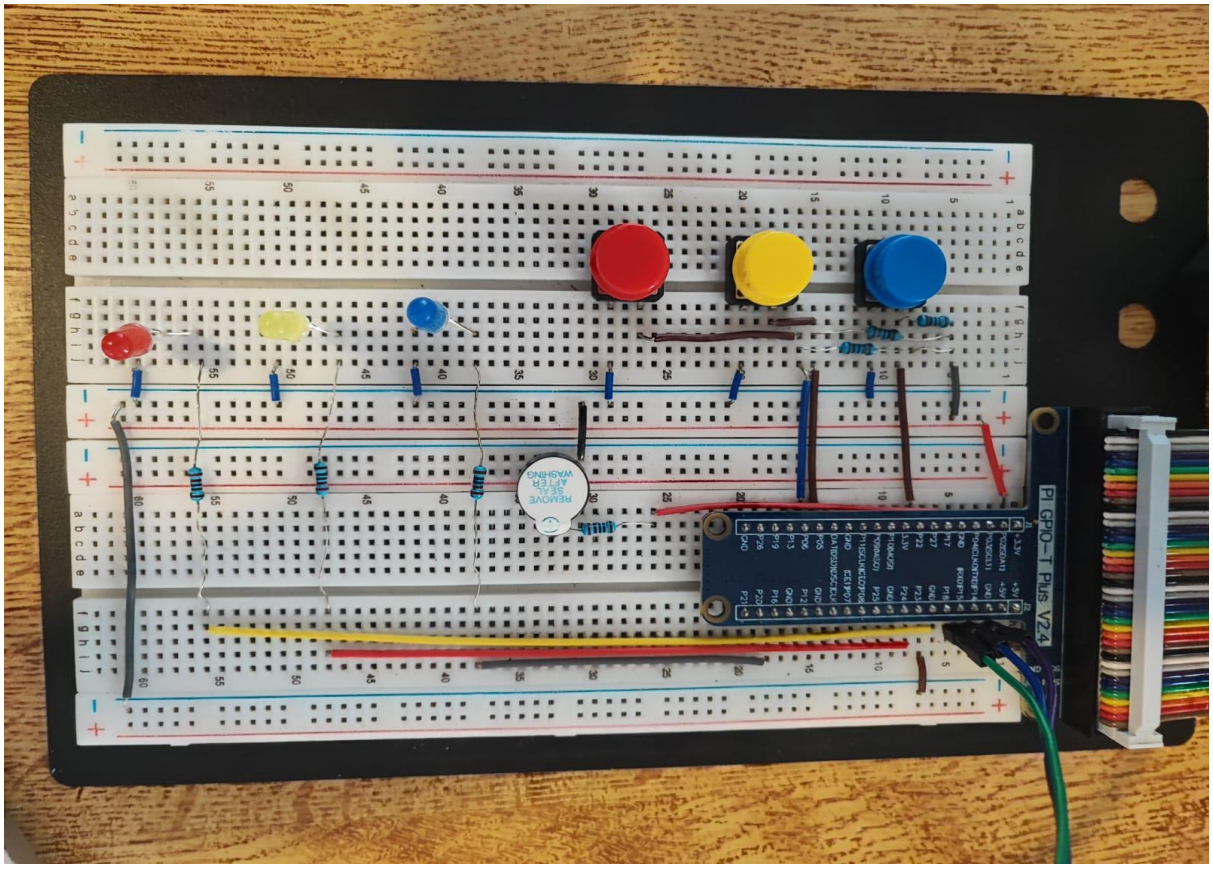


- ***READ_PRESSED_BUTTON*** gestisce la lettura dei pulsanti premuti dall'utente. Verifica quale è stato premuto e confronta il risultato con la sequenza di LED che l'utente deve replicare. Spegne quindi inizialmente tutti i LED e confronta il valore ritornato da BUTTON_CHECK con il valore del pulsante premuto. A seconda della maschera, si entrerà nel case del costrutto. Viene quindi acceso il LED del colore

corrispondente al pulsante premuto e vengono confrontati i due elementi in cima allo stack (il penultimo è legato al LED acceso, l'ultimo legato al pulsante premuto). Se sono uguali, viene settato l'ERROR a 0 ed il gioco prosegue, altrimenti viene settato l'ERROR ad 1, viene emesso un segnale tramite il buzzer ed il gioco termina.

- ***START_GAME*** avvia il gioco, imposta il round iniziale a 3, che rappresenta il numero di LED da accendere. Entra in un ciclo che continua fin quando il round non raggiunge 10 o l'utente commette un errore. Durante ogni ciclo, chiama la funzione LOOP_LED per accendere i LED in ordine casuale e poi attende l'input dell'utente usando la funzione READ_PRESSED_BUTTON. Se l'utente replica la sequenza correttamente, il round (quindi i LED che si accendono) aumentano di 1. Se sbaglia, il gioco termina.





Panoramica della breadboard e dei collegamenti I/O