7.Realizzazione sezione software

7.1 Controllo LED

Per la trasmissione e ricezione dei dati ho utilizzato il protocollo seriale. I dati vengono inviati ad Arduino dal compilatore tramite Max via porta seriale, per poi essere inviati alle strisce LED da Arduino.
Tra i vari test che ho effettuato per la realizzazione dei programmi per il controllo dei LED, solamente due metodi si sono dimostrati funzionali:

7.1.1 Primo Metodo

Il Primo Metodo è affidabile ma troppo dispendioso per il compilatore se usato con più di una striscia.

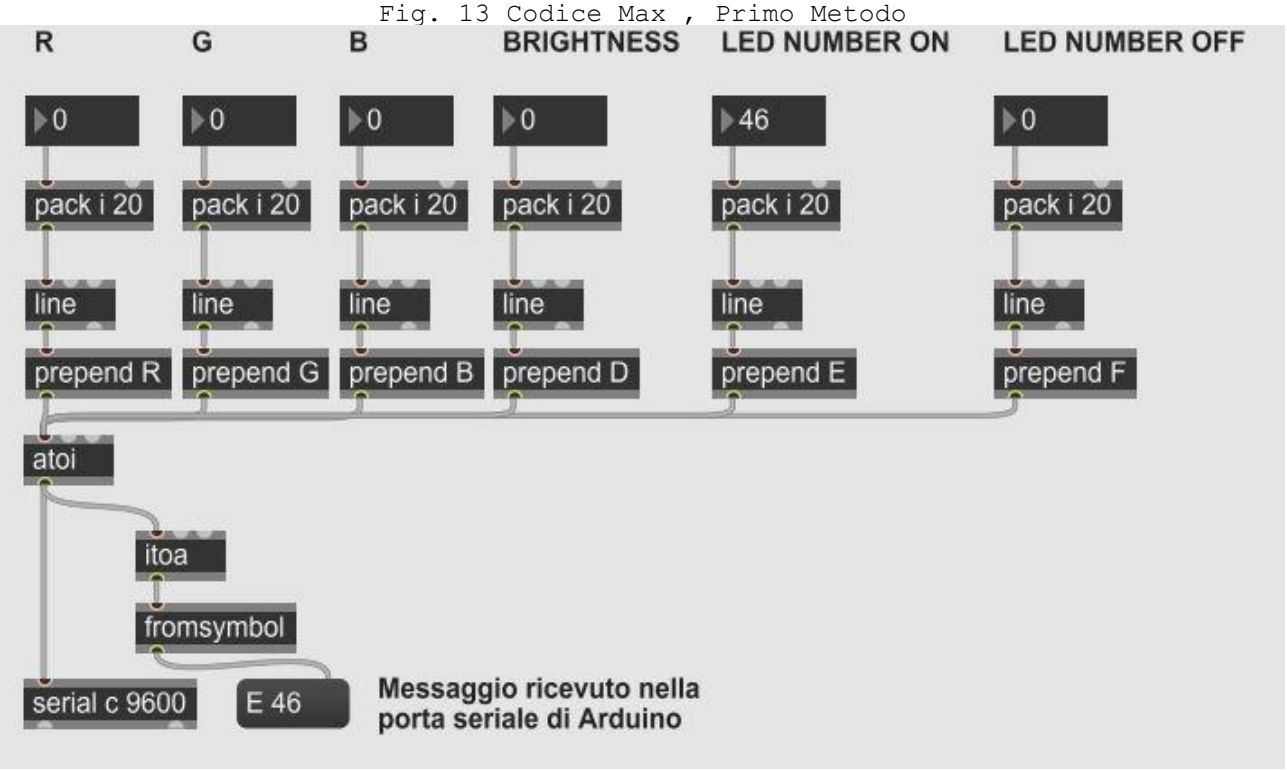
Il concetto di fondo di questo metodo si basa sul fatto che da Max vengono inviate delle stringhe di dati. Con in testa alla stringa una lettera che servirà ad Arduino all'indirizzamento corretto dei dati. Nel codice di Arduino vanno debitamente definite le variabili che corrisponderanno alle lettere inviategli da Max.
Per esempio se da Max invio un messaggio "R 150", questo messaggio quando sarà ricevuto ed analizzato da Arduino, per prima cosa verrà analizzato il primo Byte del messaggio e quindi la R, così facendo Arduino saprà in che area del programma dovrà andare ad assegnare il valore. Letto il primo Byte "R" passerà al successivo valore del messaggio ricevuto "150", e andrà ad assegnare il nuovo valore 150 nell'area dedicata a R. Altro esempio, se avessi inviato da Max un messaggio del tipo "B 50", dal momento che Arduino riceve il messaggio, prima valuta la lettera così da sapere dove posizionare il valore successivo cioè 50. Il metodo si è verificato valido per progetti di dimensioni molto ridotte, infatti come detto in precedenza il codice con questo funzionamento è molto lento nella verifica dei dati, e di conseguenza si rischia di perderne alcuni nella ricezione, questo vuol dire non essere poi inviati alle strisce LED.

In alto troviamo i *numberbox* in cui impostare il dato da inviare alla relativa variabile che verrà poi aggiornata da Arduino. L'oggetto *prepend*, inserisce il dato scritto alla sua destra d'avanti al dato che riceve in *input*. Successivamente il dato con la lettera viene tradotto dall'oggetto *itoa* in codifica di caratteri UTF-8[1] in modo che possa essere letto correttamente nella seriale di Arduino. In fine il messaggio tradotto viene inviato ad Arduino attraverso l'oggetto *serial* con impostata la stessa porta seriale e velocità di trasmissione in cui è impostato Arduino. In questo caso Arduino sta ricevendo il messaggio "E 46"[2].

[1] UTF-8 usa da 1 a 4 byte per rappresentare un carattere Unicode. Per esempio un solo byte è necessario per rappresentare i 128 caratteri dell'alfabeto ASCII.

[2] Per la trasmissione dei dati occorre impostare una velocità non superiore ai 9600 bps. Ho constatato che impostando una velocità superiore alcuni dati vengono persi in trasmissione, in realtà sarebbe comunque inutile impostare una velocità maggiore dal momento che le strisce non possono comunicare oltre 8000 bps. (bps = bit per secondo)

Fig. 13 Codice Max , Primo Metodo



Codice Arduino Primo Metodo:

```
1  #include <FastLED.h> //includo la libreria
2
3  #define LED_PIN 6
4  #define NUM_LED 59
5
6  CRGB leds[NUM_LED]; //array di 60 celle effettive che partono dalla posizione 0
7  int R, G, B, D, E, F; //inizializzo variabili  D = BRIGHTNESS  E= led accesi  F= led spenti
8
9  void setup(){ //funzione di start, contenente le informazioni primarie
10   Serial.begin(9600); //inizializzo la porta seriale, a velocità 9600 bps
11   FastLED.addLeds<NEOPIXEL, LED_PIN>(leds, NUM_LED); // imposto la striscia colegata
12 }                                                    //al pin LED_PIN formata da NUM_LED
13
14 void loop(){ //funzione di esecuzione ciclata
15  if(Serial.available() > 0){ //controllo il valore in entrata nella porta seriale
```

Fig. 34 Codice Arduino, Primo Metodo, parte 1

Riga 6: ***CRGB leds[NUM_LED];*** questa riga di codice fa riferimento alla libreria FastLED, e serve ad impostare in che modo da quel momento dovrò richiamare la striscia LED, *leds* = nome striscia. *[NUM_LED]* l'indice di questo array di dimensione 60, corrisponde al LED della striscia. Per esempio scrivendo ***leds[0];*** il programma saprà che dovrà agire sul primo LED della striscia.

Riga 15: dopo aver verificato che nella porta seriale si stanno ricevendo valori validi e quindi in questo caso maggiori di zero, si passa alla parte successiva del codice, quella di verifica.

```
44  Serial.available();
45  if(Serial.peek() == 'D'){ //prende primo byte, valuta se corrsiponde a D
46                            // e lo assegno alla variabile D
47  Serial.read();
48  D = Serial.parseInt(); //legge primo valore intero della stringa dal serial
49        FastLED.setBrightness( D ); //Brightness
50           FastLED.show(); //trasmetto i dati alla striscia LEd
51    }else{
52  Serial.available();
53  if(Serial.peek() == 'E'){ //prende primo byte, valuta se corrisponde a E
54  Serial.read();
55  E = Serial.parseInt(); //legge primo valore intero della stringa dal serial
56                         //e lo assegno alla variabile E
57        leds[E].setRGB( R, G, B); //posizione LED
58           FastLED.show(); //trasmetto i dati alla striscia LEd
```

Fig. 35 Codice Arduino, Primo Metodo, parte 2

Ricorda: messaggio inviato da Max = "E 46"

Dal momento che Arduino si è accertato di ricevere dei dati validi dal primo ***if(***riga 15) dalla porta seriale inizia a confrontare il primo byte del messaggio attraverso vari ***if*** a cascata finchè non trova quello corrispondente.

Serial.peek = prende il primo byte dalla riga corrente nella porta seriale.

Essendo il messaggio ricevuto "E 46", si fermerà al ***if***(riga 53), in quanto E == E(primo byte del messaggio). Successivamente nella riga 55, ***E = Serial.parseInt*** viene assegnata alla variabile E il primo valore intero contenuto nel messaggio, quindi "46".

Riga 57: aggiorno il dato da inviare alla striscia LED. Essendo il dato aggiornato la "E" che corrisponde alla posizione del LED, il risultato sarà quello di aggiornare con i valori correnti RGB il LED in posizione 46 (il quarantasettesimo LED della striscia).

Questo processo è identico per tutte le variabili del programma R, G, B, D, E, F. Ovviamente ognuna con il proprio spazio di codice dedicato.
Osservazione: Questa metodologia di codice come detto in precedenza è valida, ma solamente se si desidera utilizzare una singola striscia. Il fatto di fare il confronto iniziale produce una latenza non indifferente nel codice, che si rispecchia inevitabilmente sulla qualità dei dati effettivi inviati ai LED.

Il prossimo metodo che spiegherò è molto più funzionale, veloce e in un certo senso "semplice". Infatti è stato quello che ho utilizzato per pilotare il LEDCube.

7.1.2 Secondo Metodo

Il funzionamento di questo metodo si basa sull'acquisizione di un determinato dato, situato rispettivamente in una precisa posizione di un indice. Da Max invio dei messaggi alla porta seriale di Arduino, in questo caso però i messaggi non contengono *char*, ma solamente int(numeri interi). Ogni numero contenuto nel messaggio corrisponde ad un indice(numero

posizione), e grazie all'ordine di lettura successivamente Arduino smista i valori ai dati corretti. Il primo e l'ultimo valore contenuti nel messaggio fungono da valori di controllo, infatti non verranno effettivamente inviati alle strisce, servono solamente come conferma nel codice in modo di segnalare ad Arduino l'inizio e la fine del messaggio. Per esempio se Arduino riceve un messaggio "253 3 34 23 20 15 45 255". Ogni cifra del messaggio corrisponderà ad un indice di lettura, 253 = indice 0, 3 = indice 1 e via così fino a 255 = indice posizione 7.

Indice	0	1	2	3	4	5	6	7
Valore	253	3	34	23	20	15	45	255
Comando	Verif.	strip	R	G	B	Brig.	LED	Verif.

Tab. 1 Esempio messaggio di comando inviato da Max, con rispettivi Indici e Comandi

Quando Arduino riceve il messaggio controlla ogni indice, assegnando alla variabile esatta il valore in posizione di quel indice. Con una serie di condizioni ***if*** a cascata aggiorna il valore corrispondente e avanza di un indice fino ad arrivare all'ultimo che corrisponderà al valore di verifica fine messaggio "255". Al termine del codice invia alla striscia corrispondente i nuovi valori da aggiornare.
Codice Max Secondo Metodo:

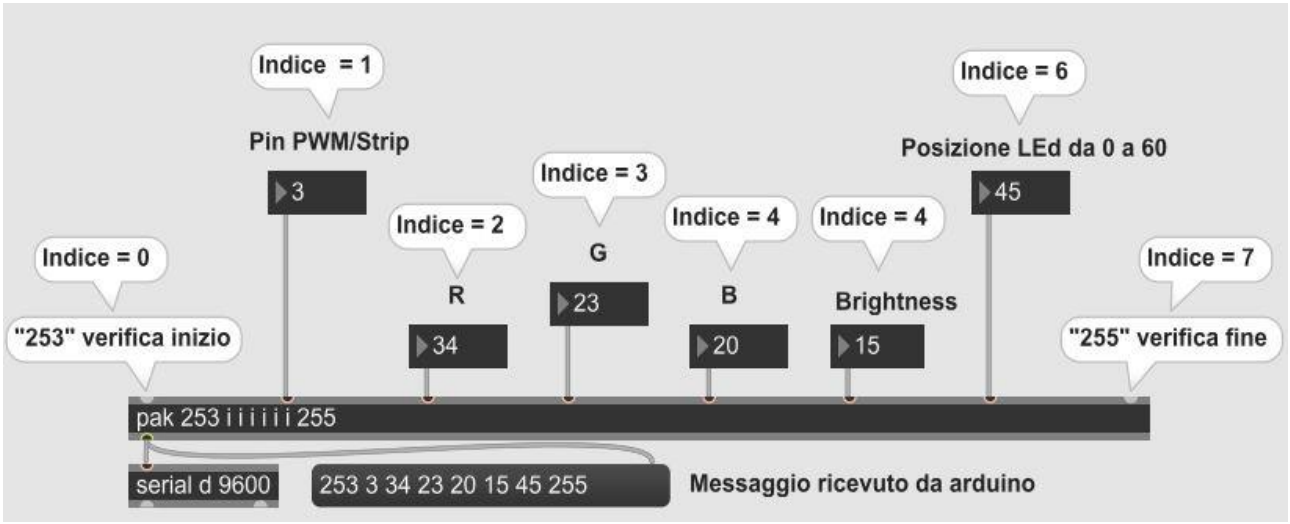


Fig. 36 Codice Max, Secondo Metodo

Il codice è una versione semplificata rispetto a quello utilizzato nell'installazione, per migliorare la velocità di lettura da parte del lettore.

Nei *numberbox* vengono riportati i valori dei parametri variabili, successivamente uniti in un unico messaggio dall'oggetto *pak*. Come si può vedere dall'immagine il primo e l'ultimo input di *pak* non sono collegati ad un *numberbox*, questo perché l'indice iniziale e finale contengono dei valori costanti di verifica, che non devono assolutamente variare. L'ordine degli input di pak corrisponde esattamente con l'ordine di indici ricevuti da Arduino. Questo vuol dire che se volessimo aggiungere o eliminare il numero di parametri controllabili basterà variare il numero di input e di conseguenza di indici, per poi aggiornare questo numero anche

nel codice Arduino. Ovviamente ricordando di mantenere il primo e l'ultimo indice di verifica costanti. Nel messaggio in uscita da *pak* possiamo vedere effettivamente il messaggio che verrà letto da Arduino.

Codice Arduino Secondo Metodo:

```
1  #include <FastLED.h>
2
3  #define NUM_LED 59 //60 LED per striscia
4
5  #define NUM_STRIPS 5 //6 strisce LED
6
7
8  CRGB led[NUM_STRIPS][NUM_LED]; //primo array = striscia
9                                 // secondo array = posizione LED
10
11 byte index= 0; //indice di lettura
12 byte R = 0; //RED
13 byte G = 0; //GREEN              //INIZIALIZZO LE VARIABILI
14 byte B = 0; //BLUE               //CON DEI VALORI INIZIALI
15 byte D = 0; //Brightness totale
16 byte E = 0; //posiz. led accesi
17 byte pin = 1; //pin
18
19 void setup(){
20   Serial.begin(9600); //apro la porta seriale a velocità 9600bps
21   FastLED.addLeds<NEOPIXEL, 2>(led[0], NUM_LED); //striscia al pin 2
22   FastLED.addLeds<NEOPIXEL, 3>(led[1], NUM_LED); //striscia al pin 3
23   FastLED.addLeds<NEOPIXEL, 4>(led[2], NUM_LED); //striscia al pin 4
24   FastLED.addLeds<NEOPIXEL, 5>(led[3], NUM_LED); //striscia al pin 5
25   FastLED.addLeds<NEOPIXEL, 6>(led[4], NUM_LED); //striscia al pin 6
26   FastLED.addLeds<NEOPIXEL, 7>(led[5], NUM_LED); //striscia al pin 7
27 }
```

Fig. 37 Codice Arduino, Secondo Metodo, parte 1

La parte iniziale del codice è molto simile a quella del metodo precedente. L' unica vera differenza è il fatto che in questo caso è predisposto per l'utilizzo di più strisce.
Infatti a riga 8, ***CRGB led[NUM_STRIPS][NUM_LED];*** ci troviamo a gestire un codice a array bidimensionale, in cui l'indice del primo ***[NUM_STRIPS]*** fa riferimento a quale striscia inviare i dati, e il secondo ***[NUM_LED]*** come nel primo metodo, a quale LED della striscia. Per esempio se avessimo una situazione del tipo ***led[2][34];*** i parametri saranno inviati al Led 35 della terza striscia[3].

[3] Il numero specificato nell'array è sempre di un unità inferiore al numero effettivo che ci occorre. Questo perché ricordo che gli array considerano lo 0 come primo indice.

```
30 void loop(){
31  while(Serial.available()) { //controllo la porta seriale
32     byte val= Serial.read(); //val =  byte ricevuto dalla seriale
33     if((index == 0) && (val == 253)) { //prima fase di verifica
34       index = 1;  //proseguo di 1 indice
35     } else if((index == 1)) {
36       pin = val; // imposto su quale pin/striscia agire
37       index = 2; //proseguo di 1 indice
38     } else if((index == 2)) {
39       R = val; // imposto RED
40       index = 3;
41      } else if((index == 3)) {
42       G = val; // imposto GREEN
43       index = 4;
44      } else if((index == 4)) {
45       B = val; // imposto BLUE
46       index = 5;
47      } else if((index == 5)) {
48       D = val; // imposto Brightness totale
49       index = 6;
50      } else if((index == 6)) {
51       E = val; // imposto su quale LED agire
52       index = 7;
53      } else if((index == 7) && (val == 255)) {//ultima fase di verifica
54       FastLED.setBrightness( D ); //imposto la Brightness con valore D
55       led[pin][E].setRGB( R, G, B); //invio alle strisce
56                                     //i nuovi dati aggiornati
57             FastLED.show(); //invio i dati alle strisce
58       index= 0;  //azzero l'indice e ricomincia il ciclo
59     } else {
60       index= 0;
61     }
```

Fig. 38 Codice Arduino, Secondo Metodo, parte 2

Nel ciclo principale avviene il processo di verifica e aggiornamento dei valori da inviare alle strisce, mediante una sequenza a cascata di condizioni ***if***. In ogni ciclo viene assegnato alla variabile predisposta per quel ciclo che poi verrà inviata alle strisce, il valore di ***val*** in posizione dell'indice corrente.

Riga 33: se l'indice si trova in posizione 0 e il valore in quella posizione è 253, avanza di una posizione di indice e prosegui alla prossima istruzione.

Da questo punto fino a riga 53, il codice effettua la stessa operazione tante volte quante sono le variabili da inviare alle strisce. Ad ogni condizione assegna il valore in posizione dell'indice corrente alla variabile della striscia, e avanza di una posizione l'indice. Fino ad arrivare all'ultima posizione.

Riga 53: se l'indice si trova in posizione 7 e il valore in quella posizione è 255, invia definitivamente alle strisce i nuovi valori aggiornati.

In fine viene azzerato l'indice ***index = 0;*** e il ciclo riparte.

Anche questo metodo presenta delle limitazioni, perchè ad ogni ciclo si aggiornano i valori di un singolo LED. Effettivamente anche se da Max stiamo inviando contemporaneamente dei messaggi a più strisce il programma non può effettivamente aggiornarle tutte in simultanea.
Questo perché il codice viene letto e effettuato procedendo dall'alto al basso, e non può effettuare più di un'operazione alla volta, ma, entro certi limiti, da un punto di vista percettivo le operazioni sembreranno simultanee. Ci sono dei metodi per effettuare più operazioni simultaneamente, come per esempio il *multithreading* in cui vengono predisposti dei sistemi a multiprocessore, in cui ogni uno è dedicato a effettuare una operazione. Così facendo si possono ottenere dei processi in parallelo, tutti indipendenti. Usare questo metodo in Arduino vorrebbe dire predisporre una scheda dedicata ad ogni processo, nel caso delle strisce LED una scheda Arduino per ogni striscia.

Anche il fatto di non poter utilizzare una velocità di trasmissione superiore ai 9600bps limita il sistema, infatti se i LED potessero ricevere dei dati ad una velocità maggiore si riuscirebbe a simulare di utilizzarne più di tre contemporaneamente.

7.3 Patch Max

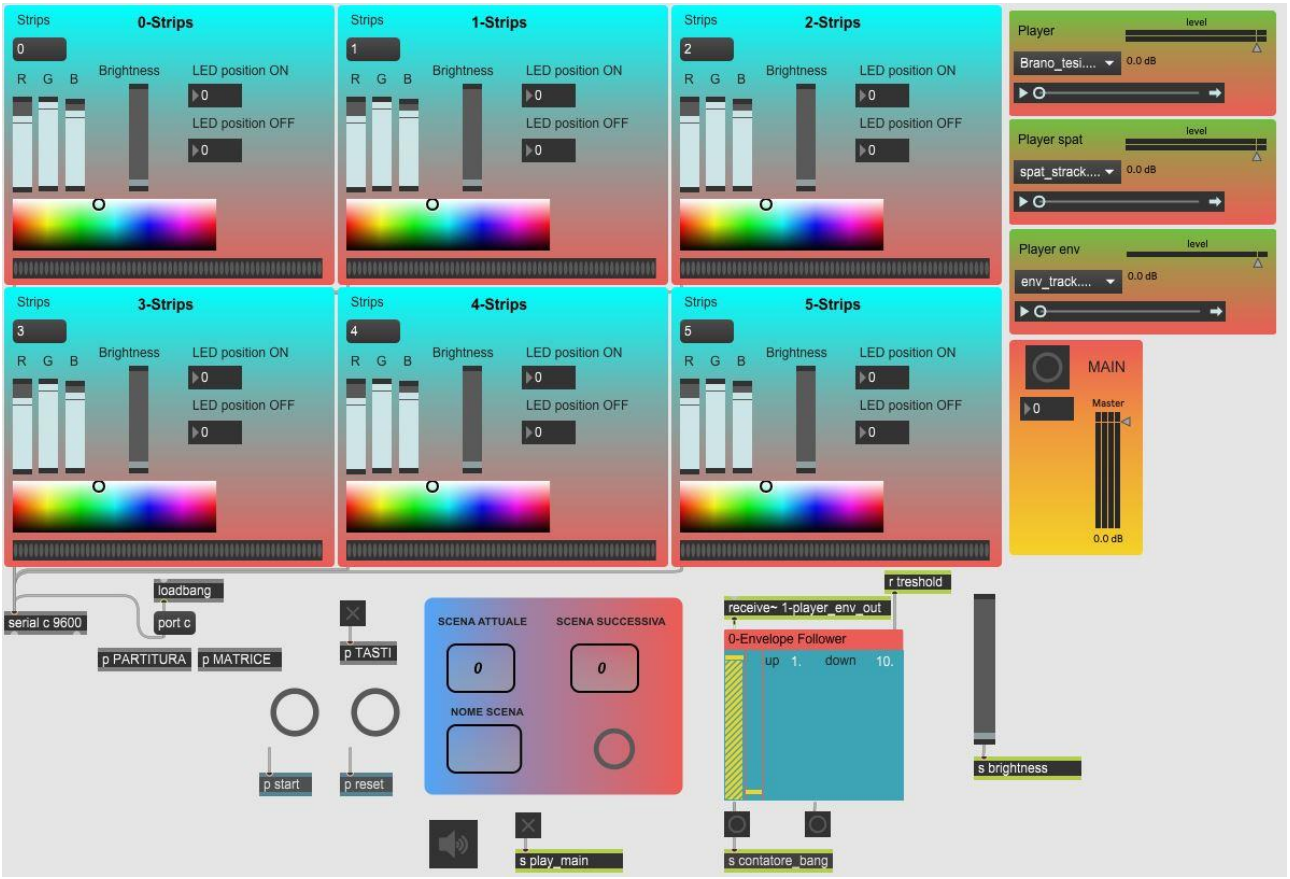


Fig. 39 Patch principale Max

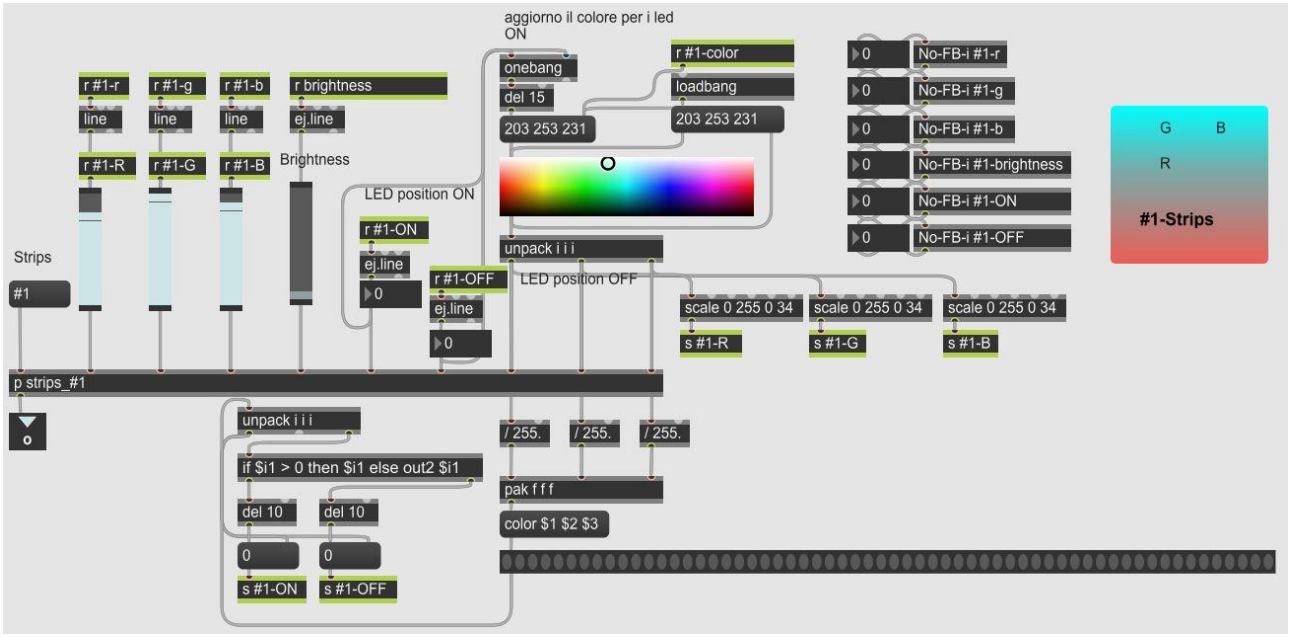


Fig. 43 Codice generale della patch di controllo della singola striscia LED