# Application Note di Francesco Sattolo

## game.c

1. Inizializza il sistema, il display, il touch screen e il RIT (50ms)
2. Effettua la calibrazione del touch screen.
3. Disegna il labirinto, i pulsanti e scrive le indicazioni richieste (nome del gioco, nomi dei pulsanti, istruzione per iniziare la partita).
4. Fa partire il RIT.
5. Mette il processore in power-down mode e attende le interruzioni del RIT.

## Labyrinth.h

Definizione delle misure e dei colori utilizzati per disegnare l'interfaccia. Definizione delle strutture dati usate.

## Labyrinth.c

1. Settaggio delle condizioni iniziali del labirinto e del robot.
   Per il labirinto: 0=casella libera, 1=ostacolo non visualizzato 2=uscita, 3=ostacolo attualmente visualizzato.
2. compute_distance():
   - calcola la distanza dall'ostacolo/muro più vicino.
   - se il robot si trova di fronte un ostacolo non ancora visualizzato, lo mostra a schermo e aggiorna il labirinto settando a 3 il valore dell'ostacolo visto.
   - setta il flag end a 1 quando il robot proseguendo dritto raggiungerebbe l'uscita.

## GLCD.h

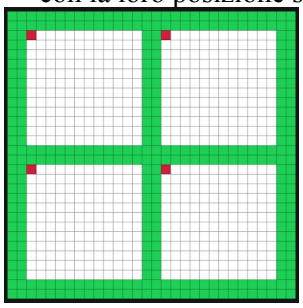
Aggiunto colori personalizzati e prototipi delle funzioni create in GLCD.c e funct_GLCD.c.

## GLCD.c

Aggiunte funzioni di supporto per disegnare linee orizzontali e verticali di una lunghezza data (MyDrawLine), disegnare quadrati e rettangoli (DrawSquare e DrawRectangle) e cancellare una casella di testo (DeleteText).

## funct_GLCD.c

Vengono aggiunte le funzioni index_to_position_x e index_to_position_y per mappare con facilità le caselle del labirinto con la loro posizione sul display.



In questo modo avendo due indici i,j che scorrono le caselle del labirinto, si può accedere ai corrispondenti pixel sul labirinto semplicemente tramite index_to_position_x(i) e index_to_position_y(j). Per disegnare un quadrato sul labirinto avendo le coordinate della casella basterà quindi richiamare la funzione DrawSquare passando come posizione di partenza index_to_position_x(i) e index_to_position_y(j).
(nell'esempio in rosso si hanno i pixel ritornati con <i=0, j=0>, <i=0, j=1>, <i=1,j=0>, <i=1,j=1>)

Le funzioni DrawLabyrinth e DrawButton sfruttano MyDrawLine, DrawSquare, DrawRectangle e le dimensioni definite in labyrinth.h per creare l'interfaccia grafica.
DrawRobot() usando cicli for, una variabile di conteggio e le funzioni LCD_SetPoint,index_to_position_x,y disegna un triangolo rappresentante il robot. Per impostare la posizione sul labirinto, il colore usato e la direzione del triangolo sfrutta la struttura dati globale Robot r.

## IRQ_RIT.c

1. RIT_IRQHandler() ogni 50ms fa il polling sia del touch screen che del joystick.

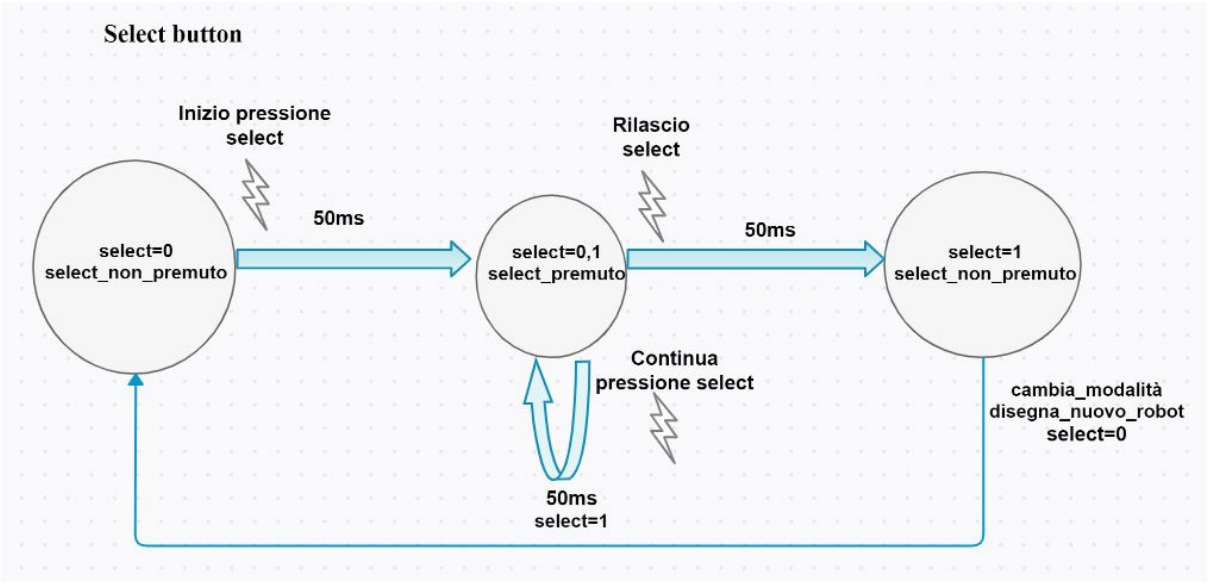
| started | win | Funzionalità utilizzabili |
|---|---|---|
| 0 | 0 | Click sul labirinto |
| 0 | 1 | Configurazione non supportata |
| 1 | 0 | Click sul tasto Reset; Click sul tasto Clear; Uso del joystick |
| 1 | 1 | Click sul tasto Reset |

## TOUCH SCREEN:

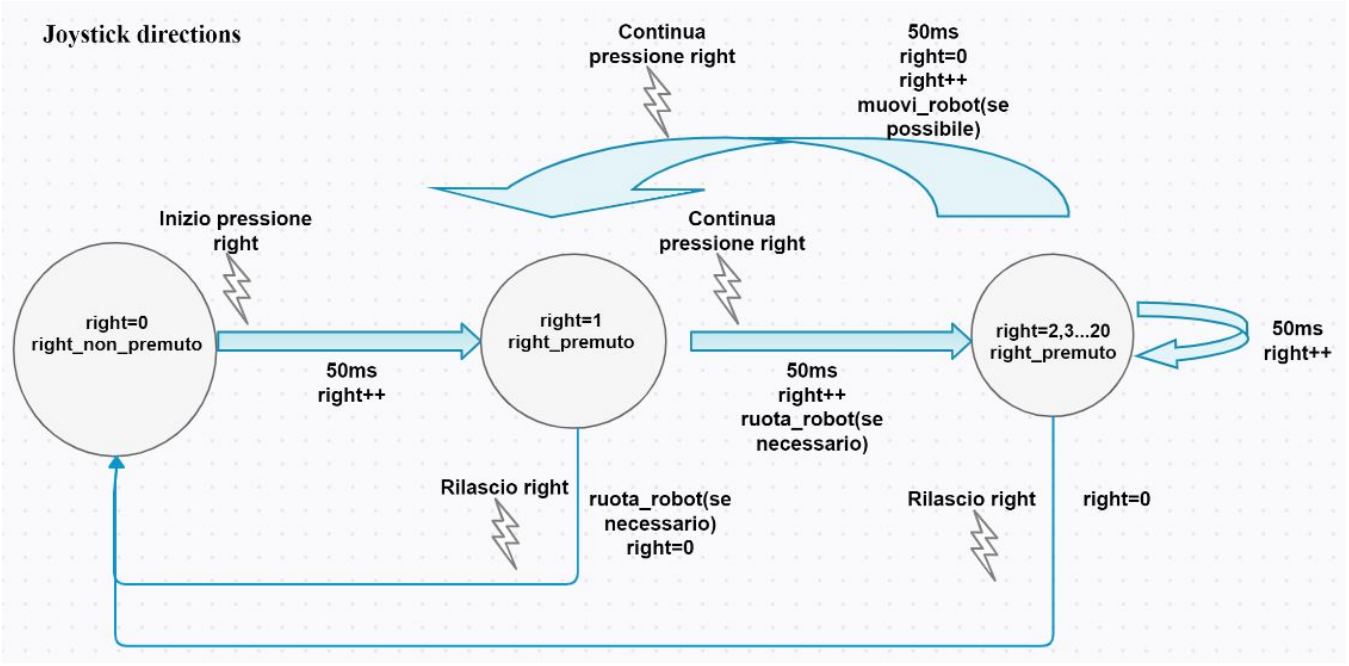
- RESET ha lo stesso effetto di premere il pulsante reset, ossia riporta il programma nello stato iniziale.
- CLEAR cancella dal display solo gli ostacoli effettivamente rilevati durante la partita attuale e riporta il labirinto nella condizione iniziale. Al termine dell'operazione richiama la funzione compute_distance() per poter rilevare nuovamente un ostacolo che si trovi nel range del robot.

## JOYSTICK:

- Il pulsante select viene attivato solo al rilascio e, dopo aver cambiato la modalità del robot, lo disegna, sovrascrivendo il precedente con il colore corretto. Non è necessario fare il debouncing del pulsante, essendo già fatto a livello hardware.




- Le funzionalità relative alle 4 direzioni vengono attivate durante la pressione:
  - La prima volta che l'interrupt del RIT rileva la pressione di una direzione, se il robot era rivolto in un'altra direzione rispetto a quella premuta, viene ruotato il robot, indipendentemente dalla modalità in cui si trovava, chiamando la funzione rotate().
  - Dopo 20 interrupt consecutivi del RIT in cui quella direzione è rimasta premuta (20*50ms=1s) se il robot è in grado di muoversi (r.distance!=0 && r.mode==MOVE) allora viene richiamata la funzione run() e viene resettato il conteggio degli interrupt per quella direzione.




2. rotate(DIRECTIONS newDir) cancella il robot precedente, cambia la direzione in cui è rivolto, disegna il nuovo robot e chiama compute_distance() per aggiornare la distanza dal prossimo ostacolo ed eventualmente visualizzarlo sul display.
3. run() cancella il robot precedente, aggiorna la posizione in cui si trova, disegna il nuovo robot e aggiorna la distanza dall'ostacolo successivo. E' necessario usare compute_distance() solo nel caso in cui, prima del movimento, la zona visibile dal robot fosse libera (r.distance==5). In questo caso, infatti, il movimento potrebbe rendere visibile un nuovo ostacolo che quindi necessiterebbe di essere rappresentato a schermo. Negli altri casi invece è sufficiente decrementare la distanza attuale poiché l'ostacolo sarebbe già stato rappresentato dalla compute_distance() richiamata da una run() o da una rotate() precedenti.
   Infine si controlla se la casella in cui è terminato il robot è una di uscita (r.end==1 && r.distance==0) e in tal caso si fanno comparire 2 messaggi a schermo, si cancella il tasto clear, non più utilizzabile, e si setta win=1, per far sì che ai successivi interrupt del RIT l'unico tasto funzionante sia quello di reset.