# Application Note

## 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

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

* Position pos: posizione attuale del robot.
* DIRECTIONS dir: direzione in cui è rivolto.
* \_Bool end: vale 1 quando il robot proseguendo dritto raggiungerebbe l'uscita.
* uint8\_t distance: distanza del robot dall'ostacolo/muro più vicino. Può assumere valori da 0 a RANGE, con RANGE che sta a significare qualunque distanza superiore alla portata del sensore di prossimità.
* MODES mode: modalità in cui si trova il robot.

## 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.

1. 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.

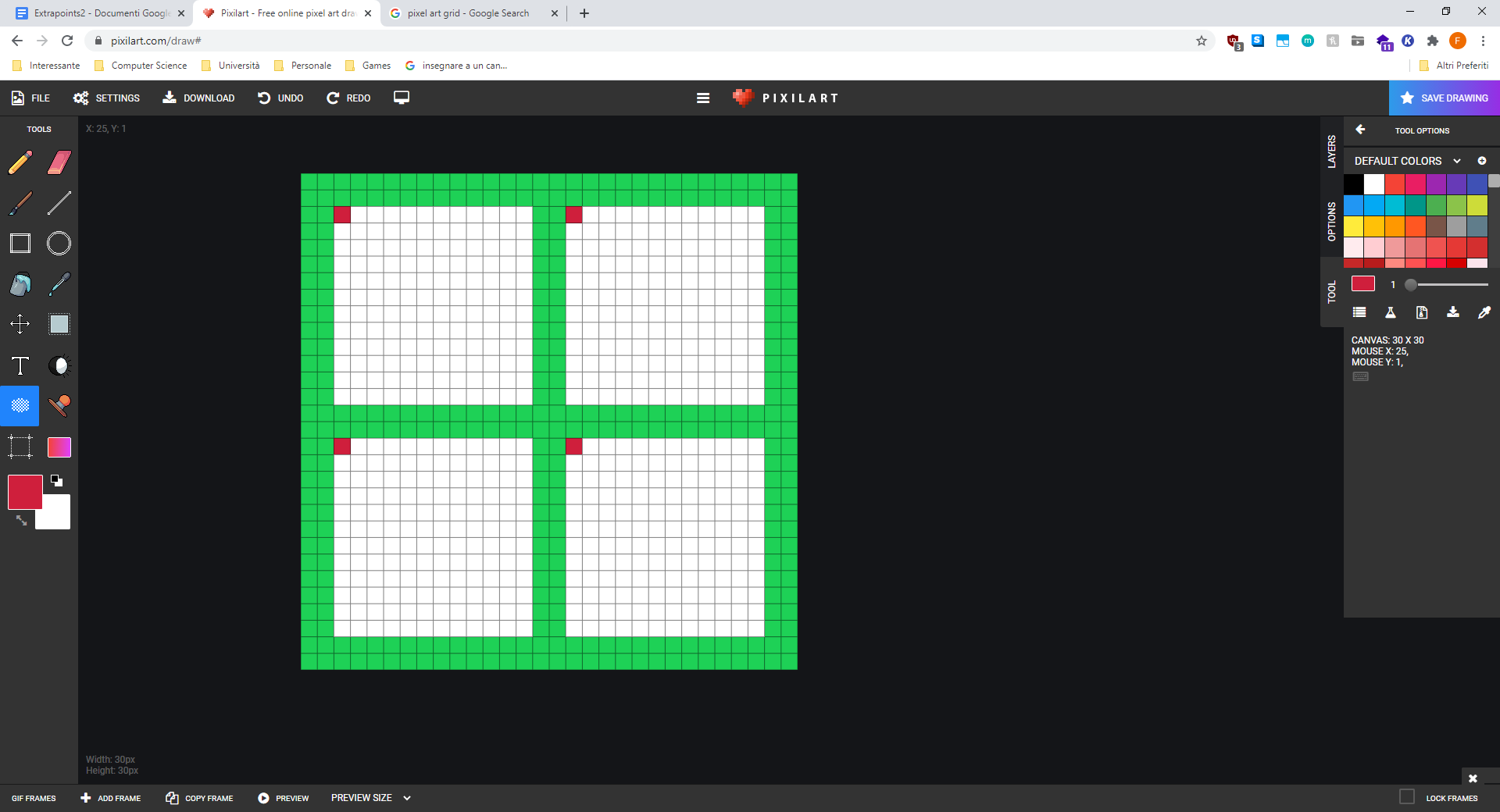
## GLCD.h

Aggiunto colori personalizzati e prototipi delle funzioni create in GLCD.c e funct\_GLCD.c.

## GLCD.c

1. MyDrawLine(uint16\_t x0, uint16\_t y0, uint16\_t size, uint16\_t color, LINE\_DIRECTIONS dir) a partire dalle coordinate passate come parametro disegna una linea lunga size, orizzontale o verticale a seconda del valore dell'enum dir, del colore scelto.
2. DeleteText(uint16\_t Xpos, uint16\_t Ypos, uint16\_t strlen, uint16\_t bkColor) prende in input la posizione in pixel dove inizia la casella di testo, la lunghezza del testo contenuto e il colore dello sfondo e sovrascrive la casella di testo con un rettangolo del colore scelto.
3. DrawSquare(uint16\_t Xpos, uint16\_t Ypos, uint16\_t Side, uint16\_t Color) prende in input la posizione in pixel dove iniziare a disegnare il quadrato, la dimensione del suo lato e il colore.
4. DrawRectangle(uint16\_t Xpos, uint16\_t Ypos, uint16\_t sideX, uint16\_t sideY, uint16\_t Color) funziona come la DrawSquare ma prende in input la dimensione di entrambi i lati.

## funct\_GLCD.c

1. uint16\_t index\_to\_position\_x(uint16\_t index\_x) e uint16\_t index\_to\_position\_y(uint16\_t index\_y) prendono in input l'indice x/y di una casella del labirinto e ritornano la coordinata x/y del pixel in cui inizia la parte interna del quadrato corrispondente sul labirinto mostrato sul display. Serve 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).

(nell'esempio <i=0, j=0>, <i=0, j=1>, <i=1,j=0>, <i=1,j=1>)

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).

1. void DrawLabyrinth(uint16\_t Xpos, uint16\_t Ypos) a partire dalle coordinate passate disegna la griglia del labirinto, spessa 2 pixel, e riempie una a una le caselle ottenute.
2. DrawButton(uint16\_t Xpos, uint16\_t Ypos, uint8\_t \*str) a partire dalle coordinate passate disegna il contorno del pulsante, lo riempie e vi scrive la stringa passata come parametro.
3. DrawRobot() usando cicli for, una variabile di conteggio e la funzione LCD\_SetPoint 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.
4. DeleteRobot() sovrascrive il robot con una casella vuota.

## IRQ\_RIT.c

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

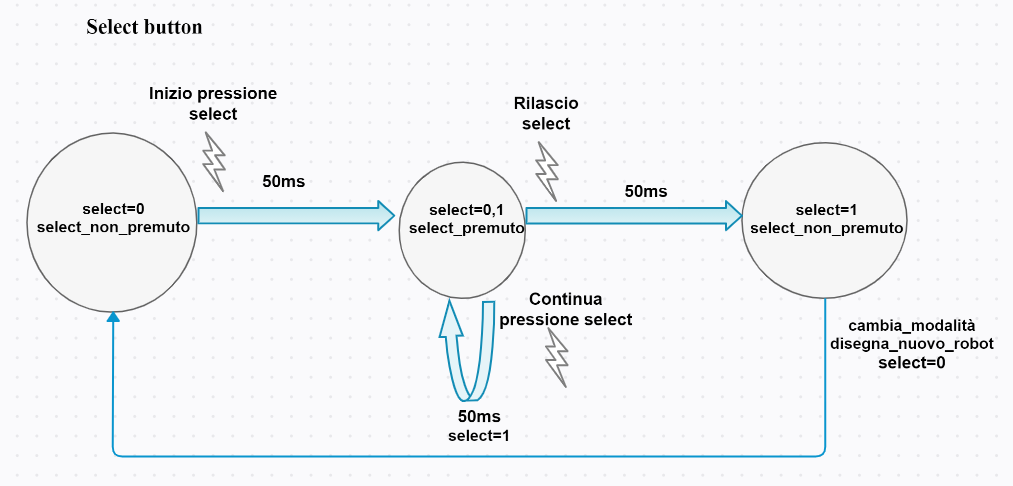
TOUCH SCREEN:

* La pressione sul labirinto viene rilevata solo se il gioco non è ancora cominciato (started==0 && win==0), e dà inizio alla partita.
* La pressione sul tasto reset è possibile solo dopo l'inizio del gioco (started==1) ed è l'unica funzione disponibile dopo il termine della partita (end==1). Ha lo stesso effetto di premere il pulsante reset, ossia riporta il programma nello stato iniziale.
* Il tasto clear ha effetto solo durante la partita (started==1 && win==0 ). Permette di cancellare dal display solo gli ostacoli effettivamente rilevati durante la partita attuale e riportare 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.

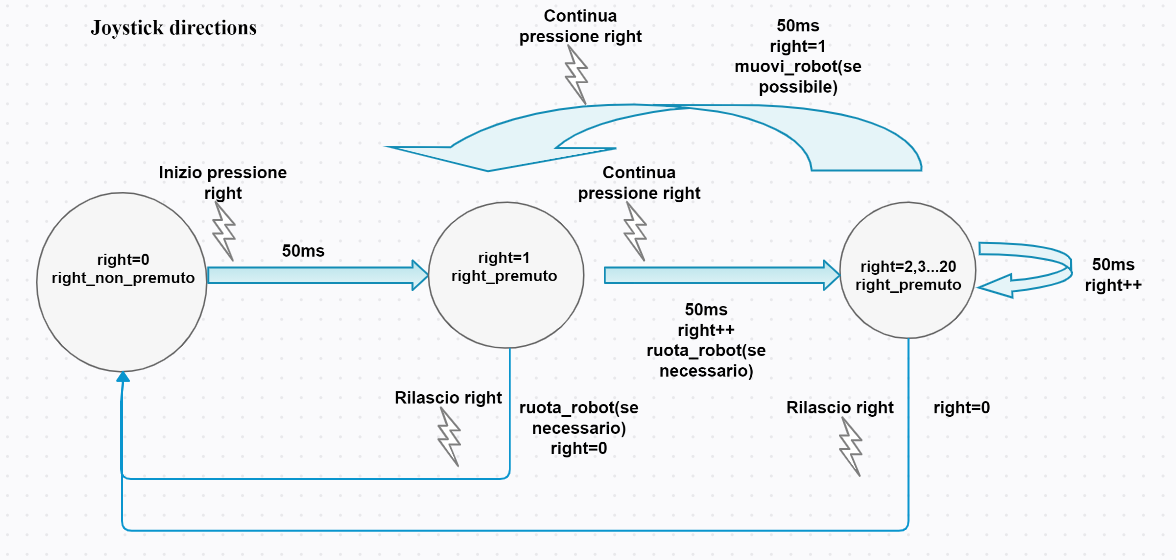
JOYSTICK:

Può essere utilizzato solo durante la partita (started==1 && win==0).

* 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.



* Gli altri pulsanti vengono attivati 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.



1. 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.
2. 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.