Bouncing LED

Generato da Doxygen 1.9.2

Capitolo 1

Simulazione di pallina rimbalzante su barra a LED

1.1 Introduzione

Questo progetto ha lo scopo di simulare graficamente una caduta verticale di una pallina con relativo rimbalzo utilizzando una striscia a LED come dispositivo di visualizzazione.

Per quanto possibile verranno applicate le leggi della fisica.

Nota

Semplificazioni ed ottimizzazioni saranno consentite per la più semplice elaborazione dei dati.

1.2 Descrizione

La pallina è simulata su una striscia a LED posta verticalmente, con il primo LED in alto.

Ad intervalli di tempo fissi è calcolata la posizione della pallina ed aggiornata la visualizzazione.

L'urto con il terreno virtuale è considerato anelastico con perdita del 15% della quantità di moto (coefficiente di restituzione: $e = 85\,\%$).

A pallina ferma la simulazione riparte cambiando il colore della stessa.

1.3 Concetti di base

La simulazione è, per quanto possibile, basata sulle leggi della fisica classica.

- Accelerazione di gravità media sulla Terra (costante):

$$g_n = 9.80665\,\mathrm{m}\cdot\mathrm{s}^{-2} \qquad (1)$$

- Formule del moto uniformemente accelerato:

$$v = v_0 + a \cdot t \qquad (2)$$

$$s = v_0 \cdot t + \frac{1}{2} \cdot a \cdot t^2 \qquad (3)$$

  dove $a = g_n$.

- In caso di urto, supposto perpendicolare, la velocità è invertita –diventando negativa– e moltiplicata per il *coefficiente di restituzione* dell'urto anelastico:

$$v_0 = -e \cdot v_0 \qquad (4)$$

  dove $e = 0.8$.

1.4 Implementazione

Di seguito è descritta l'implementazione dei moduli ad alto livello.

1.4.1 Hardware

Il sistema è composto dai seguenti componenti:

1. **Arduino Uno** Rev3.
2. **Striscia LED WS2812B** 1m, 60 LEDs, 5 VDC.
3. **Alimentatore USB** 5V, $\geq$1A.
4. **Cavo USB** di alimentazione per Arduino.

La pallina è simulata su una striscia a LED tipo **WS2812B** di $1\,\mathrm{m} \times 60\,\mathrm{LEDs}$.

Il segnale di controllo della striscia a LED è connesso al pin **`D6`** di Arduino.

La frequenza di rinfresco del display è impostata a $50\,\mathrm{Hz}$.

1.4.2 Software

Il progetto dipende dalle seguenti librerie di Arduino:

1. **FastLED** by Daniel Garcia, *versione 3.5.0*.

1.4.3 Inizializzazione (setup)

La procedura di inizializzazione è composta dalle seguenti fasi:

- Inizializzazione della libreria grafica, con conseguente spegnimento di tutti i LEDs.
- Inizializzazione del timer periodico a $50\,\mathrm{Hz}$ in modo da generare interrupt.
- Inizializzazione dei parametri della pallina. I valori di inizializzazione sono i seguenti:
  - $v_0 = 0$
  - $s = 0$
  - $colore = bianco$
  - $t_\mathrm{r} =$ `<ora attuale>`
- Attivazione dell'interrupt periodico.
- Uscita dalla funzione e conseguente ingresso nel loop dell'applicazione.

1.4.4 Ciclo periodico (loop)

Durante il ciclo periodico la CPU verifica lo stato del flag che indica la necessità di aggiornare il display.

In caso affermativo sono eseguite le seguenti procedure:

1. Registrazione dell'ora corrente, tramite la funzione `mills()`.
2. Aggiornamento dello stato della pallina, considerando il tempo trascorso dall'ultimo aggiornamento.
3. Aggiornamento dello stato della striscia a LEDs, in baso allo stato della pallina.
4. Verifica della collisione con il terreno ed eventuali azioni conseguenti.
5. Verifica del termine della simulazione (pallina ferma) e riavvio della stessa utilizzando un altro colore.

Al termine delle operazioni sopraindicate la CPU entra in modalità a basso consumo in attesa del successivo risveglio da interrupt.

1.4.5 Interruzione periodica

L'interruzione periodica segnala la necessità di aggiornare la visualizzazione impostando un apposito *flag*.

Il `flag` è *atomico* e condiviso con la routine di `loop()` principale.

1.4.6 Aggiornamento dello stato della pallina

L'aggiornamento dello stato della pallina avviene calcolando i nuovi parametri di posizione e velocità.

1. Lo spazio percorso è calcolato in base alla formula **(3)**.
2. La velocità è calcolata in base alla formula **(2)**.
3. è aggiornata l'ora degli ultimi calcoli.

1.4.6.1 Semplificazioni

Per semplificare i calcoli l'accelerazione di gravità è approssimata a:

$$g_n = 10\,\mathrm{m}\cdot\mathrm{s}^{-2} \qquad (5)$$

Per evitare arrotondamenti eccessivi nei calcoli i tempi sono espressi in **millisecondi** (ms), le distanze in **micrometri** (µm) e l'accelerazione di gravità in $\mu m \cdot ms^{-2}$.

1.4.6.2 Visualizzazione

La visualizzazione utilizza le informazioni dell'oggetto palla per ricavare l'indice del LED da accendere lungo la striscia.

Sono spenti tutti i LED, ed acceso solo il LED relativo alla posizione corrente, calcolata come:
`indice = <posizione in mm> * NUMERO_LED_PER_METRO / 1000.`

1.4.7 Termine

La simulazione termina quando la pallina è ferma ($v < v_{\min}$). La velocità minima della pallina, sotto la quale è considerata ferma è fissata in $500\,\mathrm{mm} \cdot \mathrm{s}^{-1}$.

Dopo aver cambiato il colore della palla la simulazione riparte dalle condizioni iniziali.

La sequenza ciclica definita dei colori è:

1. Bianco
2. Rosso
3. Verde
4. Blu
5. Giallo
6. Viola

Capitolo 2

Indice delle strutture dati

2.1 Strutture dati

Queste sono le strutture dati con una loro breve descrizione:

Capitolo 3

Indice dei file

3.1 Elenco dei file

Questo è un elenco dei file documentati con una loro breve descrizione:

Capitolo 4

Documentazione delle classi

4.1 Riferimenti per la struct BALL_T

Definizione dell'oggetto `palla`

Campi

- unsigned long lastUpdateTime_ms

  Ora assoluta dell'ultimo aggiornamento dei dati (ms).
- uint16_t position_mm

  Attuale distanza lineare dall'origine (mm).
- int16_t speed_mm_s

  Attuale velocità (mm/s), positiva in caso di caduta, negativa in caso la palla si allontani dal terreno.
- CRGB color

  Colore della pallina.

4.1.1 Descrizione dettagliata

Definizione dell'oggetto `palla`

Definizione alla linea 270 del file BouncingLED.ino.

4.1.2 Documentazione dei campi

4.1.2.1 color

```
CRGB color
```

Colore della pallina.

Definizione alla linea 275 del file BouncingLED.ino.

4.1.2.2 lastUpdateTime_ms

```
unsigned long lastUpdateTime_ms
```

Ora assoluta dell'ultimo aggiornamento dei dati (ms).

Definizione alla linea 272 del file BouncingLED.ino.

4.1.2.3 position_mm

```
uint16_t position_mm
```

Attuale distanza lineare dall'origine (mm).

Definizione alla linea 273 del file BouncingLED.ino.

4.1.2.4 speed_mm_s

```
int16_t speed_mm_s
```

Attuale velocità (mm/s), positiva in caso di caduta, negativa in caso la palla si allontani dal terreno.

Definizione alla linea 274 del file BouncingLED.ino.

La documentazione per questa struct è stata generata a partire dal seguente file:

- BouncingLED.ino

Capitolo 5

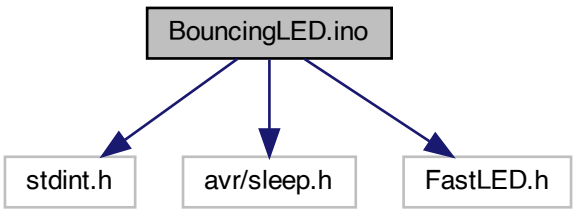
Documentazione dei file

5.1 Riferimenti per il file BouncingLED.ino

Simulazione di pallina rimbalzante su barra a LED.

```
#include <stdint.h>
#include <avr/sleep.h>
#include "FastLED.h"
```

Grafo delle dipendenze di inclusione per BouncingLED.ino:



Strutture dati

- struct BALL_T

  Definizione dell'oggetto `palla`

Definizioni

- #define GRAVITY_M_S2 9.80665f

  Accelerazione di gravità (m/s^ 2)
- #define COLLISION_COEFFICIENT_OF_RESTITUTION_PERCENT 85

  Coefficiente di restituzione del rimbalzo (percentuale)
- #define DISPLAY_TYPE NEOPIXEL

Tipo di display.

- #define STRIP_NUM_OF_LED_PER_M 60u

  Densità di LEDs per metro della striscia.
- #define STRIP_LENGHT_IN_MM 1000u

  Lunghezza della striscia LED (mm)
- #define STRIP_PIN 6

  Pin di controllo della striscia LED.
- #define REFRESH_RATE_HZ 50u

  Periodo di rinfresco del display (Hz)
- #define GRAVITY_APPROX_M_S2 ((uint32_t)(GRAVITY_M_S2 + 0.5f))

  *Approssimazione di **g** come intero (m/s^2)*
- #define STRIP_NUM_OF_LED ((STRIP_NUM_OF_LED_PER_M) ∗ (STRIP_LENGHT_IN_MM) / 1000uL)

  Numero di LED della striscia.
- #define MIN_BALL_SPEED_ON_COLLISION_MM_S 500u

  Velocità minima della palla al rimbalzo sotto la quale è considerata ferma in $mm \cdot s^{-1}$.

Ridefinizioni di tipo (typedef)

- typedef struct BALL_T **ball_t**

  Definizione dell'oggetto `palla`

Funzioni

- static void Ball_Init (ball_t ∗const pBall, unsigned long now, CRGB color)

  Inizializzazione dell'oggetto `palla`
- static void Ball_Update (ball_t ∗const pBall, unsigned long now)

  Aggiorna i dati correnti dell'oggetto `palla`
- static void Ball_OnCollision (ball_t ∗const pBall)

  Evento di collisione della palla con il terreno (rimbalzo)
- static uint16_t Ball_GetPosition (ball_t const ∗const pBall)

  Posizione della palla rispetto all'origine (mm)
- static uint16_t Ball_GetSpeed (ball_t const ∗const pBall)

  Velocità della palla (modulo) (mm/s)
- static CRGB Ball_GetColor (ball_t const ∗const pBall)

  Colore della palla (CRGB)
- static bool Ball_IsNotMoving (ball_t ∗const pBall)

  Verifica che la palla sia ferma.
- static CRGB GetNextBallColor ()

  Determina il prossimo colore della palla.
- static bool CheckForCollision (ball_t ∗const pBall)

  Verifica l'evento di collisione della palla con il terreno.
- static void InitDisplay ()

  Inizializzazione del display.
- static void RefreshDisplay (ball_t const ∗const pBall)

  Visualizza l'oggetto palla sul display.
- static void SetupTick ()

  Preparazione dell'interrupt periodico.
- static void StartTick ()

  Attiva l'interruzione periodica.

- ISR (TIMER1_COMPA_vect)

  Interrupt service routine del timer periodico.
- void setup ()

  Arduino `setup()`
- void loop ()

  Arduino `loop()`

Variabili

- static const CRGB BallColorSequence [ ]

  Sequenza dei colori della palla.
- static volatile bool m_isDisplayToBeRefreshed = false

  Semaforo usato per la visualizzazione, controllato dal tick periodico.
- static ball_t m_ball

  Oggetto `palla`
- static CRGB m_leds [STRIP_NUM_OF_LED]

  Matrice contenente lo stato dei LED della striscia, usata dalla libreria FastLED.
- static uint8_t m_ballCurrentColorIndex = 0

  Indice del colore corrente della palla.

5.1.1 Descrizione dettagliata

Simulazione di pallina rimbalzante su barra a LED.

Autore

Carlo Banfi

Copyright

2022 - Carlo Banfi

Definizione nel file BouncingLED.ino.

5.1.2 Documentazione delle definizioni

5.1.2.1 COLLISION_COEFFICIENT_OF_RESTITUTION_PERCENT

```
#define COLLISION_COEFFICIENT_OF_RESTITUTION_PERCENT 85
```

Coefficiente di restituzione del rimbalzo (percentuale)

Definizione alla linea 218 del file BouncingLED.ino.

5.1.2.2 DISPLAY_TYPE

```
#define DISPLAY_TYPE NEOPIXEL
```

Tipo di display.

Definizione alla linea 224 del file BouncingLED.ino.

5.1.2.3 GRAVITY_APPROX_M_S2

```
#define GRAVITY_APPROX_M_S2 ((uint32_t)(GRAVITY_M_S2 + 0.5f))
```

Approssimazione di **g** come intero (m/s^2)

Definizione alla linea 242 del file BouncingLED.ino.

5.1.2.4 GRAVITY_M_S2

```
#define GRAVITY_M_S2 9.80665f
```

Accelerazione di gravità (m/s^2)

Definizione alla linea 215 del file BouncingLED.ino.

5.1.2.5 MIN_BALL_SPEED_ON_COLLISION_MM_S

```
#define MIN_BALL_SPEED_ON_COLLISION_MM_S 500u
```

Velocità minima della palla al rimbalzo sotto la quale è considerata ferma in $mm \cdot s^{-1}$.

Definizione alla linea 248 del file BouncingLED.ino.

5.1.2.6 REFRESH_RATE_HZ

```
#define REFRESH_RATE_HZ 50u
```

Periodo di rinfresco del display (Hz)

Definizione alla linea 236 del file BouncingLED.ino.

5.1.2.7 STRIP_LENGHT_IN_MM

```
#define STRIP_LENGHT_IN_MM 1000u
```

Lunghezza della striscia LED (mm)

Definizione alla linea 230 del file BouncingLED.ino.

5.1.2.8 STRIP_NUM_OF_LED

```
#define STRIP_NUM_OF_LED ((STRIP_NUM_OF_LED_PER_M) * (STRIP_LENGHT_IN_MM) / 1000uL)
```

Numero di LED della striscia.

Definizione alla linea 245 del file BouncingLED.ino.

5.1.2.9 STRIP_NUM_OF_LED_PER_M

```
#define STRIP_NUM_OF_LED_PER_M 60u
```

Densità di LEDs per metro della striscia.

Definizione alla linea 227 del file BouncingLED.ino.

5.1.2.10 STRIP_PIN

```
#define STRIP_PIN 6
```

Pin di controllo della striscia LED.

Definizione alla linea 233 del file BouncingLED.ino.

5.1.3 Documentazione delle funzioni

5.1.3.1 Ball_GetColor()

```
static CRGB Ball_GetColor (
            ball_t const *const pBall )  [static]
```

Colore della palla (CRGB)

Parametri

pBall	Puntatore all'oggetto `palla`.

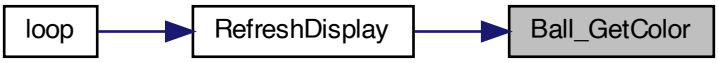
Restituisce

Colore della palla CRGB.

Definizione alla linea 413 del file BouncingLED.ino.

```
00414 {
00415   return pBall->color;
00416 }
```

Questo è il grafo dei chiamanti di questa funzione:



5.1.3.2 Ball_GetPosition()

```
static uint16_t Ball_GetPosition (
            ball_t const *const pBall )  [static]
```

Posizione della palla rispetto all'origine (mm)

Parametri

pBall	Puntatore all'oggetto `palla`.

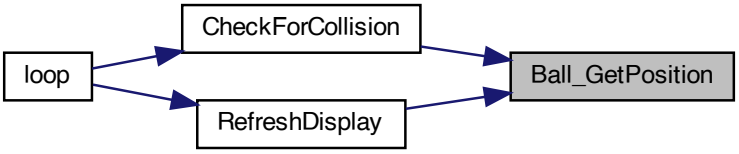
Restituisce

Posizione della palla rispetto all'origine (mm).

Definizione alla linea 391 del file BouncingLED.ino.

```
00392 {
00393   return pBall->position_mm;
00394 }
```

Questo è il grafo dei chiamanti di questa funzione:



5.1.3.3 Ball_GetSpeed()

```
static uint16_t Ball_GetSpeed (
            ball_t const *const pBall )  [static]
```

Velocità della palla (modulo) (mm/s)

Parametri

pBall	Puntatore all'oggetto `palla`.

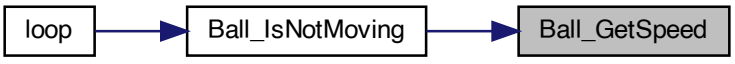
Restituisce

Velocità della palla (modulo) in mm/s.

Definizione alla linea 402 del file BouncingLED.ino.

```
00403 {
00404   return abs( pBall->speed_mm_s );
00405 }
```

Questo è il grafo dei chiamanti di questa funzione:



5.1.3.4 Ball_Init()

```
static void Ball_Init (
             ball_t *const pBall,
             unsigned long now,
             CRGB color )  [static]
```

Inizializzazione dell'oggetto `palla`

La palla è posta all'origine (il punto più alto), con velocità pari a zero.

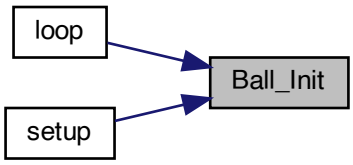
Parametri

pBall	Puntatore all'oggetto `palla`.
now	Ora corrente, tempo assoluto in ms.

Definizione alla linea 331 del file BouncingLED.ino.

```
00332 {
00333   pBall->lastUpdateTime_ms = now;
00334   pBall->position_mm = 0;
00335   pBall->speed_mm_s = 0;
00336   pBall->color = color;
00337 }
```

Questo è il grafo dei chiamanti di questa funzione:



5.1.3.5 Ball_IsNotMoving()

```
static bool Ball_IsNotMoving (
             ball_t *const pBall )  [static]
```

Verifica che la palla sia ferma.

Da chiamare dopo la collisione, quando la palla è in risalita.

Parametri

pBall	Puntatore all'oggetto `palla`.

Restituisce

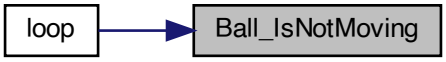
`true` se la palla è ferma, `false` altrimenti.

Definizione alla linea 426 del file BouncingLED.ino.

```
00427 {
00428   return ( Ball_GetSpeed(pBall) < MIN_BALL_SPEED_ON_COLLISION_MM_S );
00429 }
```

Questo è il grafo delle chiamate per questa funzione:

Questo è il grafo dei chiamanti di questa funzione:



5.1.3.6 Ball_OnCollision()

```
static void Ball_OnCollision (
            ball_t *const pBall )  [static]
```

Evento di collisione della palla con il terreno (rimbalzo)

Simula la collisione con il terreno, la velocità è invertita ed è applicato il coefficiente di restituzione dell'urto.

Parametri

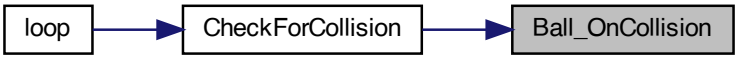
pBall	Puntatore all'oggetto `palla`.

Definizione alla linea 376 del file BouncingLED.ino.

```
00377 {
00378   /* Rimbalza solo se la palla sta cadendo (v positiva) */
00379   if( pBall->speed_mm_s > 0 )
00380   {
00381     pBall->speed_mm_s = -(int16_t)((int32_t)pBall->speed_mm_s *
      COLLISION_COEFFICIENT_OF_RESTITUTION_PERCENT / 100L);
```

```
00382   }
00383 }
```

Questo è il grafo dei chiamanti di questa funzione:



5.1.3.7 Ball_Update()

```
static void Ball_Update (
            ball_t *const pBall,
            unsigned long now )  [static]
```

Aggiorna i dati correnti dell'oggetto `palla`

Aggiorna velocità e posizione della palla in base allo stato precedente ed al tempo trascorso.

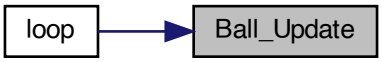
Parametri

pBall	Puntatore all'oggetto `palla`.
now	Ora corrente, tempo assoluto in ms.

Definizione alla linea 347 del file BouncingLED.ino.

```
00348 {
00349   /*
00350    * Nota: l'accelerazione in metri/secondo^2 è equivalente a micrometri^millisecondo^2
00351    */
00352
00353   /* Tempo trascorso dall'ultimo aggiornamento */
00354   unsigned long elapsed_time_ms = now - pBall->lastUpdateTime_ms;
00355
00356   /* Spazio percorso dall'ultimo aggiornamento in micrometri */
00357   /* s = v0 * t ... */
00358   int32_t space_um = pBall->speed_mm_s * elapsed_time_ms;
00359   /* ... + 1/2 * a * t^2 */
00360   space_um += elapsed_time_ms * elapsed_time_ms * GRAVITY_APPROX_M_S2 / 2;
00361
00362   /* Assegnazione parametri correnti */
00363   pBall->lastUpdateTime_ms = now;
00364   pBall->position_mm += space_um / 1000L;
00365   pBall->speed_mm_s += elapsed_time_ms * GRAVITY_APPROX_M_S2;
00366 }
```

Questo è il grafo dei chiamanti di questa funzione:



5.1.3.8 CheckForCollision()

```
static bool CheckForCollision (
            ball_t *const pBall )  [static]
```

Verifica l'evento di collisione della palla con il terreno.

Nel caso di collisione calcola il rimbalzo e restituisce `true`.

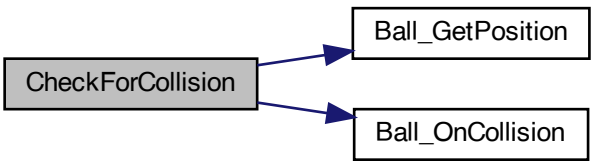
Restituisce

`true` nel caso di collisione avvenuta, `false` altrimenti.

Definizione alla linea 452 del file BouncingLED.ino.

```
00453 {
00454   /* Al termine della striscia la palla rimbalza */
00455   if ( Ball_GetPosition( pBall ) >= STRIP_LENGHT_IN_MM )
00456   {
00457     Ball_OnCollision( pBall );
00458
00459     return true;
00460   }
00461
00462   return false;
00463 }
```

Questo è il grafo delle chiamate per questa funzione:



Questo è il grafo dei chiamanti di questa funzione:

5.1.3.9 GetNextBallColor()

```
static CRGB GetNextBallColor ( )  [static]
```

Determina il prossimo colore della palla.

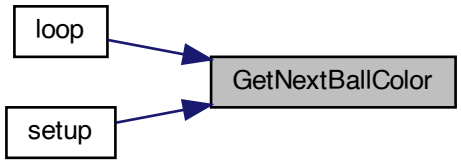
Restituisce

Colore della palla CRGB.

Definizione alla linea 438 del file BouncingLED.ino.

```
00439 {
00440   m_ballCurrentColorIndex = (m_ballCurrentColorIndex+1) % (sizeof(BallColorSequence) /
      sizeof(BallColorSequence[0]));
00441
00442   return BallColorSequence[ m_ballCurrentColorIndex ];
00443 }
```

Questo è il grafo dei chiamanti di questa funzione:



5.1.3.10 InitDisplay()

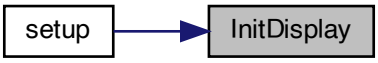
```
static void InitDisplay ( )  [static]
```

Inizializzazione del display.

Definizione alla linea 470 del file BouncingLED.ino.

```
00471 {
00472   m_ballCurrentColorIndex = 0;
00473
00474   FastLED.addLeds<DISPLAY_TYPE, STRIP_PIN>(m_leds, STRIP_NUM_OF_LED);
00475
00476   m_isDisplayToBeRefreshed = false;
00477 }
```

Questo è il grafo dei chiamanti di questa funzione:



5.1.3.11 ISR()

```
ISR (
              TIMER1_COMPA_vect  )
```

Interrupt service routine del timer periodico.

Questa routine si limita ad attivare il flag di segnalazione, il resto dell'elaborazione è demandato al loop principale.

Definizione alla linea 543 del file BouncingLED.ino.

```
00544 {
00545   m_isDisplayToBeRefreshed = true;
00546 }
```

5.1.3.12 loop()

```
void loop ( )
```

Arduino `loop()`

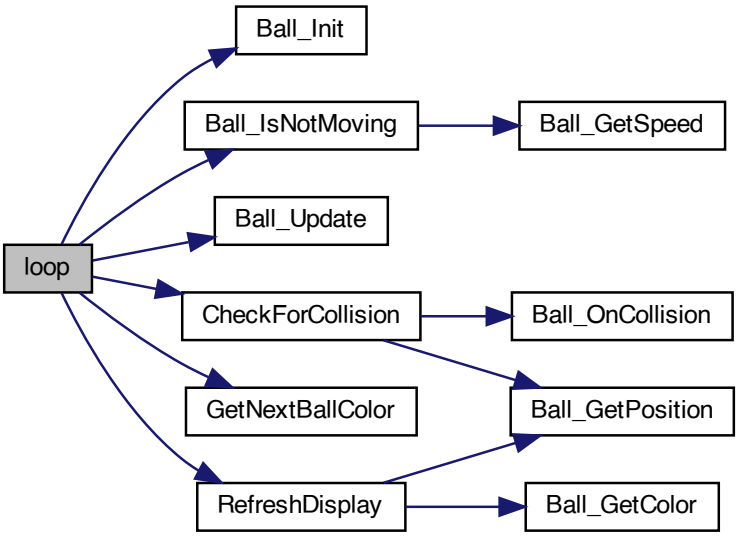
Eseguita per un periodo infinito.

Definizione alla linea 587 del file BouncingLED.ino.

```
00588 {
00589   /* Attende che l'interrupt liberi il semaforo */
00590   if ( m_isDisplayToBeRefreshed )
00591   {
```

```
00592     m_isDisplayToBeRefreshed = false;
00593
00594     /* Ora corrente, in tempo assoluto */
00595     unsigned long now = millis();
00596
00597     /* Aggiorna lo stato della palla */
00598     Ball_Update( &m_ball, now );
00599
00600     /* Aggiorna il display */
00601     RefreshDisplay( &m_ball );
00602
00603     /* Collisione? */
00604     if( CheckForCollision( &m_ball ) )
00605     {
00606       /* Riavvia la simulazione se la palla è ferma, cambiando colore */
00607       if ( Ball_IsNotMoving( &m_ball ) )
00608       {
00609         Ball_Init( &m_ball, now, GetNextBallColor() );
00610       }
00611     }
00612   }
00613
00614   /* Entra in modalità a basso consumo in attesa del prossimo interrupt */
00615   sleep_cpu();
00616 }
```

Questo è il grafo delle chiamate per questa funzione:



5.1.3.13 RefreshDisplay()

```
static void RefreshDisplay (
            ball_t const *const pBall )  [static]
```

Visualizza l'oggetto palla sul display.

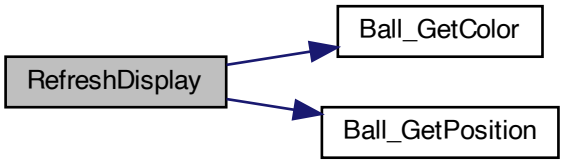
Parametri

ball	Puntatore all'oggetto `palla`.

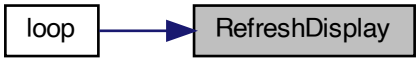
Definizione alla linea 484 del file BouncingLED.ino.

```
00485 {
00486   /* Spegne tutti i led */
00487   FastLED.clear();
00488
00489   /* Calcola il LED da accendere in base alla posizione della palla */
00490   uint32_t ledIndex = (uint32_t)Ball_GetPosition( pBall ) * STRIP_NUM_OF_LED_PER_M / 1000uL;
00491
00492   /* Se il LED è nel display... */
00493   if (ledIndex < STRIP_NUM_OF_LED )
00494   {
00495     /* ...lo accende del colore opportuno */
00496     m_leds[ledIndex] = Ball_GetColor( pBall );
00497   }
00498
00499   /* Aggiorna il display */
00500   FastLED.show();
00501 }
```

Questo è il grafo delle chiamate per questa funzione:



Questo è il grafo dei chiamanti di questa funzione:



5.1.3.14 setup()

```
void setup ( )
```

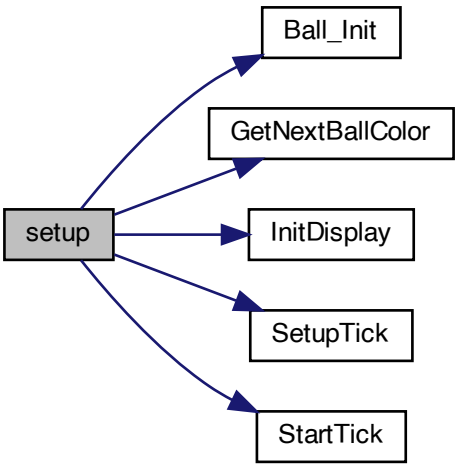
Arduino `setup()`

Eseguita solo all'avvio.

Definizione alla linea 558 del file BouncingLED.ino.

```
00559 {
00560   /* Nessuna interruzione consentita durante l'inizializzazione */
00561   noInterrupts();
00562
00563   /* Imposta lo stato di basso consumo */
00564   set_sleep_mode(SLEEP_MODE_IDLE);
00565
00566   /* Inizializza il display */
00567   InitDisplay();
00568
00569   /* Inizializza l'interruzione periodica */
00570   SetupTick();
00571
00572   /* Inizializza l'oggetto palla */
00573   Ball_Init( &m_ball, millis(), GetNextBallColor() );
00574
00575   /* Riabilita le interruzioni */
00576   interrupts();
00577
00578   /* Avvia il timer periodico */
00579   StartTick();
00580 }
```

Questo è il grafo delle chiamate per questa funzione:



5.1.3.15 SetupTick()

```
static void SetupTick ( )  [static]
```

Preparazione dell'interrupt periodico.

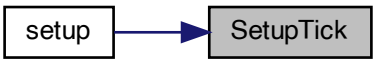
L'interruzione periodica è utilizzata per l'aggiornamento del display.

Il timer utilizzato è `TIM1`.

Definizione alla linea 512 del file BouncingLED.ino.

```
00513 {
00514   /* Timer compare register = F_CPU / (Hz*<prescaler>) - 1 (DEVE essere < 65536) */
00515   constexpr uint16_t timerCompareValue = F_CPU / (REFRESH_RATE_HZ * 1024u) - 1u;
00516
00517   TCCR1A = 0;
00518   TCCR1B = 0;
00519   TCNT1  = 0;
00520   OCR1A = timerCompareValue;
00521
00522   /* Attiva modalità CTC */
00523   TCCR1B |= (1 << WGM12);
00524   /* Imposta prescaler a 1024 */
00525   TCCR1B |= (1 << CS12) | (1 << CS10);
00526 }
```

Questo è il grafo dei chiamanti di questa funzione:



5.1.3.16 StartTick()

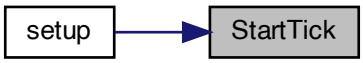
```
static void StartTick ( )  [static]
```

Attiva l'interruzione periodica.

Definizione alla linea 531 del file BouncingLED.ino.

```
00532 {
00533   /* Abilita 'timer compare interrupt' */
00534   TIMSK1 |= (1 << OCIE1A);
00535 }
```

Questo è il grafo dei chiamanti di questa funzione:



5.1.4 Documentazione delle variabili

5.1.4.1 BallColorSequence

```
const CRGB BallColorSequence[]  [static]
```

Valore iniziale:

```
=
{
  CRGB::White,
  CRGB::Red,
  CRGB::Green,
  CRGB::Blue,
  CRGB::Yellow,
  CRGB::Purple,
}
```

Sequenza dei colori della palla.

Definizione alla linea 252 del file BouncingLED.ino.

5.1.4.2 m_ball

```
ball_t m_ball  [static]
```

Oggetto `palla`

Definizione alla linea 311 del file BouncingLED.ino.

5.1.4.3 m_ballCurrentColorIndex

```
uint8_t m_ballCurrentColorIndex = 0  [static]
```

Indice del colore corrente della palla.

Definizione alla linea 317 del file BouncingLED.ino.

5.1.4.4 m_isDisplayToBeRefreshed

```
volatile bool m_isDisplayToBeRefreshed = false  [static]
```

Semaforo usato per la visualizzazione, controllato dal tick periodico.

Definizione alla linea 309 del file BouncingLED.ino.

5.1.4.5 m_leds

```
CRGB m_leds[STRIP_NUM_OF_LED]  [static]
```

Matrice contenente lo stato dei LED della striscia, usata dalla libreria FastLED.

Definizione alla linea 314 del file BouncingLED.ino.

5.2 BouncingLED.ino

Vai alla documentazione di questo file.

```
00001
00199 /* ------------------------------------------------------------------------ */
00200 /* --- Include ---------------------------------------------------------- */
00201 /* ------------------------------------------------------------------------ */
00202
00203 #include <stdint.h>       /* Tipi di dati standard      */
00204 #include <avr/sleep.h>    /* Modalità a basso consumo   */
00205
00206 #include "FastLED.h"      /* Libreria FastLED           */
00207
00208 /* ------------------------------------------------------------------------ */
00209 /* --- Costanti --------------------------------------------------------- */
00210 /* ------------------------------------------------------------------------ */
00211
00212 /* Parametri fisici */
00213
00215 #define GRAVITY_M_S2                                        9.80665f
00216
00218 #define COLLISION_COEFFICIENT_OF_RESTITUTION_PERCENT     85
00219
00220
00221 /* Display */
00222
00224 #define DISPLAY_TYPE                                  NEOPIXEL
00225
00227 #define STRIP_NUM_OF_LED_PER_M                            60u
00228
00230 #define STRIP_LENGHT_IN_MM                              1000u
00231
00233 #define STRIP_PIN                                          6
00234
00236 #define REFRESH_RATE_HZ                                   50u
00237
00238
00239 /* Ottimizzazioni ed approssimazioni locali */
00240
00242 #define GRAVITY_APPROX_M_S2                             ((uint32_t)(GRAVITY_M_S2 + 0.5f))
00243
00245 #define STRIP_NUM_OF_LED                            ((STRIP_NUM_OF_LED_PER_M) * (STRIP_LENGHT_IN_MM) /
       1000uL)
00246
00248 #define MIN_BALL_SPEED_ON_COLLISION_MM_S                500u
00249
00250
00252 static const CRGB BallColorSequence[] =
00253 {
00254   CRGB::White,
00255   CRGB::Red,
00256   CRGB::Green,
00257   CRGB::Blue,
00258   CRGB::Yellow,
00259   CRGB::Purple,
00260 };
00261
00262
00263 /* ------------------------------------------------------------------------ */
00264 /* --- Definizioni di tipo ---------------------------------------------- */
00265 /* ------------------------------------------------------------------------ */
00266
00270 typedef struct BALL_T
00271 {
00272   unsigned long lastUpdateTime_ms;
00273   uint16_t position_mm;
00274   int16_t speed_mm_s;
00275   CRGB color;
00276 } ball_t;
```

```
00277
00278 /* ------------------------------------------------------------------------- */
00279 /* --- Dichiarazione delle funzioni locali ------------------------------------ */
00280 /* --- La documentazione delle funzioni è posta prima dell'implementazione ----- */
00281 /* ------------------------------------------------------------------------- */
00282
00283 /* Controllo dell'oggetto 'palla' */
00284 static void Ball_Init( ball_t * const pBall, unsigned long now, CRGB color );
00285 static void Ball_Update( ball_t * const pBall, unsigned long now );
00286 static void Ball_OnCollision( ball_t * const pBall );
00287 static uint16_t Ball_GetPosition( ball_t const * const pBall );
00288 static uint16_t Ball_GetSpeed( ball_t const * const pBall );
00289 static CRGB Ball_GetColor( ball_t const * const pBall );
00290 static bool Ball_IsNotMoving( ball_t * const pBall );
00291
00292 /* Funzioni relative alla simulazione */
00293 static CRGB GetNextBallColor();
00294 static bool CheckForCollision( ball_t * const pBall );
00295
00296 static void InitDisplay();
00297 static void RefreshDisplay( ball_t const * const pBall );
00298
00299 /* Timer periodico */
00300 static void SetupTick();
00301 static void StartTick();
00302
00303
00304 /* ------------------------------------------------------------------------- */
00305 /* --- Dichiarazione degli oggetti locali ------------------------------------- */
00306 /* ------------------------------------------------------------------------- */
00307
00309 static volatile bool m_isDisplayToBeRefreshed = false;
00311 static ball_t m_ball;
00312
00314 static CRGB m_leds[STRIP_NUM_OF_LED];
00315
00317 static uint8_t m_ballCurrentColorIndex = 0;
00318
00319 /* ------------------------------------------------------------------------- */
00320 /* --- Implementazione delle funzioni locali ---------------------------------- */
00321 /* ------------------------------------------------------------------------- */
00322
00331 static void Ball_Init( ball_t * const pBall, unsigned long now, CRGB color )
00332 {
00333   pBall->lastUpdateTime_ms = now;
00334   pBall->position_mm = 0;
00335   pBall->speed_mm_s = 0;
00336   pBall->color = color;
00337 }
00338
00347 static void Ball_Update( ball_t * const pBall, unsigned long now )
00348 {
00349   /*
00350    * Nota: l'accelerazione in metri/secondo^2 è equivalente a micrometri^millisecondo^2
00351    */
00352
00353   /* Tempo trascorso dall'ultimo aggiornamento */
00354   unsigned long elapsed_time_ms = now - pBall->lastUpdateTime_ms;
00355
00356   /* Spazio percorso dall'ultimo aggiornamento in micrometri */
00357   /* s = v0 * t ... */
00358   int32_t space_um = pBall->speed_mm_s * elapsed_time_ms;
00359   /* ... + 1/2 * a * t^2 */
00360   space_um += elapsed_time_ms * elapsed_time_ms * GRAVITY_APPROX_M_S2 / 2;
00361
00362   /* Assegnazione parametri correnti */
00363   pBall->lastUpdateTime_ms = now;
00364   pBall->position_mm += space_um / 1000L;
00365   pBall->speed_mm_s += elapsed_time_ms * GRAVITY_APPROX_M_S2;
00366 }
00367
00376 static void Ball_OnCollision( ball_t * const pBall )
00377 {
00378   /* Rimbalza solo se la palla sta cadendo (v positiva) */
00379   if( pBall->speed_mm_s > 0 )
00380   {
00381     pBall->speed_mm_s = -(int16_t)((int32_t)pBall->speed_mm_s *
      COLLISION_COEFFICIENT_OF_RESTITUTION_PERCENT / 100L);
00382   }
00383 }
00384
00391 static uint16_t Ball_GetPosition( ball_t const * const pBall )
00392 {
00393   return pBall->position_mm;
00394 }
00395
00402 static uint16_t Ball_GetSpeed( ball_t const * const pBall )
```

```
00403 {
00404   return abs( pBall->speed_mm_s );
00405 }
00406
00413 static CRGB Ball_GetColor( ball_t const * const pBall )
00414 {
00415   return pBall->color;
00416 }
00417
00426 static bool Ball_IsNotMoving( ball_t * const pBall )
00427 {
00428   return ( Ball_GetSpeed(pBall) < MIN_BALL_SPEED_ON_COLLISION_MM_S );
00429 }
00430
00431 /* ---------------------------------------------------------------------------- */
00432
00438 static CRGB GetNextBallColor()
00439 {
00440   m_ballCurrentColorIndex = (m_ballCurrentColorIndex+1) % (sizeof(BallColorSequence) /
       sizeof(BallColorSequence[0]));
00441
00442   return BallColorSequence[ m_ballCurrentColorIndex ];
00443 }
00444
00452 static bool CheckForCollision( ball_t * const pBall )
00453 {
00454   /* Al termine della striscia la palla rimbalza */
00455   if ( Ball_GetPosition( pBall ) >= STRIP_LENGHT_IN_MM )
00456   {
00457     Ball_OnCollision( pBall );
00458
00459     return true;
00460   }
00461
00462   return false;
00463 }
00464
00465 /* ---------------------------------------------------------------------------- */
00466
00470 static void InitDisplay()
00471 {
00472   m_ballCurrentColorIndex = 0;
00473
00474   FastLED.addLeds<DISPLAY_TYPE, STRIP_PIN>(m_leds, STRIP_NUM_OF_LED);
00475
00476   m_isDisplayToBeRefreshed = false;
00477 }
00478
00484 static void RefreshDisplay( ball_t const * const pBall )
00485 {
00486   /* Spegne tutti i led */
00487   FastLED.clear();
00488
00489   /* Calcola il LED da accendere in base alla posizione della palla */
00490   uint32_t ledIndex = (uint32_t)Ball_GetPosition( pBall ) * STRIP_NUM_OF_LED_PER_M / 1000uL;
00491
00492   /* Se il LED è nel display... */
00493   if (ledIndex < STRIP_NUM_OF_LED )
00494   {
00495     /* ...lo accende del colore opportuno */
00496     m_leds[ledIndex] = Ball_GetColor( pBall );
00497   }
00498
00499   /* Aggiorna il display */
00500   FastLED.show();
00501 }
00502
00503 /* ---------------------------------------------------------------------------- */
00504
00512 static void SetupTick()
00513 {
00514   /* Timer compare register = F_CPU / (Hz*<prescaler>) - 1 (DEVE essere < 65536) */
00515   constexpr uint16_t timerCompareValue = F_CPU / (REFRESH_RATE_HZ * 1024u) - 1u;
00516
00517   TCCR1A = 0;
00518   TCCR1B = 0;
00519   TCNT1  = 0;
00520   OCR1A = timerCompareValue;
00521
00522   /* Attiva modalità CTC */
00523   TCCR1B |= (1 << WGM12);
00524   /* Imposta prescaler a 1024 */
00525   TCCR1B |= (1 << CS12) | (1 << CS10);
00526 }
00527
00531 static void StartTick()
00532 {
```

```
00533   /* Abilita 'timer compare interrupt' */
00534   TIMSK1 |= (1 << OCIE1A);
00535 }
00536
00543 ISR(TIMER1_COMPA_vect)
00544 {
00545   m_isDisplayToBeRefreshed = true;
00546 }
00547
00548
00549 /* ------------------------------------------------------------------------- */
00550 /* --- Implementazione delle funzioni predefinite di Arduino --------------- */
00551 /* ------------------------------------------------------------------------- */
00552
00558 void setup()
00559 {
00560   /* Nessuna interruzione consentita durante l'inizializzazione */
00561   noInterrupts();
00562
00563   /* Imposta lo stato di basso consumo */
00564   set_sleep_mode(SLEEP_MODE_IDLE);
00565
00566   /* Inizializza il display */
00567   InitDisplay();
00568
00569   /* Inizializza l'interruzione periodica */
00570   SetupTick();
00571
00572   /* Inizializza l'oggetto palla */
00573   Ball_Init( &m_ball, millis(), GetNextBallColor() );
00574
00575   /* Riabilita le interruzioni */
00576   interrupts();
00577
00578   /* Avvia il timer periodico */
00579   StartTick();
00580 }
00581
00587 void loop()
00588 {
00589   /* Attende che l'interrupt liberi il semaforo */
00590   if ( m_isDisplayToBeRefreshed )
00591   {
00592     m_isDisplayToBeRefreshed = false;
00593
00594     /* Ora corrente, in tempo assoluto */
00595     unsigned long now = millis();
00596
00597     /* Aggiorna lo stato della palla */
00598     Ball_Update( &m_ball, now );
00599
00600     /* Aggiorna il display */
00601     RefreshDisplay( &m_ball );
00602
00603     /* Collisione? */
00604     if( CheckForCollision( &m_ball ) )
00605     {
00606       /* Riavvia la simulazione se la palla è ferma, cambiando colore */
00607       if ( Ball_IsNotMoving( &m_ball ) )
00608       {
00609         Ball_Init( &m_ball, now, GetNextBallColor() );
00610       }
00611     }
00612   }
00613
00614   /* Entra in modalità a basso consumo in attesa del prossimo interrupt */
00615   sleep_cpu();
00616 }
```

Indice analitico