Schede didattiche di Otto Cardy

8 schede didattiche

primaria

secondaria

35-45 min

Esercizi guidati per apprendere l'uso del coding a blocchi per il controllo del movimento e dei suoni del robot **Otto Cardy.**

S 1.0 Piano della lezione per l'insegnante

1. Preparazione
   - Puoi preparare alcuni esempi (*sketch*) di movimento scelti fra i file **Examples** di *OttoBlockly*.
   - Individua delle varianti alle consegne da assegnare ai singoli gruppi

2. Coinvolgimento (5 min.)
   - Fai una panoramica dei movimenti e suoni forniti dal programma che è possibile aggiungere.
   - Presenta la funzione del "sensore di profondità" e la distanza che rileva
   - Fai notare l'altezza delle note e la lunghezza del suono.

3. Esplorazione (15 min.)
   - Invita gli studenti a lavorare a coppie o in gruppi
   - Invita a elaborare una strategia di lavoro utilizzando le funzioni di **suoni-movimenti-danza-distanza**.

4. Elaborazione (20 min.)
   - Invita gli studenti a programmare dei movimenti con utilizzo dei **suoni** e dei **gesti**.
   - Chiedi di inserire delle sequenze musicali di loro gradimento fra i movimenti del robot

5. Valutazione
   - Crea una scheda di valutazione dei lavori dei singoli studenti prendendo spunto dalle sezioni di valutazione fornite.

Supporti per l'insegnante

Obiettivi principali
Gli alunni saranno in grado di controllare i movimenti ed aggiungere i suoni

Tempo: 20-30 min
Gruppo di lavoro: tutta la classe

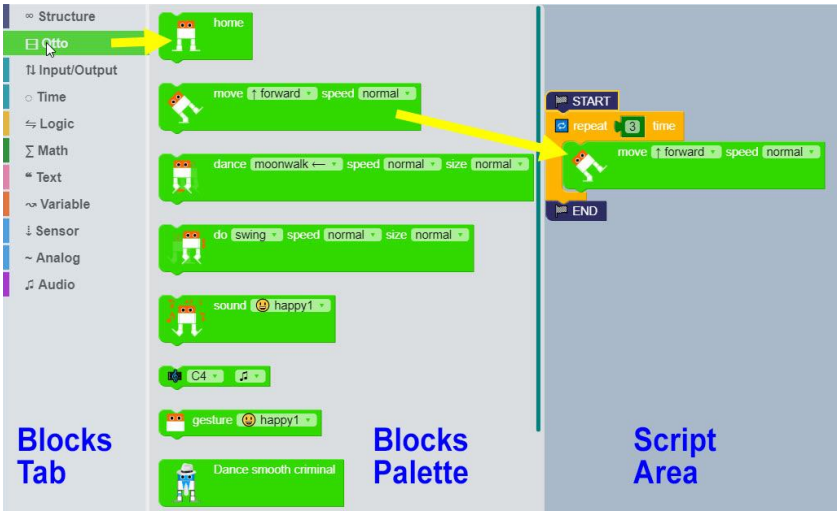
Materiali: Otto Cardy + OttoBlockly

Rubrica delle competenze
Competenze di asse: matematica
- individuare le strategie appropriate per la soluzione di problemi
-

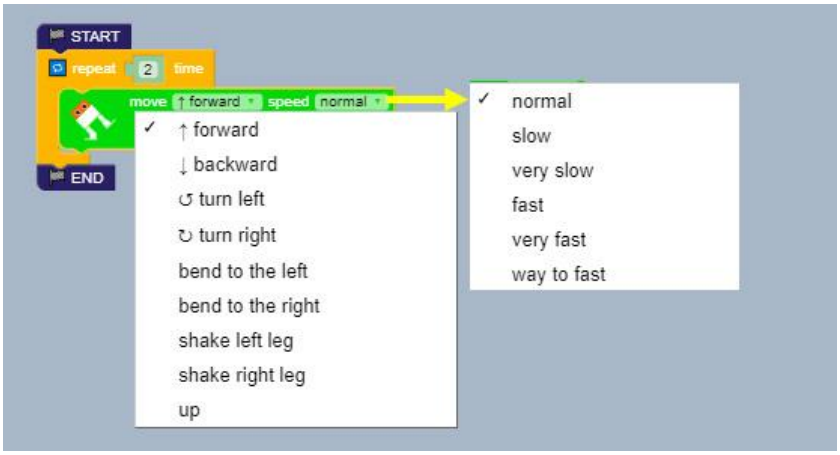
Competenze di asse: informatica
- scegliere strategie utili alla soluzione di un problema
- elaborare semplici istruzioni per controllare il comportamento del robot

S1.1 Insegniamo a camminare al robot

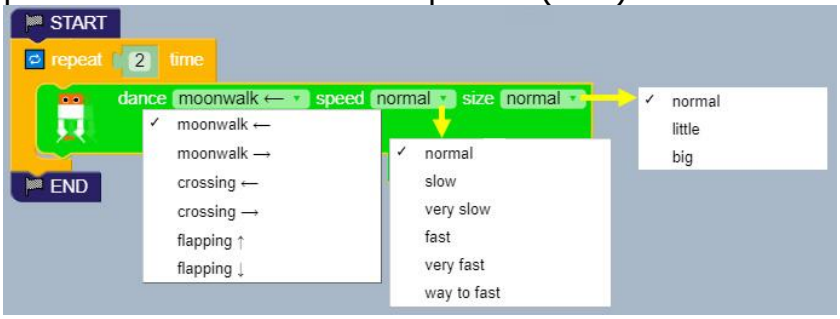
Nel menu *Blocks Tab* scegliere la voce ***Otto*** che fornisce, nella *Blocks Palette,* alcuni blocchi con movimenti già predisposti.



Movimenti associati al blocco **MOVE**: ad ogni movimenti si può assegnare una velocità di esecuzione.



Movimenti del blocco **DANCE:** oltre alla velocità si può controllare anche l'ampiezza (*size*) dell'azione.

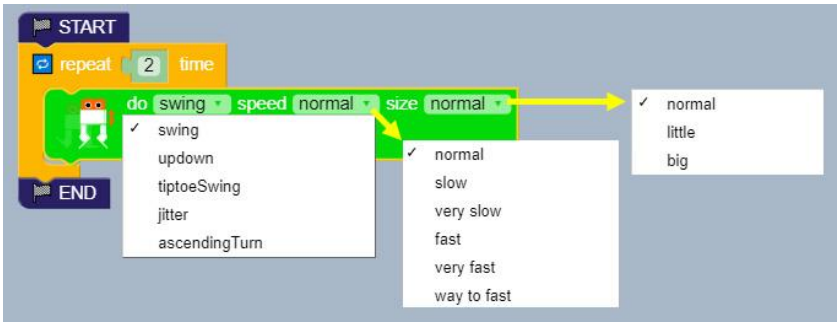


Supporti per l'insegnante

Movimenti di OttoBlockly

- move: si muove in avanti, in dietro, si piega e si solleva sulle punte;
- dance: sono n.3 tipi di danza quali *moonwalk*, *crossing* e *flapping*;
- do: muove le gambe con i seguenti movimenti di *swing*, *updown*, *tiptoeSwing*, *Jitter* e *ascendingTurn*;
- gesture e sound: sono due elenchi di *gesti* e *suoni* che possono essere usati anche combinati insieme.

Il blocco **DO** aggiunge dei semplici gesti come strisciare, alzarsi sulle punte ed altri ancora: ad ogni movimento si può assegnare una velocità e un'ampiezza di esecuzione.



I blocchi **SOUND** e **GESTURE** permettono di inserire alcuni *suoni* del cicalino e dei *g esti* che possono essere coordinati fra di loro.

Movimenti del blocco **dance:** oltre alla velocità si possono controllare anche l'ampiezza (*size*) dell'azione.

Supporti per l'insegnante

Suoni e movimenti coordinati

Associare dei movimenti a dei suoni può rappresentare una sfida e allo stesso tempo un'occasione di discussione sui significati da dare a suoni e movimenti.

Ad esempio, che movimento associare al suono "surprise" potendo disporre solo del movimento delle gambe e dei piedi?

Si potrebbe iniziare con la simulazione dal vero associando i suoni di Otto con i movimenti reali delle gambe dei ragazzi. Sarebbe divertente anche associare i suoni ai movimenti reali delle gambe e dei piedi.

E se il robottino avesse anche le braccia? Che movimenti si potrebbe associare ad ogni suono?

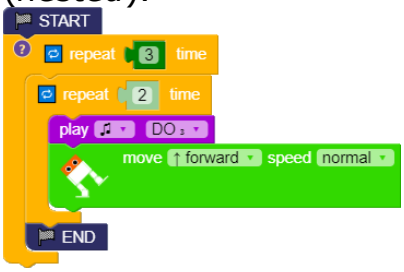
S 1.2 Inserire i movimenti

Se si inserisce un solo blocco nell'area *Script*, esempio **MOVE,** il programma esegue l'istruzione all'infinito.

Per interrompere un'istruzione o una serie di istruzioni s'inserisce l'istruzione **END**. L'istruzione di blocco ha in genere una durata di un secondo.

Il blocco **REPEAT** (LOOP) fornisce un ciclo di istruzioni contenute al suo interno per un numero determinato di volte. Alla fine passa all'istruzione successiva e se non la trova ripete all'infinito.
Per interrompere la sequenza va inserito il blocco **END**. Il blocco **START** non è necessario.

Si può anche inserire due loop uno dentro l'altro (*nested*).



Codice Arduino
(senza le righe di testa)

```
void loop() {
  Otto.walk(1,1000,1); //
FORWARD
}
```

```
void loop() {
  Otto.walk(1,1000,1); //
FORWARD
  while(true);
}
```

```
void loop() {
  for (int count=0 ; count<3 ;
count++) {
    Otto.walk(1,1000,1); //
FORWARD
  }
 while(true);
}
```

Esercizi 1.2

1. Inserire due movimenti con una pausa di 1 secondo fra i due;
2. Quante volte viene suonata la nota nello schetch riportato a fianco?

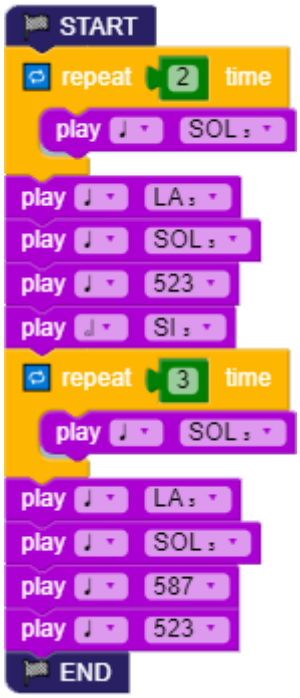
S 1.3 Inserire i suoni

Il Menu Audio permette di inserire delle note attraverso il *cicalino*, quindi di non grande qualità sonora, oppure caricare un file mp3 se è installato un lettore con altoparlante.

1. 

Il blocco **PLAY** permette di inserire una nota controllano sia la sua durata che l'altezza. L'altezza delle note si realizza in due ottave.

Nell'esempio seguente sono visualizzate in sequenza le note corrispondenti alla motivo musicale ***Happy Birday*.** Notare il blocco **REPEAT** utilizzato due volte per prolungare le note del ritornello.



Codice Arduino

```
void loop() {
  tone(13,523,2000);
  delay(2000);
}
```

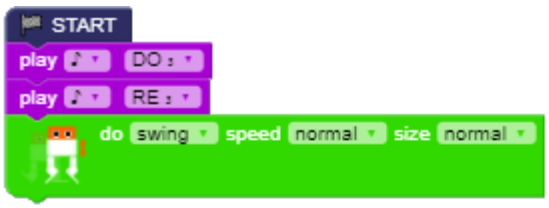
```
void loop() {
  for (int count=0 ; count<2 ;
count++) {
tone(3,392,500);delay(500);}
tone(13,440,500);delay(500);
tone(13,392,500);delay(500);
tone(13,523,500);delay(500);
tone(13,493,1000);delay(1000);

for (int count=0 ; count<3 ;
count++) {
tone(13,392,500);delay(500);}
tone(13,440,500);delay(500);
tone(13,392,500);delay(500);
tone(13,587,500);delay(500);
tone(13,523,500);delay(500);
  while(true);
}}
```

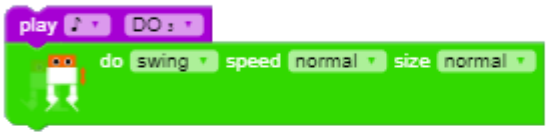
Esercizi 1.3

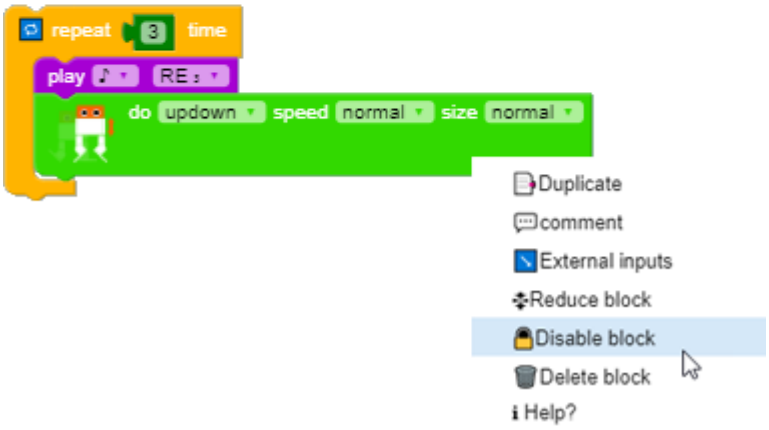
1. Aggiungere le note musicali che completano il motivetto
2. Comporre un nuovo motivo fra quelli che ti piacciono

S 1.4 Regole dei blocchi

Per testare i diversi movimenti può essere utile inserire delle note all'inizio di ogni azione.
A differenza di **Scratch** e di altri programmi a blocchi, tutti i blocchi presenti nella ***Script Area*** vengono processati anche se non sono collegati tra di loro come in figura.
Infatti se osserviamo il codice Arduino a lato, tutte le linea del codice sono attive anche se i blocchi non sono collegati.







Cliccando il tasto destro del mouse sopra un blocco, si apre una finestra di selezione con alcune funzioni di scelta, come *Duplicate* o *Disable block* per disattivare le istruzioni di un blocco senza cancellarlo.
Per *duplicare* un blocco è sempre possibile usare le funzioni di copia di Windows con *Ctrl+C* e *Ctrl+V* per il *copia-incolla*.

Codice Arduino

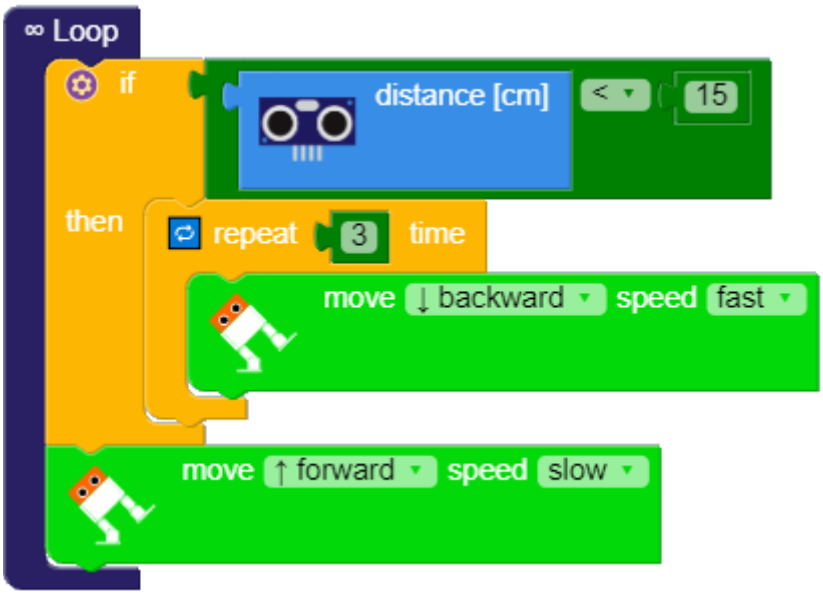
```
void loop() {
// primo blocco
  tone(13,261,250); delay(250);
  tone(13,293,250); delay(250);
  Otto.swing(1, 1000, 25);

// secondo blocco
  tone(13,392,250); delay(250);
  Otto.updown(1, 1000, 25);

// terzo blocco
  for (int count=0; count<3 ; coun
t++) {
    tone(13,440,250);delay(250);
    Otto.jitter(1, 1000, 25);
    }
}
```

Esercizi 1.4

1. Inserire un commento in un blocco
2. Cosa modifica la funzione "External inputs"?

S 1.5 Sensore di distanza

OttoBlockly fornisce anche il blocco **distance** per controllare la distanza, in centimetri, tramite il **sensore di prossimità** che rileva, mediante ultrasuoni, lo spazio fra il sensore e un oggetto.



Per controllare la **distanza**, si usa il blocco **IF** che verifica se la distanza è inferiore ad un valore dato, nella figura 15 cm. Se la condizione è vera, allora esegue le istruzioni interne altrimenti salta alli blocchi seguenti. In questo caso inverte la camminata andando indietro per 3 secondi (*time*). Se non si verifica, cioè se non ci sono ostacoli, Otto prosegue la sua camminata.

A fianco viene fornito il codice dello sketch con due note diverse che avvisano quando rileva l'ostacolo e quando poi termina la marca all'indietro.

Codice Arduino

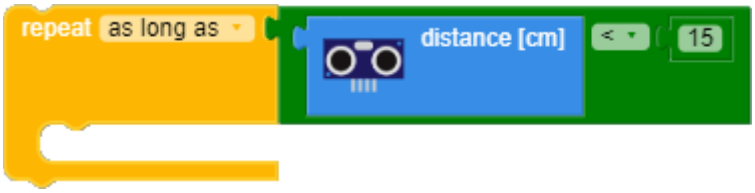
```
void loop()
  {if (Otto.getDistance() < 15)
     {for (int count=0 ; count<3
; count++)
     {Otto.walk(1,750,-1); //
BACKWARD }
        }
    Otto.walk(1,2000,1); //
FORWARD
  }

void loop() {
  if (Otto.getDistance() < 15)
     {tone(13,261,1000);
delay(1000);
       for (int count=0 ;
count<3 ; count++)
 {Otto.walk(1,750,-1);//
BACKWARD
}
       tone(13,349,1000);
delay(1000);}
      Otto.walk(1,2000,1); //
FORWARD
  }
```

Esercizi

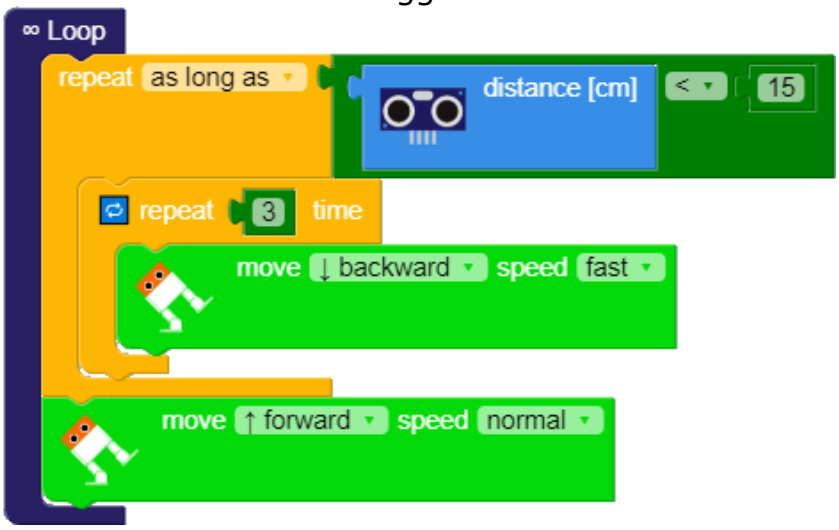
1. Inserire un avviso sonoro (*nota*) quando Otto rileva un ostacolo.
2. Inserire un altro avviso

S 1.6 Condizione di distanza

Per **ripetere** una serie di blocchi a seconda di una condizione, come ad esempio fino a quando la distanza rimane minore di 15 cm, si può usare il blocco **repeat as long as** al posto del blocco precedente **IF**. Corrisponde alla comune funzione di programmazione **WHILE.**

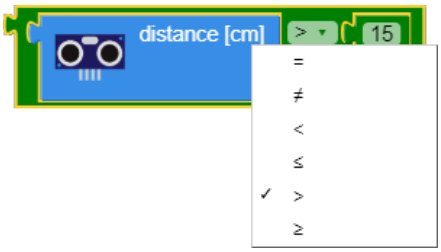


Nel nostro caso al posto del blocco **IF** possiamo utilizzare il blocco **REPEAT** per fa arretrare Otto finché la distanza con l'oggetto non aumenta.



La struttura è simile alla precedente dove è stato modificato solo il blocco **REPEAT**. Nel codice a fianco notare che l'istruzione **IF** è stata sostituita con **WHILE**.

E' possibile modificare anche le **condizioni di distanza** per modificare il movimento quando la condizione di realizza.



Codice Arduino

```
while (Otto.getDistance() < 15) {
  Otto.walk(1,1000,1); // FORWARD
 }
```

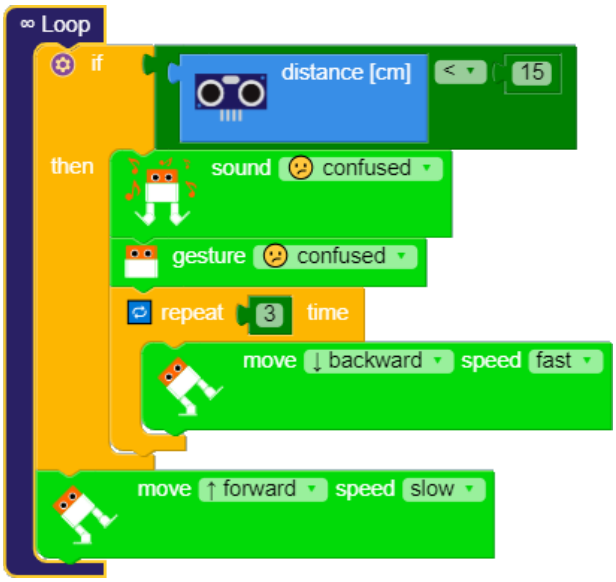
```
void loop() {
  while (Otto.getDistance() < 15)
{
  for (int count=0 ; count<3 ;
count++) {
  Otto.walk(1,750,-1); // BACKWARD
    }
  }
 Otto.walk(1,1000,1); // FORWARD
}
```

- Esercizi 1.6

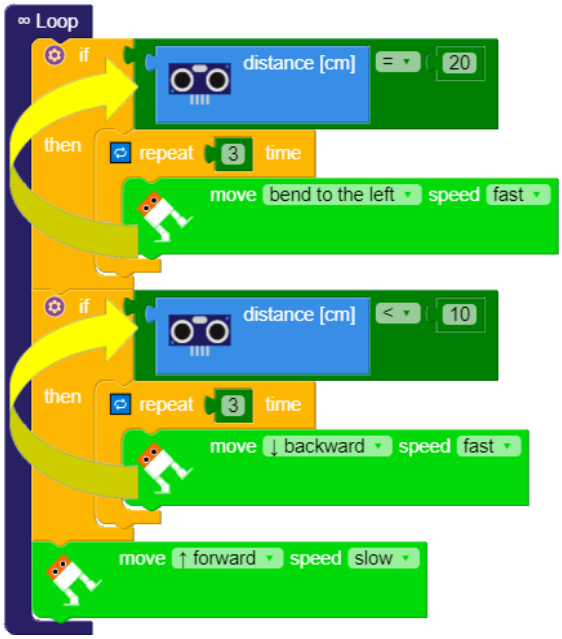
1. Modificare la distanza e cambiare la direzione di Otto
2. Se modifichiamo la condizione di distanza con **>15** cosa cambia?

S 1.7 Blocchi *sound* e *gesture*

Per enfatizzare l'incontro con un ostacolo si possono aggiungere dei suoni più articolati come ad esempio un *sound* fra quelli disponibili e un *gesture* di sorpresa per l'inconveniente dell'ostacolo.



Se vogliamo inserire due condizioni di distanza, ad esempio a 20 cm esegue un'azione mentre a 10 cm un'altra, allora dobbiamo inserire due blocchi IF come in figura:



Codice Arduino

```
void loop()
{if (Otto.getDistance() < 15)
   {Otto.sing(S_confused);
Otto.playGesture(OttoConfused);
    for (int count=0 ; count<3 ;
count++)
   { Otto.walk(1,750,-1); //
BACKWARD}}
    Otto.walk(1,2000,1); //
FORWARD
}

void loop()
{if (Otto.getDistance() == 20)
   {for (int count=0 ; count<3 ;
count++)
   {Otto.bend(1,750,1);  }}
  if (Otto.getDistance() < 10)
     {for (int count=0 ; count<3
; count++)
    {Otto.walk(1,750,-1); //
BACKWARD }}
  Otto.walk(1,2000,1); //
FORWARD
}
```

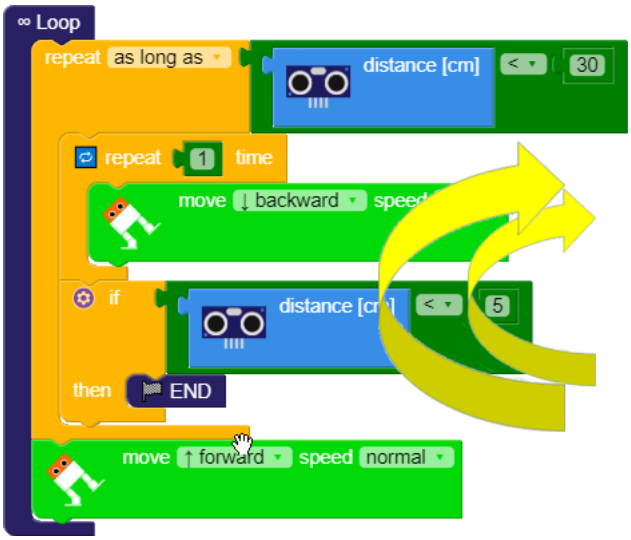
- Esercizi 1.7

1. Perché nella seconda figura è stato inserita la condizione **=20** e non **<20**?
2. Proponi un'altra soluzione per avere **due** condizioni di distanza: a **10** e **20**

S 1.8 Condizioni di doppio ciclo

E' possibile utilizzare due o più cicli uno interno all'altro (*cicli annidati*) per ripetere una serie di operazioni integrate.

Nel nostro caso abbiamo inserito un blocco interno di **END**: se la distanza è inferiore a 5 cm. il robot Otto si ferma.



L'esempio non è molto elegante dal punto di vista della programmazione ma è un esempio di come bloccare l'esecuzione del programma.

Le condizioni inserite sono le seguenti:

- **SE** la distanza è **<** (*minore*) di 30 cm. allora arretra;
- **SE** la distanza è **<** (*minore*) di 5 cm. allora si ferma (STOP).

Notare che la prima condizione è quella della distanza maggiore che viene verificata per prima e dopo, solo verifica se è vera anche la seconda condizione, distanza **<5**.

Codice Arduino

```
void loop()
{ while
  (Otto.getDistance() < 30)
     {for (int count=0 ;
count<3 ; count++)
     {Otto.walk(1,750,-1); //
BACKWARD }
  if (Otto.getDistance() < 5)
      {while(true);}}
  Otto.walk(1,1000,1); //
FORWARD
}
```

- Esercizi 1.8

1. E' possibile inserire un altro blocco **repeat as long as**, come nella prima condizione, al posto del blocco **IF**?
2. Trova un'altra soluzione

S 2.0 PROGETTI FINALI

Il progetto finale riguarda la realizzazione di **una applicazione** realizzata con **OttoBlockly** nel creare movimenti, suoni o gesti del robottino.

Vediamo alcune idee di progetto:

1. Aggiungere al robot la capacità di **rilevare** le distanze attraverso una serie di suoni diversi a seconda della distanza rilevata;
2. Associare dei nuovi **movimenti** e **suoni** ideati dagli studenti;
3. Creare delle **variabili** che controllano suoni e movimenti;
4. Aggiungere un **led** che si illumina quando il robot si muove in avanti;
5. Creare degli **abbigliamenti** al robot son tessuti o carta secondo delle storie da inventare;
6. Lavorando a gruppi, sceneggiare delle **storie**, fantastiche o di soggetto storico, dove i personaggi sono i robot;
7. Aggiungere le **braccia** al robot con altri due servomotori;
8. Aggiungere un **display** 8x8 per visualizzare messaggi e testi;
9. Aggiungere un controllo **Bluetooth** per poter controllare i movimenti con il cellulare;
10. Sviluppare applicazioni che fanno svolgere al robot dei **movimenti complessi** interagendo con i vari sensori.

Sviluppi del progetto

La costruzione del robot **Otto Card** permette ai ragazzi di aver la conoscenza degli aspetti principali della costruzione di un **oggetto** funzionante con arduino.
Le possibilità di approfondimento sono molteplici: forniamo qui alcuni link di siti dove trovare informazioni tecniche, didattiche e idee per sviluppi di applicazioni varie.

Siti del progetto Otto:
https://www.ottodiy.com/
- https://ottoschool.com/scratch/
- https://wikifactory.com/+OttoDIY/otto-diy

Sul sito si possono trovare i riferimenti per acquistare gli altri componenti da aggiungere al robot base.