

# Laboratorio di Progettazione di Sistemi Software

## Progetto: Nxt-Robot

Riccardo Solmi

# Definizione del problema

---

- **Progettare ed implementare un robot virtuale programmabile**
  - Il linguaggio di programmazione per robot è definito informalmente in questo documento a meno di qualche parte opzionale o sottospecificata.
  - Ciascun gruppo *deve* decidere come è fatto il robot ed il suo ambiente di esecuzione compresi la disposizione e il comportamento di sensori e attuatori.
- **Scopo del progetto**
  - Progettare ed implementare il linguaggio dei robot: nxt-robot.
  - Progettare ed implementare l'ambiente per robot scelto.
  - Implementare un interprete per il linguaggio che funzioni nell'ambiente scelto.
  - Scrivere tanti programmi nxt-robot di esempio quanti sono i membri del gruppo.

# Approfondimenti

- Robot programmabili <http://mindstorms.lego.com/> e <http://www.ni.com/academic/mindstorms/>
- Tutorial del linguaggio Nxt-G [http://www.ortop.org/NXT\\_Tutorial/index.html](http://www.ortop.org/NXT_Tutorial/index.html)
- Materiale didattico <http://legoengineering.com/>
- Video di robot in azione <http://nxtasy.org/>
- Costruzione di robot <http://www.nxtprograms.com/>
- Esempi Nxt-G <http://home.earthlink.net/~xaos69/NXT/>
- <http://www.nxt-mindstorms.com/wiki/>
- <http://www.hightechkids.org>

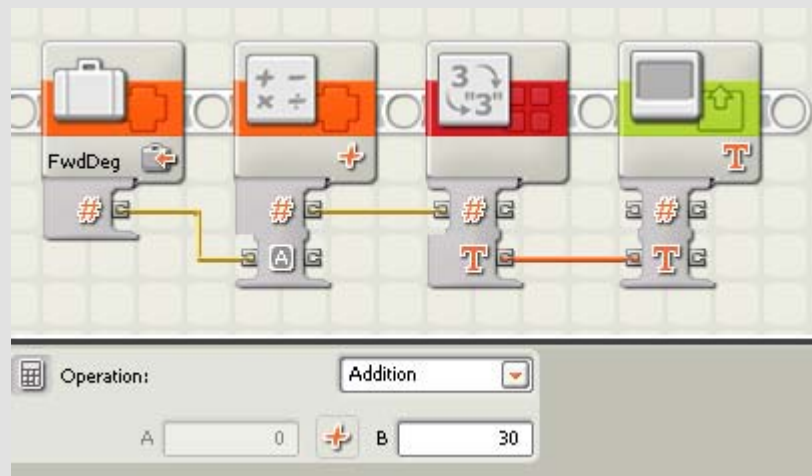


# Linguaggio Nxt-robot

---

- È un linguaggio ispirato a Nxt-G.
  - Le immagini nei lucidi che seguono fanno parte della notazione di Nxt-G
- **Caratteristiche:**
  - Assenza di una notazione concreta (si programma usando le API creazionali)
  - Capacità di esprimere esplicitamente sia il flusso dei dati che il flusso del controllo.
  - Pochi e semplici tipi di dati.
  - Costrutti specifici per gestire i sensori e gli attuatori dei robot.
- **Costrutti (chiamati blocchi) generali:**
  - Sequence, Parallel, Switch, Loop
  - Variable, MyBlock
  - Compare, Logic, Math, Range
  - NumberToText, Text, Random

# Esempio di programma (frammento)



# Data types

---

- **Text**
  - Rappresenta una stringa.
- **Number**
  - Rappresenta un intero positivo o negativo.
- **Logic**
  - Rappresenta un valore booleano.

# Data wires, hubs e plugs

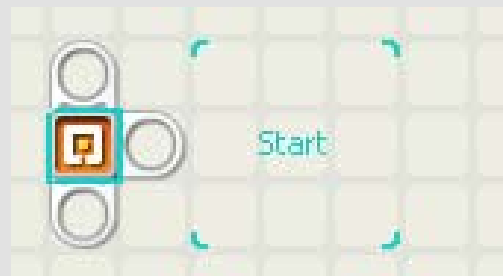
---

- **Permettono di esplicitare il flusso dei dati (dataflow)**
- **Data Wire**
  - È un canale di comunicazione dati mono direzionale.
  - Quando un dato arriva all'ingresso la connessione lo trasporta all'uscita.
  - Gli estremi di una connessione devono essere entrambi ancorati.
- **Data Plug**
  - È il nome dato alla sede dove è possibile ancorare zero o più data wires in uscita oppure zero o uno in ingresso.
  - Fa sempre parte di una struttura capace di fornire o ricevere un dato.
- **Data Slot**
  - È un contenitore per un valore di un tipo dato (e non modificabile).
  - Uno slot può essere senza valore.
- **Data Hub**
  - È una struttura composta da un input Plug, uno Slot e un output Plug.
  - È possibile assegnare un valore allo slot.
  - Quando arriva un valore in input viene (sovra)scritto nello slot.
  - Ogni volta che viene eseguito, se lo slot ha un valore questo viene consumato e propagato in output altrimenti
    - viene segnalato un errore oppure
    - viene atteso un valore in input. [OPZIONE supporto concorrenza]

# Programma

---

- Un Programma ha un nome, una descrizione opzionale, una sequenza di dichiarazioni e un blocco che rappresenta il corpo del programma.
- Ci sono due tipi di dichiarazioni:
  - Variable Declaration
    - Ha un nome e un tipo (data type).
  - MyBlock Declaration [OPZIONE supporto costrutti definibili]
    - Ha un nome, una descrizione opzionale, un insieme ordinato di Hubs e un blocco.
    - Gli Hubs rappresentano gli input/output del MyBlock.
    - Il blocco rappresenta la definizione del comportamento del MyBlock.



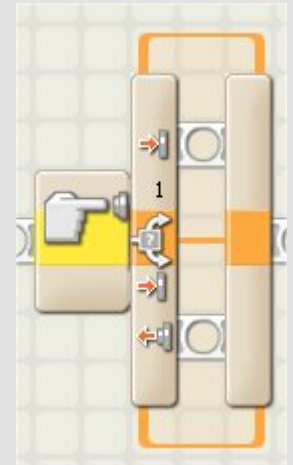
# Blocchi di composizione

- **Sequence**
  - Rappresenta una sequenza di blocchi da eseguire in ordine.
  - L'esecuzione di una Sequence termina quando termina l'ultimo blocco.
- **Parallel [OPZIONE supporto concorrenza]**
  - Rappresenta un insieme di blocchi da eseguire in un ordine non specificato.
  - L'esecuzione di un Parallel termina quando tutti i blocchi sono terminati [OPZIONE si può scegliere un'altra semantica ad esempio fork].



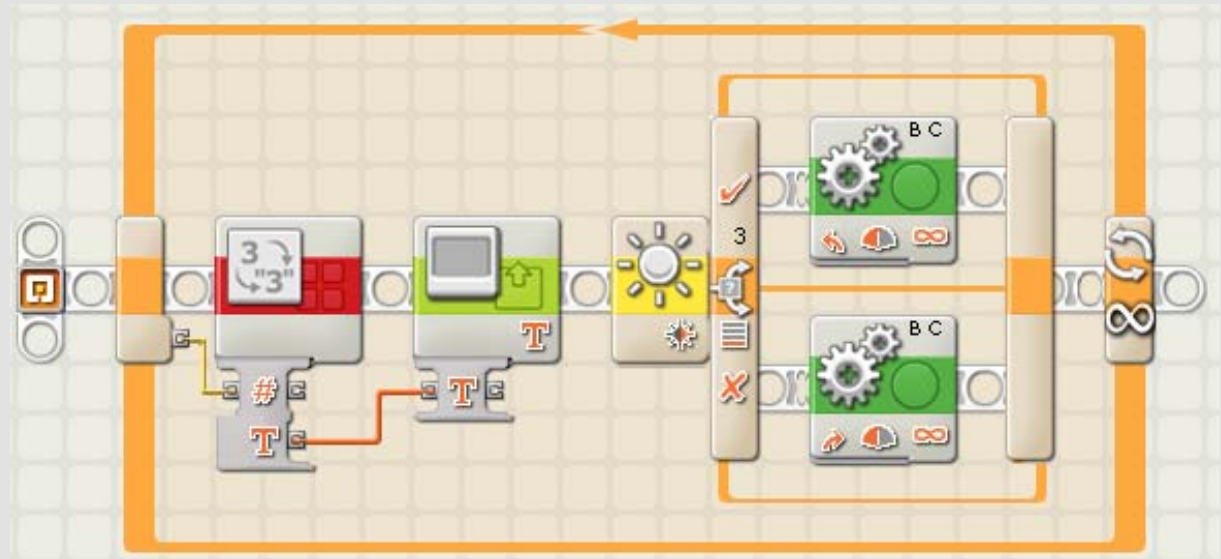
# Blocco di scelta: Switch

- Permette di scegliere un percorso (path) da eseguire tra un insieme ordinato di percorsi dato.
- Ogni percorso è governato da una condizione rappresentata da un valore.
- Ha un *input Plug* per ricevere un valore.
- Ha un *tipo* che vincola il valore in ingresso e il valore delle condizioni.
- Il valore di input (Plug) viene confrontato con la condizione di ciascun percorso finché non se ne trova una uguale.
- E' possibile marcare un percorso come default.
- Se nessuna condizione viene soddisfatta viene eseguito il percorso di default se presente altrimenti lo switch termina.



# Blocco di ripetizione: Loop

- Permettere di ripetere un blocco.
- Il comportamento di un blocco è determinato da un loopControl che può assumere i valori: forever, times, logic.
- Ha due Hub: numero di ripetizioni (per times), e logic (per logic).
- Ha un output Plug per comunicare il numero di iterazione in corso (da 0).



# Blocco Variable

---

- Rappresenta l'accesso ad una variabile definita nella parte di dichiarazioni di un programma.
- Ha una azione, un nome e un Hub per il valore.
- L'azione determina l'uso della variabile: lettura, scrittura
- Il nome indica la variabile su cui verrà eseguita l'azione
- L'Hub viene utilizzato per ricevere e propagare il valore della variabile.
- Se l'azione è di scrittura
  - È possibile connettere un input Hub oppure assegnare un valore allo slot.
  - Ogni volta che viene eseguito, il valore contenuto nell'Hub viene scritto nella variabile e poi viene propagato in output.
- Se l'azione è di lettura
  - È possibile connettere solo un output all'Hub.
  - Ogni volta che viene eseguito, viene propagato in output il valore della variabile.



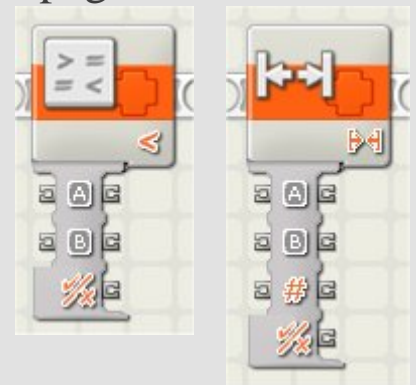
# Blocco MyBlock

---

- [OPZIONALE supporto costrutti definibili]
- Rappresenta una chiamata ad un MyBlock definito nella parte di dichiarazioni del programma.
- Ha un nome e un insieme ordinato di Hubs.
- Se non esiste la definizione di un MyBlock con il nome richiesto deve essere segnalato un errore.
- Se gli Hubs non corrispondono in numero e tipo a quelli della definizione di MyBlock deve essere segnalato un errore.
- Gli Hubs servono per scambiare i parametri e i risultati.

# Blocchi Compare e Range

- **Compare**
  - Ha un selettore di operazione: uguale, maggiore di, minore di.
  - Confronta il valore di due Hub numerici (A e B) e propaga il risultato su un output Plug logico.
- **Range**
  - Ha un selettore di operazione: inside, outside.
  - Ha due Hub numerici (A, B) per definire gli estremi dell'intervallo
  - Ha un test value Hub numerico per il valore da testare.
  - Confronta il test value con gli estremi dell'intervallo e propaga il risultato su un output Plug logico.



# Blocchi Logic e Math

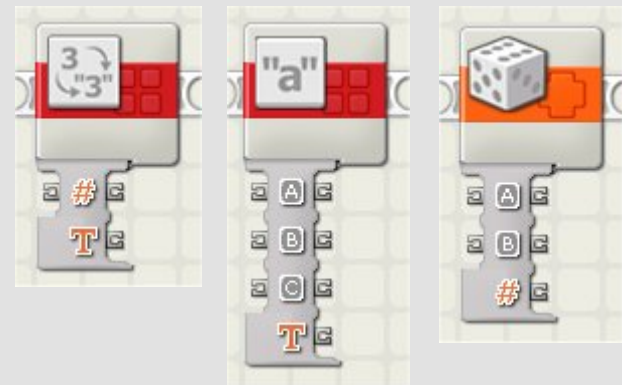
---

- **Logic**
  - Ha un selettore di operazione: and, or, xor, not.
  - Confronta il valore di due Hub logici (A e B) e propaga il risultato su un output Plug logico.
- **Math**
  - Ha un selettore di operazione: +, -, \*, /.
  - Esegue l'operazione richiesta usando due Hub numerici (A e B) come argomenti e propaga il risultato su un output Plug numerico.



# Blocchi NumberToText, Text e Random

- **NumberToText**
  - Ha un Hub numerico e un output Plug testuale.
  - Converte il numero in una stringa da propagare in output.
- **Text**
  - Ha tre Hub testuali (A, B, C) e un output Plug testuale.
  - Appende in sequenza il contenuto di A, B e C e lo propaga in output.
- **Random**
  - Ha due Hub numerici (A, B) per definire gli estremi dell'intervallo
  - Genera un numero compreso tra gli estremi dell'intervallo (inclusi) e lo propaga su un output Plug numerico.



# Idee per altri blocchi generali

---

- Un gruppo *può* decidere di aggiungere dei tipi di blocchi generali a quelli richiesti al fine di facilitare la scrittura del codice.
- Si richiede di mantenere inalterato il livello di astrazione e lo stile del linguaggio.
- **Proposte per nuovi tipi di blocchi:**
  - TextToNumber – Complementare di NumberToText
  - TextSplit – Complementare di Text
  - Stop – Per fermare l'esecuzione o uscire da un loop
  - Wait – Per attendere un tempo prefissato
  - ArrayVariable – Per supportare gli array come data type. Simile a Variable con un Hub numerico in più per l'indice.
  - RecordVariable – Per supportare i record come data type. Simile a Variable con un Hub stringa in più per il campo.
  - RecordPlayBack – Per memorizzare una sequenza di spostamenti degli attuatori in modo da comunicarla ad un altro robot o per ripeterla.

# Blocchi sensori e attuatori

---

- Ogni gruppo *deve* aggiungere un certo numero di tipi di sensori e attuatori in base alle esigenze del proprio progetto.
- Nei lucidi che seguono vengono illustrate alcune idee da cui si può attingere liberamente.
- Si possono installare più sensori di uno stesso tipo sul robot.
- Si consiglia di mettere un selettore di “porta” nei blocchi sensori/attuatori in modo da poter scegliere l’occorrenza del dispositivo da controllare con il blocco.
- Si richiede di definire i blocchi tenendo presente la loro natura di sensori/attuatori da installare in un robot.

# Idee per blocchi sensori e attuatori

---

- **Blocco Display**
  - Ha un Hub testuale per il messaggio da inviare sul display
- **Blocco Keyboard o NumPad**
  - Ha un selettore di tipo (data type) e un output Plug per i dati letti.
- **Blocco Motor**
  - Ha un selettore per controllare la direzione di rotazione
  - Ha un Hub numerico per controllare il numero di rotazioni da eseguire.
- **Blocco Pen**
  - Ha un Hub logico per controllare l'attivazione della penna e due Hub numerici per scegliere il colore e la dimensione della punta.
- **Blocco Message**
  - Ha un selettore di operazione: Send o Receive.
  - Ha due Hub testuali: uno per il messaggio e l'altro per scegliere il mittente/destinatario.

# Idee per blocchi sensori e attuatori

---

- **Blocco GPS**
  - Ha tre output Plug numerici (X, Y, D) usati per propagare la posizione del robot e la sua direzione ogni volta che il controllo di flusso lo richiede.
- **Blocco Ultrasonic**
  - Ha un Hub numerico per configurare la sensibilità del sensore
  - Ha un outputPlug numerico per comunicare la distanza rilevata
- **Blocco Light o Scanner**
  - Ha un Hub numerico per controllare il raggio di rilevazione
  - Ha un Hub numerico per rilevare la luminosità/colore
- **Blocco Touch**
  - Ha due Hub numerici: uno per selezionare la sensibilità e l'altro per rilevare la pressione.

# Idee per il robot e il suo ambiente

---

Ogni gruppo *deve* scegliere un ambiente e un robot; le tre (famiglie di) proposte illustrate di seguito sono fornite a titolo di idee guida

- **Nxt-Expert**
  - Il robot è un *Sistema Esperto* in un qualche dominio che comunica con l'ambiente tramite Display e Keyboard al fine di mettere a disposizione le proprie conoscenze ad un utente.
  - Riferimento: <http://en.wikipedia.org/wiki/ELIZA>
- **Nxt-Turtle**
  - L'ambiente è un foglio di disegno.
  - Il robot è una tartaruga in stile linguaggio Logo con motori che gli permettono di muoversi e di ruotare su se stessa, con una penna per disegnare e con sensori di luminosità (o colore) per agire in base alla parte di disegno che vede sotto di sé.
  - Riferimento: [http://en.wikipedia.org/wiki/Logo\\_\(programming\\_language\)](http://en.wikipedia.org/wiki/Logo_(programming_language))

# Idee per Il robot e il suo ambiente /2

---

- **Nxt-Digger**
  - L'ambiente è un campo da gioco a scacchiera su cui sono disposti un numero limitato di tipi di elementi.
    - Questo ambiente si presta bene ad ospitare più robot
  - Il robot è un minatore in stile gioco Boulder Dash con motori che gli permettono di muoversi lungo gli assi e di ruotare su se stesso e con sensori di contatto e di distanza per interagire con gli elementi presenti.
  - Riferimenti:
    - <http://en.wikipedia.org/wiki/Boulderdash>
    - <http://www.firststarsoftware.com/boulderdash.htm>

# Vincoli di progettazione

---

- Il linguaggio Nxt-robot deve essere rappresentato nel Sistema Software da un *(Meta)modello* e i programmi dei robot da dei *modelli*.
- L'implementazione del linguaggio deve fornire i seguenti servizi:
  - API creazionali
  - API di manipolazione specifica e/o generica
  - Supporto ai Pattern per la definizione modulare del comportamento
- L'interprete deve essere implementato al di fuori della struttura del linguaggio
  - I controlli di correttezza sintattica del programma devono essere concentrati in un validatore richiamato all'inizio dell'esecuzione dell'interprete.
  - I blocchi con un comportamento selezionabile devono essere scritti in modo tale che l'aggiunta di un nuovo comportamento non richieda cambiamenti nell'interprete.
- Il Sistema Software deve prevedere la possibilità di configurare, almeno in fase di lancio, il/i robot con i rispettivi programmi e di eseguire la simulazione in un ambiente dato o anch'esso configurabile.

# Altri requisiti e vincoli

---

- Il design del Sistema Software deve fare un uso estensivo ma ragionato dei Design Pattern visti a lezione e deve essere documentato con diagrammi UML
  - Il design e l'implementazione devono essere *conformi* (nella struttura e nella terminologia) ai Design Pattern studiati.
- Ogni gruppo deve scegliere un *nomeGruppo* di fantasia o derivarlo concatenando i prefissi di 3 lettere dei cognomi dei membri del gruppo.
- I sorgenti del progetto devono essere organizzati in packages che abbiano un prefisso comune del tipo: *labss.nomegruppo*
- Il corretto funzionamento del linguaggio deve essere comprovato da apposite unit test (scritte con JUnit) almeno per i blocchi più complessi
- Il (Meta)modello del linguaggio deve essere documentato con uno o più diagrammi delle classi
- I robot programmati dai membri del gruppo devono avere:
  - il sorgente del programma scritto in un package *labss.nomegruppo.robots*
  - il programma documentato con un diagramma degli oggetti o con un disegno che ricordi la notazione usata da Nxt-G.
  - una procedura di esecuzione semplice e documentata.

# Consegna

---

- Esportare il progetto (Eclipse) in un archivio ZIP e spedire come allegato in una e-mail indirizzata a: [solmi@cs.unibo.it](mailto:solmi@cs.unibo.it).
  - Come soggetto della e-mail usare qualcosa del tipo:  
Consegna progetto LABSS *nomeGruppo*
  - Come nome dell'archivio allegato usare il nome del gruppo.
  - La e-mail deve essere spedita da un membro del gruppo.
- Le date di consegna sono pubblicate sul sito del corso.