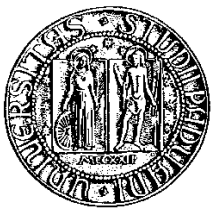

Università degli Studi di Padova

Facoltà di Ingegneria

Corso di Laurea Triennale in Ingegneria Informatica

tesi di laurea

Analisi del linguaggio Logo BYOB e delle potenzialità aggiuntive rispetto a Scratch

Relatore: Prof. Michele Moro

Laureando: Francesca Stival

22 luglio 2012

Autore: Francesca Stival

Indice

Capitolo 1

Introduzione

Questa tesi ha come obiettivo lo studio e il confronto di due linguaggi di programmazione di tipo LOGO che stanno prendendo sempre più piede tra chi inizia a fronteggiare l'informatica e i linguaggi di programmazione: **Scratch** e **BYOB**.
Logo è un linguaggio di programmazione fortemente orientato alla grafica e alla geometria di base. La programmazione avviene guidando un cursore sullo schermo, questo cursore può essere spostato con i comandi *avanti* e *indietro* seguiti dal numero di passi che deve compiere e può essere ruotato con *destra* e *sinistra* seguiti dall'angolo di rotazione espresso in gradi.
Con *pen up* e *pen down* è possibile ordinare alla tartaruga (l'oggetto che stiamo guidando sullo schermo) di tracciare una linea lungo il proprio cammino o di non farlo.
Logo include anche molti comandi per la gestione di input/output testuale e per l'elaborazione di dati (operatori di confronto, variabili, cicli, selezioni condizionali).

Il linguaggio LOGO fu ideato e realizzato negli anni '60 dal professor Seymour Papert del MIT e successivamente ne vennero sviluppate diverse versioni personalizzate a seconda degli obiettivi prefissati.
In origine il LOGO fu utilizzato per muovere un semplice robot, con lo sviluppo dei monitor il linguaggio LOGO divenne più accessibile e negli anni ottanta ne vennero realizzate versioni per personal computer, utilizzate a scopi didattici.
Il LOGO è stato utilizzato con vantaggio nelle scuole elementari e medie inferiori perché permette anche a un principiante di ottenere subito risultati visibili. Dal punto di vista didattico, il LOGO insegnava un metodo di programmazione più strutturato rispetto al più famoso BASIC in cui anche i programmi più banali costringono ad un uso massiccio del costrutto GOTO.
Il LOGO incoraggia la programmazione modulare con uso intensivo di procedure e offre molta estensibilità per gli utenti più esperti.

Scratch è appunto un linguaggio di tipo LOGO con funzioni principalmente didattiche e destinate a bambini e ragazzi dagli 8 ai 16 anni. Il tipo di programmazione è estremamente intuitivo e si è volontariamente omesso lo sviluppo di costrutti troppo complessi. Pur essendo destinato a un pubblico giovane anche molti esperti informatici si sono avvicinati al mondo di Scratch, il suo essere Open Source permette a chiunque fosse interessato di modificare il codice ed adattarlo alle proprie esigenze.
Jens Mönig e Brian Harvey svilupparono un'estensione di Scratch molto interessante che approfondisce una serie di strumenti volontariamente trascurati da Scratch e che rendono BYOB un vero linguaggio di programmazione orientato gli oggetti, destinato non solo ai bambini, ma anche a programmatori e utenti più esperti.

Capitolo 2

Scratch

Il progetto di Scratch nacque nel gennaio 2003. Fu sviluppato dal Lifelong Kindergarten Group al Massachusetts Institute of Technology (MIT). La prima versione rilasciata al pubblico fu la versione 1.0.2. Nell'ottobre 2007 venne rilasciata la versione Beta di Scratch 1.2 in cui venivano fissati dei bug e aggiunti nuovi blocchi, la versione definitiva uscì nel dicembre 2007.
Dal 19 luglio 2009 la versione disponibile di Scratch è la 1.4. E' anche presente una comunità online di Scratch che è molto attiva e negli anni si è ingrandita fino a contenere 1.600.000 membri, 343.000 creatori dei progetti e 2.500.000 progetti caricati (dati del giugno 2012). Chiunque voglia si può iscrivere e caricare e condividere i propri progetti con gli altri utenti.

Figura 2.1: Evoluzione dell'attività nel sito di Scratch nel tempo

Una nuova versione di Scratch, la 2.0 sarà rilasciata a fine estate 2012. A differenza della precedente, questa versione è programmata in Adobe Flash e i progetti saranno sviluppati principalmente on line, ma sarà possibile anche scaricare una versione per programmare off line. Il fatto di essere sviluppato in flash fa sì che Scratch 2.0 non sarà supportato da sistemi iOS. Anche se sviluppato in modo differente sarà possibile implementare progetti sviluppati in

versioni precedenti di Scratch. Altre modifiche importanti sono l'introduzione delle procedure, simili a quelle sviluppate in BYOB, la possibilità di clonare uno sprite mentre il progetto è in esecuzione, Cloud Data che permette il salvataggio di variabili e liste su server e moltre altre.

2.1 Introduzione

Scratch è un linguaggio rivolto ai bambini o in generale a chi si sta avvicinando al mondo della programmazione. Il fatto che permetta di creare semplici programmi senza bisogno di imparare un linguaggio di programmazione e avendo subito un output grafico, lo rende un ottimo metodo per avvicinarsi al problem solving, alla logica e ai primi concetti di programmazione.
I progetti di Scratch sono composti da oggetti chiamati *sprite*, altamente personalizzabili anche grazie a un editor di immagini fornito da Scratch. L'utente può dare istruzioni allo sprite dicendogli di muoversi, suonare, reagire agli altri sprite. Si comanda lo sprite collegando tra loro dei blocchi grafici formando così degli elenchi chiamati *script*.
Quando si clicca su uno script, Scratch esegue i blocchi che lo compongono dal primo all'ultimo.

2.2 Squeak

Scratch è scritto in Squeak, un'implementazione open-source del linguaggio Smalltalk-80, linguaggio che ha pesantemente influenzato alcuni linguaggi come Objective C, C#, C++, Actor, Java e Ruby.
Il codice sorgente di Scratch è disponibile e può essere modificato da chiunque fosse interessato.

2.3 Ambiente di sviluppo

Grazie al modo in cui è strutturato, la maggior parte degli utenti impara a utilizzare Scratch da autodidatta o analizzando progetti già sviluppati e disponibili in rete. Per incoraggiare questo auto-apprendimento l'ambiente di sviluppo di Scratch è stato creato per essere di facile comprensione e per fornire un output immediato.
L'interfaccia utente è in un'unica finestra così da fornire una facile navigazione e da permettere una visione globale del progetto.

La finestra di Scratch è composta da quattro differenti zone.
Nella zona 1 sono presenti tutti i blocchi disponibili per la programmazione, divisi per colore a seconda del tipo di comando.

- I blocchi di colore blu riguardano comandi relativi al movimento dello sprite;
- I blocchi di colore giallo si occupano del controllo dello sprite e contengono i blocchi che gestiscono i cicli e le decisioni (if, else);
- I blocchi di colore viola gestiscono l'aspetto dello sprite, modificandolo, cambiando le sue dimensioni;
- I blocchi di colore azzurro hanno la funzione di sensori, ad esempio avvertono quando un tasto del mouse o della tastiera viene premuto;
- I blocchi di colore rosa riproducono musica ed effetti sonori;
- I blocchi di colore verde chiaro contengono gli operatori di tipo booleano o che consentono operazioni aritmetiche e confronti (maggiore, minore o uguale);
- I blocchi di colore verde scuro sono relativi all'uso della penna. Con l'istruzione penna giù è come se lo sprite si trascinasse dietro una penna appoggiata a un foglio che quindi mostra i movimenti che fa;
- Infine i blocchi di colore arancione consentono la creazione di nuove variabili.

La parte 2 è chiamata *Area degli Script* in cui vengono trascinati i blocchi di comando che verranno eseguiti uno dopo l'altro dallo sprite. Per eseguire un blocco è sufficiente cliccarci sopra. Quando si trascina un blocco all'interno dell'Area degli Script, la comparsa di una linea bianca indica i punti in cui può essere aggiunto formando un collegamento valido con un altro blocco.
Alcuni blocchi hanno delle caselle bianche il cui testo contenuto all'interno può essere modificato.

Il pannello 3 in alto a destra è lo *Stage*. Questa è l'area in cui gli sprite si muovono e interagiscono tra loro.

Nel pannello 4 possono essere creati nuovi sprite, scelti da una vastissima collezione.
Dal menù in alto è possibile salvare il progetto o aprirne di già esistenti. E' particolarmente interessante l'opzione avvia passo-passo qui contenuta. Questa opzione permette di veder eseguire gli script di Scratch un passo alla volta. Ogni blocco viene evidenziato nel momento in cui viene eseguito. Questa possibilità può risultare utile per evidenziare errori negli script e per aiutare i nuovi programmatori a capirne il flusso.
La bandiera verde nell'angolo in alto a destra dello stage fa partire tutti gli script che iniziano con la bandiera verde nello stesso istante.

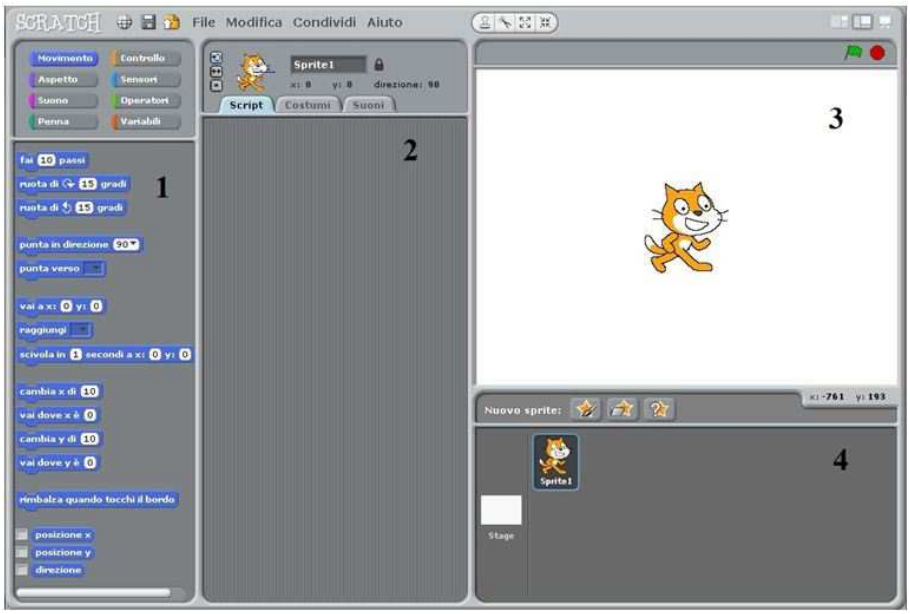

Figura 2.2: Ambiente di sviluppo di Scratch

Scratch non ha messaggi di errore. Gli errori di sintassi non possono verificarsi perché i blocchi si combinano tra loro solo in modi che hanno un senso. Inoltre Scratch ha fatto si che i messaggi di errore non fossero necessari rendendo i blocchi fail soft, ossia si arrestano eventuali componenti non essenziali in caso di guasto, mantenendo il progetto in esecuzione sul computer. In caso di errore, ogni blocco cerca di fare qualcosa di sensato.
Ovviamente l'assenza di messaggi d'errore non elimina gli errori. Gli utenti devono prestare attenzione mentre costruiscono le azioni da far svolgere allo script e se questo fa qualcosa di diverso da quanto ci si aspetta occorre analizzare i blocchi utilizzati, magari analizzandoli passo passo con l'apposita funzione. Nel caso si commettesse un errore i blocchi non vengono eseguiti e vengono contornati da un bordo rosso.

Figura 2.3: Esempio di errore in Scratch: divisione per zero

2.4 Linguaggio di programmazione

2.4.1 Sintassi

Come è già stato detto, gli script di Scratch sono formati attaccando tra loro dei blocchi che rappresentano dichiarazioni, espressioni e strutture di controllo. La forma dei blocchi suggerisce come questi vadano collegati e il sistema rifiuta di attaccare tra loro blocchi la cui unione non ha significato. Quindi in Scratch la grammatica visiva della forma dei blocchi e le regole per la loro unione ha il ruolo della sintassi in un linguaggio di tipo testuale. Ci sono quattro tipi di blocchi in Scratch:

- Command blocks;
- Function blocks;
- Trigger blocks;
- Control structure blocks.

I *control structure blocks* sono delle specie di blocchi di comando con una ulteriore sequenza di comandi nidificata. La loro struttura li rende facili da usare e il fatto di essere una struttura indivisibile fa sì che i suoi estremi siano facilmente indentificabili, al contrario di quanto avviene nei linguaggi di tipo testuale.
I *command blocks* sono come le dichiarazioni dei linguaggi di programmazione testuali, mentre i function blocks come gli operatori. A differenza dei linguaggi testuali, in Scratch i *function blocks* non sono aggiunti in una sequenza lineare, ma vengono utilizzati come argomenti di comandi e nidificati per costruire le espressioni.
Infine i *trigger blocks* collegano gli eventi di trigger (come ad esempio l'avvio, click del mouse, tasto premuto) alle pile che gestiscono gli eventi.

2.4.2 Tipi di dati

I dati in Scratch sono di prima classe, cioè possono essere passati come parametri ad altre funzioni e possono essere restituiti come valori di ritorno da altre funzioni.
Scratch ha tre tipi di dati di prima classe: booleani, numero e stringhe. Questi tipi di dati sono gli unici che possono essere utilizzati nelle espressioni, memorizzati in variabili, o restituiti da funzioni incorporate.

L'editor di Scratch permette a un blocco funzionale di essere inserito in uno slot di parametro se il risultato non è in contrasto con i vincoli del tipo di dato che vi può essere inserito. Gli slot dei parametri booleani sono i più rigorosi,

dato che accettano solo blocchi funzionali di tipo boolean. Gli slot di parametri numerici e stringhe sono meno rigorosi, essi possono accettare un blocco funzionale di qualsiasi tipo, costringendo il parametro al tipo di destinazione qualora fosse necessario.
Una variabile di Scratch può contenere valori di qualunque tipo di dati. Ciò evita la richiesta all'utente di specificare il tipo di una variabile quando questa viene creata. Scratch converte automaticamente numeri e stringhe a seconda del contesto. Per esempio, se la stringa 123 viene passata a una operazione aritmetica, viene convertita in numero, mentre se un numero è passato al comando dire, viene convertito in una stringa. Tenuto conto degli obiettivi di Scratch, la conversione automatica è preferibile rispetto al richiedere all'utente di specificare in modo esplicito il tipo di dati usato.

2.4.3 Modello a oggetti

Gli Sprite sono oggetti, infatti essi incorporano uno stato (rappresentato dalle variabili) e un comportamento (script). Tuttavia, poiché Scratch non ha né le classi né sviluppa l'ereditarietà, è un linguaggio basato sugli oggetti, ma non un linguaggio orientato agli oggetti. Questi tipi di linguaggi sono definiti come *instance-oriented*. Infatti un linguaggio per essere definito come orientato agli oggetti deve supportare l'ereditarietà [Wegner 1987].
E' facile comprendere il perché di questa scelta, in quanto l'ereditarietà renderebbe gli script troppo complessi, anche se l'assenza di classi o meccanismi di code-sharing può rendere più gravosa la gestione di più sprite con comportamento identico.
Come accennato, gli sprite non possono chiamare direttamente gli script gli uni degli altri. Per supportare la comunicazione tra sprite e la sincronizzazione, Scratch utilizza un meccanismo di trasmissione di tipo broadcast mediante la trasmissione di una stringa arbitraria. Ad esempio, in risposta al messaggio trasmesso Scena due, sprite diversi che rappresentano i personaggi di una storia potrebbero cominciare ad agire in quella scena. Tale tipo di comunicazione è asincrona.

Il fatto che gli sprite non si possono controllare tra loro porta due vantaggi. In primo luogo, quando si cerca di capire perché uno sprite fa qualcosa (ad esempio, si muove in un certo modo), l'ambito delle possibilità è limitato agli script di quello sprite. In secondo luogo, e cosa più importante, perché questo rende gli sprite autosufficienti, così possono essere spostati in progetti differenti senza comportare una rottura delle dipendenze. Questo permette e incoraggia la condivisione degli sprite, il riutilizzo del codice e la collaborazione.

2.4.4 Procedure

Le prime versioni di Scratch avevano un meccanismo per la creazione di procedure. Tuttavia questo concetto confuse molti utenti che le trovavano troppo complesse, tanto che si decise di rimuoverle nonostante l'astrazione procedurale sia una delle idee più potenti dell'informatica.

2.4.5 Concorrenza

La concorrenza (o multi-threading) è una tecnica di programmazione avanzata, tuttavia il nostro mondo è altamente concorrente e gli utenti di Scratch non sono sorpresi dal fatto che uno sprite possa fare diverse cose contemporaneamente, anche se questo concetto non è affatto scontato per un utente più esperto. In Scratch mancano gli espliciti meccanismi di controllo della concorrenza che si possano trovare in altri linguaggi di programmazione, come ad esempio i semafori o i monitor.
Scratch basa il controllo della concorrenza nel suo modello di threading in modo da evitare la maggior parte delle condizioni di concorrenza, in questo modo gli utenti non devono occuparsi di queste questioni.

2.4.6 Ricorsione

Uno dei mezzi più potenti nell'informatica è la ricorsione, ovvero la chiamata di un'espressione a se stessa.

Nei vari linguaggi di programmazione la ricorsione è generalmente attuata con uno stack di espressioni da valutare. Scratch, tuttavia, non implementa la ricorsione con il meccanismo di stack, ma ne fornisce una forma limitata, in cui la chiamata ricorsiva è l'ultima cosa fatta nello script (tail recursion).
Non è possibile inserire la chiamata ricorsiva in un luogo diverso dalla fine dello script perché quando Scratch riceve un messaggio che innesca uno script già in esecuzione, dimentica dove si trovava (invece che memorizzarlo nello stack) e ricomincia dall'inizio. Per far si che uno script continui un processo dopo aver fatto una chiamata ricorsiva, è necessario implementare un altro script che viene lanciato dallo script originale.

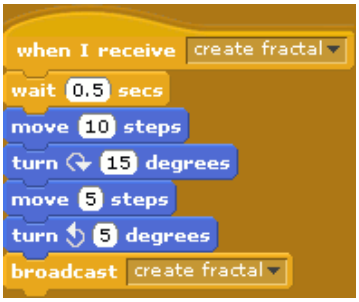

Figura 2.4: Esempio di ricorsione funzionante in Scratch

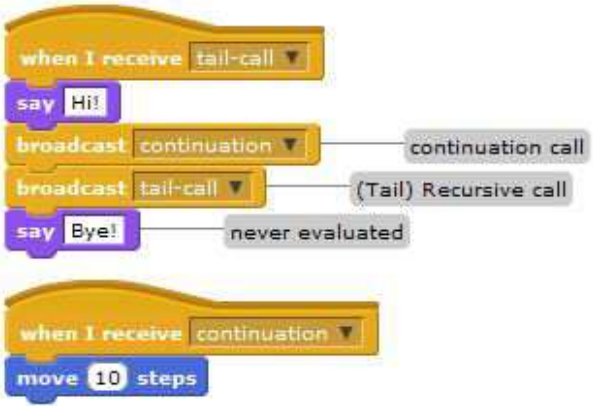

Figura 2.5: Esempio di ricorsione non funzionante in Scratch

2.5 Limiti di Scratch

Come si è visto nei precedenti paragrafi, Scratch ha una serie di notevoli limitazioni che fanno sì che non possa essere utilizzato da utenti più esperti come linguaggio di programmazione. Le limitazioni di Scratch non vanno viste come una sua debolezza, gli sviluppatori sono ben consci del fatto che molte potenzialità potrebbero essere implementate in questo linguaggio, ma si è deliberatamente scelto di non approfondirle proprio per la filosofia di Scratch, ovvero il suo essere un linguaggio rivolto ai bambini, per cui è più importante che l'ambiente sia piacevole e permetta la creazione di disegni o suoni, piuttosto che permetta l'implementazione della ricorsione, procedure e quant'altro. Le mancanze di Scratch sono colmate da BYOB (Build Your Own Block) che è una versione editata di Scratch che, oltre a contenere quanto già visto fin'ora, tratta argomenti più avanzati rendendolo un linguaggio ottimale anche per utenti più esperti.

Capitolo 3

BYOB

Quanto detto finora per Scratch vale anche per BYOB, che quindi può essere visto come un superset del primo. L'ambiente di programmazione, la sintassi, il modo in cui i progetti sono costruiti è lo stesso. Tuttavia BYOB ha delle funzionalità in più rispetto a Scratch che verranno analizzate in questo capitolo. La presenza di queste funzionalità aggiuntive fa sì che i file eseguibili di BYOB abbiano un'estensione differente rispetto a quelli di Scratch (.ypr BYOB e .sb Scratch). E' impossibile aprire in Scratch un progetto realizzato con BYOB, se invece si prova ad aprire un file di Scratch in BYOB questo viene aperto, ma non sempre funziona.

3.1 Introduzione

3.1.1 Storia

Build Your Own Blocks (costruisci i tuoi blocchi, più spesso indicato con il suo acronimo, BYOB), è una versione editata di Scratch, sviluppata principalmente da due suoi utenti, Jens Mönig e Brian Harvey.
Esso include molte caratteristiche che non sono disponibili in Scratch e queste aggiunte sono state apportate senza grosse modifiche al codice sorgente. Come dice lo stesso nome del programma, BYOB consente all'utente di creare propri blocchi personalizzabili secondo le esigenze. L'obiettivo principale di questa modifica di Scratch è l'aggiunta di tipi di dati di prima classe oltre a quelli forniti da Scratch, visti nel capitolo precedente.
Attualmente, l'ultima versione rilasciata di BYOB è la 3.1.1, rilasciata il 19 maggio 2011, ma gli sviluppatori stanno lavorando alla prossima versione, la 4.0 in cui il programma cambierà nome, non sarà più BYOB, ma Snap, a causa dei molti altri significati del vecchio nome.
Attualmente, nel sito di BYOB è possibile vedere e provare la nuova versione (attualmente in versione alfa di prova) che viene eseguita nel browser. La grafica del programma risulta notevolmente differente, è molto più seria, tanto che il

nuovo sprite è una freccia e non più Gobo, il simbolo di BYOB.

Appare evidente che BYOB, o meglio Snap, non è un linguaggio destinato ai bambini o ai neofiti, ma un linguaggio di programmazione vero e proprio.
Snap 4.0 sarà scritto in JavaScript, utilizzando i canvas element di HTML5.

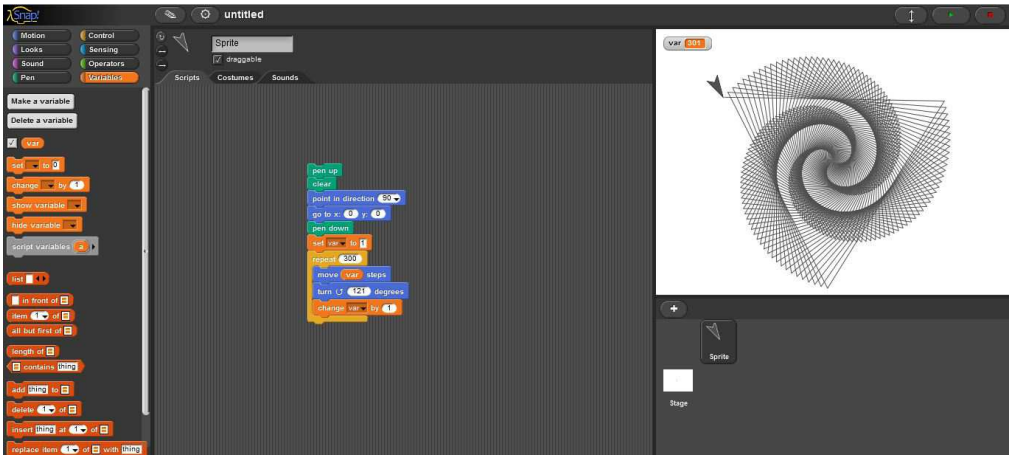

Figura 3.1: Ambiente di lavoro di Snap, nuova versione di BYOB

Userà anche una libreria Morphic creata da Jens Mönig chiamata Morphic.js.

3.1.2 Struttura

Come è già stato accennato l'ambiente di sviluppo di BYOB è praticamente identico a quello di Scratch. A parte qualche blocco aggiuntivo:

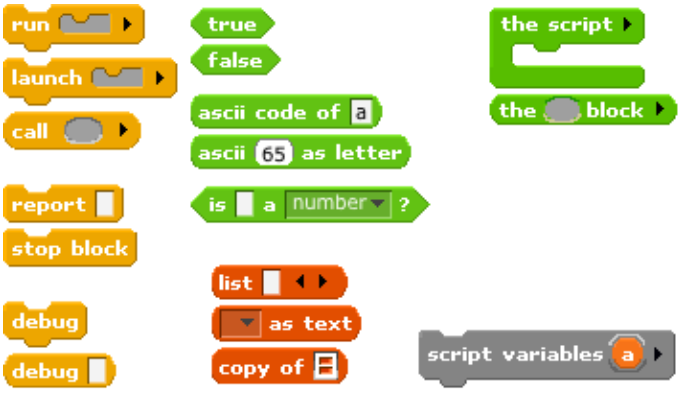

Figura 3.2: Nuovi blocchi introdotti da BYOB

La grande differenza tra i due programmi sta nell'istruzione Make a block contenuta nella categoria Variables.

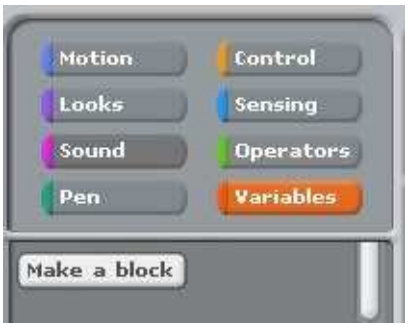

Figura 3.3: Make a block

Make a block consente all'utente di costruire i propri blocchi. Cliccando sul pulsante viene aperta una finestra di dialogo che consente di scegliere che tipo di blocco realizzare e gli si assegna un nome. Per ogni categoria di blocchi (Motion, Control, ecc) si può scegliere se realizzare un blocco di tipo command (comando, non riporta valori), reporter o predicate (entrambi output, ma i predicate restituiscono valori di tipo Booleano).
E' possibile scegliere una qualunque categoria per realizzare il blocco, le funzioni che questo eseguirà sono le stesse. La differenza del colore fa si che BYOB sappia in che categoria inserire il nuovo blocco.

Figura 3.4: Interfaccia per la realizzazione di un nuovo blocco

I blocchi di tipo command non ritornano alcun valore, come i metodi void in Java. I blocchi di tipo reporter restituiscono un valore, sono analoghi ai return method in Java. I blocchi predicate hanno la stessa funzione dei blocchi reporter, con l'unica differenza che restituiscono un valore di tipo booleano.
Nei prossimi esempi verrà mostrato dettagliatamente l'uso di Make a block.

3.2 Ricorsione

Come visto nel capitolo precedente, Scratch ha una grande lacuna che BYOB colma, ossia la possibilità di usare la ricorsione. Per realizzare la ricorsione in BYOB è sufficiente realizzare dei blocchi che chiamano se stessi.
Vediamo nel dettaglio la ricorsione e la creazione dei blocchi con degli esempi.

ESEMPIO 1

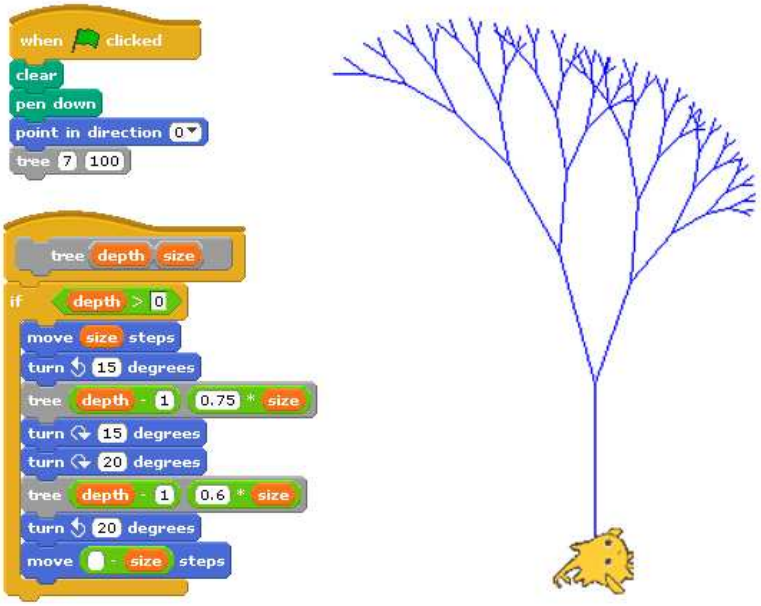

Figura 3.5: Esempio di ricorsione con BYOB

In questo esempio per creare un albero si crea il tronco, si ruota lo sprite di 15 gradi verso sinistra e poi si richiama la stessa funzione che continuerà a creare i rami verso sinistra finchè depth è >0. Una volta raggiunto tale valore si continua con le istruzioni presenti dopo il blocco ricorsivo, lo sprite si riallinea con il ramo, poi ruota di 20 gradi a destra e da qui comincia a costruire i rami di destra. Il concetto base della ricorsione è quello di costruire l'elemento base (che

poi verrà ripetuto) e non preoccuparsi di quello che accadrà dopo. Per far si che la ricorsione non continui all'infinito occorre porre una condizione e la ricorsione continuerà finchè tale condizione risulta verificata. In questo caso è stata introdotta la variabile depth che viene decrementata ogni volta che si costruisce un nuovo ramo. Quando depth giunge a zero allora la ricorsione viene interrotta.

ESEMPIO 2

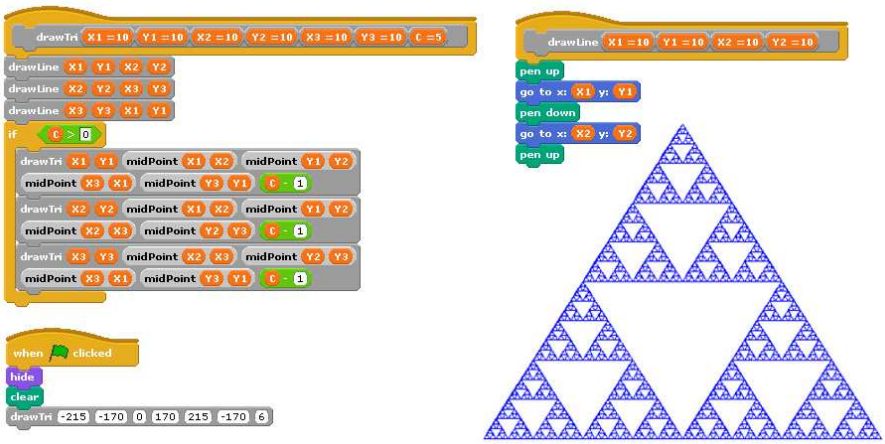

Figura 3.6: Esempio di ricorsione con BYOB

In questo esempio vengono costruiti i triangoli di Sierpinski. Disegnato un triangolo (che supporremo equilatero) detto di livello 0, si disegna il triangolo i cui vertici sono i punti di mezzo dei lati del triangolo principale, poi si applica ricorsivamente lo stesso procedimento ai tre sotto-triangoli nord, sud-est e sud-ovest. La struttura è un frattale in quanto è autosomigliante. Una semplice osservazione ci dice che man mano che si scende di livello le lunghezze dei lati dei triangoli si dimezzano.

ESEMPIO 3

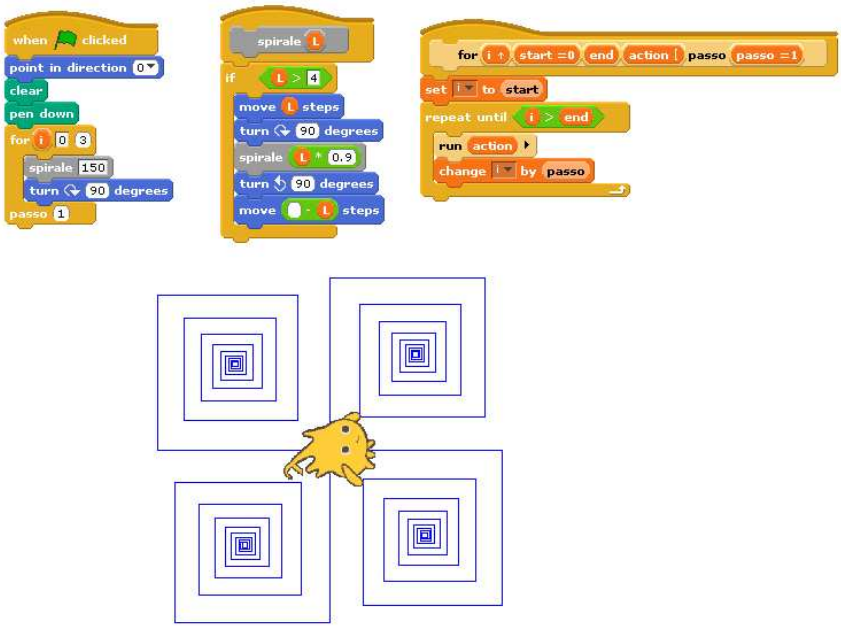

Figura 3.7: Esempio di ricorsione con BYOB

Quest'ultimo esempio è più semplice, ma nella realizzazione di queste quattro spirali è stato introdotto un blocco che permette l'esecuzione di un ciclo for il cui funzionamento verrà spiegato in dettaglio più avanti.

3.3 Tipi di dati

In un linguaggio di programmazione, un tipo di dati è di prima classe se quel tipo di dati può essere:

- Il valore di una variabile;
- Un ingresso ad un procedimento;
- Il valore restituito da una procedura;
- Un membro di un aggregato di dati;
- Anonimo.

Come abbiamo visto, in Scratch solo numeri, booleani e stringhe sono dati di prima classe, invece le liste di Scratch non sono di prima classe. Una volta creata la lista mediante il bottone *Crea una lista* le si assegna un nome, ma non è possibile inserire la lista in una variabile, nello slot di ingresso di un blocco o in un elenco di elementi e soprattutto non si possono avere liste di liste.

Un principio fondamentale alla base di BYOB è che tutti i dati devono essere di prima classe. Se quel particolare costrutto è disponibile nel linguaggio, allora dovremmo essere in grado di utilizzarlo pienamente e liberamente.
Si noti che è il tipo di dati che ad essere di prima classe, non una sua singola istanza.

3.3.1 Liste

Si supponga di avere un insieme S di n elementi memorizzati in ordine lineare, così da poter indicare gli elementi di S come primo, secondo, terzo e così via. Tale insieme è indicato con il nome di lista e ogni suo elemento può essere univocamente determinato mediante un numero intero uguale al numero di elementi di S che lo precedono.
In Scratch per creare una lista occorre cliccare su Make a list:

Figura 3.8: Modello di lista presente anche in Scratch

Come si vede nell'esempio occorre inserire gli elementi uno per volta utilizzando un blocco apposito per l'inserimento.
Le liste introdotte da BYOB invece, permettono di creare delle liste anonime, cioè senza assegnare loro un nome. Questo nuovo tipo di lista può essere usato come input in altri blocchi, infatti a differenza delle liste di Scratch che sono costituite da blocchi di tipo command, quelle di BYOB sono formate da blocchi di tipo reporter.

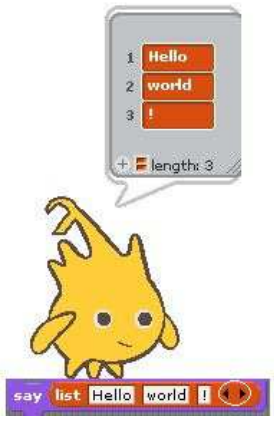

Figura 3.9: Esempio di lista

Cliccando le frecce cerchiate in figura si possono aggiungere elementi vuoti o rimuoverne dalla lista. Una volta creata la lista, questa può essere utilizzata, venire stampata, il contenuto può venire unito in una stringa e altre operazioni.

Figura 3.10: Esempi con la nuova lista anonima di BYOB

Purtroppo però non è possibile inserire elementi nella lista se non manualmente, questo può risultare pratico da un lato, ma costituisce una limitazione nel caso si volesse fare un inserimento in seguito alla verifica o meno di una condizione.

Esiste anche un tipo di dato lista:

Figura 3.11: Nuovo tipo di dato Lista introdotto da BYOB

I rettangoli rossi all'interno dello slot di ingresso assomigliano a come Scratch visualizza le liste sullo stage con ogni elemento contenuto in un rettangolo rosso.

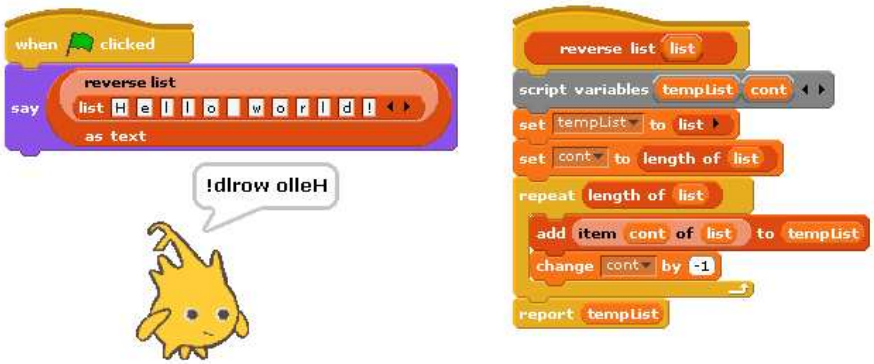

Figura 3.12: Esempio di uso delle nuove liste di BYOB

In questo esempio si inserisce una lista in un blocco di tipo reporter, qui la lista verrà rovesciata utilizzando un'altra struttura lista di appoggio e delle variabili temporanee.

3.3.2 Liste di liste - Alberi

Le liste possono essere a loro volta elementi di altre liste per formare una struttura dati non lineare. Partendo da questo presupposto si possono creare tutte quelle strutture dati informatiche che vengono costruite da liste di liste creando gli appositi costruttori.
Una delle strutture dati non lineare più utilizzata nei processi computazionali sono gli alberi. Un albero è un tipo di dati astratto che memorizza i suoi elementi in modo gerarchico: ogni elemento, escluso l'elemento superiore, possiede un elemento padre e zero o più elementi figlio. Gli alberi più comunemente usati sono di tipo binario, ossia hanno al massimo due figli.
Questo tipo di struttura permette la realizzazione di un insieme di algoritmi che risultano assai più veloci di quelli realizzati su strutture dati lineari come la lista.

Nell'esempio sotto riportato, lo script riconosce se quello che si sta esaminando è o meno un albero binario.

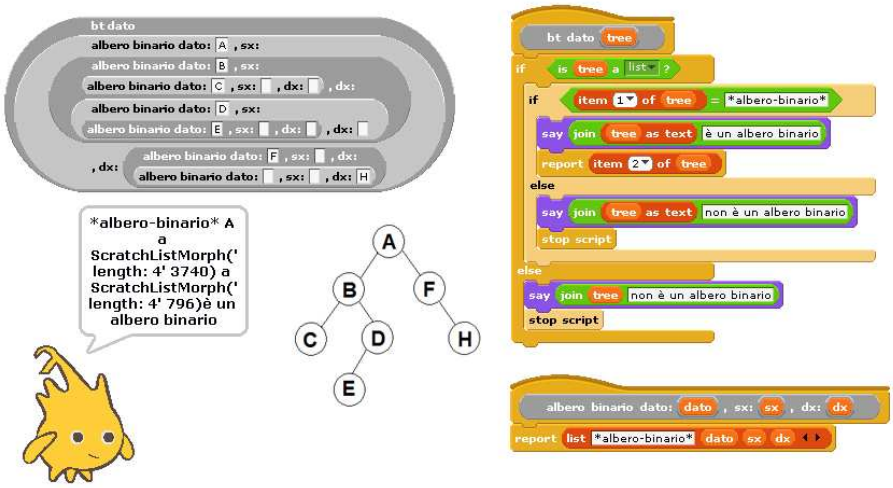

Figura 3.13: Esempio di implementazione di un albero binario con BYOB

In questo primo caso è stato realizzato l'albero binario rappresentato in figura e lo sprite riconosce giustamente che si tratta proprio di un albero binario.

Figura 3.14: Esempio di albero binario con BYOB

Inserendo una semplice lista, lo sprite riconosce che non si tratta di un albero binario e lo segnala.

3.4 Input

Come abbiamo già visto, in Scratch gli input di un blocco possono essere di tipo testuale/numerico o solo numerico. In alcuni blocchi di controllo di forma esagonale può essere utilizzato un terzo tipo di dati, i booleani.
Gli input in BYOB possono essere di ben 12 tipi differenti:

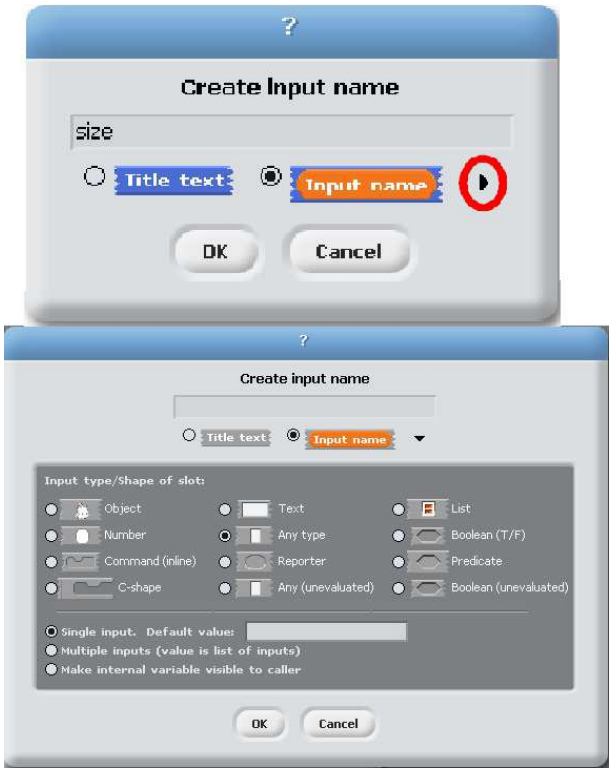

Figura 3.15: Creazione di un nuovo Input

Ogni tipo di dato può inoltre appartenere a una di tre categorie mutualmente esclusive:

- Single Input:
  In Scratch tutti gli input appartengono a questa categoria. Come si vede dal disegno, c'è uno spazio per l'input nel blocco. Se un input è di tipo Any, Number o Text, allora è possibile specificare un valore di default che verrà visualizzato direttamente nel blocco.

Figura 3.16: Esempio di Single Input

- Multiple Input:
  Il blocco lista introdotto precedentemente accetta un numero variabile di ingressi che specificano gli elementi della lista. Per permettere questo, BYOB introduce delle frecce per espandere o comprimere il blocco aggiungendo o rimuovendo slot. Blocchi personalizzati definiti dagli utenti di BYOB hanno le stesse potenzialità. Se si sceglie l'opzione Multiple Input le frecce appariranno nello slot di inserimento del blocco. Quando il blocco viene eseguito tutti i valori negli slot di ingresso sono raggruppati in una lista.

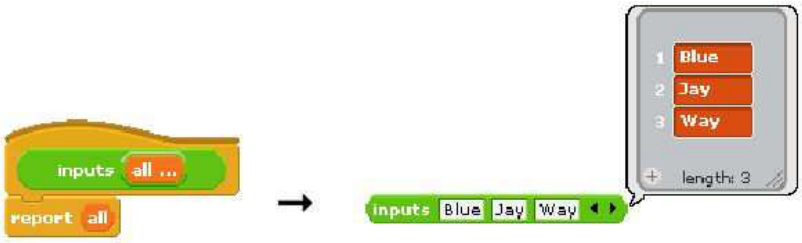

Figura 3.17: Esempio di Multiple Input

- Make Internal Variable Visible:
  Non sono veramente un tipo di ingresso, bensì una sorta di output del blocco per l'utente. Una freccia che punta verso l'alto identifica variabili di questo tipo.

Figura 3.18: Esempio di Make Internal Variable Visible

Di default viene impostato l'input di tipo singolo ed è possibile specificare un valore da assegnare automaticamente alla variabile.
Se non si sceglie il tipo di dato, viene assegnato di default il tipo any che può accettare qualsiasi tipo di valore.
La disposizione dei tipi di dato nella griglia non è casuale:

- Prima riga: nuovi tipi di dato non procedurale introdotti da BYOB;
- Seconda riga: tipi di dati presenti anche in Scratch;

- Terza e quarta riga: tipi di dati delle procedure di BYOB.

La suddivisione in colonne invece separa i dati di tipo command (prima colonna), i dati di tipo report (colonna centrale) e i predicati (ultima colonna).

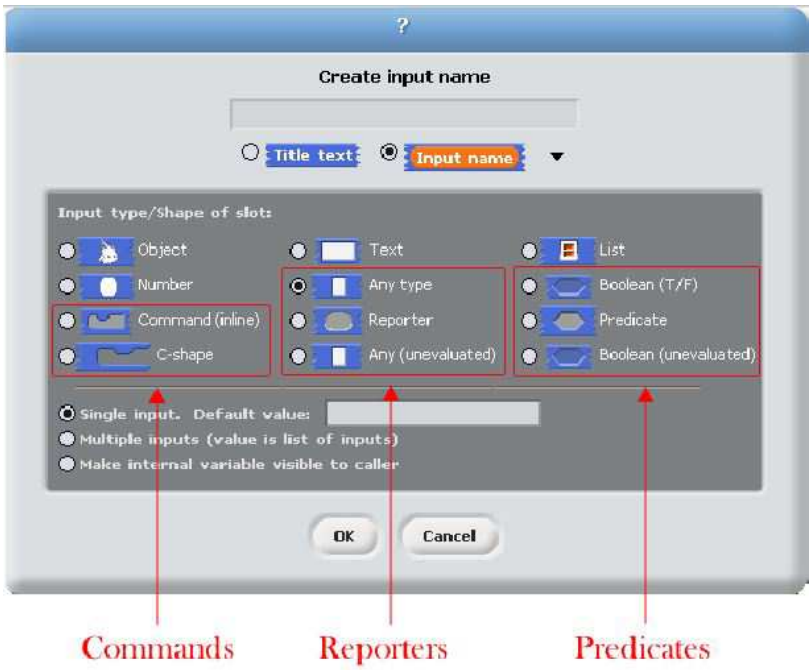

Figura 3.19: Suddivisione per tipi di Input

Il tipo Text è una variante del tipo Any e ha una forma che suggerisce un input di testo.
I blocchi di Scratch hanno tre forme:

- Puzzle per i blocchi di comando;
- Ovale per i report (collegati al tipo ANY);
- Esagonale per i predicati (ovvero report booleani).

In BYOB tutti questi blocchi sono di dati di prima classe e l'ingresso di un blocco può essere uno di questi tre tipi. A differenza degli ultimi due, il tipo command non restituisce alcun valore.

3.5 Procedure come dati

3.5.1 Tipi di input per le procedure

I blocchi **call** e **run** sono il fondamento delle caratteristiche delle procedure di prima classe di BYOB: essi permettono agli script e ai blocchi di essere usati come dati ed eventualmente posti sotto il controllo del programma dell'utente. Il run esegue i blocchi inseriti nei suoi slot. Quando si clicca la freccia verso destra si possono specificare i valori in ingresso nel blocco sotto forma di semplici input o come elementi che compongono una lista.

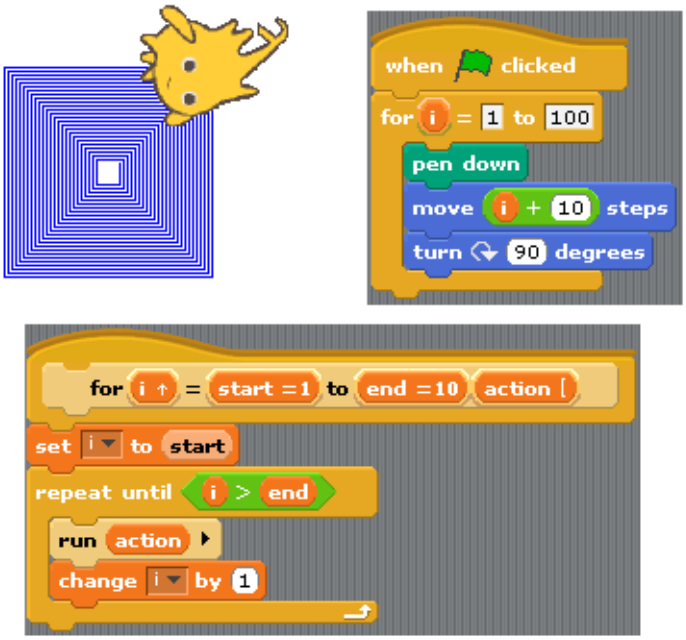

Figura 3.20: Esempio di uso del blocco run

Nell'esempio si mostra come sia possibile realizzare un blocco personalizzato che realizzi un ciclo for che non è presente in BYOB. Nella dichiarazione di tale costrutto si è utilizzata una variabile i di tipo interno, ma visibile anche all'esterno (infatti viene utilizzata nel programma in cui si usa il ciclo for). Viene usata anche la variabile action di tipo Command. In questo caso il blocco run esegue il blocco action di tipo *Command*. Facendo un'analogia con un programma di tipo testuale, la run è simile a funzioni di tipo void in cui la funzione non restituisce alcun valore, infatti stiamo operando con blocchi Command che non restituiscono alcun valore: per fare si che le modifiche si ripercuotano all'esterno si usa

una variabile globale che mantiene il suo valore anche quando viene utilizzata fuori dal blocco in cui è stata definita.

L'operazione di call è simile alla run, solo che la run esegue blocchi di comando, mentre la call esegue blocchi report che quindi consentono di restituire un valore. L'operazione di restituzione avviene mediante il blocco report.

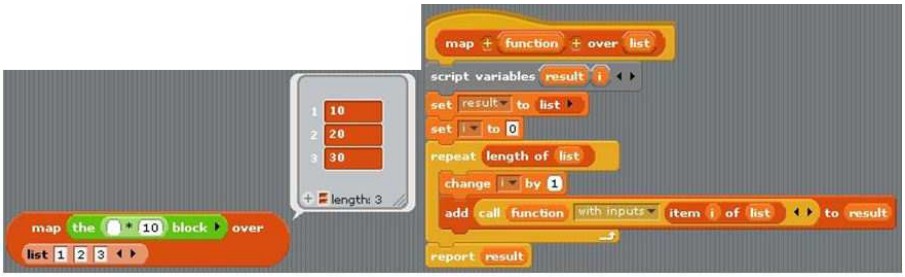

Figura 3.21: Esempio di uso del blocco call

Nell'esempio si vuole moltiplicare per 10 ogni elemento di una lista. Il blocco call esegue la moltiplicazione per 10 (call function) utilizzando come input gli elementi di una lista passata come input. Nell'esecuzione della moltiplicazione viene specificato un valore solo per uno dei sue input (10), in questo modo il blocco call capisce di dover usare l'altro valore di input fornito (la lista) per riempire lo slot vuoto della moltiplicazione per 10.
Come anticipato, il blocco report specifica che variabile deve essere restituita dalla funzione. Nell'esempio i valori moltiplicati per 10 vengono memorizzati in una lista result ed è proprio questa lista a dover essere restituita al blocco chiamante.
Sia il blocco call che il run hanno una freccia a destra alla fine del blocco, cliccandola compare la frase *with inputs* e lo slot per poter inserire gli input. La freccia può essere cliccata un numero arbitrario di volte così da ottenere il numero di input desiderato.
Un'altra cosa interessante da notare nell'ultimo esempio è che quando nel blocco map è stato inserito l'input *10, questo non è stato inserito semplicemente nello slot, ma è stato aggiunto il blocco operator:

Figura 3.22: Blocco the block

D'ora in avanti tale blocco sarà indicato con **the block**.

Questo blocco, non presente in Scratch, fa si che sia il blocco e non il valore che esso assume ad essere riportato come input quando si effettua la chiamata. L'esempio seguente chiarirà meglio il concetto.

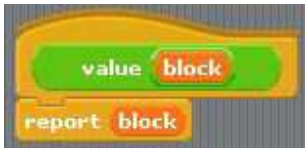

Figura 3.23: Blocco value

Figura 3.24: Esempio di uso del blocco the block

E' stato realizzato un blocco di nome value. Se in tale blocco inseriamo semplicemente il blocco somma 1+2 il risultato restituito è la soluzione dell'espressione indicata nel blocco, ovvero 3. Se però il blocco somma 1+2 viene inserito in the block allora il risultato di value è tutto il blocco 1+2 e non il suo risultato.
Allo stesso modo, nell'esempio precedente inserendo semplicemente *10 nello slot, il blocco map restituisce un messaggio di errore:

Figura 3.25: Esempio errato di blocco the block

Quello che vogliamo fare è passare a function il blocco *10, sarà poi processato da questa operazione:

Figura 3.26: Esempio di blocco call

che eseguirà il prodotto *10 utilizzando come input i valori della lista. A questo punto, nell'istruzione add si vuole veramente calcolare il valore di function, infatti questa volta il blocco non va inserito in the block.
Altri blocchi non presenti in Scratch, ma introdotti da BYOB, con funzioni simili a quelli appena visti sono:

- **Launch**: simile al run, il launch esegue lo script contemporaneamente al successivo. La stessa cosa accade con la pressione della freccia a destra;
- **Stop block**: istruzione simile al break dei linguaggi testuali, ferma l'esecuzione di un blocco senza riportare nulla.

Nell'esempio visto è stato usato il blocco **script variables**. Esso è di colore grigio nonostante sia un blocco di variabili. Crea delle variabili che possono essere usate solo nello script in cui sono definite, in pratica delle variabili locali.

3.5.2 Scrivere procedure higher order

Una procedura high order è una procedura che ne riceve un'altra in input o ne ritorna una come output. Anche se viene accettato in ingresso un blocco di tipo Any che accetta l'input di una procedura, per ritornare in uscita tale blocco e non il valore da esso assunto occorre comunque che esso venga inserito nel blocco the block.
Per dichiarare come ingresso il tipo Procedura è necessario espandere il blocco di input.
Per utilizzare la procedura come input in un blocco vengono utilizzati il blocco run per i dati di tipo command e il blocco call per quelli di tipo report. Il primo è un blocco su una singola riga e l'input è solitamente una variabile il cui valore è uno script.
Quando i blocchi run e call hanno le frecce alla fine, è possibile aggiungere o togliere slot di input come già visto in precedenza. Nel caso in cui alcuni blocchi rimangano vuoti, BYOB automaticamente sa come gestire la situazione. In questi casi vi sono più possibilità:

- se il numero di blocchi vuoti è uguale al numero di input aggiunti, BYOB riempie i blocchi da sinistra verso destra;

Figura 3.27: Esempio di gestione dei blocchi di input

- se vi è esattamente un solo input, BYOB riempie i blocchi vuoti con esso;

Figura 3.28: Esempio di gestione dei blocchi di input

- in tutti gli altri casi BYOB non fa nulla dato che non sono chiare le intenzioni dell'utente.

Se l'utente desidera sovrascrivere queste regole, la soluzione è utilizzare blocchi o script con nomi espliciti così da poter inserirli nei blocchi o negli script dati per indicare come vengono utilizzati gli ingressi.

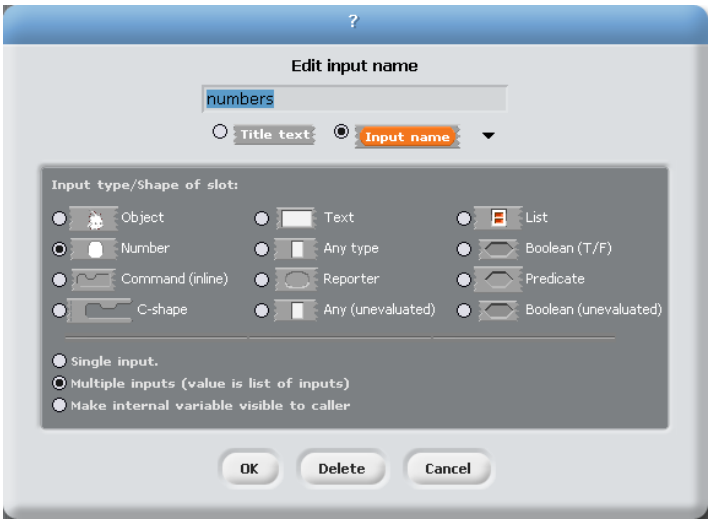

Figura 3.29: Esempio di input di tipo Multiple input

L'input è di tipo Number, ma è stata scelta l'opzione Multiple inputs, per cui come input si avrà una lista di number.

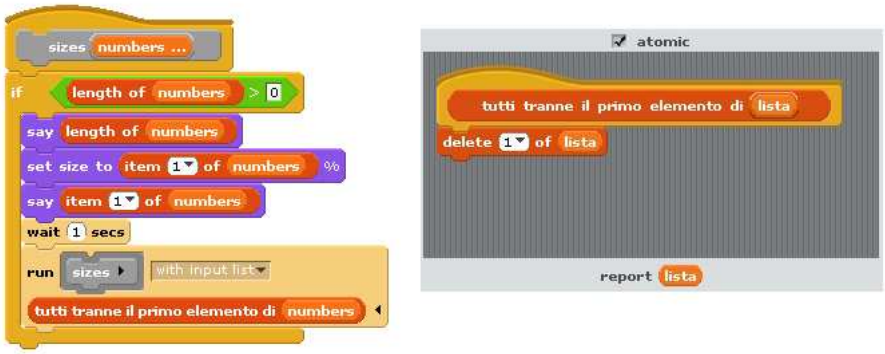

Figura 3.30: Esempio di input di tipo Multiple input

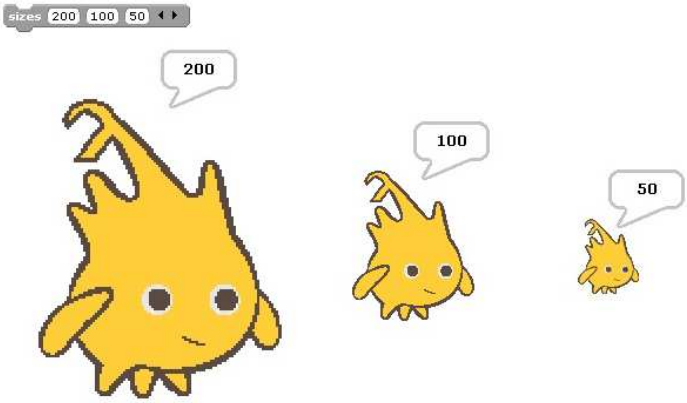

Figura 3.31: Esempio di input di tipo Multiple input

L'utente di questo blocco chiama lo script con una quantità qualsiasi di numeri come input. Dentro la definizione del blocco, tutti questi numeri formano una lista, che ha un nome unico: *sizes*. Sizes non prende in input una lista, ma dei valori numerici. L'opzione *with input list* sostituisce i valori dello slot di tipo *any-type* con uno slot del tipo *list-type*.

Figura 3.32: Esempio di run con ingresso input list

Gli elementi della lista sono presi singolarmente come input per lo script. Dal

momento che numeri è una lista di numeri, ogni elemento è a sua volta un numero.

3.5.3 Procedure come dati

In BYOB è possibile trovare un forma abbreviata per il blocco the block e the script appartenenti alla categoria Operators. Tale abbreviazione si traduce come un blocco dal contorno grigio.

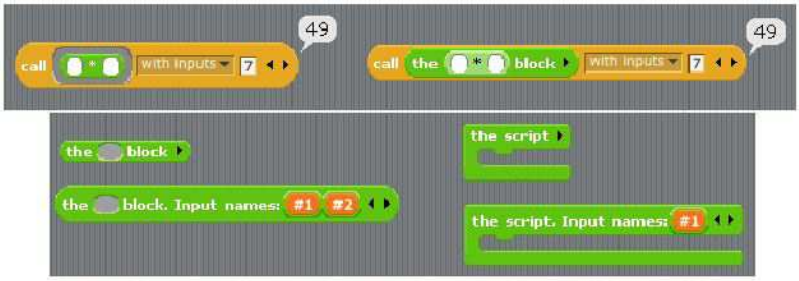

Figura 3.33: Forme alternative per the block e the script

Le frecce nei blocchi precedentemente citati sono utilizzate per fornire gli ingressi ad un blocco o ad uno script invece che utilizzare slot vuoti di ingresso. Ci sono due motivi per cui può essere necessario utilizzare the block o the script:

- si può avere la necessità di avere un maggior controllo nell'uso degli ingressi in una procedura di incapsulamento;
- si vuole utilizzare una procedura come valore per uno slot di ingresso che non è dichiarato per essere di tipo procedure perché anche altri tipi d'ingresso avrebbero comunque senso in quel contesto.

Si può fare chiarezza su questo concetto astratto con un esempio.

Figura 3.34: Esempio di inserimento di input in blocchi differenti

In questo esempio si vuole semplicemente inserire uno degli input in uno degli slot differenti. Se lasciamo tutti e tre gli slot vuoti, BYOB non ne riempirà

nessuno perché il numero di input forniti (2) non combacia con il numero di slot liberi. Comunque, una volta chiamato, il blocco fornisce il nome per i suoi input. BYOB non riempirà automaticamente i campi vuoti dato che l'utente ha preso il controllo. Infatti potrebbe esserci una ragione per cui si vuole assegnare un nome all'input esplicitamente.

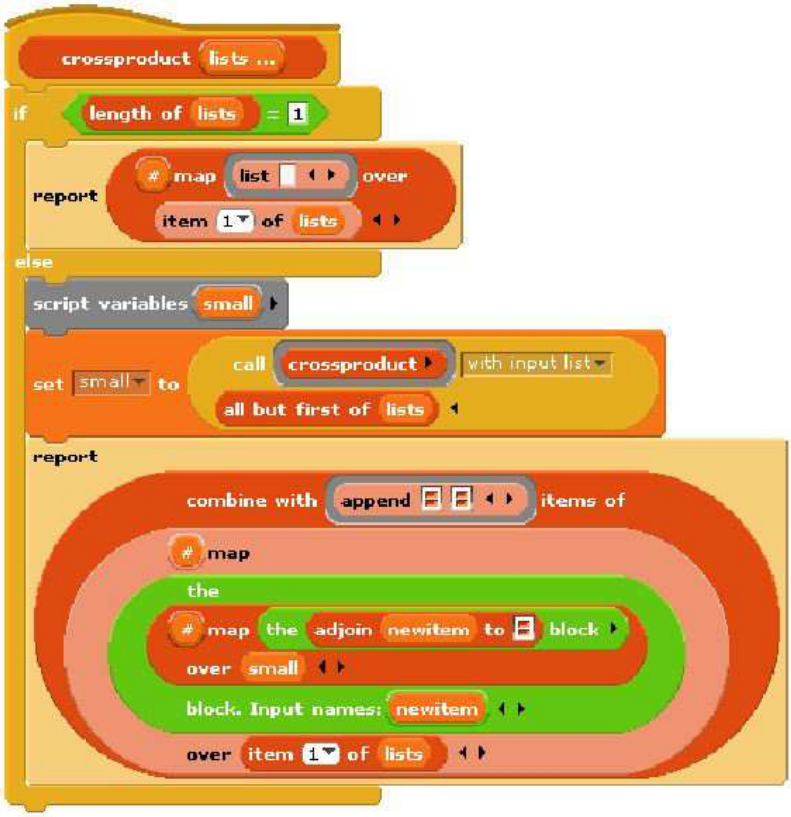

Figura 3.35: Esempio di affinamento del controllo dei dati di ingresso

In questo secondo esempio, prima cosa vi è la necessità di affinare il controllo dell'uso dei dati di ingresso.
Questa è la definizione di un blocco che prende un qualsiasi numero di liste e ritorna la lista di tutte le possibili combinazioni prendendo un elemento da ogni lista. La parte importante di questa discussione è che vicino alla fine ci sono due chiamate nidificate a map, una funzione higher order che utilizza una funzione in input e la applica ad ogni elemento di una lista passata come ingresso. Nel blocco più interno la funzione mappata è adjoin e il blocco prende due ingressi. Gli slot vuoti di tipo list prenderanno il loro valore da ogni chiamata da parte

di un elemento della lista degli ingressi della map interna. Non c'è modo però per la map più esterna di comunicare valori agli slot vuoti del blocco adjoin, bisogna quindi dare un nome esplicito, new item, al valore che la map esterna sta passando a quella più interna e poi trascinare la variabile nel blocco adjoin. The block e the script marcano i propri ingressi come dati, il primo prende blocchi di report e il secondo di script. Entrambi sonno provvisti di una freccia che punta verso destra che può essere usata per fornire esplicitamente i nomi per gli input. Cliccando sulla freccia compare un ovale arancione contenente il nome di default #1 per il primo input, #2 per il secondo e così via. Questi nomi di default possono essere cambiati con nomi più significativi cliccando sopra l'ovale arancione. Queste variabili possono essere trascinate nei block o script divenendo incapsulate. I nomi delle variabili di input sono detti parametri formali della procedura di incapsulamento.
Oltre al blocco list visto nell'esempio vi sono altri blocchi ove è possibile inserire procedure: set (il valore di una variabile a una procedura), say e think (per mostrare una procedura all'utente) e report (per un report che ritorna una procedura).

Figura 3.36: The block per controllare l'utilizzo degli ingressi con nome variabile del blocco incapsulato

Quando viene utilizzato the block per controllare l'utilizzo degli ingressi con nome variabile del blocco incapsulato, bisogna stare attenti a non inserire un bordo grigio attorno a the block il quale incapsulerebbe the block al posto di qualsiasi blocco utilizzato come ingresso.

Figura 3.37: Esempio errato

Sebbene sia inusuale è possibile creare una lista di blocchi che include the block, è questo il motivo per cui il bordo grigio non è automatico.

3.5.4 Forme speciali

Scratch ha un blocco if-else con due slot in cui inserire i blocchi a forma di C. Viene scelto uno o l'altro blocco in relazione alla verifica di una condizione di tipo booleano. Se tale condizione è verificata si accede al primo dei due blocchi, altrimenti si passa al secondo.

Dato che Scratch non dà importanza alla programmazione funzionale (un paradigma di programmazione in cui il flusso di esecuzione del programma assume la forma di una serie di valutazioni di funzioni matematiche) manca un blocco di report che consenta la scelta tra due espressioni. I blocchi rappresentati nell'esempio funzionano in questo semplice caso, ma non se si stanno usando operatori ricorsivi.

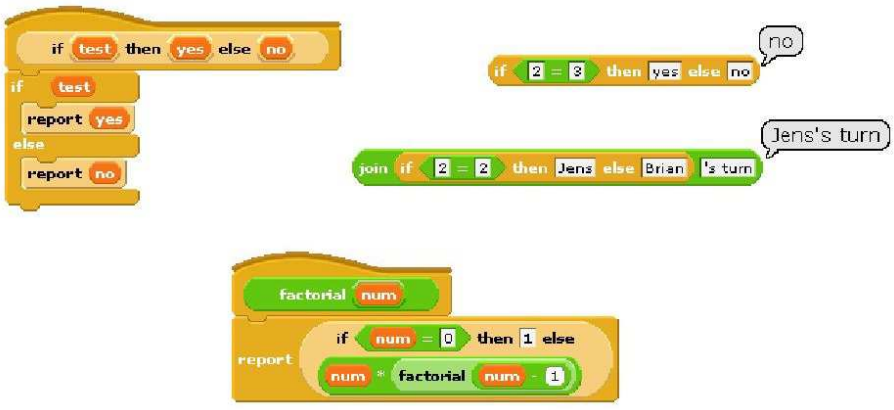

Figura 3.38: Esempio Scratch, funzionante solo se non si usa la ricorsione

Il problema è che quando un qualsiasi blocco di Scratch viene chiamato, tutti gli input sono calcolati prima che il blocco stesso venga eseguito. Il blocco conosce solo il valore degli input, non l'espressione utilizzata per calcolarli. In particolare tutti gli input del blocco if-then-else vengono calcolati per primi, questo significa che, anche nel caso base, il fattoriale proverà a chiamare se stesso ricorsivamente causando un loop infinito.

Vi è dunque la necessità che il nostro blocco if-then-else sia in grado di selezionare solo una delle due alternative che sono state calcolate. Vi è un meccanismo che permette questo: si dichiarano gli ingressi del then e dell'else come di tipo reporter invece che di tipo any. Quando il blocco viene chiamato, questi ingressi vanno inseriti nel blocco the block così che le espressioni, e non i valori, siano gli input.

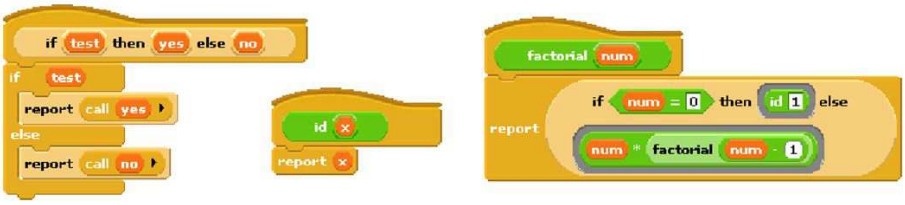

Figura 3.39: Esempio di fattoriale ricorsivo

Figura 3.40: Fattoriale di 15

La soluzione ideale sarebbe un blocco if che si comporta come quello appena descritto ritardando il calcolo degli ingressi, ma apparendo come la prima versione, facile da usare, ma non funzionante. Un blocco di questo tipo è possibile e prende il nome di forma speciale.
Per trasformare un blocco if in forma speciale si modifica il prototipo del blocco, dichiarando gli ingressi yes e no come tipo any invece che reporter. Lo script per il blocco rimane quello della seconda immagine, includendo l'uso di call per il calcolo di yes o no, ma non di entrambi, così gli slot appariranno come rettangoli bianchi di tipo any e non come reporter ovali e il blocco fattoriale apparirà come nel primo esempio. In un prototipo di un blocco speciale, gli input non calcolati sono indicati con un asterisco che segue il nome dell'ingresso.

Figura 3.41: Forma speciale

Le forme speciali non sono una nuova invenzione introdotta da BYOB: molti blocchi condizionali o cicli di Scratch sono forme speciali. Lo slot di ingresso esagonale nel blocco di if è direttamente un valore booleano perché viene calcolato una volta, prima che il blocco di if decida di lanciare o meno l'azione di ingresso. Tuttavia gli ingressi dei blocchi forever if, repeat until e wait until non possono essere booleani, ma devono essere booleani-non-valutati così che Scratch possa calcolarli ogni volta che si ripete il ciclo.

Finché Scratch non permetterà lo sviluppo di blocchi personalizzati può permettersi di non fare distinzioni tra booleani valutati e non, ma BYOB non può.

3.6 Programmazione orientata agli oggetti

Gli elementi principali del paradigma di progettazione orientato agli oggetti sono chiamati oggetti. Un oggetto proviene da una classe, costituente una specifica dei campi dati (detti anche variabili d'istanza) che l'oggetto contiene, oltre che dei metodi (operazioni) che l'oggetto può eseguire. A chi ne fa richiesta, una classe presenta una visione concisa e coerente degli oggetti che ne sono istanze, senza entrare in dettagli inutili e senza consentire l'accesso al funzionamento interno degli oggetti.
La programmazione orientata agli oggetti si pone tre obiettivi:

- Robustezza: il codice prodotto deve essere in grado di trattare input imprevisti, non esplicitamente definiti per quella applicazione;
- Adattabilità: il software deve poter evolvere nel tempo per rispondere a cambiamenti di esigenze;

- Riusabilità: il codice deve essere utilizzabile come una componente di sistemi diversi in varie applicazioni.

Inoltre si basa su quattro principi:

- Astrazione: un sistema complesso va scomposto nelle sue parti fondamentali e queste parti vengono descritte singolarmente in un linguaggio semplice e preciso;
- Incapsulamento: le diverse componenti di un sistema software non devono rivelare i dettagli interni delle rispettive implementazioni;
- Modularità: le diverse componenti del sistema software vengono divise in tante unità di funzionamento separate tra loro;
- Organizzazione gerarchica: vengono raggruppate assieme le funzionalità analoghe, in modo da definire un comportamento particolare come estensione di quello generale.

Nella programmazione orientata agli oggetti l'utente può interagire con i metodi mediante dei messaggi che possono essere semplicemente un nome, un numero o un oggetto. Il metodo esegue una particolare funzione e restituisce un valore all'utente che lo ha lanciato.
Le ragioni per utilizzare una programmazione orientata agli oggetti sono varie: uno dei motivi può essere il fatto che i dati sono nascosti (infatti ogni oggetto ha delle variabili locali a cui gli altri oggetti possono accedere solo inviando messaggi all'oggetto che li possiede). Un altro aspetto è quello della simulazione, per cui ogni oggetto rappresenta un'astrazione di un concetto reale e le interazioni tra gli oggetti sono analoghe alle interazioni tra persone reali o oggetti.
Il fatto che i dati siano nascosti è significativo per grossi progetti di programmazione, ma per gli utenti di BYOB è il concetto di simulazione ad essere importante. L'approccio alla programmazione ad oggetti da parte di BYOB è meno restrittivo che in altri linguaggi.
Dal punto di vista tecnico la programmazione ad oggetti si regge su tre aspetti:

- Ogni oggetto più inviare messaggi agli altri oggetti per comunicare con loro;
- Ogni oggetto tiene traccia e ricorda le istruzioni più significative che ha eseguito;
- Sarebbe poco pratico se ogni oggetto dovesse contenere metodi per tutti i messaggi che può accettare, molti di questi metodi sono infatti ridondanti. Occorre invece un modo per dire che il nuovo oggetto è uguale al vecchio eccetto che per poche differenze e solo quelle differenze devono essere esplicitamente programmate e aggiunte al nuovo oggetto: questo concetto è detto ereditarietà.

3.6.1 Sprite di prima classe

In Scratch ogni sprite è un oggetto. Infatti ogni sprite possiede delle variabili locali e ha dei metodi privati. Una animazione è una simulazione delle interazioni dei personaggi in gioco.
Tuttavia ci sono tre motivi per cui gli sprite sono meno versatili degli oggetti dei linguaggi orientati agli oggetti:

- L'invio dei messaggi ad altri sprite è primitivo dato che il messaggio può essere solo di tipo broadcast e quindi inviato a tutti indistintamente: non è possibile inviare un messaggio ad un unico sprite. Inoltre i messaggi non ricevono input e i metodi non possono ritornare un valore al chiamante;

- Non ci sono meccanismi di ereditarietà (è possibile duplicare uno sprite, ma il nuovo sprite non risente di alcun cambiamento nei metodi dello sprite d'origine);

- Nei linguaggi orientati agli oggetti, quest'ultimi sono dei dati: possono essere il valore di una variabile, l'elemento di una lista e così via, in Scratch non è ovviamente così.

In BYOB gli sprite sono un tipo di dato di prima classe. Possono essere creati e cancellati da uno script, salvati in una variabile o in una lista, è possibile inviare dei messaggi individuali ad altri sprite ed ereditare le proprietà di un altro sprite. Mentre in Scratch uno sprite può essere duplicato, in BYOB questo può venire clonato, creando un nuovo sprite che condivide metodi, variabili e liste con il genitore.
Il mezzo fondamentale con cui i programmi accedono agli sprite sono i blocchi object reporter. Il blocco object permetta la scelta dello sprite su cui agire. Tra le scelte ci sono anche due speciali blocchi *myself* e *all sprite*, per comunicare rispettivamente solo con lo sprite su cui sto agendo o su tutti gli sprite, inviando un messaggio broadcast (come quelli di Scratch).
Lo sprite selezionato in object può essere usato come input in altri blocchi che accettano input di qualunque tipo. Nell'esempio l'object è inserito nel blocco say: in questo modo lo sprite1 (su cui si sta lavorando) mostra un'immagine dello sprite2 (che è stato selezionato nel blocco object).

Figura 3.42: Esempio di uso del blocco object reporter

Il blocco object si trova nella categoria Sensings.

3.6.2 Inviare messaggi agli sprite

Uno sprite può accettare messaggi da altri sprite: i messaggi sono i blocchi (compresi i blocchi allsprites e this-sprite-only). Scratch prevede che uno sprite possa accedere ad alcuni attributi di un altro sprite con il blocco the <var> of <sprite>. BYOB estende questo blocco per poter accedere ad ogni proprietà dello sprite.

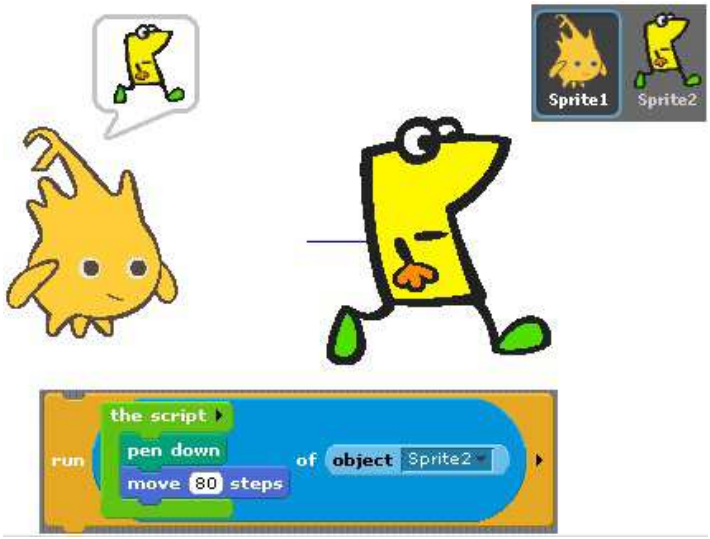

Figura 3.43: Esempio di uso del blocco x position of

Nell'esempio, lo Sprite1 comunica con lo Sprite2 utilizzando il blocco

Figura 3.44: Blocco x position of

In x position si inserisce il comando da dare allo Sprite2 e al posto di Sprite1 si inserisce colui che deve svolgere le istruzioni, ovvero l'oggetto Sprite2. Tutti i blocchi sono contenuti in un blocco run.
Lo Sprite1 è dunque riuscito a inviare un messaggio, in questo caso sotto forma di ordine di effettuare un movimento, allo Sprite2.
Si può notare che il blocco move è normalmente di tipo globale, quando è usato in uno script fa muovere qualunque sprite esegua lo script. Quando è combinato con x position of Sprite1 come nel precedente esempio, risulta effettivamente un blocco locale che fa muovere Sprite2 indipendentemente da che sprite lo stia azionando.
La funzione launch, che è già stata accennata, è identica al run, con l'eccezione che chiama il metodo come uno script separato, così lo script chiamante nel frattempo può fare qualcos'altro. Launch può essere pensato come un broadcast migliorato, mentre l'operazione di run di un metodo di un altro sprite è più simile a un'operazione di broadcast e attesa.

3.6.3 Stati locali negli sprite: variabili e attributi

Il modo in cui uno sprite memorizza la sua storia passata può essere di due tipi. Esso ha variabili create esplicitamente dall'utente con il bottone Make a variable ed ha attributi, ossia caratteristiche automaticamente assunte da uno sprite, come la posizione, direzione, colore della penna. Ogni variabile può essere esaminata attraverso l'ovale arancione che la caratterizza e modificata grazie al blocco set.
In Scratch gli attributi hanno un'interfaccia di programmazione meno uniforme. La direzione di uno sprite può essere esaminata con il blocco direction e modificata con il blocco point in direction <dir>. Può essere anche modificata utilizzando i blocchi turn, point towards e if on edge, bounce. Non ci sono modi per uno script di Scratch di esaminare il colore della penna di uno sprite, ma ci sono blocchi set pen color to <color>, set pen color to <number>, e change pen color by <number> per modificarlo. Il nome di uno sprite non può essere nè esaminato nè modificato dagli script, può essere modificato solo digitando un nuovo nome direttamente nello spazio che lo mostra. I blocchi che esaminano le variabili o gli attributi sono detti getter, quelli che li modificano setter.
In certi contesti è utile distinguere tra il valore di una variabile o di un attributo

e la variabile o l'attributo come oggetto a se stante. In BYOB una variabile o un attributo sono rappresentati da il loro blocco getter. Come visto precedentemente, abbiamo già il mezzo per esprimere la distinzione tra il valore e l'oggetto. Per quest'ultimo inseriamo il blocco getter nel blocco the block. Per esempio, il valore riportato da direction è un numero che indica la direzione corrente dello sprite, mentre il valore riportato da the (direction) block è un blocco costituito dall'attributo direction stesso.

Figura 3.45: Il blocco direction

L'uso di the block per rappresentare un attributo funziona solo se c'è un blocco getter per quell'attributo. Vogliamo essere in grado di rappresentare ogni componente dello stato dello sprite. Invece di aggiungere dozzine di blocchi di getter alle palette, BYOB introduce un singolo blocco attribute nella categoria Sensing. Il suo ingresso del menù a cascata ha un'entry per ogni attributo dello sprite. Il valore riportato da attribute (direction) è lo stesso numero riportato dal blocco direction della categoria Motion. Il valore riportato dal the (attribute (direction)) block è lo stesso attributo direzione. In modo simile, attribute (draggable?) ritorna vero se lo sprite può essere trascinato nella modalità di presentazione (cioè se l'icona pad lock in alto vicino all'area di scripting è sbloccata).

Figura 3.46: Unlock

The (attribute (draggable?)) block indica se lo sprite può essere trascinato.

Figura 3.47: Esempio di blocco attribute draggable?

Un beneficio dei blocchi di prima classe di BYOB è che non è necessario l'equivalente di attribute per i setter. Invece si estende il blocco set<variable> to <value> permettendo ad un attributo (con the block) di essere trascinato nella lista in cascata delle variabili: set (the (attribute (draggable?)) block) to <true>. Si può notare che questo funziona solo per variabili e attributi getter e non per variabili e attributi in generale. Non è possibile dire set (the (sqrt of (x)) block) to 5 per assegnare il valore 25 alla variabile x.

Figura 3.48: Esempio di blocco attribute draggable?

Figura 3.49: Esempio di blocco attribute draggable?

3.6.4 Prototipi: genitori e figli

La maggior parte dei linguaggi di programmazione orientati agli oggetti usano un approccio di tipo classe/istanza nella creazione di oggetti. Una classe è un tipo particolare di oggetto e una sua istanza è un reale oggetto di quel tipo. Ad esempio se la classe è Automobile, le istanze di tale classe sono l'Automobile di Mario, l'Automobile di Giovanni, l'Automobile di Elisa. La classe specifica i metodi condivisi da tutte le istanze e ogni istanza contiene dati specifici.
BYOB usa un approccio differente, detto prototipo, in cui non c'è distinzione tra classi e istanze. Questo tipo di approccio è conveniente per un tipo di lavoro più sperimentale: l'utente crea un singolo sprite dell'istanza con i suoi metodi (i blocchi) e dati (variabili e liste) secondo le sue esigenze, una volta terminato viene usato come prototipo da cui clonare le altre istanze. Nel caso si dovesse scoprire un bug o si volesse modificare il codice, basta modificarlo nello sprite padre ed esso risulterà modificato automaticamente anche negli sprite figli. L'uso di prototipi si adatta bene alla filosofia di Scratch in cui ogni componente del

progetto dev'essere ben visibile nello stage. Le classi dei linguaggi orientati agli oggetti invece, sono entità astratte ed invisibili che devono essere costruite prima di poter realizzare un qualunque concreto oggetto. Ci sono tre modi per creare uno sprite figlio:

- Cliccando col tasto destro del mouse sullo sprite da clonare nello *sprite corral* (il riquadro in basso a destra nello schermo), apparirà un menu a tendina in cui va scelto clone;
- Nella palette Operators si trova un blocco clone che crea e riporta uno sprite figlio;
- Gli sprite hanno un attributo parent che può essere impostato come un qualunque attributo.

Figura 3.50: Il blocco clone

Figura 3.51: Impostazione di un attributo parent

In progetti in cui vengono clonati sprite, il loro numero totale può essere anche molto elevato e molti potrebbero sembrare identici esteticamente, cosa che rende difficile capire qual è il prototipo originario.
Cliccando con il tasto destro sopra ogni sprite si può vedere l'elenco degli eventuali figli e il padre.

Figura 3.52: Genitore dello Sprite3

3.6.5 Ereditarietà mediante delega

L'ereditarietà è una relazione di generalizzazione/specificazione: la superclasse definisce un concetto generale e la sottoclasse rappresenta una variante specifica di tale concetto generale. Uno dei maggiori vantaggi dell'ereditarietà è la possibilità di creare versioni specializzate di classi già esistenti, cioè di crearne dei sottotipi.
Supponiamo che in un programma si usi una classe Animale contenente dati per specificare, ad esempio, se l'animale è vivo, il luogo in cui si trova, quante zampe ha, ecc.; in aggiunta a questi dati la classe potrebbe contenere anche metodi per descrivere come l'animale mangia, beve, si muove, si accoppia, ecc. Se si volesse creare una classe Mammifero molte di queste caratteristiche rimarrebbero esattamente le stesse di quelle dei generici animali, ma alcune sarebbero diverse. Diremmo quindi che Mammifero è una sottoclasse della classe Animale. Nel definire la nuova classe non è necessario specificare nuovamente che un mammifero ha le normali caratteristiche di un animale, ma basta aggiungere le caratteristiche peculiari che contraddistinguono i mammiferi rispetto agli altri animali e ridefinire le funzioni che, pur essendo comuni a tutti gli altri animali, si manifestano in modo diverso.

In BYOB il concetto di ereditarietà si realizza clonando uno sprite. Il clone così creato eredita le proprietà del genitore, con proprietà si intendono gli script, i blocchi, le variabili, i costumi, le liste e i suoni.
Quando nello sprite padre viene creato un nuovo blocco e si seleziona l'opzione 'For this sprite only' allora il blocco sarà visibile solo nello sprite in cui è stato creato e nei figli.

Figura 3.53: Make a block con opzione For this sprite only

Nel blocco figlio, i blocchi ereditati hanno un colore più chiaro, mentre le proprietà del figlio hanno i colori classici.
Quando viene creato un clone, di default esso condivide tutte le sue proprietà con il padre, eccetto gli attributi di sistema. Se il valore di una proprietà condivisa cambia nel genitore, allora il figlio risentirà della modifica, mentre se cambia nel figlio si interrompe la condivisione, nel figlio viene creata una nuova versione privata della variabile e il padre non risentirà delle modifiche.

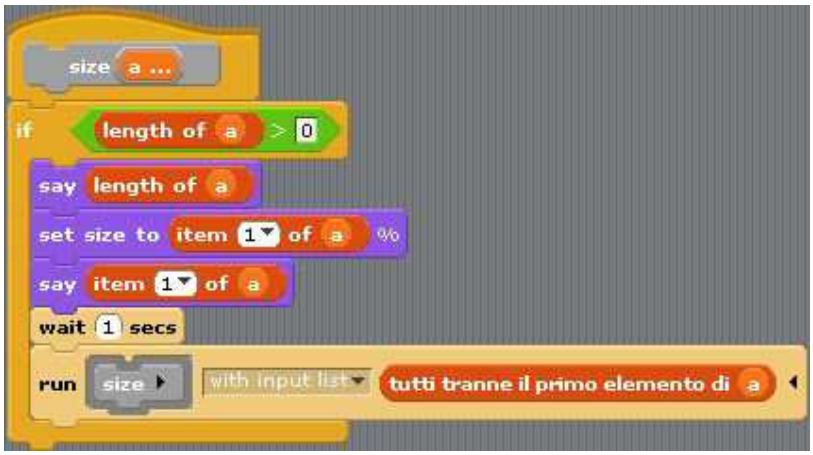

Figura 3.54: Blocco originario padre

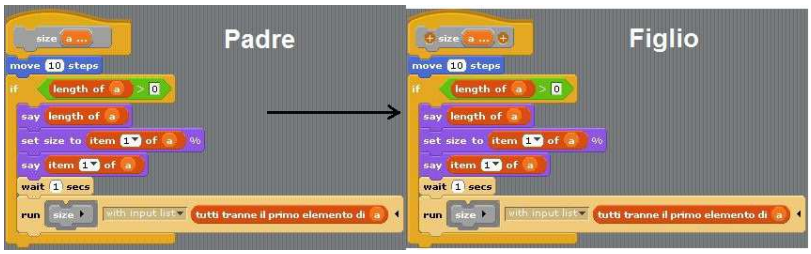

Figura 3.55: La modifica nel padre si ripercuote nel figlio

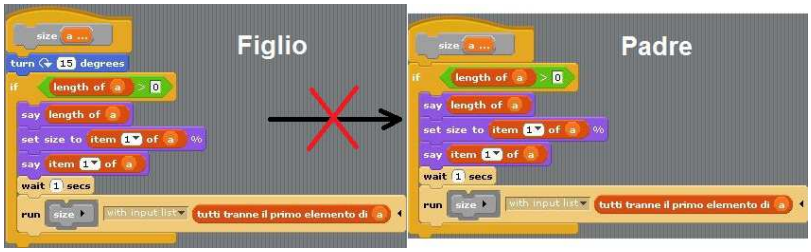

Figura 3.56: La modifica nel figlio non si ripercuote nel padre

Ogni singola proprietà del genitore può essere ereditata o meno dal figlio, e il figlio può avere proprietà aggiuntive rispetto al genitore.
Come per tutte le variabili, anche in questo caso vengono modificate con i blocchi *set* o *change*, mentre i blocchi possono essere editati con il Block Editor e i suoni e i costumi degli sprite con gli apposi editor.
Se non si seleziona l'opzione 'For this sprite only', le modifiche ad un blocco si ripercuoteranno in tutti gli sprite che lo contengono, indipendentemente che la modifica sia stata fatta in uno sprite genitore o figlio.
Se il figlio vuole che una delle sue proprietà da privata diventi pubblica, esso elimina la versione privata della proprietà utilizzando il blocco delete. Delete ha uno slot in cui inserire l'input che accetta qualunque proprietà dello sprite. Essendo lo slot input di tipo reporter è possibile inserirvi un blocco the block. Occorre notare comunque che gli sprite non sono proprietà, quindi non si può inserire uno sprite nel blocco the block.

Figura 3.57: Messaggio delagato

Quando uno sprite riceve un messaggio per cui non ha un blocco corrispondente, allora il messaggio è delegato al genitore, invece se il blocco necessario è disponibile, il messaggio non è delegato e viene gestito dallo sprite.
Un esempio completo di sprite clonato che eredità le proprietà del genitore è visibile nel paragrafo *3.7 Costruzione esplicita di oggetti* in cui viene realizzato un contatore buzzer che eredità e estende le proprietà del padre contatore.

3.6.6 Sprite annidati

A volte è utile creare una specie di super sprite composto da pezzi che possono funzionare assieme, ma che possono anche essere articolati separatamente.
Il classico esempio è il corpo di una persona composto dal tronco, gli arti e la testa. BYOB fa sì che uno sprite sia usato come collegamento tra i vari pezzi. Per creare sprite di questo tipo occorre scegliere uno sprite ancora a cui saranno attaccati gli altri sprite. Nell'esempio di figura 3.58 si è scelto di usare come ancora lo Sprite1, quindi si trascina il piccolo riquadro dello Sprite2 fino a sovrapporlo con lo Sprite1 nello Stage.

Figura 3.58: Impostazione dello sprite ancora

A questo punto, per ogni movimento compiuto dallo sprite1, lo sprite2 farà altrettanto. Si può notare che nell'icona dello sprite2, in alto a destra è rappresentata una piccola freccia, questo simbolo sta a significare che, per ogni rotazione dello sprite1, lo sprite2 si muoverà in maniera solidale alla sua ancora, ovvero l'angolo tra lo sprite1 e lo sprite2 resta costante.

Figura 3.59: Miniatura degli sprite, caso con freccia

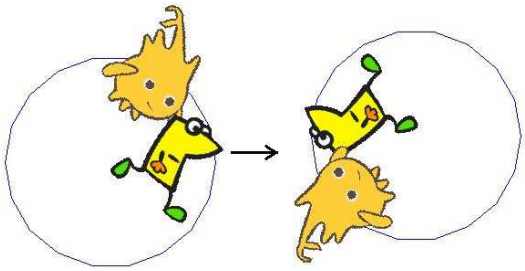

Figura 3.60: Esempio di sprite annidati con movimento solidale all'ancora

Cliccando sulla freccia, questa diventa un puntino. Con questa impostazione, quando lo sprite1 ruota, lo sprite2 non si muove più in modo solidale al primo, bensì resta fisso e non cambia angolazione rispetto a quella che aveva inizialmente.

Figura 3.61: Miniatura degli sprite, caso con punto

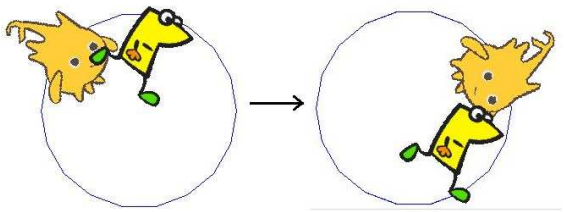

Figura 3.62: Esempio di sprite annidati con movimento fisso

3.6.7 Lista di attributi

L'immagine che segue rappresenta una lista di attributi di uno sprite. Quattro di essi non sono attributi, bensì liste o cose legate agli attributi:

- Costumes: lista di nomi dei costumi dello sprite;
- Sounds: elenco dei nomi dei suoni dello sprite;
- Children: lista di sprites figli dello sprite;
- Parts: lista di sprites, il cui attributo ancora è lo sprite corrente.

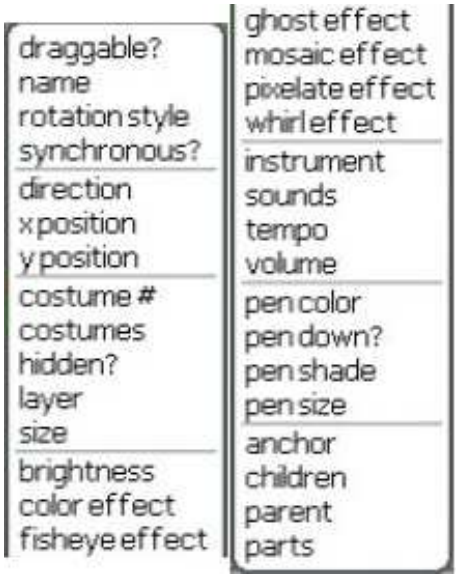

Figura 3.63: Lista di attributi

3.7 Costruzione esplicita di oggetti

Qualsiasi linguaggio con procedure di prima classe permette di implementare esplicitamente oggetti: questo è un utile esercizio per renderli un concetto più pratico e meno misterioso. L'idea centrale e la sua implementazione è che un oggetto è rappresentato come una procedura di consegna che prende i messaggi in input e li ritorna al metodo corrispondente.

3.7.1 Stato locale con variabili di script

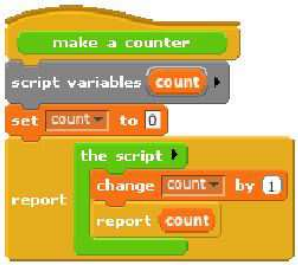

Figura 3.64: Esempio di contatore

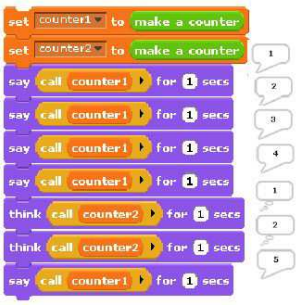

Figura 3.65: Output di contatore

Lo script implementa una object class, un tipo di oggetto, detta classe counter. Nella prima versione semplificata c'è solo un metodo così non è necessario passare messaggi espliciti. Quando è chiamato il blocco make a counter, esso ritorna una procedura, creata dal blocco dello script all'interno del suo corpo. Quella procedura implementa uno specifico oggetto contatore, un'istanza della classe counter. Quando viene invocato un'istanza di contatore aumenta e ritorna la sua variabile count. Ogni contatore conta localmente.
Dal punto di vista della procedura Make a counter, ogni invocazione causa la

creazione di una nuova variabile count associata al blocco. Solitamente un blocco di variabili è temporaneo e cessa di esistere quando termina la procedura, ma questa è differente perché la procedura ritorna il valore ad un'altra procedura che crea un collegamento alla variabile count così che rimanga attiva.
In questo approccio alla programmazione orientata agli oggetti si stanno rappresentando entrambe le classi e le istanze di una procedura. Il blocco make a counter rappresenta la classe, mentre ogni istanza è rappresentata da uno script senza nome creato ogni volta che make a counter è chiamato. Le variabili di script create all'interno del blocco make a counter, ma al di fuori del blocco the script, sono variabili di istanza appartenenti ad un particolare contatore.

3.7.2 Messaggi e procedure di consegna

Nella classe semplificata qui sotto c'è solo un metodo e non ci sono messaggi, viene solo chiamata l'istanza per portare fuori il risultato del suo unico metodo.

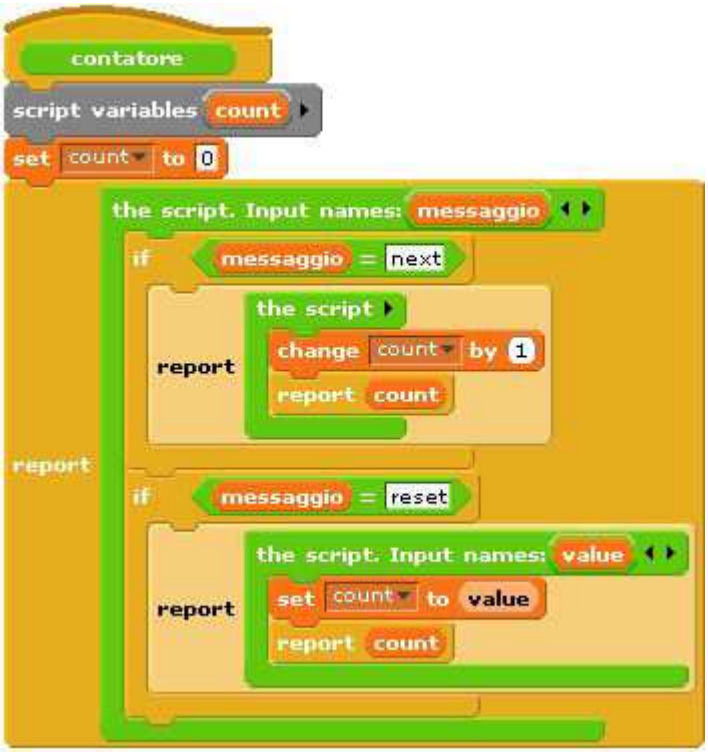

Figura 3.66: Esempio di contatore con dispatch procedure

In questo caso, il blocco make a counter rappresenta la classe counter e lo script crea una variabile locale count ogni qualvolta viene invocato. Il blocco esterno

più largo, the script, rappresenta un'istanza. Esso è una dispatch procedure: prende come ingresso un messaggio (anche solo una parola) e riporta un metodo. I due blocchi più piccoli the script sono metodi. Quello più in alto è il metodo next, quello più in basso è il metodo reset. Quest'ultimo richiede un ingresso chiamato value. Nella versione precedente, chiamando l'istanza viene fatto tutto il lavoro, in questa versione l'istanza chiamata dà accesso ad un metodo, il quale deve essere chiamato per concludere il lavoro. E' possibile creare un blocco ask che esegue entrambe le procedure in un'unica volta.

Figura 3.67: Blocco ask che esegue entrambe le procedure in una volta

Il blocco ask richiede due ingressi, un oggetto e un messaggio. Inoltre accetta opzionalmente ulteriori ingressi che BYOB inserisce in una lista. Tale lista è chiamata args all'interno del blocco. Il blocco ha due blocchi call nidificati, il più interno chiama l'oggetto (per esempio la dispatch procedure). La dispatch procedure accetta solo un ingresso, message. Esso riporta un metodo, il quale può prendere un qualsiasi numero di ingressi (questa è una situazione che necessita dell'opzione with input list di call).

Figura 3.68: Output di contatore

3.7.3 Ereditarietà tramite delega

A questo punto il nostro oggetto ha variabili di stato locali e consente il passaggio di messaggi. Rimane da trattare il concetto dell'ereditarietà. E' possibile fornire

questa caratteristica utilizzando la tecnica della delega. Ogni istanza della classe figlio contiene un'istanza della classe genitore.

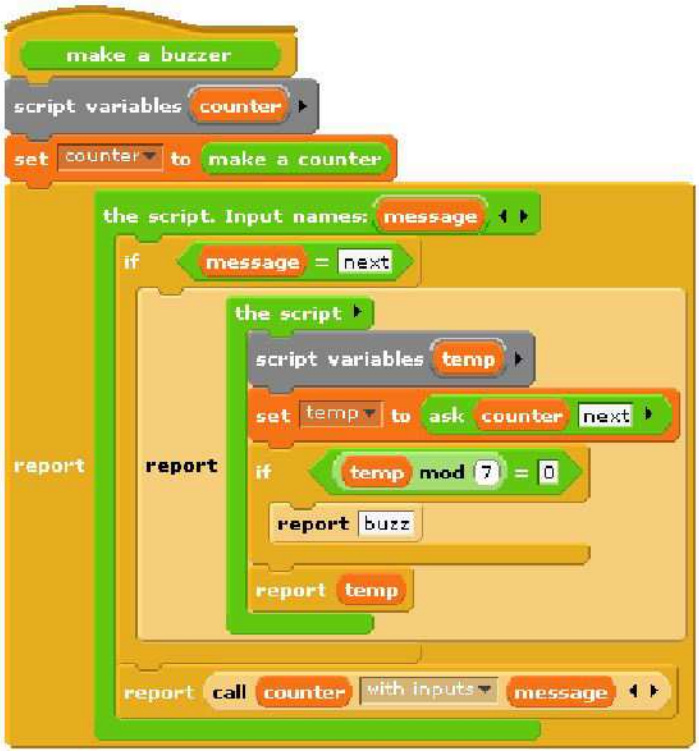

Figura 3.69: Esempio di contatore buzzer

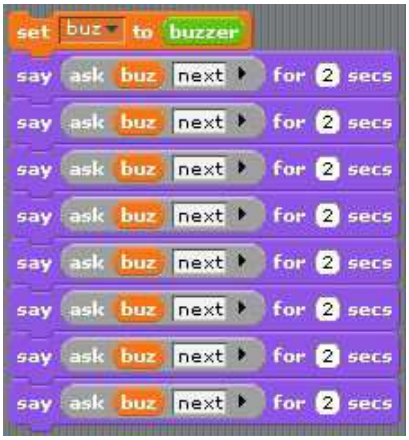

Figura 3.70: Esempio di contatore buzzer

L'esempio implementa una classe buzzer che è figlia di counter. Invece di avere un contatore (un numero) come variabile di stato locale, ogni buzzer ha un contatore (un oggetto) come una variabile di stato locale. La classe sovrascrive e specializza il metodo next, riportando quello che il counter restituisce finché quel valore è divisibile per 7, nel cui caso riporta buzz. Se il messaggio è qualcosa di diverso da next, allora il buzzer invoca semplicemente la dispatch procedure dell'oggetto counter. In questo modo il counter gestisce ogni messaggio che il buzzer non è in grado di gestire.

Capitolo 4

Conclusione

Dopo aver analizzato dettagliatamente Scratch e BYOB emerge chiaramente come questi due linguaggi siano estremamente differenti pur poggiando sulle stesse basi. Scratch è un linguaggio estremamente semplice che offre possibilità limitate. Come già detto, quelli che possono essere visti come difetti di Scratch, sono in realtà suoi punti di forza. Il fatto che non permetta all'utente di svolgere comandi più complessi, ad esempio non permetta di gestire la ricorsione, né di creare blocchi personalizzati, non è una mancanza, bensì una scelta dei progettisti.

Scratch è un linguaggio rivolto ai più giovani o comunque a chi non ha familiarità con l'informatica e i linguaggi di programmazione. Pone molta attenzione ad una grafica piacevole e alla possibilità di interagire con gli sprite. Navigando nel sito di Scratch e osservando i progetti caricati dagli utenti, si può notare come nella maggioranza dei casi questi siano semplici animazioni o qualche gioco estremamente rudimentale. L'utente medio di Scratch, che non sa nulla di linguaggi di programmazione testuale, non si pone il problema della realizzazione di blocchi personalizzati o del perché Scratch non permetta la gestione della ricorsione. Quindi fornire strumenti più raffinati ad un utente di Scratch non solo è inutile, dato che non verranno mai utilizzati, ma anche controproducente. Infatti, in una versione precedente di Scratch erano stati introdotti dei blocchi per la realizzazione di procedure, questo strumento più che portare un vantaggio ha portato confusione negli utenti, tanto che nelle versioni successive è stato scelto di eliminarlo.
Un altro aspetto estremamente positivo di Scratch è il fatto che la comunità on line di sviluppatori è molto attiva. Nel caso di difficoltà nella realizzazione di un progetto si può chiedere aiuto nel forum che è molto frequentato e gli utenti più esperti non mancheranno di aiutare i nuovi arrivati. Nel sito è possibile caricare i propri progetti che possono essere visualizzati e scaricati da altri utenti. Il numero di progetti caricati è impressionante e possono essere utilissimi spunti per chi si accinge a creare il proprio progetto.

Gli utenti più esperti, o che comunque hanno già familiarità con la programmazione, incontrano delle limitazioni in Scratch. Fortunatamente viene loro incontro BYOB che permette la realizzazione di blocchi personalizzati e consente una reale programmazione orientata agli oggetti. Purtroppo BYOB è utilizzato da un ristretto numero di utenti (rispetto a Scratch) e, nel caso si incontri qualche difficoltà, si può intervenire in una sottocategoria del forum di Scratch riservata a BYOB. Purtroppo la navigazione nel forum è molto disagevole dato che si tratta di un unico topic di 225 pagine e trovare un'informazione utile lì dentro è abbastanza complicato. I due creatori di BYOB sono sempre molto attivi nel forum e chiariscono spesso i dubbi degli utenti e sono sempre pronti ad accogliere nuove idee per migliorare il loro programma.
Osservando i progetti realizzati in BYOB pubblicati in rete, si può osservare come in pochissimi di questi siano utilizzate strutture dati complesse, come liste o alberi. Raramente viene utilizzata anche la ricorsione. Quindi la potenzialità che viene maggiormente sfruttata dall'utenza è la possibilità di realizzare dei blocchi personalizzati.
Nei vari esempi presenti in rete non ce n'è nessuno in cui vengono utilizzate strutture dati complesse, anche a titolo didattico. Nei vari esempi realizzati per la stesura di questa tesi ho avuto modo di provare a realizzare queste strutture più complesse, la loro realizzazione però non è stata molto agevole. Finché si realizzano dei semplici progetti composti da non troppi blocchi, allora questi sono facili da gestire, se i progetti cominciano a diventare più corposi la loro realizzazione non è molto pratica. Inoltre bisogna notare che le liste di BYOB non sono delle liste classiche, quanto piuttosto degli array, quindi la loro trattazione risulta semplificata, ma questo perché non si sta operando su liste classiche, bensì su una loro versione semplificata.

In conclusione, sia Scratch che BYOB sono strumenti con grandissime potenzialità nonché un ottimo modo per avvicinarsi alla programmazione. Parlando per esperienza personale, il mio primo approccio con il mondo della programmazione è stato proprio attraverso un linguaggio di tipo LOGO. Le prime volte che si scrive un programma, le azioni svolte possono risultare poco chiare, anche perché spesso occorre inserire delle dichiarazioni o comunque altri elementi che vengono approfonditi più avanti. Utilizzando un linguaggio visivo si unisce il vantaggio e la soddisfazione di vedere il proprio programma funzionare senza però utilizzare costrutti complicati, che quindi possono risultare poco chiari. BYOB offre anche il vantaggio di un approccio orientato agli oggetti, un'ottima base di partenza per gli studenti che si avvicinano al mondo della programmazione. Tuttavia, una volta apprese le basi e fatto un po' di pratica è meglio passare ad un linguaggio di tipo testuale che permette una trattazione paradossalmente più semplice dei costrutti più complicati.

Bibliografia

[1] “Manuale di byob.” http://byob.berkeley.edu/BYOBManual.pdf.

[2]

[3] “Ricorsione per scratch e byob.” http://www.nutt.net/2011/10/08/teaching-recursion-with-scratch-byob-2/.

[4] “Ricorsione.” http://wiki.scratch.mit.edu/wiki/Recursion.

[5] “Manuale di scratch.” http://web.media.mit.edu/ jmaloney/papers/ScratchLangAndEnvironment.pdf.

[6] “Forum di scratch.” http://scratch.mit.edu/forums/.

[7] “Progetti realizzati in byob.” http://www.xleroy.net/ByobTuto/Thumbnails.html.

[8] “Sito ufficiale di byob.” http://byob.berkeley.edu/.

Elenco delle figure