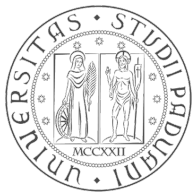


Università degli Studi di Padova

Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica

Introduzione a Byob: ambiente e applicazioni

Laureando

Daniele Bovo

Relatore

Prof. Michele Moro

Anno Accademico 2011/2012

Indice

Sommario

In questo elaborato ci si accinge ad analizzare Byob, un linguaggio di programmazione visuale ancora giovane, che è tuttora in continuo sviluppo tramite il rilascio di nuove versioni. Questo ambiente è un diretto discendente di Logo, il primo ad essere stato creato tra i linguaggi di questo genere, e una vera e propria evoluzione di Scratch, altro linguaggio visuale che vanta un successo non indifferente in tutto il mondo da alcuni anni a questa parte. Ma ciò che accomuna questi tre linguaggi, oltre al modus operandi simile, è principalmente la filosofia che sta dietro al loro apprendimento che prende il nome di costruzionismo. Dopo una breve introduzione a questa dottrina dall'impronta fortemente didattica, che getta le basi per un nuovo paradigma di programmazione, verranno descritti proprio i legami tra questi linguaggi più o meno recenti, focalizzando l'attenzione su Byob e sulle novità che esso ha introdotto rispetto a Scratch. In ultima analisi verranno esposti i risvolti meno intuitivi di questo ambiente apparentemente semplice, che concernono la programmazione orientata agli oggetti, i dati di prima classe e concetti più ostici come quello dell'ereditarietà. Per facilitare la lettura e integrarla con una parte operativa è utile, ma non indispensabile, disporre di Byob, facilmente reperibile sia come eseguibile installante sia in comoda versione stand alone.

Capitolo 1

Introduzione

In informatica, un linguaggio di programmazione è un linguaggio formale, paragonabile ad un qualsiasi linguaggio naturale per la presenza di caratteristiche distintive quali il lessico, la sintassi o la semantica, sempre ben definiti secondo rigide regole. Programmando un linguaggio di questo tipo si rende utilizzabile una macchina formale (o una sua implementazione, tipicamente un computer) attraverso il controllo del suo comportamento. I comandi sono impartiti tramite la scrittura di un programma sotto forma di codice.

In generale non ha senso parlare di linguaggi migliori o peggiori in quanto ogni linguaggio nasce per affrontare una classe di problemi più o meno ampia, a seconda degli ambiti in cui viene utilizzato e per i quali viene progettato. Dovendo però valutare se un dato linguaggio sia conforme o meno ad un certo impiego, è necessario considerarne a priori le caratteristiche, suddivisibili in due gruppi: le caratteristiche intrinseche e quelle esterne. Tra le caratteristiche intrinseche compaiono espressività, leggibilità, generalità, robustezza, modularità, flessibilità, efficienza e coerenza e infine didattica. Quest'ultima indica la semplicità del linguaggio e la rapidità con cui lo si può imparare. Il Basic, per esempio, è un linguaggio facile da imparare: poche regole, una sintassi molto chiara e limiti ben definiti fra quello che è permesso e quello che non lo è. Oppure il Pascal che non solo ha i pregi del Basic ma educa anche il neo-programmatore ad adottare uno stile di programmazione corretto che evita molti errori e porta a scrivere codice migliore. Al contrario, il C non è un linguaggio didattico perché pur avendo poche regole ha una semantica molto complessa, a volte oscura, che lo rende molto efficiente ed espressivo a scapito del tempo necessario a padroneggiarla.

Negli ultimi anni, i dipartimenti di informatica presso diverse università stanno compiendo sforzi per attirare un maggiore numero di studenti. Alcuni paesi, come gli Stati Uniti d'America, hanno dichiarato che risolvere la caren-

za di studenti in campo informatico è una priorità nazionale. Uno dei metodi principali di questo sforzo è l'abbandono di un approccio eccessivamente improntato sulle tecniche di programmazione e la ricerca di un linguaggio che eviti la complessità sintattica.

La risposta a questa problematica sembra arrivare negli anni sessanta con l'avvento di Logo, un linguaggio di programmazione funzionale che esprime caratteristiche di modularità, estensibilità, interattività e flessibilità. Con l'avanzare del tempo linguaggi improntati su tali caratteristiche e in particolare influenzati dal Logo non mancarono di essere sviluppati; tra tutti citiamo Byob e Scratch (l'uno derivato direttamente dall'altro rispettivamente) che assieme formano gli argomenti principali di questa tesi.

Con l'arrivo di questo nuovo tipo di programmazione, più intuitivo e votato alla didattica, hanno presto iniziato a diffondersi i primi corsi per l'istruzione all'informatica anche per quelle fasce d'età nelle quali, prima, non si pensava di cominciare un vero e proprio approccio all'utilizzo del computer. Inizialmente i linguaggi erano intesi a educare studenti in particolari discipline quali l'informatica o i sitemi ad età perlopiù avanzate mentre, dopo l'introduzione di Logo e dei suoi discendenti, l'uso del computer mirava allo sviluppo intellettuale in generale oltre che nell'apprendimento di particolari materie.

Ad oggi il linguaggio Logo risulta in parte superato e viene spesso preso in considerazione l'utilizzo di programmi alternativi che ne riprendono la struttura operativa e il modello di pensiero. Ci si soffermerà inizialmente su Scratch, sulle sue lacune e sulle sue potenzialità che verranno trattate abbastanza da vicino. La descrizione nel particolare di questo linguaggio che non è l'argomento portante della tesi non vuole essere una divagazione ma un avvicinamento graduale a Byob il quale, fatta eccezione per le caratteristiche introdotte rispetto a Scratch, risulta, per l'utente alle prime armi, molto simile nella struttura e nell'aspetto. La parte iniziale riguarda la teoria del costruttivismo che invece è uguale per ogni linguaggio improntato sulla didattica. Per quanto tale introduzione non sia essenziale dal punto di vista prettamente informatico della programmazione, è utile a spiegare le premesse che stanno dietro la realizzazione di questo tipo di linguaggi per poterne poi accettare i limiti con più consapevolezza.

I contenuti dell'elaborato riguardanti entrambi i programmi sono facilmente interpretabili anche senza conoscenze pregresse sui linguaggi trattati ma, nonostante ciò, un'infarinatura generale non può che aiutare la lettura e la comprensione, specie degli argomenti più ostici che verranno trattati

riguardanti tecniche di programmazione più avanzate. Nelle ultime fasi di analisi, infatti, (dal capitolo 8 in poi) verranno trattati il paradigma della programmazione orientata agli oggetti e una sua possibile interpretazione tramite le potenzialità fornite da Byob. Nonostante il prologo di questa sessione più avanzata sia composto da un breve riassunto che fornisce le principali definizioni della programmazione ad oggetti è necessaria una competenza almeno basilare dei concetti che la riguardano e dei linguaggi che la adoperano.

Viene caldamente consigliato l'utilizzo concreto dei programmi trattati e l'esercitazione tramite gli esempi man mano che si avanza con la lettura; questo renderà più immediata l'assimilazione.

L'obiettivo generale è quello inquadrare Scratch e Byob, valutandone lacune e potenzialità per poi, di conseguenza, collocarli in un contesto nel quale possano restituire attraverso il loro utilizzo un buon risultato per l'apprendimento, sia per i neofiti sia per gli utenti più esperti.

Capitolo 2

Filosofia della programmazione

2.1 Costruzionismo

> *L'obiettivo è di insegnare in modo tale da offrire il maggiore apprendimento col minimo di insegnamento [...] L'altro fondamentale cambiamento necessario rispecchia un proverbio africano: se un uomo ha fame gli puoi dare un pesce, ma meglio ancora è dargli una lenza e insegnargli a pescare.*
>
> Seymour Papert

Seymour Papert, matematico, informatico e pedagogista, è uno dei pionieri dell'intelligenza artificiale ed è internazionalmente riconosciuto come grande pensatore sui modi di cambiare l'apprendimento grazie ai computer. Nato e formatosi in Sudafrica, Papert svolse ricerca matematica all'Università di Cambridge e successivamente collaborò con Jean Piaget all'Università di Ginevra dal 1958 al 1963. È stata proprio questa esperienza che lo ha portato ad introdurre il concetto di costruzionismo legato alla teoria dell'apprendimento. Infatti, secondo Papert, il processo di apprendimento è un processo di costruzione di rappresentazioni più o meno corrette e funzionali del mondo con cui si interagisce. Rispetto al costruttivismo, il costruzionismo introduce il concetto di artefatti cognitivi, ovvero oggetti e dispositivi che facilitano lo sviluppo di specifici apprendimenti. Inutile dire che secondo una delle idee

di Papert in mezzo a tali artefatti presenziava il computer, usato come strumento didattico abbinato ad un linguaggio di sua creazione, il LOGO. Da un punto di vista prettamente filosofico e generale il costruttivismo è una dottrina secondo la quale la vita è un processo cognitivo che nasce dall'esperienza individuale ed è quindi riconducibile ad una sequenza di singoli eventi necessari alla formazione di un qualsiasi contenuto di conoscenza. Attraverso tale processo ogni essere vivente genera il proprio mondo, passando stadi di disequilibrio e assestamento. L'essere umano, a prescindere dall'età, ha bisogno di avere a disposizione materiali concreti affinché la conoscenza acquisita sia tanto più vicina alla realtà. Papert parte dall'osservazione di attività di alcune civiltà africane in cui i bambini costruivano case in scala o manufatti in giunco. Secondo Papert, la mente ha bisogno di materiali da costruzione appropriati, esattamente come un costruttore: il prodotto concreto può essere mostrato, discusso, esaminato, sondato e ammirato. La lentezza dello sviluppo di un particolare concetto da parte del bambino non è dovuta alla maggiore complessità o formalità, ma alla povertà della cultura di quei materiali che renderebbero il concetto semplice e concreto. Il bambino apprende così con l'aiuto di artefatti cognitivi. In particolare, Papert sostiene l'uso del computer come supporto all'istruzione e ambiente d'apprendimento che aiuta a costruirsi nuove idee. Il computer viene così usato come macchina per simulare. Realizza anche il LOGO, un linguaggio di programmazione formalmente molto rigoroso, derivato dal LISP, orientato alla gestione delle liste ed alla grafica della tartaruga (mutuata dal Pascal), comprensibile ed usabile anche da bambini delle scuole elementari, dimostrando tra l'altro l'utilità del computer come supporto per l'apprendimento anche per i più piccoli. L'interprete del LOGO è, infatti, uno strumento che consente ai bambini di utilizzare il computer per ottenere rapidamente, ma utilizzando principi matematici e logici rigorosi, risultati concreti: disegni, musica, poesie generate automaticamente. È un modo per dare ai bambini, e anche a chiunque altro, il controllo del computer. In quest'ambiente, il docente si trasforma in animatore della comunità, promotore di attività in cui i bambini progettano e imparano esplicitando e discutendo teorie sul mondo con cui interagiscono. La classe funziona come comunità di pratiche scientifiche in cui i bambini comunicano e condividono le loro idee, giuste o sbagliate che siano. Si discute ed ognuno apprende dall'altro. Le idee proposte possono essere valide, altre un po' meno, ma comunque tutti gli allievi partono da uno stesso piano: ogni idea ha la stessa dignità. Nelle didattiche proposte da Papert, ha grande importanza la gestione dell'errore: la sua idea è che l'unico modo per imparare in modo significativo sia quello di prendere coscienza dei propri errori. Compito dell'insegnante è quindi anche quello di guidare il bambino nel caso di errore.

Quello che ha valore in LOGO quindi, non è tanto il suo uso, oggi mediamente in calo, ma la sua filosofia di base, riassumibile in una serie di 7 principi che sostanzialmente costituiscono un vademecum dell'apprendimento costruzionista, come ama definirlo Papert:

1. Il primo principio è proprio il ruolo dello studente come protagonista che, come diceva il ricercatore americano, *[...] programma il computer e non si fa da lui programmare*. Una decina di anni dopo la creazione di Logo, questa fu definita come la "rivoluzione dell'informatica cognitiva" che rivedeva l'uso del computer allo scopo di incentivare e sostenere lo sviluppo delle abilità cognitive e metacognitive del soggetto, contrariamente alla sua classica idea di utilizzo. Prima di allora, infatti, il computer svolgeva il compito di insegnare esplicitamente l'informatica e di istruire alla programmazione.

2. Il secondo principio è rappresentato dalla cosiddetta *inversione epistemologica*, riassumibile quasi totalmente nella sostituzione dell'asserzione "imparare per usare" con "usare per imparare". Viene lasciato maggior spazio ai momenti di apprendimento rispetto a quelli di insegnamento che molto spesso si traducono in una sorta di mutuo insegnamento tra studenti e insegnante o addirittura tra pari, che interagiscono tra loro dandosi sostegno.

3. Il terzo principio contempla la rivalutazione del pensiero operatorio concreto su quello formale logico-deduttivo. Tradotto in un semplice esempio si può dire che anche le teorizzazioni più geniali nel campo scientifico, sono spesso nate da esperienze concrete, a volte fortuite.

4. Il quarto principio riguarda l'apprendimento sintonico e le sue caratteristiche evidenziate anche nel Logo con la concretezza e l'operosità che coinvolge profondamente il soggetto sia a livello corporeo che di identità.

5. Il quinto principio consiste nel mantenere sempre attiva e in costante allenamento la capacità di apprendimento dello studente, proponendogli con frequenza esercizi pratici per l'individuazione e la risoluzione di problemi. Questo si concretizza in progetti personali, carichi di significato per gli studenti, non imposti dall'alto ma eventualmente orientati dall'insegnante e senz'altro aggiustati in itinere dagli stessi allievi che, di fronte alle difficoltà emergenti, discuteranno e magari ridefiniranno il progetto iniziale, modularizzandolo, semplificandolo e variandolo.

6. Il sesto principio viene definito in gergo tecnico come l'epistemologia dell'indeterminazione gestita. Esso descrive il procedere verso il risultato finale attraverso un continuo confronto con il prodotto reale delle proprie operazioni, annettendo una riflessione sui processi che lo hanno realizzato.

7. Il settimo e utlimo principio è quella che Papert definì matetica ovvero l'imparare sbagliando e riflettendo sui propri errori. Questo atteggiamento porta ad un'apologia dell'errore che diventa pedagogia stessa dell'errore.

Il costruzionismo, quindi, è basato sulla teoria del costruttivismo secondo la quale l'individuo che apprende costruisce modelli mentali per comprendere il mondo intorno a lui e sostiene che l'apprendimento avviene in modo più efficiente se chi apprende è coinvolto nella produzione di oggetti tangibili. In questo senso è connesso all'apprendimento esperienziale e ad alcune teorie di Jean Piaget. Seymour Papert delinea il termine costruzionismo in un documento intitolato Constructionism: A New Opportunity for Elementary Science Education definendolo: Una parola che indica due aspetti della teoria della didattica delle scienze alla base di questo progetto. Dalle teorie costruttiviste in psicologia prendiamo la visione dell'apprendimento come una ricostruzione piuttosto che come una trasmissione di conoscenze. Successivamente estendiamo il concetto dei materiali manipolativi nell'idea che l'apprendimento è più efficiente quando è parte di un'attività come la costruzione di un prodotto significativo. I linguaggi di programmazione che andiamo a trattare sono basati sul lavoro di Papert e sono oltretutto diretti discendenti del linguaggio LOGO che Papert sviluppò. Vale la pena quindi aver investigato i principi fondamentali della teoria costruttivista. Continuiamo ora con l'approfondimento del concetto di micromondo. Derivato direttamente dalla dottrina costruttivista, è fortemente legato ai linguaggi di programmazione visuali didattici come LOGO e tutti quelli che direttamente o indirettamente sono stati influenzati dalla sua filosofia di programmazione.

2.2 Micromondo

Un micromondo è una ridotta ma completa versione di un qualche dominio di interesse, spesso modellato da un utente sulla base delle proprie conoscenze. L'esempio più conosciuto di strumento per la costruzione di micromondi è il Logo, linguaggio di programmazione che permette di modellare appunto,

tramite il computer, una varietà di domini, come la geometria e la fisica, ma anche la storia, la geografia, la meteorologia e l'insegnamento delle lingue. Il Logo nacque dalle idee di Daniel Gureasko Bobrow e Wally Feurzeig, che ne curarono la progettazione e la prima realizzazione verso la fine degli anni sessanta presso la Bolt, Beranek e Newman (BBN), nota società informatica di Cambridge (Massachusetts). Un apporto fondamentale per la sua crescita e diffusione internazionale in campo scolastico arrivò tuttavia da Seymour Papert, studioso di processi della conoscenza e pioniere dell'intelligenza artificiale, che si dedicò tra le altre cose alla descrizione dettagliata dei suoi possibili ambiti applicativi.

Ispirandosi ad alcuni principi della psicologia genetica di Jean Piaget, Papert sviluppò il Logo con l'obbiettivo principale di comprendere come fosse possibile utilizzare il computer per favorire, nei bambini, lo sviluppo della conoscenza e dell'intelligenza, in uno dei primi tentativi di conciliare psicopedagogia e informatica. Grazie ai suoi studi in materia il ricercatore sudafricano è tuttora riconosciuto internazionalmente come grande pensatore sui modi di cambiare l'apprendimento grazie all'utilizzo del computer.

Dapprima collega di Piaget all'università di Ginevra, fu in seguito a questa esperienza che Papert fece suo il modello piagettiano, secondo il quale i bambini sono fautori delle proprie strutture intellettuali. Successivamente fondò con Marvin Minsky al MIT di Boston (Massachusetts Institute of Tecnology) il laboratorio di intelligenza artificiale dove cominciò gli studi di ricerca che lo portarono a sviluppare il più noto linguaggio di programmazione a fini didattici, il Logo. Quest'ultimo è un linguaggio formale dai contenuti interattivi che prevede l'impiego di procedure in cui si includono principi euristici. Papert si dedicò alla sua realizzazione con il fine di ribaltare il concetto classico di programmazione. Se nei linguaggi tradizionali rigidi comandi costringono a seguire una semantica e una sintassi strutturate, grazie al Logo i bambini riconoscono nel computer uno strumento che permette loro di realizzare un apprendimento libero dalle ferree regole imposte dalla scuola. Così facendo sono i bambini a programmare i calcolatori e non viceversa, creando attivamente il proprio apprendimento.

Secondo gli studi di Papert l'apprendimento autoregolato ha tre caratteristiche principali:

1. il soggetto trova l'ambiente intrinsecamente motivante;
2. i soggetti che autoregolano l'apprendimento sono metacognitivamente attivi: essi intrattengono attività di pianificazione, si propongono degli

obiettivi e sono capaci di monitorare e valutare il loro operato;

3. tali soggetti sono attivi dal punto di vista comportamentale in quanto strutturano l'ambiente secondo le loro esigenze e motivazioni in modo tale che l'ambiente stesso risponda alle loro esigenze motivazionali ed al loro personale stile comportamentale;

inoltre questo metodo avrebbe portato numerosi risvolti positivi per la crescita cognitiva degli studenti che avrebbero imparato a:

- dividere qualsiasi problema in più sotto-problemi più semplici e quindi di più facile soluzione;
- presentare le proprie idee in modo chiaro e conciso;
- aiutare, apprendere e ascoltare gli uni dagli altri;
- essere critici in maniera costruttiva.

Introducendo una delle più originali idee da lui sviluppate, quella di micromondo, Papert dimostrò che l'ambizione di Logo non era quello di realizzare una rappresentazione accurata della realtà, ma di fare qualcosa che mostrasse e sviluppasse la qualità dell'immaginazione del bambino. Gli scopi del micromondo sono:

- mostrare il ponte che può essere costruito tra la conoscenza scientifica e di quella personale;
- fornire un esempio concreto di analisi di un determinato ambiente;
- comprendere le variabili che sono presenti nell'ambiente e le loro funzioni;
- fornire una conoscenza multidimensionale del problema.

Dunque iniziavano a delinearsi delle piccole aule virtuali dette micromondi, personalizzate da persona a persona, nelle quali non si cercava di istruire informatici più o meno in erba ma di utilizzare il computer e la programmazione come uno strumento potente per concepire ed esprimere progetti personali, carichi di significato e sviluppati all'interno di un dominio familiare all'utente, nel quale la conoscenza e l'esperienza più o meno ingenue elaborate dagli studenti rivoluzionarono l'uso del computer nella didattica, considerandolo uno strumento la cui gestione doveva essere completamente affidata all'iniziativa

personale, valorizzando l'autonomia operativa della persona e la pura reattività esecutiva della macchina. Proporre la programmazione informatica come strumento per apprendere ha inevitabilmente contribuito, a rivoluzionare il modo di concepire l'uso delle tecnologie informatiche nell'apprendimento e nella formazione.

Capitolo 3

Da Logo a Byob, passando per Scratch

Esiste un naturale sviluppo nel mondo dell'informatica e più in generale nell'ambito ingegneristico, secondo il quale ogni novità non è altro che un perfezionamento mirato a qualcosa che già in precedenza esisteva. Il concetto di evoluzione vale tanto in natura quanto per un programma e per le idee che ne compongono le fondamenta intellettuali. Così Logo, partorito dalla mente di Papert sulla base del concetto di costruzionismo da lui stesso introdotto, ha trovato in numerosi linguaggi sviluppati più recentemente, il progredire della dottrina che lo caratterizzava. Il riferimento è, in particolare, per due linguaggi molto simili tra loro, Scratch e Byob, che hanno ereditato il bagaglio culturale che Papert iniettò nella sua creatura che prese poi la forma di una tartaruga. Da notare che allo stesso modo il costruzionismo è a sua volta l'evoluzione del concetto filosofico di costruttivismo: anche in questo riconosciamo nuovamente il concetto di miglioramento ed evoluzione.

3.1 Logo

Logo è un linguaggio di programmazione nato verso la fine degli anni sessanta, realizzato presso i laboratori del MIT dal professor Seymour Papert. Pensato per soddisfare una concezione costruzionista di insegnamento, ha ereditato le tecniche di calcolo simbolico del Lisp, dal quale riprende parte della sintassi ed il modo di gestire le liste. Altra importante caratteristica è la possibilità di creare sottounità di programma come le procedure, ognuna delle quali svolge un compito specifico rendendolo così un linguaggio procedurale. Ogni procedura può a sua volta essere interpretata come una funzione in senso matematico, ulteriore prerogativa ereditata da Lisp che fa di Logo

un linguaggio funzionale.

In origine il Logo veniva utilizzato per muovere un apparato meccanico la cui corazza era simile a quella di una tartaruga, ma con lo sviluppo dei primi monitor di questo robot rimase solo il nome, col quale venne battezzato un piccolo cursore detto appunto tartaruga che popolava un micromondo a se stante così come lo aveva pensato Papert.

Grazie all'uso di questo particolare cursore che diventò anche una sorta di mascotte del programma, veniva proposta all'utente un particolare tipo di geometria conosciuta come geometria della tartaruga (turtle geometry), basata sul concetto di orientamento locale, differenziandosi da quella più comunemente utilizzata, basata sull'orientamento cartesiano. Concetti semplici e un metodo di programmazione intuitivo permisero differenti approcci alla programmazione, basati sulle caratteristiche personali di chi utilizzava il linguaggio offrendo così allo studente, la duplice modalità operatoria di un traduttore:

- quella che permette di impartire al computer comando dopo comando verificando, ad ogni passo la funzionalità dell'operazione, codificata ed immediatamente eseguita dall'elaboratore, attraverso l'output grafico, ed eventualmente di correggerla con un successivo comando di aggiustamento;

- la possibilità, una volta stabilita l'efficacia dell'intera sequenza di comandi, di passare in ambiente editor, dove compilare, in maniera il più possibile compatta e strutturata, l'intero programma (magari usando variabili al posto di dati) per poi, usciti dall'editor, mandarlo in esecuzione digitando semplicemente il suo nome, ed eventualmente i valori che si vogliono assegnare, di volta in volta, alle variabili incluse nel programma e dichiarate nel suo titolo (costruendo, ad esempio più quadrati dai lati di lunghezza diversa);

e nonostante fosse un linguaggio interpretato, quindi lento e inadatto per le applicazioni commerciali, già negli anni ottanta si era diffuso e veniva utilizzato a scopi didattici, spesso nei laboratori di geometria, stimolando chi lo usava da un punto di vista didattico. Logo è un linguaggio di programmazione multiparadigma fortemente orientato alla grafica e alla geometria di base. Tuttavia, essendo stato progettato come uno strumento per facilitare l'apprendimento i suoi ambiti applicativi sconfinano anche alla matematica più generale, passando per la musica, la robotica, le telecomunicazioni e la scienza. È spesso utilizzato per lo sviluppo di simulazioni, e per creare

rappresentazioni multimediali. Esiste inoltre una sua parte meno nota che include molti comandi per la gestione di input/output testuale e per l'elaborazione di dati (operatori di confronto, variabili, cicli, selezioni condizionali). Logo incoraggia altresì la programmazione modulare con uso intensivo di procedure, offrendo molta estensibilità per gli utenti più esperti.

Le sue principali caratteristiche sono:

- Interattività: Logo è un linguaggio interpretato, dunque può essere usato in modo interattivo. L'interattività di questo approccio fornisce all'utente un immediato feedback delle istruzioni individuali, aiutandolo nella correzione del programma e nel processo di apprendimento.

- Modularità ed Estensibilità: il programma Logo è costituito da un insieme di piccole procedure, definite attraverso la scrittura in un editor di testo. Ad esempio la parola to è seguita dal nome della procedura, mentre la parola end segnala la fine di tale procedura.

- Flessibilità: Logo lavora con liste e parole. Una parola in Logo è una stringa di caratteri. Una lista, invece, è una collezione ordinata di parole e liste. Anche i numeri in Logo sono parole.

In uso sin dagli anni sessanta, non viene tuttavia accantonato nonostante la sua età: nel marzo del 2009 esistevano ben 197 implementazioni e dialetti di Logo, ognuno con i suoi punti di forza. Esempi famosi di linguaggi pesantemente influenzati da Logo sono Smalltalk e una sua variante, Squeak, utilizzato per scrivere Etoys, un linguaggio pensato per l'educazione che ricalca i comportamenti e i tratti distintivi di Logo. In ultima analisi, citiamo i due linguaggi di programmazione argomento di questo elaborato, ereditieri della filosofia di programmazione costruttivista che accomuna tutti i linguaggi sopra citati.

3.2 Scratch

Scratch è un linguaggio di programmazione che si presenta come un ambiente di apprendimento grafico. Racchiude lo spirito costruzionista di Logo e di Etoys e il suo carattere di tipo visuale non ostacola l'utente con sintassi rigide, diminuendo così la possibilità di errori e dando una sensazione di immediatezza alla stesura del codice.

La sua prima versione fu sviluppata nel 2006 dal gruppo Lifelong Kindergarten capeggiato da Mitchel Resnik presso il Mit Media Lab a Boston. Venne creato per un segmento di pubblico giovane con lo scopo di favorire l'apprendimento grazie all'utilizzo di soluzioni intuitive e di un'interfaccia decisamente user friendly. Scratch consente infatti la costruzione di programmi e una loro successiva ed efficace verifica rendendo la programmazione facilmente assimilabile anche da utenti di giovane età. Il tutto è reso possibile grazie alla costruzione di codice tramite blocchi prefabbricati, abbinati ad un intelligente approccio del tipo *drag and drop* (clicca, trascina e rilascia). Ogni blocco agisce direttamente o indirettamente su attori detti *sprite*, rappresentati da piccoli personaggi la cui funzione è quella di eseguire le azioni previste dal codice all'interno del progetto. L'introduzione di questi personaggi interattivi, gli sprite, può essere interpretato come il tentativo di un avvicinamento da parte di Scratch al paradigma di programmazione orientato agli oggetti (object oriented).

Un obiettivo chiave di Scratch è quello di introdurre la programmazione a coloro che non hanno esperienze pregresse in questo campo. Questa è anche la motivazione di alcune peculiarità che caratterizzano il suo design: l'interfaccia utente vede lo schermo diviso in diversi riquadri: a sinistra si trova la tavolozza dei blocchi, al centro le informazioni relative allo sprite corrente e la zona script, sulla destra lo stage (ovvero il background) e la lista degli sprite. L'obiettivo è quello di lasciare a disposizione un numero minimo di blocchi non inficiando la possibilità di trattare una vasta gamma di progetti. Una strategia applicata è stata quella di raggruppare operazioni simili in singoli blocchi dotati di menù a tendina rendendo visibili le singole operazioni solo su richiesta. Con queste premesse, la prospettiva di un ambiente dotato di un'interfaccia a finestra singola, un set di comandi contenuto e una programmazione grafica a blocchi rendono questo linguaggio una scelta più appetibile ad un segmento giovane di utenti. Sempre più spesso infatti, Scratch viene usato dagli studenti dei college per corsi di introduzione all'informatica come, ad esempio, succede tutt'oggi ad Harvard. Il motto della comunità online di Scratch recita: Imagine, Program, Share. Il suo significato evidenzia la condivisione e la creatività come parti importanti della filosofia che muove gli utenti di Scratch. Per sollecitare il suo utilizzo e la condivisione dei progetti è stata creata una comunità online di Scratch, distribuita in molti paesi del mondo che contava, al dicembre 2011, più di 950000 utenti registrati.

La *palette dei blocchi*, o *tavolozza dei blocchi*, contiene pittogrammi, detti appunto blocchi. La copia di un blocco può essere trascinata sull'area script, giustapponendola ad altri per creare programmi. Per far si che la

tavolozza sia di dimensioni contenute, essa viene organizzata in otto insiemi, che raggruppano i blocchi a seconda delle funzioni che eseguono: movement (movimento), looks (sguardo), sound (suono), pen (penna), control (controllo), sensing (rilevamento), operators (operatori) e variables (variabili). Colori e forme diversi contraddistinguono differenti tipi di blocchi.

Nelle versioni 1.3.1 e inferiori, gli operatori si chiamavano numbers. Una caratteristica fondamentale di Scratch è il codice multi-thread con scambio di messaggi, ma la versione corrente non tratta le procedure come strutture di prima classe e non ha alcuna opzione di I/O per i file e supporta solo gli array monodimensionali a lunghezza arbitraria, noti come liste. Scalari in virgola mobile e stringhe sono supportati a partire dalla versione 1.4, ma con limitata capacità di manipolazione delle stringhe. Vi è un forte contrasto tra le potenti funzioni multimediali abbinato allo stile di programmazione multi-thread e la visibilità (scope) piuttosto limitata del linguaggio di programmazione Scratch.

Un certo numero di ambienti derivati da Scratch, detti Scratch modifications sono stati creati usando il codice sorgente della versione 1.4. Alcuni hanno modificato lo Scratch originale aggiungendo qualche blocco extra e cambiando l'interfaccia grafica mentre altri invece, non limitandosi agli aspetti superficiali, hanno introdotto cambiamenti più radicali. Uno di questi linguaggi è BYOB, dall'acronimo Build Your Own Blocks, che non solo consente all'utente di costruire i propri blocchi come suggerisce il nome, ma gestisce la possibilità di usare procedure, liste e sprite di prima classe. Quest'ultimi sono stati pensati e sviluppati, in questa evoluzione di Scratch, con un reale approccio object-oriented, e sono stati dotati di ereditarietà tramite prototipazione.

In Scratch non c'è distinzione tra compilazione, prova o stesura di un programma; esiste un unico stato all'interno del quale l'utente può scrivere codice trascinando e componendo blocchi che possono poi essere testati, cliccandovi sopra e ricevendo un riscontro grafico immediato dell'azione eseguita. Procedendo in questo modo vengono eliminate tutte le transizioni, forzando chi crea un programma ad essere in ogni momento attento e impegnato a scrivere codice e a testarlo per poi eventualmente modificarlo e correggerlo. Così facendo si stimola una approccio alla sviluppo di tipo bottom-up, organizzato in passaggi successivi, per raggiungere l'obiettivo finale passando per traguardi intermedi. A livello pratico il tutto si traduce nella composizione di frammenti di script testati singolarmente, successivamente combinati e assemblati tra loro in unità più grandi fino a raggiungere la soluzione del

problema. Questo metodo nasconde risvolti positivi come il fatto che l'utente scopre con facilità le funzioni di nuovi blocchi.

Scratch non ha messaggi di errore. Gli errori di sintassi sono stati eliminati in quanto i blocchi sono componibili tra loro solo in modi che possano avere senso. La loro forma infatti, suggerisce il corretto aggancio con altri blocchi non lasciando spazio a quelli che in un linguaggio di programmazione testuale sarebbero errori di sintassi. I creatori di Scratch si sono ingegnati anche per eliminare gli errori in fase di esecuzione (run time) rendendo tutti i blocchi sensibili ai valori passati in ingresso ma, nonostante lo sforzo, non è stato possibile eliminare del tutto ogni errore e deve essere il programmatore a premurarsi di scrivere script che eseguano ciò che lui vuole. L'aggiunta di piccoli ma non indifferenti dettagli nel design (flashing dei blocchi quando testati, degli oggetti quando acceduti, ...)fanno una grossa differenza per la comprensione del codice scritto, riducendo al minimo al possibilità di errori.

Gli script in Scratch vengono costruiti componendo più blocchi. La forma di quest'ultimi suggerisce come debbano essere agganciati e il sistema drag and drop rifiuta automaticamente connessioni prive di significato. In Scratch la grammatica visiva dei blocchi gioca il ruolo delle regole di sintassi, più comunemente usate negli altri linguaggi di programmazione.

Esistono 4 tipi di blocchi in Scratch:

1. blocchi comando (command blocks): rappresentano l'equivalente delle espressioni, condizionali e non, dei linguaggi testuali classici. Composti tra loro, utilizzando la forma ad incastro tramite il drag and drop, formano sequenze di comandi detti stack.

Figura 3.1: Esempio di blocco comando

I blocchi comando presentano un incavo nella parte superiore e una protuberanza simmetrica nella parte inferiore. Possono essere usati per creare sequenze di comandi dette stack (pile).

2. blocchi funzione (function blocks): sono da considerare come degli operatori. Questo tipo di blocchi non sono adoperati in sequenze lineari di codice, in cui blocchi successivi vengono eseguiti sequenzialmente,

come i blocchi comando, bensì vengono usati come argomenti da passare ai blocchi comando oppure vengono annidati tra loro per costruire espressioni.

Figura 3.2: Esempio di blocco funzione

I blocchi funzione ritornano un valore. La loro forma è tondeggiante ai lati e non presenta incastri a differenza degli altri blocchi.

3. blocchi per la gestione di eventi (trigger blocks): collegano gli eventi, quali la pressione di un tasto o il clic del mouse, con le operazioni che li gestiscono. Alcuni di questi blocchi presentano uno spazio per inserire parametri incorporato che può accogliere numeri, stringhe, valori booleani e in alcuni frangenti anche blocchi funzione. Un caso particolare è quello in cui uno stack di codice inizia con un blocco di questo tipo avente come argomento una bandiera verde, esso verrà eseguito quando l'utente cliccherà il pulsante start.

Figura 3.3: Esempio di blocco per la gestione di eventi

I blocchi per la gestione degli eventi presentano una caratteristica forma arrotondata nella parte superiore. L'esecuzione dei blocchi ad essi sottostanti comincia nel caso l'evento descritto abbia luogo.

4. blocchi per le strutture di controllo (control structure blocks): sono una specie di blocchi comando con la differenze che possono contenere una o più sequenze di comandi nidificate. La forma e l'uso intuitivi di questi blocchi li rende l'ideale per l'introduzione dell'informatica e dei suoi concetti base. Rappresentano molti dei costrutti condizionali tipici dei linguaggi di programmazione testuali (cicli if, while, for, ...).

Figura 3.4: Esempio di blocco per le strutture di controllo

I blocchi per le strutture di controllo presentano delle aperture atte ad accogliere sequenze nidificate di comandi.

Scratch ha 3 tipi di dato di prima classe: booleani, numerici e stringhe. Sono quindi gli unici 3 tipi di dato che possono essere usati in un'espressione, salvati in una variabile o restituiti da un blocco reporter di base. In quanto linguaggio visuale, Scratch suggerisce il tipo di parametro che si aspetta di ricevere con la forma dello slot dove andrà a incastrarsi l'eventuale blocco parametro e il tipo di dato che restituirà un blocco reporter con la forma del blocco reporter stesso. Nel primo caso Scratch mette a disposizione tre forme differenti per i tre tipi di dato differenti ovvero booleani, numerici e stringhe. Nel secondo caso invece, un blocco funzione può essere di sole due forme. Questa è la conseguenza della non tipizzazione di Scratch, che raggruppa assieme stringhe e valori numerici accettandoli indistintamente come se fossero di un unico tipo.

Gli sprite possono essere considerati oggetti a tutti gli effetti. Essi incapsulano variabili di stato e hanno comportamenti che li caratterizzano, gli script. Tuttavia non esistono i concetti di classe o ereditarietà, prerogative essenziali per considerare un linguaggio come orientato agli oggetti (object oriented). E' per questo che Scratch è considerato un linguaggio non orientato bensì basato sugli oggetti (object based). Ogni comando opera solamente sullo sprite nel quale appare; in altri termini, il ricettore implicito di ogni comando è sempre lo sprite che lo invoca. Questo fa sì che ogni sprite abbia un insieme di script personale, portando con sé tutti i vantaggi e gli svantaggi del caso. Da un lato c'è la facilità di comprensione: gli script presenti all'interno di uno sprite descrivono completamente il suo comportamento, senza equivoci, e l'utente non deve ripercorrere la catena di prototipazione per risalire allo script ereditato. Inoltre i cambiamenti sul codice sono localizzati e così da non avere ripercussioni su altri sprite. Per contro, in assenza di classi o di altri meccanismi che permettano la condivisione di codice tra più entità, è richiesto più lavoro per gestire più sprite aventi gli stessi comportamenti e quindi gli stessi script. In questi casi è richiesto che l'utente crei un singolo sprite che contenga lo script desiderato e che successivamente venga clonato con lo strumento timbro (stamp) un numero arbitrario di volte.

Come già detto sopra, uno sprite non può invocare direttamente lo script di un altro sprite. Tuttavia Scratch supporta un meccanismo per ovviare al problema della comunicazione e sincronizzazione tra sprite. Ognuno di questi, infatti, è in grado di trasmettere in broadcast un messaggio; ciò significa

che tale messaggio verrà ricevuto da ogni altro sprite, il quale gestirà l'evento a seconda di come è stato programmato. Per sua natura, il modello per la trasmissione in broadcast di Scratch è uno-a-molti, in quanto un messaggio lanciato da uno sprite può essere ricevuto da più sprite ed attivare quindi più script; è debolmente accoppiato (loosely-coupled) perché chi invia il messaggio non si cura di quanti saranno a riceverlo; ed è infine asincrono in quanto il comando di broadcast non attende che gli script che lo hanno ricevuto e gestito vengano terminati.

Si ottengono due vantaggi non permettendo agli sprite di controllarsi tra di loro direttamente. Per primo, quando si cerca di capire il comportamento di uno sprite è sufficiente andare a cercare negli script presenti all'interno dello sprite stesso, è facilitata quindi la comprensione del codice. In secondo luogo e non meno importante c'è la capacità degli sprite di essere autosufficienti con il vantaggio di poter essere spostati da un progetto all'altro senza il pericolo di rompere dipendenze, incentivando la condivisione degli sprite e di conseguenza anche il riutilizzo di codice già scritto e le collaborazioni tra utenti.

Sebbene siano stati fatti dei tentativi per introdurre l'uso di procedure nelle più recenti versioni di Scratch, nell'interesse della semplicità e del minimalismo che fino ad oggi lo hanno contraddistinto, le procedure sono state rimosse in quanto la loro somiglianza con i messaggi di broadcast creava confusione durante il loro utilizzo. Eppure l'astrazione procedurale è una delle idee forti dell'informatica e le procedure hanno valore pratico per come strutturare il codice durante crescita di un progetto. Il team di Scratch sta valutando la possibilità di reintrodurre le procedure come idea per dare modo per gli utenti di definire dei blocchi di script personalizzati. Altri ricercatori hanno già sviluppato una variante di Scratch, Byob, che non supporta solo le procedure ma anche funzioni di prima classe, chiusure e un sistema completo di programmazione funzionale.

La capacità di Scratch di riuscire a far agire più sprite alla volta, eseguendo più script nello stesso momento, gli ha conferito la reputazione di linguaggio multi-threading. Nonostante ciò lo affligge la lacuna per la mancanza di espliciti meccanismi di controllo della concorrenza che spesso si trovano in altri linguaggi di programmazione, come ad esempio i semafori o i monitor. Tuttavia Scratch costruisce il controllo della concorrenza attorno al suo modello multi-threading in modo da evitare la maggior parte dei casi di concorrenza facendo in modo che l'utente non debba preoccuparsi della questione. Ciò è reso possibile vincolando le situazioni nelle quali lo può verificarsi il cambiamento di stato di un thread. In molti modelli pre-emptive ciò

può verificarsi tra due istruzioni qualunque mentre in Scratch può avvenire solo in due situazioni: in un comando che attende esplicitamente (ad esempio il comando wait) o alla fine di un ciclo. Il cambiamento non può verificarsi nel mezzo di un'espressione di non-attesa, o tra il test di un costrutto if e il suo corpo. Il modello di threading Scratch consente agli utenti di ragionare su uno script come se fosse isolato, dando poca o nessuna considerazione agli effetti collaterali al sopraggiungere dell'esecuzione di altri script. Sebbene il modello threading di Scratch eviti la maggior parte delle condizioni di concorrenza, non si può dire che tutti i problemi di questo genere siano stati eliminati. Il più comune che si pone in Scratch è quando più script vengono attivati da un evento o da una trasmissione in broadcast e l'ordinamento di questi script non è quello che l'utente si aspetta.

3.3 Byob

Byob è una cosiddetta *Scratch modifications* che si pone come obiettivo principale quello di colmare le lacune di Scratch introducendo la possibilità di creare blocchi personalizzati e con essa la libertà di scrivere funzioni e procedure, anche ricorsive, similmente ai più comuni linguaggi testuali. Un ulteriore motivazione al suo studio arriva dall'ampia scelta dei dati di prima classe che genera, procedendo con un'analisi più attenta, risvolti non del tutto intuitivi e premeditati, che verranno spiegati in seguito nel dettaglio.

Byob è stato sviluppato da Jens Mönig con documentazione fornita da Brian Harvey, presso l'università della California a Berkeley. La principale novità che vanta questo programma è anche la stessa che lo descrive con l'acronimo che rappresenta il suo nome: *Build Your Own Blocks* ovvero *costruisci i tuoi blocchi*. Attualmente, l'ultima versione rilasciata di BYOB è la 3.1.1, rilasciata il 19 maggio 2011. Gli sviluppatori stanno tuttavia 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.

3.4 Snap!

Ad oggi le due personalità a cui è attribuita la creazione di Byob sono impegnate nello sviluppo di un nuovo progetto che è inteso essere l'evoluzione di Byob e prende il nome di *Snap!*. A livello di contenuti, per ora, non si distacca troppo dal suo genitore Byob; ma ciò che più lo distingue tra i suoi predecessori è la spiccata portabilità. Snap! è infatti stato pensato per es-

sere *web based* e quindi facilmente utilizzabile dal browser senza necessità di previa installazione. Questa caratteristica porta vantaggi non indifferenti dal punto di vista dell'insegnamento dell'informatica nelle scuole e nell'istruzione in generale, soverchiando i problemi legati alla località di una particolare macchina o quelli indotti dalla limitata disponibilità di risorse.

L'intera piattaforma è attualmente disponibile in una versione alpha di prova al URL:

http://snap.berkeley.edu/run

o alternativamente:

http://snap.berkeley.edu/snapsource/snap.html

È possibile inoltre sfruttare una versione con dei tool già precaricati all'URL:

http://snap.berkeley.edu/init

Eseguendo un accesso è facile cogliere, anche a prima vista, il forte legame che lega Snap! a Byob e a Scratch prima di lui. L'interfaccia utente grafica è stata modificata e resa più moderna e minimale mentre la disposizione delle tavolozze e dei comandi principali non ha subito stravolgimenti sostanziali.
La mascotte simbolo di Byob è stata sostituita dal classico cursore a forma di freccia, familiare agli utenti di Logo. Utilizzando la versione ampliata, reperibile al terzo dei link sopra elencati, si può disporre di blocchi aggiuntivi per eseguire compiti che spesso ricorrono durante la programmazione come mappare degli ingressi in un blocco, ecc.
La versione 4.0 di Snap! sarà scritta in JavaScript, utilizzando i canvas element di HTML5 e serà anche una libreria Morphic, creata dallo stesso Jens Mönig, chiamata Morphic.js.

Una prima panoramica di Snap! enfatizza ancora di più quei tratti che già rendevano Byob un programma destinato all'utilizzo da parte di utenti esperti, a differenza di Scratch. Infatti, indagando oltre i semplici e intuitivi script, facilmente componibili tramite un altrettanto intuitiva interfaccia, si possono scoprire risvolti teorici e pratici che includono la programmazione ad oggetti e le istanze, i meccanismi di ereditarietà degli sprite e gli immancabili dati di prima classe che sono tuttora il cavallo di battaglia di Byob e che ci accingiamo a descrivere nel dettaglio nel prossimo capitolo.

Capitolo 4

Byob

4.1 Blocchi semplici

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

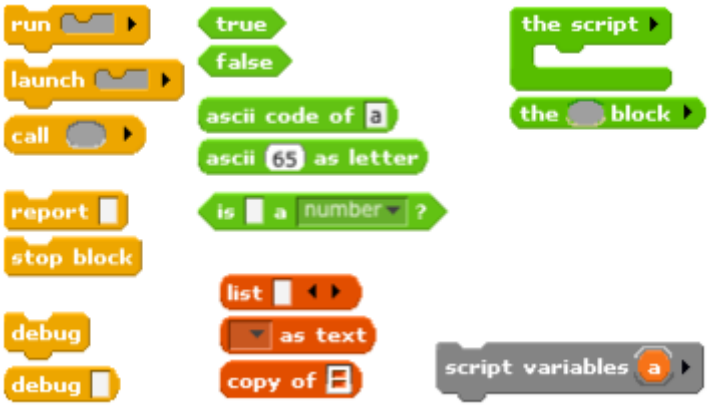


Figura 4.1: I nuovi blocchi introdotti da Byob

In basso alla tavolozza, o palette, delle variabili si trova il bottone per costruire un nuovo blocco personalizzato, etichettato con la scritta Make a block.

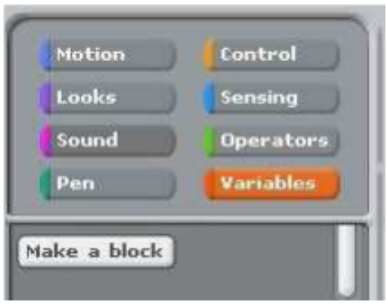


Figura 4.2: Le tavolozze di Byob

Cliccando il pulsante comparirà una finestra nella quale verranno richieste le caratteristiche del nuovo blocco che si andrà a creare, come il suo nome, la sua forma e il suo colore. Quest'ultimo non avrà alcuna influenza sul blocco creato se non quella di posizionarlo nella relativa tavolozza in base al colore scelto. In fase di creazione verrà inoltre deciso se il nuovo blocco sarà avviabile per tutti gli sprite o solo per un determinato sprite e per i suoi figli.

Alternativamente è possibile far comparire la finestra di dialogo per creare un nuovo blocco cliccando col tasto destro del mouse in piattaforma windows/linux oppure cliccando il tasto di controllo più mouse in piattaforma MacOs e selezionando dal menù a tendina la voce relativa Make a block.

Figura 4.3: La finestra di dialogo Make a block

Ogni palette è contraddistinta da un particolare colore eccezion fatta per la tavolozza delle variabili che, contrassegnata dal colore arancione, include tuttavia anche i blocchi rossi relativi alle liste e quelli grigi contrassegnati in fase di creazione come Others, di altro genere.

Viene data la possibilità di scegliere tra 3 forme differenti per i blocchi che si vanno a creare, seguendo una convenzione che dovrebbe essere familiare agli utenti di Scratch:

- i blocchi di tipo command, con una forma incavata, non restituiscono alcun valore, similmente ai metodi void in Java;
- i blocchi di tipo reporter, con una forma ovale, che restituiscono un valore;
- i blocchi di tipo predicate, con una forma esagonale, che non sono altro che un caso particolare di blocco reporter con unica peculiarità quella di ritornare un valore di tipo booleano.

Una volta scelta la definizione del nuovo blocco si accede ad un editor di blocchi nel quale è possibile sviluppare il suo comportamento componendo blocchi, trascinandoli dalla tavolozza e rilasciandoli nell'editor. Ogni blocco aggiunto va attaccato a quello già presente a forma di cappello, detto prototipo del nuovo blocco. Il richiamo con Scratch è forte: blocchi con questa forma infatti, servivano a gestire eventi mentre adesso contengono il nome del blocco che sta per venire creato e rappresentano in un certo senso la sua definizione.

Nella parte alta di questa finestra c'è una checkbox etichettata con la parola atomic. Se spuntata l'intero script sottostante verrà eseguito da Byob in un solo ciclo mentre in caso contrario verrà messa in evidenza l'atomicità dei singoli blocchi che formano lo script con un'esecuzione passo-passo. Normalmente l'atomicità sarà attiva per i blocchi reporter e disattiva per i blocchi comando.

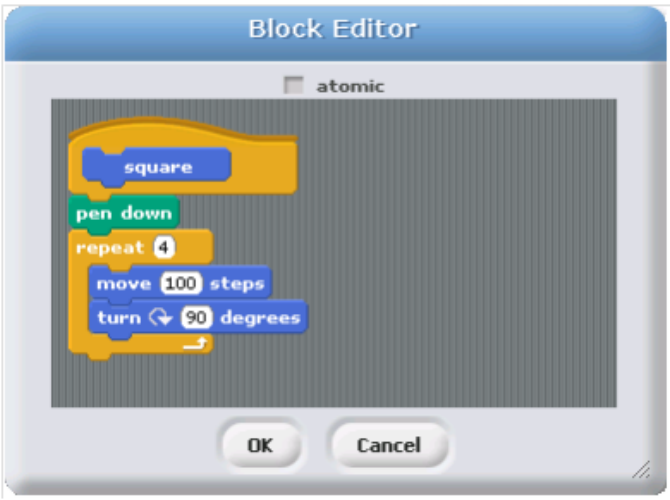


Figura 4.4: L'editor dei blocchi personalizzati

Spostando il mouse sopra il prototipo del nuovo blocco si vedono comparire due piccoli pulsanti tondi etichettati con un + ai lati del nome.

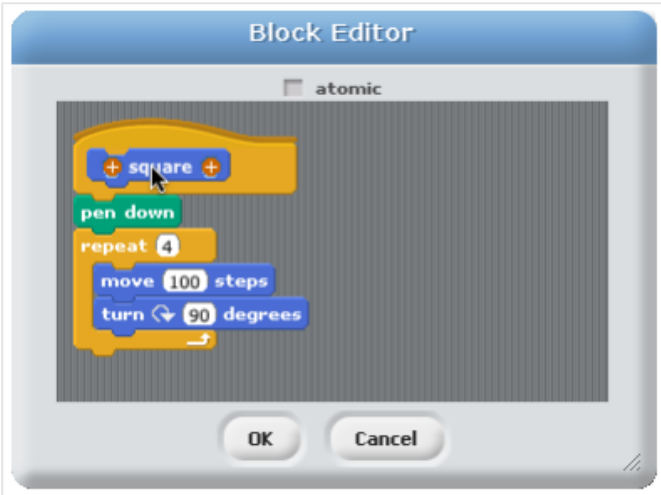


Figura 4.5: Pulsanti + per l'aggiunta di ingressi

Cliccando sopra a uno dei due comparirà la finestra per l'inserimento di parametri in ingresso. La posizione del pulsante + cliccato (destra o sinistra) discriminerà solamente la posizione che assumerà l'ingresso nel nome finale del blocco, a livello operativo non ci sarà alcun effetto sull'esecuzione dello script. Questa pratica serve solamente a mantenere una notazione tipica di Scratch nella quale i nomi degli script potevano essere anche intere frasi,

composte da parole e ingressi, che aiutavano il processo, per lo più mnemonico, di risalire al comportamento dello script stesso. Come ad esempio succede nel blocco move ( ) steps per il quale è facile intuire il risultato dopo un'esecuzione.

Figura 4.6: La finestra di dialogo per aggiungere un nuovo ingresso

Le scelte che ci si pongono sono due: Title text che, come è stato appena spiegato permette l'inserimento di altro testo da aggiungere al nome, oppure Input name per aggiungere un ingresso. Mantenendo la seconda selezione comparirà sulla destra una freccetta nera. Cliccandovi sopra è possibile ampliare la scelta del tipo di ingresso con opzioni extra che verranno illustrate nel dettaglio successivamente. Componendo il metodo di aggiunta di testo e ingressi con i pulsanti + è possibile una composizione libera di nomi misti ad ingressi come precedentemente spiegato.

Una volta dato l'OK sulla finestra di aggiunta dell'ingresso vedremo comparire un nuovo input nel prototipo del blocco.

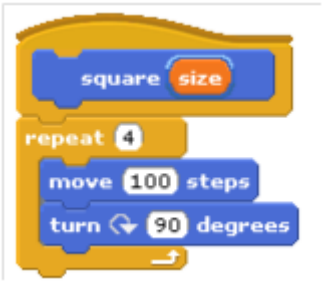


Figura 4.7: L'ingresso è stato inserito...

Ora è possibile trascinare la nuova variabile nello script per poterla utilizzare.

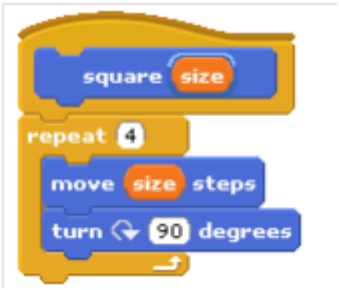


Figura 4.8: ...e può essere riutilizzato nella costruzione del blocco

Dando nuovamente l'OK il blocco appena creato comparirà nella tavolozza per la quale inizialmente si era scelta la destinazione (in base al colore), con la possibilità di utilizzarlo in altri script o semplicemente testandolo col doppio clic.

4.2 Ricorsione

Nonostante Papert sosteneva la ricorsione come una delle più grandi idee matematiche che i bambini dovrebbero apprendere per imparare a programmare, nelle prime versioni di Scratch, fino alla release 1.4 era fornita solo una limitata forma di ricorsione, detta ricorsione in coda (tail recursion), nella quale la chiamata ricorsiva risultava essere l'ultima azione eseguita dallo script. Ciò era possibile attraverso l'uso del blocco broadcast, programmandolo a chiamare lo script in cui si trovava.

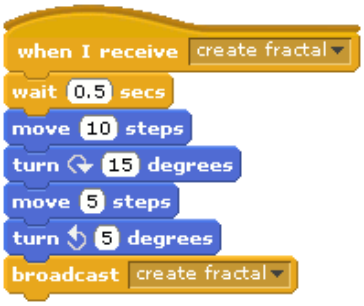


Figura 4.9: Tentativo di ricorsione in Scratch

questo creava una forma di ricorsione in coda che tuttavia non teneva traccia della elaborazione dei dati chiamata dopo chiamata. Infatti quando Scratch riceve un messaggio che attiva uno script già in esecuzione, dimentica dov'era, anziché salvare il suo stato, ricominciando l'esecuzione dello script stesso dall'inizio. Per far si che lo script continui un processo dopo una chiamata ricorsiva in coda, era necessario implementare un altro script che venisse lanciato, come un thread separato, dallo script originale.

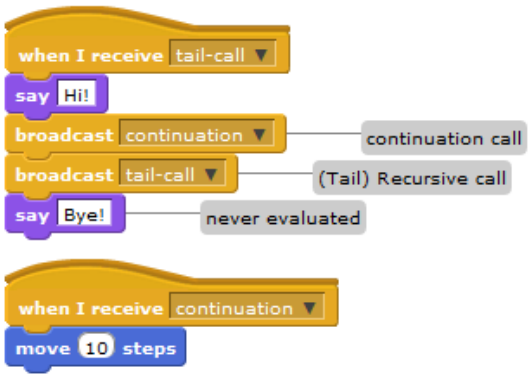


Figura 4.10: Struttura della ricorsione in coda

Le cose migliorarono in Scratch 2.0, dov'era stata aggiunta la possibilità di utilizzare le potenzialità delle pile attraverso dei blocchi stack personalizzati, con parametri in ingresso che introducevano intrinsecamente l'opportunità di creare ricorsioni. La mancanza di reporter personalizzati poteva essere superata attraverso l'uso di variabili globali per immagazzinare i dati temporanei.

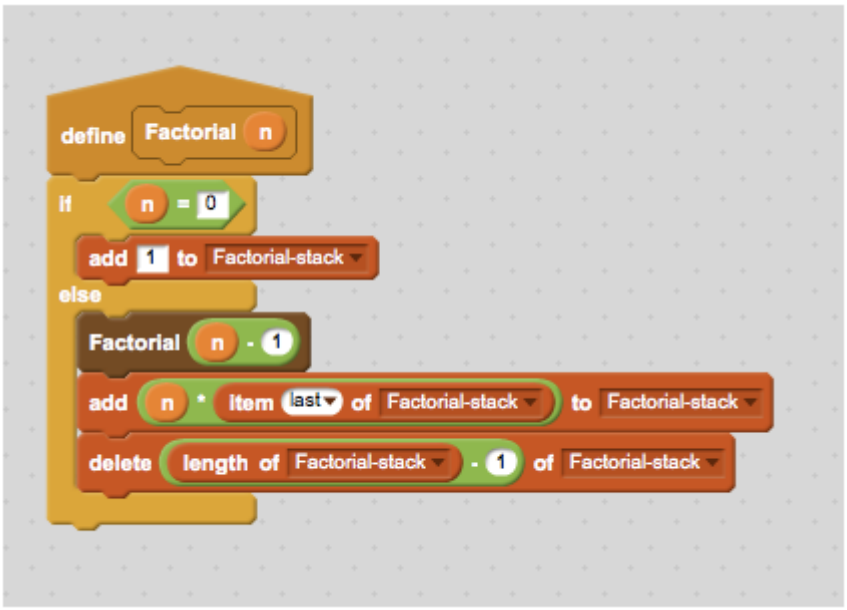


Figura 4.11: Utilizzo di uno stack

La vera rivoluzione, che colmò questa lacuna di Scratch, arrivò con Byob, dove era stata implementata la possibilità di ricorsione, non necessariamente in coda, grazie all'uso delle procedure.

Nel momento in cui un nuovo blocco viene creato tramite il procedimento di cui sopra, esso sarà disponibile per l'utilizzo. Andando a modificarne il comportamento tramite l'editor dei blocchi sarà possibile creare ricorsioni facendo si che il blocco richiami se stesso all'interno del suo script. Un semplice esempio di ricorsione può essere quello di un blocco che esegue il fattoriale di un numero:

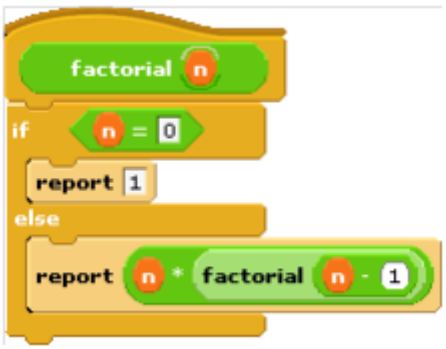


Figura 4.12: Esempio di ricorsione in Byob: calcolo del fattoriale

Figura 4.13: Uso del blocco per il calcolo del fattoriale

Questo esempio introduce anche la questione sull'utilizzo del blocco report. Mentre la ricorsione in un blocco di comando non necessita di molte spiegazioni se non dell'intuizione del programmatore, c'è da sottolineare che nei blocchi reporter, l'uso del blocco report segue una regola fondamentale per la quale una volta giunta la sua esecuzione, l'intero script viene bloccato e termina il suo lavoro ritornando il valore al suo ingresso; eventuali altre istruzioni non vengono valutate.

Capitolo 5

Liste di prima classe

L'espressione dati di prima classe fu coniato dall'informatico Christopher Strachey nel 1960, il quale sosteneva che qualsiasi tipo di dati in un linguaggio di programmazione dovrebbe essere di prima classe. Ciò significa che dati di questo tipo:

- possono essere il valore di una variabile
- possono essere l' ingresso di una procedura (di un blocco in Scratch)
- possono essere l' uscita di una procedura
- possono essere parte di un insieme di dati (liste in Scratch)
- possono esistere senza avere un nome (dati anonimi)

Nella versione 1.4 di Scratch gli unici dati ad essere di prima classe erano i numerici e le stringhe di testo, mentre le liste non lo erano. Questo precludeva la possibilità di sviluppare strutture dati complesse come gli alberi che si basano sul concetto di liste di liste.

Un principio fondamentale alla base di BYOB è che tutti i dati devono essere di prima classe. Se un 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.

5.1 Il blocco lista

La possibilità di fornire liste di prima classe in Byob è dovuta alla capacità di creare liste anonime ovvero senza la necessità di dare loro un nome. Risulta

inoltre molto facile aumentare e diminuire il numero di elementi della lista adoperando le apposite freccette nere che si trovano nella parte destra di ogni blocco lista.

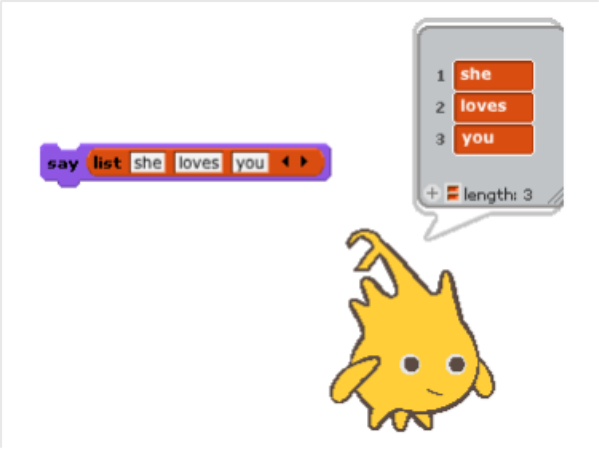


Figura 5.1: Liste in Byob

E' doveroso un confronto con Scratch dove ciò non accade: per creare una lista occorre richiamare lo specifico comando Make a list, agendo sulla finestra di dialogo e successivamente utilizzare l'apposito blocco per popolarla.

Figura 5.2: Modello di lista presente anche in Scratch

Torniamo a Byob. In quanto dati di prima classe, le liste devono poter essere usate come ingressi per altri blocchi. Il che è possibile dal momento che sono formate da blocchi di tipo reporter, a differenza delle liste di Scratch che, costituite da blocchi di tipo command, non sono utilizzabili per gli input di altri blocchi.

Figura 5.3: Il tipo di dato lista

Esistono tuttavia anche delle limitazioni a questo nuovo tipo di dati introdotto da Byob: è possibile infatti inserire elementi nella lista solo manualmente. Si è così privati dei più comuni metodi di popolamento utilizzati in molti altri linguaggi di programmazione, che si basano sulla valutazione di condizioni specifiche.

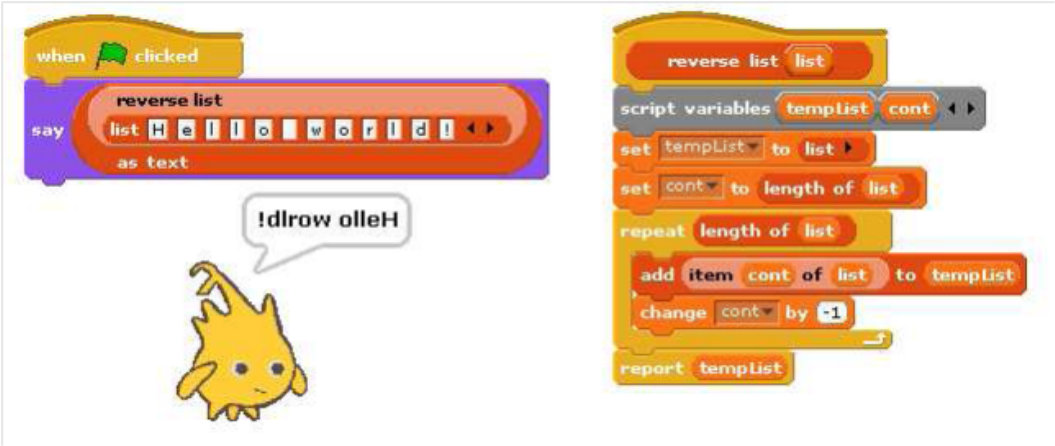


Figura 5.4: Esempio di utilizzo delle liste in Byob

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.

5.2 Liste di Liste

Sfruttando le liste di prima classe in Byob, è possibile creare liste di liste e di conseguenza dare forma a strutture dati complesse formate come gli alberi, tabelle hash e grafi, definendone metodi modificatori e costruttori (blocchi e script nel caso di Byob).

Analizziamo, per meglio comprendere l'argomento, lo sviluppo di un albero binario per il quale è descritto uno script che ne valuti il tipo.

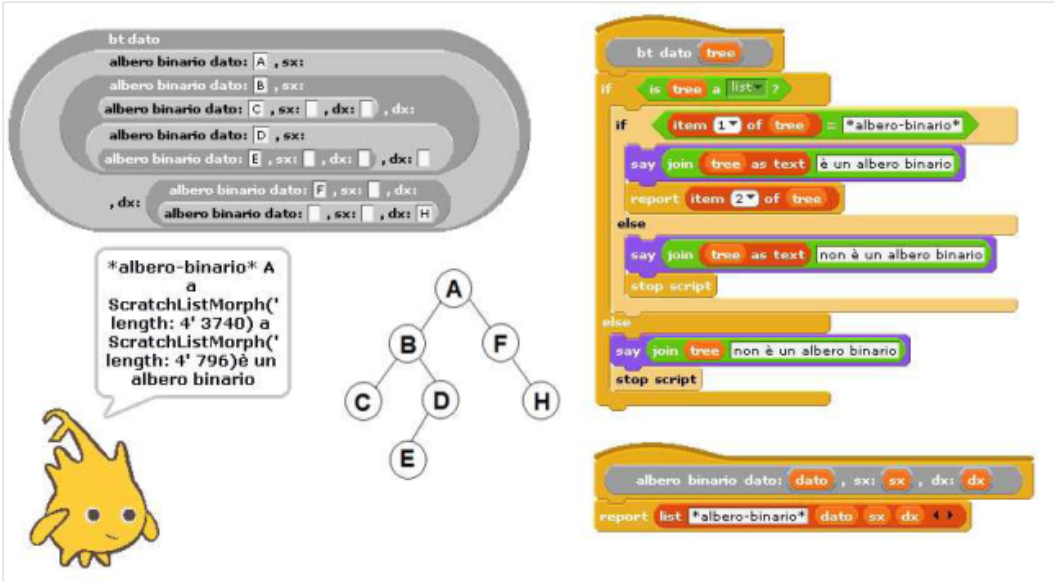


Figura 5.5: Implementazione di un albero binario in Byob

Capitolo 6

Ingressi tipizzati

Gli ingressi dei blocchi in Scratch possono essere di tre tipi:

1. numerico/testuale: inseribili in slot ingressi rettangolari;
2. numerico: inseribili in slot ingressi ovali;
3. booleano: caso particolare, può essere usato solo in certi blocchi con slot ingressi esagonali.

In Byob i dati di prima classe sono molti di più, si utilizzano quindi ingressi di molti più tipi, che comprendono anche procedure, liste e oggetti.

Richiamiamo ora la descrizione in dettaglio delle opzioni extra per la scelta dell'input, che avevamo rimandato dalla sezione sulla costruzione di un nuovo blocco. Una volta cliccato un pulsante + nel prototipo del nuovo blocco verrà visualizzata una finestra, nella quale, se viene selezionata l'opzione Input name, si passa all'inserimento di un nuovo parametro in ingresso.

Figura 6.1: Finestra di dialogo per l'inserimento di un nuovo ingresso

Cliccando sulla freccia nera che compare nel caso venga mantenuta questa selezione si passa alla finestra per la definizione del tipo di ingresso:

Figura 6.2: Finestra di dialogo estesa per l'inserimento di un nuovo ingresso

Dall'immagine si evince che in Byob si può scegliere tra 12 tipi di ingresso differenti, ognuno dei quali discriminato secondo 3 diverse categorie, mutualmente esclusive tra loro: Single input, Multiple input e Make internal variable visible to caller , di seguito brevemente descritte.

- Single input: tutti gli ingressi in Scratch appartengono a questa categoria, se il tipo di dato appartiene a una delle categorie Text, Number o Any è data la possibilità di scegliere un valore di default che verrà mostrato nello slot di input durante l'utilizzo;

- Multiple input: introducendo le liste di prima classe si è concretizzata la possibilità di avere più di una variabile in ingresso per i blocchi. In questo caso infatti gli input sono mantenuti all'interno di una lista espandibile o comprimibile a piacere tramite i pulsanti freccia. Trovandosi all'interno del block editor la presenza di un ingresso multiplo viene segnalata con l'aggiunta di un suffisso al nome del parametro in ingresso, costituito da 3 punti di sospensione.

- Make internal variable visible to caller: non rappresenta realmente 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. Analogamente al caso precedente, la presenza di un ingresso di questo tipo viene segnalata con l'aggiunta di un suffisso al nome del parametro in ingresso, costituito da una freccetta bianca che punta verso l'alto.

Nel caso non venga aperta la finestra di dialogo con le opzioni avanzate per la scelta degli ingressi, la combinazione adottata da Byob di defualt è quella con ingresso singolo e tipo di dato qualsiasi - any.

Sebbene ad un primo sguardo i 12 tipi di input possano sembrare un'accozzaglia di opzioni elencate senza alcun criterio, la loro collocazione nella griglia non è affatto casuale e interpretandone l'ordine risulta più facile trovare il tipo di ingresso che fa al proprio caso:

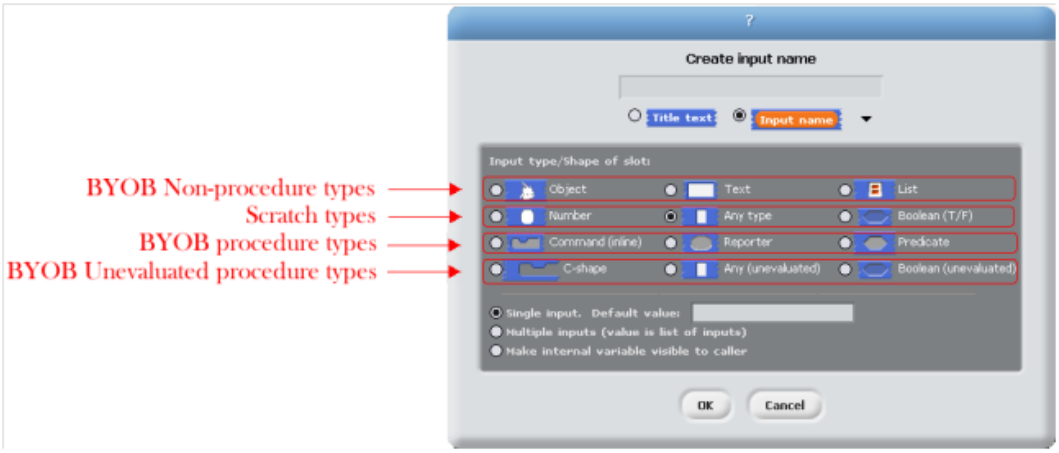


Figura 6.3: Tipi di ingressi in Byob, suddivisione per righe

Procedendo dall'alto verso il basso si riconoscono nella prima riga i tipi di dati di Byob che non riguardano procedure ovvero Object, Text e List. Nella seconda riga ritroviamo i tipi di dati caratteristici di Scratch. La terza e la quarta riga contengono invece dati inerenti le procedure, che verranno discusse più approfonditamente nel seguito. La scelta di un ingresso appartenente alla terza riga è riscontrabile al di fuori dell'editor di blocchi per il colore grigio dello slot di input; questo permette la distinzione con ingressi dalla forma simile appartenenti alla seconda riga che differiscono, appunto, per il colore dello slot. La scelta di un ingresso non valutato - unevaluated, che faccia parte quindi della quarta riga, è riscontrabile nell'editor dei blocchi

con l'aggiunta di un suffisso al nome del parametro in ingresso che si è andati ad aggiungere, costituito da un asterisco.

Come abbiamo già accennato, il tipo di dato List è necessario per usare le liste come dati di prima classe e analogamente il tipo Object si usa per usare gli sprite come dati di prima classe; l'icona ne richiama il concetto. Il tipo di dato Text, rappresentato con un'icona che richiama la forma di un campo di testo, è solo una variante del tipo Any.

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).

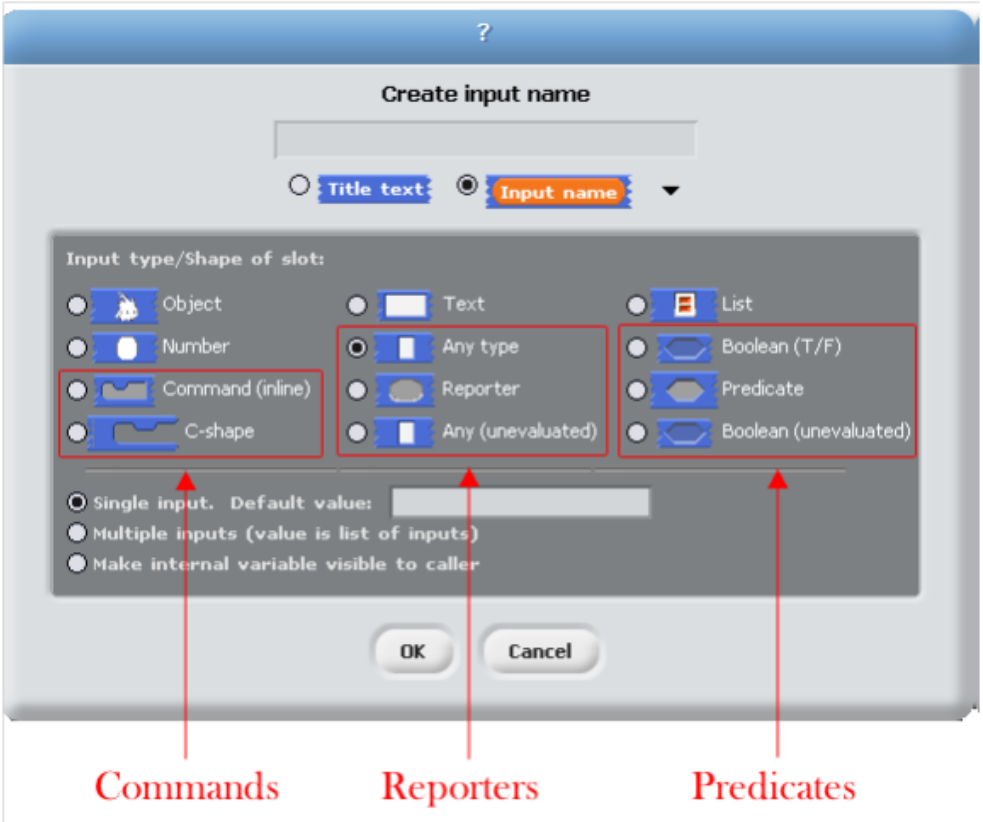


Figura 6.4: Tipi di ingressi in Byob, suddivisione per colonne

Capitolo 7

Procedure come dati

7.1 Tipi di ingresso

Il supporto per usare le procedure come dati di prima classe in Byob è fornito da due blocchi molto simili tra loro nell'utilizzo: il blocco call e il blocco run. Entrambi servono a richiamare altri blocchi con la differenza che il primo dei due accetta in ingresso uno o più blocchi, di tipo reporter o predicate e, dopo averne eseguito le istruzioni, ne restituisce il risultato che è d'obbligo. Il secondo svolge una funzione analoga per i blocchi di tipo command e di conseguenza non restituisce alcun valore al termine. Segue un esempio chiarificatore:

Figura 7.1: Esempio di uso del blocco run

Nell'esempio l'utilizzo del blocco run. Viene creato il blocco for che non esiste in Byob. Similmente all'approccio adoperato con le liste, i blocchi call e run permettono l'utilizzo di ingressi multipli per mezzo dei due pulsanti a freccette neri. Nell'eventualità che l'ingresso sia più di uno viene visualizzata la scritta with inputs, cliccando un numero arbitrario di volte si può raggiungere il numero di input desiderato.

Figura 7.2: Blocco call e blocco run

Se il numero di ingressi dato a uno di questi due blocchi (senza contare il blocco richiamato, quindi iniziando a contare da dopo la scritta with inputs) è lo stesso del numero di input slot liberi del blocco chiamato (quindi si parla degli slot liberi a sinistra della scritta with inputs) allora quest'ultimi verranno riempiti da destra verso sinistra con gli input forniti. Nell'eventualità venga dato un solo ingresso (a destra della scritta with inputs) allora tutti gli slot liberi verranno riempiti con quell'unico valore passato. In tutti gli altri casi Byob non fa nulla dato che non sono chiare le intenzioni dell'utente.

Figura 7.3: Riempimento automatico degli ingressi, primo caso

Figura 7.4: Riempimento automatico degli ingressi, secondo caso

Un altro concetto ostico per il quale vale la pena di spendere qualche parola è il passaggio in ingresso di blocchi, possibile grazie all'utilizzo del blocco operator denominato the block.

Figura 7.5: Il blocco the block

Questo blocco, non presente in Scratch, fa si che sia un blocco come tale e non il valore che esso assume una volta interpretato ad essere riportato come input quando si effettua la sua chiamata. Il blocco the block è reperibile nella tavolozza operator, di colore verde o è in alternativa utilizzabile implicitamente con un metodo più veloce che non richiede l'aggiunta di altri blocchi. E' sufficiente trascinare il blocco che rappresenta l'ingresso del call (o del run) più o meno vicino allo slot dell'input perché cambi la grafica dell'incastro. Quando il cursore viene gli viene avvicinato, lo slot di input si illuminerà esternamente e Byob assumerà che in ingresso sia stato passato un valore; ciò equivale a non utilizzare il the block. Se, proseguendo con il movimento del cursore, si raggiunge una posizione sufficientemente vicina allo slot di input, la grafica per l'incastro cambierà dall'illuminazione esterna a una illuminazione interna al blocco, segno che conferma il passaggio di un blocco come ingresso; in questo caso l'equivalenza è con l'utilizzo del the block.

Figura 7.6: Passaggio dell'ingresso per valore e per blocco, rispettivamente, con utilizzo della vicinanza del cursore

Segue un esempio semplice che faccia luce sui risultati appena descritti: è stato realizzato un blocco personalizzato 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.

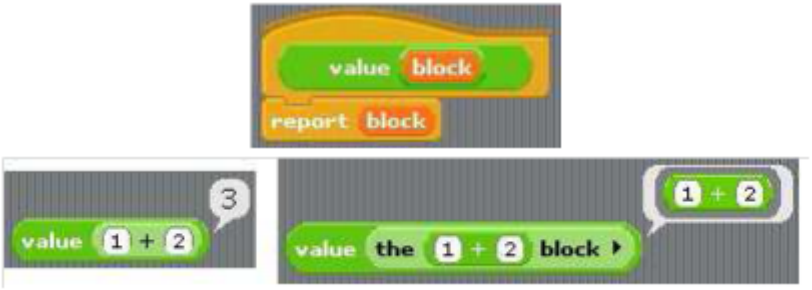


Figura 7.7: Il blocco Value e un esempio di utilizzo del blocco the block

o equivalentemente:

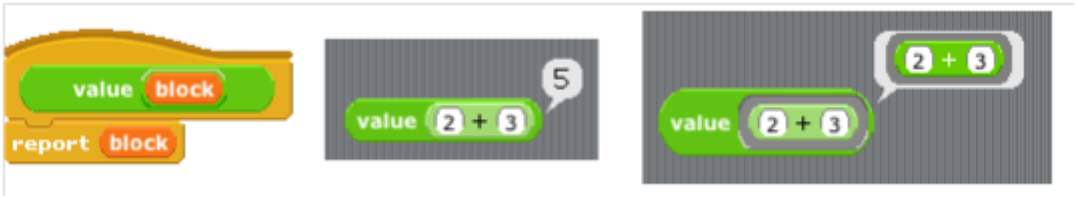


Figura 7.8: Alternativa all'uso del the block, viene usato il bordo grigio

Non avendo compreso appieno il comportamento di blocchi come call e run si rischia di incappare in errori come quelli mostrati nelle seguenti immagini:

Figura 7.9: Errore: l'ingresso al call è un valore, non una funzione

o alternativamente:

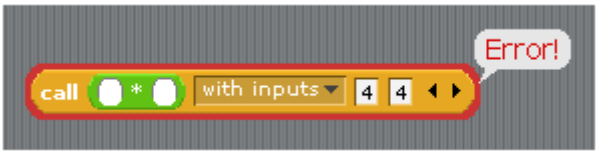


Figura 7.10: Errore: l'ingresso al call è un valore, non una funzione

L'idea che deve essere ben chiara è quella che blocchi quali call e run vogliono in ingresso altri blocchi. Nelle immagini non si sta chiamando non un blocco bensì il risultato di una operazione. C'è quindi una discrepanza tra il tipo di ingresso aspettato e quello effettivamente ricevuto che causa l'insorgere di un errore.

7.2 Procedure di ordine superiore

Una procedura è definita come di ordine superiore quando riceve in input o restituisce come output un'altra procedura. Sebbene uno slot di input di tipo Any accetti una procedura come ingresso, non si formerebbe il bordo grigio a segnalare l'evento come descritto nella sezione precedente e quindi l'effetto voluto di ritardo nell'interpretazione del parametro non ci sarebbe. Vale la pena quindi di eseguire le dichiarazioni degli ingressi, quando giustificato, come di tipo procedura.

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.

Portiamo ora un esempio di ingresso multiplo: l'input è di tipo Number, ma è stata scelta l'opzione Multiple inputs, quindi come input si avrà una lista di number.

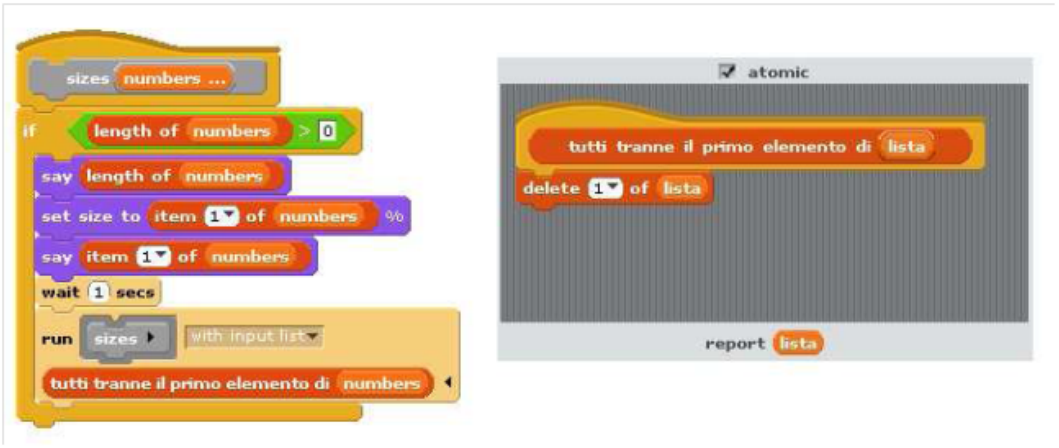


Figura 7.11: Esempio di input di tipo Multiple input: svolgimento

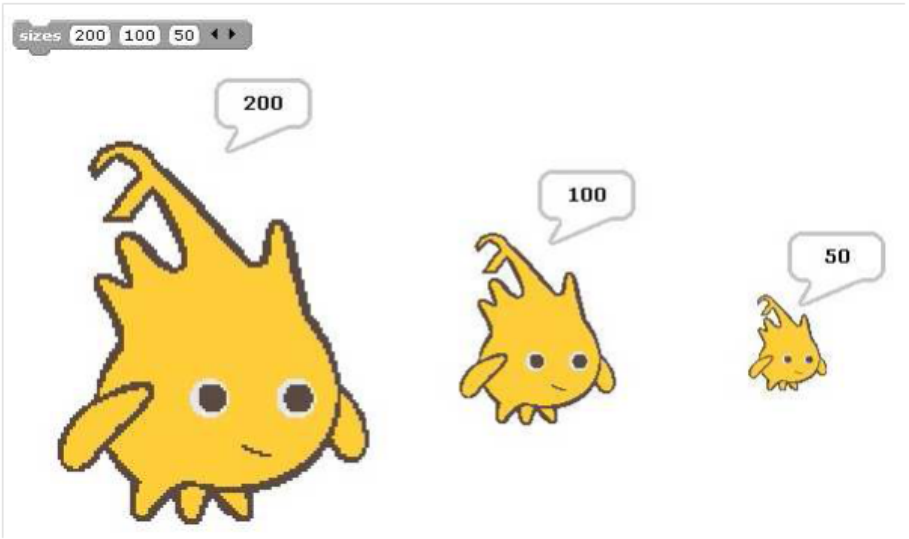


Figura 7.12: Esempio di input di tipo Multiple input: risultato

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: numbers. 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. Questo ci fa intuire

che con l'opzione multiple inputs, gli ingressi sono passati ed accettati come da dichiarazione (ovvero se dichiarati number si accetteranno una serie di number, se dichiarati any si accetteranno una serie di stringhe/numerici e così via), ma una volta arrivati all'esecuzione essi sono raggruppati in un'unica lista. C'è da tenere presente questo comportamento proprio nei casi di utilizzo delle procedure come dati, o in script ricorsivi.

Figura 7.13: Esempio di run con ingresso input list

A livello pratico, l'opzione *with input list* rimpiazza l'ingresso multiplo di tipo Any presente nel blocco run con uno singolo di tipo List. Gli elementi della lista sono presi singolarmente come input per lo script. Dal momento che numbers è una lista di numeri, ogni elemento è a sua volta un numero.

7.3 Procedure come dati

Da quanto detto finora si evince che nel caso si abbia una procedura in ingresso questo viene segnalato da Byob attraverso la comparsa di un bordo grigio attorno al blocco che rappresenta la procedura stessa. Più esplicitamente è possibile utilizzare i blocchi operator di colore verde the block o the script che eseguono una funzione equivalente. Esistono principalmente due motivi per i quali risulta necessario l'utilizzo di questi due blocchi:

1. c'è bisogno di avere un maggior controllo sull'uso degli ingressi in una procedura incapsulata ovvero usata come dato di prima classe;

2. si vuole usare una procedura come parametro in ingresso da inserire in uno slot di input il quale non è stato dichiarato essere di tipo procedura.

L'idea è ben resa da questi due punti i quali, tuttavia, non lasciano adito alla concretezza che può dare un'applicazione. Studiamo quindi qualche esempio che riassuma le casistiche più interessanti.

Partiamo da un semplice caso nel quale una procedura voglia essere esplicitamente utilizzata come un dato:

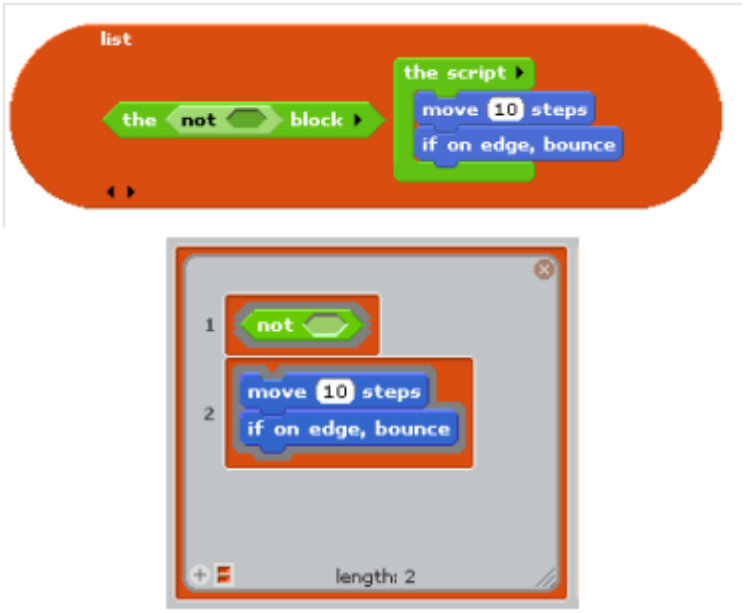


Figura 7.14: Esempio di procedura come dato di una lista

Viene costruita una lista di procedure ma il blocco lista, accettando ingressi di ogni tipo, non consente di inserire blocchi o script aggiungendo automaticamente il bordo grigio. Bisogna quindi indicare esplicitamente che l'input è costituito dal blocco stesso e non dal valore risultato della sua valutazione, tramite l'utilizzo di un blocco the block e di uno the script. Si noti come, nella seconda immagine, Byob mostri la lista di blocchi con il bordo grigio nonostante in fase di creazione siano stati usati the block e the script. E' facile capire che questo esempio rientra nella seconda delle motivazioni elencate a inizio paragrafo.

Prima di proseguire mettiamo in evidenza una caratteristica dei due blocchi appena citati che consentirà di comprendere più facilmente l'esempio successivo. Il blocco the block e quello the script segnano i loro input come dati, il primo accetta blocchi reporter e il secondo script. Entrambi hanno i pulsanti freccia a destra che permettono di fornire nomi espliciti agli ingressi previsti. Cliccando sulla freccia compare un ovale arancione contenente il nome predefinito #1 per il primo ingresso , #2 per il secondo e così via. I nomi assegnati di default possono essere cambiati con altri più contestualizzati cliccando sul ovale arancione. Quest'ultimi possono inoltre essere trascinati

nel blocco o nello script incapsulati. I nomi delle variabili di ingresso sono detti parametri formali della procedura incapsulata.

Un secondo esempio, conciso ma un po' artificioso, mostra come inserire uno degli input in due slot differenti:

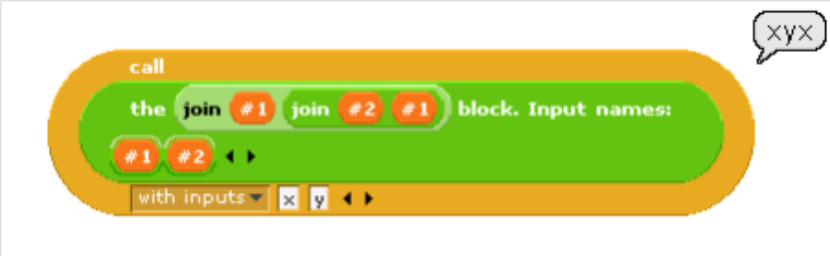


Figura 7.15: Inserimento di ingressi nel blocco call

Se vengono lasciati tutti e tre gli slot vuoti, Byob non riempirà nessuno di essi in quanto il numero di ingressi fornito (che nell'esempio sono due) non combacia con il numero di slot di input liberi. Una volta che il blocco chiamato ha fornito i nomi per i suoi ingressi, Byob non riempirà automaticamente i campi vuoti, sulla base dell'idea che ora l'utente possiede maggior controllo dell'esecuzione. In questo caso l'utilizzo di uno schema simile è motivato dal fatto che si voglia che Byob smetta di riempire slot di input che dovrebbero realmente rimanere vuoti. Ciò che abbiamo appena descritto è ben generalizzato dalla prima delle motivazioni elencate a inizio paragrafo.

Tuttavia è bene prestare attenzione quando viene utilizzato il blocco the block per controllare la denominazione degli ingressi del blocco incapsulato. E' facile commettere errori come inserire il bordo grigio attorno a the block il quale incapsulerebbe a sua volta il blocco the block, vanificando il semplice passaggio dell'ingresso originale come blocco:

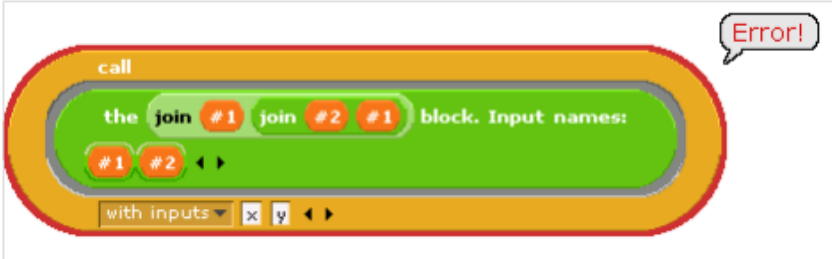


Figura 7.16: Errore causato utilizzando la notazione the block/bordo grigio

Per quanto sia difficile pensare ad un caso di applicazione pratico, è nelle potenzialità di Byob poter creare una lista contente anche blocchi di tipo the block; motivo per il quale viene visualizzato il bordo grigio automaticamente.

Un ultimo esempio più complesso porterà a galla la questione del controllo dei dati in ingresso, che può essere affinato usando le giuste tecniche:

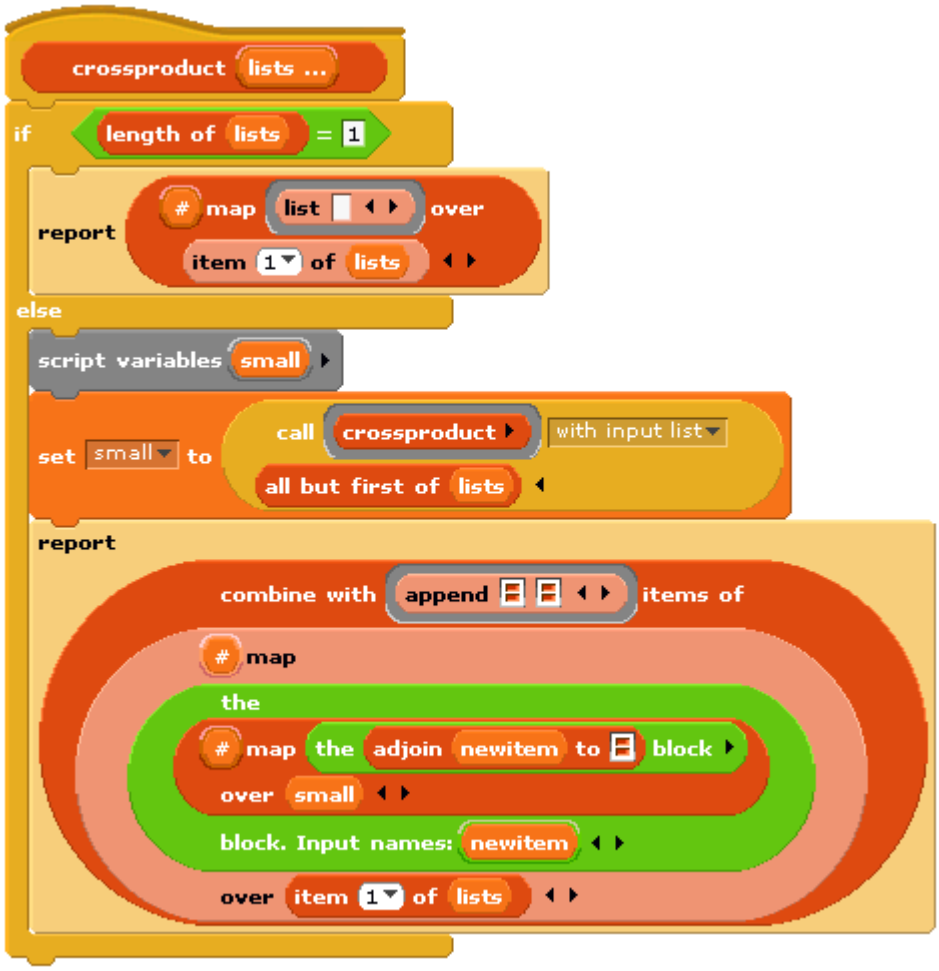


Figura 7.17: Esempio per evidenziare il controllo dei dati

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.

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 7.18: Esempio di procedure inserite nel blocco report

7.4 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.

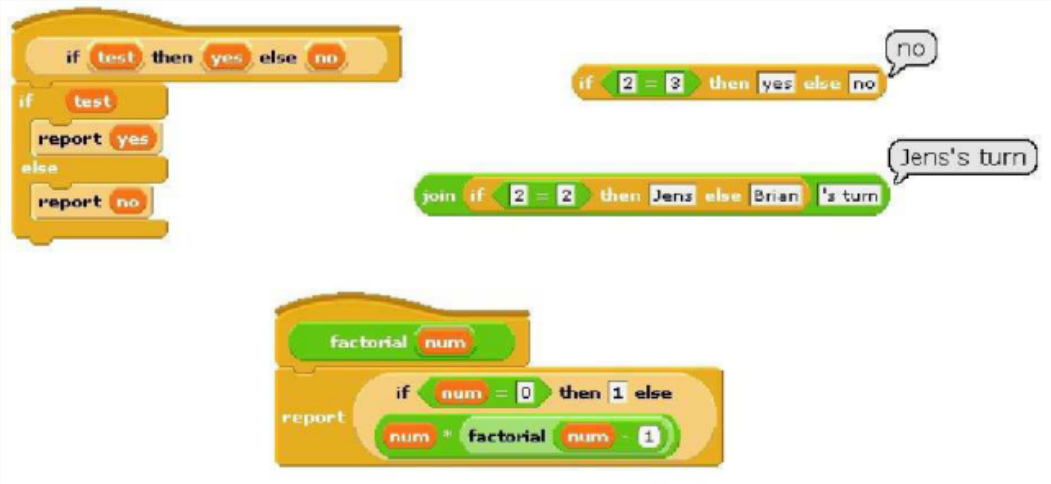


Figura 7.19: Esempio Scratch, funzionante solo se non si usa la ricorsione

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.

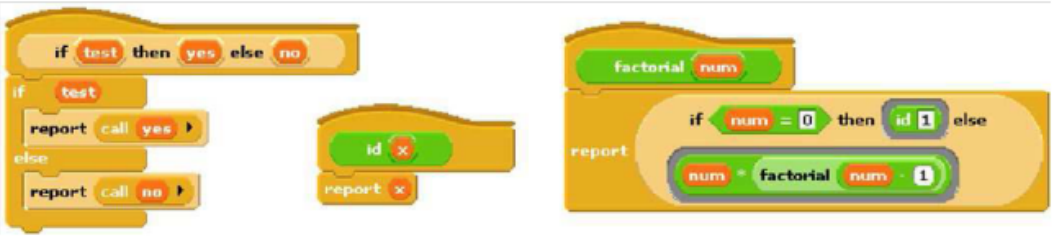


Figura 7.20: Esempio di fattoriale ricorsivo

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 7.21: 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 non saranno necessarie distinzioni tra booleani valutati e non.

Capitolo 8

Sprite e oggetti

La programmazione orientata agli oggetti è un paradigma di programmazione, la cui definizione è intrinsecamente legata a degli elementi detti, appunto, oggetti. Questi sono solitamente rappresentati dalle istanze di una classe e sono costituiti da campi (dati contenuti nelle variabili d'istanza) e metodi (operazioni che l'oggetto può eseguire). Molti studiosi e ricercatori hanno tentato di definire un elenco delle caratteristiche fondamentali che deve avere un linguaggio orientato agli oggetti, senza mai però giungere ad una visione unificata. E' possibile tuttavia includervi alcune caratteristiche fondamentali quali:

- astrazione: un sistema complesso va scomposto nelle sue parti fondamentali e queste parti vengono descritte singolarmente in un linguaggio semplice e preciso. Tutto ciò rappresenta un aspetto molto importante per gli utenti Byob, i quali sanno che la simulazione di un realtà è strettamente connessa al concetto chiave di micromondo;

- incapsulamento: le diverse componenti di un sistema software non devono rivelare i dettagli interni delle rispettive implementazioni;

- comunicazione a scambio di messaggi: è una tipologia di comunicazione tra processi che prevede che non ci siano risorse condivise e che tutte le comunicazioni avvengano attraverso l'invio di messaggi tra i processi;

- modularità: le diverse componenti del sistema software vengono divise in tante unità di funzionamento separate tra loro;

- riusabilità: la possibilità di riutilizzare un intero sistema software o parti di esso nello sviluppo di altri prodotti software, che comprende a sua volta 3 concetti più specifici: polimorfismo, ereditarietà e genericità.

8.1 Sprite di prima classe

Scratch nasce con gli sprite, attori sullo stage la cui astrazione si presta facilmente ad una loro interpretazione come oggetti. Ogni sprite infatti possiede delle variabili locali e degli script (metodi) che lo caratterizzano rispetto agli altri. Tuttavia ci sono almeno tre ragioni per cui gli sprite di Scratch sono meno versatili degli oggetti di un linguaggio OOP (Object Oriented Programming):

1. lo scambio di messaggi limitato in quanto i messaggi possono essere trasmessi solo in broadcast e non indirizzati ad un singolo sprite, i messaggi non hanno ingressi, i metodi non possono restituire valori alla loro chiamante;

2. non esiste un vero e proprio meccanismo di ereditarietà: è possibile duplicare uno sprite senza però che i cambiamenti avvenuti nello sprite originale vengano estesi al suo duplicato;

3. mentre nei linguaggi che utilizzano un paradigma orientato agli oggetti, quest'ultimi sono considerati alla stregua di dati (possono essere immagazzinati in una variabile, in una lista, ...), in Scratch questo non è possibile.

Con l'avvento di Byob però, gli sprite sono diventati dati di prima classe e possono essere creati e cancellati da uno script, memorizzati in una variabile o all'interno di una lista, possono inviare messaggi a destinatari singoli ed ereditare le proprietà di un altro sprite. Uno sprite può essere clonato, dando vita a un nuovo sprite che condivide metodi, variabili e liste con il suo genitore. Come vedremo in dettaglio più avanti, ogni proprietà dello sprite clonato può essere condivisa individualmente o separata dalla proprietà corrispondente del genitore.

Il mezzo fondamentale attraverso il quale i programmi hanno accesso agli sprite è il blocco object di tipo reporter. Questo blocco dispone di un menù a tendina che, se cliccato, elenca tutti gli sprite, lo stage e le parole chiave speciali myself e all sprite , l'ultima di queste opzioni restituisce un elenco di sprite invece che un singolo oggetto. Un oggetto riportato da un blocco object può essere utilizzato come ingresso per qualsiasi blocco che accetti ingressi di tipo any, come ad esempio il blocco set o il blocco say.

8.2 Comunicazione

I messaggi che uno sprite accetta sono rappresentati dai blocchi presenti nelle sue tavolozze, inclusi quelli this sprite only e all sprite. Per i blocchi personalizzati, i metodi corrispondenti sono gli script così come si vedono nell'editor dei blocchi.

In Scratch il blocco <var> of <sprite>, reperibile nella palette sensing, permette che uno sprite faccia riferimento a determinati attributi di un altro. Ora, in Byob, questo blocco viene esteso per consentire l'accesso a qualsiasi proprietà di uno sprite. Inoltre, per non perdere compatibilità con Scratch, sono stati aggiunti due ingressi composti da alcuni menu a tendina. Questa pratica non pone comunque limitazioni in quanto qualsiasi espressione che riporti un oggetto può essere trascinata nello slot di destra e ogni blocco incapsulato in the block o the script può essere inserito nello slot di sinistra. Il risultato può essere applicato ad un call o un run per utilizzare il metodo corrispondente.

Figura 8.1: Esempio di utilizzo del blocco <var> of <sprite>

Si noti che il blocco move è di solito globale; se usato in uno script, muove qualunque sprite sia in possesso di quello script. Ma quando combinato con il blocco of della palette sensing, come nella foto sopra, il risultato è un blocco mirato a muovere sempre e solo lo Sprite2 indipendentemente da quale sprite lo esegua.

Per comodità, BYOB comprende un blocco launch, identico nell'esecuzione al blocco run tranne che per la chiamata del metodo come uno script separato, peculiarità che da modo allo script chiamante di continuare la sua esecuzione svolgendo altri compiti. In un certo senso il Launch può essere pensato come un broadcast migliorato, mentre il run del metodo di un altro sprite è analogo al broadcast and wait.

A dispetto di quanto si potrebbe intuire dall'esempio portato nell'immagine precedente, gli slot di input di un messaggio possono essere lasciati anche vuoti e gli ingressi forniti tramite le opzioni aggiuntive dei blocchi call o run.

8.3 Stato locale

La memoria in uno sprite è mantenuta in due modi differenti:

- dalle sue variabili, create esplicitamente dall'utente con l'utilizzo del pulsate Make a variable
- dai suoi attributi ovvero quelle proprietà di cui ogni sprite è dotato, come la posizione, la direzione o il colore della sua penna.

In generale un qualsiasi blocco che modifichi una variabile o un attributo viene detto modificatore (o setter method) e uno che ne permetta la lettura restituendone il valore viene detto esaminatore (o getter method). Ogni variabile può essere esaminata adoperando il relativo blocco ovale arancione e può essere modificata grazie all'uso del blocco set. Per gli attributi il discorso è diverso e vale la pena richiamare il funzionamento di Scratch per poi confrontarlo con quello di Byob, così da comprenderne appieno l'evoluzione e il miglioramento.

In Scratch solo alcuni attributi hanno dei blocco dedicato che ne consentono la lettura e la modifica. Ad esempio la direzione di uno sprite può essere esaminata tramite il blocco direction e modificata con quello point in direction. Esistono anche vie alternative e meno dirette per raggiungere lo stesso scopo utilizzando blocchi come turn, point forward, if on edge e bouce. Per contro, esistono anche attributi, come il colore della penna, per i quali esistono blocchi modificatori ma non esaminatori e viceversa. Addirittura il nome di uno Sprite in Scratch non può essere né esaminato né modificato a livello di codice.

In alcuni contesti è utile distinguere tra una variabile, o un attributo, intesi come elementi a se stanti e il valore che la variabile, o l'attributo, stessi assumono. In Byob il valore di variabili e attributi è ottenibile tramite l'output del relativo blocco esaminatore mentre le variabili e gli attributi, visti come elementi, sono rappresentati dal blocco esaminatore stesso. Per isolare variabili e blocchi come elementi si usa il blocco the block, analizzato precedentemente. Ad esempio il valore riportato dal blocco getter direction è un numero che indica il valore della direzione corrente che ha lo sprite mentre il valore riportato dal blocco the block a cui viene applicato in ingresso il blocco direction, rappresenta l'attributo direzione stesso.

L'utilizzo di the block per rappresentare un attributo funziona solo se esiste un blocco getter per quel particolare attributo. Volendo rappresentare

Figura 8.2: Il blocco direction

tutti i componenti dello stato di uno sprite, invece di aggiungere decine di blocchi esaminatori nelle tavolozze, Byob introduce un unico blocco attribute, nella palette azzurra sensing, il quale, grazie ad un menù a tendina, elenca tutti gli attributi che uno sprite ha.

Figura 8.3: Il blocco per gli attributi

Uno dei vantaggi dei blocchi di prima classe in Byob è che non c'è bisogno di un equivalente del blocco attribute che svolga la funzione di modificatore per ogni attributo. Un blocco con caratteristiche simili è facilmente costruibile con l'applicazione del blocco attribute a quello set <variable> to <value>.

Figura 8.4: Esempio di modifica di un attributo

Si noti che la funzione di modifica è applicabile solo ai blocchi esaminatori di variabili e attributi e non per quelli reporter in generale. Non è possibile, ad esempio, una situazione come quella in figura per impostare la variabile x a 25.

8.4 Prototipazione

Attualmente la maggior parte dei linguaggi orientati agli oggetti utilizzare un approccio classe/istanza per la creazione di oggetti. Riprendiamo la definizione di classe introdotta all'inizio della sezione precedente. Una classe è un costrutto usato come modello per creare oggetti. Tale modello comprende attributi e metodi che saranno condivisi da tutti gli oggetti creati sulla base di quella classe, detti istanze od oggetti istanziati. BYOB utilizza un approccio diverso chiamato prototipazione, in cui non vi è alcuna distinzione tra classi e istanze.Esso permette la creazione di uno sprite figlio tramite la clonazione di uno sprite già esistente, detto prototipo, che rappresenterà il genitore. Inoltre modifiche effettuate su un genitore, quali il miglioramento del codice o la correzione di errori, vengono estese automaticamente ai figli. Alla luce di questi comportamenti si può anche pensare di simulare un paradigma a istanze nascondendo con il comando hiding lo sprite prototipo, che fungerà da classe base. La prototipazione è anche più in linea con il principio di progettazione di Scratch secondo il quale tutto, in un progetto, dovrebbe essere concreto e visibile sullo stage, a differenze del modello a istanze per cui tutto comincia con il concetto astratto di classe, che può essere concretizzato solo dopo una formale definizione della stessa. Ci sono tre modi per creare uno sprite figlio partendo dal padre:

1. all'interno della finestra degli sprite in basso a destra della schermata principale di Byob, selezionando il comando clone dal menù a tendina che si apre sopra lo sprite padre cliccando il testo destro del mouse in ambiente windows o control-cliccacando in ambiente iOS;

2. utilizzando un blocco clone, prelevabile dalla tavolozza verde operators, che crea e riporta uno sprite figlio;

3. impostando adeguatamente l'attributo parent di uno sprite cambiando così il suo genitore

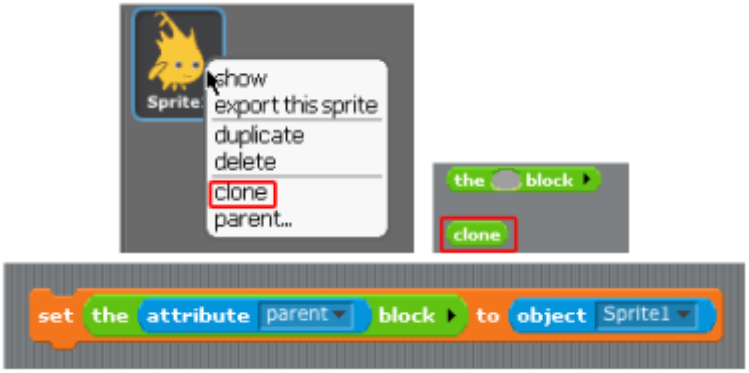


Figura 8.5: Il blocco/comando clone e la modifica dell'attributo parent

Nei progetti che creano cloni, il numero totale di sprite può crescere considerevolmente rendendo difficoltosa la distinzione tra genitori e figli. Per ovviare a questo inconveniente è sufficiente cliccare con il tasto destro del mouse (o control-cliccare) sull'icona dello stage, nella finestra degli sprite, è verrà visualizzato un elenco contenente le icone ed i nomi dei soli sprite senza genitori. Poi, cliccando su uno sprite, esso verrà selezionato mostrando i suoi figli nel caso ne abbia. Il processo è ripetibile anche con i figli dei figli e così via.

8.5 Ereditarietà tramite delega

Ogni clone eredita delle proprietà del suo genitore, che includono script, blocchi personalizzati, variabili, liste, attributi di sistema, costumi e suoni. Ogni proprietà, indipendentemente dalle altre, può essere condivisa tra genitore e figlio o, viceversa, può non essere condivisa; nel secondo caso ne viene mantenuta una versione diversa nel figlio, separata da quella del padre. Il blocco esaminatore di una proprietà comune viene visualizzato, nella tavolozza del figlio, con un colore più chiaro; proprietà diverse vengono visualizzate con i colori normali. Vediamone il principio di funzionamento.

Quando viene creato un nuovo clone le impostazioni predefinite prevedono che esso condivida tutte le sue proprietà con il genitore, fatta eccezione per gli attributi di sistema. Se il valore di una proprietà condivisa subisce

un cambiamento nel genitore, esso verrà esteso anche i figli, che ne vedranno il nuovo valore. Viceversa, se il valore di una proprietà condivisa viene modificata nel figlio, allora il legame di condivisione viene spezzato e viene creata una nuova versione privata della proprietà nel figlio. Mediante questo meccanismo il figlio sceglie di non condividere una proprietà con il padre. Le modifiche in questo contesto comprendono l'applicazione di un blocco set o change ad una variabile, la modifica di un blocco nell'editor dei blocchi o la modifica, l'inserimento, l'eliminazione, il riordino di un costume o di un suono. Per portare una proprietà dalla non condivisione alla condivisone è sufficiente cancellare la versione privata presente nel figlio applicandovi il blocco command delete. Questo blocco accetta come ingressi sia sprite che loro proprietà (i relativi blocchi esaminatori), inseribili nello slot di input di tipo Reporter che consente di utilizzare la notazione a bordo grigio. Si noti che gli sprite, non essendo proprietà, non necessitano del bordo grigio una volta inseriti.

Figura 8.6: Il blocco delete e degli esempio del suo utilizzo

Quando un sprite riceve un messaggio per il quale non ha una corrispondente blocco, il messaggio viene delegato al genitore dello sprite. In caso contrario, com'è intuibile, il messaggio viene gestito dallo sprite che ha ricevuto originariamente il messaggio.

Ricordiamo inoltre che ogni blocco o variabile possono essere istanziati con attivando l'opzione *This sprite only* che rende private fin da subito ciò che si va a creare, evitando i meccanismi della prototipazione.

8.6 Sprite annidati

A volte è desiderabile creare uno sprite che abbia alcune caratteristiche di altri sprite, le quali possano lavorare assieme o separatamente a seconda che la situazione lo richieda o meno. In Byob è possibile designare uno sprite ad essere l'ancora di una forma combinata, composta dalle parti di sprite

differenti. Una simile fusione viene detta nidificazione e non è altro che una forma di collegamento tra sprite, diversa da quella già trattata che collega un padre/prototipo ai propri figli. La nidificazione può essere impostata sia in modo interattivo che tramite script. Nel primo caso si procede trascinando l'icona della parte di uno sprite dalla finestra sprite (che si trova in basso a destra della schermata principale di Byob) e rilasciandola sull'icona, nello stage, dello sprite designato come ancora. Nel secondo caso si agisce tramite script modificando l'attributo anchor di uno sprite come si farebbe con un qualsiasi altro attributo. Una volta effettuata, la nidificazione è riscontrabile visivamente sulle icone dello sprite ancora e delle varie parti ad esso collegate.

Figura 8.7: Sprite annidati

Qualsiasi cambiamento nella posizione o le dimensioni dello sprite ancora viene sempre esteso ad ogni sua parte.

8.7 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 8.8: La lista di attributi

Capitolo 9

Paradigma ad oggetti

È dimostrabile che un qualunque linguaggio di programmazione che disponga di procedure di prima classe permette di implementare oggetti in maniera esplicita. Si pensi di realizzare un linguaggio nel quale ogni oggetto sia rappresentato come una procedura di invio prenda un messaggio come ingresso e ne riporti il corrispondente metodo.

9.1 Stato locale

Iniziamo con un esempio introduttivo che mostri come le variabili che formano lo stato locale lavorino con fine ultimo quello di costruire un'implementazione completa del paradigma di programmazione a classe/istanza e a prototipazione.

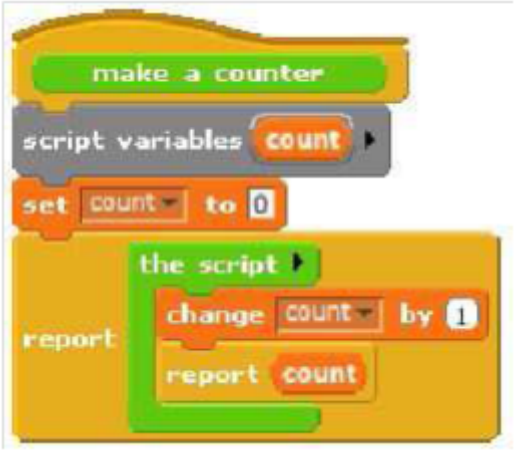


Figura 9.1: Esempio di contatore

Questo script implementa un particolare tipo di oggetti ovvero una classe, vale a dire la classe contatore counter. In questa prima versione semplificata è stato sviluppato solo un metodo, quindi non è necessario un passaggio di messaggi esplicito. Quando il blocco personalizzato make a conuter viene chiamato, esso restituisce una procedura creata dal blocco the script posto al suo interno. Tale procedura implementa uno specifico oggetto di tipo contatore cioè un'istanza della classe counter che, quando invocata, incrementa e riporta in uscita la sua variabile di conteggio count. Si può pensare di creare con lo stesso metodo più oggetti contatore, ognuno dei quali avrà la propria variabile count, indipendente da quella delle altre istanze.

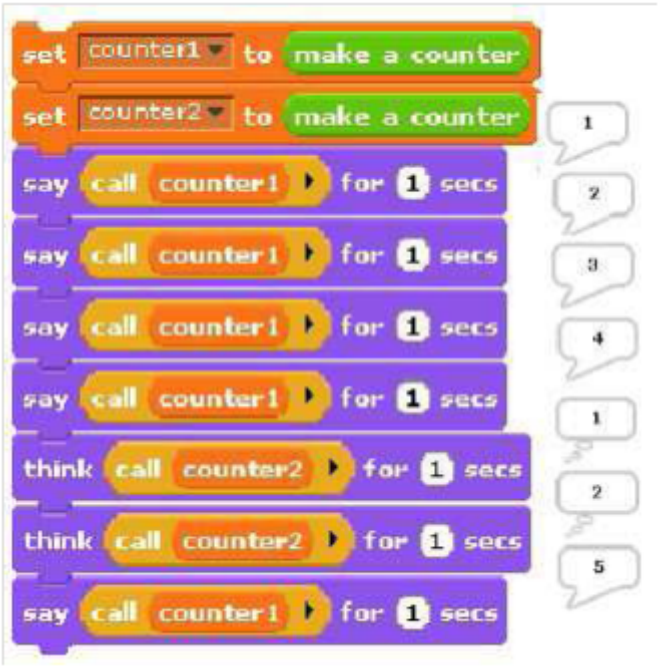


Figura 9.2: Uscita di contatore

Dal punto di vista della procedura make a counter, ogni sua invocazione porta alla creazione di una nuova variabile count associata al blocco. Normalmente le variabili di blocco non durano in eterno e vengono cancellate non appena la procedura termina, ma, in questo caso, si fa in modo che la procedura restituisca a sua volta un'altra procedura la quale ha il compito di creare una variabile count che rimarrà attiva. Il blocco script variables, infatti, rende le variabili locali allo script nel quale viene invocato. Tali variabili di script non possono essere trascinate fuori dallo script in cui vengono create, ma possono essere esportate indirettamente se usate e restituite da

una procedura.

In questo approccio alla programmazione ad oggetti si è riusciti a rappresentare sia le classi che le relative istanze con l'uso di procedure. Il blocco make a counter rappresenta la classe mentre ogni istanza è rappresentata dallo script senza nome creato ogni volta che make a counter viene invocato. Le variabili di script create all'interno del blocco make a counter ma al di fuori del blocco the script sono considerate variabili di istanza appartenenti ad uno specifico contatore.

9.2 Messaggi e procedure di invio

Nell'esempio precedente era sufficiente richiamare un'istanza per mandare in esecuzione l'unico metodo presente, delineando una situazione nella quale non era obbligatorio l'uso di messaggi. Basandoci sull'esempio precedente, introduciamo ora un secondo esempio, più strutturato, che utilizzi questa volta la comunicazione tramite passaggio di messaggi:

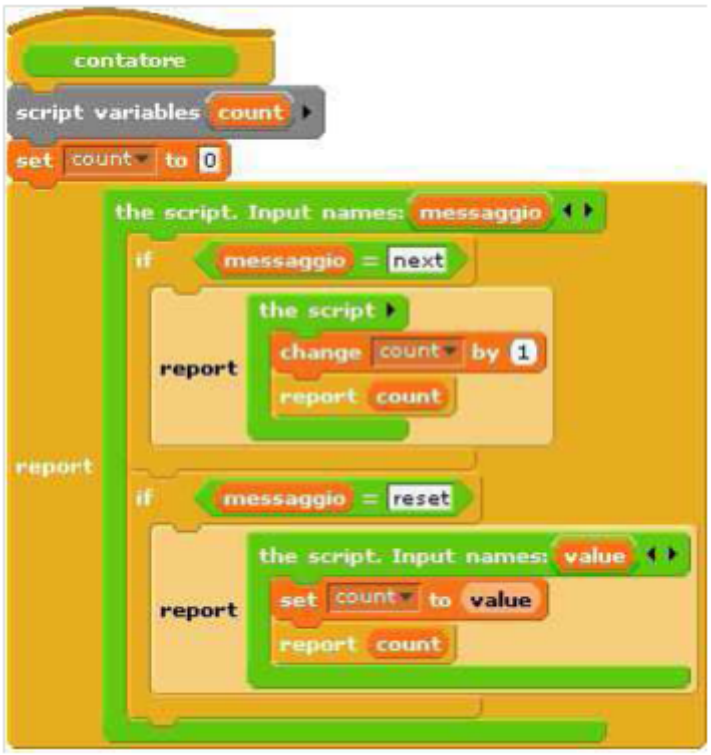


Figura 9.3: Esempio di contatore con dispatch procedure

Anche in questo caso il blocco make a counter rappresenta la classe contatore e, di nuovo, lo script crea una variabile count ogniqualvolta esso venga invocato. Il blocco the script più esterno rappresenta l'istanza della classe. Si tratta di una procedura di invio che accetta in ingresso un messaggio, ad esempio una stringa di testo e in base a questa riporta il metodo opportuno. I due blocchi the script più piccoli, inglobati in quello più grande, sono invece i metodi. Quello più in alto è il metodo per passare alla fase di conteggio successiva mentre quello in basso è il metodo di reset per portare il conteggio ad un valore arbitrario.

Mentre nella prima versione era sufficiente chiamare l'istanza per terminare l'esecuzione, adesso bisogna agire in due fasi; prima viene effettuata una chiamata all'istanza che dà la possibilità di scegliere uno dei metodi da utilizzare e successivamente è possibile terminare l'esecuzione chiamando il metodo desiderato. Siamo in grado di creare un blocco personalizzato che esegua entrambe le chiamate in una volta sola:

Figura 9.4: Blocco ask che esegue entrambe le procedure in una volta

Come si vede dall'immagine il blocco ask richiede due ingressi: un oggetto e un messaggio. Accetta anche ulteriori ingressi opzionali che Byob conserverà nella lista args. Analizzando invece il corpo del blocco si notano due call annidati: quello interno chiama l'oggetto ovvero la procedura di invio che chiede sempre e solo un ingresso, vale a dire il messaggio. In risposta si riceve un metodo che può prendere un qualsiasi numero di ingressi. Si noti che questo è uno dei casi in cui è utile usare l'opzione with input list del blocco call.

9.3 Ereditarietà

Dopo aver implementato le variabili locali e la comunicazione tramite passaggio di messaggi, passiamo all'ereditarietà. Possiamo sviluppare questa funzionalità utilizzando la tecnica della delega. Ogni istanza della classe figlio contiene un'istanza della classe genitore a cui semplicemente trasmette i

messaggi che non le competono, ovvero quelli che non sono stati sovrascritti e re-implementati:

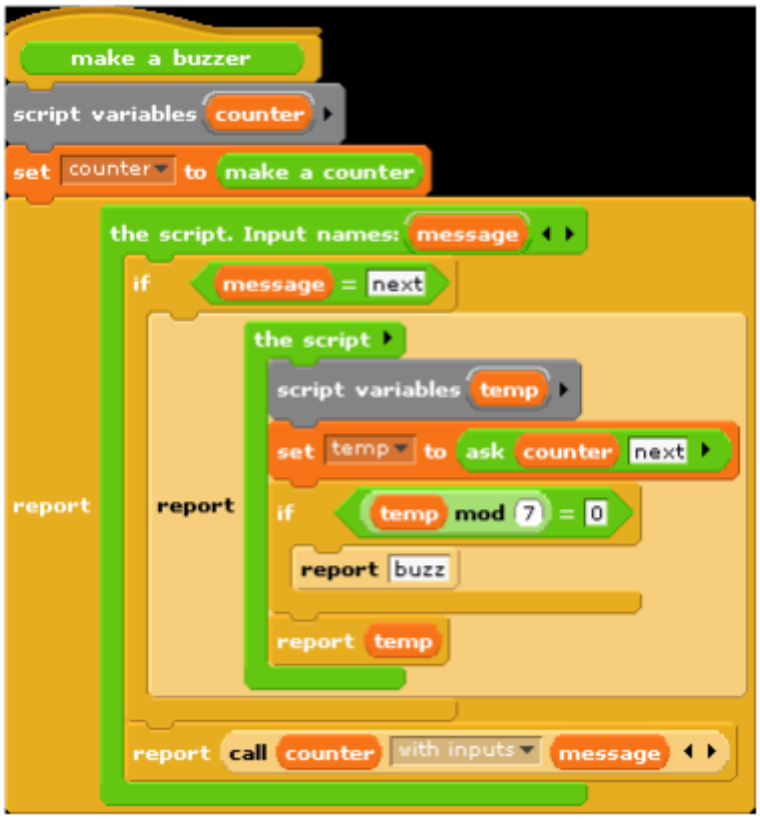


Figura 9.5: Esempio di contatore buzzer

L'immagine rappresenta uno script che implementa la classe buzzer, figlia della classe contatore. Invece di avere un count, ciascuna istanza di buzzer ha un oggetto counter come variabile di stato locale. All'interno di buzzer viene sovrascritto il metodo next facendo in modo che restituisca lo stesso risultato di counter a meno che esso non sia divisibile per 7, nel qual caso viene restituito un buzz. Se il messaggio è qualcosa di diverso da next viene attivata la procedura di invio al genitore, counter che gestirà qualunque messaggio che il buzzer non sa trattare esplicitamente.

9.4 Prototipazione

Nel sistema classe/istanza sviluppato negli esempi precedenti è necessario progettare il comportamento completo della classe prima di poterne creare

delle istanze. Questo è un ottimo approccio per una progettazione top-down, ma non eccezionale per la sperimentazione. Qui delineare l'implementazione di un sistema basato sulla prototipazione ed orientato agli oggetti, modellato sulla base del comportamento degli sprite in Byob, per i quali le proprietà di un figlio sono condivise con il genitore fino a quando non vengono modificate. Poiché questi oggetti esplicitamente costruiti non sono sprite, non ci sono nemmeno attributi di sistema; le loro proprietà consistono solo di metodi e variabili locali.

Proprio perché vogliamo essere in grado di creare ed eliminare proprietà dinamicamente, non usiamo le variabili di Byob per conservare variabili o metodi di un oggetto. Si adoperano invece due tabelle per ogni oggetto, chiamate methods e data chiamati metodi e dati, ciascuna delle quali rappresenta una lista di associazioni ovvero i cui dati sono composti da liste di due elementi: una chiave e il corrispondente valore. Viene poi messa a disposizione una procedura di ricerca per individuare la coppia chiave-valore corrispondente alla data chiave in una determinata tabella.

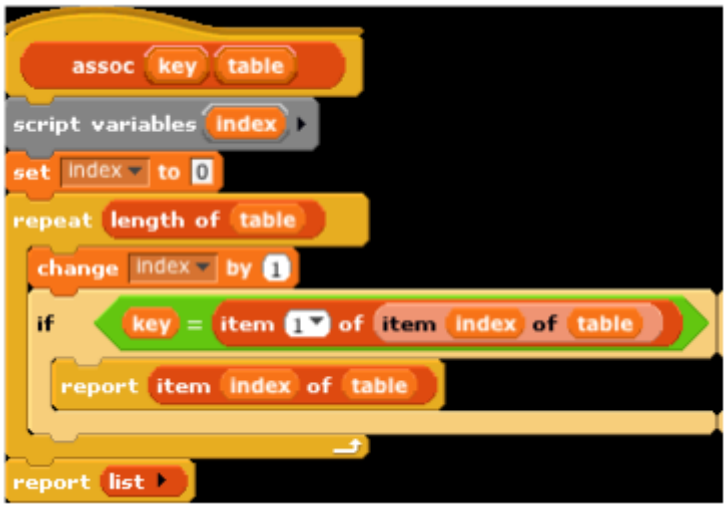


Figura 9.6: Blocco per la ricerca della coppia chiave/valore

Ci sono anche comandi per inserire e cancellare le voci nella lista:

Figura 9.7: Blocchi per inserire e cancellare elementi

Come nel versione a istanze, un oggetto è rappresentato come una procedura di invio che accetta in ingresso un messaggio e riporta in uscita il metodo corrispondente. Quando un oggetto riceve un messaggio lo cercherà nella sua tabella di metodi, secondo la parola chiave ricevuta. Se viene poi trovato, il valore corrispondente sarà il metodo che deve essere invocato. In caso contrario, l'oggetto proseguirà la ricerca nella sua tabella dati. Se viene trovata una corrispondenza, quello che restituirà l'oggetto non sarà un valore bensì un metodo reporter che, se chiamato, riporterà il valore. Ciò che restituisce un oggetto è quindi sempre un metodo.

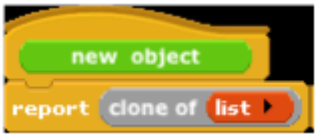
Se l'oggetto non ha né un metodo né un dato con il nome desiderato ed ha un genitore, quest'ultimo o meglio, la sua procedura di invio, viene richiamata con il messaggio come ingresso. Se nemmeno in questo caso viene trovata una corrispondenza significa che l'oggetto non ha un genitore e che

quindi l'utente ha richiesto un metodo ad un oggetto che non lo possiede nel suo repertorio, generando così un errore.

I metodi possono accettare un numero arbitrario di ingressi, come nel modello a istanze, ma in quello basato sulla prototipazione ogni metodo ottiene automaticamente l'oggetto a cui il messaggio è stato originariamente inviato come primo ingresso. E' necessario agire in questo modo in quanto se un metodo si trovasse nel genitore (o nel genitore del genitore, ecc) del destinatario originale e quel metodo riferisse a una variabile o ad un altro metodo, allora verrebbe utilizzata la variabile o il metodo del figlio nel caso ne conservi una propria versione.

Il blocco clone of riportato nell'immagine 9.12 a fine capitolo, prende un oggetto come ingresso e ne genera un figlio.

Ogni oggetto viene creato con metodi predefiniti quali set, method, delete-var, delete-method e clone e con una variabile predefinita parent. Oggetti senza un genitore sono creati chiamando nuovo oggetto:



Come prima sono fornite procedure per chiamare la procedura di invio di un oggetto e quindi per chiamare il metodo appropriato. In questa versione forniamo l'oggetto desiderato come ingresso al primo metodo:

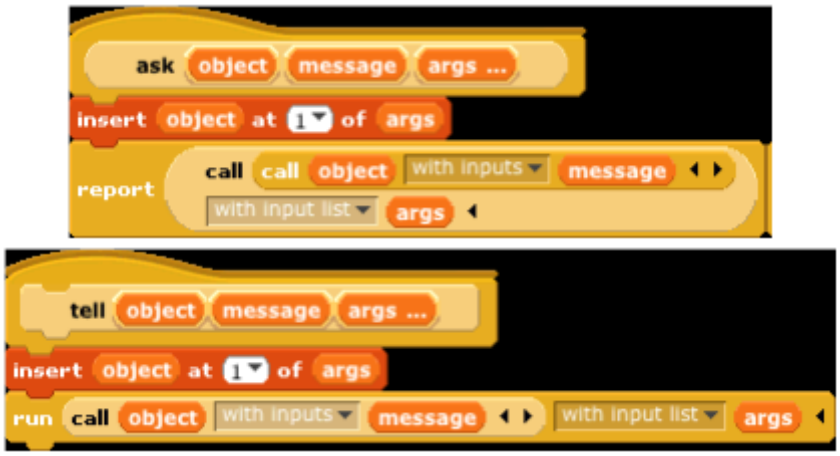


Figura 9.8: Procedure rivisitate per le chiamate ad un oggetto

Lo script nella pagina seguente mostra come questo sistema di prototipazione può essere usato per creare oggetti contatori counter. Si comincia con un contatore prototipo, chiamato counter1. Questo primo contatore viene quindi usato un paio di volte e successivamente creato un figlio counter2 a cui viene assegnata una variabile count personale, ma non una variabile total personale. Il metodo next imposta sempre la variabile total di counter1, che mantiene pertanto il conteggio del numero totale di volte che ogni contatore viene incrementato. L'esecuzione di questo script dovrebbe dire tramite il blocco say e pensare tramite il blocco think le seguenti liste: [1 1] [2 2] [3 3] [4 4] (1 5) (2 6) (3 7) [5 8] [6 9] [7 10] [8 11]

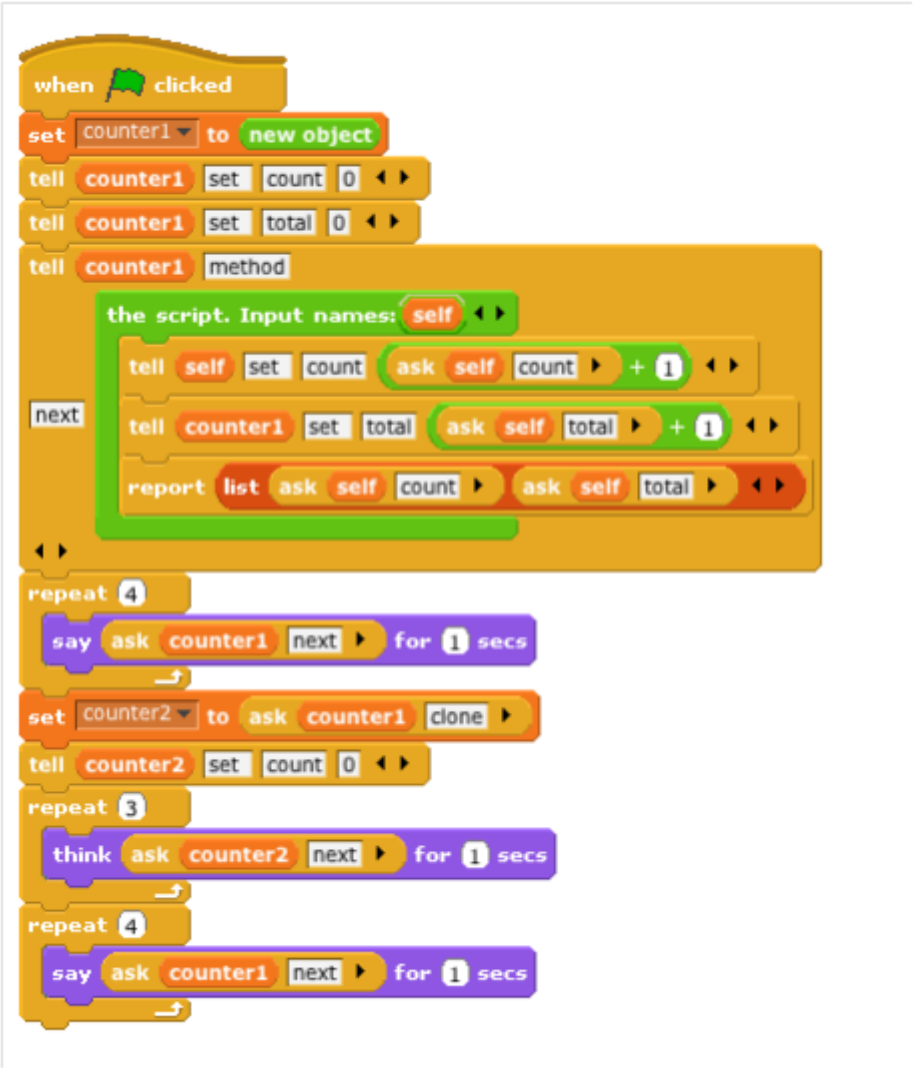


Figura 9.9: Esempio di creazione di oggetti contatori

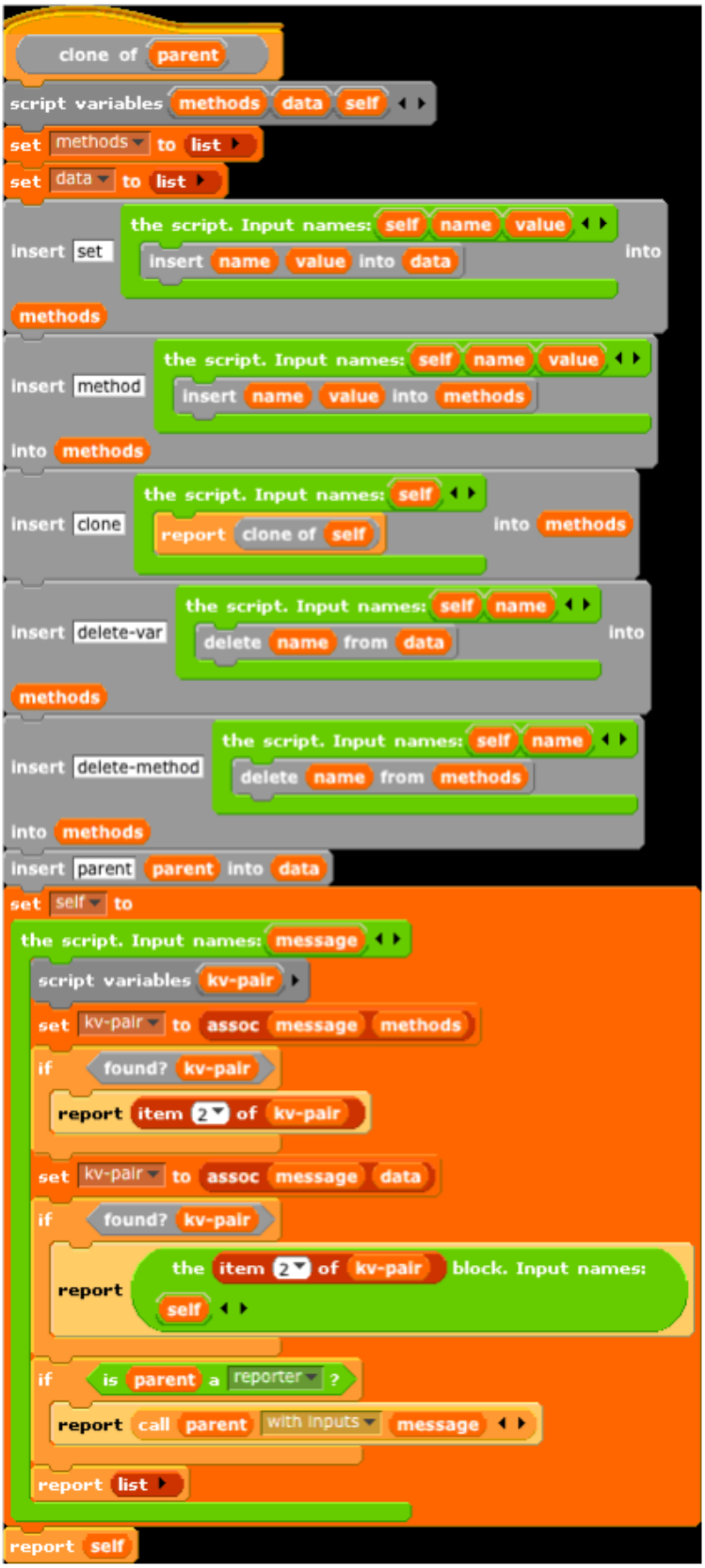


Figura 9.10: Blocco per la generazione di un figlio dato l'oggetto padre

Capitolo 10

Coclusioni

Dopo aver fornito una visione d'insieme su Byob e sui principali linguaggi di programmazione visuali, dai quali esso eredita i tratti distintivi, iniziano a delinearsi gli aspetti comuni e le peculiarità di ognuno.

Concentriamo per un momento la nostra attenzione su Scratch e Byob. Il primo dei due risulta essere un linguaggio il cui utilizzo è indirizzato a quella fascia di utenti neofiti che cercano un approccio morbido all'informatica e alla programmazione. Questo genere di persone abbisogna infatti di un linguaggio modulare, intuitivo e più di ogni altra cosa sempre attivo, che restituisca dei feedback immediati ad ogni comando, così da iniziare a prendere confidenza con i costrutti, le strutture e il metodo di ragionamento. Tutte caratteristiche, queste, che descrivono integralmente Scratch rendendolo un buon candidato per la prima istruzione e lo riconfermano degno successore del Logo costruttivista che Papert sviluppò alla fine degli anni sessanta.
Passando oltre e iniziando a parlare di informatica più avanzata, Scratch si esaurisce senza avere più nulla da offrire. La questione è banalmente riscontrabile nel momento in cui si tenti la costruzione di strutture dati complesse o l'applicazione di tecniche leggermente più sofisticate come quelle della ricorsione o della gestione di blocchi personalizzati e procedure. La ridotta possibilità nell'affrontare argomenti non troppo avanzati con Scratch è riscontrabile anche online, semplicemente navigando ed esplorando la sezione progetti della popolata community ufficiale di Scratch, nella quale, nonostante il grande numero di progetti condivisi, non si trovano che progetti semplici perlopiù basati sulla grafica e sull'interazione con l'utente. Considerando quindi le sue potenzialità e gli ambiti in cui trova applicazioni concrete, viene rinnovata l'impressione che Scratch, probabilmente per scelta dei suoi progettisti, sia rivolto ad utenti alle prime armi, comunque aiutati nell'apprendimento anche dalle numerose risorse disponibili.

Veniamo quindi all'evoluzione, parlando di Byob. Quest'ultimo, dal punto di vista didattico ed educativo di un neofita, non aggiunge nulla al suo predecessore, rendendolo uno strumento altrettanto valido, in grado di sostituire Scratch. Ma la sua evoluzione non va intesa in questo senso, bensì in quello di riportare alla programmazione visuale le possibilità tipiche dei più comuni linguaggi testuali.
Oltre ad introdurre la possibilità di creare blocchi personalizzati, è stata aperta la strada per l'applicazione, diretta o indiretta, di concetti più ostici come quelli della programmazione ad oggetti, della ricorsione o dei dati di prima classe. Inutile dire che la loro trattazione è stata pensata per un pubblico di utenti che abbiano già avuto un'infarinatura informatica di base. Purtroppo le potenzialità di Byob non sembrano essere sfruttate adeguatamente stando alla tipologia di progetti reperibili in rete, frutto della ridotta diffusione da cui è penalizzato il programma. Siamo tuttavia fiduciosi che la ancora giovane comunità di utenti Byob possa svilupparsi, in particolare con l'avvento di Snap!, per dare un contributo allo sviluppo ed all'affermazione della programmazione visuale.

Capitolo 11

Bibliografia

11.1 Costruzionismo e Micromondo

http://it.wikipedia.org/wiki/Costruttivismo_%28filosofia%29
http://www.ipbz.it/ImagesUpload/File/Calum_tsang.pdf
http://www.farnt.unito.it/tutorb/Sintesi/costruttivismo.pdf
http://www.mediamente.rai.it/home/bibliote/biografi/p/papert.htm
http://it.wikipedia.org/wiki/Seymour_Papert
http://it.wikipedia.org/wiki/Costruzionismo_%28teoria_dell%27apprendimento%29
http://galileo.cincom.unical.it/professors/libri/gioco/CAPITOLO%20SESTO.htm
www.edscuola.it/archivio/didattica/varisco2.html
www.robocupjr.it/margi/pubblicazioni/1203.pdf

11.2 Logo e Scratch

http://it.wikipedia.org/wiki/Logo_%28informatica%29
http://en.wikipedia.org/wiki/Logo_%28programming_language%29
http://en.wikipedia.org/wiki/Scratch_%28programming_language%29
http://web.media.mit.edu/ jmaloney/papers/ScratchLangAndEnvironment.pdf
http://wiki.scratch.mit.edu/wiki/Recursion
http://scratch.mit.edu/forums

11.3 Byob

http://www.xleroy.net/ByobTuto/Thumbnails.html
http://byob.berkeley.edu
http://byob.berkeley.edu/BYOBManual.pdf