ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

SCUOLA DI PSICOLOGIA E SCIENZE DELLA FORMAZIONE

DIPARTIMENTO DI SCIENZE DELL'EDUCAZIONE "G.M. BERTIN"

CORSO DI LAUREA MAGISTRALE A CICLO UNICO IN SCIENZE DELLA FORMAZIONE PRIMARIA

IL PENSIERO COMPUTAZIONALE ALLA SCUOLA PRIMARIA. UNA PROPOSTA DIDATTICA CON SCRATCH

Tesi di laurea in **Didattica generale e Tecnologie educative**

Relatore

PROF.SSA ELENA PACETTI

Presentata da

FRANCESCO OLIVARI

Correlatore

DOTT. MICHAEL LODI

Anno accademico 2015-2016

INDICE

INTRODUZIONE

Nel corso dell'ultimo decennio il concetto di *pensiero computazionale* – insieme a quello di *coding*, con cui talvolta viene confuso – ha goduto di una fortuna decisamente crescente. Dall'ambito della ricerca (dapprima in campo informatico, quindi anche nelle scienze dell'educazione), il pensiero computazionale è balzato nell'agenda dei decisori politici, ritagliandosi un ruolo significativo nelle politiche scolastiche di molti paesi. Parallelamente ha fatto irruzione sulla stampa generalista: non è raro trovare articoli di quotidiani che presentano una sperimentazione in corso, una testimonianza, un parere più o meno autorevole sul tema. Come sempre accade in queste circostanze, al fermento nel settore della ricerca e alle esperienze sul campo possono sovrapporsi reazioni "emotive", che riproducono la polarizzazione apocalittici-integrati, verso un argomento che è diventato – nel bene e nel male – *à la page*.

Inizialmente mi sono avvicinato alla questione per semplice curiosità. Un momento decisivo, in cui ho cominciato a coltivare l'idea di questa tesi di laurea, è consistito nella mia partecipazione ad alcuni laboratori organizzati dal CoderDojo di Bologna, basati sull'uso di Scratch. Fin da subito ho avuto l'impressione di trovarmi di fronte a un'applicazione molto efficace di principi pedagogici e didattici studiati nel mio percorso universitario, riconducibili al paradigma teorico costruttivista e alla didattica attiva. Vedere bambini "scoprire" funzioni e logiche di programmazione, apprendere concetti e procedure in modo motivato, aiutarsi reciprocamente nella realizzazione di progetti significativi mi ha spinto ad approfondire l'argomento e ad elaborare la proposta presentata in questo lavoro. Ma forse l'aspetto più interessante risiede nel fatto che non sono mai stato un appassionato di informatica e non ho competenze specifiche: mi ritengo un utente "medio" delle tecnologie, anzi con un approccio talvolta un po' timoroso. Se la tematica del pensiero computazionale e le possibili applicazioni didattiche di uno strumento come Scratch hanno suscitato tanto entusiasmo in un "profano" di programmazione informatica, forse può essere una strada per coinvolgere molti altri docenti, come me privi di una preparazione o un interesse pregressi.

Verrà ora illustrato sinteticamente il contenuto dei capitoli di questo lavoro, cercando di mettere in luce il percorso logico lungo cui si snoda.

Il Capitolo 1 prende in esame la letteratura scientifica allo scopo di delimitare il campo teorico e pervenire a una definizione operativa (provvisoria) di *pensiero computazionale*, distinguendolo al contempo dal *coding*, ossia l'attività di programmazione informatica. Il *coding* è un campo privilegiato per l'applicazione, l'insegnamento e l'apprendimento del pensiero computazionale, ma non è l'unico; più in generale, è necessario tenere ben distinti i due concetti, che si riferiscono ad ambiti di ordine differente. Quindi il capitolo apre la visuale a un più ampio dibattito sull'utilizzo delle tecnologie digitali, cercando di argomentare l'opportunità di proporre l'insegnamento del pensiero computazionale fin dalla scuola primaria.

Il Capitolo 2 offre una panoramica delle esperienze e delle iniziative messe in atto negli ultimi anni, sia a livello internazionale sia nel caso specifico dell'Italia. L'attenzione è posta in prevalenza su quelle patrocinate o promosse dalle agenzie educative formali, spesso in collaborazione con organizzazioni o associazioni che vedono la *partnership* di aziende private, istituzioni ed enti di ricerca. Tuttavia non manca uno sguardo a interessanti iniziative di agenzie educative non formali, da cui si possono sviluppare ulteriori collaborazioni con gli Uffici Scolastici Regionali o con singoli istituti.

Per poter declinare l'insegnamento del pensiero computazionale in una proposta didattica coerente e motivata, il Capitolo 3 propone una ricognizione delle principali teorie dell'apprendimento a partire dai primi decenni del Novecento, ponendole in relazione con le applicazioni e gli orientamenti elaborati in ambito didattico. Da un lato si è cercato di restituire un resoconto storiografico complessivo, in cui fossero chiari i principali paradigmi teorici, gli snodi e l'evoluzione dei modelli, tanto nel campo della Psicologia quanto in quello della Didattica. Dall'altro si sono sottolineati e approfonditi maggiormente gli spunti teorici più rilevanti per la proposta avanzata in questo lavoro, riconducibili al paradigma costruttivista.

Il Capitolo 4 è una presentazione "ragionata" di Scratch, l'ambiente di programmazione visuale utilizzato per la proposta didattica avanzata in questo lavoro. Non si tratta di un sintetico "manuale d'uso" del software, bensì di

un’analisi delle caratteristiche e delle funzionalità che presentano implicazioni rilevanti per l’utilizzo didattico dello strumento.

Il Capitolo 5, infine, articola una proposta operativa per l’apprendimento del pensiero computazionale in classi quarte e quinte di scuola primaria, basata sull’utilizzo di Scratch in una prospettiva didattica costruttivista. In riferimento a quanto esposto nei capitoli precedenti, infatti, Scratch presenta caratteri e peculiarità che si coniugano in modo ideale con un apprendimento attivo, incentrato su alcuni aspetti qualificanti: sviluppo di progetti, scoperta, *problem solving*, creatività, collaborazione, co-costruzione della conoscenza. Pertanto la proposta non è disegnata sul modello dell’unità didattica, ma su un approccio attivo/creativo dagli esiti necessariamente più “aperti”. Vengono cioè suggerite e delineate alcune modalità di utilizzo didattico dello strumento, alcune tipologie di attività, sempre motivando le scelte sulla base dei riferimenti teorici, e senza perdere di vista gli obiettivi di apprendimento ricavati dalla definizione operativa di pensiero computazionale. Un aspetto importante è costituito dal ruolo cruciale svolto dalla costante attività di monitoraggio e valutazione *in itinere* da parte del docente.

1. IL PENSIERO COMPUTAZIONALE

1.1 Che cos'è il pensiero computazionale

Nell'ultimo decennio il concetto di *pensiero computazionale* ha suscitato un vivo e crescente interesse, tanto nel campo della ricerca quanto in quello istituzionale, oltre a stimolare una quantità di esperienze e proposte "diffuse". La letteratura è concorde nell'individuare un preciso punto di partenza di questo "movimento culturale": un breve articolo del 2006 di Jeannette Wing, all'epoca direttrice del Dipartimento di Informatica della Carnegie Mellon University.

1.1.1 Il dibattito teorico

Il concetto non è nuovo, anzi è «già presente da decenni, in diverse declinazioni e sotto varie nomenclature» (Lodi 2014, iii). Il primo a utilizzare la locuzione *computational thinking* è stato il matematico e informatico sudafricano Seymour Papert, già nel 1980 nel suo celebre libro *Mindstorms*, e in seguito in un articolo del 1996 sull'insegnamento della matematica con strumenti informatici (Papert 1996). Negli anni Sessanta, lavorando al MIT, Papert ha sviluppato il Logo, un linguaggio di programmazione pensato espressamente per fini didattici. Il suo lavoro si basava su una teoria dell'apprendimento denominata Costruzionismo, una variante del Costruttivismo in cui un ruolo fondamentale è assegnato agli *artefatti cognitivi*, ossia "oggetti" (non necessariamente tangibili) su cui il discente può operare direttamente, in una continua mediazione tra i propri modelli mentali e la "realtà" (si veda il Capitolo 3). In questa cornice teorica, un software o un linguaggio di programmazione può essere appunto un artefatto cognitivo, in grado di facilitare gli apprendimenti in campo matematico e scientifico.

Papert però andava già oltre l'uso del computer in uno specifico contesto disciplinare, assegnando al pensiero computazionale un'autonoma validità euristica e "culturale". In *Mindstorms* (Papert 1980) egli

> afferma che la programmazione favorisce il pensiero procedurale: insegna come spezzare il problema in componenti più semplici e "fare debug" su di esse se non funzionano. Questo modo di pensare, valido nella programmazione, può essere applicato a tutti gli altri aspetti della vita […] in quanto favorisce un modo procedurale di approcciare i problemi (Lodi 2014, 6).

Nonostante abbia illustri precursori, il saggio di Jeannette Wing del 2006 ha comunque avuto il merito di aprire un campo di discussione e di elaborazione molto vivace e fecondo, esplicitamente focalizzato sul sistema educativo, probabilmente perché la diffusione delle tecnologie informatiche e telematiche ha fatto sì che i tempi fossero ormai maturi.

Il contributo di Wing non è né analitico né sistematico, è anzi volutamente "suggestivo". L'autrice elenca una serie di proprietà del pensiero computazionale, o di strumenti e strategie cognitive riferibili a esso, spesso "traducendo" in termini cognitivi costrutti tipici dell'informatica; ma è estremamente netta ed esplicita nell'assegnare un valore generale a questa forma di pensiero: «Il pensiero computazionale è un'abilità fondamentale per tutti, non solo per gli informatici. A leggere, scrivere e calcolare dovremmo aggiungere il pensiero computazionale come abilità di base per ogni bambino» (Wing 2006, 33; traduzione mia).

Il pensiero computazionale non va confuso con la programmazione: «Pensare come un informatico va ben oltre la capacità di programmare un computer. Richiede di pensare a livelli multipli di astrazione» (*ibidem*, 35). E non è nemmeno il modo in cui "pensa" il calcolatore: «[…] è il modo in cui gli esseri umani risolvono i problemi; non è il tentativo di far pensare gli umani come i computer» (*ibidem*, 35), pertanto comprende e anzi valorizza le dimensioni della creatività e dell'immaginazione. In estrema sintesi, il pensiero computazionale concerne «risolvere problemi, progettare sistemi, comprendere il comportamento umano basandosi sui concetti fondamentali dell'informatica» (*ibidem*, 33).

Come detto, dopo questo articolo hanno visto la luce numerosi contributi, principalmente a opera di informatici, con l'obiettivo di articolare le basi teoriche ed epistemologiche su cui poggiare l'insegnamento del pensiero computazionale. Una parte della ricerca ha indagato lo statuto scientifico dell'informatica (Lodi 2014, 3-5), mentre altri studiosi hanno

proposto definizioni di pensiero computazionale, cercando di identificarne i concetti e i processi logico-cognitivi fondamentali.

La necessità di una definizione sistematica non è una questione meramente accademica, anzi scaturisce da un'esigenza decisamente concreta: «è da molti riconosciuta la necessità pratica di una definizione "operativa" di pensiero computazionale, per permettere – nel complesso e altamente burocratizzato sistema dell'educazione pre-universitaria[1] – la definizione di obiettivi educativi e la loro valutazione» (*ibidem,* 8). In altri termini, per poter inserire l'insegnamento del pensiero computazionale nei curricoli scolastici, è indispensabile disporre di una base teorica ed epistemologica condivisa, sulla quale articolare traguardi, obiettivi di apprendimento, strategie didattiche.

1.1.2 Una definizione operativa

Michael Lodi, nella sua tesi di laurea in Didattica dell'Informatica, ha esaminato una cospicua mole di pubblicazioni sull'argomento, giungendo a stilare un elenco dei concetti ricorrenti nelle definizioni di pensiero computazionale analizzate (Lodi 2014, 10):

1. **Collezione e analisi dei dati**. Il processo di raccolta delle informazioni appropriate, e di analisi – per dare loro un senso, trovando pattern comuni e traendo conclusioni dai dati stessi.
2. **Rappresentazione dei dati**. Il processo di rappresentazione e organizzazione di dati e risultati, sia visiva (grafici, testo o immagini) sia astratta (strutture dati).
3. **Decomposizione dei problemi**. Il processo di divisione del problema in parti più piccole e affrontabili.
4. **Astrazione**. Il processo di riduzione della complessità, per far emergere l'idea principale mantenendo solo alcuni aspetti e tralasciandone altri.
5. **Generalizzazione e riconoscimento di pattern**. L'abilità di riconoscere come alcune parti di soluzione possono essere riusate nella stessa o riapplicate a problemi simili.
6. **Algoritmi**. Una serie ordinata di passi per risolvere un problema o raggiungere un obiettivo.

[1] Nel caso dell'Italia, si intende l'insieme dei cicli che vanno dalla scuola dell'infanzia alla fine della secondaria di secondo grado (quello che in ambito anglosassone viene indicato con la sigla *K-12*: dal *Kindergarten* al dodicesimo grado di istruzione).

7. **Automazione**. Lasciare ad una macchina i compiti ripetitivi o noiosi, formalizzandoli e facendoglieli eseguire.
8. **Simulazione, test, debug**. Modellare un processo ed eseguire esperimenti su di esso. Individuare problemi/errori e correggerli.
9. **Parallelizzazione**. Organizzare risorse per far loro eseguire task simultanei allo scopo di raggiungere un obiettivo comune.
10. **Complessità e calcolabilità**. Individuare un metodo che raggiunga un risultato, possibilmente il migliore e usando meno risorse (tempo, memoria, potenza di calcolo, energia).

Assumendo come base questi concetti, e rielaborando la definizione operativa proposta dalla International Society for Technology in Education (ISTE) e dalla Computer Science Teachers Association (CSTA), Lodi propone quindi la seguente, ripresa (con differenze trascurabili) anche nel sito web[2] di "Programma il Futuro" (*ibidem*, 11):

> Il *pensiero computazionale* è un processo di problem-solving che consiste nel:
> - formulare problemi in una forma che ci permetta di usare un *computer* (nel senso più ampio del termine, ovvero una macchina, un essere umano, o una rete di umani e macchine[3]) per risolverli;
> - organizzare logicamente e analizzare dati;
> - rappresentare i dati tramite astrazioni, modelli e simulazioni;
> - automatizzare la risoluzione dei problemi tramite il pensiero algoritmico;
> - identificare, analizzare, implementare e testare le possibili soluzioni con un'efficace ed efficiente combinazione di passi e risorse (avendo come obiettivo la ricerca della soluzione migliore secondo tali criteri);
> - generalizzare il processo di problem-solving e trasferirlo ad un ampio spettro di altri problemi.

Lodi elenca poi una serie di competenze specifiche, di (pre)requisiti e di possibili strategie didattiche, o modalità di gestione dell'ambiente di apprendimento che possono favorire lo sviluppo del pensiero computazionale. Si tratta di un'articolazione espressamente orientata all'elaborazione di un curricolo di insegnamento.

In seguito Lodi ha rielaborato queste sue proposte, giungendo a una definizione operativa maggiormente articolata[4], che distingue tre livelli.

---

[2] http://www.programmailfuturo.it/progetto/cose-il-pensiero-computazionale

[3] In generale, un "agente che processa informazioni".

- *Concetti* computazionali, tipici dei linguaggi dell'informatica:
  - Sequenze
  - Condizionali
  - Ripetizioni
  - Eventi
  - Parallelismo
  - Operatori
  - Dati
- *Pratiche* messe in atto nell'attività di programmazione:
  - Essere incrementali e iterativi
  - *Testing* e *debugging* (procedere per prove ed errori)
  - Riuso e *remixing*
  - Decomposizione
  - Astrazione
  - Riconoscimento di *pattern* e generalizzazione
  - Automazione
  - Simulazione
  - Attenzione all'efficienza, calcolabilità e complessità
- *Prospettive* generali – su sé stessi e sul mondo – promosse nei discenti:
  - Esprimere se stessi
  - Connettersi
  - Farsi domande
  - Saper gestire la complessità e i problemi difficili
  - Tolleranza per l'ambiguità e i problemi aperti

Questa articolazione riprende e amplia un *framework* per l'insegnamento del pensiero computazionale con Scratch sviluppato dalla Harvard Graduate School of Education (Brennan, Balch & Chung 2014), che verrà presentato nel Capitolo 5 perché sarà la cornice di riferimento della proposta didattica che si intende avanzare.

[4] In due testi in corso di pubblicazione: Marchignoli, R., & Lodi, M., *EAS e pensiero computazionale*, Bescia: La Scuola; e Lodi, M., Prefazione a Giordano, M., & Moscetti, C.,, *Coding e Pensiero computazionale. Materiali per l'insegnante*, Loreto: ELI – La Spiga.

1.2 Pensiero computazionale e *coding*

Se la letteratura offre un vasto panorama di contributi teorici volti a definire il pensiero computazionale, lo stesso non accade per il concetto di *coding*, che non sembra suscitare problemi o ambiguità. Con tale termine si intende l'attività di programmazione[5], ossia la "scrittura" del codice di un programma informatico. La relazione tra *coding* e pensiero computazionale è dunque quella esistente tra un ambito operativo/applicativo e il complesso di costrutti concettuali soggiacenti, un po' come il disegno di figure geometriche e la dimostrazione di teoremi sono campi di applicazione del pensiero geometrico (in letteratura si possono trovare vari esempi simili, riferiti alle discipline tradizionali).

In quanto complesso di competenze e strategie cognitive, il pensiero computazionale non è necessariamente legato all'attività di programmazione informatica; come si vedrà nel prossimo paragrafo, è possibile rintracciare esempi di pensiero computazionale nella vita quotidiana, così come progettare attività volte a svilupparlo senza l'uso del computer.

Tuttavia è innegabile che il *coding* costituisca un campo di applicazione privilegiato. L'essenziale, dal punto di vista educativo, è avere ben chiaro il rapporto tra le due sfere: l'obiettivo è promuovere e sviluppare il pensiero computazionale, *non* scrivere un programma che funzioni; quest'ultimo è semplicemente un mezzo didattico (che presenta indubbi vantaggi) per arrivare a un traguardo di ordine culturale. Come scrisse già nel 1968 George Forsythe, fondatore del Dipartimento di Informatica della Stanford University:

> Le acquisizioni più valide nell'educazione scientifica e tecnologica sono quegli strumenti mentali di tipo generale che rimangono utili per tutta la vita. Ritengo che il linguaggio naturale e la matematica siano i due strumenti più importanti in questo senso, e l'informatica sia il terzo (cit. in Nardelli 2015).

Prima di illustrare brevemente due approcci "di confine" rispetto all'oggetto di questo lavoro, è opportuna una precisazione. Esiste una grande varietà di linguaggi informatici, ma tutti presentano una dimensione che

[5] In questo caso il ricorso al vocabolo inglese è giustificato dal vastissimo impiego dell'italiano *programmazione* nel campo della progettazione didattica, che potrebbe generare confusione.

comporta difficoltà eccessive per studenti della scuola primaria: la sintassi testuale. Perciò, quando si parla di *coding* per questa fascia di età, ci si riferisce implicitamente all'utilizzo di software o *ambienti di programmazione visuali*, in grado di aggirare lo scoglio della sintassi, come nel caso di Scratch. A livello pratico, in questo caso programmare non significa dunque "scrivere il codice", bensì "comporlo" unendo blocchetti appositamente predisposti, eventualmente modificando determinati parametri al loro interno.

1.2.1 Pensiero computazionale senza *coding*

A riprova del fatto che il binomio pensiero computazionale-*coding* non è indissolubile, ma esprime una relazione (senz'altro forte) tra due oggetti che hanno statuto diverso, esistono approcci al pensiero computazionale che non prevedono l'uso di un calcolatore.

Un esempio – per la verità rintracciabile all'interno di trattazioni teoriche, ma più raramente declinato in concrete proposte didattiche – riguarda l'individuazione delle strategie di pensiero computazionale normalmente impiegate nella vita quotidiana. Ogni giorno qualunque persona utilizza, in modo intuitivo e "naturale", modalità di pensiero che, se analizzate da un punto di vista formale, si rivelano tipiche della computazione; esempi in questo senso erano già elencati nell'articolo di Wing (2006), e altri si trovano facilmente in letteratura: dalla semplice osservazione che eseguire una moltiplicazione significa iterare una somma, alla scelta della fila al supermercato, alla possibile descrizione in termini algoritmici di molte azioni quotidiane, come preparare la valigia o montare un mobile.

Chiaramente si tratta di un argomento forte a favore dell'insegnamento del pensiero computazionale: se è una forma di pensiero già così presente nella nostra vita – indipendentemente dall'utilizzo delle tecnologie informatiche –, è facile convenire sull'importanza di inserirlo nel curricolo scolastico[6].

[6] A livello di proposta operativa, esplicitare questa formalizzazione, ragionando su tali procedure in prospettiva metacognitiva, potrebbe comunque essere un esercizio interessante all'interno di un curricolo di insegnamento del pensiero computazionale, anche se difficilmente alla portata di alunni di scuola primaria.

Una proposta didattica strutturata è invece *Computer Science Unplugged*, una guida elaborata dal CS Education Research Group dell'Università di Canterbury (Nuova Zelanda) e liberamente scaricabile con licenza Creative Commons. Si tratta di una raccolta di attività di tipo ludico, corredata di tutti i materiali necessari in formato stampabile, per l'apprendimento di concetti informatici senza l'uso del calcolatore. Attraverso giochi sfidanti realizzabili semplicemente con carta, matita e con il corpo, vengono affrontati argomenti come il codice binario, la compressione dei dati, gli algoritmi di ricerca e di ordinamento impiegati nei calcolatori.

È evidente la valenza "democratica" di questa proposta, soprattutto in ambito educativo: la possibilità di svincolare, almeno a un livello di base, l'insegnamento del pensiero computazionale dalla disponibilità di risorse strumentali (non ancora presenti in tutte le scuole) è decisiva per non creare un nuovo modello di esclusione e contrastare il *digital divide*.

Non a caso l'approccio *unplugged* è una modalità prevista anche nella piattaforma Code.org, utilizzata per il progetto "Programma il futuro" del MIUR, che verrà analizzato nel Capitolo 2. Tutte le attività previste in questa articolata proposta didattica sono declinate sia nella versione definita "tecnologica" (che richiede PC e connessione Internet), sia nella versione "tradizionale", fruibile con carta, matita e altri strumenti di normale cancelleria. Si tratta di attività più chiaramente riconducibili al pensiero computazionale, mentre la proposta di *CS Unplugged* risulta un po' legata a un approccio di tipo teorico all'informatica, centrato su aspetti quali codice binario, compressione dei dati, efficienza degli algoritmi.

1.2.2 Territori limitrofi: la robotica educativa

Si impone a questo punto una breve digressione su un campo "contiguo", che non rientra nell'oggetto di questo lavoro, ma che è oggi in gande espansione e suscita l'interesse crescente del mondo della scuola. La *robotica educativa* consiste nella programmazione (eventualmente preceduta dalla costruzione o dall'assemblaggio) di robot, attuata in un contesto didattico; con il termine *robot* si indica una macchina in grado di eseguire compiti in modo autonomo.

Il mercato rivolto alle scuole è oggi particolarmente vivace, con nuovi prodotti e interessanti *startup* che, affiancandosi a realtà già consolidate (per esempio i set di LEGO Educational), compongono un ricco panorama di proposte per tutto l'arco della scolarità, dall'infanzia alla secondaria di primo e secondo grado (dove si affianca a un ulteriore settore "limitrofo": l'elettronica educativa). L'attenzione delle istituzioni scolastiche verso questo settore è testimoniata dal recente Piano Nazionale Scuola Digitale, in particolare dall'Azione #7, che riguarda le dotazioni degli *atelier creativi*[7].

La connessione con il *coding* è molto stretta, in un certo senso "costitutiva": tutti gli strumenti per la robotica educativa sono programmabili, alcuni direttamente *on board*, ossia con tasti e comandi posti sul *device* stesso (come per esempio Bee-Bot, robot a forma di ape che è diventato una piccola "celebrità" per le scuole dell'infanzia che si cimentano con la robotica educativa), altri tramite un software proprietario[8] per tablet, smartphone o PC, o in alternativa anche con programmi *open source* come lo stesso Scratch. È questo il caso della ricca – e molto nota – gamma di prodotti LEGO Education, tra cui LEGO WeDo e LEGO Mindstorms[9], ormai un "classico" del settore. I bambini costruiscono robot o piccoli macchinari utilizzando i mattoncini LEGO, corredati di elementi quali motori elettrici, ingranaggi, sensori; quindi, ultimata la realizzazione fisica dell'oggetto, passano alla programmazione del movimento tramite un software visuale, che come detto può essere lo stesso Scratch.

Per gli scopi del presente lavoro non è possibile affrontare il campo vasto e articolato della robotica educativa. Tuttavia è utile evidenziare, di passaggio, un punto forte che caratterizza l'approccio al pensiero computazionale realizzabile con questi strumenti.

La robotica educativa rende ancora "più concreto" l'effetto del codice: il risultato della programmazione non è più visualizzato su uno schermo, ma si traduce in azioni nel mondo fisico. È vero che un programma informatico è già di per sé un *artefatto cognitivo*, secondo l'accezione proposta da Seymour

[7] Vi si farà cenno nel Capitolo 2.

[8] Di solito questi software condividono molte delle caratteristiche di Scratch: sono ambienti di programmazione visuali a blocchetti, progettati per la fruizione di bambini dai 6-7 anni di età.

[9] Il nome è un omaggio al fondamentale libro di Papert (1980) cui si è accennato in precedenza.

Papert nell'ambito della teoria dell'apprendimento chiamata Costruzionismo[10]: sarebbe ingenuo postulare una contrapposizione tra computer come macchina del "virtuale" (termine quanto mai frainteso, spesso erroneamente associato ad "astratto") e modellini fisici come macchine concrete. In altri termini, lo schermo del computer e il programma che viene eseguito dopo essere stato scritto dal bambino sono essi stessi strumenti *concreti*, in grado di funzionare come artefatti cognitivi, specialmente nell'attuale "civiltà dell'immagine", in cui schermi e *device* digitali sono onnipresenti nell'esperienza quotidiana di adulti e bambini. Tuttavia è innegabile che la robotica permetta un salto ulteriore, connettendo codice, programmazione, computazione ad azioni svolte da un oggetto fisico nello spazio fisico. Un passaggio particolarmente interessante ed efficace proprio perché mette in evidenza questo collegamento, mostrando le possibili applicazioni "pratiche", tangibili del software.

Ciò comporta un ulteriore vantaggio nel contesto educativo: se già programmare un videogioco o una storia interattiva è un'attività che si giova dell'interesse e della motivazione da parte dei bambini, a maggior ragione il "balzo" nel mondo dei robot suscita solitamente un entusiasmo che ha profonde ricadute positive sugli apprendimenti, come attestano le ricerche in psicologia cognitiva (si veda il Capitolo 3).

1.3 Insegnare il pensiero computazionale nella scuola primaria

In una realtà sempre più caratterizzata da "ubiquitous computing" (Wing, 2006, p. 33), oggi probabilmente ben pochi contesterebbero l'opportunità (o meglio la necessità) di inserire un corso di programmazione – strutturato e orientato allo sviluppo del pensiero computazionale – all'interno del curricolo della scuola secondaria. Ma perché fin dalla primaria, o addirittura dalla scuola dell'infanzia?

Una posizione molto decisa in questo senso, come visto sopra, deriva dall'articolo di Wing del 2006, che già proponeva con convinzione il pensiero computazionale come quarta abilità di base, in una visione in un certo senso

[10] Se ne tratterà diffusamente nel Capitolo 3.

"mcluhaniana": le tecnologie (la stampa, la televisione, il computer, i media telematici) influenzano in modo determinante non solo le abitudini di vita, ma le stesse capacità e i processi cognitivi dell'uomo; alfabetizzare al pensiero computazionale sarebbe la logica conseguenza dell'impatto dei mezzi informatici sulle nostre strutture concettuali e cognitive.

Anche lasciando sullo sfondo la questione del pensiero computazionale come nuova *literacy* di base (che incontra sempre più sostenitori), si può proporre un ragionamento molto semplice a sostegno del suo insegnamento precoce: se viene riconosciuta la rilevanza di questa "forma di pensiero" per la formazione dei cittadini di domani, il suo inserimento nei curricoli di studio dipende solo dalla possibilità di adattare l'insegnamento alle capacità cognitive dei bambini. Un po' come è avvenuto nei secoli scorsi con discipline quali biologia, chimica e fisica: attestata l'importanza di una competenza di base *per tutti* in queste materie (cioè al di là di possibili sbocchi professionali), esse sono state introdotte non solo alle scuole medie e superiori, ma anche alle elementari, con l'opportuna mediazione didattica in relazione all'età dei discenti (Nardelli 2004).

Per approfondire queste semplici considerazioni, verranno ora analizzati alcuni "ambiti di discorso" molto differenti (ricerca, divulgazione, documenti ufficiali di istituzioni italiane ed europee), da cui è possibile ricavare indicazioni concordi sull'importanza dell'insegnamento del pensiero computazionale fin dai primi anni di scolarità.

1.3.1 Il dibattito sui "nativi digitali"

La locuzione *nativi digitali*, negli ultimi quindici anni, è stata una delle *buzzword* più citate e discusse, nel contesto dell'enorme impatto prodotto dalla diffusione capillare della Rete e dei media telematici nella vita quotidiana. Come sempre accade in momenti percepiti come svolte epocali a livello tecnologico, è naturale che etichette sintetiche ed efficaci, in grado di "condensare" un tema complesso in modo semplice (e spesso semplicistico…), possano godere di una grande fortuna nel discorso mediatico. Ed è altrettanto comune che suscitino polarizzazioni opposte, che rimandano alle categorie degli "apocalittici" e degli "integrati", per citare due etichette

almeno altrettanto fortunate, coniate da Umberto Eco proprio in ambito massmediologico.

La locuzione è stata introdotta da Marc Prensky in un articolo del 2001, insieme a quella correlata di *immigrati digitali*, utilizzando una metafora mutuata dall'apprendimento della lingua madre (Prensky 2001). La dicotomia è basata su uno spartiacque generazionale tra coloro che sono nati in un contesto caratterizzato dalla presenza pervasiva delle tecnologie digitali e della Rete, e coloro che hanno dovuto imparare a utilizzare queste tecnologie in età adulta. La distinzione si fonda su un presupposto forte: crescere "immersi" nelle tecnologie digitali influenza profondamente le strutture cognitive dei bambini, che dunque "pensano" in un modo radicalmente differente dalle generazioni precedenti. Si tratterebbe di una vera e propria mutazione antropologica, di amplissima portata:

> Gli studenti di oggi rappresentano la prima generazione cresciuta con queste tecnologie. Hanno passato tutta la loro vita utilizzando ed essendo circondati da computer, videogiochi, lettori musicali digitali, videocamere, telefoni cellulari […].
>
> Ora è chiaro che, come risultato dell'esposizione permanente a questo contesto e della continua interazione con esso, gli studenti di oggi *pensano e processano l'informazione in modo radicalmente differente* dai loro predecessori. Tali differenze vanno molto oltre e molto più in profondità di quanto sospetti o riconosca la maggior parte degli educatori. […] è molto probabile che *i cervelli dei nostri studenti siano cambiati dal punto di vista fisico* – e siano diversi dai nostri – come risultato del modo in cui essi sono cresciuti. Ma anche se questo non fosse *letteralmente* vero, possiamo affermare con certezza che i loro *schemi di pensiero* [*thinking patterns*] sono cambiati[11] (Prensky 2001, 1).

Prensky elenca quindi una serie di abilità, o meglio "attitudini cognitive", che caratterizzerebbero i nativi digitali: l'abitudine a un accesso molto rapido all'informazione, il *multitasking*, la preferenza per le immagini rispetto al testo scritto, la fruizione non lineare, sulla base del modello dell'ipertesto.

Uno degli aspetti interessanti dell'articolo di Prensky è che il *focus* è fin da subito sulle conseguenze per il sistema educativo; lo scenario delineato, con

[11] Traduzione mia. I corsivi sono dell'Autore.

evidente preoccupazione, è quello di una totale inadeguatezza, una sorta di "incompatibilità cognitiva": «I nostri insegnanti immigrati digitali, che parlano una lingua obsoleta (quella dell'era pre-digitale), cercano disperatamente di insegnare a una popolazione che parla un linguaggio totalmente nuovo» (*ibidem*, 2).

Le tesi di Prensky sono state approfondite in Italia da Paolo Ferri, docente del dipartimento di Scienze della Formazione all'Università di Milano Bicocca, in un libro del 2011 intitolato *Nativi digitali*. Ferri, riprendendo le idee di Henry Jenkins, studioso statunitense che si è occupato delle "culture emergenti" dalla diffusione dei nuovi media, elenca una serie di attitudini o comportamenti cognitivi che connotano i nativi digitali:

> Gioco, simulazione, *performance*, appropriazione, *multitasking*, conoscenza distribuita, intelligenza collettiva, giudizio critico, navigazione transmediale, *networking*, negoziazione: sono queste le caratteristiche specifiche delle nuove forme di appropriazione comunicativa dei media digitali che vengono sviluppate dai bambini e dai preadolescenti (ma anche dai teenager) del nuovo millennio (Ferri 2011, 56).

E conclude: «La nostra opinione, in proposito, è piuttosto radicale: i nativi digitali esistono e la loro differenza specifica è l'*intelligenza digitale*» (*ibidem*, 80; corsivo mio), collegandosi esplicitamente alla teoria delle intelligenze multiple di Howard Gardner (si veda il Capitolo 3) e proponendo l'integrazione di questo ulteriore stile cognitivo, tipico dei media telematici.

La posizione originale di Prensky ha scatenato un acceso dibattito, che ha coinvolto studiosi di vari settori delle scienze umane (scienze dell'educazione, antropologia, sociologia, filosofia), informatici, massmediologi, oltre a giornalisti più o meno specializzati. Trattandosi di una posizione assai netta, in cui è evidente una forma di determinismo tecnologico "duro", molti studiosi l'hanno sottoposta a critiche serrate, che si sono concentrate su alcuni aspetti principali, analizzati da Antonio Fini (2011) in un'ottima sintesi del dibattito internazionale.

- Anzitutto pare eccessiva l'enfasi posta sul fattore anagrafico come unico discrimine tra nativi e immigrati digitali. La "frattura generazionale" non tiene conto di una pluralità di questioni, dal

digital divide (ineguale accesso alle tecnologie e alla connettività) alle differenze intragenerazionali, che da alcuni studi risultano essere non così dissimili da quelle intergenerazionali; anche gli adulti, fra l'altro, possono sviluppare competenze digitali avanzate e "diventare nativi" (*going natives*).

- Non ci sono ricerche che avvalorino l'ipotesi della "mutazione antropologica", che sembra essere più un postulato che non una teoria scientifica. In altri termini, se è ragionevole sostenere che, nel tempo, l'esperienza pervasiva dell'uso dei media digitali produrrà *una qualche* modifica nelle strutture e nei processi cognitivi umani, i caratteri di tale evoluzione andranno indagati nel dettaglio.
- L'aspetto probabilmente cruciale, particolarmente interessante ai fini del presente lavoro, riguarda l'analisi più attenta delle competenze digitali dei (supposti) nativi. Molti hanno iniziato a chiedersi se la familiarità con le tecnologie informatiche e telematiche produca di per sé una forma di competenza, o se il modello di utilizzo da parte dei giovani e giovanissimi non ricalchi piuttosto un uso passivo di ambienti e sistemi percepiti come "trasparenti", in uno scenario in cui le interfacce sono diventate sempre più *user-friendly* e non richiedono nessuna conoscenza a livello dell'architettura soggiacente. I bambini e gli adolescenti che passano ore sui social network, con i videogiochi o sui canali YouTube sarebbero cioè utilizzatori passivi di strumenti di cui non conoscono il funzionamento, semplicemente perché non è affatto necessario per i loro scopi ludici e comunicativi.

Sempre più studiosi e osservatori concordano con quest'ultima posizione, avvalorata da recenti ricerche, tra cui una condotta dall'Università di Milano Bicocca (Gui 2013). In ambito italiano, per esempio, hanno sempre espresso forti perplessità sulle competenze dei "nativi digitali" (e dunque sulla validità stessa della categoria) Pier Cesare Rivoltella, docente all'Università Cattolica di Milano, e Paolo Attivissimo, che con piglio polemico sposta il discorso anche sulla "chiusura" dei sistemi e dei software proprietari:

> […] poiché usano dispositivi che si connettono in modo trasparente, invisibile, [i nativi digitali] non percepiscono Internet come un'infrastruttura di base alla quale ci si deve prima collegare per poter fare qualcosa. […]
>
> Con pochissime eccezioni, non hanno la più pallida idea di come funzionino realmente i dispositivi che usano. […] stanno crescendo in un mondo nel quale non solo non sanno, ma non possono smontare, *smanettare*, sperimentare, in parole povere diventare hacker, nell'accezione originale, positiva e sempre più spesso dimenticata, di questo termine. […]
>
> I dati indicano che stiamo rinunciando progressivamente agli elementi tecnici fondamentali che hanno permesso lo sviluppo della Rete, sostituendoli con un ecosistema hardware e software progressivamente sempre più chiuso. La mia preoccupazione è che tutto questo non crea nativi digitali. Crea polli di batteria (Attivissimo 2013).

Lo stesso Prensky, in un articolo del 2009, ha profondamente rivisto la propria posizione iniziale, integrando le critiche più fondate e convincenti e proponendo un nuovo concetto, quello di "saggezza digitale" (*digital wisdom*), in cui le abilità di utilizzo delle tecnologie si coniugano con la consapevolezza su *come* le tecnologie vanno usate in modo responsabile:

> Quello della saggezza digitale è un concetto duplice, poiché si riferisce sia alla saggezza derivante *dall*'utilizzo delle tecnologie digitali per accedere a potenzialità cognitive al di là della nostra capacità innata, sia alla saggezza *nell*'utilizzo prudente della tecnologia per accrescere le nostre capacità (Prensky 2009, 1).

Chiunque può essere o diventare un "saggio digitale", a prescindere dall'età: Prensky dunque abbandona definitivamente la distinzione basata sull'appartenenza generazionale e rivolge l'attenzione al tipo di competenze digitali in campo. Pier Cesare Rivoltella, discutendo l'articolo di Prensky, introduce altre due categorie accanto a quella del saggio digitale:

> b) quella dello smanettone digitale (*digital skillness*). È colui che possiede le competenze tecniche già attribuite al nativo: rapido, esperto, dotato di grande dimestichezza rispetto ai diversi supporti;
>
> c) quella dello stupido digitale (*digital stupidity*). È colui che delle tecnologie fa usi impropri, dannosi, trasgressivi; o anche colui che rifiuta a priori di avvicinarsi ad esse ritenendole fonte di tutti i mali (Rivoltella 2010).

E conclude con un collegamento esplicito non solo al tema della *media education*, ma soprattutto alla sua inclusione tra le competenze chiave di cittadinanza, un punto fondamentale per il nostro discorso:

> Non mi sembra vi sia molta differenza tra la saggezza di cui Prensky parla e l'obiettivo che da decenni la Media Literacy si propone: responsabilità, senso critico, consapevolezza nell'uso dei media sono da sempre "nel mirino" di un movimento vastissimo e con una tradizione enorme.
>
> Sicuramente la saggezza digitale corrisponde a quell'idea di competenza digitale cui la Comunità Europea pensa quando la indica all'interno del framework delle competenze di cittadinanza.
>
> Infine, proprio prendendo spunto da quest'ultimo cenno, mi pare che la competenza digitale (la saggezza digitale) costituisca oggi un problema importante non solo dell'educazione ai media digitali, ma dell'educazione alla cittadinanza *tout court* (*ibidem*).

Riassumendo, il dibattito sui nativi digitali, partito da posizioni iniziali molto rigide e "schierate", ha avuto il merito di cercare di articolare via via il campo delle competenze digitali. Un ulteriore sforzo della ricerca in questa direzione verrà discusso nel prossimo paragrafo; qui è opportuno volgere di nuovo l'attenzione alle implicazioni in ambito educativo.

Il superamento della contrapposizione rigida tra nativi e immigrati digitali permette di uscire dall'*impasse* dovuta alla presunta incomunicabilità tra i due gruppi: non è vero che gli alunni di oggi hanno un linguaggio e delle strutture cognitive radicalmente *altre*, fuori dalla portata delle strategie didattiche scolastiche. D'altro canto è pur vero che le tecnologie digitali costituiscono una parte rilevante dell'esperienza di bambini e ragazzi, e propongono modalità di accesso alle informazioni, di condivisione, di cooperazione che hanno peculiarità interessanti; negare la frattura nativi/immigrati non significa risolvere la questione escludendo la portata delle tecnologie nel nostro modo di apprendere, conoscere, interagire. In definitiva, per gli insegnanti cade l'alibi dell'incomunicabilità e rimane aperta la sfida (stimolante) di integrare il digitale negli ambienti di apprendimento, utilizzando le tecnologie come strumenti, con precise caratteristiche e potenzialità, da inserire in una consapevole progettazione didattica ed educativa. E alcuni degli obiettivi dovranno essere *interni* alla sfera stessa delle tecnologie (Terravecchia 2013), in un'ottica di *media education* che miri

a condurre i bambi dalla *digital skillness* alla *digital wisdom*. In questo contesto un ruolo centrale può essere svolto dall'insegnamento del pensiero computazionale.

1.3.2 Le competenze digitali e il concetto di *fluency*

Riprendendo il tentativo di articolare in modo analitico le competenze relative all'uso delle tecnologie digitali, Michael Lodi (2014, 2) trae dalla letteratura una classificazione di abilità e conoscenze, riferita al campo specifico dell'informatica:

- la prima è la capacità di utilizzare programmi applicativi di base (editor, browser, file system...), che chiameremo **alfabetizzazione informatica** (*computer literacy*);
- la seconda è una comprensione generale del funzionamento di un sistema informatico, chiamata **padronanza informatica** (*computer fluency*);
- la terza è l'insieme di strumenti intellettuali e critici che un professionista ha bisogno di padroneggiare per poter utilizzare le metodologie o le applicazioni informatiche per affrontare i problemi della propria disciplina (scienze fisiche, biologiche, sociali, materie umanistiche e arte), che potremmo chiamare **pensiero computazionale** (*computational thinking*).

Il primo concetto riguarda le abilità di base nell'uso degli strumenti informatici, ben sintetizzate dal termine *alfabetizzazione*. Gli altri due livelli articolano competenze più elevate, relative da un lato alle conoscenze sul funzionamento del sistema (oltre l'interfaccia), dall'altro agli strumenti intellettuali, compiendo dunque un'astrazione dal piano concreto delle applicazioni e dell'utilizzo degli strumenti a quello dei concetti soggiacenti.

È interessante notare che il termine *fluency* viene utilizzato con insistenza in un TED Talk tenuto nel novembre del 2012 da Mitchel Resnick[12], allievo di Papert e direttore del Lifelong Kindergarten del MIT Media Lab, il dipartimento in cui è stato progettato e sviluppato Scratch. In questa breve conferenza Resnick esprime la propria soddisfazione per le modalità con cui milioni di bambini e ragazzi utilizzano Scratch, mettendola in contrapposizione con l'uso passivo delle tecnologie da parte dei presunti nativi

12 https://www.ted.com/talks/mitch_resnick_let_s_teach_kids_to_code?language=it#t-268528

digitali. La sua critica di questo concetto è perfettamente in linea con quanto esposto nel paragrafo precedente: la stragrande maggioranza dei "nativi" si limita in realtà a un utilizzo passivo, acritico e ripetitivo delle tecnologie; l'obiettivo del suo lavoro e del team del Lifelong Kindergarten è proprio quello di promuovere un approccio radicalmente diverso, creativo ed espressivo. Resnick utilizza il concetto di *fluency* in analogia con il campo semantico di provenienza del termine, quello linguistico: una persona diventa *fluent* in una lingua quando è in grado di utilizzarla non solo per scambiare informazioni, ma anche per esprimere se stessa, il proprio pensiero e la propria personalità; analogamente, la *fluency with technologies* è la capacità di utilizzarle in modo espressivo e creativo, dunque con un'accezione abbastanza diversa da quella esposta sopra a proposito della competenza informatica *tout court*.

Il passaggio successivo del ragionamento di Resnick chiama in causa direttamente l'attività di *coding*: programmare, in un ambiente appositamente progettato come Scratch, è una via privilegiata per promuovere nei bambini un tale uso creativo, espressivo e critico delle tecnologie.

Come si vedrà nel prosieguo di questo lavoro, l'enfasi sulla creatività come aspetto caratterizzante della competenza digitale è centrale nell'approccio al *coding* e al pensiero computazionale sviluppato dal MIT Media Lab.

1.3.3 Pensiero computazionale e competenze chiave di cittadinanza

Alla luce di quanto esposto fin qui, sulla scorta della definizione operativa di pensiero computazionale e delle ulteriori indicazioni emerse (che alla dimensione cognitiva delle strategie di *problem solving* connettono la sfera della creatività), è molto interessante analizzare un documento emanato ormai quasi un decennio fa dal MIUR, per evidenziare il ruolo che l'insegnamento del pensiero computazionale può rivestire nella formazione dei nuovi cittadini.

Il punto di partenza è la Raccomandazione del Parlamento europeo e del Consiglio, del 18 dicembre 2006 (2006/962/CE), relativa alle *competenze chiave per l'apprendimento permanente*. Come noto, si tratta di un documento

fondamentale, che si inserisce nella cosiddetta "strategia di Lisbona", un ampio programma di riforme che aveva l'obiettivo di fare dell'Europa "l'economia basata sulla conoscenza più competitiva e dinamica del mondo". Nello scenario della società della conoscenza, si identificavano dunque otto competenze fondamentali [13] che tutti i cittadini dovrebbero acquisire e sviluppare, per inserirsi con successo in un contesto sociale e professionale sempre più caratterizzato da dinamismo, interconnessione, flessibilità e formazione continua (*lifelong learning*)[14].

Basandosi su tale documento, nel 2007 il MIUR, con il "Regolamento recante norme in materia di adempimento dell'obbligo di istruzione" (decreto 22 agosto 2007, n. 139), ha delineato gli obiettivi da conseguire al termine del ciclo di istruzione obbligatoria[15], articolati in quattro *assi culturali* e in otto *competenze chiave di cittadinanza*. Gli assi culturali (dei linguaggi; matematico; scientifico-tecnologico; storico-sociale) sono i macro-ambiti in cui si organizzano gli apprendimenti. Le competenze chiave sono invece il risultato che si può conseguire attraverso la reciproca integrazione tra i saperi caratteristici degli assi culturali; si tratta dunque di competenze trasversali, non disciplinari, fortemente orientate al *lifelong learning*. È opportuno riportarle per esteso.

- **Imparare ad imparare**: organizzare il proprio apprendimento, individuando, scegliendo ed utilizzando varie fonti e varie modalità di

[13] Le otto competenze chiave del quadro di riferimento europeo sono: 1. comunicazione nella madrelingua; 2. comunicazione nelle lingue straniere; 3. competenza matematica e competenze di base in scienza e tecnologia; 4. competenza digitale; 5. imparare a imparare; 6. competenze sociali e civiche; 7. spirito di iniziativa e imprenditorialità; 8. consapevolezza ed espressione culturale.

[14] A questo proposito, si segnala anche un importante *framework* recentemente elaborato, in ambito statunitense, da P21(Partnership for 21st Century Learning), un'associazione che si propone di mettere a punto linee guida molto generali per la formazione dei cittadini e l'apprendimento lungo tutto l'arco della vita, in collaborazione con enti governativi, sistema educativo, mondo dell'impresa e altri partner. Tra le quattro macroaree identificate come cruciali per la realizzazione personale e professionale nella società della conoscenza, una riguarda *Information, media and technology skills* e comprende *Information literacy*, *Media literacy* e *ICT literacy*, mentre un'altra è relativa a *Learning and innovation skills*, declinate in Pensiero critico, Collaborazione, Comunicazione e Creatività (http://www.p21.org/our-work/p21-framework).

[15] Anche se non si ritrovano espressi negli stessi termini, questi traguardi generali fanno inoltre da "sfondo" alle *Indicazioni nazionali per il curricolo* (MIUR 2012).

informazione e di formazione (formale, non formale ed informale), anche in funzione dei tempi disponibili, delle proprie strategie e del proprio metodo di studio e di lavoro.

- **Progettare**: elaborare e realizzare progetti riguardanti lo sviluppo delle proprie attività di studio e di lavoro, utilizzando le conoscenze apprese per stabilire obiettivi significativi e realistici e le relative priorità, valutando i vincoli e le possibilità esistenti, definendo strategie di azione e verificando i risultati raggiunti.
- **Comunicare**:
  - comprendere messaggi di genere diverso (quotidiano, letterario, tecnico, scientifico) e di complessità diversa, trasmessi utilizzando linguaggi diversi (verbale, matematico, scientifico, simbolico, ecc.) mediante diversi supporti (cartacei, informatici e multimediali);
  - rappresentare eventi, fenomeni, principi, concetti, norme, procedure, atteggiamenti, stati d'animo, emozioni, ecc. utilizzando linguaggi diversi (verbale, matematico, scientifico, simbolico, ecc.) e diverse conoscenze disciplinari, mediante diversi supporti (cartacei, informatici e multimediali).
- **Collaborare e partecipare**: interagire in gruppo, comprendendo i diversi punti di vista, valorizzando le proprie e le altrui capacità, gestendo la conflittualità, contribuendo all'apprendimento comune ed alla realizzazione delle attività collettive, nel riconoscimento dei diritti fondamentali degli altri.
- **Agire in modo autonomo e responsabile**: sapersi inserire in modo attivo e consapevole nella vita sociale e far valere al suo interno i propri diritti e bisogni riconoscendo al contempo quelli altrui, le opportunità comuni, i limiti, le regole, le responsabilità.
- **Risolvere problemi**: affrontare situazioni problematiche costruendo e verificando ipotesi, individuando le fonti e le risorse adeguate, raccogliendo e valutando i dati, proponendo soluzioni utilizzando, secondo il tipo di problema, contenuti e metodi delle diverse discipline.
- **Individuare collegamenti e relazioni**: individuare e rappresentare, elaborando argomentazioni coerenti, collegamenti e relazioni tra fenomeni, eventi e concetti diversi, anche appartenenti a diversi ambiti disciplinari, e lontani nello spazio e nel tempo, cogliendone la natura sistemica, individuando analogie e differenze, coerenze ed incoerenze, cause ed effetti e la loro natura probabilistica.
- **Acquisire ed interpretare l'informazione**: acquisire ed interpretare criticamente l'informazione ricevuta nei diversi ambiti ed attraverso diversi strumenti comunicativi, valutandone l'attendibilità e l'utilità, distinguendo fatti e opinioni.

È decisamente interessante notare come l'insegnamento del pensiero computazionale attraverso attività di *coding*, in base alle linee tratteggiate nei paragrafi precedenti, si dimostri particolarmente adatto a sviluppare diverse di queste competenze chiave. *Progettare* e *risolvere problemi*, in particolare, hanno evidentemente una connessione strettissima con il pensiero computazionale *tout court*. Inoltre la programmazione utilizza un linguaggio specifico, che sia un codice testuale oppure un ambiente visuale in cui combinare blocchi di istruzioni; in questo senso sviluppa la capacità di *comunicare* nel senso di "rappresentare eventi, fenomeni, principi, concetti, norme, procedure […] utilizzando linguaggi diversi (verbale, matematico, scientifico, simbolico, ecc.)". Infine, l'approccio al *coding* che verrà proposto nel Capitolo 5 di questo lavoro, fortemente orientato a un apprendimento per scoperta sia individuale sia collaborativo, stimola e promuove la capacità di *imparare ad imparare* e l'attitudine a *collaborare e partecipare*.

Al termine di questo *excursus*, dunque, l'insegnamento/apprendimento del pensiero computazionale emerge come un campo di grande interesse e rilevanza per le istituzioni educative, destinato con tutta probabilità a un significativo sviluppo all'interno dei curricoli in ogni grado scolastico. Nel prossimo capitolo si cercherà di tracciare una mappa delle iniziative e delle esperienze in corso, nel campo dell'educazione formale e non solo.

2. INIZIATIVE ED ESPERIENZE IN CORSO

2.1 Il panorama internazionale

A partire dagli stimoli provenienti dall'ambito della ricerca, illustrati nel Capitolo 1, negli ultimi anni le istituzioni educative di molti paesi hanno iniziato a dedicare attenzione all'insegnamento del pensiero computazionale. L'8 dicembre 2013, in occasione della "Settimana di educazione all'informatica" in cui si inseriva l'iniziativa "Hour of Code" del progetto Code.org (si veda oltre), il Presidente degli USA Barack Obama ha registrato un breve video in cui lanciava un appello ai giovani per avvicinarli allo studio dell'informatica. Esattamente un anno dopo, l'8 dicembre 2014, Obama ha nuovamente partecipato alla promozione dell'iniziativa, questa volta "divertendosi" a programmare insieme a un gruppo di studenti. Le immagini di questi due eventi, ampiamente diffuse dai mass media di tutto il mondo, hanno costituito un segnale (volutamente) forte, dal punto di vista comunicativo, dell'importanza assegnata a questo tema dall'amministrazione statunitense, e hanno contribuito a dare una notevole visibilità alla questione.

Naturalmente non è semplice delineare un panorama, anche sintetico, della pluralità di iniziative in corso a livello internazionale. In prima approssimazione, è utile distinguere gli ambiti in cui esse si articolano:

- riforme dei curricoli scolastici, da parte delle istituzioni che governano il sistema dell'istruzione;
- progetti sviluppati o patrocinati dalle istituzioni educative;
- progetti, esperienze e risorse proposti da enti, organizzazioni, associazioni all'esterno del settore dell'educazione formale.

Per quanto riguarda l'ambito istituzionale, le riforme dei curricoli riguardano prevalentemente l'insegnamento dell'informatica (*Computer Science*) nei gradi che corrispondono alla nostra scuola secondaria di primo e di secondo grado. Un'eccezione interessante è rappresentata dal National Curriculum britannico, che traccia le linee guida e fissa gli standard per il sistema educativo in Inghilterra, Galles e Irlanda del Nord. Nel settembre del 2013 è stato inserito un curricolo di

Computing molto organico, articolato in verticale su tutti gli anni di istruzione primaria e secondaria[1]. Al suo interno l'apprendimento del pensiero computazionale ha una rilevanza centrale, soprattutto nei primi gradi di istruzione; ma anche nell'istruzione secondaria, accanto a obiettivi e attività più direttamente riconducibili all'informatica *tout court*, permane un'evidente attenzione alle competenze trasversali e alle *life skills* che il pensiero computazionale contribuisce a formare.

Questo approccio è esplicitato fin dalle prime righe della premessa: «Un'educazione di alta qualità alla computazione fornisce agli studenti la capacità di utilizzare il pensiero computazionale e la creatività per comprendere e cambiare il mondo»[2] (traduzione mia). Anche nell'elenco sintetico dei principali obiettivi che si intende perseguire, il pensiero computazionale è connesso da un lato con l'uso consapevole e responsabile delle ICT (la *digital wisdom* di Prensky 2009, si veda il Capitolo 1), dall'altro con la sfera della creatività.

Al di là del caso britannico, la maggior parte delle iniziative e delle esperienze nel panorama internazionale si situa all'esterno delle istituzioni educative formali, come è probabilmente inevitabile in un momento "pionieristico" per lo sviluppo e il consolidamento di un nuovo insegnamento. Organizzazioni di vario tipo (associazioni, università, anche imprese private) hanno messo a punto e reso disponibile sul web un ventaglio di risorse che comprende definizioni e articoli di ricerca, tutorial, *frameworks*, *lesson plans*, fino a veri e propri manuali, guide per gli insegnanti e corsi *on line*. Tra le fonti più importanti figurano associazioni quali Computer Science Teachers Association, International Society for Technology in Education, National Science Foundation; Google, che raccoglie e cataloga moltissimi materiali, utilizzando anche un approccio multidisciplinare (sviluppare il pensiero computazionale all'interno delle altre discipline); università come la Carnegie Mellon University (USA); il dipartimento Media Lab del MIT, in particolare il gruppo di ricerca Lifelong Kindergarten.

Il tratto comune a tutte queste proposte, spesso molto ricche e complete, è che sono pensate principalmente per docenti curiosi o interessati all'argomento, motivati ad approfondirlo per essere in grado di proporre attività di *coding* ai loro

[1] Il curricolo è suddiviso nei *key stages* 1-4, che complessivamente vanno dai 5 ai 16 anni di età degli studenti.

[2] https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study/national-curriculum-in-england-computing-programmes-of-study

studenti. I materiali e i corsi reperibili *on line* presentano livelli di complessità abbastanza differenziati, e in generale non è difficile trovare risorse adatte anche a docenti neofiti di programmazione informatica. Tuttavia è innegabile che, per intraprendere uno di questi percorsi, sono necessarie una buona motivazione di partenza, e almeno un'alfabetizzazione e una "attitudine positiva" verso le tecnologie digitali in senso lato, oltre a una certa disponibilità di tempo e di energie. Queste proposte, infatti, si pongono di solito un obiettivo abbastanza elevato: fornire al docente una formazione di base ma a suo modo completa, sia dal punto di vista teorico, sia da quello didattico. In altri termini, il punto di arrivo ideale non è semplicemente saper gestire una "lezione" di *coding*, ma padroneggiare le basi teoriche e metodologiche del pensiero computazionale per essere in grado di condurre in modo consapevole un breve curricolo.

Tali risorse, dunque, possono rivelarsi preziosissime per quei docenti che, già in possesso o meno di competenze di programmazione, abbiano comunque una certa "confidenza" con le tecnologie, o almeno un atteggiamento aperto e curioso. Nell'attesa che, nei prossimi anni, l'insegnamento del pensiero computazionale venga formalmente integrato nei curricoli ufficiali, questi insegnanti (che nella scuola ovviamente ci sono, e non sono pochi) possono già dotarsi degli strumenti – concettuali e materiali – per organizzare una serie di lezioni, un modulo o un laboratorio.

Il problema è che molto difficilmente questa strada può essere seguita dagli *altri* insegnanti, quelli che vivono il rapporto con le tecnologie con un senso di inadeguatezza, con la paura di non saper affrontare e padroneggiare un campo in cui si sentono meno competenti degli stessi alunni.

Per cercare di "arrivare" anche a loro, offrendo così un'esperienza potenzialmente fruibile dai bambini di tutte le classi (a partire addirittura dai 4 anni di età, prima dell'apprendimento della lettura), è nato Code.org, un progetto che costituisce oggi un punto di riferimento a livello internazionale per la promozione e la diffusione del pensiero computazionale.

2.1.1 Code.org

Fondato nel 2013 da Hadi e Ali Partovi, Code.org è il risultato della collaborazione tra numerosissimi partner, fra cui le più importanti multinazionali

dell'informatica e del web (Google, Apple, Microsoft, Facebook, Twitter, Amazon), associazioni dei settori dell'informatica e dell'istruzione (Computer Science Teachers Association, Association for Computing Machinery, Teach For America), università, istituti di ricerca, scuole.

> Code.org è un'organizzazione non-profit dedicata a espandere l'accesso all'informatica, incrementando la partecipazione delle donne e degli studenti di colore. La nostra prospettiva è che ogni studente di ogni scuola dovrebbe avere l'opportunità di imparare l'informatica. Noi crediamo che l'informatica debba fare parte dei curricoli scolastici, accanto a discipline come biologia, chimica e algebra[3].

Una dichiarazione d'intenti molto chiara, che punta esplicitamente verso una democratizzazione dell'accesso a questo campo di apprendimento, con un'attenzione particolare alle categorie sociodemografiche che risultano sottorappresentate. Gli obiettivi generali, elencati nella stessa pagina web, comprendono il raggiungimento del maggior numero di studenti e di classi in tutto il mondo («go global»), la formazione degli insegnanti, la promozione di politiche volte all'introduzione dell'informatica e del pensiero computazionale nei curricoli scolastici.

Date queste premesse, Code.org ha lanciato un'iniziativa particolarmente efficace e accattivante, chiamata "L'Ora del Codice": «una lezione di introduzione all'informatica della durata di un'ora, progettata per rimuovere l'alone di mistero che spesso avvolge la programmazione dei computer e per mostrare che l'informatica non è affatto difficile da capire, chiunque può impararne le basi»[4]. Chiunque può contribuire a organizzare un'Ora del Codice in qualsiasi periodo dell'anno, anche se i destinatari principali sono le scuole; l'obiettivo ideale è fare in modo che in un istituto venga svolta nelle stesse giornate da tutte le classi. Per incrementare l'*appeal* della proposta, però, le scuole di tutto il pianeta sono invitate a organizzare l'Ora del Codice preferibilmente in una settimana ben precisa, la "Settimana di educazione all'informatica", che nello scorso anno scolastico si è svolta dal 7 al 13 dicembre 2015.

La creazione di un *evento globale* di questo tipo può essere una strategia vincente per promuovere un atteggiamento positivo e avvicinare un gran numero

[3] https://code.org/about (traduzione mia).

[4] https://hourofcode.com/it

di studenti e insegnanti all'informatica e al pensiero computazionale, grazie ad alcuni punti di forza: la motivazione e la soddisfazione di partecipare a un "movimento" mondiale; nessuna competenza specifica richiesta; attività piacevoli, con un approccio di tipo ludico.

Altre strategie di promozione del progetto, il cui obiettivo primario, come detto, è arrivare a coinvolgere e stimolare il maggior numero di studenti possibile, sono:

- l'impiego di una varietà di mezzi per "pubblicizzarlo" e renderlo accattivante, compreso il contributo di testimonial famosi in video motivazionali (politici, celebrità del mondo dell'ICT come Bill Gates e Mark Zuckerberg, stelle dello sport e dello spettacolo);
- l'utilizzo, all'interno delle attività di *coding*, di personaggi e ambientazioni tratti dai più recenti *blockbuster* cinematografici (*Frozen*, *Star Wars*) e videoludici (*Minecraft*, *Angry Birds*), sfruttando il fatto che le aziende detentrici dei relativi diritti commerciali sono tra i partner di Code.org.

Dopo soli due anni e mezzo, i numeri[5] testimoniano un notevole successo dell'intero progetto (che non si limita all'Ora del Codice):

- la piattaforma di Code.org è utilizzata in moltissimi paesi del mondo, anche tramite partnership stabilite con le agenzie educative (è il caso dell'Italia, come si vedrà); i tutorial dell'Ora del Codice sono stati tradotti in 46 lingue;
- nell'ambito dell'Ora del Codice si è registrato lo svolgimento di quasi 200 milioni di "attività" (non si tratta di utenti unici), di cui il 49% da parte di studentesse;
- i corsi di Code.org sono stati svolti in 250.000 classi scolastiche, per un totale di 8 milioni di studenti;
- sono stati formati 20.000 docenti per l'insegnamento dell'informatica nei gradi scolastici *K-12*, ossia i cicli di istruzione primaria e secondaria[6].

[5] I dati, aggiornati a gennaio 2016, sono tratti da https://code.org/about/2015

Sono significativi anche i seguenti dati, raccolti direttamente da Code.org presso i docenti che hanno organizzato l'Ora del Codice nella propria scuola:

- il 98% definisce l'esperienza buona o eccellente;
- l'85% dei neofiti dell'informatica riferisce che l'Ora del Codice ha fatto aumentare il proprio interesse per l'insegnamento dell'informatica;
- il 49% ha intenzione di continuare a insegnare informatica oltre la singola ora del progetto;
- il 18% ha già inserito nel curricolo l'insegnamento dell'informatica a seguito di una precedente partecipazione all'Ora del Codice;
- l'87% riferisce che i propri studenti hanno programmato per più tempo dell'ora inizialmente prevista[7].

Naturalmente, trattandosi presumibilmente (il report non è esplicito al riguardo) di risposte volontarie a un questionario, risentono dell'inevitabile distorsione dovuta al fatto che, di solito, tendono a rispondere soprattutto i più motivati e i più soddisfatti. Sarebbe interessante sapere quale percentuale degli aderenti al progetto ha effettivamente partecipato alla rilevazione.

In ogni caso, sembra lecito concludere che finora il progetto ha raggiunto il suo obiettivo primario, ossia avvicinare all'informatica e al pensiero computazionale il maggior numero possibile di studenti in tutto il mondo, a prescindere dalle eventuali competenze dei loro insegnanti.

La proposta didattica

Entrando nello specifico dell'offerta didattica, la piattaforma di Code.org comprende:

- la sezione "L'Ora del Codice", con una gamma di ambienti e personaggi tra cui scegliere per svolgere una singola lezione, video motivazionali e un breve tutorial introduttivo;
- la sezione "Studenti", con quattro corsi completi di livello crescente. Ogni corso è costituito da diverse lezioni riguardanti concetti o attività

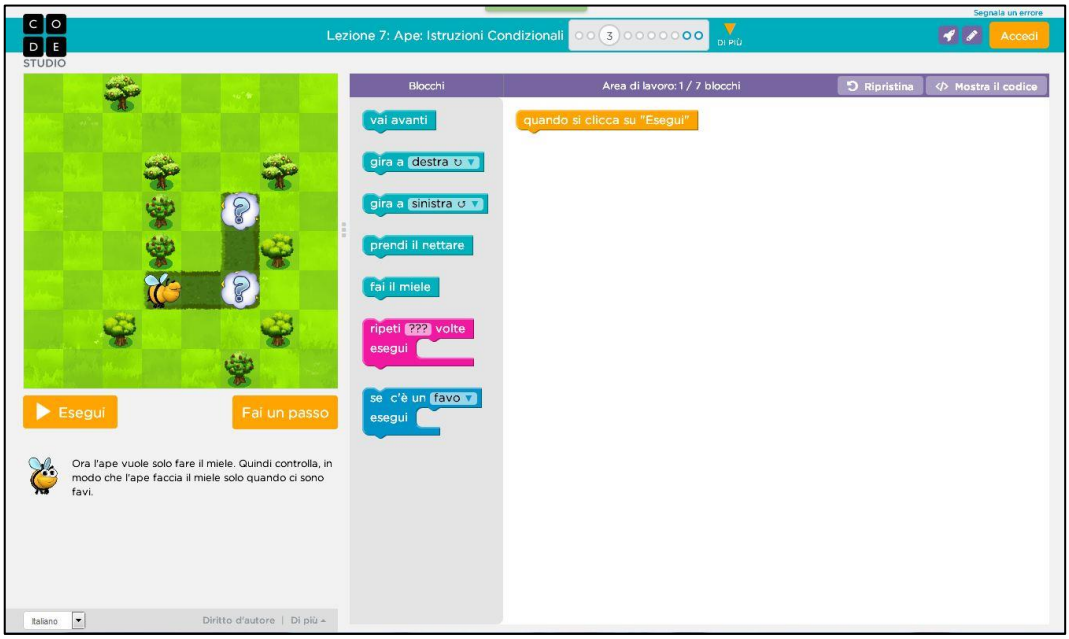
[6] Si veda la nota 1 del Capitolo 1.

[7] https://code.org/about/evaluation/hourofcode (traduzione mia).

specifiche di programmazione (per esempio sequenze, cicli, istruzioni condizionali, *debug*), a loro volta composte da più attività o "esercizi". È presente inoltre una raccolta di materiali per la versione *unplugged*, cioè per attività da svolgere senza il computer;

- risorse e materiali per gli insegnanti;
- una galleria di progetti realizzati e condivisi dagli utenti.

La programmazione avviene in un ambiente visuale, tramite un linguaggio a blocchi simile a Scratch: le istruzioni sono espresse con blocchi colorati da concatenare e combinare. La schermata è suddivisa in tre parti principali: a sinistra si trova l'area di gioco, in cui si può eseguire il programma; al centro la *tool box*, contenente i blocchi utili per quella determinata attività; a destra, infine, c'è l'area di lavoro, in cui trascinare i blocchi presi dalla *tool box*.



È particolarmente interessante, ai fini del presente lavoro, analizzare l'approccio didattico sotteso ai corsi di Code.org. Le lezioni sono composte da singole attività con una consegna precisa: in un dato scenario, lo studente deve far compiere un certo percorso a un personaggio, oppure fargli eseguire una certa azione in base all'oggetto che trova sulla sua strada, possibilmente scrivendo un programma "economico", cioè con il minor numero possibile di "righe di codice" (cioè, in questo caso, di blocchi). Lo sviluppo della lezione consiste in variazioni successive dello stesso tipo di attività, allo scopo di consolidare l'utilizzo di un

certo costrutto informatico (ciclo, istruzione condizionale, ecc.); di fatto si tratta di una forma di *esercizio*. Al termine della lezione si trovano di solito attività di verifica delle competenze acquisite.

Nel complesso, dunque, Code.org propone una didattica molto "guidata", in cui il compito da svolgere è stabilito a priori; per ogni attività c'è un modo *corretto* di risolvere il problema posto, mentre le altre possibili concatenazioni di blocchi sono *sbagliate* e generano un messaggio di errore che invita a riprovare. Solo al termine di alcune lezioni è proposta un'ultima attività libera, in cui non c'è uno scopo da raggiungere e lo studente può scrivere il programma che preferisce.

In questa cornice didattica e metodologica, è significativo il fatto che, per ciascun esercizio, la *tool box* contenga solo *alcuni* blocchi selezionati: le istruzioni necessarie alla risoluzione del problema, più alcune altre della stessa classe (se per svolgere l'esercizio bisogna usare il comando "gira a sinistra", nella cassetta degli attrezzi sarà presente anche "gira a destra", ma non le variabili o i condizionali, per esempio). In questo modo il compito è facilitato, ma l'ambiente risulta molto più povero di possibilità da esplorare. Anzi si può affermare che le prospettive dell'esplorazione e dell'apprendimento per scoperta sono estranee a questo tipo di proposta didattica, articolata in modo sostanzialmente trasmissivo.

Tuttavia occorre rilevare come questo approccio possa rivelarsi tutto sommato coerente con le premesse e gli obiettivi generali di Code.org: avvicinare all'informatica, al *coding* e al pensiero computazionale il maggior numero di bambini e ragazzi, tramite un'esperienza alla portata di tutti, che non richiede competenze pregresse né agli alunni, né soprattutto agli insegnanti.

2.1.2 Fuori dalla scuola: CoderDojo

CoderDojo[8] è una rete di club senza scopo di lucro, aperti, organizzati da volontari (*mentor*), in cui bambini e ragazzi possono imparare a programmare; le specifiche attività variano in base alle preferenze o alle attitudini dei *mentor* e all'età dei partecipanti, e possono riguardare diversi ambienti e linguaggi di programmazione (oltre a Scratch, che di solito occupa un posto centrale, per i più

[8] Il nome è formato dalla giustapposizione del termine inglese *coder*, ossia programmatore, con il giapponese *dojo*, la palestra in cui si praticano le arti marziali.

grandi anche HTML, CSS, PHP, Python, JavaScript), robotica ed elettronica educativa (per esempio Arduino).

Il primo CoderDojo è stato fondato a Cork (Irlanda) nel 2011, e da allora il movimento si è rapidamente diffuso in moltissimi paesi. Non si tratta di una vera e propria organizzazione strutturata, bensì appunto di un movimento "aperto" di cui fanno parte iniziative locali; per aprire un CoderDojo è sufficiente affiliarsi alla rete e condividere e rispettare alcuni principi etici di base, tra cui gratuità della partecipazione, condivisione, software *open source*. Più nello specifico, ogni Dojo è tenuto a rispettare il seguente statuto:

- Ci impegniamo a ispirare e supportare i giovani nell'imparare come creare tecnologia.
- Ci impegniamo a sostenere sempre gli interessi dei giovani partecipanti al nostro Dojo e ad assicurare che le migliori pratiche vengano seguite.
- Ci impegniamo a mantenere lo standard più elevato in termini di tutela dei minori nel nostro territorio.
- Ci impegniamo a non far pagare i ragazzi partecipanti né i loro genitori.
- Ci impegniamo a incoraggiare la partecipazione dei genitori al Dojo.
- Ci impegniamo a condividere le nostre conoscenze in modo libero (nel senso di gratuito e aperto).
- Ci impegniamo a condividere le nostre conoscenze all'interno del nostro Dojo e degli altri Dojo.
- Ci impegniamo a prenderci cura e sostenere il buon nome della comunità globale CoderDojo.
- Ci impegniamo a incoraggiare la collaborazione, il tutoraggio tra pari e il lavoro di squadra tra i partecipanti.
- Ci impegniamo ad accogliere i volontari e i ragazzi a prescindere da genere, razza, orientamento sessuale, credo, religione o abilità.

Oltre ai principi generali, è di particolare interesse anche l'approccio più squisitamente didattico, fortemente improntato all'apprendimento per scoperta e allo sviluppo di progetti, che promuovono un uso creativo delle tecnologie; sono incentivati l'apprendimento cooperativo e il *peer tutoring*; i *mentor*, più che come "docenti", si pongono come facilitatori e svolgono una funzione di *scaffolding*. Un'impostazione didattica che, a grandi linee, verrà seguita nel Capitolo 5 del presente lavoro.

Un altro aspetto interessante del movimento CoderDojo è che, pur trattandosi di un'agenzia educativa non formale, spesso ricerca contatti,

collaborazioni, contaminazioni con il mondo della scuola. Anzitutto è una risorsa preziosa per i docenti interessati all'insegnamento del pensiero computazionale: non è raro che tra i *mentor* vi siano insegnanti di scuola primaria o secondaria, che tramite questa esperienza acquisiscono competenze da trasferire in classe. Inoltre succede che singoli *mentor*, a titolo personale, stabiliscano contatti con le scuole per proporre percorsi o laboratori di programmazione.

2.2 L'insegnamento del pensiero computazionale in Italia

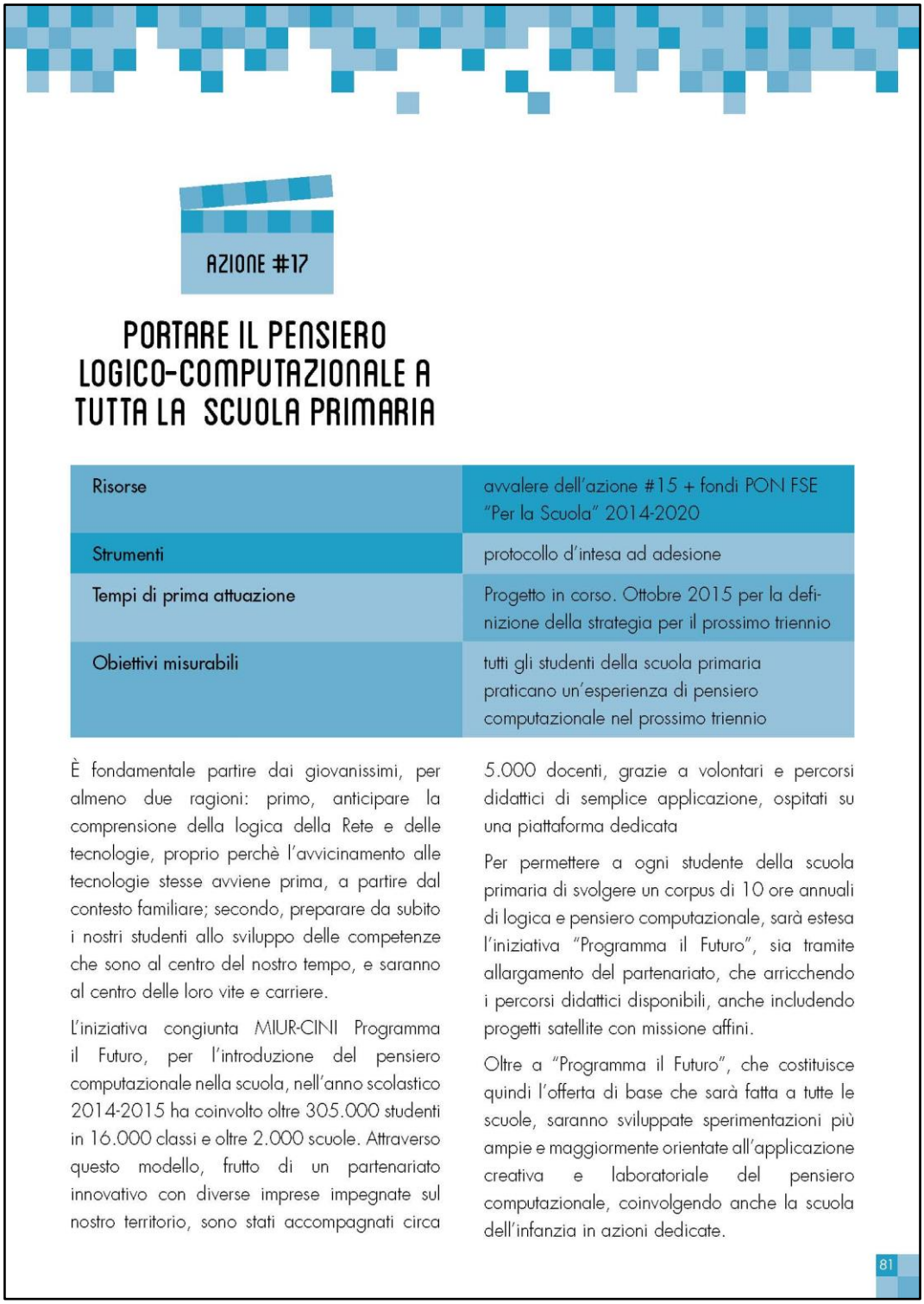
Anche in Italia, negli ultimi due-tre anni, si è registrata una crescente attenzione verso l'insegnamento del pensiero computazionale all'interno del primo ciclo di istruzione, che in parte si inserisce nel più generale clima di rinnovamento del sistema scolastico. L'attuale Governo, come noto, ha dedicato molte energie a una riforma assai dibattuta e discussa, attuata con la Legge 13 luglio 2015, n. 107: *Riforma del sistema nazionale di istruzione e formazione e delega per il riordino delle disposizioni legislative vigenti* (meglio nota come "La Buona Scuola"). La riforma investe una pluralità di aspetti generali, dall'assunzione di personale docente alle modalità di formazione in servizio, dal livello amministrativo e gestionale degli istituti all'edilizia scolastica, e riguarda in modo assai tangenziale l'offerta formativa e i curricoli scolastici.

Molto più pertinente rispetto all'oggetto di questo lavoro è uno degli strumenti attuativi della riforma, il Piano Nazionale Scuola Digitale (PNSD), varato dal MIUR alla fine del 2015. Si tratta di un piano pluriennale che prevede e organizza l'impiego di risorse di diversa provenienza, principalmente fondi strutturali europei e fondi stanziati dalla citata Legge 107/2015. Si articola in 35 "Azioni" raggruppate in quattro ambiti fondamentali: *strumenti*, *competenze*, *contenuti*, *formazione e accompagnamento*; dunque i provvedimenti riguardano tanto aspetti amministrativi e infrastrutturali (per esempio la connettività nelle scuole e gli ambienti per la didattica digitale) quanto la formazione del personale (compresa la nuova figura dell'animatore digitale) e le competenze da promuovere negli studenti:

> Anche in considerazione degli investimenti previsti dalla Buona Scuola, è tempo di investire in un disegno organico di innovazione delle scuole italiane, con programmi e azioni coerenti che comprendano l'accesso, gli ambienti di

apprendimento, i dispositivi, le piattaforme, l'amministrazione digitale, la ricerca, la formazione e ovviamente la didattica, la metodologia e le competenze (Ministero dell'Istruzione, dell'Università e della Ricerca 2015, 20).

A quest'ultimo ambito appartiene l'Azione #17, che si prefigge di «portare il pensiero logico-computazionale a tutta la scuola primaria», riprodotta per intero di seguito.



AZIONE #17

PORTARE IL PENSIERO LOGICO-COMPUTAZIONALE A TUTTA LA SCUOLA PRIMARIA

Risorse	avvalere dell'azione #15 + fondi PON FSE "Per la Scuola" 2014-2020
Strumenti	protocollo d'intesa ad adesione
Tempi di prima attuazione	Progetto in corso. Ottobre 2015 per la definizione della strategia per il prossimo triennio
Obiettivi misurabili	tutti gli studenti della scuola primaria praticano un'esperienza di pensiero computazionale nel prossimo triennio

È fondamentale partire dai giovanissimi, per almeno due ragioni: primo, anticipare la comprensione della logica della Rete e delle tecnologie, proprio perchè l'avvicinamento alle tecnologie stesse avviene prima, a partire dal contesto familiare; secondo, preparare da subito i nostri studenti allo sviluppo delle competenze che sono al centro del nostro tempo, e saranno al centro delle loro vite e carriere.

L'iniziativa congiunta MIUR-CINI Programma il Futuro, per l'introduzione del pensiero computazionale nella scuola, nell'anno scolastico 2014-2015 ha coinvolto oltre 305.000 studenti in 16.000 classi e oltre 2.000 scuole. Attraverso questo modello, frutto di un partenariato innovativo con diverse imprese impegnate sul nostro territorio, sono stati accompagnati circa 5.000 docenti, grazie a volontari e percorsi didattici di semplice applicazione, ospitati su una piattaforma dedicata

Per permettere a ogni studente della scuola primaria di svolgere un corpus di 10 ore annuali di logica e pensiero computazionale, sarà estesa l'iniziativa "Programma il Futuro", sia tramite allargamento del partenariato, che arricchendo i percorsi didattici disponibili, anche includendo progetti satellite con missione affini.

Oltre a "Programma il Futuro", che costituisce quindi l'offerta di base che sarà fatta a tutte le scuole, saranno sviluppate sperimentazioni più ampie e maggiormente orientate all'applicazione creativa e laboratoriale del pensiero computazionale, coinvolgendo anche la scuola dell'infanzia in azioni dedicate.

81

Di fatto questo obiettivo è incentrato sulla più importante iniziativa di promozione del pensiero computazionale messa in atto in Italia nell'ultimo biennio: il progetto "Programma il Futuro", direttamente basato sulla piattaforma di Code.org. Ma si sottolinea esplicitamente che tale progetto va inteso come «l'offerta di base», cui andranno affiancate proposte di tipo più laboratoriale e creativo.

2.2.1 "Programma il Futuro"

Il progetto "Programma il Futuro" è stato sviluppato dal MIUR in collaborazione con il CINI (Consorzio Interuniversitario Nazionale per l'Informatica) ed è al secondo anno di attuazione; come detto, utilizza i materiali didattici di Code.org. Si sviluppa su due piattaforme web distinte ma collegate: un *sito di supporto* appositamente creato (http://www.programmailfuturo.it), che fornisce spiegazioni, informazioni, approfondimenti, tutorial video e gli account di registrazione; il *sito di fruizione* delle attività didattiche (https://studio.code.org), che è la piattaforma Code.org, localizzata in italiano. Le due principali opzioni didattiche sono state denominate "modalità base" (le singole lezioni dell'Ora del Codice) e "modalità avanzata" (i corsi completi, sia nella versione "tecnologica" sia in quella *unplugged*).

Il progetto è espressamente indirizzato a intere classi, in particolare di scuola primaria, tramite una singola registrazione effettuata da un insegnante referente. Tuttavia è possibile fruire delle lezioni anche come singolo utente (volendo anche senza registrarsi).

In linea con la prospettiva di Code.org, gli obiettivi dichiarati all'inizio del progetto erano prettamente quantitativi: raggiungere e sensibilizzare all'insegnamento del pensiero computazionale un numero crescente di scuole.

> Il progetto si propone gli obiettivi sotto elencati.
>
> - a.s. 2014/15
> - coinvolgere il 30% delle scuole primarie (circa 4.600);
> - completare il percorso base L'Ora del Codice in almeno del 15% delle classi delle scuole primarie;

    - completare il percorso avanzato, o nella modalità tecnologica o nella modalità tradizionale, in almeno il 2% delle classi delle scuole primarie (circa 310).
- a.s. 2015/16
    - coinvolgere il 35% delle scuole primarie (circa 5.400);
    - completare il percorso base L'Ora del Codice in almeno il 20% delle classi delle scuole primarie;
    - completare il percorso avanzato, o nella modalità tecnologica o nella modalità tradizionale, in almeno il 5% delle classi delle scuole primarie (circa 770).
- a.s. 2016/17
    - coinvolgere il 40% delle scuole primarie (circa 6.100);
    - completare il percorso base L'Ora del Codice in almeno il 25% delle classi delle scuole primarie;
    - completare il percorso avanzato, o nella modalità tecnologica o nella modalità tradizionale, in almeno il 9% delle classi delle scuole primarie (circa 1380)[9].

Per quanto riguarda la metodologia didattica, valgono naturalmente le osservazioni svolte a proposito di Code.org.

Tuttavia va rilevato che quest'anno è stata lanciata una nuova iniziativa, sempre nell'ambito di Programma il Futuro, che costituisce un primo passo in direzione di un utilizzo creativo ed espressivo della programmazione. Si tratta del concorso "Codi-Amo", che «intende sensibilizzare gli studenti alla riflessione sullo sviluppo del pensiero computazionale, fornendo loro l'opportunità di cimentarsi con forme di espressione originali e stimolanti, e mettendoli in condizione di esprimere le proprie peculiarità e le proprie visioni»[10]. Per partecipare occorre realizzare un programma utilizzando lo stesso ambiente di programmazione delle lezioni, ma in modo libero e creativo; l'unica indicazione è sulla tipologia di elaborato, che deve essere una *storia* per le classi di scuola primaria, un *gioco* per la secondaria di primo grado, un *elaborato grafico* per la secondaria di secondo grado. Ogni classe può partecipare con un solo elaborato, tramite le credenziali dell'insegnante.

All'interno di una proposta didattica piuttosto rigida e trasmissiva, basata su esercizi che puntano a consolidare singoli concetti della computazione, Codi-Amo

[9] http://www.programmailfuturo.it/progetto/descrizione-del-progetto

[10] http://www.programmailfuturo.it/progetto/concorso/introduzione

costituisce dunque un passo significativo verso una prospettiva differente, che favorisca un approccio esplorativo, costruttivista ed espressivo, di certo più motivato, che mette in gioco i processi cognitivi superiori.

Monitoraggio del progetto

I processi di monitoraggio e valutazione sono parte integrante di Programma il Futuro. In marzo il MIUR pubblica un documento relativo al precedente periodo settembre-gennaio, in cui si concentra la maggior parte delle attività nelle scuole (la "Settimana di educazione all'informatica" si svolge in dicembre, mentre lo scorso ottobre si è tenuta la "Settimana europea del codice"), riassunto anche in una pagina del sito. Il documento di quest'anno (Ministero dell'Istruzione, dell'Università e della Ricerca 2016) riporta dati quantitativi sulla partecipazione, con elaborazioni statistiche disaggregate per periodo, grado di scuola, tipologia di partecipanti, regione geografica; inoltre presenta i risultati delle rilevazioni relative al grado di soddisfazione, agli atteggiamenti e alle eventuali difficoltà riscontrate dagli insegnanti e dai volontari di supporto.

In sintesi, i dati quantitativi, se pur lontani dagli obiettivi iniziali riportati sopra, evidenziano un risultato molto incoraggiante: rispetto all'anno scolastico precedente, nel 2015-16 l'adesione è sostanzialmente raddoppiata. Alla data del 15 dicembre 2015 hanno partecipato alle attività 3.289 scuole (rispetto alle 1.911 dell'anno precedente), 9.146 insegnanti (contro 4.417), 29.446 classi (contro 14.948), 601.575 studenti (contro 290.516). La media delle ore di attività svolte da ciascuno studente si attesta a 5,54, con un incremento rispetto alle 4,44 del dicembre 2014.

Un altro dato lusinghiero deriva dal confronto internazionale: durante la settimane dell'Ora del Codice l'Italia è stata la prima nazione al mondo dopo gli USA per numero di eventi organizzati nelle scuole, sia in termini assoluti sia relativi (11.028 eventi = 182 per milione di abitanti).

Per quanto concerne la rilevazione qualitativa dell'esperienza dei partecipanti, l'84% si è dichiarato pienamente soddisfatto. Nell'ottica del miglioramento della proposta, l'indagine ha anche raccolto dati sui motivi di insoddisfazione del restante 16%. Altri aspetti indagati sono stati:

- le difficoltà riscontrate dagli insegnanti nello svolgimento delle attività;

- la valutazione, da parte degli insegnanti, di elementi specifici (contenuti informativi del sito, tutorial, servizio di supporto, ecc.);
- il livello di interesse da parte degli allievi, valutato come "alto" o "molto alto" da ben il 99% dei docenti;
- l'esperienza dei volontari di supporto al progetto.

Un dato interessante, infine, riguarda l'utilizzo dei materiali didattici. Come detto, le lezioni sono fruibili sia nella modalità definita "tecnologica", ossia utilizzando PC connessi a Internet, sia in quella "tradizionale", cioè con attività di tipo carta-e-matita. Ebbene, soltanto il 5,5% degli insegnanti ha utilizzato esclusivamente il PC, il 14,4% si è avvalso in ugual misura delle due modalità, mentre l'80,1% ha svolto prevalentemente le lezioni tecnologiche, ma proponendone anche alcune *unplugged*.

> Questa modalità integrativa nell'uso degli strumenti didattici mostra l'efficacia dell'obiettivo del progetto, che intende diffondere non tanto una conoscenza di aspetti tecnologici quanto una comprensione delle basi culturali del mondo digitale (Ministero dell'Istruzione, dell'Università e della Ricerca 2016, 31).

In conclusione, Programma il Futuro si presenta come uno strumento utile ed efficace per promuovere un primo approccio al coding e al pensiero computazionale nella scuola, soprattutto primaria. Da qui occorrerà però partire per articolare proposte più ricche e più "ambiziose", sia in termini quantitativi, sia soprattutto per quanto riguarda i modelli didattici soggiacenti.

2.2.2 Altre iniziative, dentro e attorno alla scuola

Si propone ora una panoramica, necessariamente parziale, di iniziative, esperienze e attività di vario tipo, a livello sia nazionale sia locale, che coinvolgono anche settori limitrofi come la robotica educativa e il mondo dei *makers* e dei *Fab Lab*.

- Il PON (Programma Operativo Nazionale) 2014-2020 "Per la Scuola – Competenze e ambienti per l'apprendimento", varato dal MIUR, è un piano che mette a disposizione Fondi Strutturali Europei per finanziare interventi volti all'innovazione organizzativa e didattica delle scuole. In particolare, nel marzo 2016 il MIUR ha indetto un bando per finanziare

la realizzazione di *atelier creativi*, previsti anche all'interno dell'Azione #7 del PNSD, denominata "Piano laboratori". Gli atelier creativi sono ambienti per la didattica laboratoriale in cui «sviluppare il punto d'incontro tra manualità, artigianato, creatività e tecnologie», utilizzando strumenti di «robotica ed elettronica educativa, logica e pensiero computazionale, artefatti manuali e digitali, *serious play* e *storytelling*» (Ministero dell'Istruzione, dell'Università e della Ricerca 2015, 50).

- A livello regionale esistono unità operative, che fanno capo agli Uffici Scolastici Regionali, con specifiche competenze di supporto alle scuole in tema di innovazione tecnologica. In Emilia-Romagna, per esempio, opera il Servizio Marconi TSI, che comprende tra le proprie attività: il monitoraggio dell'attuazione del PNSD nella regione; l'affiancamento alle scuole nel campo dell'innovazione didattica connessa all'uso delle tecnologie; la formazione dei docenti; la promozione di sperimentazioni e progetti specifici. Per esempio il Servizio Marconi organizza e gestisce RoboCoop, un progetto finanziato dalla rete delle cooperative di consumatori per portare la robotica educativa nelle scuole dei comuni emiliani colpiti dal sisma del 2012; nell'anno scolastico 2015-16 il progetto ha fornito alle scuole sia una dotazione strumentale, sia attività di formazione e accompagnamento per i docenti.
- Tra i progetti più interessanti svolti negli ultimi anni si segnala "Girls code it better", attivo al momento in Lombardia, Emilia-Romagna e Toscana. Si tratta di club pomeridiani per ragazze della scuola secondaria di primo grado, in cui si sviluppano progetti che spaziano in tutti i campi di applicazione della computazione e del digitale: dalla programmazione di app e videogiochi alla realizzazione di siti web, dalla robotica all'elettronica, alla stampa 3D. Finora[11] hanno partecipato 33 scuole in 17 città, per un totale di 760 ragazze coinvolte e 74 "coach". Il progetto è nato per scardinare un persistente stereotipo di genere, secondo cui le ragazze sarebbero "meno portate" per l'informatica e le scienze in generale; questo stereotipo produce una sorta di *confidence gap* tra maschi e femmine, che alla lunga innesca una

[11] http://www.girlscodeitbetter.it/#/home/progetto (consultato l'8-5-2016)

profezia che si autoadempie, per cui davvero nei corsi di studio cosiddetti STEM (*Science, Technology, Engineering and Math*) la presenza femminile è molto minoritaria. Come notato in precedenza, la diffusione dell'informatica tra categorie sociodemografiche "sottorappresentate" è uno degli obiettivi anche del progetto Code.org, a testimonianza di quanto questo aspetto sia oggi preso in seria considerazione.

- Un'altra iniziativa degna di nota è "Robocup Junior", una competizione di robotica organizzata da una rete di istituti scolastici di tutta Italia (inizialmente erano 8, oggi sono 49, in prevalenza scuole secondarie di secondo grado), giunta all'ottava edizione.
- Anche in Italia il movimento CoderDojo si sta diffondendo rapidamente: ad oggi sono attive alcune decine di club, in centri sparsi su tutto il territorio nazionale. In alcuni casi si sono innescate interessanti sinergie sul territorio, con ricadute positive sul mondo della scuola; per esempio il citato Servizio Marconi dell'USR dell'Emilia-Romagna e l'Università di Bologna hanno sviluppato rapporti di collaborazione con alcuni *mentor* del CoderDojo del capoluogo emiliano, sfociati – fra le altre cose – in percorsi di formazione per l'insegnamento del pensiero computazionale destinati a docenti presenti e futuri.
- Oltre ai corsi di formazione organizzati dagli Uffici Scolastici Regionali, i docenti interessati o curiosi possono reperire corsi on line in italiano, come per esempio (nonostante il titolo inglese…) "Coding in your classroom, now!", o siti web di associazioni che offrono supporto per l'insegnamento del pensiero computazionale a scuola, come Studiare Digitale ONLUS.

3. TEORIE DELL'APPRENDIMENTO E MODELLI DIDATTICI

Qualsiasi proposta didattica, per essere sensata e coerente, deve poggiare su fondamenti teorici e metodologici chiaramente delineati: «Alla base di ogni forma didattica c'è un modo di concepire la conoscenza e una relativa teoria che informa le pratiche didattiche circa il senso di quel tipo di operare» (Gherardi 2010, 35). Da sempre l'uomo si interroga sulla propria conoscenza, sui principi e sui meccanismi che la regolano, e questa riflessione ha costituito nei secoli uno dei campi privilegiati dell'indagine filosofica; ma è nel secolo scorso che si è costituita, affermata e consolidata una disciplina scientifica, la Psicologia, e in particolare una sua branca specifica, la Psicologia dell'apprendimento e dell'istruzione. I modelli teorici elaborati dagli psicologi in questo campo sono diventati i necessari punti di riferimento per le Scienze dell'educazione, in particolare per la disciplina incaricata di "trasferirli" nell'ambito delle situazioni di insegnamento/apprendimento e di organizzare la mediazione tra insegnante, allievi e sapere: la Didattica.

Il presente capitolo propone un *excursus* dei principali paradigmi teorici che si sono succeduti – e in parte sovrapposti – negli ultimi cento anni, e delle più rilevanti implicazioni/applicazioni in campo didattico. Data la vastità dell'argomento e l'oggetto specifico di questa tesi, la trattazione sarà necessariamente molto sintetica e "orientata": si cercherà di rendere conto, almeno a grandi linee, della prospettiva storica, ma ci si concentrerà in modo particolare su approcci, teorie e spunti maggiormente pertinenti con la proposta didattica che si intende avanzare.

In prima battuta, per dare un inquadramento "storiografico" molto generale, si può osservare che:

> […] due "paradigmi", o comunque due fasi cruciali, si impongono nella storia del pensiero didattico degli ultimi cinquant'anni: il primo si colloca negli anni cinquanta-sessanta, il secondo negli anni ottanta-novanta; il primo presuppone una concezione lineare e gerarchica della scienza e trova il suo sbocco applicativo più evidente nel movimento per la *progettazione curricolare*, il secondo si colloca all'interno di una concezione più complessa e problematica della conoscenza, che oggi nel dibattito internazionale viene comunemente definita con il termine

costruttivismo e trova le sue più forti implicazioni didattiche in concetti quali quello di "ambiente", "comunità di dialogo", "comunità di apprendimento", "circoli di apprendimento" (Calvani 2000, 45-46).

3.1 Comportamentismo

Il Comportamentismo si è sviluppato soprattutto in ambito statunitense a partire dai primi decenni del Novecento, e si è affermato come paradigma dominante nel decennio successivo alla conclusione della Seconda guerra mondiale. Analizzando il fenomeno da una prospettiva storico-culturale molto ampia, il "clima" generale del periodo ha senz'altro influito sull'elaborazione e sulla fortuna dell'approccio comportamentista: il mondo era appena uscito da un conflitto terribile, che fra le altre cose aveva sancito l'importanza fondamentale dello sviluppo tecnologico; e ancor più nello scenario della Guerra fredda, che si stava rapidamente delineando, la competizione scientifico-tecnologica sarebbe stata decisiva per regolare i rapporti di forza tra le potenze. Era perciò un periodo di incrollabile fiducia positivistica nelle scienze "dure": i modelli epistemologici dominanti erano quelli dell'ingegneria e della matematica (*ibidem*, 46-47).

Il Comportamentismo riporta quella stessa fiducia in un campo particolare come quello dell'istruzione: nell'intento esplicito di rifondare la psicologia su basi scientifiche, viene espunto dal campo di indagine tutto ciò che non è direttamente osservabile e misurabile, come per esempio i concetti di "mente", "coscienza", "introspezione". L'unico oggetto di analisi è il *comportamento manifesto* degli individui. Tale comportamento è influenzato in misura determinante dall'apprendimento: nella dicotomia tra *innato* e *appreso*, il Comportamentismo si colloca decisamente verso il secondo polo, assegnando un ruolo decisivo all'ambiente nel plasmare il comportamento (Macchi Cassia, Valenza & Simion 2012, 33-34).

> Da qui deriva l'importanza attribuita all'apprendimento: predisponendo un ambiente in maniera adeguata, si possono forgiare tutti i comportamenti desiderati in un individuo. In altri termini, l'apprendimento altro non è se non la continua creazione di nuove associazioni tra stimoli dell'ambiente e risposte dell'individuo (Mason 2006, 15).

Il meccanismo fondamentale per instaurare i comportamenti desiderati (e quindi, nella visione comportamentista, per produrre "apprendimento") è il *condizionamento operante*, più efficace del condizionamento classico già studiato da Pavlov nei suoi celebri esperimenti. Mentre quest'ultimo può spiegare solo comportamenti molto semplici, il condizionamento operante ha un maggiore potere esplicativo.

In particolare, nel condizionamento classico una risposta a un determinato stimolo "naturale" (incondizionato) viene associata a uno stimolo differente (condizionato); questo meccanismo però non spiega come avviene l'apprendimento di comportamenti *nuovi* (Macchi Cassia, Valenza & Simion 2012). È qui che interviene il condizionamento operante, in cui un comportamento messo in atto in modo casuale viene rinforzato da uno stimolo successivo, chiamato *rinforzo*: «se un comportamento (che fa parte del repertorio di risposte emesse da un organismo) prodotto in assenza di uno stimolo particolare viene rinforzato, se ne aumenta la frequenza» (Mason 2006, 16). In questo caso si tratta di un rinforzo positivo. Al contrario, per portare all'abbandono di un certo comportamento, è più utile ed efficace (almeno nell'ambito dell'istruzione) ricorrere all'*estinzione operante*, cioè all'eliminazione del rinforzo positivo, piuttosto che a un rinforzo negativo, che può provocare avversione o rifiuto per la disciplina oggetto di apprendimento.

Questi semplici principi generali sono ritenuti sufficienti a spiegare l'apprendimento in qualsiasi campo; il *focus* è tutto sul contenuto da trasmettere: se viene adeguatamente analizzato e parcellizzato in porzioni minime, ciascuna di esse può essere "insegnata" attraverso il meccanismo del condizionamento operante. Su questa base Skinner, il più importante studioso comportamentista, ha proposto il suo modello di insegnamento, chiamato *istruzione programmata*, che consiste appunto nel suddividere un argomento in pacchetti di informazione "elementari", rigorosamente ordinati in modo sequenziale (Parente 2004, 76-77). Se ben strutturata, l'unità di insegnamento può anche essere somministrata da una macchina (una *teaching machine*).

È utile sottolineare brevemente due ulteriori elementi che emergono da questo quadro teorico: la concezione dell'*errore*, che è visto semplicemente come qualcosa da evitare o estinguere; e il principio dell'*individualizzazione* dell'insegnamento, che – pur in contesti e forme diverse – ha mantenuto e

mantiene tuttora un ruolo di primo piano nella pedagogia (Frabboni 2007; Vannini 2009).

Nel panorama generale degli approcci all'educazione, oggi il Comportamentismo è universalmente considerato superato. La visione che propone dell'apprendimento (e del discente) è assai limitata e meccanica, e può rendere conto solo di acquisizioni molto semplici (Macchi Cassia, Valenza & Simion 2012; De Beni 2003, 20). Tuttavia non va dimenticato che essa è piuttosto radicata in un certo "senso comune", e che fa ancora capolino anche all'interno della scuola. In realtà non si può negare, per esempio, che in molti casi possa essere efficace articolare un obiettivo complesso in sotto-obiettivi elementari; ma questo può valere per l'insegnamento di tecniche, procedure, algoritmi (non a caso i principi del Comportamentismo hanno trovato applicazione nell'addestramento militare). L'*apprendimento*, inteso nel senso più generale e comprensivo, comprende dimensioni di contenuto e processi cognitivi ben più complessi e "ricchi", in cui l'elaborazione attiva delle informazioni da parte del discente gioca un ruolo fondamentale, come è stato evidenziato dal paradigma cognitivista.

Un altro rischio in parte ereditato dal Comportamentismo riguarda l'enfasi sul rinforzo positivo, che dovrebbe incentivare il comportamento (l'unico) ritenuto corretto. Naturalmente nella scuola, come in qualsiasi altro campo, è giusto che ci sia un livello di gratificazione personale a fronte dell'impegno e dei risultati positivi; tuttavia è utile riflettere sui possibili eccessi: da parte dell'alunno, la ricerca costante del "premio" (il bel voto) può sollecitare – anziché un reale apprendimento – la manifestazione di comportamenti conformisti. Per evitare questo pericolo è essenziale che il docente valorizzi le soluzioni alternative, il pensiero divergente, i processi oltre ai prodotti, gli errori stessi (Baldacci 2004). Tutti elementi completamente estranei al paradigma comportamentista, sui quali si è iniziato a riflettere nei decenni successivi.

3.2 Cognitivismo

L'approccio cognitivista nasce nella seconda metà degli anni '50 proprio in contrapposizione al Comportamentismo. In particolare viene rovesciato uno dei suoi assunti di base: l'idea della mente come "scatola nera" dai meccanismi

inconoscibili, e perciò completamente espunta dalla ricerca scientifica. Per i cognitivisti, anche se – come è ovvio – non si può osservare direttamente ciò che accade nel cervello di un individuo, è tuttavia possibile avanzare ipotesi e inferenze e costruire modelli che spieghino i meccanismi con cui gli esseri umani elaborano le informazioni e acquisiscono le conoscenze. Anzi, non è solo possibile: è il principale interesse della psicologia di stampo cognitivista (Macchi Cassia, Valenza & Simion 2012, 85).

Questa svolta nell'approccio allo studio della conoscenza, oltre che dall'insoddisfazione per i risultati aggiunti dalle ricerche dei comportamentisti, è stato favorito in modo decisivo dalla nascita di due nuove discipline come la robotica e l'informatica. La *metafora del computer* per lo studio della mente umana, fin dall'inizio, ha fortemente caratterizzato l'approccio cognitivista, che non a caso ha ricevuto notevoli impulsi nel periodo di massimo fervore delle ricerche sull'intelligenza artificiale (Calvani 2000).

Va comunque precisato che l'analogia con il funzionamento del computer ha sempre riguardato il livello funzionale, ossia quello che normalmente si indica con *mente*: i cognitivisti, cioè, hanno cercato di studiare le strutture, i processi e i meccanismi che regolano l'apprendimento umano, senza però spingersi a ipotizzare dove abbiano luogo, in quali zone specifiche del cervello[1].

Prima di esaminare le principali elaborazioni di questo approccio, un'altra precisazione. Etichette come *Cognitivismo* e *Costruttivismo* sono senz'altro utili per sintetizzare una grande mole di teorie, studi e ricerche che presentano un denominatore comune in certi presupposti di base; nel corso del tempo, però, i paradigmi possono evolvere, svilupparsi in diverse direzioni, stabilire punti di contatto reciproco, tanto che non è sempre facile – e talvolta è fuorviante – decidere se un dato autore o una data teoria fa parte della "famiglia" cognitivista, per esempio[2]. Pertanto, sulla scorta dei testi consultati e con la consapevolezza che si tratta di un'interpretazione storiografica, il Cognitivismo verrà presentato come articolato in due fasi principali: la prima, che ha visto l'elaborazione di un

[1] In questa direzione si sono aperte interessanti prospettive in anni recenti, grazie allo sviluppo delle neuroscienze e delle tecniche di *neuroimaging*. Approcci come il neurocostruttivismo puntano proprio a individuare i substrati neurali dei processi mentali (Macchi Cassia, Valenza & Simion 2012).

[2] Questo non succede con il Comportamentismo, un approccio molto più definito e con confini ben delineati.

nucleo di concetti e modelli per spiegare come gli esseri umani elaborano le informazioni a partire dagli *input* sensoriali; la seconda, in cui la visuale si è molto ampliata, che ancora oggi domina la scena e presenta rilevanti intersezioni con l'approccio costruttivista. A questa seconda fase si possono ricondurre teorie e approcci particolari, che verranno trattati di seguito.

3.2.1 La mente umana e lo *Human Information Processing*

L'oggetto di indagine fondamentale dei primi modelli cognitivisti è il trattamento dell'*informazione* nella mente umana, articolabile in una pluralità di processi: ricezione, trasformazione, elaborazione, immagazzinamento, recupero mnestico, ecc. Partendo dall'osservazione dei comportamenti (*output*) prodotti in situazioni sperimentali opportunamente progettate, vengono costruiti modelli del funzionamento della mente e dell'elaborazione delle rappresentazioni mentali (Mason 2006; Macchi Cassia, Valenza & Simion 2012).

Nel 1968 Atkinson e Shiffrin hanno proposto un modello che ha costituito la base per la ricerca successiva, e che individua tre sistemi fondamentali attraverso cui transita l'informazione: il registro sensoriale, la memoria a breve termine (in seguito denominata memoria di lavoro) e la memoria a lungo termine. Il primo è collegato con l'organo di senso corrispondente, conserva l'informazione per un tempo molto breve e non riveste un interesse particolare; i due magazzini di memoria, invece, sono le strutture fondamentali sulle quali si è concentrata la ricerca, particolarmente interessata alla funzione delle strutture mnestiche nei processi di apprendimento.

La *memoria a lungo termine* è l'archivio, potenzialmente illimitato, che contiene tutto ciò che comunemente chiamiamo "conoscenza". L'enorme massa di informazioni di cui un individuo dispone è conservata qui, e i singoli elementi vengono richiamati dalla memoria di lavoro quando devono essere utilizzati. La questione di grande interesse per i cognitivisti è ipotizzare *come* sono codificate, archiviate e connesse tra loro queste conoscenze, ovvero se e come si può costruire un modello della memoria a lungo termine. Inizialmente la ricerca si è orientata verso una struttura di tipo reticolare (Calvani 2000, 71-72), anche sulla scorta delle elaborazioni di discipline contigue come la semiotica e la linguistica. La conoscenza immagazzinata nella memoria a lungo termine è stata cioè

rappresentata come un enorme insieme di nodi (i concetti) collegati gli uni agli altri da nessi (relazioni) più o meno forti e stabili; l'apprendimento può consistere nell'aggiunta o nella cancellazione di nodi o di nessi, con conseguente ristrutturazione della rete semantica nel suo insieme.

Qualsiasi modifica è prodotta dall'elaborazione che avviene nella *memoria di lavoro*, che è il vero "motore" dell'apprendimento e, più in generale, di tutti i processi cognitivi. Moltissimi studi si sono dedicati all'esplorazione di questo dispositivo fondamentale; il risultato è un modello abbastanza condiviso, con alcuni caratteri distintivi.

La funzione della memoria di lavoro è quella di interfaccia tra il registro sensoriale e la memoria a lungo termine; per elaborare i dati provenienti dal primo (*input*) è necessario richiamare informazioni pertinenti dalla seconda, con cui farli interagire e produrre eventualmente una nuova rappresentazione da archiviare nella memoria a lungo termine.

> La memoria di lavoro è un sistema per il mantenimento temporaneo e per la manipolazione dell'informazione durante l'esecuzione di differenti compiti cognitivi […] In altre parole, è "*uno spazio di lavoro*" in cui non solo tratteniamo temporaneamente l'informazione ma operiamo anche attivamente su di essa. La memoria di lavoro è quindi la parte del sistema di elaborazione dell'informazione "*attiva*" perché mette insieme le informazioni provenienti dai registri sensoriali con la memoria a lungo termine (Macchi Cassia, Valenza & Simion 2012, 93).

Tuttavia la memoria di lavoro ha una capienza limitata sia in termini temporali (da pochi secondi a circa un minuto, in base all'età del soggetto, alle caratteristiche individuali, al tipo di "materiale" da gestire), sia come numero di item (in media 7 unità per l'adulto). Ciò significa che l'elaborazione attiva dell'informazione ha una soglia massima ben precisa: non si può gestire contemporaneamente una quantità troppo elevata di dati.

Per ovviare a questo limite e aumentare le capacità di elaborazione cognitiva, è possibile operare in due direzioni diverse ma collegate: sulla *struttura della conoscenza* immagazzinata nella memoria a lungo termine e sui *processi di recupero* di tali contenuti. Diverse ricerche hanno indagato questi ambiti, cercando di identificare, in sintesi, i meccanismi con cui l'*expertise* (essere esperti in un determinato campo) modifica l'attività di elaborazione delle informazioni (*ibidem*, 109-113). L'"esperto" ha una conoscenza di dominio molto sviluppata,

che consiste in un reticolo semantico più fitto e denso: oltre a una maggiore quantità di concetti, anche le connessioni sono più numerose. Alcune di queste connessioni – quelle "attivate" più di frequente – possono assumere un "peso" tale da divenire automatiche, e quindi non richiedere più l'impiego della memoria di lavoro per essere stabilite. In altri termini, l'esperto è in grado di utilizzare in modo più efficiente la memoria di lavoro perché le informazioni che richiama dalla memoria a lungo termine si presentano già ricche, articolate e strutturate, ma in modo automatico, e pertanto apportano un grande contenuto informativo occupando un solo *slot* di memoria di lavoro. Ciò consente di avere una maggiore capacità di elaborare e integrare le nuove informazioni.

In sintesi, gli elementi fondamentali di questa prima fase della ricerca cognitivista vengono così elencati da Calvani (2000, 52):

> Si può dire che è negli anni settanta che l'orientamento razionalistico alla base del cognitivismo *Information Processing* tocca il suo culmine; sono ormai chiare le idee su cui esso poggia: la conoscenza è riflesso della realtà, è formalizzabile, può essere descritta attraverso particolari tipi di elaborazione, è implementabile in una macchina; il computer è una sorta di laboratorio della mente che permette sia di simulare l'intelligenza umana, sia di fornire modelli sul suo funzionamento. Allo stimolo esterno comportamentistico si è sostituita l'informazione; questa viene ricevuta dall'esterno ed elaborata all'interno, accolta dapprima in una memoria di lavoro, eventualmente trasferita in una memoria a lungo termine; fioriscono modelli sul trattamento dell'informazione e sulle tipologie delle memorie interne.

Dagli anni '70 la ricerca amplia e arricchisce questo nucleo teorico, indagando in particolare le modalità di rappresentazione della conoscenza nella mente umana; viene abbandonato il modello della rete semantica di tipo "enciclopedico", piuttosto rigido, a favore di strutture più generali, duttili e malleabili.

Il concetto fondamentale è quello di *schema*, definibile come una «unità organizzativa della memoria che rappresenta le nostre conoscenze relative a oggetti, situazioni, eventi e azioni» (Mason 2006, 27), a un certo livello di generalità e di astrazione. Gli schemi non sono "definizioni"; contengono delle variabili, sono inseribili gli uni negli altri, consentono di fare previsioni sui dati sensoriali in arrivo. In qualunque processo di apprendimento sono implicati uno o più schemi, che vengono richiamati dalla memoria a lungo termine in quella di lavoro, e interagiscono con le informazioni in ingresso. L'apprendimento consiste

proprio in una modifica dello schema richiamato, che – in base al grado di coerenza con i nuovi dati da integrare – può avvenire secondo tre modalità: per *accrescimento*, quando le nuove informazioni sono coerenti con lo schema posseduto, e vanno semplicemente ad arricchirlo; per *sintonizzazione*, quando l'integrazione delle nuove conoscenze richiede limitati aggiustamenti dello schema; per *ristrutturazione*, nel caso in cui sia necessaria una modifica sostanziale dello schema, o la costruzione di uno nuovo tramite astrazione e generalizzazione dei dati dell'esperienza. È interessante notare come questi meccanismi richiamino i concetti piagetiani di assimilazione e accomodamento, esprimendo la stessa interazione complessa fra strutture cognitive e dati sensoriali.

Oltre a questo modello generale, sono state proposte alcune tipologie più specifiche di schemi, in base ai contenuti di conoscenza: un *frame* (o cornice) è «una struttura di dati utile a rappresentare una situazione stereotipata o contesto familiare» (*ibidem*, 30), come possono essere i contesti "aula scolastica" o "supermercato"; uno *script* (o copione) rappresenta invece una conoscenza di tipo procedurale, «una sequenza di eventi che organizza in ordine temporale una serie di azioni compiute per conseguire uno scopo» (*ibidem*, 29), per esempio prendere un aereo o andare al cinema. Di ordine differente è invece il *modello mentale*, una rappresentazione di qualcosa che esiste nella realtà; al contrario di schemi, frame e script, il modello mentale di solito si riferisce a qualcosa di specifico e non è dotato di una «natura proposizionale, perché non è costituito da stringhe di simboli corrispondenti al linguaggio naturale» (*ibidem*, 30).

3.2.2 L'apprendimento significativo

Un importante contributo alla comprensione del modo in cui vengono assimilate le nuove conoscenze è derivato dal lavoro dello psicologo statunitense David P. Ausubel, che si è occupato a fondo dei meccanismi dell'apprendimento e delle loro implicazioni nella pratica scolastica. Tra le sue numerose pubblicazioni riveste un'importanza fondamentale *Educazione e processi cognitivi. Guida psicologica per gli insegnanti* [1978; ed. or. 1968], un ponderoso trattato in cui egli espone in modo sistematico le sue teorie, che prendono in esame una pluralità di dimensioni dell'apprendimento e le relative implicazioni nella pratica educativa. Per esempio, se è innegabile che l'interesse preminente di Ausubel è

per i processi cognitivi, non vengono trascurati i fattori affettivi, sociali e situazionali.

Il nucleo del modello teorico di Ausubel risiede nella distinzione fra due tipi di apprendimento: quello *meccanico*, che consiste nella semplice memorizzazione di un'informazione "così come viene ricevuta", e quello *significativo*, con cui le nuove informazioni vengono integrate in modo organico nella struttura cognitiva del soggetto. Questa seconda modalità è quella più efficace, ossia quella che produce una vera conoscenza; mentre l'apprendimento meccanico è tipico delle pratiche didattiche di ispirazione comportamentista, con cui Ausubel polemizza. Per quanto riguarda l'apprendimento significativo, l'autore indaga in profondità le caratteristiche della struttura cognitiva e della memoria, elaborando un modello complesso. In sintesi, l'apprendimento significativo è agevolato se le conoscenze pregresse rilevanti per quel dato argomento sono abbastanza stabili e definite. Ausubel parla di *anchoring ideas*, concetti di ancoraggio, che devono avere una forte correlazione con le nuove informazioni ed essere ben strutturate e facilmente richiamabili dalla memoria a lungo termine. L'intervento didattico deve agire proprio su questo:

> […] il trasferimento cognitivo nell'apprendimento scolastico consiste soprattutto nel modellare la struttura cognitiva dello studente, nel manipolare il contenuto e la strutturazione delle sue precedenti esperienze di apprendimento in una particolare materia, in modo tale che le ulteriori esperienze di apprendimento siano facilitate al massimo [Ausubel 1978, 222].

La principale strategia per ottenere questo scopo implica l'utilizzo di "anticipatori" (*advance organizers*), ossia materiali introduttivi chiari, a un livello elevato di astrazione e generalità, che possano mobilitare le conoscenze pregresse degli alunni e fungere da "impalcatura" per l'integrazione di quelle nuove.

La teoria di Ausubel presenta però anche un'altra indicazione molto rilevante per la pratica didattica. La dicotomia tra i due tipi di apprendimento si incrocia infatti con quella relativa alle modalità con cui le nuove informazioni provengono al discente: per *ricezione* o per *scoperta*. Il risultato è una tabella a doppia entrata con quattro caselle, ciascuna delle quali rappresenta un modello di "apprendimento". Ausubel in questo modo mette in guardia dal rischio di incorrere in due correlazioni troppo semplicistiche (e polemizza con una certa tendenza in atto): apprendimento per ricezione → meccanico; apprendimento per

scoperta → significativo. In realtà anche la ricezione di contenuti preconfezionati *può* produrre apprendimento significativo, così come è possibile che un'attività di scoperta sfoci in una semplice memorizzazione, se le nuove conoscenze non trovano una struttura cognitiva in cui inserirsi in modo organico. Si tratta di un'indicazione molto importante per la pratica didattica, in cui il pericolo di questa semplificazione è spesso presente.

Diversi autori si sono occupati di elaborare approcci didattici alla luce della teoria dell'apprendimento significativo. Uno degli strumenti più interessanti, ormai patrimonio acquisito della prassi scolastica, è costituito dalle *mappe concettuali*, che rappresentano la struttura dei contenuti oggetto di apprendimento, mettendo in luce i possibili snodi in cui identificare i concetti di ancoraggio e gli anticipatori, e facilitando così l'assimilazione attiva delle nuove informazioni nella rete di conoscenze già possedute dal discente (Novak 2001).

3.2.3 Gli studi sulla metacognizione

Un altro campo di ricerca di grande importanza in chiave didattica riguarda la *metacognizione*. I processi cognitivi – si pensi per esempio alla comprensione di un testo – avvengono solitamente in modo "trasparente" per il bambino, che non è consapevole dei meccanismi specifici che mette in atto, o non vi si sofferma. Ciò vale non solo per i processi *bottom-up*, ma anche per quelli *top-down*: è naturale svolgere un ragionamento senza avere coscienza delle procedure logiche sottostanti. La metacognizione riguarda proprio la capacità di riflettere su queste dimensioni, ed è una "metacompetenza" che si può stimolare e sviluppare. L'assunto di base – molto semplice, quasi banale – è che avere una buona conoscenza e una buona padronanza delle proprie strategie cognitive aiuta ad apprendere meglio (qualsiasi contenuto, in qualsiasi campo disciplinare). E questo riveste un'importanza cruciale, nella società del *lifelong learnig*, per la competenza chiave dell'*imparare a imparare*.

La ricerca ha elaborato diversi modelli per analizzare i processi metacognitivi (Mason 2006; Macchi Cassia, Valenza & Simion 2012; Cornoldi 1995), riconducibili a una distinzione fondamentale:

- metacognizione come *conoscenza e consapevolezza* riguardante «le caratteristiche e abilità personali, i compiti e contesti di apprendimento, le strategie da adottare» (Mason 2006, 151);
- metacognizione come *regolazione e controllo* dei propri processi cognitivi, ossia «previsione, pianificazione, controllo e valutazione di processi attivati per svolgere un determinato compito» (*ibidem*, 151-152).

A livello didattico, è oggi ampiamente condivisa la necessità di dedicare tempo e attenzione a stimolare entrambe queste dimensioni (Borkowski & Muthukrishna 2011); una delle modalità più tipiche, per esempio, è la discussione e il confronto in grande gruppo sulle diverse strategie adottate in un compito di *problem-solving*.

3.2.4 La teoria delle intelligenze multiple

Può essere inserita all'interno della "famiglia" cognitivista anche una teoria oggi molto nota, sviluppata dallo psicologo statunitense Howard Gardner. Nel 1983 Gardner pubblica *Frames of Mind: The Theory of Multiple Intelligences* (trad. it. *Formae mentis*), un testo che suscita un vivace dibattito e che chiama in causa in modo esplicito il sistema educativo.

L'elaborazione di Gardner (1993) prende le mosse dall'insoddisfazione per i "test di misurazione dell'intelligenza", di cui si fa un ampio uso soprattutto in ambito statunitense. Questi test standardizzati – il primo fu messo a punto da Alfred Binet oltre un secolo fa, oggi ne esistono molti – sono "attraenti" perché valutano in modo estremamente sintetico, con un unico punteggio, un costrutto complesso come quello di *intelligenza*. In molti casi, nella pratica educativa e didattica, quel punteggio viene considerato predittivo delle possibilità di successo del soggetto, innescando aspettative che possono avere un pericoloso effetto deterministico sulla sua carriera scolastica.

La principale critica di Gardner riguarda il fatto che i test, in realtà, valutano un tipo ben preciso di intelligenza, o meglio due: quella linguistica e quella logico-matematica, che nel corso della storia culturale dell'Occidente hanno assunto una rilevanza centrale, tanto da essere identificate con l'intelligenza *tout court*. Gardner parte invece da una definizione più generale e comprensiva,

secondo la quale l'intelligenza è la «capacità di risolvere problemi o di creare prodotti che abbiano un valore riconosciuto in uno o più ambienti culturali diversi» (*ibidem*, 12).

Egli non conduce esperimenti, ma analizza una grande mole di dati, tra cui evidenze neuropsicologiche sugli effetti dei danni cerebrali, casi di bambini autistici, bambini prodigio e *idiots savants*, biografie di personaggi del passato, studi antropologici; da tutte queste evidenze ricava un modello che distingue sette *forme di intelligenza*: musicale; corporeo-cinestetica; logico-matematica; linguistica; spaziale; interpersonale; intrapersonale. In un secondo momento ne aggiunge altre due: quella naturalistica e quella filosofico-esistenziale. Queste intelligenze sono una sorta di "dotazione biologica" della specie umana: «Tutti gli individui normali possiedono ciascuna di tali [intelligenze] in una certa misura; gli individui differiscono per il grado in cui possiedono ciascuna [intelligenza] e per il modo in cui esse sono combinate» (*ibidem*, 22).

Le implicazioni di questa teoria in campo educativo sono profonde, a partire dalla critica mossa alla scuola in quanto focalizzata esclusivamente su due specifiche forme di intelligenza, a scapito delle altre. L'indicazione di Gardner è che il sistema educativo dovrebbe anzitutto dare pari "dignità" a tutte. Inoltre dovrebbe valutare e riconoscere il profilo individuale di ciascun alunno e operare in una duplice direzione: da un lato sfruttare il tipo di intelligenza "predominante" per favorire una l'apprendimento dei contenuti; dall'altro, però, non deve trascurare di stimolare anche le intelligenze di cui l'alunno è meno dotato, che sono comunque educabili, entro certi limiti.

La trasposizione didattica di questi principi, tuttavia, non è semplice, anche per la possibile confusione con concetti e costrutti simili ma non sovrapponibili, come per esempio quello di *stile di apprendimento*. McKenzie (2006), dopo avere esaminato la questione, avanza una proposta incentrata sull'utilizzo delle tecnologie digitali, viste come strumenti potenzialmente efficaci per una mediazione didattica incentrata sulla teoria di Gardner. Anzi egli ribalta il discorso, sostenendo che il modello delle intelligenze multiple fornisce una cornice teorica ideale per un utilizzo motivato, consapevole e coerente delle tecnologie in classe:

> L'unico modo per essere certi che le tecnologie emergenti avranno successo nella classe sta nel fare sì che abbiano un solido fondamento nella teoria pedagogica,

che siano implementate in modo ponderato e che siano oggetto di riflessione. Nessuna teoria riesce meglio del modello di Gardner ad armonizzare le tecnologie ai bisogni degli alunni (*ibidem*, 48).

3.3 Curricolo, obiettivi, tassonomie: l'elaborazione didattica dagli anni '50 agli anni '80

Come detto, Comportamentismo e Cognitivismo sono due approcci storicamente presentati come contrapposti: se il primo vedeva l'apprendimento come mera modifica del comportamento manifesto e non si occupava dei meccanismi psicologici sottesi, il secondo era interessato proprio a indagare e spiegare quei meccanismi. Da un punto di vista estremamente generale, tuttavia, è possibile individuare un tratto comune (almeno per quanto riguarda il Cognitivismo "di prima generazione"): entrambi i paradigmi assumono l'esistenza di contenuti di conoscenza precisi e strutturati, da "trasferire" al soggetto che apprende. In altri termini, il *focus* è tutto sui contenuti, sul problema di come articolarli e presentarli per favorire il più possibile il processo di apprendimento, secondo una prospettiva che, in senso lato, si può definire positivistica[3].

Al di là delle influenze più o meno dirette di questa o quella teoria dell'apprendimento, si tratta di un clima culturale che ha profondamente caratterizzato anche il campo dell'elaborazione didattica, nel periodo che va dalla metà del Novecento agli anni '80:

> In sintesi, alla fine degli anni Cinquanta attraverso orientamenti diversi, in certi casi anche contrapposti, quali il comportamentismo skinneriano, lo sviluppo dell'orientamento tassonomico-curricolare e la nascita della scienza cognitiva coi suoi risvolti cibernetico-informatici e psico-linguistici, si prendono le distanze da una tradizione didattica prevalentemente ispirata all'attivismo deweyano. Negli anni immediatamente successivi […] si "implementano" queste originarie formulazioni teoriche attraverso un intenso fervore di applicazioni curricolari ed esperienze operative; suggerimenti di provenienza comportamentistica e cognitivistica tendono ormai ad integrarsi nella proposizione di nuovi modelli didattici e curricolari (Calvani 2000, 50-51).

[3] Un altro aspetto fondamentale che accomuna Comportamentismo e primo Cognitivismo riguarda l'attenzione esclusiva alla sfera cognitiva e la totale assenza delle dimensioni sociali e situazionali dei processi di apprendimento. Se ne parlerà nel paragrafo 3.4.

Un momento fondamentale è costituito dalla Conferenza di Woods Hole, organizzata nel 1959 dalla National Academy of Sciences e presieduta da Jerome Bruner. Vi si riunirono psicologi, pedagogisti ed esperti di varie discipline scientifiche con l'obiettivo ambizioso di ripensare e riorganizzare l'intero sistema scolastico statunitense, ancora fondato sul modello attivista di ispirazione deweyana.

La conferenza è considerata il punto di partenza del cosiddetto *curriculum movement*, un indirizzo di ricerca teso a "ottimizzare" i curricoli delle varie discipline per rendere l'insegnamento più efficace possibile (*ibidem*). Il cardine di queste elaborazioni è la materia di insegnamento, con la sua struttura epistemologica e il suo apparato concettuale; nella dialettica insegnamento-apprendimento, l'attenzione è decisamente spostata sul primo elemento.

Negli stessi anni prendeva le mosse un filone di ricerca – per certi versi collegato al precedente – che ha caratterizzato a lungo il mondo della didattica. Nell'intento di rendere valutabile l'apprendimento, è stato messo a punto il costrutto di *obiettivo di apprendimento*, definito come la *prestazione cognitiva* che si vuole ottenere su un determinato *contenuto*. Entrambi gli elementi – prestazione e contenuto – devono essere descritti in modo preciso, non ambiguo e circoscritto (Frabboni 2007). In questo modo un docente è in grado di organizzare unità di apprendimento estremamente strutturate, con una precisa progressione di "passaggi" dagli esiti misurabili:

> Si tratta dunque di "operazionalizzare" gli obiettivi, vale a dire di non limitarsi a descriverli verbalmente ma di indicare le prove concrete ed i criteri di valutazione che assumiamo come indicatori del conseguimento dell'obiettivo stesso (*ibidem*, 77).

L'impulso fondamentale a questo filone di ricerca, e tuttora il suo esito più noto, è la tassonomia degli obiettivi cognitivi elaborata da Benjamin Bloom, pubblicata nel 1956. Si tratta di una tabella che articola una grande varietà di abilità e prestazioni cognitive in ordine di complessità, applicabili in sede di programmazione agli specifici contenuti di conoscenza[4]. Tra i suoi pregi c'è sicuramente il fatto di avere proposto un lessico molto ricco delle possibili

[4] Gli obiettivi sono raggruppati in sei classi generali, a loro volta articolate in sottocategorie, che descrivono la progressione dai processi cognitivi elementari a quelli "superiori": conoscenza, comprensione, applicazione, analisi, sintesi, valutazione.

prestazioni cognitive, eliminando il termine-passepartout *conoscere*, troppo generico e non operazionalizzabile.

Nei decenni seguenti molti studiosi hanno sviluppato questo filone di ricerca, che ha goduto di grande fortuna in ambito didattico. Si è registrato un vero e proprio proliferare di tassonomie e schemi di classificazione più o meno alternativi, con *focus* differenziati[5] (Parente 2004). Alcuni autori hanno anche cercato di ovviare al principale e più evidente limite di questo approccio, ossia l'attenzione esclusiva per la sfera cognitiva, e hanno proposto tassonomie degli obiettivi affettivi e relazionali.

Senza dubbio, in un certo periodo, queste elaborazioni sono sfociate in un eccesso di tecnicismo, in una iperstrutturazione dell'insegnamento da parte dei docenti della scuola (*ibidem*). Oggi, più correttamente, le tassonomie degli obiettivi sono utilizzate nel modo in cui erano state proposte inizialmente dallo stesso Bloom, ossia come utili strumenti in sede di programmazione didattica, da consultare e impiegare quando necessario, senza ridurre l'insegnamento all'applicazione di astruse griglie analitiche:

> […] in generale oggi si concorda sul fatto che solo una parte limitata dell'apprendimento può essere operazionalizzata; è opportuno che altre finalità rimangano più indeterminate, con la possibilità che esse vengano definite *in itinere* (Calvani 2000, 78-79).

3.3.1 Il modello dell'unità didattica

Tirando le fila di quanto fin qui esposto, si può forse individuare un "prodotto" che riassume i punti fondamentali emersi e che è tuttora protagonista nelle aule scolastiche: l'*unità didattica* (Frabboni 2007; Vannini 2009). I suoi tratti essenziali sono così schematizzati da Gherardi (2010, 39):

Il modello "per unità didattiche"

- Si rifà soprattutto al comportamentismo e a modelli mutuati dal cognitivismo.

[5] Importante quella elaborata in Italia da Frabboni e Arrigo, che sintetizza ulteriormente le sei classi di Bloom in quattro: apprendimenti elementari, intermedi, superiori convergenti, superiori divergenti (Frabboni 2007).

- Ha un taglio oggettivista e razionalista: si assume che le conoscenze e le competenze da acquisire siano delimitabili, rappresentabili, riducibili in parti più semplici.
- Il percorso didattico è derivato in primo luogo dagli obiettivi, strutturato secondo un approccio top-down: ha carattere sistematico e sequenziale.
- L'apprendimento si svolge generalmente in forma astratta, decontestualizzata.
- La valutazione si avvale di un frequente uso di test in ingresso, *in itinere*, alla fine.
- Ci si può avvalere di tecnologie più specifiche come l'istruzione programmata o il *mastery learning*.
- La materia di studio, come insieme di conoscenze strutturate, individua ciò che deve essere appreso.
- Si propone principalmente di salvaguardare il principio dell'uguaglianza della conoscenza: portare tutti alla stessa meta.

Naturalmente quello qui delineato è una sorta di modello ideale: all'interno di un'unità didattica è sempre possibile prevedere anche attività di ricerca/scoperta, discussione in grande gruppo, ecc.; tuttavia, di solito c'è la necessità di circoscriverle e indirizzarle verso un esito preordinato (a differenza delle stesse tecniche utilizzate in chiave realmente progettuale e "aperta").

Il modello dell'unità didattica è stato e continuerà a essere uno strumento fondamentale nei processi di insegnamento/apprendimento [6]. L'importante è conoscerne a fondo i presupposti epistemologici ed essere consapevoli del fatto che non è *sempre* la modalità più idonea ed efficace per *tutti* i traguardi formativi, gli obiettivi di apprendimento, i campi del sapere.

3.4 Costruttivismo

Gli approcci fin qui descritti, che si rifanno a una base sostanzialmente comune, a una «particolare "solidarietà" tra modello della conoscenza (come acquisizione-elaborazione di informazioni), modello curricolare (di tipo gerarchico-sequenziale) e modello didattico (di tipo istruttivo)» (Calvani 2000, 53), entrano in crisi negli anni '80.

In parte ciò si inserisce un mutamento di pensiero molto più generale, una sorta di "rivoluzione epistemologica" di portata amplissima che ha investito le

[6] Inoltre è il modello che prevale nettamente nei libri di testo.

fondamenta della cultura occidentale sullo scorcio del secolo, con profonde ripercussioni in tutti i campi del sapere: per spiegare un mondo sempre più globalizzato e interconnesso, le catene causali si rivelano inadeguate, e vengono elaborate nuove categorie fondamentali quali quelle di *complessità* e di *sistema*.

È comunque possibile identificare alcune debolezze di fondo del paradigma razionalista-trasmissivo su cui si sono concentrate critiche e insoddisfazioni, in particolare l'approccio eccessivamente "ingegneristico", che ha condotto a una visione dell'apprendimento come una fredda successione di processi che trasformano un *input* in un *output*[7]. In questo senso l'analogia tra mente umana e computer si è eccessivamente irrigidita, portando all'immagine di una «mente disincarnata», secondo una critica avanzata da Bruner, un grande protagonista di quella stagione che ha saputo rivedere e sviluppare il proprio pensiero in nuove direzioni. In sintesi, per i comportamentisti e i cognitivisti degli anni '60-'70, il soggetto che apprende è un individuo isolato, avulso da qualsiasi contesto sociale e situazionale, e ridotto alla sola sfera cognitiva (quella emotiva e quella relazionale non sono prese in considerazione).

Gli orientamenti che si rifanno alla matrice costruttivista rappresentano una decisa reazione a questi aspetti problematici. Va premesso che, rispetto ai paradigmi analizzati in precedenza, con *Costruttivismo* si indica generalmente un campo assai variegato, una "costellazione" di teorie, proposte e orientamenti accomunati da alcuni presupposti di base: il termine "costruttivismo" è una sorta di «"vessillo" sotto la cui egida si vanno attualmente raccogliendo epistemologi, studiosi dell'area cognitiva, progettisti educativi, tecnologi» (Calvani 2000, 79).

Gli elementi fondamentali di una visione costruttivista della conoscenza si possono riassumere come segue (*ibidem*; Mason 2006; Nigris, Negri & Zuccoli 2007; Carletti & Varani 2005).

- La conoscenza è il prodotto di una *costruzione attiva* da parte del discente, in netto contrasto con l'idea di una semplice trasmissione di informazione. In questo senso viene rinforzata e approfondita una concezione già presente in molti approcci di stampo cognitivista (quelli

[7] Il corrispettivo a livello di pratica didattica è stato un eccesso di formalizzazione, che spesso ha portato i docenti a dibattersi in un proliferare di tassonomie, griglie, liste di obiettivi iper-articolate.

che sottolineano il carattere attivo dell'elaborazione delle conoscenze e delle rappresentazioni mentali).

- La conoscenza è sempre *situata*: il *contesto* entro cui si svolge il processo di insegnamento-apprendimento non è un semplice "sfondo neutro", bensì una dimensione cruciale di un evento che è essenzialmente relazionale. Entra in gioco qui la concezione sistemica cui si è accennato poco sopra.
- Alla base delle teorizzazioni che si situano entro il paradigma costruttivista c'è una concezione dell'ontologia e dell'epistemologia che si oppone al senso comune, secondo cui esiste una *realtà* esterna di cui la conoscenza sarebbe un semplice riflesso. Al contrario, per i costruttivisti «la realtà ontologica è al più irrilevante (se esiste): la realtà va intesa epistemologicamente» (Lodi 2014, 44). In altri termini, non esiste qualcosa di oggettivo di cui il soggetto che apprende deve appropriarsi; soggetto e oggetto di conoscenza si definiscono solo nella relazione reciproca (Carletti 2005). La conoscenza è una costruzione attiva di significato, in cui vengono elaborati stimoli, percezioni e sensazioni, tramite un fondamentale strumento mediatore: il linguaggio. Perciò si tratta di una costruzione allo stesso tempo individuale e sociale, perché avviene sempre all'interno di una determinata cultura, che fornisce le categorie concettuali con cui vengono elaborati gli stimoli, e che rende anche comunicabili le conoscenze. In questa prospettiva perde di senso il concetto stesso di *verità*: «Si assume quindi un approccio di carattere pragmatico e non ontologico, focalizzando l'attenzione sul processo di costruzione dei significati e della loro comunicazione Il criterio della verità viene sostituito dal criterio di adattamento funzionale e di *viabilità* [o *adattabilità*] secondo il quale i concetti, costruiti a partire dalle regolarità che si incontrano nell'esperienza, hanno prima di tutto una funzione predittiva, sono strumentali all'azione e vengono appunto definiti viabili quando permettono di raggiungere uno scopo pratico» (*ibidem*, 17).
- Nel ricco e frastagliato panorama di modelli teorici e proposte operative di stampo costruttivista, è possibile individuare due direttrici principali. La prima si concentra sulla conoscenza come *costruzione individuale*,

recuperando la visione di Piaget dello sviluppo come processo proattivo da parte dell'individuo; in questa prospettiva si inseriscono molte teorie cognitiviste "di seconda generazione" (si pensi per esempio agli studi sulle strategie cognitive e sulla metacognizione). La seconda direttrice, che si rifà invece alla lezione di Vygotskij (ampiamente ripreso da Bruner e oggi punto di riferimento per molta ricerca pedagogica), è quella *sociocostruttivista*, che focalizza l'attenzione sulla dimensione intersoggettiva dell'apprendimento ed enfatizza il ruolo della cultura; gran parte delle elaborazioni della didattica più recente, dal *cooperative learning* alle comunità di apprendimento, è scaturita da questa prospettiva.

Restringendo lo sguardo sulle implicazioni in campo educativo, il Costruttivismo promuove e stimola una didattica fondata sui seguenti principi fondamentali.

- L'apprendimento deve basarsi il più possibile sull'*esperienza* diretta degli alunni, piuttosto che su contenuti astratti comunicati verbalmente (spiegazione frontale o libro di testo).
- I significati sono frutto di una *costruzione collettiva*; il sapere si acquisisce dunque in modo negoziale, nell'ambito delle relazioni in gioco nel contesto-classe.
- Occorre riservare una particolare attenzione ai *processi* più che ai prodotti, anche in sede di valutazione. In quest'ottica, l'errore assume una valenza positiva, perché apre una prospettiva di osservazione e di analisi dei processi cognitivi.

Dopo questa panoramica generale, saranno ora presi in esame tre campi di ricerca più specifici, che fanno parte della "famiglia" costruttivista e sono particolarmente rilevanti per la proposta didattica che verrà articolata nell'ultimo capitolo.

3.4.1 La motivazione ad apprendere

La ricerca sul ruolo della *motivazione* nei processi di apprendimento si è sviluppata soprattutto negli ultimi trent'anni, all'interno di modelli e quadri teorici

diversi. Se da un lato l'interesse per questo ambito di studio è molto vivo, è impossibile identificare un modello univoco a cui riferirsi: «La ricerca sulla motivazione è, oggi, un arcipelago, di cui teorie diverse hanno esaminato e concettualizzato vari aspetti, non di rado usando termini diversi per concetti molto simili» (Boscolo 2006, 92).

Seguendo questo autore, si possono individuare tre dimensioni attorno alle quali si è mossa la ricerca.

- La prima concerne la motivazione come *disposizione all'apprendimento*, nell'ottica di un ruolo attivo del discente, secondo la prospettiva costruttivista: «La motivazione può essere definita come l'attivazione e la direzione del comportamento» (*ibidem*, 94).
- La seconda dimensione riguarda il modo in cui il discente si percepisce in relazione a un compito e alle prospettive di successo o insuccesso; tale *percezione* influenza la reale prestazione, e ne è a sua volta rafforzata o indebolita. In questo ambito sono stati elaborati costrutti come l'autopercezione di competenza, il senso di efficacia e il concetto di sé.
- La terza riguarda l'*autoregolazione* nello svolgimento dell'attività, ossia le strategie cognitive impiegate, il monitoraggio del processo, la gestione delle emozioni.

Il secondo e il terzo punto presentano evidenti aree di sovrapposizione con il campo di indagine della metacognizione, con l'integrazione della componente emotiva. Per i nostri scopi, ci si concentrerà sulla prima dimensione, ossia sull'attivazione dei processi di apprendimento, che può scaturire da un reale desiderio di competenza dell'individuo, oppure da stimoli e spinte "esterni". Il modello più accreditato prevede un *continuum* che va dalla *motivazione intrinseca* alla *motivazione estrinseca*. Naturalmente la motivazione intrinseca porta a un apprendimento più efficace, mentre nella scuola è inevitabile sperimentare molte situazioni di motivazione estrinseca (studio perché sono obbligato a farlo, perché non voglio un brutto voto, ecc.). Un insegnante deve anzitutto cercare di mantenere la motivazione intrinseca degli allievi, compito non così scontato:

> […] cercare di creare e mantenere per quanto possibile – e quando possibile – condizioni favorevoli al comportamento intrinsecamente motivato: un certo grado di autonomia degli allievi, la possibilità di soddisfare il bisogno di competenza, il lavoro collaborativo in classe, le attività "sfidanti" (*ibidem*, 106).

Si tratta cioè di non "affossare" la motivazione intrinseca, già presente in uno studente per un dato argomento o campo disciplinare, con attività ripetitive, banali o troppo difficili. Più complicato è agire sull'altro lato del *continuum*: molto semplicemente, come si può rendere "intrinseca" la motivazione allo studio di una materia che non piace?

Viene qui in aiuto il costrutto di *interesse*, un altro concetto del senso comune che la ricerca ha cercato di analizzare e operazionalizzare. L'interesse, definibile come un particolare tipo di relazione che intercorre tra un individuo e un oggetto all'interno di un contesto, può essere *di tratto* (interesse personale) o *di stato* (interesse situazionale). Mentre il primo costituisce una disposizione relativamente stabile e durevole del soggetto verso una classe di oggetti (essere appassionati di fantascienza, o di cucina, o di bricolage), «la ricerca recente si è focalizzata soprattutto sull'interesse situazionale che […] è manipolabile – si possono predisporre elementi nuovi, originali o insoliti per attivare l'interesse – e quindi più rilevante per l'istruzione» (*ibidem,* 108). In particolare, tra i caratteri che un compito dovrebbe avere per suscitare interesse (e quindi motivazione ad apprendere) ci sono la novità, la possibilità di interazione sociale e l'incongruenza, che produce dissonanza cognitiva e spinge a cercare un'integrazione fra le conoscenze pregresse e le nuove informazioni "spiazzanti".

3.4.2 Il Costruzionismo

La teoria dell'apprendimento denominata Costruzionismo è stata sviluppata da Seymour Papert, matematico sudafricano recentemente scomparso. Papert è stato una figura chiave in diversi campi, profondamente segnati dai suoi studi e dalle sue sperimentazioni. Alla fine degli anni '50 ha lavorato a Ginevra al fianco di Jean Piaget, e questa esperienza ha stimolato il suo interesse per lo studio dei processi di apprendimento nei bambini; in seguito si è trasferito a Boston al MIT, dove nel corso degli anni ha potuto sviluppare le sue ricerche in diverse direzioni: si è occupato – insieme a Marvin Minsky – di intelligenza artificiale; ha realizzato

il Logo, un linguaggio di programmazione informatica a scopo didattico, vera e propria pietra miliare nell'uso dei computer in educazione[9]; è stato tra i fondatori del MIT Media Lab, uno dei più importanti centri di ricerca e innovazione tecnologica al mondo.

Il Costruzionismo è la cornice teorica che collega molte di queste ricerche ed esperienze. I suoi principi di base si inseriscono nel più ampio paradigma costruttivista: l'apprendimento è un processo attivo da parte del discente, che parte dai dati dell'esperienza e conduce all'elaborazione, all'integrazione e alla modifica di modelli mentali; il modo migliore per imparare non è ricevere informazioni bensì esplorare, scoprire, risolvere problemi.

L'aspetto più originale dell'elaborazione teorica di Papert consiste nel ruolo centrale assegnato alla "manipolazione" di oggetti e materiali per la realizzazione di un progetto che sia significativo per lo studente. Papert utilizza il concetto di *artefatti cognitivi* per designare qualunque "oggetto" artificiale (non necessariamente tangibile: può essere anche un software o un linguaggio di programmazione) che possa costituire un ponte tra la realtà sensibile e il modello mentale del discente; un oggetto dunque su cui operare per raggiungere un determinato obiettivo, e poi riflettere su tale operato e sugli effetti prodotti, iterando il "circolo". La conoscenza si struttura in questa dinamica tra intervento sul "mondo esterno" e confronto con gli schemi concettuali.

> Uno dei miei principi educativi fondamentali è che la costruzione che avviene "nella testa" risulta particolarmente efficace quando è supportata da una costruzione di natura più pubblica, "nel mondo" – un castello di sabbia o una torta, una casa fatta con i Lego o un'azienda, un programma per computer, una poesia o una teoria dell'universo. In quello che intendo con "nel mondo" è compreso il fatto che il prodotto possa essere mostrato, discusso, esaminato, valutato e ammirato. È là fuori… Perciò il costruzionismo, la mia personale declinazione del costruttivismo, […] assegna un'importanza particolare al ruolo della costruzione nel mondo come supporto per quella nella mente (Papert 1993, 142-143; traduzione mia).

[9] Come in molti altri casi, si è trattato di una di quelle intuizioni che si definiscono "visionarie", in netto anticipo sui tempi: Papert ha pensato e realizzato il Logo in un momento in cui i personal computer "alla portata di tutti" (anche dal punto di vista economico) erano ben di là da venire, così come le interfacce *user-friendly*.

Il Logo è proprio uno di questi «oggetti per pensare», espressamente progettato per una didattica costruzionista[10]. È molto interessante notare che fin dall'inizio l'intento di Papert e dei suoi collaboratori non era affatto quello di insegnare l'informatica: il Logo veniva utilizzato per l'apprendimento della matematica e della geometria, ma soprattutto di concetti computazionali, strategie di risoluzione di problemi, modelli di pensiero. In altri termini, in un periodo in cui l'informatica era ancora molto "giovane", Papert aveva già elaborato l'idea e intuito l'importanza del *pensiero computazionale* come viene inteso oggi (non a caso è stato proprio lui a coniare la locuzione). Questi principi sono espressi nel suo libro più famoso, che si intitola significativamente *Mindstorms. Children, Computers, and Powerful Ideas*, e nei decenni seguenti Papert ha continuato a promuovere con forza l'integrazione delle tecnologie informatiche nella scuola.

Oltre all'apprendimento come costruzione attiva e al ruolo degli artefatti cognitivi, è opportuno sottolineare altri due aspetti estremamente interessanti di questo approccio, tra loro correlati. Il primo consiste nell'attenzione e nell'importanza assegnata ai processi metacognitivi: «Cominciai a vedere come i bambini che avevano appreso a programmare un computer potevano servirsi di modelli informatici per riflettere su come si pensa, per apprendere come si apprende, e, così facendo, per sviluppare le loro capacità di psicologi e di epistemologi» (Papert 1980, 3). Il secondo riguarda la dimensione intersoggettiva in cui gli apprendimenti e la metacognizione si possono elaborare nel modo più efficace e fruttuoso, all'interno di una comunità di apprendimento in cui presentare i propri progetti, discuterli, esplicitare le strategie usate per raggiungere l'obiettivo.

Per tutti gli aspetti fin qui illustrati, l'approccio costruzionista è l'orizzonte teorico e metodologico privilegiato entro cui verrà articolata la proposta didattica del Capitolo 5.

3.4.3 Apprendimento e creatività

Fin dagli anni '50 e '60, al di fuori del paradigma *Human Information Processing* che si andava imponendo come dominante, si è sviluppato un filone di studi attorno al concetto di *creatività*, soprattutto per opera dello psicologo

[10] Papert è stato anche uno dei pionieri della robotica educativa.

statunitense J.P. Guilford. Questo approccio teorico rifiuta la concezione della creatività come un "talento" innato che solo alcune persone possiedono, e la considera invece una capacità cognitiva che può essere "educata" e sviluppata. Guilford introduce la distinzione tra *pensiero convergente* e *pensiero divergente*: il primo individua la soluzione più convenzionale a un dato problema, applicando in modo univoco le conoscenze e le strategie a disposizione; il secondo, invece, genera soluzioni multiple, spesso impreviste, ricombinando in maniera creativa idee, concetti e conoscenze già posseduti.

La stretta connessione fra creatività e pensiero divergente è decisiva per uscire, come detto, dall'ambito dei talenti naturali ed entrare in quello dell'intervento educativo: trattandosi di una tipologia di processi cognitivi, è possibile favorirne lo sviluppo attraverso un ambiente di apprendimento che valorizzi la risoluzione di problemi, il conflitto cognitivo, la discussione e il confronto tra differenti strategie.

Questa tematica ha assunto un'importanza crescente nel dibattito pedagogico più recente. Nella società della conoscenza e dei servizi, in un mondo del lavoro sempre più competitivo in cui le attività produttive richiedono sempre meno manodopera, la capacità di *innovazione* è diventata la chiave di volta per il successo professionale. E le agenzie educative si stanno ponendo l'obiettivo di promuovere questa competenza trasversale, sempre più strategica, fin dai primi anni di scolarizzazione.

Non a caso, una delle otto "competenze chiave per l'apprendimento permanente" indicate dall'Unione Europea (si veda il paragrafo 1.3.3) riguarda *il senso di iniziativa e l'imprenditorialità*, che indubbiamente – accanto a competenze di carattere organizzativo – presuppongono la capacità di elaborare idee e progettare soluzioni innovative. Mentre le *Indicazioni nazionali per il curricolo* del 2012 elencano in modo significativo tra le finalità di un ambiente di apprendimento efficace:

> Favorire l'esplorazione e la scoperta, al fine di promuovere il gusto per la ricerca di nuove conoscenze. In questa prospettiva, la problematizzazione svolge una funzione insostituibile: sollecita gli alunni a individuare problemi, a sollevare domande, a *mettere in discussione le conoscenze già elaborate*, a trovare appropriate piste d'indagine, a *cercare soluzioni originali* (MIUR 2012, 35; corsivo mio).

Dato l'argomento di questo lavoro, è molto interessante esaminare il particolare approccio all'apprendimento creativo proposto dal MIT Media Lab. L'iniziativa denominata *Media Lab Learning* si basa proprio sui presupposti appena illustrati: «Viviamo in un mondo che sta cambiando più rapidamente di qualunque epoca passata. Molto di ciò che impariamo oggi sarà già obsoleto domani. Il successo dipende dalla nostra capacità di *pensare e agire in modo creativo*»[11].

Il modello di *Creative Learning* sviluppato dal Media Lab è anzitutto una modalità di lavoro adottata dagli stessi ricercatori dell'istituto e un orizzonte ideale entro cui vengono sviluppati i progetti, con l'obiettivo di promuoverlo presso il maggior numero di utenti, secondo lo slogan «How the MIT Media Lab learns – and how everyone else can learn this way too»[12]. In estrema sintesi, si basa su quattro principi fondamentali:

- *Projects*: si apprende meglio quando si lavora a progetti concreti (in pieno spirito costruzionista);
- *Peers*: esprime la dimensione sociocostruttivista dell'apprendimento (collaborazione, condivisione, lavoro di squadra);
- *Passion*: l'apprendimento è più efficace quando l'argomento – o meglio il progetto – ci interessa e ci coinvolge, come dimostrato anche dagli studi sulla motivazione;
- *Play*: l'approccio ludico è fondamentale, perché favorisce l'esplorazione e l'intraprendenza, oltre a interagire positivamente con i punti precedenti (motivazione, collaborazione).

3.5 La didattica costruttivista: gli ambienti di apprendimento

I principi del paradigma costruttivista dell'apprendimento hanno ispirato negli ultimi due-tre decenni un'intensa elaborazione didattica, sfociata in quello che oggi appare come un modello ampiamente condiviso, tanto nel campo della ricerca quanto in quello del sistema educativo (per lo meno nelle sue linee

[11] http://learn.media.mit.edu/creative-learning (traduzione mia).

[12] *Ibidem*.

programmatiche). Un modello che comprende entrambe le direttrici principali del Costruttivismo – quella individuale, della conoscenza come costruzione attiva, e quella sociale-interazionista, dell'apprendimento come costruzione collettiva situata in un contesto relazionale – e che si rifà al principio fondamentale della *centralità del discente* e della sua esperienza.

Si tratta di una ripresa forte e consapevole della prospettiva sviluppata nei primi decenni del Novecento dalle cosiddette "scuole nuove" (o "scuole attive"), un'esperienza fondamentale e per certi versi rivoluzionaria che ha coinvolto pedagogisti, educatori e insegnanti sia in Europa che negli Stati Uniti (Nigris 2007b; Calvani 2000). Sul piano teorico, il "referente" principale di quel movimento è stato John Dewey, filosofo che ha profondamente segnato il pensiero pedagogico del secolo scorso. Tuttavia l'attivismo, che ha avuto il grande merito di mettere radicalmente in discussione il tradizionale modello trasmissivo del sapere, è scivolato nel corso degli anni in un eccesso di spontaneismo, in cui il "fare" del bambino rischiava di rimanere fine a sé stesso, senza condurre a un apprendimento strutturato e formalizzato. L'affermarsi del Comportamentismo costituiva proprio una reazione verso questa "deriva spontaneista" (contro la quale, peraltro, lo stesso Dewey aveva più volte messo in guardia).

L'attuale didattica costruttivista recupera la grande lezione dell'attivismo, ma la rielabora alla luce di decenni di studi e ricerche che consentono di evitare quella degenerazione. Oggi la professionalità docente si esprime proprio nella progettazione, costruzione, gestione attenta e consapevole delle situazioni di apprendimento in cui situare l'esperienza attiva del discente:

> Il costruttivismo per molti aspetti è dunque un *déjà vu*, la miscela degli elementi che riscopre ha una lontana origine: sul piano didattico non si può evitare di mettere in rapporto il costruttivismo con l'attivismo. […] L'esigenza di uscire da un apprendimento formale, astratto e decontestualizzato, a favore di un apprendimento basato su compiti autentici, situato, rimanda inequivocabilmente alle riflessioni sul ruolo dell'esperienza in educazione presenti in tutta l'opera di Dewey. Bisogna però anche comprendere che le proposte didattiche di taglio costruttivistico hanno ben poco a che fare con un banale spontaneismo attivistico; in ogni progetto la costruzione di una "impalcatura" (*scaffolding*) […] è molto forte e strutturata: si dà spazio allo studente agendo più energicamente sul contesto con norme cooperative

> molto precise, forte intervento di responsabilizzazione, presenza ed impiego analitico di dispositivi e strumentazioni ecc. (Calvani 2000, 84).

Il concetto chiave, emerso con forza nella ricerca e recepito anche nelle indicazioni ministeriali, è quello di *ambiente di apprendimento*, in un'accezione ricca e complessa che include lo spazio fisico, gli "attori" coinvolti, i tempi, i materiali e gli strumenti, gli obiettivi educativi e didattici, le norme di comportamento, le relazioni interpersonali, la tonalità emotiva.

> In quest'ottica il docente diviene progettista di ambienti di apprendimento, costruiti intenzionalmente per consentire percorsi attivi e consapevoli in cui lo studente sia orientato ma non diretto. Luoghi ricchi e variegati per esperienze possibili e materiali di lavoro, caratterizzati da una forte struttura, ma allo stesso tempo aperti e polisemici in cui gli studenti possano aiutarsi reciprocamente, utilizzando una varietà di strumenti e di risorse in attività guidate. Un ambiente arricchito da momenti di riflessione individuale e collettiva, da domande euristiche e da consegne che lo studente può affrontare autodeterminando modi e percorsi, sulla base del proprio stile, degli interessi e delle strategie personali (Carletti & Varani 2005, 47).

In questo contesto si è affermato il modello dell'*atelier*, un'aula o uno spazio specificamente attrezzato per determinati tipi di attività. Ma l'*atelier* può funzionare anche in senso metaforico: la stessa aula della classe si può trasformare in laboratorio con pochi accorgimenti, perché il fulcro è la *didattica laboratoriale*, non la quantità di materiali e strumenti di cui si dispone (Frabboni 2007).

Le caratteristiche e le implicazioni più rilevanti di questa impostazione generale si possono riassumere come segue.

- Al posto del modello dell'unità didattica, si privilegiano le *didattiche attive*, l'*apprendimento per scoperta* e l'*apprendimento per problem-solving*. Un punto essenziale consiste nel porre gli studenti di fronte a fenomeni o problemi realistici e complessi (e non semplici "esercizi"). Una *situazione-problema*, per essere tale, deve riguardare un ambito del sapere o dell'esperienza personale che sia almeno in parte familiare agli studenti; provocare un conflitto cognitivo con la conoscenza pregressa; aprire a ipotesi e percorsi cognitivi differenziati e non predeterminati (Nigris 2007a, 102). Un'altra tipologia di ispirazione costruttivista

riguarda la *didattica per progetti*, dall'esito il più possibile aperto e non prevedibile (Frabboni 2007; Vannini 2009).

- L'accento è posto sull'apprendimento come *co-costruzione della conoscenza in un contesto collaborativo* (in opposizione sia all'"isolamento" del discente nelle prospettive comportamentista e cognitivista HIP, sia alle dinamiche competitive). La valorizzazione della dimensione collaborativa è motivata anzitutto dalla maggiore efficacia per gli apprendimenti individuali, ma è importante anche per lo sviluppo di competenze trasversali sempre più centrali per la formazione di cittadini e lavoratori della società odierna (lavoro in *team* e in rete, capacità di mediazione e negoziazione; tolleranza, valorizzazione delle diversità, decentramento, empatia) (Carletti & Varani 2005; Comoglio & Cardoso 1996).
- La prospettiva costruttivista degli ambienti di apprendimento costituisce anche un contesto ideale per l'integrazione didattica delle *tecnologie*, in particolare quelle digitali. Esse sono viste come *mediatori didattici*, ciascuna con le proprie peculiarità, potenzialità e limiti, da utilizzare in modo consapevole e motivato sulla base dell'impostazione didattica e metodologica che si intende adottare di volta in volta. Pertanto non ha senso la polarizzazione "tecnologie sì/no": ciò che conta è che qualunque strumento (anche il libro di testo o la lavagna!) venga impiegato in maniera coerente all'interno della cornice didattica.
- Rispetto alla didattica trasmissiva, muta completamente il ruolo dell'insegnante, e la sua professionalità richiede competenze e saperi ulteriori rispetto a quelli disciplinari. «In sintesi, il docente sarà responsabile della predisposizione dei contesti fisici e degli spazi, così come della scelta di materiali idonei e vari; avrà il ruolo di individuare compiti mirati che aprano al dialogo e alla riflessione e non al semplice fare, che coinvolgano il corpo e la mente, le strutture mentali e i processi socioaffettivi ed emotivi; sarà chiamato a costruire un clima aperto e accettante in cui i discenti si sentano di poter esprimere opinioni e manifestare contrasti e conflitti così come di mettere in gioco i loro vissuti più profondi» (Nigris 2007a, 120). Un concetto cruciale è quello di *scaffolding*, proposto da Bruner, che condensa questa idea

dell'insegnante come "costruttore di impalcature" che sostengano l'apprendimento attivo da parte degli studenti.

- L'*errore* assume una valenza positiva, in quanto spia delle strategie cognitive adottate dal discente. «Adottare […] una didattica che parte dall'esperienza significa considerare l'errore come parte del processo e cercare di capirne l'origine, per comprendere quando l'allievo sta procedendo per strade inedite e inaspettate ma utili a trovare una soluzione, oppure procedendo in modo da rendere il bambino attivamente consapevole del suo percorso mentale» (*ibidem*, 108). È evidente qui l'importanza della *dimensione metacognitiva*, altro punto essenziale della didattica costruttivista (Varani 2005), che a livello di apprendimento individuale include molte acquisizioni teoriche di derivazione cognitivista.
- Anche la *valutazione* deve armonizzarsi con i principi e le modalità didattiche esposti sopra. Perciò, rispetto alle esercitazioni ripetitive o alla verifica di conoscenze e abilità puntuali, l'insegnante dovrebbe privilegiare un'*osservazione* attenta dello svolgimento delle attività da parte di ciascun alunno (per cogliere i processi oltre ai prodotti), e proporre compiti complessi, che richiedano la mobilitazione e la selezione di competenze e strategie, secondo l'approccio della *valutazione autentica* (Capperucci 2011).

È interessante notare come questi principi siano stati pienamente recepiti nelle *Indicazioni nazionali per il curricolo* del 2012. Nella parte dedicata alla scuola del primo ciclo, infatti, viene dedicato un paragrafo specifico all'ambiente di apprendimento (MIUR 2012, 34-35), che deve:

- valorizzare l'esperienza e le conoscenze degli alunni;
- attuare interventi adeguati nei riguardi delle diversità;
- favorire l'esplorazione e la scoperta;
- incoraggiare l'apprendimento collaborativo;
- promuovere la consapevolezza del proprio modo di apprendere;
- realizzare attività didattiche in forma di laboratorio.

Il panorama delle metodologie che si rifanno alla prospettiva costruttivista degli ambienti di apprendimento è molto ricco e variegato, e comprende per esempio didattiche attive come il *brainstorming*, la discussione, l'approccio ludico o ludiforme, il *role playing* (Nigis, Negri e Zuccoli 2007; Gherardi 2013). Per concludere verrà esaminata una metodologia specifica, che riveste un'importanza particolare ed è ormai un patrimonio acquisito nella prassi didattica a scuola.

3.5.1 Il *cooperative learning*

In letteratura si sottolinea una differenza fondamentale tra *collaborazione* e *cooperazione*: il primo termine è più comprensivo e in un certo senso più generico, e include una varietà di pratiche didattiche non necessariamente codificate in modo rigoroso (per esempio i cosiddetti "lavori di gruppo", nelle varie declinazioni possibili); il secondo implica invece un riferimento a una metodologia didattica specifica, sia pure con diverse varianti: il *cooperative learning*.

In prima approssimazione, il *cooperative learning* può essere definito come «un insieme di tecniche di conduzione della classe nelle quali gli studenti lavorano in piccoli gruppi per attività di apprendimento e ricevono valutazioni in base ai risultati conseguiti» (Comoglio & Cardoso 1996, 24). Questa metodologia presenta però alcuni aspetti caratterizzanti, riassumibili nei seguenti punti (Cacciamani 2008; Comoglio & Cardoso 1996).

- *Interdipendenza positiva*: è uno dei caratteri centrali, e distingue il CL da altre forme di collaborazione. Non solo il raggiungimento dell'obiettivo del gruppo dipende dall'apporto di ciascun membro, ma anche i singoli membri dipendono dagli altri per la propria "parte" di lavoro; «l'interdipendenza positiva promuove la comunicazione, il dare e ricevere aiuto, lo scambio di informazioni, il sentirsi accettati e l'accettare l'altro proprio perché ognuno può offrire le proprie risorse ed è consapevole che la sua partecipazione personale è importante e necessaria per la riuscita comune» (Cacciamani 2008, 34).
- *Responsabilità individuale e di gruppo*, connessa con le modalità di formazione e organizzazione dei gruppi (di 3-5 membri ciascuno).

Questi vengono costituiti dall'insegnante, secondo «criteri di eterogeneità in relazione sia alle caratteristiche personali che alle abilità dei membri» (Comoglio & Cardoso 1996, 24). Inoltre a ogni membro viene assegnato un *ruolo* specifico, che favorisce l'assunzione di responsabilità individuale e promuove dinamiche di *leadership* condivisa.

- *Interazione faccia a faccia*: oltre al raggiungimento del "risultato", viene sottolineata l'importanza del clima relazionale instaurato all'interno del gruppo e delle dinamiche interpersonali.
- *Competenze sociali*: sono in parte richieste come prerequisito per il buon funzionamento del gruppo, in parte sviluppate e affinate proprio nel lavoro cooperativo, e riguardano quelle competenze trasversali (comunicare il proprio punto di vista e ascoltare quello degli altri, decentrarsi, negoziare decisioni, risolvere conflitti) cui si è fatto cenno poco sopra.
- *Valutazione individuale e di gruppo*: una peculiarità del CL rispetto ad altre metodologie collaborative è che deve essere fatta anche una valutazione individuale, eventualmente ponderata con quella di gruppo. Il tema è importante e complesso: è necessario valutare non solo i prodotti ma anche i processi, utilizzando determinati strumenti e tecniche di rilevazione, monitoraggio e valutazione. Inoltre, di solito è previsto anche il coinvolgimento degli studenti stessi in un'attività di autovalutazione, che integra quella del docente.

4. SCRATCH

Scratch è un ambiente di programmazione visuale, sviluppato dal Lifelong Kindergarten del MIT Media Lab sotto la direzione di Mitchel Resnick, allievo e "successore" di Seymour Papert[1]. La prima versione è stata rilasciata nel 2003, mentre dal 2007 è possibile condividere online i propri progetti; la versione attualmente disponibile è la 2.0, che si può scaricare e installare gratuitamente su sistemi Windows, MacOS e GNU/Linux, oppure utilizzare online tramite browser, collegandosi al sito https://scratch.mit.edu/.

Oggi Scratch è utilizzato in oltre 150 nazioni e tradotto in più di 40 lingue; gli utenti registrati superano i 13 milioni, i progetti condivisi in rete sono più di 16 milioni. La maggior parte degli utenti ha un'età compresa fra 7 e 16 anni[2].

Fin dall'inizio, l'obiettivo generale degli sviluppatori non era costruire uno strumento per "insegnare a programmare", bensì *promuovere presso il pubblico più ampio possibile la fluency con le tecnologie informatiche* (si veda 1.3.2), in una prospettiva consapevolmente ed esplicitamente orientata alla diffusione del *pensiero computazionale* (Resnick et al. 2009). L'utente ideale, perciò, non è il bambino già interessato all'informatica, che desideri iniziare a programmare con un linguaggio alla sua portata, ma potenzialmente *tutti* i bambini e i ragazzi che utilizzano quotidianamente le tecnologie digitali, limitandosi però al ruolo di utenti "passivi".

La progettazione di Scratch si è basata in modo deciso sui principi del Costruzionismo (3.4.2) e dell'apprendimento creativo (3.4.3), che in generale ispirano tutta l'attività del Media Lab: apprendimento per scoperta e per *problem-solving*, mediato dallo sviluppo di progetti concreti; cooperazione e confronto tra pari; approccio ludico e creativo. In particolare, il gruppo di lavoro guidato da Resnick ha seguito tre principi-guida nel *design* di Scratch: «make it more *tinkerable*, more *meaningful*, and more *social* than other programming environments» (*ibidem*, 63; corsivi miei).

[1] Scratch può essere considerato un "discendente" del Logo.

[2] https://scratch.mit.edu/statistics/ (consultato il 28 settembre 2016).

- L’aggettivo *tinkerable* rimanda al *tinkering*, un campo oggi di grande interesse e attualità in ambito educativo. Il termine, di cui non esiste una traduzione sintetica in italiano, si riferisce alla possibilità di maneggiare oggetti e materiali (in questo caso informatici) in modo inizialmente anche casuale, senza un progetto definito, lasciandosi guidare dal piacere di comporre, assemblare, “trafficare”, per poi esaminare il risultato, modificare, riesaminare ecc. Indica cioè un approccio di tipo *bottom-up* alla manipolazione di artefatti cognitivi, che via via si intreccia con la progressiva costruzione di modelli mentali, e dunque con la strutturazione di processi *top-down*. Un esempio classico è quello del comportamento dei bambini messi di fronte a un set di mattoncini Lego senza istruzioni.
- Anche l’aspetto della “significatività” (*meaningful*) è tipico dell’approccio costruzionista, in cui è essenziale che il progetto concreto su cui lo studente lavora sia per lui interessante e ricco di senso, con evidenti correlazioni positive con la motivazione all’apprendimento. Questo principio ha ispirato la grande duttilità di Scratch in merito alle tipologie di prodotti realizzabili (si veda 4.3).
- L’ultimo aspetto concerne la dimensione sociale e interazionale del paradigma costruttivista/costruzionista, e verrà esaminato nel paragrafo 4.2.

Tenendo presenti questi principi, sui quali si incentreranno anche le proposte avanzate nel capitolo 5, verranno ora esaminate alcune caratteristiche particolarmente rilevanti di Scratch.

4.1 L’interfaccia

L’interfaccia utente è pulita e ordinata, ma al contempo ricca e concettualmente piuttosto complessa. Ritengo utile presentarla brevemente, perché evidenzia già alcuni caratteri distintivi di Scratch in relazione ai processi di apprendimento. Inoltre riuscire a muoversi con sicurezza in un’interfaccia di questo tipo, per un alunno di scuola primaria, è di per sé un obiettivo di una certa complessità, e implica la capacità di padroneggiare strutture e livelli di astrazione

differenti. L'organizzazione spaziale delle diverse aree, la varietà dei comandi, la struttura logica e funzionale dei menu e delle *tab*, richiedono competenze non banali, in parte "trasferibili" da e verso l'uso di altre applicazioni, in parte specifiche di Scratch.

La schermata (figura 1) è suddivisa in quattro aree principali.

- Sulla sinistra, la maggior parte dello spazio è occupato dallo *stage*, il "palcoscenico" su cui è possibile eseguire in qualunque momento il programma cui si sta lavorando, o anche un singolo *script* (come si vedrà poco oltre). L'estrema facilità con cui si può verificare "cosa succede se faccio così…" è evidentemente un punto di forza nell'ottica di un apprendimento costruttivista.
- Sotto lo *stage* c'è un'area in cui vengono elencati tutti gli *sprite* (e gli sfondi) utilizzati nel progetto. Gli *sprite* sono i "protagonisti" dei programmi realizzati con Scratch; si tratta di elementi grafici bidimensionali, che possono muoversi sullo *stage*. Per i bambini vale spesso l'associazione *sprite* = personaggi, anche se in realtà uno *sprite* è qualunque elemento grafico (anche un pulsante, una freccia, una scritta) che viene programmato tramite uno o più *script*. Infatti in Scratch la programmazione avviene per singoli *sprite*: cliccandone uno, nell'area degli *script* vengono caricati tutti quelli relativi ad esso. Questa organizzazione logica, che è una delle prime cose che i nuovi utenti devono imparare, sottende un importante concetto informatico: la *programmazione ad oggetti*, in cui a ogni oggetto (in questo caso ogni *sprite*) è associato solo il codice relativo alle azioni che esso può compiere.
- La colonna al centro è la *tavolozza dei blocchi*, la "cassetta degli attrezzi" di Scratch, in cui si trovano tutti i comandi utilizzabili (circa 120), suddivisi per categorie contraddistinte da colori diversi. Oltre all'editor degli *script*, sono presenti anche un editor grafico, che permette di lavorare sull'aspetto degli *sprite* (uno sprite può avere più "costumi") e degli sfondi, e un editor di suoni; il passaggio dall'uno all'altro è comandato dalle "linguette" (*tab*) in alto.

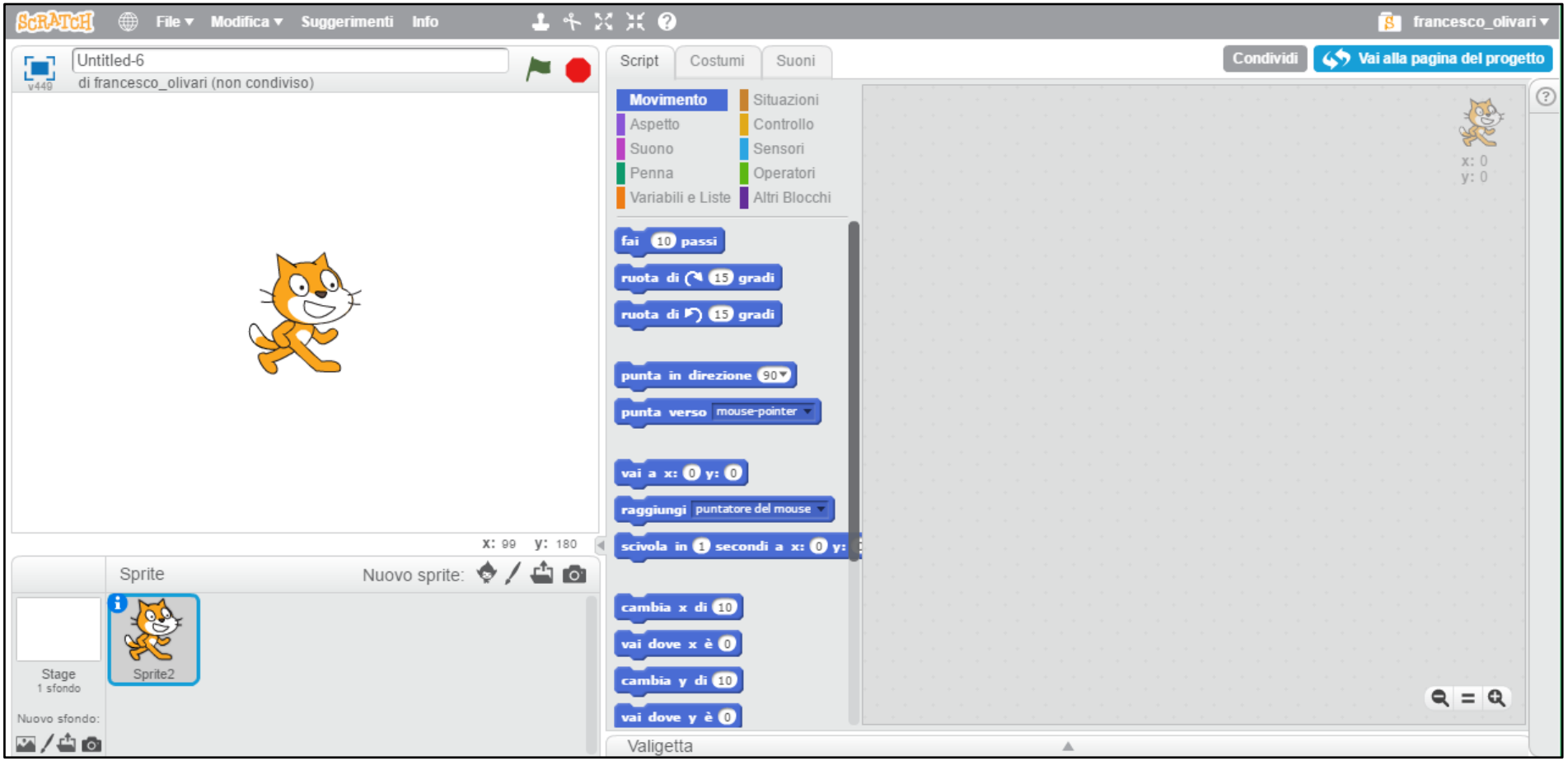


Figura 1. L'interfaccia utente di Scratch.

- A destra, infine, si trova l'ampia zona di lavoro, l'*area degli script*. Per costruire uno *script* è sufficiente trascinare qui dalla tavolozza i blocchi desiderati e combinarli tra loro. Per eliminare un blocco basta fare l'operazione inversa: trascinarlo dall'area degli *script* alla cassetta degli attrezzi. Come detto, ogni *sprite* utilizzato nel progetto ha la propria area degli *script*, che viene caricata quando si seleziona quel determinato *sprite*. Questo spazio è stato pensato e progettato come una sorta di "banco di lavoro", sul quale l'utente può lasciare blocchi non utilizzati, frammenti di *script* che potrebbero essere utili in un secondo momento, ecc. Quando si esegue un programma, infatti, gli *script* che non vengono "attivati" (perché magari l'utente ha volutamente omesso la condizione di attivazione) non creano nessun problema; semplicemente non vengono eseguiti. In questo modo Scratch è in grado di "assecondare" stili di programmazione differenti, sia quelli più orientati alla pianificazione ordinata, sia quelli più vicini al *tinkering*, alla manipolazione anche apparentemente confusa e disorganizzata degli strumenti (Resnick et al. 2009, 63). A tal proposito va sottolineata una funzionalità importantissima di Scratch: oltre a far girare l'intero programma, è possibile eseguire in qualsiasi momento un singolo *script*, semplicemente cliccandoci sopra; l'effetto viene visualizzato sullo *stage*. È quindi possibile scrivere un pezzo di programma, controllare "che cosa fa", modificarlo e ricontrollare iterativamente, in un processo che stimola e favorisce un apprendimento decisamente costruttivista.

4.1.1 Che cos'è l'*errore* in programmazione?

Realizzare un programma informatico implica una tipologia di "compito", un rapporto con l'errore e con la valutazione che presentano alcune peculiarità. Questo perché il computer, o meglio l'ambiente di programmazione che si sta utilizzando, è un *artefatto cognitivo* particolare, che fornisce il *risultato* del proprio lavoro (ossia l'esecuzione del programma) ogni volta che lo si desidera. Perciò l'*errore* non è legato a una "sanzione" emessa dall'insegnante (positiva o negativa, «giusto»/«sbagliato», con tutti i possibili correlati sul piano emotivo e relazionale: lode, incoraggiamento, delusione…), per di più facilmente confusa e

sovrapposta alla valutazione (il voto). In termini molto discorsivi, nella programmazione il bambino parte da un obiettivo[4] (che può essere assegnato dall'insegnante, o scelto individualmente, o deciso in piccolo gruppo): realizzare un programma che "faccia determinate cose". Quindi applica le proprie conoscenze e costruisce gli *script* seguendo il proprio modello mentale del programma. In qualsiasi momento può verificare se l'insieme di istruzioni su cui sta lavorando funziona come lui si aspetta: può far girare il programma e vederne gli effetti. Se tutto va secondo i piani, viene confermata la correttezza del modello mentale e si può procedere con un altro "pezzo" di programma. In caso contrario, il programma non può essere eseguito, oppure fa qualcosa di non previsto, o non fa qualcosa che ci si aspettava; questa dissonanza cognitiva costringe l'utente a riesaminare il proprio modello mentale alla luce dell'esperienza, cercando di capire *dove*, *come* e *perché* il programma "reale" non funzioni, o funzioni diversamente. Questo tipo di "errore" non è dunque legato a una sanzione esterna, ma pone una sfida di *problem-solving*.

In informatica gli errori di programmazione (chiamati *bug*[5]) si distinguono in tre categorie (Downey, Elkner, & Meyers 2002).

- Errori di *sintassi*, relativi alle regole di composizione del linguaggio utilizzato; l'esecuzione di un programma contenente errori di sintassi non può avvenire, e viene generato un messaggio di errore.
- Errori *in esecuzione* (chiamati anche *eccezioni*), visibili solo quando il programma viene eseguito e dovuti a particolari eventi "non gestibili" (per esempio dividere un numero per 0); si tratta di errori poco rilevanti per il livello considerato in questa tesi.
- Errori di *semantica*: il programma viene eseguito, ma "fa qualcosa di diverso" rispetto a quello che il programmatore si aspettava. Sono i *bug*

[4] In questo caso chiaramente non si sta pensando a un'attività di *tinkering*, ma a un progetto con un obiettivo definito in partenza. Tuttavia il discorso è applicabile anche a processi di "bricolage": inizialmente un bambino può combinare due blocchi solo per vedere "cosa succede", ma poi comincerà a fare aggiunte e modifiche in base a un modello di funzionamento che si sta costruendo.

[5] Anche se l'origine dell'uso del termine *bug* ("piccolo insetto") per indicare un errore di programmazione è controversa, c'è un celebre aneddoto che può risultare divertente e accattivante per i bambini: nel 1947 Grace Hopper, una pioniera dell'informatica, era alle prese con un malfunzionamento di un computer che sembrava inspiegabile; alla fine scoprì che la causa era dovuta a una falena (un *bug*, appunto) che si era incastrata fra i circuiti, interferendo con l'esecuzione del programma.

più “interessanti” e produttivi dal punto di vista dell’apprendimento, perché richiedono un’attività di *problem-solving* applicata ai concetti e alle procedure computazionali.

4.1.2 La “sintassi” di Scratch

Ogni linguaggio di programmazione testuale ha una propria sintassi, cioè un insieme di regole da rispettare per la scrittura di istruzioni corrette dal punto di vista formale, condizione necessaria perché il programma possa essere eseguito. È evidente che anche il linguaggio testuale con la sintassi più semplice porrebbe problemi molto rilevanti per bambini di scuola primaria; per di più – oltre al fatto che la difficoltà del compito può risultare demotivante –, individuare e correggere errori di sintassi non è particolarmente interessante per l’apprendimento del pensiero computazionale.

I linguaggi di programmazione visuale (oltre a Scratch ne esistono molti altri) hanno proprio lo scopo di aggirare questo problema: l’utente realizza il programma combinando tra loro i blocchi di istruzioni, che si situano a un livello superiore rispetto al linguaggio vero e proprio; è poi il software a “tradurli” in istruzioni sintatticamente corrette nel linguaggio sottostante.

Tuttavia non è possibile, e nemmeno proficuo, eliminare completamente la dimensione sintattica. Infatti in Scratch essa è “incorporata” nella *forma dei blocchi* e degli *agganci* che presentano (o non presentano). Non tutti i blocchi possono unirsi sequenzialmente gli uni agli altri: le modalità con cui possono combinarsi sono rese visibili dalla loro forma. Naturalmente ciò dipende dalla funzione che ciascun comando svolge. Se ne possono distinguere quattro tipi principali.

- I *blocchi di attivazione* hanno un incastro in basso, mentre in alto presentano una curvatura che esplicita come non sia possibile agganciarvi niente. Perciò si possono posizionare solo all’inizio di uno *script* e ne controllano l’attivazione: lo *script* viene eseguito quando si verifica la condizione espressa dal blocco di attivazione (fig. 2).

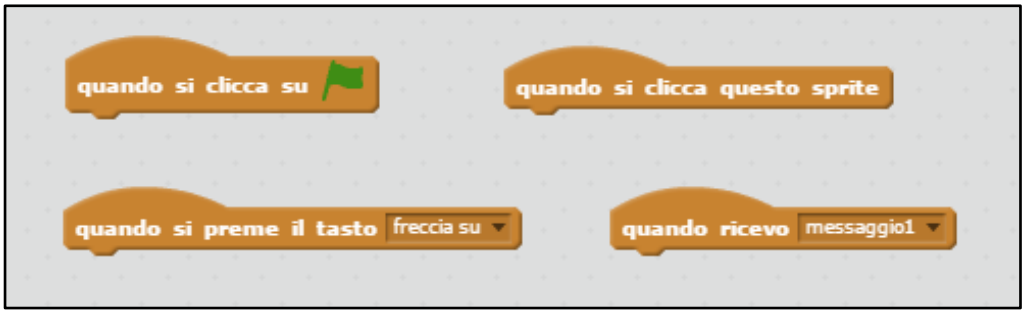


Figura 2. Blocchi di attivazione

- I *blocchi di comando* hanno forma rettangolare e presentano un incastro sia sopra sia sotto; possono dunque essere combinati in sequenza. Sono molto numerosi e appartengono a diversi "gruppi" di istruzioni (aspetto, movimento, suono, penna, ecc.; fig. 3).

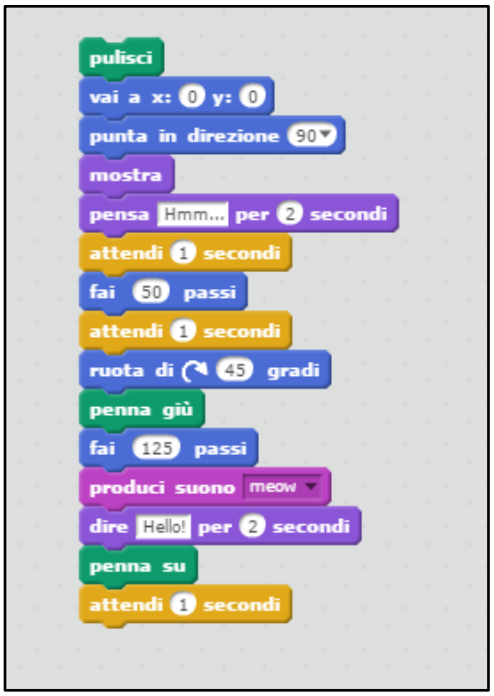


Figura 3. Blocchi di comando

- Alcuni *blocchi di controllo* hanno la forma di una "C" o di una graffetta, per evidenziare il fatto che devono "contenere" altri blocchi, la cui esecuzione viene appunto controllata in determinati modi (fig. 4). Appartengono a questo gruppo i comandi per l'esecuzione dei cicli e delle istruzioni condizionali.

Figura 4. Blocchi di controllo

- I *blocchi funzione*, infine, non possono essere concatenati in sequenza, ma vanno inseriti all'interno di quei blocchi che lo prevedono. Servono a

restituire un valore e sono di due tipi: quelli con i lati corti arrotondati rappresentano un valore numerico o testuale, quelli con i lati corti appuntiti rappresentano un valore booleano (fig. 5). Tra i blocchi funzione del gruppo "operatori", per esempio, ci sono anche quelli che servono per eseguire operazioni aritmetiche.

Figura 5. Blocchi funzione

Questa resa grafica della dimensione sintattica è molto interessante, perché da un lato elimina le difficoltà eccessive (e poco utili a questo livello) legate alla programmazione testuale, dall'altro però evidenzia come blocchi diversi svolgano azioni e funzioni concettualmente diverse, e in base a queste presentino precise limitazioni alle possibilità combinatorie. Utilizzando Scratch, il bambino è portato progressivamente a farsi domande ed elaborare modelli sul perché un certo blocco si può o non si può combinare con un altro; il tutto in una modalità decisamente *tinkerable*, che consente di manipolare i blocchi e provare gli incastri anche *prima* di avere costruito tali modelli. In altri termini, in Scratch non è possibile compiere errori di sintassi, perché il software li previene: semplicemente, i blocchi non si incastrano; questo però non esclude – anzi stimola – il fatto che l'utente cerchi di comprendere il motivo per cui una certa combinazione di blocchi è impossibile, e per questa via arrivi ad apprendere in modo autonomo le regole (o almeno alcune) della sintassi.

4.1.3 Errori di semantica e *debug*

Nella didattica dell'informatica la ricerca e la correzione dei *bug* semantici sono attività fondamentali, estremamente proficue per l'apprendimento della programmazione (Downey, Elkner, & Meyers 2002; Lodi 2014). Se ne parlerà

diffusamente nel capitolo 5; in questa sede, è opportuno rimarcare il fatto che Scratch *non visualizza messaggi di errore* (ciò vale anche per le eccezioni o errori a tempo di esecuzione). Se c'è un *bug*, semplicemente il programma non fa niente, o fa qualcosa di inatteso; così il bambino è stimolato e motivato a individuare l'errore, comprenderlo e correggerlo, e in più si evita il possibile effetto disconfermante di un messaggio esplicito.

Infine, riguardo l'interfaccia utente di Scratch, va evidenziata un'altra funzionalità utile come ausilio per il *debug* da parte degli alunni (oltre alla fondamentale possibilità di eseguire "per prova" singoli *script*, descritta in precedenza, e anche i singoli blocchi direttamente nella cassetta degli attrezzi, facendo doppio click). Quando viene fatto girare un programma, nell'area degli *script* vengono evidenziati in tempo reale quelli in esecuzione, tramite un bordo giallo che li contorna. Se uno *script* svolge un'azione "immediata" (per esempio `quando si clicca sulla bandiera verde` → `ruota di 90 gradi`), l'evidenziazione dura una frazione di secondo ed è quindi impercettibile[6]; viceversa, se è in esecuzione un ciclo, lo *script* rimane evidenziato fino a quando si conclude (nel caso di `per sempre`, fino a quando si arresta l'esecuzione del programma).

4.2 La *community*

Oltre che un ambiente di programmazione, Scratch è anche una *community* molto vivace e in continua espansione, con oltre 13 milioni di utenti registrati in tutto il mondo, che interagiscono in varie modalità rese disponibili dal sito web e hanno condiviso più di 16 milioni di progetti. Questa dimensione sociocostruttivista è un punto centrale dell'intero progetto, che fin dall'inizio ha l'obiettivo di stimolare la formazione e la crescita di una *comunità di apprendimento* (Dasgupta 2016).

Gli utenti registrati possono *pubblicare* i propri elaborati, che diventano così liberamente fruibili da parte di tutti. È possibile organizzarli in "gallerie", esplorarli utilizzando alcuni strumenti di navigazione, esprimere apprezzamenti

[6] Già questo effetto è piuttosto interessante per i bambini, che inizialmente non hanno un modello mentale efficace del tempo di esecuzione del programma e spesso si chiedono perché lo *script* «non si è illuminato».

(nella forma di cuoricini), commentarli, aggiungerli ai propri preferiti, "seguire" determinati utenti; è presente inoltre un forum di discussione, articolato in diverse sezioni.

L'elemento decisivo, però, è costituito da due funzionalità particolari, che rivestono un interesse cruciale dal punto di vista didattico, oltre a rimandare a una più generale visione "ideale" che è opportuno mettere in evidenza.

4.2.1 *Guarda dentro* e *Remix*

Quando si apre un progetto pubblicato da un altro utente, oltre a eseguirlo (e dunque vederlo "in funzione"), è possibile anche cliccare sul pulsante *Guarda dentro*; in questo modo si accede alla schermata di programmazione, in cui è presente il "codice". Si può quindi studiare come è stato realizzato il progetto, esaminare i singoli *script*, cercare di capire direttamente *come funziona*.

Non solo: in questa schermata, il comando *Remix* crea una copia editabile del progetto, che si può modificare a piacimento, sia per la curiosità di manipolare questo o quello *script* e vederne gli effetti, sia per arrivare a creare vere e proprie versioni alternative. Tali versioni sono a loro volta pubblicabili, e riportano il riferimento al progetto originale. In tal modo è possibile eventualmente stimolare una discussione sulle modifiche apportate, suggerirne altre, confrontarsi con gli altri utenti, ecc.

Come detto, fra i tanti punti forti di Scratch, questa possibilità di accedere al codice di qualsiasi progetto pubblicato è tra i più rilevanti per la didattica e l'apprendimento [7] (Dasgupta, Hale, Monroy-Hernández, & Hill 2016). Nel capitolo 5 ci si soffermerà sull'approccio *usa-modifica-crea*, che fornisce un quadro metodologico particolarmente adatto per sfruttare tali potenzialità.

4.2.2 Il software *open source*

Oltre alle finalità didattiche, con le funzionalità *Guarda dentro* e *Remix* gli sviluppatori di Scratch hanno operato una precisa scelta di campo nella contrapposizione tra software proprietario e software a sorgente aperto (*open*

[7] Di fatto riprende una modalità da sempre utilizzata (con molte più difficoltà) per imparare a programmare: trovare codice già pronto, eseguirlo, studiarlo, modificarlo…

source), che riguarda da vicino anche il mondo della scuola, in particolare le scelte degli istituti per quanto riguarda le dotazioni informatiche.

La questione è piuttosto articolata e, oltre alla sfera strettamente tecnologica e a quella economico-commerciale, ha importanti risvolti anche di natura etica e sociale in senso lato (Meo 2002; Stallman 2016). In generale, il "movimento" *open source* si oppone ai software proprietari, cioè quelli distribuiti dai produttori (di solito a pagamento, ma non necessariamente) con una licenza che ne permette soltanto l'utilizzo, impedendo l'accesso al codice sorgente, la copia e la ridistribuzione. La filosofia alla base dell'*open source* richiede che il codice sia sempre accessibile all'utente, analizzabile ed eventualmente modificabile, sulla base di specifiche licenze che lo consentano. Ideologicamente più "radicale", anche se ai fini pratici molto simile, è la posizione dei fautori del *software libero*, un movimento nato precedentemente (negli anni '80), il cui più noto esponente è Richard Stallman. In un certo senso la prospettiva dell'*open source* ha proposto un approccio più "pragmatico" alla questione, ponendo in risalto il fatto che il libero accesso ai codici sorgenti è *anche* una modalità efficiente e produttiva per lo sviluppo del software, che può conciliarsi con le esigenze delle aziende del settore. Oggi esistono numerosi applicativi *open source*, comprese suite da ufficio che costituiscono un'opzione molto interessante anche per le scuole.

Al di là dell'aspetto tecnico, ciò che più importa è la "visione" ideale promossa dal movimento *open source*. Nel caso specifico di Scratch, è un valore aggiunto far "immergere" i bambini in un ambiente caratterizzato da condivisione delle conoscenze, collaborazione e libera circolazione dei contenuti, anche perché lo sviluppo di competenze legate alla cooperazione e alla condivisione può rivelarsi strategico per il mondo del lavoro del prossimo futuro.

4.3 Che cosa si può fare con Scratch?

Tra i punti di forza di Scratch c'è anche l'estrema duttilità per quanto riguarda i prodotti realizzabili: storie, animazioni, giochi, quiz, effetti grafici. Non è possibile proporre una tassonomia rigida ed esaustiva dei "generi", e probabilmente essa non risulta davvero utile o rilevante. Se la finalità ultima è l'apprendimento del pensiero computazionale, ciò che conta è avere chiari i *concetti* e le *pratiche* e le *prospettive* computazionali che si vuole di volta in volta

stimolare o consolidare, in modo abbastanza indipendente dal tipo di progetto proposto agli studenti. Certo è possibile che alcuni concetti si presentino più facilmente e con maggiore frequenza in una certa classe di progetti (per esempio i *cicli* nell'arte interattiva, o la *parallelizzazione* nelle storie con più personaggi); ma si tratta comunque di connessioni fluide e non obbligate.

Dato che un abbozzo di tipologia, a maglie larghe e dai confini sfumati, può comunque essere utile come orientamento generale (soprattutto per gli studenti), si può utilizzare quella proposta dallo stesso sito di Scratch[8]: *animazione*, *arte*, *giochi*, *musica*, *storie*. Anche per individuare meglio gusti e preferenze individuali, e poter così sfruttare i vantaggi della motivazione intrinseca all'apprendimento e le potenzialità creative di ciascuno.

Un fattore trasversale, che può essere utile tenere presente, riguarda il grado di *interattività*, ossia il livello di interazione richiesto all'utente quando il progetto viene fatto girare, in un *continuum* che può andare dall'assenza totale (una presentazione o un'animazione cui assistere semplicemente) alla necessità di un'interazione continua (un gioco). Alcune tipologie si collocano in una zona precisa del *continuum*, mentre per altre la situazione può variare in base al singolo progetto.

[8] https://scratch.mit.edu/explore/projects/all

5. IMPARARE IL PENSIERO COMPUTAZIONALE CON SCRATCH: UNA PROPOSTA

Questo capitolo conclusivo si prefigge di delineare una proposta operativa sull'utilizzo didattico di Scratch in classi quarte e quinte di scuola primaria, alla luce di quanto esaminato in precedenza: il campo teorico ed epistemologico del *pensiero computazionale* e le relative esperienze in atto (capp. 1 e 2); il paradigma costruttivista/costruzionista sui processi di apprendimento e le metodologie didattiche che vi si rifanno (cap. 3); le caratteristiche peculiari di Scratch, sia come ambiente di programmazione, sia come comunità di apprendimento e condivisione (cap. 4).

Anzitutto è utile premettere che cosa *non è* questa proposta:

- un curricolo completo, che sarebbe un obiettivo troppo ambizioso;
- una serie di unità didattiche strutturate, che non corrisponderebbe alla prospettiva epistemologica e didattica adottata;
- una guida più o meno completa delle funzionalità di Scratch, molte delle quali non saranno prese in considerazione; il *focus* rimane sempre sull'apprendimento del pensiero computazionale, per il quale Scratch è uno strumento dalle grandi potenzialità, ma non deve trasformarsi nel fine ultimo.

Ciò che questo capitolo vuole presentare è dunque una sorta di panoramica non di progetti specifici e dettagliati, bensì di *modalità didattiche* o *tipologie di attività* da proporre, in una prospettiva fortemente orientata verso: apprendimento significativo, per scoperta e per *problem-solving*; creatività; dimensione intersoggettiva e collaborativa.

Qualche precisazione riguardante alcune scelte di fondo e il carattere necessariamente "aperto" e poco prevedibile di una didattica di questo tipo.

- In linea con quanto sopra esposto, quasi nessuna delle attività prevede esplicitamente un *tutorial* o un progetto da far costruire passo a passo, seguendo le indicazioni dell'insegnante. Tuttavia non si vuole escludere la possibilità (e talvolta l'opportunità) di attività di questo tipo.

L'avvertenza è che, nell'effettiva pratica didattica, il docente ha il compito di monitorare costantemente i processi in corso, e decidere eventualmente di inserire un *tutorial* video, una dimostrazione o una spiegazione relativa a un particolare aspetto, qualora lo ritenga opportuno.

- Anche nelle attività prevalentemente orientate all'apprendimento di uno specifico concetto computazionale, in cui è il docente a definire il tipo di progetto da elaborare (paragrafo 5.3), viene lasciata agli alunni ampia scelta in merito a personaggi, movimento degli *sprite*, dialoghi, ecc., per conciliare gli obiettivi di apprendimento con la creatività e l'espressione di sé.
- Nella didattica per progetti e per scoperta non è possibile stabilire a priori le traiettorie che verranno seguite dai singoli alunni e gli obiettivi specifici che saranno raggiunti. Un elemento centrale di questa proposta è il ruolo fondamentale svolto dal monitoraggio e dalla valutazione *in itinere* da parte dell'insegnante, come si vedrà meglio nel prossimo paragrafo.

Oltre alle fonti bibliografiche cui si è fatto riferimento fin qui e alla personale esperienza con Scratch, per la stesura di questo capitolo è stata molto preziosa la guida *Creative Computing*, sviluppata dal gruppo di ricerca ScratchEd della Harvard Graduate School of Education e distribuita gratuitamente con licenza Creative Commons (Brennan, Balch & Chung 2014), su cui si basa anche un corso *on line* liberamente fruibile[1]. La guida offre una ricca collezione di attività, risorse e spunti e riflessioni per l'insegnamento della programmazione con Scratch, fortemente orientata allo sviluppo del pensiero computazionale e della creatività.

Un'altra fonte consultata è *Coding. Progammare è un gioco*, il primo libro di testo italiano sul *coding* per la scuola primaria, edito da De Agostini Scuola (Ferraresso, Ferraresso, Colombini & Bonanome 2015). Anch'esso è incentrato sull'uso di Scratch, ma con un'impostazione completamente diversa: viene proposta una serie di progetti da realizzare passo a passo seguendo le istruzioni, completate da spiegazioni e approfondimenti.

[1] http://scratched.gse.harvard.edu/guide/

5.1 Apprendere il pensiero computazionale: un quadro di riferimento

Facendo seguito alle definizioni di pensiero computazionale discusse nel Capitolo 1, si presenta ora un *framework* elaborato dal gruppo ScratchEd della Harvard GSE, su cui si basa la guida *Creative Computing*. È uno strumento particolarmente interessante, che articola il campo in tre livelli o dimensioni: i *concetti* computazionali; le *pratiche* messe in atto dagli utenti; le *prospettive* promosse (in rapporto all'espressione di sé, alla conoscenza, alle relazioni interpersonali) (Brennan, Balch & Chung 2014; traduzione mia).

Concetti computazionali

Concetto	*Descrizione*
Sequenze	Identificare una serie di passaggi successivi per portare a termine un compito
Cicli	Eseguire più volte la stessa sequenza
Parallelismo	Gestire eventi contemporanei
Eventi	Qualcosa che si verifica al verificarsi di qualcos'altro
Condizionali	Prendere decisioni basate su determinate condizioni
Operatori	Supportare espressioni logiche e matematiche
Dati	Immagazzinare, richiamare e aggiornare valori

Pratiche computazionali

Pratica	*Descrizione*
Sperimentare e iterare	Sviluppare un pezzo di codice, provare, svilupparne un altro pezzo, riprovare, ecc.
Fare test e debug	Controllare che tutto funzioni; individuare e risolvere i problemi
Riusare e remixare	Creare qualcosa rielaborando idee o progetti esistenti
Astrarre e modularizzare	Esplorare le connessioni fra il tutto e le sue parti

Prospettive computazionali

Prospettiva	*Descrizione*
Esprimere se stessi	Comprendere che la computazione è un mezzo per creare: "Io posso creare"

Connettersi	Riconoscere le potenzialità del creare con e per gli altri: "Posso fare molte cose [in più] se sono in comunicazione con altre persone"
Porsi domande	Sentirsi in grado di fare domande sul mondo: "Posso (usare la computazione per) fare domande per dare senso al (agli oggetti computazionali nel) mondo"

Questa articolazione degli aspetti che caratterizzano ciò che chiamiamo sinteticamente *pensiero computazionale* sarà il punto di riferimento del presente capitolo.

È evidente che i tre livelli pongono questioni diverse in sede di progettazione didattica.

- Il grado di acquisizione di un determinato *concetto* computazionale nello svolgimento di un progetto libero può essere valutato dal docente tramite l'osservazione dei processi e l'analisi dei prodotti (l'alunno ha fatto uso – per esempio – di condizionali? Ha sperimentato diverse soluzioni? È in grado di trasferire quanto appreso in un nuovo contesto? Ecc.). Tuttavia è anche possibile proporre attività che puntino prevalentemente alla scoperta, all'esplorazione o al consolidamento di un determinato concetto (si veda 5.3).
- Per quanto riguarda le *pratiche*, l'approccio qui adottato si basa sostanzialmente su *sperimentazione e iterazione*, *test e debug*; mentre *riuso e remix* sono oggetto di uno specifico paragrafo. Meno semplice è monitorare i processi di *astrazione e modularizzazione*, rilevabili solo nel corso dell'attività o nella successiva rielaborazione da parte dell'alunno. Il punto essenziale, in ogni caso, è che le pratiche computazionali riguardano i *processi*, non i risultati; perciò il modo e il grado in cui esse vengono sperimentate e acquisite sono rilevabili solo tramite un costante monitoraggio da parte del docente.
- La questione è forse ancora più evidente per le *prospettive*, ossia i punti di vista e le "disposizioni" – su se stesso e sul mondo – promossi nell'alunno.

In definitiva, le tipologie di obiettivi e l'approccio orientato alla scoperta e alla creatività implicano un ruolo centrale della *valutazione in itinere*. L'attività di

stimolo, monitoraggio e *scaffolding* da parte del docente deve cioè intrecciarsi in ogni momento con la valutazione dei processi in atto, condotta con tecniche e strumenti specifici (si rimanda al paragrafo 5.8).

5.2 Per iniziare: primi passi e strumenti di lavoro

Anche se l'impostazione generale di questa proposta è decisamente orientata verso la scoperta, il *problem solving* e la creatività, il momento iniziale richiede una mediazione da parte del docente. È importante che la prima sessione con Scratch raggiunga due obiettivi: ispirare, suscitare curiosità e motivazione; fornire le competenze minime perché gli alunni possano iniziare a muoversi autonomamente senza scontrarsi con difficoltà eccessive e poco produttive.

Per conseguire il primo obiettivo si possono presentare video motivazionali (reperibili in rete) e alcuni progetti con Scratch particolarmente efficaci, piacevoli o stimolanti, selezionati dall'insegnante.

Per quanto riguarda il secondo, è opportuno guidare gli alunni nei primi passi propedeutici per l'utilizzo di Scratch: collegarsi al sito web, creare un account, orientarsi nell'interfaccia (ma senza esagerare: bastano poche indicazioni per mettere i bambini in condizione di esplorarla). Si può quindi valutare l'opportunità di creare insieme un primo progetto, tramite un *tutorial* o una spiegazione passo a passo. In generale, il vantaggio di questa modalità consiste nel focalizzare alcuni obiettivi di base in modo più efficiente rispetto alla scoperta individuale; tuttavia la motivazione e l'attenzione degli studenti non sono garantite (anche se la "novità" dello strumento dovrebbe giocare a favore). In ogni caso, è importante lasciare campo libero all'attività di sperimentazione autonoma non appena la classe si dimostra pronta e motivata.

Infine, è opportuno che ogni bambino si doti di un quaderno di appunti come materiale di supporto. In molti corsi per fasce di età più avanzate viene utilizzato un diario di bordo, da compilare regolarmente, in cui sviluppare riflessioni stimolate da domande-guida poste dai docenti, che favoriscono una rielaborazione metacognitiva. Per la scuola primaria uno strumento di questo tipo rischierebbe di appesantire l'esperienza, togliendo piacere e motivazione. Il quaderno di appunti, invece, è molto utile per supportare la creatività: vi si possono annotare idee, schizzi, *script* interessanti, eventuali riflessioni. L'utilizzo

dovrebbe essere facoltativo, ma può essere opportuno "incoraggiarlo", almeno inizialmente. L'importante è che la compilazione avvenga a discrezione dello studente, in linea con il suo stile cognitivo (scrittura, disegni, schemi, ecc.).

5.3 Verso i concetti computazionali: esempi di progetti "orientati"

In questo paragrafo si prendono in esame i singoli concetti computazionali individuati nella definizione operativa, proponendo semplici esempi di attività e spunti di lavoro volti prevalentemente all'esplorazione e alla progressiva acquisizione di ciascun concetto. Perciò si è cercato di trovare un equilibrio tra la necessità di "indirizzare" i progetti verso obiettivi determinati e l'approccio didattico costruttivista, di esplorazione, scoperta e progressiva elaborazione degli apprendimenti da parte degli alunni.

Quelli proposti, naturalmente, sono solo alcuni spunti; il docente può elaborarne moltissimi altri, attingendo a risorse esistenti (come per esempio i due testi citati in apertura del capitolo) o creandoli da sé, anche in base all'osservazione della classe e a eventuali esigenze, interessi o difficoltà emergenti.

Una precisazione (per quanto scontata): in qualunque progetto sono in gioco diversi concetti computazionali; qui si fa riferimento di volta in volta a un concetto che, nel tipo di attività proposta, deve necessariamente essere "elaborato", ma ovviamente non in modo esclusivo.

Infine, si ribadisce che l'obiettivo degli *input* suggeriti non è che gli alunni si limitino a svolgere il "compitino": la speranza è che questi siano stimoli iniziali da cui partire per poi sperimentare, arricchire, sviluppare il progetto (oppure crearne più di uno), anche in direzioni inaspettate.

Sequenze

Il concetto di *sequenza* è probabilmente quello più facilmente accessibile, perché identifica una successione lineare di cui tutti abbiamo esperienza diretta in moltissimi campi del conoscere e dell'agire. La struttura sequenziale, per esempio, è presente nel linguaggio, sia scritto sia parlato; gran parte della

conoscenza è organizzata in questo modo: gli studi cognitivisti hanno identificato negli *script*[2] (copioni) una tipologia fondamentale di schemi mentali, in cui una certa situazione (fare la spesa, andare dal dentista, prendere l'aereo) è descritta in termini di azioni successive (paragrafo 3.2.1).

La stessa struttura spaziale degli *script* di Scratch (come in qualsiasi altro linguaggio di programmazione) è basata sulla successione lineare di blocchi concatenati dall'alto verso il basso. Lo scopo di un'attività basata su questo concetto può essere quello di creare sequenze di azioni utilizzando un numero crescente di blocchi; in questo modo, al consolidamento del concetto si affianca l'esplorazione di una varietà di comandi, in particolare quelli di forma rettangolare (*blocchi di comando*, 4.1.2), che fanno "compiere azioni" agli *sprite*.

Si propongono due esempi molto semplici, che come sempre lasciano ampio spazio alla creatività degli alunni.

> STIMOLO 1: *Scegliete un personaggio (sprite) e uno sfondo, quindi create un progetto in cui il personaggio "si presenta", parlando e muovendosi sullo stage*.

Questa consegna permette di esplorare soprattutto i blocchi dei gruppi "movimento" e "aspetto" (ed eventualmente "suono"). Inoltre gli alunni inizieranno a cimentarsi con elementi tutt'altro che banali, che creano sempre difficoltà ma sono molto importanti nell'ottica del pensiero computazionale, come la velocità di esecuzione delle istruzioni e la differenza tra codice di programmazione e linguaggio naturale. Per esempio, è probabile che un bambino alle prime armi con Scratch (e con l'informatica in generale) si aspetti che, utilizzando comandi come `fai 20 passi` oppure `vai a x:120 y:80`, lo sprite si sposti lentamente, magari "camminando". Invece si tratta di spostamenti istantanei: il nostro occhio non vede il movimento, bensì uno sprite che scompare e contemporaneamente riappare in una diversa posizione. È molto utile che i bambini comincino a "scontrarsi" con questa logica e a elaborare soluzioni per ottenere gli effetti desiderati, per esempio manipolando il *timing* con l'utilizzo del comando `attendi … secondi`.

[2] Non è casuale l'omonimia con gli *script* in informatica, che sono appunto sequenze ordinate di istruzioni.

STIMOLO 2: *Utilizzando i blocchi del gruppo "penna", create un progetto in cui i movimenti del personaggio lasciano una traccia del suo percorso*.

Lo scopo di questa attività è stimolare gli alunni a sperimentare vari percorsi, alla progressiva ricerca di effetti grafici interessanti tramite la manipolazione della sequenza dei movimenti (secondo la pratica computazionale *sperimentare e iterare*). Anche senza invitarli esplicitamente a farlo, è probabile che alcuni inizieranno a creare "tracciati" con precise forme geometriche. Oltre all'utilità di progetti di questo tipo per l'apprendimento o il consolidamento di concetti matematici e geometrici (si veda 5.9), sperimentare sequenze di comandi per il disegno geometrico può aprire la strada alla "scoperta" dei cicli.

Cicli

I cicli consentono di iterare una certa sequenza di comandi e sono importantissimi in informatica. Scratch ha tre blocchi di controllo con cui si possono costruire cicli: `per sempre`; `ripeti … volte`; `ripeti fino a quando …` (in quest'ultimo la fine del ciclo dipende dal verificarsi di una certa condizione). La forma dei blocchi ("a C") indica chiaramente che devono contenere i comandi che devono essere iterati.

I cicli possono essere utilizzati per rendere più economico il codice: se per esempio una stessa azione deve essere compiuta 8 volte, è più efficiente e più elegante racchiuderla in un ciclo `ripeti 8 volte` piuttosto che utilizzare una lunga sequenza. È abbastanza naturale introdurre questi concetti sfruttando l'attività precedente sul disegno geometrico.

STIMOLO 1: *Utilizzando i blocchi del gruppo "penna", fate muovere lo sprite in modo che disegni dei poligoni*.

Oltre a far ragionare sulla lunghezza dei lati e sull'ampiezza degli angoli, questa attività è orientata alla scoperta dell'uso dei cicli per la realizzazione di poligoni, in particolare quelli regolari (il codice della figura 1 disegna un quadrato).

STIMOLO 2: *Scegliete uno sprite che abbia almeno due diversi "costumi" [può essere utile guidare la scelta] e "animatelo": fatelo camminare se è una persona, sbattere le ali se è un pipistrello, ecc.*

Questa proposta parte da una caratteristica di Scratch che spesso suscita una certa delusione nei novizi, già accennata in precedenza: quando si dà a un personaggio il comando `fai 10 passi`, i bambini (riferendosi al linguaggio naturale) si aspettano che il personaggio muova le gambe e "cammini", mentre quello che succede è una traslazione istantanea di un'immagine bidimensionale fissa. Questa attività cerca di trasformare la delusione in una sfida stimolante: è possibile riprodurre un effetto-camminata lavorando con i costumi e i movimenti in maniera ciclica (e, anche qui, affinando il *timing*) (figura 2).

Figura 1

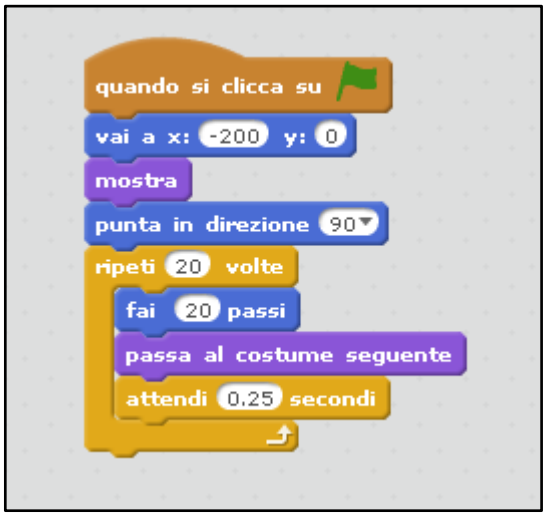


Figura 2

Infine è importante accennare alla grande utilità del ciclo `per sempre`, che spesso può risultare anti-intuitiva. Se per esempio si chiede a un principiante di scrivere un programma che faccia ruotare lo *sprite* in direzione del puntatore del mouse, è probabile che crei uno *script* composto di due soli comandi: `quando si clicca su bandiera verde` → `punta verso puntatore del mouse`. Se si esegue il programma, però, il risultato non è quello desiderato: lo *sprite* compie un solo movimento e poi resta fermo. Affinché continui a seguire il puntatore, occorre che il secondo comando sia inserito in un ciclo `per sempre`., altrimenti il

programma lo esegue una sola volta. Casi simili a questo sono molto frequenti, e i bambini apprenderanno presto la necessità di iterare all'infinito (o meglio: *per tutta la durata dell'esecuzione del programma*) certe azioni o comportamenti.

Parallelismo

Un programma informatico gestisce molti eventi contemporaneamente. In Scratch, che è basato sulla programmazione a oggetti, per ogni *sprite* vengono eseguiti i relativi *script*. Perciò è importante che gli alunni "entrino" progressivamente in questa logica e imparino a padroneggiare la simultaneità e la sincronizzazione degli eventi.

STIMOLO 1: *Create una storia con almeno tre personaggi che si muovono, parlano, interagiscono*.

È una consegna semplice, ma molto "aperta" e potenzialmente stimolante, perché lascia libero spazio alla creatività di ciascuno nell'ambito dello *storytelling*, cui Scratch si presta molto bene. Mentre i bambini saranno concentrati sulla storia da mettere in scena, sui personaggi e sui dialoghi, si "scontreranno" con il problema di sincronizzare tutto ciò che avviene sullo *stage*. Inizieranno così a "costruire" e a gestire autonomamente il concetto di parallelismo. Con ogni probabilità, la strategia più utilizzata sarà quella di controllare direttamente il *timing* di ciascun evento, calcolando i tempi che ciascun personaggio deve impiegare per compiere uno spostamento, parlare, attendere il proprio "turno" (applicando le pratiche *sperimentare e iterare*, *fare test e debug*). Se la storia è abbastanza articolata, può diventare complesso raggiungere un risultato soddisfacente con questo metodo; ciò può costituire una premessa interessante per introdurre in modo motivato il concetto computazionale relativo agli *eventi*.

Eventi

Si può presentare questa attività proprio partendo da una riflessione in grande gruppo su quella precedente e sulle difficoltà eventualmente incontrate. Sarebbe interessante se gli alunni arrivassero a esprimere l'esigenza di una

strategia più efficiente e meno complicata per sincronizzare gli eventi e i comportamenti degli *sprite*; il docente potrebbe stimolare la discussione cercando, con opportune domande o interventi di rispecchiamento, di arrivare a questo tipo di riflessioni.

A questo punto si possono presentare i blocchi `invia a tutti messaggio` e `quando ricevo messaggio` e accennare alla loro funzione (in questo caso è opportuna una minima introduzione da parte del docente, per poi lasciare che i bambini sperimentino).

STIMOLO 1: *Create una nuova versione della vostra storia, oppure una completamente nuova, utilizzando i "messaggi" tra gli sprite*.

Questa attività è probabilmente meno creativa e meno "costruttivista" delle altre, dato che l'insegnante fornisce un'indicazione su un tipo di blocchi da utilizzare. Tuttavia è probabile che gli alunni mostrino un'elevata motivazione verso la sperimentazione di comandi per la risoluzione di problemi con cui si sono scontrati in precedenza. Una dinamica di questo tipo ha buone possibilità di produrre apprendimento significativo.

STIMOLO 2: *Create un progetto interattivo in cui uno sprite fa cose diverse in base ai tasti premuti dall'utente.*

Questo è un altro semplice spunto per collegare i comportamenti dello *sprite* al verificarsi di determinati eventi. I bambini sono liberi di programmare i comportamenti che preferiscono (spostarsi, parlare, scomparire, cambiare colore, emettere suoni, ecc.) e di associarli ad alcuni tasti. Progetti di questo tipo introducono l'interazione tra il programma e l'utente, un aspetto ovviamente fondamentale, oltre che accattivante.

Condizionali

Spesso c'è la necessità che un programma si comporti in modi differenti a seconda che si verifichi o meno una data condizione. I comandi che tipicamente "esprimono" questo concetto computazionale sono quelli del tipo `se … allora …` e `se … allora … altrimenti …`

Tra i molti modi possibili per introdurre gli studenti alle istruzioni condizionali, si può optare per un gioco "di inseguimento" (nella figura 3 è riprodotto un esempio).

STIMOLO 1: *Costruite un gioco in cui uno sprite comandato da voi, o semplicemente il puntatore del mouse, viene inseguito da un altro sprite. Quando viene raggiunto, il gioco si conclude.*

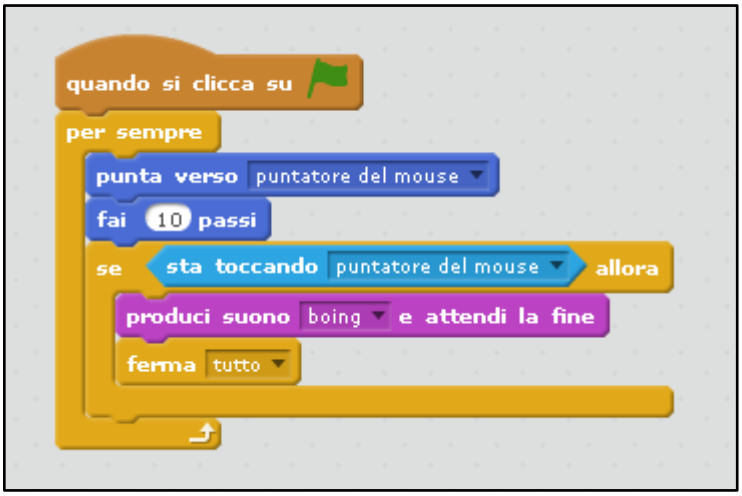


Figura 3

Operatori

Gli operatori sono funzioni che compiono elaborazioni sui valori in ingresso e restituiscono un risultato. Permettono di svolgere una varietà di operazioni e, a seconda del tipo, possono rappresentare valori numerici, stringhe o valori booleani (vero/falso). In Scratch sono raccolti nel gruppo di blocchi chiamato appunto "operatori"; non possono essere agganciati sopra o sotto altri blocchi, ma devono essere inseriti all'interno, come suggerisce anche la loro forma. Inoltre possono essere annidati, cioè si può "allungare" la catena di operazioni inserendo un altro blocco nello spazio dedicato all'argomento (esattamente come si fa in matematica, costruendo ad esempio una somma di più di due addendi).

A prima vista molti di questi blocchi potrebbero sembrare astrusi per i bambini; ma una volta comprese le "regole sintattiche" per utilizzarli, non è difficile trovare modalità per iniziare a "giocare" con essi, ed è anzi possibile che gli alunni si appassionino alla creazione di progetti basati su operazioni da

compiere su parole e numeri. Un'idea accattivante per cominciare può essere la seguente (un esempio di progetto è riprodotto nella figura 4).

STIMOLO 1: *Create un progetto in cui uno sprite dialoga con voi, ponendovi domande e riprendendo le vostre risposte.*

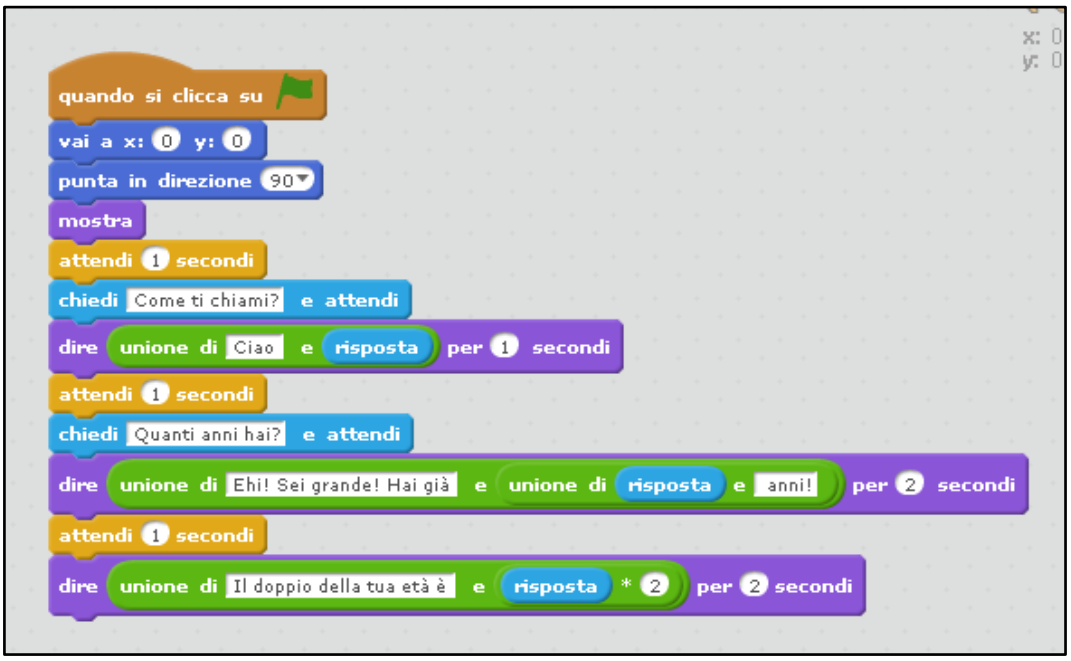


Figura 4

Dati

Un modo accessibile e significativo per avvicinare i bambini ai concetti di immagazzinamento, aggiornamento e restituzione di valori consiste nell'introduzione della variabile "punteggio" in un videogioco[3].

[3] A questo proposito è molto pregnante un aneddoto riportato da Mitchel Resnick nel TED Talk già citato nel Capitolo 1. Durante una visita in un club pomeridiano in cui si tenevano laboratori di Scratch, fu avvicinato da un ragazzino che chiese il suo aiuto per risolvere un problema: aveva realizzato un videogioco in cui bisognava "guidare" un pesce per fargli mangiare i pesciolini che nuotavano attorno a lui; però non sapeva come fare per tenere conto del punteggio, cioè del numero di pesciolini mangiati. Resnick gli mostrò come creare una variabile; il ragazzino comprese al volo e modificò il programma ottenendo il risultato desiderato, e alla fine lo ringraziò tantissimo. Resnick conclude il racconto con queste parole, suscitando una divertita ovazione nella platea: «Non so quanti insegnanti sono mai stati ringraziati per aver insegnato le variabili!». Un esempio illuminante di apprendimento significativo e motivato.

STIMOLO 1: *Prendete un gioco che avete creato in precedenza, oppure createne uno nuovo. Quindi inserite il punteggio: ogni volta che viene raggiunto l'obiettivo, il punteggio aumenta*.

È probabile che, come nell'aneddoto riportato in nota, inizialmente sia necessario guidare gli alunni alla scoperta delle variabili in Scratch: occorre selezionare la relativa *tab* nella tavolozza dei blocchi, crearne una e darle un nome. A questo punto compaiono alcuni blocchi con cui è possibile gestire la variabile all'interno dello *script*: da questo momento si può lasciare campo libero all'attività di scoperta, sperimentazione, test e *debug* da parte degli alunni.

5.4 Giocare con il *debug*

La ricerca, l'identificazione e la correzione dei *bug* (sintattici, semantici o in esecuzione) sono attività essenziali nella programmazione informatica, a tutti i livelli; di solito vengono svolte con l'ausilio di appositi software (*debugger*) disponibili negli ambienti di sviluppo stessi. Ma il *debug*, come accennato nel Capitolo 4, è importantissimo e molto proficuo anche a livello didattico, per l'apprendimento tanto della programmazione, quanto di concetti e procedure del pensiero computazionale (per esempio Papert 1980; Downey, Elkner, & Meyers 2002; Lodi 2014). Quando un programma non si comporta come previsto dal programmatore, si crea una dissonanza cognitiva tra il modello mentale di quest'ultimo e l'effettivo funzionamento del programma; l'analisi e la risoluzione del problema portano a una correzione del modello mentale, ossia a un *apprendimento* (significativo, perché consiste in una ristrutturazione di conoscenze già possedute, ma errate).

Come evidenziato nel capitolo precedente, in Scratch gli errori di sintassi non sono possibili (sono prevenuti dalle limitazioni combinatorie dei blocchi); il *debug* riguarda dunque gli errori semantici e le eccezioni.

Fare test e debug è una *pratica computazionale* continuamente presente nell'attività di programmazione: «Per qualcuno la programmazione e il *debug* sono la stessa cosa, intendendo con questo che la programmazione è un processo di rimozione di errori finché il programma fa ciò che ci si aspetta» (Downey, Elkner, & Meyers 2002, 5). Tuttavia non è detto che i bambini lo facciano

abbastanza, o che imparino subito a farlo in modo efficace. Nella pratica didattica (nel corso di qualunque attività di quelle qui suggerite), il docente dovrà monitorare costantemente se, quanto e come gli alunni si impegnano in tale pratica.

Oltre a questa dimensione "onnipresente" del *debug*, è possibile proporre una modalità specifica: presentare alcuni progetti, corredati da una descrizione scritta di ciò che "dovrebbero fare", ma affetti da *bug*. Si tratta di un gioco sfidante, una sorta di "caccia all'errore" organizzata dall'insegnante, coerente con il fatto che «in un certo senso il *debug* può essere paragonato al lavoro investigativo» (*ibidem*). Questa modalità didattica presenta due vantaggi: come detto, è una sfida di *problem solving* che di solito risulta accattivante per i bambini (motivazione elevata); inoltre può essere facilmente "indirizzata" verso uno specifico concetto computazionale che si desidera consolidare. Lo svantaggio, naturalmente, è che si tratta di un compito ben preciso e delimitato, che permette di raggiungere un determinato obiettivo di apprendimento ma tendenzialmente solo quello. Perciò è opportuno proporre i "giochi di *debug*" solo in certe occasioni e in piccole dosi.

Per quanto riguarda le risorse utilizzabili, in rete si trovano vari esempi già predisposti; molti sono linkati anche nella guida *Creative Computing* (Brennan, Balch & Chung 2014), con l'indicazione dei concetti computazionali maggiormente implicati in ciascun gruppo di esercizi[4]. Naturalmente il docente può anche creare da sé i propri giochi di *debug*, in base alle esigenze specifiche della classe.

5.5 *Remix*

Come illustrato nel Capitolo 4, la possibilità di accedere al codice di qualunque progetto condiviso, modificarlo e salvarne proprie versioni personalizzate è un aspetto cruciale di Scratch, con implicazioni molto rilevanti sui processi di apprendimento.

[4] Questi esercizi di *debug* sono in inglese, perciò l'insegnante deve preliminarmente tradurre le indicazioni per gli studenti.

Alcuni ricercatori hanno elaborato un approccio metodologico per l'insegnamento del pensiero computazionale basato proprio sul progressivo "coinvolgimento" diretto, secondo la successione *usa-modifica-crea* (Lee et al. 2011; Lodi 2014). Nella fase *usa* il discente è semplice fruitore di un "prodotto" realizzato da altri (un programma, un videogioco, ecc.); una volta familiarizzatosi con l'ambiente e gli strumenti, inizia a *modificare* il programma, partendo da dettagli più "superficiali", per poi salire a livelli di maggiore complessità; continuando a lavorare su progetti altrui, modificandoli e sperimentando, apprende via via concetti e procedure che lo porteranno a essere in grado di *creare* un proprio progetto.

L'efficacia di questo modello è stata confermata da una ricerca quantitativa condotta proprio sull'uso di Scratch (Dasgupta, Hale, Monroy-Hernández & Hill 2016). Analizzando una grande mole di dati ricavabili dalla *community* di Scratch, gli autori sono giunti a due conclusioni: gli utenti che "remixano" di più arrivano a padroneggiare un repertorio più ricco di comandi; l'"esposizione" a un determinato concetto computazionale tramite la pratica del *remix* aumenta le probabilità che quel concetto venga utilizzato in un successivo progetto.

Nella pratica didattica, si suggeriscono due modalità distinte per questo tipo di attività. Si può dedicare un'ora all'esplorazione libera dell'universo dei progetti condivisi in rete, invitando gli alunni a soffermarsi su quelli maggiormente interessanti, cercare di capire "come sono fatti" e provare a modificarli a piacimento. Questo approccio presenta enormi potenzialità: mette al centro l'interesse e la motivazione intrinseca degli alunni e favorisce la costruzione di apprendimenti significativi. Non bisogna però illudersi che dia sempre e automaticamente risultati soddisfacenti: è opportuno segnalare le possibili difficoltà cui il docente deve prestare attenzione.

Nel corso dell'esplorazione libera, è possibile che un bambino si appassioni a un progetto, per esempio un gioco o un'animazione interattiva, e continui semplicemente a "usarlo" senza "guardare dentro". In questo caso saremmo di fronte a un utilizzo passivo della tecnologia, analogo a quello che si fa normalmente con i videogiochi, quindi senza il passaggio alla dimensione della *fluency* teorizzata da Resnick (Capitolo 1). È necessario che il docente intervenga, non però in modo direttivo ("Non devi solo giocare, devi capire come è fatto!"), bensì invogliando a ideare possibili modifiche al progetto, facendo sempre leva

sulla motivazione: “Ti piace questo progetto? Quali sono gli aspetti che ti piacciono di più? C’è invece qualcosa che cambieresti? Cosa faresti per migliorarlo? Puoi provare!”). Un intervento analogo può essere necessario anche nel caso in cui un alunno prosegua nell’esplorazione senza soffermarsi su nessun progetto, attratto più dalla varietà e dal piacere della “carrellata” che dall’approfondimento.

Un altro rischio è che i progetti che attirano maggiormente l’attenzione abbiano un codice troppo complesso per il livello di competenze degli alunni. È un pericolo molto concreto, dato che in Scratch si trovano molti progetti davvero bellissimi, realizzati da adolescenti o da adulti, con un grado di complessità elevato. Perciò è opportuno che il docente conduca un’esplorazione preliminare e crei una *galleria* di progetti potenzialmente interessanti e al contempo “accessibili”. Deve essere una galleria ricca, con diverse decine (meglio centinaia) di elaborati di varie tipologie (storie, giochi, arte, animazioni, ecc.). Nella conduzione del laboratorio, si può “tenere pronta” questa risorsa nel caso in cui l’esplorazione libera si scontri con le difficoltà appena descritte.

Un’altra possibile applicazione dell’approccio *usa-modifica-crea* e della pratica del *remix* consiste nel limitare l’esplorazione ai progetti elaborati dalla classe. Si può proporre al termine di qualunque delle attività presentate in 5.3. In questo caso la motivazione intrinseca del discente può essere inferiore, ma il vantaggio è che tutti i progetti sono nati dallo stesso “stimolo” del docente, e costituiscono quindi modalità differenti di approcciare lo stesso problema. Ogni alunno può così avere una panoramica di elaborazioni possibili, facilmente ancorabili alla propria esperienza e dunque in grado di generare nuovi apprendimenti significativi.

5.6 Creatività al centro: il progetto libero

Nell’ipotesi ideale di portare avanti le attività di *coding* per un intero anno scolastico (o buona parte di esso), la realizzazione di un progetto personale liberamente scelto dagli alunni è un elemento cruciale, quasi imprescindibile. Una sorta di “saggio” complessivo, da condividere con la classe verso la fine dell’anno. È vero che, per il docente, ciò che importa sono i *processi* e gli *apprendimenti* più che i prodotti. Tuttavia la prospettiva costruzionista (si veda

3.4.2) e l'approccio dell'apprendimento creativo da essa derivato (3.4.3) hanno sottolineato l'impatto determinante, proprio sulla qualità degli apprendimenti, dell'impegnarsi in un progetto significativo (per il discente) e orientato a un risultato.

Per la scelta del "genere" (gioco, narrazione, animazione, ecc.), del soggetto e di qualunque altra caratteristica, va lasciata completa libertà: il *focus* è sull'uso creativo ed espressivo di concetti, strumenti e procedure. Rispetto alla griglia degli obiettivi per lo sviluppo del pensiero computazionale, chiaramente i *concetti* e le *pratiche* coinvolti sono variabili in base al singolo progetto e pertanto non prevedibili; a livello di *prospettive*, invece, questa attività è evidentemente connessa con l'*espressione di sé*.

Per quanto riguarda la scansione dei tempi, il progetto libero individuale andrebbe proposto dopo un certo numero di sessioni di lavoro, quando cioè l'insegnante ritiene che gli alunni abbiano familiarizzato a sufficienza con Scratch, e soprattutto si siano fatti un'idea della grande varietà di prodotti realizzabili. L'ideazione e lo sviluppo del progetto devono poter contare su tempi distesi: si potrebbe dedicarvi regolarmente un'ora ogni tre o quattro ore di attività con Scratch, magari intensificando nelle ultime settimane. Va inoltre sollecitato l'uso del quaderno degli appunti, per tenere traccia di idee, intuizioni, abbozzi di sceneggiatura, soluzioni tecniche, ecc.

Al termine del percorso – sempre valutando il grado di motivazione e interesse da parte degli alunni – si può organizzare una giornata-evento per la presentazione dei progetti al gruppo classe, alla quale eventualmente si possono invitare anche altre classi. In questo caso l'eccezionalità dell'occasione giustifica una simile scelta, che può portare a un ulteriore rafforzamento dell'autostima dei singoli, delle dinamiche interazionali e dell'interesse per le attività di *coding*.

5.7 La dimensione sociocostruttivista

Nell'approccio dell'apprendimento creativo, di matrice costruzionista, la cooperazione e la condivisione sono fattori cruciali nella costruzione della conoscenza, che è un prodotto di interazioni sociali che avvengono all'interno di una comunità di apprendimento. Tale prospettiva è coerente anche con il modello del *laboratorio* (Frabboni 2007), uno "spazio metodologico" in cui trovano posto

la progettazione didattica, la ricchezza degli scambi comunicativi e affettivi, la produzione collettiva del sapere. Nei paragrafi precedenti ci si è concentrati su varie tipologie di attività; tutte (o quasi) possono essere declinate con metodologie di lavoro collaborativo, secondo modalità differenti. Di seguito se ne propongono alcune.

Va premesso che l'attività di programmazione con Scratch, come qualsiasi altra, avviene entro un *setting* didattico che presenta caratteristiche specifiche, e dunque anche determinate limitazioni. Il fatto che sia necessario interagire con un computer – per quanto questa osservazione possa sembrare banale – costituisce un vincolo ben preciso, che rende difficili o poco efficienti certe situazioni didattiche. Ci si riferisce in particolare al *cooperative learning*, metodologia illustrata nel Capitolo 3, che prevede gruppi di 3-5 studenti, generalmente con una divisione esplicita dei ruoli e dei compiti, finalizzata a un obiettivo comune. Ritengo che questa tecnica non sia facilmente applicabile allo sviluppo di un progetto con Scratch, sia per motivi "logistici" (gestire l'uso di mouse e tastiera in un gruppo di quattro bambini può provocare facilmente frustrazione e litigi), sia perché l'approccio che qui si propone, basato su esplorazione, prove ed errori, non è semplice da conciliare con la strutturazione del lavoro necessaria per un corretto *cooperative learning*.

Vi sono però altri metodi e tecniche di lavoro collaborativo e di co-costruzione delle conoscenze che si possono proficuamente utilizzare.

5.7.1 Programmare a coppie

Tutte le attività fin qui proposte possono essere svolte individualmente oppure *a coppie*. Quest'ultima modalità può essere di tipo *collaborativo* oppure *cooperativo* (si veda 3.5.1): nel primo caso entrambi i partecipanti si occupano del progetto nel suo insieme, senza definire un'esplicita divisione di compiti; nel secondo, invece, ciascuno svolge un ruolo preciso.

Rispetto al progetto individuale, l'attività di coppia presenta due possibili vantaggi:

- richiede una maggiore capacità di *decentramento* cognitivo, in due direzioni: oltre a capire "come opera il programma", il bambino deve

anche capire “che cosa intende ottenere il compagno” con una determinata procedura;

- stimola una maggiore formalizzazione delle conoscenze in gioco, uno sforzo metacognitivo per riuscire a comunicare efficacemente con il compagno.

Sarà il docente a stabilire di volta in volta, in base all’osservazione del contesto-classe e agli obiettivi da perseguire (compresi quelli trasversali, di ordine relazionale), se sia più opportuno far realizzare un determinato progetto: individualmente; a coppie omogenee/eterogenee per competenze; a coppie omogenee/eterogenee per gusti e interessi.

La modalità “collaborativa”, come detto, è più “libera” perché non prevede una divisione formale del lavoro. Tuttavia può rivelarsi difficile, per i bambini, condurre l’attività in maniera efficace, gestire l’interazione in modo che entrambi si sentano ugualmente coinvolti nel progetto, evitare la confusione o la frustrazione. Perciò può essere interessante sperimentare una modalità di lavoro che viene utilizzata sia nell’industria del software, sia nell’ambito didattico oggetto di questa tesi, come testimonia un video del progetto Code.org che ne sintetizza i principi di base[5]: il *pair programming* (programmazione in coppia).

Uno dei due programmatori svolge il ruolo di *driver* e scrive il codice, mentre l’altro (*navigator*) supervisiona, ricerca eventuali errori e può suggerire strategie alternative. È necessario attenersi ad alcune regole: per esempio il *navigator* non può “sottrarre” il mouse al *driver*, e deve evitare di assumere un atteggiamento troppo direttivo o disconfermante; inoltre è essenziale una comunicazione continua ed efficace tra i due. I ruoli devono essere poi scambiati nel corso di ciascuna sessione di lavoro (almeno in ambito didattico).

5.7.2 Aiuto tra pari

Nel corso di una sessione di programmazione individuale con Scratch – che si tratti di progetto libero, esercizi di *debug*, progetti proposti dall’insegnante, *remix*, ecc. – ci saranno sicuramente momenti in cui alcuni bambini, pur provando e riprovando, non riescono a risolvere un problema o a ottenere un certo risultato.

[5] https://www.youtube.com/watch?v=vgkahOzFH2Q

Anche in un contesto che accoglie l'errore come un evento interessante e stimola a ricercare la soluzione, è inevitabile che talvolta occorra un intervento esterno per superare un'*impasse*. Spetta al docente valutare le singole situazione e calibrare suggerimenti e aiuti nel modo più proficuo. Tuttavia si può seguire anche un'altra strada, in alternativa o parallelamente: lasciare che sia un compagno ad aiutare l'alunno in difficoltà.

Non si tratta di *peer tutoring*, che è una metodologia di più ampio respiro e molto più strutturata: essa richiede tempi medio-lunghi e sessioni a cadenza regolare, formazione specifica del tutor, obiettivi di apprendimento ben definiti (Topping 2014; Baldacci 2004).

Quello che si vuole proporre è una situazione didattica assai più semplice, informale e occasionale, di breve o brevissima durata, in cui un problema specifico viene affrontato con l'aiuto di un compagno (non necessariamente più "esperto"). È una modalità utilizzata anche nelle prassi didattiche dei CoderDojo, riassunta nello slogan "Chiedi a tre, poi a me": prima di richiedere l'intervento di un *mentor*, i giovani programmatori sono invitati a rivolgersi a chi è seduto accanto o davanti a loro.

I vantaggi di questa modalità di lavoro, dal punto di vista cognitivo, sono simili a quelli della programmazione di coppia: vengono affinate la capacità di decentramento e l'elaborazione metacognitiva, tanto da parte dell'"aiutante" quanto da parte di chi ha richiesto l'intervento, che deve riuscire a spiegare in modo chiaro i termini del problema. Se l'aiuto risulta utile, poi, il "tutor" può ricavare benefici dal punto di vista della percezione di autoefficacia e dell'autostima in generale.

Oltre agli aspetti cognitivi, è chiaramente in gioco anche la dimensione relazionale; sarà compito dell'insegnante osservare e monitorare le dinamiche in atto, prestando attenzione all'atteggiamento sia di chi aiuta sia di chi viene aiutato, al clima delle interazioni, a eventuali segnali di disagio.

5.7.3 Gruppi di confronto

Un'idea interessante proposta nella guida *Creative Computing* (Brennan, Balch & Chung 2014, 20-21) è quella di costituire piccoli gruppi (3-4 alunni) in cui ciascuno presenta il progetto che ha appena concluso o a cui sta lavorando, e

riceve dagli altri impressioni, opinioni ed eventuali suggerimenti. Eventualmente, per ciascun progetto si può utilizzare uno strumento di rilevazione codificato in base ai colori del semaforo (*ibidem*), in cui ogni membro esplicita: che cosa sembra non funzionare o necessita di essere migliorato (rosso); quali aspetti risultano confusi o potrebbero essere realizzati diversamente (giallo); che cosa funziona bene e risulta convincente o accattivante (verde).

L'utilità di questo tipo di feedback è duplice, e anche in questo caso ha a che fare da un lato con la metacognizione (spiegare il proprio progetto a un "pubblico" che non lo conosce), dall'altro con il decentramento (comprendere il punto di vista altrui).

Naturalmente è importante che gli alunni comprendano bene lo spirito e la finalità del gruppo di confronto[6], che non deve in alcun modo favorire dinamiche competitive. In questo senso saranno determinanti la familiarità del gruppo classe con le metodologie didattiche attive (e il conseguente clima relazionale instaurato) e il costante monitoraggio da parte del docente.

5.7.4 Partecipare alla *community*

In questa proposta la classe è un laboratorio e una comunità di apprendimento, in cui le conoscenze vengono costruite, condivise, scambiate. Ma Scratch offre la possibilità di partecipare anche a una comunità molto più vasta, che non condivide uno stesso spazio fisico ma interagisce tramite canali e strumenti messi a disposizione dal sito web. Un'ottima occasione per approcciare aspetti e contenuti di *media education* (Rivoltella 2001), sempre più cruciali in ambito educativo man mano che i *social network* aumentano la loro pervasività e il potenziale impatto sulle rappresentazioni di sé e sulla sfera emotiva.

Il fatto che Scratch metta in gioco motivazione, passione, curiosità e collaborazione, cioè che – in definitiva – sia un ambiente focalizzato sui progetti e sul piacere di imparare a fare cose nuove e stimolanti, può essere una chiave molto positiva per far entrare i bambini nel mondo dei *social media*. Perciò è importante invitare gli alunni, accanto all'esplorazione dei progetti e alle attività

[6] L'originale è *critique group*; non ho tradotto "gruppo di critica" proprio per non alimentare equivoci, dato che in italiano, tra le accezioni del termine "critica", di solito si pensa automaticamente a quella negativa.

di *remix*, anche a comunicare con altri utenti, commentare i loro progetti, scambiare opinioni. Oltre alle ricadute sugli apprendimenti, ciò ha una valenza educativa anche per lo sviluppo di competenze trasversali, comunicative e di cittadinanza. Il docente dovrà fare attenzione all'appropriatezza delle interazioni e al rispetto delle norme di *netiquette*, all'occorrenza richiamandole esplicitamente.

5.8 La valutazione

Negli ultimi anni il tema della valutazione degli apprendimenti in ambito scolastico è diventato sempre più centrale, tanto nella ricerca quanto nella sfera delle politiche nel settore dell'educazione, inserendosi nel più ampio contesto della valutazione complessiva del sistema[7]:

> In un sistema scolastico e formativo moderno ed efficace la valutazione si configura come strumento insostituibile di costruzione delle decisioni e come fulcro delle azioni necessarie per governarne – a tutti i livelli di responsabilità – il funzionamento e per adeguarlo dinamicamente alle necessità consolidate ed emergenti (Domenici 2003, 5-6).

Oggi è dunque ampiamente acquisita la centralità della dimensione valutativa nei processi formativi: lungi dall'essere solo un momento conclusivo o accessorio, di certificazione/sanzione dei risultati raggiunti, essa riveste un'importanza fondamentale come *fonte di informazioni*, tanto sugli apprendimenti degli alunni quanto sulla mediazione didattica dei docenti.

> La funzione valutativa deve accompagnare ogni momento del percorso formativo del soggetto e dell'azione didattica, fornendo un riscontro costante su come sta procedendo l'apprendimento e allo stesso tempo monitorare l'efficacia degli interventi predisposti dai docenti (Capperucci 2011, 11).

> La valutazione diventa il mezzo per intervenire sull'iter formativo evidenziando i punti di forza e di debolezza delle situazioni formative e permettendo, così, di correggere e migliorare il perseguimento degli obiettivi. La valutazione assume la funzione di *feedback* (di retroazione) sullo svolgimento dell'azione formativa, e

[7] Con l'autonomia degli istituti scolastici, infatti, accanto alla valutazione degli apprendimenti nel contesto-classe (microsistema), hanno assunto grande importanza anche l'autovalutazione degli istituti (mesosistema) e la valutazione del sistema educativo nel suo complesso (macrosistema) (Domenici 2003; Vannini 2009).

> quindi di controllo attraverso la raccolta di informazioni e la loro analisi […] La valutazione, quindi, viene a configurarsi non tanto come attività accessoria alle azioni formative, quanto piuttosto come processo che interagisce costantemente con l'intero processo formativo (*ibidem*, 26).

Oltre a queste prospettive generali, anche i metodi e gli strumenti della valutazione scolastica, negli anni recenti, sono stati oggetto di ricerche e proposte varie e articolate. Accanto alle fondamentali acquisizioni della docimologia (misurazione e analisi dei dati, procedure di standardizzazione, tipologia di strumenti di rilevazione, validità e affidabilità), si è infatti sviluppato un approccio strettamente collegato alla didattica per competenze: la *valutazione autentica* (Capperucci 2011). Gli aspetti qualificanti di questa metodologia si possono riassumere come segue.

- Le prove di valutazione devono essere basate su compiti realistici, significativi, complessi, eventualmente anche mal formulati (in quanto più simili ai problemi che si affrontano nella realtà).
- In questo modo esse non richiedono la semplice applicazione di tecniche o procedure, bensì la messa in campo di competenze a loro volta complesse e articolate, compresa la capacità di fare ricorso a fonti informative esterne oltre alle proprio conoscenze.
- Oltre a conoscenze e abilità, vengono valorizzati il pensiero critico, la creatività e il pensiero divergente, la dimensione metacognitiva.

Tra gli strumenti di rilevazione un ruolo fondamentale è svolto dalle *rubriche* (Comoglio 2002; Ellerani 2005). Una rubrica consiste in una scala di punteggi associati a una lista di criteri che descrivono ciascun grado della scala. In altri termini, una volta individuati la competenza che si intende valutare e i relativi criteri di rilevazione, i vari livelli raggiungibili devono essere descritti in modo preciso con opportuni indicatori.

Naturalmente non si tratta di "schierarsi" a favore delle prove oggettive o della valutazione autentica; metodologie e strumenti diversi devono arricchire il repertorio del docente, che saprà scegliere di volta in volta quelli più adatti alla situazione didattica e al tipo di apprendimenti da valutare, come auspica Capperucci: «Attualmente sussistono i presupposti per un superamento di dette posizioni dicotomiche a favore di un approccio integrato che vede una possibile

"contaminazione" tra i due modelli in base agli oggetti e alle finalità della valutazione» (*ibidem*, 69).

5.8.1 Pensiero computazionale e valutazione degli apprendimenti

Brennan e Resnick (2012) hanno elaborato un contributo molto interessante per la valutazione dell'apprendimento del pensiero computazionale. Uno dei vantaggi di questa proposta risiede nel fatto che la valutazione si basa sulla stessa definizione operativa utilizzate nel presente capitolo (articolata in concetti, pratiche e prospettive computazionali).

Gli autori presentano e discutono criticamente tre differenti strumenti di valutazione, illustrando i punti di forza e le criticità di ciascuno.

- *Analisi del portfolio dei progetti*, con l'ausilio di un *tool* che consente un'analisi quantitativa dell'utilizzo dei singoli blocchi di Scratch. Un punto di forza di questo metodo è che in molti casi è possibile associare a un blocco un determinato concetto computazionale, assumendo quindi che l'utilizzo di quel comando implica una competenza con quel determinato concetto. Però in questo modo non è possibile valutare né le pratiche né le prospettive; inoltre l'analisi è focalizzata esclusivamente sui prodotti e non fornisce dati sui processi che hanno condotto lo studente alla realizzazione dei progetti.
- *Interviste individuali* basate sui progetti realizzati, attraverso un protocollo standard, di durata variabile da un'ora a due ore. Questo strumento permette di indagare tutte le dimensioni del pensiero computazionale, oltre ai processi di costruzione dei progetti e alla rielaborazione metacognitiva. Tuttavia richiede una grande disponibilità di tempo; inoltre i risultati sono "filtrati" da ciò che lo studente è in grado di ricordare e verbalizzare.
- *Presentazione di alcuni progetti da parte del docente*, per ciascuno dei quali l'alunno è chiamato a: spiegare il funzionamento; descrivere come si potrebbe ampliare; correggere un *bug*; remixare il programma. In questo modo è possibile valutare un'ampia gamma di competenze, ma – come nel caso precedente – occorre avere molto tempo a disposizione; inoltre la somministrazione di stimoli esterni può non incontrare gli

interessi, la motivazione e gli stili cognitivi dei discenti, e più in generale non è del tutto coerente con l'approccio dell'apprendimento creativo.

Brennan e Resnick concludono che tutti e tre i metodi presentano potenzialità ed elementi di interesse, ma nessuno si dimostra del tutto soddisfacente. Una soluzione potrebbe essere il loro utilizzo combinato, anche se può scontrarsi con limitazioni in termini di tempo.

Infine, i due autori avanzano alcuni suggerimenti per lo sviluppo di ulteriori metodi e strumenti di valutazione, tra cui l'attenzione tanto ai prodotti quanto ai processi, la considerazione degli interessi e degli stili cognitivi dei singoli studenti, l'inclusione di altri punti di vista oltre a quello del docente (genitori, educatori, alunni stessi).

Elaborando questi spunti, nella cornice teorica e metodologica esposta nel paragrafo precedente, è ora possibile delineare la dimensione valutativa all'interno del percorso didattico delineato in questo capitolo.

- Dato che questa proposta è stata pensata come introduttiva, non verrà svolta un'attività specifica di *valutazione diagnostica*. Ciò non significa che gli studenti non possano avere conoscenze o competenze pregresse (è anche possibile che qualcuno utilizzi già Scratch a casa o in altri contesti extrascolastici!), che andranno osservate e rilevate; semplicemente, non avrebbe senso somministrare prove dirette all'accertamento di tali competenze. Grande importanza è invece assegnata alla *valutazione formativa* (*in itinere*) (Vannini 2009), come più volte accennato in questo capitolo. Considerato il carattere aperto e creativo dell'approccio didattico adottato, l'idea di fondo è quella di una dialettica continua fra le attività portate avanti dagli studenti, gli interventi del docente nel ruolo di facilitatore e organizzatore dell'ambiente di apprendimento, la valutazione *in itinere*. Al termine del percorso è poi necessaria una *valutazione sommativa*, anch'essa condotta con metodi e tecniche coerenti con l'approccio didattico utilizzato.

- Per quanto riguarda l'approccio metodologico, appare abbastanza naturale rivolgersi al modello della *valutazione autentica*. Quella qui proposta è una didattica basata sullo sviluppo di progetti significativi, con livelli potenzialmente elevati di complessità e frequenti attività di *problem solving*; è perciò possibile assimilare la maggior parte dei progetti alla tipologia del compito autentico.
- A livello di tecniche e strumenti, è anzitutto fondamentale l'*osservazione*, intesa non in senso generico ma come tecnica specifica (Camaioni, Aureli & Perucchini 2004). Lo strumento di rilevazione principale è individuato nelle *rubriche*, che il docente dovrà mettere a punto per identificare i livelli di apprendimento di concetti, pratiche e prospettive computazionali. Per l'impostazione didattica qui presentata, si ritiene coerente *non* fare ricorso a prove di verifica strutturate o semistrutturate.
- Anche se il *focus* prevalente sarà sui processi, può essere utile prevedere anche un'analisi del portfolio dei progetti realizzati da ciascun allievo (in particolare il "progetto libero" di più ampio respiro), anche se in questo modo si possono ricavare informazioni solo sull'acquisizione dei concetti computazionali.
- Infine è interessante l'idea della conversazione (anche breve, a differenza delle interviste proposte da Brennan e Resnick) con ciascun alunno, che può fornire indicazioni preziose su un livello di difficile valutazione come quello delle *prospettive* computazionali, oltre a stimolare la rielaborazione metacognitiva (ma questo può essere anche un limite: non è detto che un alunno sia in grado di verbalizzare e far emergere tutti gli apprendimenti). È opportuno che questo strumento venga utilizzato anche per l'*autovalutazione* da parte degli alunni, che devono essere resi partecipi del processo.

5.9 Non solo pensiero computazionale: Scratch e gli apprendimenti disciplinari

L'oggetto di questo lavoro è l'utilizzo di Scratch per l'apprendimento di concetti, pratiche e prospettive di pensiero computazionale. Tuttavia Scratch è

anche uno *strumento di mediazione didattica per apprendimenti disciplinari* (Patassini 2014; Rabbone 2014). Il Centro Studi Impara Digitale, per esempio, organizza corsi di formazione sull'uso di Scratch come strumento di supporto per la progettazione didattica multidisciplinare, incentrati prevalentemente su italiano e matematica, ma con proposte riguardanti anche storia, geografia, scienze, musica e tecnologia[8].

Si tratta di un campo potenzialmente molto vasto, che si è scelto consapevolmente di non prendere in considerazione. A conclusione del capitolo, però, è doveroso farvi cenno, proponendo qualche esempio.

Un aspetto da sottolineare – per quanto forse scontato – è che non è necessario "scegliere" tra obiettivi curricolari e pensiero computazionale: in un progetto di Scratch mirato all'esplorazione o al consolidamento di concetti matematici, per esempio, è sempre presente *anche* la dimensione computazionale; i due ordini di obiettivi vengono perseguiti simultaneamente.

L'ambito matematico è forse quello in cui le possibilità di utilizzo di Scratch sono state maggiormente esplorate, in parte perché i blocchi di programmazione del gruppo "penna" sono una riproposizione aggiornata dei comandi del Logo. Senza addentrarsi in esempi specifici, il fatto di poter tracciare un percorso programmato dall'utente (quindi, di fatto, disegnare forme geometriche) offre un supporto molto interessante per lo studio della geometria piana: alcune proprietà degli angoli e dei poligoni possono essere "scoperte" costruendoli con Scratch.

Lo *stage*, inoltre, è di fatto un piano cartesiano[9]: la posizione di uno *sprite* è espressa da una coppia di coordinate (le ascisse vanno da -240 a +240, le ordinate da -180 a +180). Anche se non è strettamente necessario utilizzarle per far muovere uno *sprite*, ci sono molti casi in cui risultano indispensabili, per esempio quando si vuole stabilire la posizione esatta di uno *sprite* nello *stage*. Perciò i bambini hanno la possibilità di utilizzare il piano cartesiano prima di averlo affrontato come argomento di studio; certo con qualche difficoltà, ma di nuovo con tutti i vantaggi derivanti da un apprendimento per scoperta, significativo e

[8] http://www.imparadigitale.it/wp-content/uploads/2016/01/Corsi_formazione_ID-2015_2016_ok.pdf

[9] Dalla libreria degli sfondi è possibile caricare la griglia delle coordinate.

motivato da esigenze “reali” (l’alunno impara a usare le coordinate perché in quel momento *gli servono* per realizzare il suo progetto).

Per quanto riguarda i traguardi e gli obiettivi di apprendimento in italiano, Scratch è particolarmente adatto per sviluppare narrazioni multimediali (Patassini 2014): lo *storytelling* digitale è un campo molto interessante in cui far convergere competenze relative alla produzione di testi scritti e all’utilizzo combinato di diversi linguaggi e codici espressivi (immagini, suoni e musica, ecc.). Può essere stimolante, per esempio, far realizzare ai bambini una trasposizione “animata” di un racconto, magari scritto in precedenza da loro stessi. Questa attività potrebbe inserirsi in un più ampio percorso multidisciplinare che comprende il curricolo di arte e immagine, ed esplora e mette in relazione caratteristiche e peculiarità del testo scritto, del cinema e del fumetto.

Scratch comprende poi un editor musicale che può essere proficuamente utilizzato nell’insegnamento della materia (anche per esplorare alcuni concetti computazionali applicati alla musica, per esempio le sequenze e i cicli), accanto ad altri software specifici eventualmente impiegati.

Infine, per quanto concerne l’insegnamento della storia, della geografia e delle scienze, la modalità più semplice per utilizzare Scratch è quella di far realizzare presentazioni (meglio se interattive) di un determinato argomento studiato. In questo caso non si tratta di didattica costruttivista, ma di un utilizzo strumentale per produrre un’esposizione dei contenuti appresi; tuttavia le potenzialità creative ed espressive di Scratch potrebbero favorire una rielaborazione più ricca e fruttuosa rispetto ad altri “prodotti”, come una presentazione in PowerPoint o una relazione scritta. Per esempio, invece di “affidare” a uno *sprite* una semplice ripetizione di quanto appreso, lo studente potrebbe creare un gioco interattivo (sui monumenti storici della sua città, sulla fotosintesi clorofilliana, sulla civiltà egizia o su qualsiasi altro argomento), raggiungendo in questo modo un elevato livello di rielaborazione dei contenuti e un apprendimento significativo.

CONCLUSIONI

Questo lavoro, a ben guardare, si snoda lungo due direttrici fondamentali. Da un lato, partendo dalla ricerca soprattutto in ambito informatico, si è cercato di rendere conto della rilevanza generale che sta assumendo in campo educativo il pensiero computazionale e dell'opportunità di iniziare ad apprenderlo fin dal primo ciclo di istruzione. È probabile che ci troviamo nella fase iniziale di un'elaborazione (di strumenti, metodi, curricoli) destinata ad ampliarsi nei prossimi anni, che potrebbe approdare nell'inserimento stabile, in tutti i gradi scolastici, dell'insegnamento del pensiero computazionale (è presto per ipotizzare con quale statuto disciplinare: "materia" a sé stante? approccio interdisciplinare? oppure all'interno del curricolo relativo alle "tecnologie"?).

Dall'altro lato questa tesi costituisce un esempio di proposta didattica elaborata sulla base di un approccio teorico e metodologico ben definito ed esplicitato, attraverso l'impiego di uno strumento di mediazione che presenta caratteristiche e potenzialità coerenti con quell'approccio. Le peculiarità di Scratch lo rendono particolarmente adatto a promuovere un apprendimento attivo e costruttivo, sia a livello individuale sia all'interno di un ambiente collaborativo. Il punto, naturalmente, è che l'utilizzo delle tecnologie digitali (come di qualunque altro strumento di mediazione didattica) non si giustifica di per sé, ma solo se inserito in una prospettiva didattica consapevole.

Naturalmente sono molti gli aspetti che non è stato possibile affrontare. Per esempio l'utilizzo di uno strumento come Scratch a supporto della didattica multidisciplinare, cui si è solo fatto cenno nell'ultimo capitolo. Oppure al rapporto fra apprendimenti scolastici ed extrascolastici, che in questo caso è molto rilevante: è possibile – se non probabile – che, una volta "scoperto" Scratch, alcuni bambini si appassionino al punto di utilizzarlo regolarmente anche a casa, intraprendendo un percorso di approfondimento personale che può condurre all'acquisizione di competenze anche molto elevate. Sarebbe interessante rimetterle in gioco in classe: si tratterebbe di risorse ulteriori da far interagire nella comunità di apprendimento, perseguendo al contempo le strategie dell'individualizzazione e della personalizzazione.

BIBLIOGRAFIA

Attivissimo, P. (2013). *Per favore, non chiamateli nativi digitali.* http://www.agendadigitale.eu/competenze-digitali/550_per-favore-non-chiamateli-nativi-digitali.htm

Ausubel, D. P. (1978). *Educazione e processi cognitivi. Guida psicologica per gli insegnanti*. Introduzione di Cesare Cornoldi e Paolo Meazzini. Ed. it. a cura di Daniela Costamagna. Milano: FancoAngeli.

Baldacci, M. (Ed.). (2004). *I modelli della didattica*. Roma: Carocci.

Barba, L. A. (2016). *Computational Thinking: I do not think it means what you think it means.* https://medium.com/@lorenaabarba/computational-thinking-i-do-not-think-it-means-what-you-think-it-means-6d39e854fa90#.22ccbl5hw

Bell, T., Witten, I. H., & Fellows, M. (2009). *Computer Science Unplugged*. Ed. it. a cura di Bianco, G. M. e Davoli, R. http://csunplugged.org/

Ben-Ari, M. (2001). Constructivism in computer science education. *Journal of Computers in Mathematics and Science Teaching*, 20(1), 45-74.

Borkowski, J. G., & Muthukrishna, N. (2011). *Didattica metacognitiva. Come insegnare strategie efficaci di apprendimento*. Tento: Erickson.

Boscolo, P. (2006). *La motivazione ad apprendere*. In Mason (2006), 91-117.

Brennan, K., Balch, C., & Chung, M. (2014). *Creative Computing.* Cambridge (MA): Harvard Graduate School of Education. Distribuito con licenza Creative Commons.

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 annual meeting of the American Educational Research Association, Vancouver, Canada* (pp. 1-25).

Bruner, J. (1997). *La cultura dell'educazione. Nuovi orizzonti per la scuola*. Milano: Feltrinelli.

Cacciamani, S. (2008). *Imparare cooperando. Dal Cooperative Learning alle comunità di ricerca*. Roma: Carocci.

Calvani, A. (2000). *Elementi di didattica. Problemi e strategie*. Roma: Carocci.

Camaioni, L., Aureli, T., & Perucchini, P. (2004). *Osservare e valutare il comportamento infantile*. Bologna: il Mulino.

Capperucci, D. (Ed.). (2011). *La valutazione degli apprendimenti in ambito scolastico*. Milano: FrancoAngeli.

Carletti, A. (2005). Il costruttivismo: elementi epistemologici. In Carletti, A., & Varani, A. (Eds.), *Didattica costruttivista. Dalle teorie alla pratica in classe* (pp. 15-53). Trento: Erickson.

Carletti, A., & Varani, A. (Eds.). (2005). *Didattica costruttivista. Dalle teorie alla pratica in classe*. Trento: Erickson.

Comoglio, M. (2002). La «valutazione autentica». *Orientamenti pedagogici*, 49 (289), 93-112.

Comoglio, M., & Cardoso, M. A. (1996). *Insegnare e apprendere in gruppo. Il Cooperative Learning*. Roma: LAS.

Cornoldi, C. (1995). *Metacognizione e apprendimento.* Bologna: il Mulino.

Dasgupta, S. (2016). *Studying the relationship between remixing & learning*. https://medium.com/mit-media-lab/studying-the-relationship-between-remixing-learning-c1df54c302df#.bva86860a

Dasgupta, S., Hale, W., Monroy-Hernández, A., & Hill, B. M. (2016). Remixing as a pathway to computational thinking. In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing* (pp. 1438-1449). ACM.

De Beni, R. (2003). *Psicologia cognitiva dell'apprendimento: aspetti teorici e applicazioni*. Trento: Erickson.

Domenici, G. (2003). *Manuale della valutazione scolastica*. Roma-Bari: Laterza.

Downey, A., Elkner, J., & Meyers, C. (2002). *How to Think like a Computer Scientist: Learning with Python.* Wellesley (MA): Green Tea Press. Trad. it. di Alessandro Pocaterra, *Pensare da informatico. Imparare con Python*. Distribuito con GNU Free Documentation License.

Ellerani, P. (2005). Rubriche e valutazione autentica. *G. Cerini, M. Spinosi. Voci della Scuola*, 5, 457-469.

Ferraresso, A., Ferraresso, L., Colombini, E., & Bonanome, G. (2015). *Coding. Programmare è un gioco*. Novara: De Agostini Scuola.

Ferri, P. (2011). *Nativi digitali*. Milano: Bruno Mondadori.

Fini, A. (2011). *Nativi digitali. Un aggiornamento sulla discussione in rete*. http://bricks.maieutiche.economia.unitn.it/wp-content/uploads/2011/06/LIM_17_Fini.pdf

Frabboni, F. (2007). *Manuale di didattica generale*. Roma-Bari: Laterza.

Gardner, H. (1994). *Intelligenze multiple*. Milano: Anabasi, Milano.

Gherardi, V. (2010). *La didattica nella scuola di base. Professionalità e strategie nella costruzione dei saperi*. Roma: Carocci.

Gherardi, V. (2013). *Metodologie e didattiche attive. Prospettive teoriche e proposte operative*. Roma: Aracne.

Gui, M. (Ed.). (2013). *Indagine sull'uso dei nuovi media tra gli studenti delle scuole superiori lombarde*. Regione Lombardia.

Lodi, M. (2014). *Imparare il pensiero computazionale, imparare a programmare*. Tesi di Laurea Magistrale in Informatica, Università di Bologna. http://amslaurea.unibo.it/6730/

Lodi, M., & Moscetti, C. (in corso di pubblicazione). Prefazione a Giordano, M. *Coding e Pensiero computazionale. Materiali per l'insegnante*. Loreto: ELI – La Spiga.

Macchi Cassia, V., Valenza, E., & Simion, F. (2012). *Lo sviluppo della mente umana. Dalle teorie classiche ai nuovi orientamenti*. Bologna: il Mulino.

Marchignoli, R., & Lodi, M. (in corso di pubblicazione). *EAS e pensiero computazionale*. Bescia: La Scuola.

Mason, L. (2006). *Psicologia dell'apprendimento e dell'istruzione*. Bologna: il Mulino.

McKenzie, W. (2006). *Intelligenze multiple e tecnologie per la didattica*. Trento: Erickson.

Meo, A. R. (2002). Software libero e "open source". *Mondo Digitale*, 1(2), 3-17.

Ministero della Pubblica Istruzione (2007). *Il nuovo obbligo di istruzione: cosa cambia nella scuola?* Firenze: Agenzia nazionale per lo sviluppo dell'autonomia scolastica.

Ministero dell'Istruzione, dell'Università e della Ricerca (2012). *Indicazioni nazionali per il curricolo della scuola dell'infanzia e del primo ciclo d'istruzione*.

Ministero dell'Istruzione, dell'Università e della Ricerca (2015). *Piano Nazionale Scuola Digitale*.

Ministero dell'Istruzione, dell'Università e della Ricerca (2016). *Programma il Futuro. Monitoraggio dell'andamento del progetto. Settembre 2015 – gennaio 2016*.

Morin, E. (2000). *La testa ben fatta. Riforma dell'insegnamento e riforma del pensiero.* Milano: Raffaello Cortina.

Nardelli, E. (2014). *Informatica: dal coding al pensiero computazionale*. http://www.ilfattoquotidiano.it/2014/08/24/informatica-dal-coding-al-pensiero-computazionale/1097593/

Nardelli, E. (2015). *Informatica, il pensiero computazionale: una competenza per il futuro*. http://www.ilfattoquotidiano.it/2015/06/26/informatica-il-pensiero-computazionale-una-competenza-per-il-futuro/1816578/

Nigris, E. (2007a). Dalla valorizzazione dell'esperienza alla didattica sociocostruttivista. In Nigris, E., Negri, S. C., & Zuccoli, F. (Eds.), *Esperienza e didattica. Le metodologie attive* (pp. 81-124). Roma: Carocci.

Nigris, E. (2007b). Esperienza ed educazione. In Nigris, E., Negri, S. C., & Zuccoli, F. (Eds.), *Esperienza e didattica. Le metodologie attive* (pp. 25-79). Roma: Carocci.

Nigris, E., Negri, S. C., & Zuccoli, F. (Eds.). (2007). *Esperienza e didattica. Le metodologie attive*. Roma: Carocci.

Novak, J. (2001). *L'apprendimento significativo*. Trento: Erickson.

Papert, S. (1980). I computer e le culture del computer. Trad. it. del Capitolo 1 di *Mindstorms: Children, Computers, and Powerful Ideas*. New York, NY: Basic Books. Disponibile online all'indirizzo https://drive.google.com/file/d/0BwmWF43T3hVRaUpaUEJoS0F3MXM/view

Papert, S. (1993). *The Children's machine: Rethinking Schools in the Age of the Computer*. New York, NY: Basic Books.

Papert, S. (1996). An exploration in the space of mathematics educations. *International Journal of Computers for Mathematical Learning*, 1(1), 95-123.

Parente, M. (2004). Il modello delle competenze di base. In Baldacci, M. (Ed.), *I modelli della didattica* (pp. 63-96). Roma: Carocci.

Patassini, A. (2014). *Storytelling con Scratch*. https://ltaonline.wordpress.com/2014/11/14/storytelling-con-scratch/

Prensky, M. (2001). Digital natives, digital immigrants part 1. *On the horizon*, 9(5), 1-6.

Prensky, M. (2009). H. sapiens digital: From digital immigrants and digital natives to digital wisdom. *Innovate: journal of online education*, 5(3), 1-8.

Rabbone, A. (2014). *Scratch come ambiente educativo*. http://bambinicheimparanoaprogrammare.blogspot.it/2014/10/scratch-come-ambiente-educativo.html?view=magazine

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., & Kafai, Y. (2009). Scratch: programming for all. *Communications of the ACM*, 52(11), 60-67.

Rivoltella, P. C. (2001). *Media Education. Modelli, esperienze, profilo disciplinare*. Roma: Carocci.

Rivoltella, P. C. (2010). *Da Marc Prensky… a Marc Prensky.* http://piercesare.blogspot.it/2010/10/da-marc-prensky-marc-prensky.html

Selby, C., & Woollard, J. (2013). Computational thinking: the developing definition.

Sherry, B. *Constructionism. The Child at the Centre.* https://bsherry.wordpress.com/thinking-about-learning-2/constructionism/

Soloway, E. (1986). Learning to program = learning to construct mechanisms and explanations. *Communications of the ACM*, 29(9), 850-858.

Stallman, R. (2016). *Why Open Source misses the point of Free Software*. https://www.gnu.org/philosophy/open-source-misses-the-point.en.html

Terravecchia, G. P. (2013). *Il dibattito italiano sui nativi digitali*. http://www.laricerca.loescher.it/istruzione/694-il-dibattito-italiano-sui-nativi-digitali.html

Topping, K. (2014). *Tutoring. L'insegnamento reciproco tra compagni*. Trento: Erickson.

Vannini, I. (2009). *La qualità nella didattica. Metodologie e strumenti di progettazione e valutazione*. Trento: Erickson.

Varani, A. (2005). Insegnamento, apprendimento e metacognizione. In Carletti, A., & Varani, A. (Eds.), *Didattica costruttivista. Dalle teorie alla pratica in classe* (pp. 295-311). Trento: Erickson.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33-35.

DICHIARAZIONE

Il Sottoscritto OLIVARI FRANCESCO dichiara sotto la propria responsabilità ai sensi dell'articolo 47 del D.P.R. 445/2000 di aver elaborato la presente Tesi e la Relazione finale di tirocinio autonomamente. I pensieri e le formulazioni riprese da fonti non proprie sono debitamente citati.

I presenti lavori, in forma uguale o simile, non sono stati fino ad ora presentati ad altra Commissione d'esame nonché pubblicati.

Il Sottoscritto è consapevole delle conseguenze legali che una falsa dichiarazione può comportare.

Data

20/10/2016

firma Laureando