Form@re
Open Journal per la formazione in rete

articoli

Problem solving and computational thinking. Building synergies between conceptualization and coding starting from primary school

Problem solving e pensiero computazionale. Costruire sinergie tra concettualizzazione e codifica a partire dalla scuola primaria

Roberto Trinchero[a]

[a] *Università degli Studi di Torino, roberto.trinchero@unito.it*

Abstract

Coding and computational thinking are two terms increasingly present in the scholastic debate of last years. The recent *Italian guidelines and new scenarios* (MIUR, 2018) have given to computational thinking an important role today in school. However, it is often confusing the teaching of coding (that is, the teaching of a specific programming language) with the teaching of computational thinking, that requires to take into account the relationships between this, problem solving and competences. In fact, coding is only one of the moments of computational thinking, which must be preceded by a moment of conceptualization, linked to the analysis and decomposition of the problems subject to codification. The article proposes reflections and useful hints for working on computational thinking starting from the first year of primary school, through a re-edition in the form of an educational game of the famous *Logo* language.

Keywords: computational thinking; coding; guided problem solving; primary school; *Logo* language.

Abstract

Coding e pensiero computazionale sono ormai due termini sempre più presenti nel dibattito scolastico già da alcuni anni e le recenti *Indicazioni nazionali e nuovi scenari* (MIUR, 2018) hanno dato ancor più risalto al ruolo che dovrebbero avere nella scuola odierna. Spesso però si confonde l'insegnare il coding, ossia uno specifico linguaggio di programmazione, con l'insegnare il pensiero computazionale, trascurando i rapporti che vi sono tra questo, il problem solving e l'esercizio di competenze. Il coding è infatti solo uno dei momenti del pensiero computazionale, che va preceduto da un momento di concettualizzazione, legato all'analisi e alla scomposizione dei problemi oggetto di codifica. L'articolo propone riflessioni e spunti operativi utili per lavorare sul pensiero computazionale a partire dal primo anno della scuola primaria, attraverso una riedizione in forma di gioco didattico del celebre linguaggio *Logo*.

Parole chiave: pensiero computazionale; coding; problem solving guidato; scuola primaria; linguaggio *Logo*.

1. Introduzione

Come ben descritto nelle *Indicazioni nazionali per il curricolo della scuola dell'infanzia e del primo ciclo d'istruzione* (MIUR, 2012), l'obiettivo della scuola non può essere quello di inseguire lo sviluppo di singole tecniche e competenze, dato che queste possono diventare obsolete in pochi anni, ma piuttosto quello di formare saldamente gli allievi sul piano cognitivo e culturale, allo scopo di fornire strumenti per affrontare positivamente l'incertezza e la mutevolezza degli scenari sociali e professionali, presenti e futuri. Più che sulla trasmissione standardizzata di conoscenze e abilità il focus dovrebbe essere messo sul far emergere e coltivare le inclinazioni personali degli studenti, nella prospettiva di valorizzare gli aspetti peculiari della personalità di ognuno, e di sviluppare quelle capacità analitiche, riflessive e metodiche che servono a portare al massimo compimento le potenzialità di ciascun allievo. Le finalità della formazione scolastica dovrebbero quindi essere quelle di offrire agli studenti occasioni di apprendimento dei saperi e dei linguaggi culturali di base, far acquisire gli strumenti di pensiero necessari per apprendere a selezionare le informazioni, promuovere la capacità di elaborare metodi e categorie che siano in grado di fare da bussola negli itinerari personali, favorire l'autonomia di pensiero degli studenti, orientando la propria didattica alla costruzione di saperi a partire da concreti bisogni formativi.

Un sapere imprescindibile nel panorama odierno è quello legato al pensiero computazionale (Bellanca & Brandt, 2010; Clements & Nastasi, 1999; Fessakis, Gouli & Mavroudi, 2013; Trilling & Fadel, 2009; Wing, 2006; 2008) ossia un processo di pensiero, applicabile a problemi di varia natura, che entra in gioco nel formulare un problema ed esprimere le sue soluzioni in una forma tale che un computatore, umano o meccanico, possa effettivamente processare.

Il pensiero computazionale ha cambiato radicalmente la vita quotidiana e lavorativa in una pluralità di ambiti. L'uso sempre più capillare di dispositivi informatizzati ha dato un ruolo sempre più pervasivo all'interazione uomo-macchina e all'uso di codici di comunicazione con diversi dispositivi. In ambiti di ricerca scientifica, la disponibilità di risorse computazionali a basso costo ha favorito il diffondersi di metodi e tecniche statistiche ad alta intensità di calcolo (ad es. metodi bayesiani, metodi *Monte Carlo*, tecniche di data mining), ampiamente utilizzati nelle discipline scientifiche ed umanistiche (Bundy, 2007), e reso possibile lavorare su basi di dati enormi (Sin & Muthu, 2015). La simulazione computerizzata ha diffuso l'uso della modellizzazione in moltissime scienze, ponendo il problema di come i modelli pensati dai ricercatori siano influenzati dalle possibilità e dai vincoli offerti dai codici di comunicazione con gli apparati tecnologici di rilevazione, elaborazione e simulazione.

La capillarità dell'interazione umana con i dispositivi informatici ha portato Jeannette Wing (2006) ad assegnare al pensiero computazionale il ruolo di quarta abilità di base, che si aggiunge al leggere, scrivere e far di conto, e che quindi dovrebbe essere insegnato a tutti, fin dalla scuola primaria. Questo però può portare a numerosi fraintendimenti. Il pensiero computazionale rappresenta davvero un nuovo modo di ragionare o è semplicemente una sistematizzazione di modi di pensare preesistenti? Le generazioni nate e cresciute con il pensiero computazionale ragionano in modo differente rispetto a quelle precedenti? Se il pensiero computazionale è la quarta abilità di base allora tutti dovrebbero imparare a programmare? Il pensiero computazionale si impara scrivendo codici?

Per rispondere a queste domande bisognerebbe prima acquisire consapevolezza che – nonostante l'enfasi di questi ultimi anni – le idee alla base del pensiero computazionale non

nascono con i calcolatori, ma affondano le radici nei modi stessi con cui gli esseri umani si accostano ai problemi e tentano di risolverli.

2. Alle radici del pensiero computazionale

Gli esseri umani applicano il pensiero computazionale da ben prima che esistessero i computer e i linguaggi di programmazione. Tutte le operazioni umane ripetibili e rappresentabili con sequenze di passi che implicano momenti di acquisizione o rilascio di informazioni, valutazioni, decisioni ed esecuzioni sono procedure che possono essere rappresentate da una pluralità di formalismi (diagrammi di flusso, diagrammi ad albero, sequenze procedurali, script, sistemi di regole di produzione, etc.) (Russell & Norvig, 2010). Alzarsi, lavarsi, preparare la colazione, andare a scuola o al lavoro, usare l'automobile, sono azioni della vita quotidiana che compiamo in modo automatico e inconsapevole, con procedure ormai automatizzate da anni di pratica. Solo la violazione della routine provocata dall'insorgere di un problema (ad es. come posso guidare l'automobile con una mano fasciata) ci porta a riflettere sulla natura procedurale di tali azioni e sul fatto che quando le svolgiamo operiamo analisi implicite delle situazioni, scomposizioni in sottoproblemi (ad es. aprire la portiera, allacciare la cintura di sicurezza, avviare il motore, tenere saldamente il volante, etc.), associazioni di sottoproblemi a procedure note, riformulazioni di procedure di fronte a sottoproblemi inediti e riprogettazioni dell'intero processo sulla base di nuove combinazioni delle singole procedure. La differenza con il lavoro del programmatore sta nel fatto che nessuno di noi produce codifiche esplicite delle procedure che mette in atto nelle azioni di vita quotidiana.

Anche l'idea di codificare procedure di pensiero allo scopo di sistematizzarle e di utilizzare i codici prodotti allo scopo di rafforzare le procedure di pensiero stesse non è recente. Già sette secoli orsono, Ramon Llull (italianizzato in Raimondo Lullo), nella sua *Ars generalis ultima* (1305-1308), descrive una tecnica di ricerca basata su un formalismo che mette in corrispondenza simboli (lettere dell'alfabeto) e concetti teologici. Avvalendosi di schemi, figure e regoli rotanti, Lullo ritiene che sia possibile collegare tra di loro un insieme ristretto di concetti fondamentali, al duplice scopo di supportare ragionamenti ed argomentazioni in grado di descrivere verità in ogni campo del sapere e di facilitare la memorizzazione di concetti e asserti di base attraverso una logica meccanica in grado di produrre un sapere enciclopedico universale. Nel suo sistema ogni proposizione è scomponibile in elementi costitutivi e i termini complessi sono riducibili a più termini semplici o principi. In tal modo, definito un insieme limitato di termini semplici di base e combinandoli in tutti i modi possibili si ottengono tutte le proposizioni formulabili. Lullo definisce questo metodo *Ars combinatoria*. Lullo mette poi a punto l'*Ars Magna*, un'arte combinatoria che, servendosi di associazioni simbolo-contenuto, consente di immaginare diversi stati di cose. Lullo stila una lista di sei insiemi di nove entità ciascuno: principi assoluti, principi relativi, questioni, soggetti, vizi, virtù, ciascuno costituito da nove elementi indicati con una lettera dell'alfabeto. Combinando le lettere, il filosofo spagnolo arriva a formulare fino a 1680 modi diversi di rispondere a una questione riguardante i concetti sotto esame. L'*Ars Magna* è quindi utile per ricordare tutti i modi giusti per formulare argomentazioni riguardanti i concetti oggetto di combinazione e per definire regole dei discorsi umani e regole costitutive dell'universo.

Nonostante le critiche, centrate sulla rigidità del sistema e sulla possibilità attraverso il modello combinatorio di generare asserti privi di senso o scarsamente interpretabili alla luce della dottrina, l'ars lulliana incontra grande fortuna nel Rinascimento sia come tecnica

di ausilio alla memorizzazione sia come grammatica universale che permette di cogliere tutte le connessioni possibili tra entità fondamentali e quindi di sviscerare l'ordine del mondo. Giordano Bruno e successivamente Gottfried Leibniz (a cui si ispirò apertamente Charles Babbage) ne rimangono profondamente affascinati e l'opera di Lullo influenza anche il logicismo matematico di Gottlob Frege e Bertrand Russell.

Il sistema lulliano rappresenta una sorta di pensiero computazionale *ante litteram*, dove un sistema analitico interagisce sinergicamente con un sistema simbolico allo scopo di rafforzarsi e di produrre nuove idee. Il problema alla base della concezione lulliana – e anche di alcuni approcci superficiali al pensiero computazionale – è l'identificazione erronea del pensiero con la calcolabilità. Se il pensiero umano fosse riconducibile ad un algoritmo, nessun imprevisto potrebbe essere possibile e non esisterebbero la meraviglia, lo stupore e la capacità di immaginare il non visto precedentemente. Solo il pensiero deduttivo si può trasformare in algoritmo, non quello induttivo o abduttivo, ma sono proprio questi ultimi due ad essere particolarmente importanti per la produzione di idee originali in ogni ambito del sapere.

Nella sua accezione odierna, il termine pensiero computazionale fu utilizzato per la prima volta da Seymour Papert nel 1980 nel testo *Mindstorms*, e successivamente formalizzato da Jeannette Wing nel 2006 che lo ha definito come il processo di pensiero coinvolto nel formulare un problema ed esprimere le sue soluzioni in una forma tale che un *computatore*, umano o meccanico, possa effettivamente processare. Il pensiero computazionale include tre momenti, tra di loro interrelati: uno di concettualizzazione, ossia di comprensione, astrazione, formulazione, riformulazione di problemi e soluzioni; uno di codifica delle concettualizzazioni prodotte in un linguaggio formalizzato intellegibile da un esecutore, non necessariamente informatizzato; uno di esecuzione e valutazione dei codici prodotti che rende evidenti gli effetti delle concettualizzazioni prodotte. I tre momenti sono interrelati perché le possibilità e i modelli di codifica impattano sulle concettualizzazioni producibili e sulle forme in cui i risultati di queste saranno visibili. Si può dire che tra i tre momenti esistano delle affordances (Gibson, 1979), ossia dei rapporti di sinergia e di vincolo che legano le strutture del linguaggio formalizzato ai modelli di pensiero con cui il soggetto analizza il problema di partenza e alle modalità con cui vengono eseguiti e valutati i codici prodotti.

Questa definizione ha alcuni importanti corollari:

1. il pensiero computazionale non prevede che le persone pensino come un computer, ma che pensino *per* il computer, *con il* computer, dato che i computer – almeno quelli odierni – possono solo eseguire, non concettualizzare. Non è l'uomo che deve avvicinare il suo modo di pensare a quello del computer, ma al limite il computer che deve subire un processo di evoluzione per avvicinare i suoi modi di pensare a quelli umani;
2. pensiero computazionale e coding non coincidono. I prodotti del pensiero computazionale non sono necessariamente artefatti (programmi software o righe di codice in linguaggi di programmazione specifici), ma idee, concetti, asserti, procedure, modelli, traducibili in linguaggi più o meno formalizzati, incluso il linguaggio naturale. Nella didattica è quindi possibile svolgere attività legate al pensiero computazionale anche senza l'uso delle macchine, dato che le procedure definite sono procedure di pensiero propriamente dette;
3. i processi di concettualizzazione non differiscono sostanzialmente da quelli utilizzati prima della nascita dei calcolatori elettronici, semplicemente trovano nuove possibilità e sinergie nell'integrazione con le possibilità di questi ultimi.

3. Pensiero computazionale, problem solving e competenza

Nella fase di concettualizzazione, il pensiero computazionale si serve degli strumenti del pensiero analitico. Esso condivide con il pensiero logico-matematico modalità generali di approccio ai problemi e alla loro risoluzione, comunemente identificate con i processi di problem solving.

Si definisce problem solving un processo elaborativo in cui un soggetto opera per trasformare una situazione data in una situazione desiderata, non limitandosi all'applicazione di una procedura ovvia di soluzione o all'esecuzione di una o più procedure routinarie, ma combinando creativamente le risorse di cui dispone e mettendo in gioco aspetti cognitivi, motivazionali e affettivi (Funke, 2010; Klieme, 2004; Mayer, 1990; Mayer & Wittrock, 2006). Il processo è quindi caratterizzato da obiettivi, vincoli, ostacoli, dati di partenza, risorse disponibili, azioni ammissibili e strumenti per metterle in atto. Il problem solving non riguarda problemi di routine risolvibili con procedure già conosciute e automatizzate ma problemi inediti in cui è necessario mobilitare in modo creativo e originale gli strumenti concettuali che si hanno a disposizione, sulla base di una lettura della situazione, dell'applicazione di strategie di soluzione, della riflessione sulle proprie interpretazioni e strategie, ed è quindi strettamente interrelato all'espressione di competenze da parte del soggetto (Trinchero, 2012; 2018), definite dal *Quadro europeo delle qualifiche per l'apprendimento permanente* come "la comprovata capacità di usare conoscenze, abilità e capacità personali, sociali e/o metodologiche, in situazioni di lavoro o di studio e nello sviluppo professionale e/o personale" (EC, 2009, p. 11) e descritte in termini di responsabilità e di autonomia. Essendo un processo interno al sistema cognitivo del soggetto, la competenza non può essere osservata in modo diretto ma solo desunta indirettamente: (i) dai modi, esplicitati e verbalizzati, con cui il soggetto interpreta e assegna significato alle situazioni problematiche che si trova ad affrontare; (ii) dall'osservazione delle azioni che compie per perseguire gli obiettivi legati alla risoluzione del problema; (iii) dalla riflessione esplicita – verbalizzata – che egli è in grado di produrre per giustificare, e se necessario rivedere, le proprie interpretazioni ed azioni nella risoluzione del problema (modello R-I-Z-A, Trinchero, 2012; 2018). Questo processo richiede la rappresentazione e la manipolazione di vari tipi di conoscenze nell'ambito del sistema cognitivo del soggetto che risolve il problema (Mayer & Wittrock, 2006). Pensiero creativo (divergente) e pensiero critico sono quindi componenti importanti della competenza in problem solving (Mayer, 1992): il primo consente di trovare un ampio ventaglio di interpretazioni e tattiche/strategie di soluzione per problemi inediti, mai affrontati prima; il secondo consente di valutare interpretazioni e tattiche/strategie in funzione degli obiettivi prefissati, per stabilirne efficacia, efficienza e necessità di passare ad interpretazioni e tattiche/strategie alternative.

Particolarmente interessanti per modellizzare e descrivere il processo di problem solving coinvolto nell'applicazione del pensiero computazionale appaiono i sottoprocessi identificati nell'*Assessment Framework* di PISA-2012 (OECD, 2013):

- *esplorare e comprendere*. Il primo passo è quello di isolare e comprendere i singoli elementi presenti nel problema e questo richiede un'operazione di scomposizione del problema stesso nelle sue parti costituenti. Nel pensiero computazionale questo richiede che il soggetto identifichi nel problema gli elementi strutturali che potranno essere poi tradotti in procedure informatizzate (Brennan & Resnick, 2012): (i) dati, ossia rappresentazioni simboliche di stati assunti da proprietà di oggetti/soggetti presenti nel mondo reale (ad es. il dato 23 può rappresentare lo stato 23 anni sulla proprietà età del soggetto Giovanna); (ii) azioni e sequenze,

ossia singoli atti, reali o virtuali, o insiemi ordinati di atti, che hanno l'obiettivo di portare a termine un determinato compito (ad es. la sequenza di azioni da compiere per poter prendere l'autobus); (iii) eventi, ossia fatti o accadimenti che coinvolgono oggetti/soggetti del mondo reale (ad es. Giovanna prende l'autobus) o simbolico (ad es. una variabile che assume un certo valore o un click del puntatore su un'area determinata) che possono essere la conseguenza determinate azioni e che possono costituire le condizioni per innescare altri eventi o azioni; (iv) ripetizioni, ossia sequenze rieseguite ciclicamente un certo numero di volte finché perdurano determinate condizioni (ad es. la procedura per prendere l'autobus che va ripetuta tre volte se per andare al lavoro devo prendere tre autobus); (v) parallelismi, ossia eventi, sequenze o ripetizioni che devono accadere contemporaneamente per poter portare a termine un determinato compito (ad es. viaggiare sull'autobus e consultare l'elenco delle fermate per scendere a quella giusta); (vi) condizionali, ossia istruzioni che prevedono che certe azioni siano eseguite solo se si sono verificate determinate condizioni (ad es. prendere l'autobus solo se si è provvisti di biglietto); (vii) *operatori*, ossia simboli che sintetizzano sequenze di azioni volte a generare computazioni e trasformazioni (ad es. calcolare il costo totale del viaggio in autobus per andare al lavoro per una settimana, sapendo che se Giovanna ha 23 anni può usufruire di sconti per i giovani);

- *rappresentare e formulare*. Il secondo passo è quello di selezionare le parti costituenti utili al raggiungimento degli specifici obiettivi di problem solving e di ricombinarle allo scopo di modellizzare il problema. Questo produce un'approssimazione semplificata della realtà che viene espressa mediante una rappresentazione procedurale di momenti di acquisizione o rilascio di informazioni, valutazioni, decisioni ed esecuzioni. Le ipotesi di soluzione del problema verranno prodotte dall'esecuzione delle sequenze modellizzate sui dati a disposizione;
- *pianificare ed eseguire*. Il terzo passo è quello di eseguire le procedure definite dal modello, puntando direttamente agli obiettivi finali o a obiettivi intermedi mediante l'esecuzione di sottoprocedure;
- *monitorare e riflettere*. Il quarto passo prevede il monitorare i progressi fatti verso gli obiettivi intermedi e finali, rilevando eventi imprevisti e, se necessario, intervenire per correggere le procedure, riflettendo sulle soluzioni, considerando il problema da prospettive diverse, valutando criticamente ipotesi e soluzioni alternative, stabilendo l'eventuale necessità di ulteriori informazioni o chiarimenti e comunicando i progressi fatti in modo adeguato. È necessario tenere conto anche del fatto che la situazione problematica può evolvere durante il processo di soluzione, sia per la natura dinamica della situazione stessa sia per l'interazione con il processo risolutivo, questo implica che il processo di problem solving sia ciclico e possa ripartire dall'esplorare e comprendere, procedendo verso la soluzione per raffinamenti successivi.

Stante il processo illustrato, la competenza in problem solving messa in gioco dal pensiero computazionale può essere operazionalizzata secondo i descrittori proposti in Figura 1, organizzati secondo il già citato modello R-I-Z-A.

Interpretazione	Individuare nel problema di partenza dati, azioni e sequenze, eventi, ripetizioni, parallelismi, condizionali, operatori. Individuare regolarità in dati, azioni e sequenze, eventi, ripetizioni, parallelismi, condizionali, operatori che si ripetono. Individuare strutture presenti in dati e situazioni (invarianti, relazioni di complementarietà, scomposizioni). Individuare le quantità presenti in un problema e le relazioni che li legano. Individuare risorse informative pertinenti e non pertinenti utili per la definizione di procedure. Riconoscere idee valide negli approcci degli altri. Individuare punti di forza e punti di debolezza nei prodotti altrui.
Azione	Descrivere a se stessi il problema di partenza. Analizzare il problema suddividendolo in sottoproblemi più semplici. Collegare singoli elementi del sottoproblema al problema complessivo. Rappresentare il problema di partenza in termini di dati, azioni e sequenze, eventi, ripetizioni, parallelismi, condizionali, operatori, utilizzando disegni e diagrammi. Riformulare un problema per ricondurlo a situazioni analoghe già affrontate precedentemente. Organizzare i dati del problema in base a criteri logici. Formulare ipotesi risolutive tenendo conto di dati, vincoli, relazioni e obiettivi. Formulare ipotesi a partire da prospettive alternative. Costruire modelli del problema di partenza (o dei sottoproblemi in cui è stato scomposto) in termini di dati, azioni e sequenze, eventi, ripetizioni, parallelismi, condizionali, operatori. Collegare entità concettuali logico-matematiche a formalismi procedurali. Collegare elementi del modello costruito ad elementi del problema di partenza. Pianificare un percorso per attuare la soluzione formulata. Tradurre le procedure prodotte in codice informatico. Formulare una rappresentazione simbolica delle grandezze individuate. Utilizzare un modello precedentemente validato per codificare una particolare procedura. Riformulare un problema per renderlo compatibile con una codifica già disponibile. Collegare simboli del codice costruito con elementi del problema di partenza. Collegare risultati prodotti dal codice con elementi del problema di partenza. Formulare conclusioni a partire dall'applicazione delle procedure individuate. Confrontare le previsioni con i dati. Spiegare il processo logico che ha portato alla formulazione dell'algoritmo risolutivo. Distinguere ragionamenti logicamente corretti da ragionamenti logicamente errati. Formulare generalizzazioni dei processi di codifica messi in atto.
Autoregolazione	Trovare errori nelle proprie procedure. Giustificare le proprie scelte. Argomentare le proprie conclusioni sulla base dei dati fattuali a disposizione. Difendere le proprie scelte da controargomentazioni che provengono dall'insegnante o dai pari. Criticare codici differenti per stabilirne la maggiore o minore adeguatezza con i propri obiettivi, efficacia, efficienza.

Figura 1. Un possibile profilo di competenza per il problem solving applicabile alle attività di pensiero computazionale in classe.

4. EduLogo. Un gioco online per imparare l'aritmetica e la geometria con il pensiero computazionale

Come è possibile tradurre in pratica i principi descritti lavorando anche con bambini molto piccoli? Il gioco online *EduLogo* (www.edurete.org/EduLogo) nasce proprio per introdurre al pensiero computazionale i bambini già a partire dai primi anni della scuola primaria.

Esso consiste in un percorso di 12 livelli in cui il bambino deve analizzare un problema e risolverlo scrivendo un codice molto simile a quello del linguaggio Logo ideato da Seymour Papert, allo scopo di far compiere ad una tartaruga virtuale un percorso determinato. Grazie a questa caratteristica EduLogo può precedere o accompagnare anche percorsi introduttivi di robotica educativa.

Per ciascuno dei 12 livelli il bambino deve risolvere otto-dieci situazioni problematiche (sfide) tra di loro analoghe che implicano l'uso guidato di diversi elementi del pensiero computazionale (dati, azioni e sequenze, eventi, ripetizioni, parallelismi, condizionali e operatori).

Vediamo alcuni dei livelli. La sfida illustrata in Figura 2 prevede che il bambino identifichi un'azione informatica da compiere, in questo caso un avanzamento della tartaruga, e riconosca un dato illustrato sul righello che indica l'ampiezza dell'avanzamento. Il numero accanto alla tartaruga è il contapassi che indica la distanza percorsa. In Figura 2 viene esplicitato anche il profilo di competenza in termini di descrittori R-I-Z-A (risorse conoscitive che l'allievo deve richiamate, attività interpretative che gli vengono richieste, azioni che egli deve compiere, spunti riflessivi per la propria autoregolazione).

Risorse	I numeri fino a 10
Strutture di interpretazione	Cogliere la necessità di utilizzare i numeri presenti sul righello per stabilire il numero di passi che la tartaruga deve compiere. Riconoscere sul righello il dato necessario.
Strutture di azione	Tradurre azione da compiere (avanzamento) e dato (numero di passi) in un codice opportuno.
Strutture di autoregolazione	Trovare errori nel codice prodotto, se l'esecuzione non dà i risultati sperati. Motivare le proprie scelte all'insegnante e/o ai compagni.

Figura 2. EduLogo: i numeri fino a dieci.

Il gioco proposto lavora sulla concettualizzazione perché induce il bambino ad esplorare e comprendere il problema cogliendo da solo i dati necessari per risolverlo e le strutture da implementare e rappresentare e formulare il codice necessario. Come accennato, in questo caso sono coinvolte le dimensioni dati (il 3) e azioni (l'avanzamento della tartaruga). Il bambino deve quindi pianificare il codice (a 3) ed eseguirlo, monitorando l'esecuzione e riflettendo sul feedback ottenuto. Il vantaggio, rispetto ad ambienti di programmazione più complessi, è che il bambino, anche se molto piccolo (ad es. cinque anni), dopo una prima spiegazione può procedere da solo (guidato dal programma) nella stesura del codice,

acquisendo piena consapevolezza del rapporto tra elementi del problema, codice formulato e azione corrispondente. Ambienti di programmazione più complessi, che ovviamente possono essere utilizzati dopo, prevedono un intervento più marcato dell'insegnante nell'analisi del problema e nella stesura del codice e soprattutto non propongono in modo esplicito una gradualità di passaggi per acquisire le strutture di programmazione necessarie. EduLogo in questo senso è strutturato come un tutorial all'analisi di semplici problemi e alla scrittura di codici opportuni. Si differenza quindi dal classico Logo di Papert per la preminenza della funzione tutoriale su quella esplorativa, pur presente, dato che il bambino deve formulare ipotesi di codice e sperimentarle per giungere alla soluzione del problema.

In Figura 3 è illustrata la sfida successiva, dove il bambino deve cimentarsi con la traduzione in codice di un'azione utilizzando un dato che può assumere valori fino a 100.

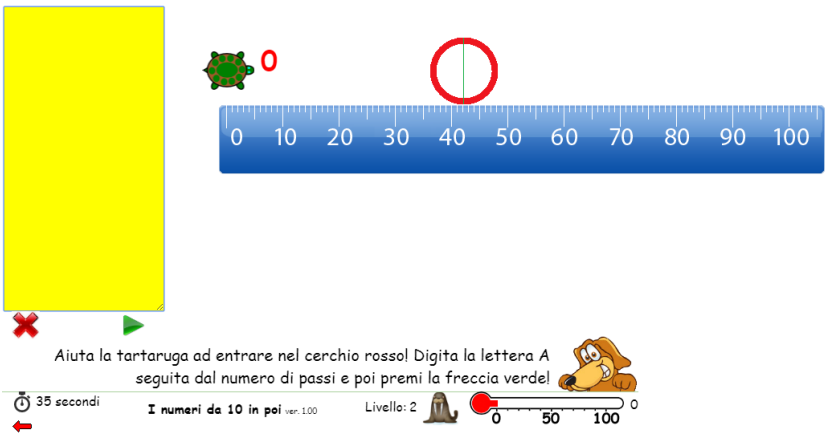

Figura 3. EduLogo: i numeri fino a 100.

La Figura 4 illustra un'altra sfida proposta dal gioco. Il bambino deve scrivere righe di codice per consentire alla tartaruga di raggiungere l'obiettivo rappresentato dall'albero. Questa sfida introduce le sequenze di azioni, composte da avanzamenti e da rotazioni a sinistra e a destra di 90 gradi.

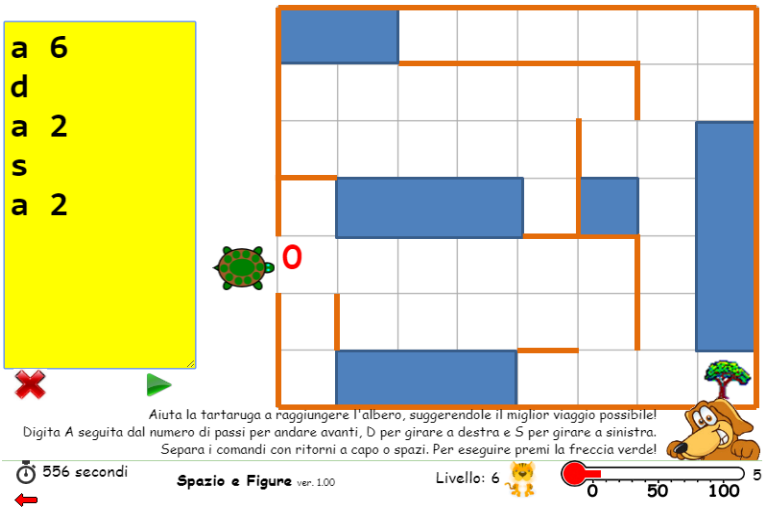

Figura 4. EduLogo: muoversi nello spazio avanti, sinistra, destra.

La Figura 5 illustra una sfida in cui il bambino deve utilizzare il concetto di angolo, anche non pari a 90 gradi.

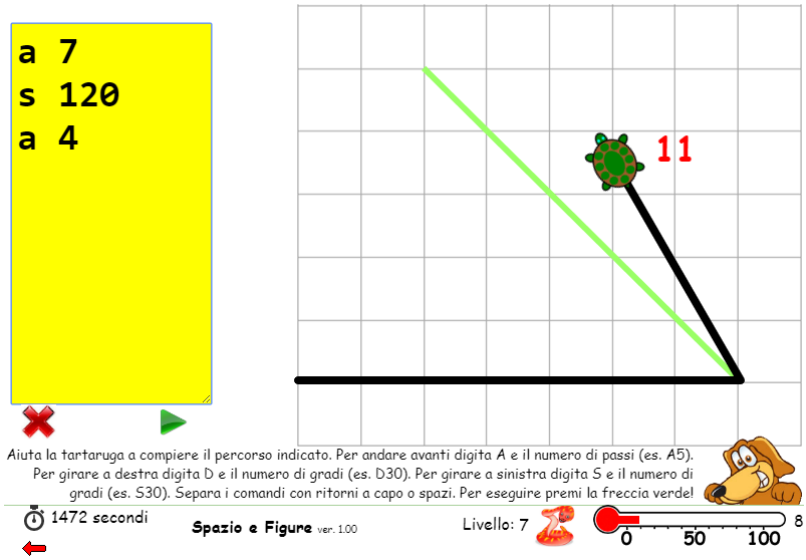

Figura 5. EduLogo: muoversi nello spazio, avanti, sinistra, destra e con angoli variabili.

La Figura 6 illustra una sfida che introduce le ripetizioni, in questo caso allo scopo di disegnare un triangolo sullo schermo come proposto dal modello in scala 1:1.

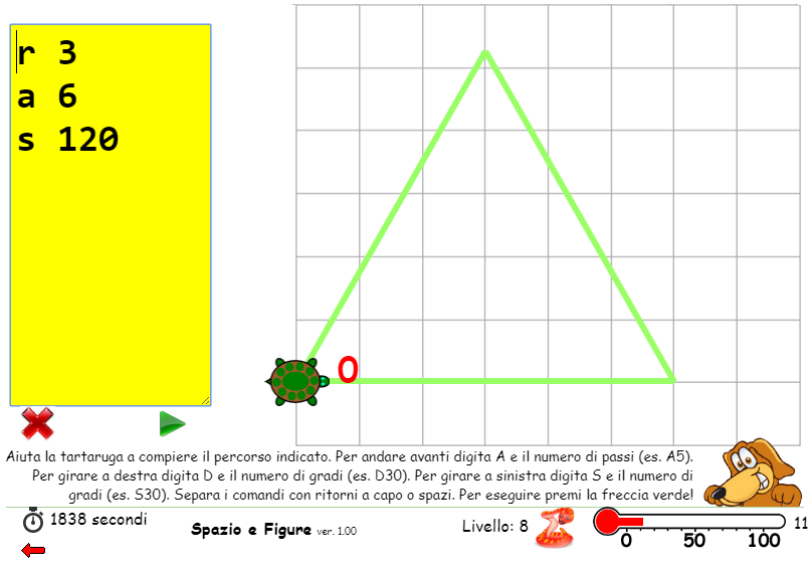

Figura 6. EduLogo: le ripetizioni di sequenze.

Altri esempi di sfide poste dal gioco sono: ideare un codice per riprodurre una figura dato un modello ma in scala differente, ideare un codice per riprodurre una figura data una descrizione, trovare il perimetro di una figura utilizzando il contapassi della tartaruga (parallelismo: muoversi e contare), disegnare una figura di una data forma con un perimetro dato, disegnare una figura di una data forma con un'area data, disegnare più figure sulla stessa videata alzando e abbassando la penna della tartaruga nei momenti opportuni,

compiere azioni in presenza di condizioni date (condizionali), definire ed utilizzare operatori che riassumono sequenze di istruzioni date (ad es. l'operatore *quadrato* che disegna un quadrato nella posizione corrente della tartaruga). Le varie sfide stimolano quindi il bambino a riconoscere, manipolare e sperimentare le varie tipologie di elementi strutturali identificabili in un problema e riproducibili con una procedura informatizzata (Brennan & Resnick, 2012).

L'idea alla base delle sfide è sempre quella di proporre situazioni problematiche che il bambino debba analizzare, guidato dal software, dagli insegnanti e dai propri pari, e tradurre in codici opportuni, anche attraverso tentativi, errori, riflessioni sull'errore e aggiustamenti successivi. Per ciascuna sfida il bambino deve costruire e sperimentare un codice che risolva il problema proposto, vedendone gli effetti sui movimenti della tartaruga. Le sfide vengono presentate in una sequenza tale da far sì che il bambino non possa cimentarsi con sfide di livello più alto se prima non ha dimostrato adeguata padronanza di sfide di livello più basso e dei concetti che le caratterizzano. È il gioco stesso a controllare la correttezza del codice costruito e a far avanzare il bambino alla sfida successiva. Durante le sperimentazioni preliminari è stato chiesto ai ragazzi di affrontare le sfide in coppia e di aiutarsi anche attraverso movimenti nello spazio fisico, in modo da poter concretizzare meglio l'effetto del codice astratto.

In sintesi, il software propone un percorso di coding guidato, pensato per sviluppare la competenza di problem solving servendosi degli strumenti del pensiero computazionale. I concetti toccati sono quelli dell'aritmetica e della geometria: lunghezze, segmenti, angoli, figure geometriche, perimetri, aree.

5. Conclusioni: come educare al pensiero computazionale?

Le considerazioni illustrate precedentemente e gli esempi descritti ci portano a concludere che il pensiero computazionale può essere visto come una forma sistematizzata di problem solving, a cui il computer (o l'interazione con robot programmabili) aggiunge la visibilità e sperimentabilità degli effetti delle proprie ipotesi risolutive. Il pensiero computazionale prevede l'esercizio di competenze che si situano su due piani: uno è di dominio dell'analista, ossia della figura professionale deputata ad analizzare, scomporre, riformulare, rendere risolvibili i problemi, quindi un esperto in problem solving propriamente detto; il secondo è di dominio del programmatore, ossia della figura professionale deputata a trasformare in codici i risultati prodotti dall'analista, testandoli e mettendoli a punto per produrre software specifici, quindi un esperto in sistemi e codici. Insegnare il pensiero computazionale a scuola non dovrebbe quindi puntare a creare dei programmatori, ossia degli esperti in codici, ma sicuramente dei buoni analisti, in grado di utilizzare in primis le possibilità offerte dal pensiero logico-matematico, anche in sinergia con quelle offerte dalla macchina. Insegnare uno specifico linguaggio di programmazione (coding) senza insegnare la concettualizzazione non sviluppa il pensiero computazionale e non sviluppa il problem solving: tra capacità di concettualizzazione e capacità di codifica non esiste nessuna relazione logicamente necessaria (Duncan, Bell & Atlas, 2017). L'attenzione dovrebbe quindi cadere sul momento di concettualizzazione, più che sui momenti di codifica e di esecuzione/verifica, privilegiando la costruzione di modelli di pensiero più che di artefatti software. Il riferimento dovrebbe essere l'ampio quadro delineato dalle *Indicazioni nazionali* nell'ottica di sviluppare negli allievi un ventaglio di competenze legate alla comprensione, applicazione, analisi, valutazione e produzione di concetti e asserti relativi a situazioni problematiche via via sempre più complesse (Di Sessa,

2001), utilizzando i modelli di pensiero tipici dell'analista informatico e sfruttando le possibilità offerte dalla macchina come mezzo di feedback.

Partire dalla concettualizzazione anziché dalla codifica ha effetti immediati anche sulle altre discipline di studio e aiuta a superare un problema ben noto agli informatici: bambini formati all'uso di linguaggi di programmazione prevalentemente visuali, senza un percorso parallelo di formazione al problem solving, possono trovarsi in difficoltà nell'utilizzare altri tipi di linguaggi, proprio perché hanno sviluppato strutture di pensiero strettamente legate alle sintassi di un linguaggio specifico, anziché strutture generali legati all'analisi di problemi. Meglio quindi, almeno in una prima fase, sviluppare il pensiero computazionale in maniera indipendente da uno specifico linguaggio di programmazione (o con linguaggi essenziali, di cui il Logo è appunto un esempio), focalizzandosi sulle strutture logiche fondamentali (dati, azioni e sequenze, eventi, ripetizioni, parallelismi, condizionali, operatori) che accomunano problem solving e pensiero computazionale e che consentono di dare senso a tutta una serie di strutture di programmazione presenti in linguaggi specifici, primo passo per poterle padroneggiare adeguatamente in futuro.

Bibliografia

Bellanca, J., & Brandt, R. (2010). *21st century skills: rethinking how students learn*. Bloomington: Solution Tree Press.

Brennan, K., & Resnick, M. (2012). *Using artifact-based interviews to study the development of computational thinking in interactive media design*. Paper presented at annual American Educational Research Association meeting. Vancouver, BC. http://scratched.gse.harvard.edu/ct/files/AERA2012.pdf (ver. 15.04.2019).

Bundy, A. (2007). Computational thinking is pervasive. *Journal of Scientific and Practical Computing Noted Reviews*, *1*(2), 67–69.

Clements, B.H., & Nastasi, D.K. (1999). Metacognition, learning, and educational computer environments. *Information Technology in Childhood Education Annual*, *1*, 5–38.

Di Sessa, A. (2001). *Changing minds: computers, learning and literacy*. Cambridge, MA: Mit Press.

Duncan, C., Bell, T., & Atlas, J. (2017). What do the teachers think?: Introducing computational thinking in the primary school curriculum. In *Proceedings of the Nineteenth Australasian Computing Education Conference* (pp. 65-74). New York, NY: ACM.

EduLogo. www.edurete.org/EduLogo (ver. 15.04.2019).

European Commission. (2009). *Quadro europeo delle qualifiche per l'apprendimento permanente* (EQF). Lussemburgo: Commissione Europea.

Fessakis, G., Gouli, E., & Mavroudi, E. (2013). Problem solving by 5-6 years old kindergarten children in a computer programming environment: a case study. *Computers & Education*, *63*, 87–97.

Funke, J. (2010). Complex problem solving: A case for complex cognition?. *Cognitive Processing*, *11*, 133–142.

Gibson, J.J. (1979). *The ecological approach to visual perception*. Boston, MA: Houghton Mifflin.

Klieme, E. (2004). Assessment of cross-curricular problem-solving competencies. In J.H. Moskowitz & M. Stephens (eds.), *Comparing learning outcomes. International assessments and education policy* (pp. 81-107). London: Routledge Falmer.

Lullo, R. (1986). *Ars generalis ultima*. Turnhout: Prebols Publishers (Original work published 1303-1308).

Mayer, R.E. (1990). Problem solving. In M.W. Eysenck (ed.), *The Blackwell dictionary of cognitive psychology* (pp. 284-288). Oxford: Basil Blackwell.

Mayer, R.E. (1992). *Thinking, problem solving, cognition* (2nd ed.). New York, NY: Freeman.

Mayer, R.E., & Wittrock, M.C. (2006). Problem Solving. In P.A. Alexander & P.H. Winne (eds.), *Handbook of educational psychology* (2nd ed.) (pp. 287-303). Mahwah, NJ: Lawrence Erlbaum Associates.

MIUR. Ministero dell'Istruzione dell'Università e della Ricerca (2012). *Indicazioni nazionali per il curricolo della scuola scuola dell'infanzia e del primo ciclo di istruzione*. No. Speciale.

MIUR. Ministero dell'Istruzione dell'Università e della Ricerca (2018). Nota prot. n. 3645 del 1 marzo 2018. *Indicazioni nazionali e nuovi scenari*.

OECD. Organization for Economic Co-operation and Development (2013). *PISA 2012 Assessment and analytical framework: mathematics, reading, science, problem solving and financial literacy*. Paris: OECD Publishing.

Papert, S. (1980). *Mindstorms: children, computers, and powerful ideas*. New York, NY: Basic Books.

Russell, S., & Norvig, P. (2010). *Intelligenza artificiale: un approccio moderno*. Milano-Torino: Pearson Prentice Hall.

Sin, K., & Muthu, L. (2015). Application of big data in education data mining and learning analytics - A literature review. *ICTACT Journal on Soft Computing, 5*(4), 1035.

Trilling, B., & Fadel, C. (2009). *21st century skills: learning for life in our times*. San Francisco, CA: Wiley & Sons.

Trinchero, R. (2012). *Costruire, valutare, certificare competenze. Proposte di attività per la scuola*. Milano: FrancoAngeli.

Trinchero, R. (2018). *Costruire e certificare competenze con il curricolo verticale nel primo ciclo*. Milano: Rizzoli Education.

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

Wing, J.M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, *366*(1881), 3717–3725.