ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA
Corso di Laurea in Ingegneria e Scienze Informatiche

QUI IL TITOLO DELLA TESI

Elaborato in
SISTEMI EMBEDDED E INTERNET OF THINGS

Relatore
Prof. ALESSANDRO RICCI

Presentata da
ANGELA SPERANZA

Corelatore
Prof. ELVIS MAZZOMI
Dott.ssa YLENIA BATTISTINI

Anno Accademico 2022 – 2023

eventuale dedica o citazione

Indice

Introduzione

Qui il testo dell'introduzione alla tesi. Generalmente l'introduzione non dovrebbe superare le 2/3 pagine e dovrebbe essere scritta solo alla fine.

Capitolo 1

Background e Stato dell'Arte

1.1 Educazione digitale

In una società come quella odierna, ormai completamente immersa nella tecnologia, è impensabile che anche il mondo dell'istruzione non ne sia fortemente influenzato. Gli stessi studenti, che al giorno d'oggi sono tutti nativi digitali, stanno crescendo accompagnati da dispositivi tecnologici sempre più avanzati e da un'ampia varietà di applicazioni e servizi digitali. In questo contesto, si ritiene di fondamentale importanza che la scuola, il centro di gravità della vita di ogni studente, sia in grado di fornire ai giovani tutti gli strumenti che servono non soltanto a comprendere il mondo digitale che hanno tra le mani, ma anche a saperne fare un uso responsabile nonché fruttuoso per quanto riguarda la propria crescita personale e professionale.

L'educazione digitale, dunque, non può più essere considerata un'attività opzionale da parte delle scuole e dei docenti, ma piuttosto un necessario investimento per fare in modo che le nuove generazioni siano in grado di sfruttare al massimo le opportunità che la tecnologia può offrire. Non solo: in realtà, l'educazione digitale è un argomento che copre una vasta gamma di ambiti di applicazione. Si pensi soltanto alla sicurezza online e alla privacy: gli utenti di Internet devono essere informati sulle minacce digitali e imparare come proteggere se stessi e i propri dati sensibili. In secondo luogo, l'educazione digitale si estende alla promozione di un comportamento etico e responsabile online, incoraggiando l'empatia, il rispetto e la tolleranza nelle interazioni sociali. Per quanto riguarda, invece, lo sviluppo di competenze e *soft skills* ormai indispensabili nella vita personale e lavorativa di ogni individuo, è importante che sin dagli anni scolastici venga promosso e incentivato il pensiero critico, il *problem solving* e la creatività, incoraggiando gli individui a utilizzare gli strumenti digitali per esprimere idee, creare contenuti originali e collaborare online. Questa visione educativa non solo promuove lo sviluppo delle com-

petenze digitali, ma anche una partecipazione attiva nella società digitale in continua evoluzione.

1.2 Pensiero computazionale

Con l'espressione "pensiero computazionale" si fa riferimento ad un concetto che negli ultimi decenni ha guadagnato un'importanza sempre maggiore nell'ambito dell'educazione digitale. Questo termine si compone di due parole chiave: il primo, "pensiero", si riferisce alla caratteristica che rende l'essere umano unico rispetto a tutti gli altri esseri viventi, ovvero la capacità di ragionare, analizzare dati e situazioni e risolvere problemi. Il secondo termine, "computazionale", deriva dall'inglese *to compute*, che può essere tradotto come "calcolare". Questi due termini, insieme, definiscono un concetto che molti, al giorno d'oggi, ritengono sia considerabile come la quarta abilità fondamentale per l'apprendimento, insieme alla lettura, alla scrittura e all'aritmetica: **pensare come un computer**. Tuttavia, questa speciale competenza non nasce in concomitanza con l'utilizzo del computer, ma si tratta di processi mentali molto più antichi e sottili che non riguardano soltanto il mondo dell'informatica, ma possono essere applicati a qualsiasi contesto della vita quotidiana.

Seymour Papert

If children really want to learn something, and have the opportunity to learn it in use, they do so even if the teaching is poor. For example, many learn difficult video games with no professional teaching at all! Others use Nintendo's system of telephone hot lines or read magazines on strategies for games to find the kind of advice for video games that they would get from a teacher if this were a school subject.

Se i bambini vogliono davvero imparare qualcosa e hanno l'opportunità di imparare facendo, loro lo fanno anche se l'insegnamento è di scarsa qualità. Per esempio, molti imparano videogiochi difficili senza alcun insegnamento professionale! Altri utilizzano il sistema di Nintendo di linee telefoniche o leggono riviste sulle strategie per i videogiochi, il tipo di consigli che avrebbero ottenuto da un insegnante se questa fosse una materia scolastica.

[1]

[1]Seymour Papert, *The Children's Machine: Rethinking School in the Age of the Computer*, 1993.

Il primo ad avere utilizzato il termine "Pensiero computazionale" fu **Seymour Papert**, matematico, informatico e pedagogista sudafricano naturalizzato statunitense. Papert può essere considerato uno dei padri dell'informatica educativa, ovvero di quella branca dell'informatica che si occupa di studiare e sviluppare metodi e strumenti per l'insegnamento e l'apprendimento in un nuovo contesto digitale.

Papert fu allievo di Jean Piaget – psicologo, biologo, pedagogista e filosofo svizzero – nonché il fondatore dell'epistemologia genetica, un settore della psicologia che si occupa dello studio dei processi mentali e cognitivi messi in atto dal cervello umano al momento dell'apprendimento di nuove conoscenze. In particolare, Piaget viene ricordato per le sue teorie costruttiviste, fortemente contrapposte ad un'altra tipologia di approccio didattico, l'istruzionismo.

La **filosofia istruzionista** si basa sulla convinzione che la conoscenza e le competenze devono essere trasmesse dall'insegnante allo studente, il quale assume una posizione piuttosto passiva in un contesto dove altri strumenti – ad esempio il computer – non sono altro che un semplice supporto per l'apprendimento.

La **teoria costruttivista** di Piaget, al contrario, si incentra soprattutto su una visione di scuola dove i discenti sono attori attivi e partecipativi nel loro processo di apprendimento, poiché costruiscono e raffinano in modo autonomo e incrementale le proprie conoscenze e competenze. In particolare Papert apprezzò moltissimo il lavoro di Piaget sul collegamento intrinseco che, secondo Piaget, esiste tra l'intelligenza – o, per meglio dire, capacità cognitiva – e l'abilità dell'individuo di adattarsi efficacemente all'ambiente sociale e fisico in cui si trova. In particolar modo, Papert si dedicò a consolidare il costruttivismo di Piaget in una sua versione più specifica, il **costruzionismo**.

Papert, dall'incontro con le teorie di Piaget, comprese che il processo di appredimento può essere esplicato come la costruzione mentale di rappresentazioni funzionali del sistema con il quale si interagisce: ciò si applica sia al bambino che all'adulto. L'**esperienza empirica** aiuta l'essere umano a costruire materiali concreti di riferimento da "consultare" in relazione alla realtà esterna: questo, per giunta, avviene in modo particolarmente efficace e soddifacente quando vi è anche una realizzazione pratica e concreta di questi concetti. La teoria costruzionista di Papert pone l'accento non solo sul ruolo attivo dello studente, ma anche sul fatto che la "costruzione" divenga ancora più efficace in un contesto in cui l'individuo produce un risultato per lui ricco di significato e di utilità.

In questo modo, come si legge nel libro "*Mindstorms: Children, Computers, and Powerful Ideas*", Seymour Papert propose un nuovo modello educativo in cui strumenti come il computer non sono più visti come "contenitori di informazioni" da cui attingere, ma come strumenti per costruire e manipolare

le proprie idee e la propria fantasia. Inoltre, in un altro suo scritto del 1993 "*Children's Machine – Rethinking school in the age of the computer* ", Papert citò il famoso proverbio africano "Dai un pesce ad una persona, e quella persona sarà sazia per un giorno, insegna ad una persona a pescare, e quella persona si sazierà per tutta la vita.". Il Costruzionismo si basa proprio sull'idea che il bambino impari in modo migliore "pescando" in autonomia le competenze e le informazioni di cui ha bisogno: l'istituzione scolastica ha il compito di supportarlo moralmente, psicologicamente, materialmente e intellettualmente nei suoi sforzi.

In particolare, Papert capì che, specialmente per quanto riguararda i giovani studenti, la **programmazione informatica** può essere l'attività ideale per sviluppare questo tipo di competenze. Lo studioso, infatti, era convinto che il focus di questo tipo di attività dovesse essere il ragionamento e la costruzione di un algoritmo che risolva un problema, insieme alla capacità di riconoscere e correggere gli errori. In generale, ritenne poi necessario compiere delle operazioni di semplificazione di alcune istruzioni e concetti informatici, i quali potevano risultare troppo complessi ma anche non necessari allo scopo.

Secondo tali premesse, Papert sviluppò un proprio linguaggio, **LOGO** - il cui nome deriva dal greco antico ("parola" oppure "pensiero") - , un ambiente di programmazione visuale che permette di controllare una tartaruga, scrivendo codice e imparando a padroneggiare i concetti di base dell'informatica in modo semplice e divertente. L'ambiente di questo linguaggio, fortemente grafico ma allo stesso tempo semplice, è adatto a bambini di tutte le età, permettendo loro non solo di imparare a programmare, ma anche di sfruttare al massimo la loro **creatività e fantasia**. LOGO, inoltre, fu costruito in modo tale da permettere di costruire quei procedimenti cognitivi che permettono di individuare errori e inconvenienti e di imparare a costruire un ragionamento critico che aiuti a far fronte a questo tipo di situazioni, accettando tutte le soluzioni come possibili e non etichettandone alcune come sbagliate a priori.

Tra le caratteristiche più importanti di LOGO, inoltre, vi è la possibilità di utilizzarlo non soltanto come supporto alle materie matematiche o scientifiche, ma anche come **strumento interdisciplinare** per la creazione di attività di carattere artistico e umanistico. Il computer diventa in questo modo strumento di espressione e di costruzione di conoscenza, e non solo un mezzo per l'apprendimento di nozioni preconfezionate: inoltre, grazie all'elaboratore, lo studente può vedere il risultato concreto delle proprie conoscenze tradotte in algoritmi e programmi. Come anticipato, LOGO permette di controllare una tartaruga. La tartaruga in questione non è altro che un cursore grafico che, quando si muove, lascia il segno del proprio percorso dietro di sé grazie ad una penna: in questo modo, muovendosi, la tartaruga può disegnare figure geometriche di ogni tipo. La "geometria della tartaruga" è un'area didattica creata da

Seymour Papert e i suoi colleghi proprio per rendere più semplice e divertente l'apprendimento, in questo caso, della matematica e della geometria. I comandi base della geometria della tartaruga sono molto semplici: Forward (avanti); Back (indietro); Right (destra); Left (sinistra); Penup (penna su, ovvero non appoggiata sul foglio), Pendown (penna sul foglio). A partire da questi semplici comandi si possono ideare problemi e sfide di ogni tipo e di ogni grado di difficoltà, mettendo in campo conoscenze pregresse sulla materia e acquisendone di nuove. In ogni caso, LOGO, come tanti altri strumenti di programmazione visuale e non solo, costituisce una preziosa risorsa sia per gli studenti che per gli insegnanti, poiché promette un **nuovo modo di imparare**, sicuramente più attivo e coinvolgente, ma anche più efficace e duraturo.

Jeannette Wing

Il concetto di "pensiero computazionale" è stato poi ripreso e ampliato da **Jeannette Wing**, informatica teorica e attuale vicepresidentessa di Microsoft Research.

Nel 2006, Wing pubblicò un articolo intitolato "*Computational Thinking*" in cui definì la propria visione di pensiero computazionale; nell'articolo in questione, Wing parla di pensiero computazionale come "un approccio universalmente applicabile e un set di skills che tutti noi dovremmo imparare e usare, non soltanto gli informatici". Nello stesso scritto, fu proposto proprio questo concetto come la quarta abilità fondamentale da insegnare nelle scuole, insieme alla lettura, alla scrittura e all'aritmetica.

Infatti, secondo Wing, l'informatica non offre soltanto tecnologie straordinarie, ma anche un **framework intellettuale** per un nuovo modo di pensare, utile a tutti. Wing scrisse: "Il pensiero computazionale coinvolge la risoluzione di problemi, la progettazione di sistemi e la comprensione del comportamento umano, attingendo dai concetti fondamentali del mondo dell'informatica.[...] Il pensiero computazionale è riformulare un problema che sembra difficile in uno che sappiamo già come risolvere, ad esempio attraverso processi di riduzione, incorporamento, trasformazione o simulazione". *Divide et impera*: ecco la prima competenza fondamentale che il pensare in modo computazionale insegna. La **decomposizione di un problema in sottoproblemi più piccoli**, più facili da risolvere, aiuta ad esercitare un certo grado di controllo e ordine sul problema originale, e contemporaneamente, lo rende più gestibile e più facilmente comprensibile. Sulla stessa linea, nasce anche la **capacità di riconoscere e utilizzare pattern**, ovvero soluzioni a problemi già risolti in passato che possono essere riutilizzate in nuovi contesti. Ma per poter risolvere un problema in modo efficiente, è anche necessario **saper astrarre e generalizzare**, analizzando in modo critico i dati e le informazioni a disposizione,

eliminando quelle superflue e concentrandosi su quelle essenziali. Infine, Wing invita a **pensare in modo algoritmico**, ovvero a sviluppare una sequenza di passi logici e ben definiti che, se seguiti correttamente, portano alla soluzione del problema. Si alternano, dunque, momenti in cui si riflette sul problema, si analizzano i dati e si progettano soluzioni, applicando conoscenze pregresse e nuove, a momenti in cui si mette in pratica il frutto di questo lavoro mentale, eventualmente scrivendo codice e testando la soluzione proposta, correggendo eventuali errori. Pensare in modo computazionale, dunque, non ha solo a che vedere con l'essere in grado di programmare un computer: significa anche essere in grado di **pensare a più livelli di astrazione**. Jeannette Wing è convinta, come molti esperti del settore, che il pensiero computazionale non sia una competenza riservata soltanto agli informatici e alle persone di scienza: insegnarlo a scuola non vuol dire necessariamente formare futuri programmatori o ingegneri, ma cittadini più consapevoli e competenti in un mondo sempre più tecnologico e complesso. In questo mondo, è la macchina a dover pensare come l'uomo, e non il contrario: l'essere umano è creativo e intelligente, la macchina è passiva; è l'utente a dover rendere la macchina interessante per poterla utilizzare al massimo delle sue potenzialità.

1.3 Il Micromondo

Fu Seymour Papert 1.2 per primo ad introdurre il concetto di "micromondo" nel contesto informatico ed educativo. Il micromondo di Papert nacque contestualmente all'introduzione del suo linguaggio di programmazione, LOGO, e della relativa protagonista, la tartaruga. L'ambiente di apprendimento della tartaruga fu pensato e realizzato per permettere ai bambini di ragionare su concetti e sfide di carattere matematico e geometrico: non a caso, Papert lo aveva soprannominato "provincia di Mathland", come ad indicare un piccolo mondo, confinato e protetto, costruito ad hoc con un obiettivo ben preciso, quello di fare scuola in modo assolutamente innovativo.

Secondo Papert, infatti, il micromondo nasce come un ambiente circoscritto, in cui è possibile sperimentare e coltivare le proprie idee con un alto grado di libertà e autonomia, acquisendo conoscenze in modo naturale e graduale, proprio come avviene nei primi anni di vita dell'essere umano. Generalmente, la costruzione di micromondi avviene attraverso una serie di fasi e principi ben definiti, che contribuiscono a rendere l'ambiente il più efficace e stimolante possibile.

In primo luogo, come già accennato, i micromondi si basano sempre su un tema ben preciso: il dominio in cui si svolge l'attività deve fare in modo di spostare il focus dell'attenzione degli studenti sul mettere in pratica le co-

noscenze acquisite e i loro personali modi di pensare. Naturalmente, il tema scelto deve essere non solo facilmente comprensibile da chi lo programma, ma deve lasciare spazio alla creatività e alla fantasia. Papert stesso scrisse: "*Low floors, high ceilings*". *Low floors*, traducibile come "pavimenti bassi", richiama la facilità di accesso e di comprensione che ogni micromondo deve garantire: l'ambiente deve essere alla portata di tutti, senza discriminazioni di alcun tipo. *High ceilings*, ovvero "alti soffitti", invece, si riferisce alla possibilità di oltrepassare i limiti imposti dall'ambiente stesso e di dare spazio alla componente fantasiosa e creativa di ogni studente; in tal modo le possibilità di realizzazione del micromondo sono pressoché infinite. Successivamente l'espressione venne ampliata da Mitchel Resnick, uno degli allievi di Papert e attualmente professore presso il MIT di Boston, con l'aggiunta di "*wide walls*", ovvero "mura larghe". Resnick si riferiva alla possibilità di estendere il micromondo includendo concetti e sfide sempre nuove e attuali, facendo in modo di poter continuare a raccogliere contributi e idee da parte di tutti coloro che hanno accesso ad esso.

In questo modo, si rende possibile la creazione di contesti didattici stimolanti e immersivi, dove **l'aspetto ludico e quello educativo si fondono in modo armonioso e naturale**, permettendo agli studenti di sentirsi personalmente coinvolti e motivati.

Tutto ciò, nondimeno, non sarebbe possibile senza il coinvolgimento attivo degli insegnanti, che devono essere in grado sia di gettare le basi per la costruzione del micromondo, sia di supportare e guidare l'alunno nel suo percorso di apprendimento, che risulta significativamente più autonomo e personale rispetto a quello tradizionale. Il docente, in questo nuovo modo di fare scuola, assume il ruolo di guida e di facilitatore, piuttosto che di dispensatore di nozioni e conoscenze: deve necessariamente lasciare spazio a tutte le idee degli studenti, accettando e valorizzando anche il ruolo degli "**errori**" come parte integrante del processo di apprendimento.

Non solo: gli studenti possono sperimentare un approccio più attivo e partecipativo rispetto alla didattica frontale. Il micromondo, infatti, è un ottima occasione di *learn by doing*, in cui i discenti sperimentano e costruiscono per pezzi, in modo incrementale, le loro idee e i loro ragionamenti, producendo un risultato significativo e soprattutto tangibile. In particolare, i risultati più importanti di questo metodo educativo si ritrovano nello sviluppo di una lunga serie di **competenze trasversali**, o *soft skills*. Da una maggiore capacità di *problem-solving* e pensiero critico, a una più sviluppata autonomia e responsabilità, non dimenticando un importante sviluppo nella capacità di collaborazione e di comunicazione se si lavora in gruppo, il micromondo si rivela un'ottima occasione di crescita e di apprendimento per gli studenti di ogni età.

1.4 Il *Coding*

I concetti emersi nei paragrafi precedenti aprono le porte a un'ulteriore riflessione, quella riguardante gli strumenti e le metodologie corrette per trasmettere la *forma mentis* del pensiero computazionale. L'informatica possiede una dignità intrinseca quale disciplina scientifica fondamentale, caratterizzata da un corpus consolidato di concetti e metodologie indipendenti. Inoltre, riveste un valore significativo anche come **disciplina trasversale**, offrendo un approccio che contribuisce a una più profonda comprensione delle altre discipline. Le "Indicazioni nazionali per il curricolo della scuola dell'infanzia e del primo ciclo d'istruzione[1]", pubblicate dal MIUR nel 2012, sottolineano come la scuola italiana debba evolversi, permettendo alle nuove generazioni di sfruttare le competenze base – che, in passato, erano l'obiettivo primo del processo di alfabetizzazione – come strumenti per una miglior comprensione del contesto sociale in cui è inserito lo studente.

Ken Robinson, nella prefazione allo scritto "*Lifelong Kindergarten - Cultivating Creativity through Projects, Passion, Peers, and Play*" di Mitchel Resnick, riflette sul fatto che la tecnologia ha sempre assistito l'uomo nell'espansione del proprio corpo e della propria mente. La capacità umana di **astrazione e immaginazione** è distintiva e fondamentale per l'evoluzione della società e della cultura. Tuttavia, per poter alimentare queste qualità, è necessario disporre di strumenti e materiali adatti a tale scopo. Poiché l'informatica consente di costruire rappresentazioni e soluzioni per situazioni anche fisicamente irrealizzabili, limitate solo dalla nostra immaginazione, essa si presenta come un ambiente potente, utile per esercitare la creatività, assumendo anche un elevato valore educativo. In un contesto in cui le informazioni sono facilmente accessibili e i computer sono sempre più abili nel risolvere compiti noiosi e ripetitivi, diventa ancor più cruciale educare gli studenti a trovare **soluzioni innovative**, anziché concentrarsi sull'applicazione meccanica di procedure mnemoniche o standardizzate. Resnick, a tal proposito, sottolinea che il concetto di creatività viene associato principalmente all'espressione artistica o alla "Creatività con la C maiuscola" di grandi inventori e artisti le cui opere decorano i musei. La creatività, però, non appartiene solo agli artisti, ma anche agli scienziati che fanno scoperte, ai medici che diagnosticano malattie o agli imprenditori che sviluppano nuovi prodotti o strategie di mercato. In modo più generale, la creatività entra in gioco quando si generano idee utili per sé stessi o per gli altri nella vita quotidiana, non necessariamente "rivoluzionarie" a livello globale, ma sicuramente significative a livello personale. In altre parole, si tratta di una qualità alla portata di tutti, purché sia coltivata fin dalla giovane età, ancora meglio se nei contesti educativi e scolastici.

[1] `gne`

Se si pensa al mondo dell'informatica, dunque, il *coding*, ovvero la scrittura di codice, si configura come uno strumento di straordinaria potenza nell'ambito dell'educazione digitale e nello sviluppo del pensiero computazionale: si tratta di un profondo tessuto di competenze che vanno ben oltre la mera scrittura di algoritmi e istruzioni.

In primo luogo, la programmazione agisce come **catalizzatore del pensiero critico**, spingendo gli studenti a frazionare intricati problemi in componenti più gestibili, fornendo loro un approccio strutturato nella risoluzione delle sfide. Quando inserito in contesti di piccoli gruppi di lavoro, il *coding* favorisce **la collaborazione e il sinergico lavoro di squadra**, sottolineando l'importanza della condivisione di idee e della cooperazione per raggiungere obiettivi comuni. La **componente creativa** del *coding* emerge in modo preminente: risolvere un problema non richiede solo la padronanza di conoscenze pregresse, ma anche la capacità di **immaginare e progettare soluzioni**, valutandone attivamente i pro e i contro e selezionando la strategia più idonea. In questo contesto, il *coding* diventa un veicolo per l'espressione personale e il confronto con diverse prospettive, aprendo a un ventaglio infinito di possibilità creative. Infine, al di là delle competenze tecniche specifiche, il *coding* prepara gli studenti per un futuro digitale in continua evoluzione, potenziando la loro **alfabetizzazione tecnologica** e aprendo porte a innumerevoli opportunità nel vasto campo della tecnologia. In sintesi, il *coding* non solo istruisce sul linguaggio della programmazione, ma plasma **menti agili, creative e collaborative**, pronte a navigare le sfide della società digitale moderna.

Tuttavia, l'introduzione delle discipline informatiche nel contesto educativo non è un'operazione immediata né tantomeno semplice. Al giorno d'oggi vi sono ancora pregiudizi e critiche che circondano l'utilizzo - spesso eccessivo e improprio - della tecnologia da parte dei giovani.

Una possibile sfida, ad esempio, riguarda l'associazione concettuale che vede l'informatica come un'attività principalmente giocosa e ludica: se da un lato è vero che il *coding* può essere un'attività divertente e stimolante, dall'altro è importante sottolineare che la programmazione è una disciplina seria e complessa, che richiede molto impegno e dedizione. L'aspetto ludico che il *coding* può suscitare in chi lo utilizza non deve essere confuso con la sua reale natura, ovvero, come già detto, quella di uno strumento potente e versatile per lo sviluppo di competenze trasversali e per l'acquisizione di un pensiero critico e creativo.

Un altro possibile ostacolo, diametralmente opposto per quanto riguarda ciò di cui si è parlato poc'anzi, risiede nei luoghi comuni e nei pregiudizi che ancora oggi circondano la disciplina. Idee sbagliate e stereotipate vedono attività quali programmare e ragionare da informatici come appannaggio di poche persone che hanno quello che gli americani chiamano il "gene del nerd" (*Geek*

gene) spesso associato a personalità asociali e competitive e, più in generale, ad un target prevalentemente maschile. Proprio a causa di queste idee, gli esperti del settore tendono a sottolineare in modo deciso che l'obiettivo non è quello di insegnare informatica, nonostante non vi sia nulla di immeritevole in tale disciplina, ma si cerca di spostare l'attenzione sul *coding* come strumento per lo sviluppo di competenze trasversali e per l'acquisizione di un pensiero critico e creativo.

In ultimo, vi sono anche delle difficoltà di natura pratica, legate alla formazione degli insegnanti e alla disponibilità di strumenti e risorse adeguate. La formazione degli insegnanti, infatti, è un aspetto cruciale per garantire il successo di un'iniziativa di questo tipo: non solo è necessario che gli insegnanti siano in grado di padroneggiare le competenze tecniche necessarie per insegnare il *coding*, ma è anche importante che siano in grado di integrare queste competenze in un contesto educativo più ampio, in modo da garantire che il *coding* non sia un'attività isolata, ma un'attività che si integri in modo armonioso con le altre materie e discipline. Per quanto riguarda, invece, la possibilità di avere accesso a tecnologie e strumenti utili per questa tipologia di insegnamento, bisogna considerare che non tutte le scuole del mondo sono in grado di permettersi di acquistare i materiali necessari. In aree più svantaggiate, ad esempio, dove è già difficile garantire l'accesso a materiali didattici di base, l'idea di introdurre il *coding* come disciplina può sembrare un lusso eccessivo. Tuttavia, moltissimi studi hanno dimostrato che l'uso di strumenti tecnologici non è una *conditio sine qua* per insegnare la programmazione informatica: esistono infatti metodologie molto efficaci e, al contempo assolutamente economiche, che riescono a trasmettere il pensiero computazionale anche senza l'uso del computer. Il *coding* può essere insegnato anche attraverso attività di *role-playing*, di *problem-solving* e di *game-based learning* dette *unplugged*. Un esempio di questi studi è stato compiuto da Arinchaya Threekunprapa e Pratchayapong Yasri, che hanno dimostrato come l'uso di *unplugged coding* usando dei *flow-blocks* abbia portato a risultati ottimi per quanto riguarda la promozione del pensiero computazionale tra studenti della scuola secondaria. Il paper, pubblicato nel 2020, mette in luce come, anche utilizzando delle semplici forme di carta - rappresentanti i diversi blocchi di codice e i relativi significati algoritmici - si possa ottenere un apprendimento significativo e duraturo. L'attività proposta era *game based*, ovvero posta in un contesto ludico e divertente, e ha portato a risultati molto positivi, sia in termini di apprendimento che di coinvolgimento degli studenti. I partecipanti, lavorando in coppia, non solo hanno dovuto collaborare e comunicare tra loro per risolvere le diverse sfide, ma hanno avuto anche un supporto morale e psicologico nel momento in cui si sono trovati ad affrontare sfide sempre più difficili, incontrando anche errori e fallimenti. Il confronto, però, ha reso la fase del *debugging* molto più semplice

e veloce, e, insieme anche ad una buona dose di competitività tra i vari gruppi, tutti i partecipanti sono riusciti a portare a termine con successo le sfide proposte. La fase finale della sperimentazione di Threekunprapa e Yasri ha visto i partecipanti - che non avevano alcuna esperienza pregressa di programmazione - migliorare significativamente rispetto al test iniziale.

Capitolo 2

Sperimentazione e valutazione

2.1 Metodologia di sperimentazione

2.2 Percorso formativo e strumenti utilizzati

2.3 Raccolta e analisi dei dati

2.4 Risultati ottenuti

Capitolo 3

Analisi critica degli strumenti utilizzati

3.1 Snap!

Snap! Build Your Own Blocks è un ambiente di programmazione visuale che si basa sulla sintassi dei blocchi, progettato per rendere l'apprendimento della programmazione accessibile e divertente. Sviluppato dal MIT Media Lab, è una variante di Scratch, un altro ambiente di programmazione visuale. Di seguito, fornirò una descrizione approfondita di Snap! Build Your Own Blocks, esplorando diverse caratteristiche e aspetti del linguaggio di programmazione visuale.

1. Interfaccia utente:

Canvas: L'interfaccia principale di Snap! è un canvas vuoto su cui gli utenti posizionano e collegano i blocchi. Blocchi: I blocchi sono gli elementi fondamentali del linguaggio. Ogni blocco rappresenta un comando o una funzione specifica. 2. Tipi di Blocchi:

Blocchi di Comandi: Questi blocchi eseguono azioni specifiche, come "muovi avanti di 10 passi" o "gira a sinistra di 90 gradi". Blocchi di Controllo: Gestiscono la sequenza di esecuzione del programma con strutture come "ripeti" o "se". Blocchi di Operatori: Effettuano operazioni matematiche, logiche o di confronto. Blocchi di Variabili: Consentono agli utenti di creare e utilizzare variabili nelle loro programmazioni. 3. Funzionalità Principali:

Modularità: Gli utenti possono creare propri blocchi personalizzati per organizzare e riutilizzare il codice. Costume e Sprite: Gli oggetti visivi (Sprite) possono avere diverse apparenze (Costume), e gli utenti possono controllarli tramite il codice. Eventi: Gli script possono essere attivati da vari eventi, come clic del mouse, tastiera o in risposta a condizioni specifiche. Sensori e Variabili Globali: Snap! supporta sensori virtuali (posizione del mouse, tempo)

e variabili globali per la memorizzazione di informazioni a livello di progetto. 4. Programmazione di Livello Avanzato:

Procedure Definite dall'Utente: Gli utenti possono creare procedure personalizzate, raggruppando blocchi e dando loro un nome. Ricorsione: Snap! supporta la ricorsione, consentendo agli utenti di definire funzioni che chiamano se stesse. Manipolazione delle Liste: Gli utenti possono lavorare con liste e array per gestire dati in modo strutturato. 5. Esportazione e Condivisione:

Esportazione del Codice: Il codice creato in Snap! può essere esportato in vari formati, inclusi JavaScript e Python, per essere eseguito al di fuori dell'ambiente. Condivisione Online: Gli utenti possono condividere i propri progetti online attraverso il sito web di Snap!, incoraggiando la condivisione e la collaborazione. 6. Comunità e Supporto:

Forum e Risorse: Snap! dispone di una comunità attiva con forum online e risorse di apprendimento per supportare gli utenti. Documentazione: Una documentazione completa fornisce informazioni dettagliate su ogni aspetto del linguaggio. 7. Aspetti Pedagogici:

Apprendimento Interattivo: L'approccio visuale rende l'apprendimento della programmazione intuitivo ed interattivo, adatto a un pubblico giovane e a principianti. Promozione del Pensiero Computazionale: Snap! mira a sviluppare il pensiero computazionale, incoraggiando la risoluzione di problemi e la creatività attraverso la programmazione. In conclusione, Snap! Build Your Own Blocks offre un ambiente di programmazione visuale completo, ricco di funzionalità e progettato per facilitare l'apprendimento e la pratica della programmazione in modo divertente e coinvolgente.

3.2 CoSpaces Edu

3.3 Croquet

3.4 Confronto tra strumenti

Capitolo 4

Progettazione di un nuovo software

4.1 Requisiti del software

4.2 Architettura del software

4.3 Caratteristiche chiave

Capitolo 5

Conclusioni

5.1 Riassunto dei risultati ottenuti

5.2 Contributi della ricerca

5.3 Limiti dello studio e sviluppi futuri

Ringraziamenti

Qualora lo si desideri è possibile inserire qui il testo dei ringraziamenti alle persone che hanno contribuito in qualche modo al percorso che ha portato al lavoro di tesi.

Bibliografia