
UNIVERSITÀ DI PARMA

*Dottorato di Ricerca in Tecnologie dell'Informazione*

*XXX Ciclo*

# Tangible e Block Programming per introdurre il pensiero computazionale

Coordinatore:

*Chiar.mo Prof. Marco Locatelli*

Tutor:

*Chiar.mo Prof. Michele Tomaiuolo*

Dottorando: *Alberto Ferrari*

Anni 2014/2017

*a Francesca, Irene, Lea e Veronica*
*. . . in rigoroso ordine alfabetico*

*I know I've made some very poor decisions recently, but I can give you my complete assurance that my work will be back to normal. I've still got the greatest enthusiasm and confidence in the mission. And I want to help you.*

*HAL*

**Sommario**

# Sommario

*"Most good programmers do programming*
*not because they expect to get paid or get adulation by the public,*
*but because it is fun to program."*

– Linus Torvalds

Negli ultimi anni ha ripreso vigore, non solo in ambito accademico, il dibattito relativo al *pensiero computazionale (Computational Thinking)* e sempre più numerosi sono oggi i progetti volti all'introduzione dei concetti basilari della programmazione (*coding*) in ogni livello di istruzione: dai primi anni scolastici, alla scuola media superiore, alla formazione professionale, ai primi corsi universitari.

Nonostante gli sforzi tesi a facilitare l'approccio iniziale con il mondo dell'informatica, risultano ancora evidenti le difficoltà incontrate almeno da una parte dei principianti nell'affrontare la scrittura dei primi programmi. La maggior parte di queste difficoltà derivano da problematiche inerenti il problem solving e la progettazione delle applicazioni, ma non sono da trascurare i problemi di natura sintattica legati all'utilizzo dei più diffusi linguaggi di programmazione.

L'obiettivo di questa tesi è quello di analizzare le potenzialità derivanti da un approccio iniziale al coding basato sul Block e Puzzle Programming contrapposto alla classica programmazione text-based. In relazione all'introduzione al coding rivolta a giovani e giovanissimi (formazione K12) riteniamo sia inoltre necessario uno sforzo per rendere più semplice e user-friendly la "scrittura" dei primi programmi per l'elaboratore ridefinendo, oltre alle modalità didattiche, anche gli strumenti da utilizzare analogamente a come, nel tempo, si sono evolute le modalità di interazione uomo-computer (*HCI - Human-Computer Interaction*).

Analizziamo il concetto di Computational Thinking e come questo sia legato al mondo

dell'informatica e al coding. L'analisi dei vari progetti mirati alla diffusione del pensiero computazionale mediante il coding in ambito scolare e prescolare ci porta ad approfondire poi la programmazione a blocchi. Il Block Programming presenta ambienti di programmazione non più testuali ma basati su semplici interfacce grafiche (*GUI - Graphical User Interface*) in cui avviene la selezione e il collegamento dei blocchi-istruzioni. In questo contesto è presentato il nostro progetto codOWood (Code of wood) che sostituisce i blocchi virtuali delle GUI con blocchi fisici che vengono disposti sulla superficie di lavoro in modo da formare un puzzle che rappresenta l'algoritmo di risoluzione del problema proposto. Il riconoscimento ottico (*image recognition*) dei vari blocchi e la loro conversione nel codice equivalente avviene tramite l'applicazione che abbiamo sviluppato e che opera sia come web application, sia in ambiente Android su tablet e smartphone. Il nostro ambiente di sviluppo sostituisce quindi l'interfaccia grafica di programmazione (GUI) con una interfaccia "fisica" (TUI - Tangible User Interface).

Dopo aver analizzato le modalità didattiche, gli strumenti e i contenuti dei corsi di introduzione alla programmazione (CS1) in ambiente universitario e di scuola superiore, focalizziamo l'attenzione sul modello *object-first* che prevede l'introduzione del paradigma object-oriented già nei corsi introduttivi.

All'Università di Parma abbiamo sviluppato OOPP (Object Oriented Puzzle Programming): un ambiente didattico di progettazione che ha l'obiettivo di fondere le caratteristiche positive del Block e Puzzle Programming con il paradigma object-oriented.

OOPP è stato oggetto di una prima sperimentazione nell'ambito di un corso post diploma di introduzione alla programmazione con risultati più che soddisfacenti soprattutto per gli studenti che non avevano precedenti esperienze di coding.

## Abstract

# Abstract

Over the last few years the debate on Computational Thinking has risen not only in the academic sphere. There is an increasing number of projects aimed at introducing the basic coding concepts in each level of education: early school years, high school, vocational training and first university courses.

Despite efforts to facilitate the initial approach to the world of information technologies, the difficulties faced by at least part of beginners in addressing the writing of the first programs are still evident. Most of these difficulties arise from problems related to problem solving and the design of applications, but also the syntactic problems, associated with the use of the most common programming languages, cannot be neglected.

The aim of this thesis is to analyze the potential of an early approach to coding based on Block and Puzzle Programming, as opposed to classic text-based programming.

In relation to the introduction of coding for young and very young students (K12 training), we also believe that efforts are needed to make the first programs for the computer easier and user-friendly. This requires to define, in addition to teaching methods, also the tools to use, similarly to how man-computer interaction (HCI-Human-Computer Interaction) has evolved over time.

We analyze the concept of Computational Thinking and how this is tied to the world of computing and coding. The analysis of the various projects aimed at the spread of computational thinking by coding in the school and preschool years leads us to further spread the block programming paradigm. Block Programming environments are based on simple graphical user interfaces (GUIs) where blocks are selected and linked. In this context, we present our codOWood (CODe Of WOOd) project that replaces the virtual blocks of the GUIs with physical blocks that are arranged on the work surface to form a puzzle that represents the solution algorithm of the proposed problems. We developed a web application and an Android app for tablet and smartphones which recognizes each block and converts it into

the equivalent code. Our development environment then replaces the graphical programming interface (GUI) with a “physical” (TUI) interface.

After analyzing the teaching methods, the tools and the content of the introductory programming courses (CS1) in university and high school, we focus our attention on the Object-First model that introduces the Object Oriented paradigm in introductory courses.

At the University of Parma, we have developed OOPP (Object Oriented Puzzle Programming): a didactic environment for designing applications that aims to blend the positive features of Block and Puzzle Programming with the Object Oriented paradigm.

OOPP has been the subject of a first experimentation in an introductory coding course with more than satisfactory results, especially for students who did not have previous coding experiences.

# Contents

# List of Figures

# List of Tables

| | |
|---|---|
| API | Application Program Interface |
| BGR | Blue Green Red |
| BSD | Berkeley Software Distribution |
| BYOD | Bring Your Own Device |
| CINI | Consorzio Interuniversitario Nazionale per l'Informatica |
| CS0 | Computer Science 0 (Corsi di Informatica per bambini) |
| CS1 | Computer Science 1 (Corsi Iniziali di Informatica) |
| CT | Computational Thinking |
| EMMA | European Multiple MOOC Aggregator |
| Exif | Exchangeable image file format |
| GII | Gruppo Ingegneria Informatica |
| GRIN | Gruppo di Informatica |
| GUI | Graphical User Interface |
| HCI | Human-Computer Interaction |
| HSL | Hue Saturation Lightness |
| HSV | Hue Saturation Value |
| IDE | Integrated Development Environment |
| IT | Information Technology |
| JNI | Java Native Interface |
| K-12 | Educazione primaria e secondaria |
| LIM | Lavagna Interattiva Multimediale |
| MIUR | Ministero dell'Istruzione dell'Università e della Ricerca |
| MOOC | Massive Open Online Course |
| NDK | Android Native Development Kit |
| OOP | Object Oriented Programming |
| OOPP | Object Oriented Puzzle Programming |
| RGB | Red Green Blue |
| TP | Tangible Programming |
| TUI | Tangible User Interface |
| UML | Unified Modeling Language |
| WIMP | Window, Icon, Menu and Pointing device |
| XML | eXtensible Markup Language |

# Introduzione

*"A programming language is low level*
*when its programs require attention to the irrelevant."*

– Alan J. Perlis

L'obiettivo di questa tesi è quello di cercare di dare una risposta al seguente quesito di ricerca: *"È possibile semplificare il primo approccio al coding mediante ambienti di sviluppo il più possibile user-friendly che permettano di superare le difficoltà sintattiche intrinseche alla programmazione text based?"*

Per rispondere a questo quesito presentiamo nella prima parte il concetto di Computational Thinking e le varie interpretazioni che sono state date a questo termine. In particolare approfondiamo il discorso del coding come strumento per sviluppare gli aspetti principali del pensiero computazionale.

Nel primo capitolo analizziamo il dibattito sul rapporto fra Computational Thinking, Computer Science e Coding attraverso una rassegna della letteratura che tratta questi argomenti.

Nel secondo capitolo analizziamo inoltre i corsi di introduzione alla programmazione in ambito universitario e di scuola superiore. In particolare focalizziamo l'attenzione sulle metodologie didattiche object-first, design-first e game-first.

Nel terzo capitolo prendiamo in esame i vari progetti che, soprattutto negli ultimi anni, promuovono la diffusione del coding per i giovani e i giovanissimi. In particolare analizziamo le caratteristiche del Block Programming che è la tipologia di linguaggio utilizzata nella maggior parte dei progetti per la diffusione del coding per i giovani.

Nella seconda parte analizziamo due sperimentazioni che abbiamo messo in atto in situazioni didattiche di introduzione alla programmazione. Nel quarto capitolo presentiamo l'esperienza degli stage estivi di introduzione alla programmazione che si sono tenuti negli ultimi anni presso il nostro dipartimento. Nel corso dello stage settimanale gli studenti, la maggior parte dei quali senza precedenti esperienze di programmazione, ha sviluppato, procedendo per gradi, un videogame interattivo. Una serie di questionari anonimi ha permesso poi di valutare il gradimento di questa esperienza e i risultati raggiunti dagli studenti.

Con l'obiettivo di fondere le caratteristiche del Block Programming e della metodologia object-first per i corsi CS1 presentiamo poi nel capitolo 5 il progetto Object Oriented Puzzle Programming (OOPP) che abbiamo sviluppato e sperimentato. L'ambiente OOPP favorisce un primo approccio alla definizione della struttura generale delle classi in un'applicazione basata su paradigma object-oriented. Collegando fra loro blocchi delle varie tipologie proposte dal software si definisce la struttura delle varie classi e delle associazioni fra queste ottenendo contemporaneamente la generazione del codice Java corrispondente.

La terza parte tratta del Tangible Computer Programming. Nel capitolo 6 presentiamo una rassegna dei progetti basati su blocchi fisici attivi o passivi utilizzati per la programmazione.

Nel capitolo 7 presentiamo il progetto codOWood che abbiamo realizzato nel laboratorio SoWIDE dell'Università di Parma. Si tratta di una proposta didattica per l'introduzione al coding in ambito di scuola primaria basata sul Tangible Programming. Gli studenti, organizzati in gruppi di lavoro, collegano blocchetti di legno in modo da formare un puzzle/programma e, utilizzando un tablet o uno smartphone catturano l'immagine del puzzle che viene trasformata mediante image recognition in istruzioni del linguaggio codOWood.

# Part I

# Computational Thinking e Coding

# Chapter 1

# Computational Thinking

*"Computational Thinking.*
*It represents a universally applicable attitude and skill set*
*everyone, not just computer scientists, would be eager to learn and use."*

– Jeannette M. Wing

## 1.1 Il pensiero computazionale

Il concetto di *Computational Thinking* (CT), anche se è argomento di discussione in ambito accademico da decine di anni, è stato reso popolare da Jeannette Wing nel suo famoso articolo del 2006 [1]. Wing definisce CT come l'insieme di competenze e abilità che oggi ogni persona, non solo gli esperti nel campo informatico, dovrebbe possedere; fra queste competenze mette in evidenza la capacità di applicare metodi per l'astrazione, la scomposizione e la soluzione di problemi.

Per pensiero computazionale si intende quindi un'attitudine, un processo mentale, che consente di risolvere problemi di varia natura seguendo metodi e strumenti specifici, CT è definibile come la capacità di risolvere un problema pianificando una strategia. Si tratta di un processo logico-creativo che mira a scomporre un problema complesso in varie parti, più semplici da gestire se affrontate una per volta. Individuata

una soluzione per ciascuna di queste è quindi possibile risolvere il problema generale.

Capita spesso di intuire la soluzione ad un problema ma di non essere in grado di formularla in modo formale e operativo per poi metterla in pratica. Fa parte quindi del pensiero computazionale la capacità di immaginare e, soprattutto di saper descrivere, un procedimento costruttivo per raggiungerla. Questa capacità trasversale va sviluppata, seppur per gradi, il prima possibile.

Sviluppare il pensiero computazionale comporta:

- confidenza nel trattare la complessità,
- ostinazione nel lavorare con problemi di varia difficoltà,
- tolleranza all'ambiguità, pur mantenendo il necessario rigore che assicuri la correttezza della soluzione,
- abilità nel trattare problemi anche se questi sono definiti in modo incompleto,
- capacità di comunicare e lavorare con gli altri per raggiungere una soluzione condivisa.

Ogni persona dovrebbe comprendere e padroneggiare i vari aspetti dell'informatica, ma, purtroppo, troppo spesso la percezione è limitata a uno solo di questi.

Mentre per la maggior parte delle persone comuni conoscere l'informatica significa "saper utilizzare le varie applicazioni", spesso, gli specialisti evidenziano il solo aspetto tecnico: "saper realizzare le applicazioni". L'aspetto scientifico dell'informatica dovrebbe essere invece patrimonio di ogni persona con educazione superiore. I principi fondamentali della scienza informatica sono infatti in grado di fornire uno strumento culturale e modelli di interpretazione utili per affrontare le sfide del mondo moderno.

In qualche modo *Computational Thinking* può essere considerata un'evoluzione del concetto di *Algorithmic Thinking* che ha identificato l'atteggiamento mentale inerente la descrizione di un problema come trasformazione dai dati di input a quelli di output e della sua traduzione nel relativo algoritmo.

Il pensiero computazionale e le sue implicazioni nella società odierna sono stati elementi di discussione in vari ambiti scientifici.

Nei *Report of a workshop on the scope and nature on the pedagogical aspects of computational thinking* del 2010 [2] e del 2011 [3] si mettono in relazione *Computational Thinking* e *Matemathical Thinking* evidenziando come entrambi siano fondati sull'astrazione, il ragionamento e la modellizzazione.

Sempre in [2] David Moursund, individua la stretta relazione fra *Computational Thinking* e *Procedural Thinking* rifacendosi al testo *Mindstorms* [4] di Seymour Papert in cui l'autore delinea le idee portanti di un tipo di pedagogia che assieme ad altri modelli teorici è compresa sotto il nome di costruttivismo.

Esistono quindi varie, e a volte discordanti, definizioni di CT e varie sono anche le risposte ai quesiti relativi a come sia possibile valutare le competenze degli studenti in questo ambito [5].

### 1.1.1 Computer Science, Computational Thinking e Coding

> *"It has often been said that a person does not really understand something until he teaches it to someone else. Actually a person does not really understand something until he can teach it to a computer, i.e., express it as an algorithm."*
>
> – Donald E. Knuth [6]

L'obiettivo principale del pensiero computazionale non è quello di sviluppare applicazioni informatiche, tuttavia è certamente vero che i linguaggi di programmazione sono utili e spesso irrinunciabili strumenti per apprendere i concetti dell'informatica.

Non mancano nel mondo informatico le visioni critiche che mostrano come siano errate alcune equazioni semplicistiche tendenti a considerare equivalenti Computer Science e CT. Denning [7], per esempio, cerca di smorzare l'entusiasmo effettivamente esagerato di chi afferma:

> *"It is nearly impossible to do research in any scientific or engineering discipline without an ability to think computationally....[We] advocate for the widespread use of computational thinking to improve people's lives."*

Carnegie Mellon University's Center for Computational Thinking

Lo stesso Denning [8] contesta l'idea che l'informatica sia equivalente alla programmazione, ma è pur vero che i linguaggi di programmazione sono un utile strumento per raggiungere gli obiettivi caratterizzanti la disciplina.

Favorire e semplificare l'approccio allo sviluppo di applicazioni non è certo il fulcro del CT, riteniamo però che riuscire a proporre ambienti, strumenti e linguaggi che facilitino l'avvicinamento alla programmazione possa essere un punto di partenza per introdurre poi aspetti di metodologia informatica quali per esempio l'astrazione e la modellizzazione.

Nella definizione dei curricula 2013 relativi a Computer Science [9] si fa notare come la maggior parte dei corsi introduttivi all'informatica siano programming-focused; gli studenti apprendono i concetti fondamentali, quali ad esempio astrazione e scomposizione, attraverso lo studio di linguaggi di programmazione e lo sviluppo di applicazioni software.

Nel *Workshop on The Scope and Nature of Computational Thinking* [2] si evidenzia come la programmazione sia un modo per esprimere idee e come sia necessario imparare a leggere e scrivere in questo linguaggio per poter *pensare* in questo linguaggio.

Villani [10] sottolinea il valore del coding come disciplina di base e definisce la programmazione e la matematica come linguaggi che aiutano l'uomo nella sua lotta per la comprensione del mondo.

Una importante ricerca di Google e Gallup [11] riporta che la maggior parte degli studenti, genitori e insegnanti negli Stati Uniti non distinguono correttamente la *computer literacy* (abilità nell'utilizzare gli strumenti informatici) dalla *computer science* e che spesso nei corsi di informatica non è insegnata la programmazione. Altri studi riportano che l'86% degli americani ritiene l'informatica una delle discipline fondamentali e che il saper utilizzare un computer oggi è importante tanto quanto il saper leggere e scrivere.

Per distinguere le competenze basilari dalla scienza che sta dietro l'*Information*

*Technology*, si preferisce utilizzare il termine *"Informatics"*, che è già diffuso nell'Europa continentale, ma che sta acquisendo il significato di *"Computer Science"* anche nel mondo anglosassone.

L'informatica è una disciplina scientifica che è nata prima dei computer, con gli studi di Turing e Church negli anni '30. Nei decenni successivi, con il suo sviluppo verso una disciplina completa, l'informatica è diventata un interessante mix di teoria matematica e ingegneria elettronica, con i sui concetti specifici, tra cui: algoritmi, concetti legati alla complessità e alle prestazioni, strutture dati, concorrenza, parallelismo, e programmazione distribuita, linguaggi formali e astrazioni.

L'importanza dell'introduzione dell'informatica nella scuola superiore è confermata dal fatto che gli studenti che hanno studiato questa disciplina nel loro percorso scolastico di base hanno una probabilità sei volte maggiore degli altri di approfondire poi questi studi e che l'occupazione nel campo informatico è oggi la maggior fonte di guadagno negli Stati Uniti. Per questa ragione nel 2016 l'amministrazione americana ha investito 150 milioni di dollari per promuovere progetti per la diffusione di questa disciplina.

Le proposte di nuovi ambienti, strumenti e linguaggi atti a facilitare il primo approccio alla programmazione possono essere il punto di partenza per poi introdurre gli aspetti fondamentali del pensiero computazionale come l'astrazione e la modellizzazione.

*"Every student in every school should have the opportunity to learn computer science"*

È questo l'obiettivo specifico di code.org, uno dei più importanti progetti per la diffusione nel mondo del pensiero computazionale che promuove la cosiddetta *CS-for-all K-12 education*.

## 1.2 Esperienze internazionali

La situazione dell'istruzione nel campo dell'informatica in Europa non è ideale come evidenzia un recente studio, redatto da ACM & IE, il cui titolo è abbastanza significativo: *"Informatics education: Europe cannot afford to miss the boat"* [12]. Lo studio parte dalla considerazione che sia i governi che i cittadini dell'UE sono consapevoli

di vivere nella cosiddetta "Società dell'informazione", ma troppo spesso si accontentano dell'idea che è sufficiente apprendere l'uso di mezzi digitali (*alfabetizzazione digitale*). Al contrario, l'alfabetizzazione digitale deve essere considerata come una pratica di base, e non come uno strumento intellettuale adeguato per affrontare le nuove sfide.

Fra gli impegni e le proposte in atto per la diffusione della cultura, non solo informatica, possiamo citare il progetto EMMA (The European Multiple MOOC Aggregator) supportato dall'Unione Europea [1]. EMMA ha l'obiettivo di evidenziare l'eccellenza nelle metodologie innovative di insegnamento attraverso la sperimentazione di MOOC su diversi argomenti.

Come abbiamo affermato in precedenza, la situazione generale non è ideale ma è da segnalare a tal proposito l'impegno messo in atto da vari paesi (fra questi: Regno Unito [13], Francia [14], Stati Uniti [15]) per introdurre lo studio dell'informatica in ogni ordine di scuola.

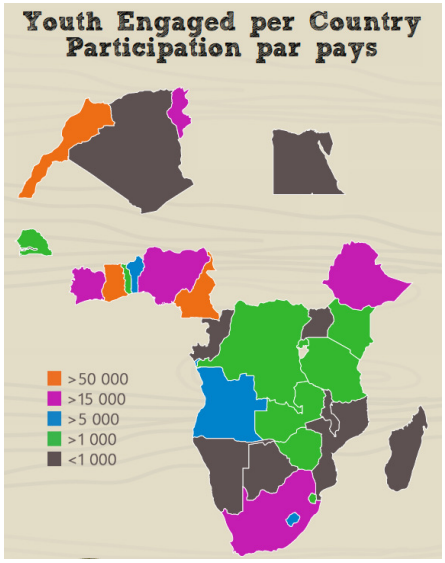



Figure 1.1: *Africa Code Week*

[1] platform.europeanmoocs.eu

Nel 2016 è stato avviato il progetto *AfricaCodeWeek* [2] che ha coinvolto più di 400.000 bambini in 30 paesi del continente africano (fig. 1.1) che hanno avuto il primo contatto con i computer e hanno scritto le loro prime righe di codice. Nel 2017 si sono aggiunti ulteriori 5 paesi e il progetto si pone ora obiettivi ambiziosi a breve, medio e lungo termine per riuscire a coinvolgere 500.000 bambini nel 2017, un milione di bambini nei prossimi 5 anni e 10 milioni di bambini nell'arco dei prossimi 10 anni con la parola d'ordine *Read, Write, Count, Code* e la mission *"To empower future generations with the coding tools and skills they need to thrive in the 21st century workforce and become key actors of Africa's economic development."*.

## 1.3 Progetti italiani

In Italia, l'introduzione dell'informatica nei vari percorsi scolastici è solo parziale. La cosiddetta *alfabetizzazione informatica* ha spesso avuto l'unico obiettivo di educare all'uso del computer e delle applicazioni di uso comune mentre le valenze formative di un approccio informatico sono in realtà quelle di favorire la creatività e al contempo il ragionamento rigoroso che spesso sono lasciate in secondo piano a favore di un approccio più "tecnico" che fa perdere di vista l'obiettivo primario.

L'aspetto tecnologico, che non deve tuttavia essere completamente trascurato, ha inoltre il problema dell'obsolescenza: cambia lo strumento e la tecnologia e quindi cambiano le competenze.

Oggi anche in Italia il mondo dell'istruzione comincia a interessarsi ai concetti legati al pensiero computazionale. Per molto tempo l'approccio didattico è stato incentrato sull'aspetto operativo e tecnologico della disciplina trascurando quello culturale: il metodo informatico. Questo nonostante varie sensibilizzazioni, provenienti soprattutto dall'ambiente universitario [16], che hanno evidenziato come l'informatica sia un elemento essenziale della società moderna.

Una importante riforma della scuola secondaria ha avuto inizio nel 2010, ne è seguita una serie di discussioni con opinioni spesso contrastanti. Sono subito apparse critiche in opposizione alla diffusione generalizzata dell'insegnamento della

[2] africacodeweek.org

programmazione e dei fondamenti della scienza informatica. Molte di queste portavano come esempio il fatto che "non è necessario essere un ingegnere meccanico per poter guidare un'auto". Ricercatori e professionisti del settore informatico hanno, al contrario, sostenuto l'idea della necessità di una più ampia diffusione dell'informatica come disciplina scientifica e si sono dichiarati a favore di un aumento delle ore dedicate a questi studi nella scuola.

Importanti gruppi di ricerca, CINI (Consorzio Interuniversitario per l'Informatica), GII (Gruppo Ingegneria Informatica) e GRIN (Gruppo di Informatica), hanno stilato congiuntamente un manifesto dal titolo *"Informatica nella riforma della scuola superiore"* [3]. Il *manifesto* presenta l'informatica come elemento essenziale della società moderna in quanto il suo sviluppo plasma e determina quello dell'intera società. Dell'informatica vengono presentati i tre aspetti:

- *operativo*: un insieme di applicazioni e manufatti (i computer);
- *tecnologico*: una tecnologia che realizza le applicazioni;
- *culturale*: una disciplina scientifica che fonda e rende possibile la tecnologia.

L'informatica è quindi intesa come complesso di conoscenze operative, scientifiche e tecnologiche che permettono di realizzare il cosiddetto *metodo informatico*.

La comprensione delle fondamenta culturali e scientifiche dell'informatica, che è alla base delle tecnologie digitali, permette ai giovani di essere soggetti consapevoli di tutti gli aspetti in gioco ed attori attivamente partecipi del loro sviluppo piuttosto che consumatori passivi ed ignari di tali servizi e tecnologie.

Oggi è fondamentale introdurre, già dai primi livelli scolastici, i concetti relativi all'alfabetizzazione informatica e digitale (*information literacy e digital literacy*), che mettono al centro il ruolo dell'informazione e dei dati in una società sempre più basata sulle informazioni e sulla loro gestione. Familiarizzare con gli aspetti operativi delle tecnologie è importante, ma gli studenti, oltre a diventare utenti consapevoli di ambienti e strumenti digitali, devono soprattutto diventare progettisti di applicazioni e creatori di informazioni.

[3] www.grin-informatica.it/opencms/export/sites/default/grin/files/manifesto.pdf

Nel quadro delle proposte di riforma della scuola, il Ministero dell'Istruzione dell'Università e della Ricerca (MIUR) ha aperto una discussione proponendo il documento *La Buona Scuola* [4] in cui si auspica un Piano Nazionale che consenta di introdurre la programmazione già a partire dalla scuola primaria anche attraverso attività di gaming. In collaborazione con il CINI è stato proposto dal 2014 il progetto *Programma il Futuro* [5] [17]. *Programma il Futuro* fa riferimento al più ampio progetto *Code.org* [6] e ha coinvolto in Italia nel 2017 più di 1500000 studenti (fig. 1.2). Da una analisi delle reazioni dei docenti [18] risulta che la maggior parte degli studenti ha trovato i materiali e il progetto in generale estremamente interessanti avvalorando quindi l'ipotesi che l'approccio ai concetti dell'informatica già dai primi anni di scuola è visto in maniera positiva sia dagli alunni che dalle alunne. I commenti dei docenti che hanno partecipato al progetto evidenziano inoltre come sia stato soprattutto l'aspetto ludico delle proposte ad aver suscitato un inatteso interesse e una forte motivazione da parte degli studenti.

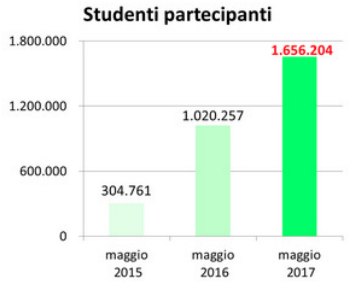



Figure 1.2: *Programma il Futuro*

La riforma della scuola ha introdotto elementi di informatica nella scuola media superiore già dal primo anno negli istituti tecnici e nei licei di scienze applicate. In altre scuole (licei classici, licei scientifici e licei pedagogici) concetti di informatica sono proposti all'interno degli insegnamenti di matematica. Inoltre molte iniziative

[4] labuonascuola.gov.it
[5] programmailfuturo.it
[6] code.org

basate su attività ludiche sono si sono rivolte soprattutto alla scuola primaria e lo stesso ministero auspica e promuove progetti per sollecitare i ragazzi della scuola secondaria [19] a diventare *"Digital Makers"*.

Oggi, partendo dalle proposte suggerite dal MIUR, si sta sempre più diffondendo l'opinione che l'informatica, come scienza, debba far parte del patrimonio di ogni persona con educazione superiore e che i ragazzi debbano essere sempre più protagonisti nell'era digitale.

# Chapter 2

# Introduzione al coding

*"A computer is like a violin. You can imagine a novice trying first a phonograph and then a violin. The latter, he says, sounds terrible. That is the argument we have heard from our humanists and most of our computer scientists. Computer programs are good, they say, for particular purposes, but they aren't flexible. Neither is a violin, or a typewriter, until you learn how to use it."*

– Marvin Minsky.

## 2.1 Didattica nei corsi CS1

Il tema del coding è sempre più sentito nelle università e non solo nei corsi di laurea scientifici. Un esempio in merito lo troviamo all'Università Bocconi di Milano che, dal corrente anno accademico, ha reso obbligatorio l'esame di programmazione per tutti gli indirizzi di studio.

Esiste una vasta letteratura che si occupa dei contenuti, dei linguaggi e dell'approccio didattico nei corsi universitari di introduzione all'informatica (CS1) [20].

Per quanto riguarda l'approccio didattico possiamo citare il lavoro di Bergin [21] che suggerisce a riguardo una serie di pattern metodologici. Fra questi, in particolare, ci sentiamo di condividere a pieno il pattern *Early Bird* che stabilisce l'importanza di introdurre già nelle fasi iniziali dei corsi i concetti più importanti, le "gradi idee" che

verranno poi riaffrontati a più riprese. Gli studenti spesso ricordano meglio quello che imparano prima; se, per fare un esempio, il design e la progettazione sono ritenuti più importanti della programmazione, allora si deve trovare il modo di introdurre almeno gli elementi basilari di questi concetti il prima possibile. La metodologia object-first, che analizzeremo in seguito, si base proprio su questo pattern. Object-first sarà infatti il punto di partenza per i nostri progetti poichè riteniamo che se si introduce e si sperimenta la decomposizione procedurale all'inizio di un corso risulta poi più difficile per gli studenti cambiare l'approccio e passare al paradigma object-oriented.

Presentare i concetti fondamentali per poi riprenderli ed approfondirli in momenti successivi si colloca all'interno di un altro pattern presentato da Bergin, il pattern *Spiral* che suggerisce di divedere in piccole unità concettuali gli argomenti di un corso e di introdurre queste in un ordine che faciliti la comprensione degli studenti. Nei cicli aggiuntivi successivi si presentano poi unità di rinforzo che analizzano il tema più in dettaglio.

Nel pattern *Larger Than Life* si suggerisce di utilizzare inizialmente strumenti evoluti ancor prima di comprenderne a pieno la struttura interna e di avere le competenze necessarie per progettarli e costruirli. In un corso di programmazione orientata agli oggetti ad esempio, gli studenti inizialmente utilizzeranno librerie di classi standard, o in ogni caso fornite loro dal docente, prima di avere le competenze per poi sviluppare autonomamente librerie o progetti di questo tipo.

Sempre in relazione alle argomentazioni presentate da Bergin l'approccio didattico alla programmazione può basarsi su due modelli: il primo prevede già da subito la stesura da parte degli studenti di interi programmi per comprendere a pieno il modo in cui le varie parti del codice si combinano fra loro, il secondo prevede inizialmente che gli studenti modifichino e completino strutture di programmi già predisposti. Nel primo caso i problemi affrontati devono essere necessariamente di minor complessità mentre nel secondo caso, che risulta più simile alla reale esperienza, è possibile sviluppare programmi sicuramente più complessi.

La maggior parte dei docenti che si occupano di introduzione alla programmazione riconosce l'importanza dell'introduzione anticipata dei concetti relativi al design del sistema per evitare che gli studenti vadano a sviluppare le loro tecniche ad-hoc che

possono poi rafforzare cattive abitudini di programmazione (antipattern).

Queste motivazioni sono alla base del progetto Object Oriented Puzzle Programming che andremo in seguito a presentare: se il nostro obiettivo è ottenere bravi sviluppatori object-oriented allora i concetti relativi a questo paradigma devono essere trattati già nella prima parte dei corsi introduttivi, fornendo, se possibile, strumenti per rendere meno difficoltoso questo primo approccio.

## 2.2 La scelta del linguaggio

Nei corsi di introduzione all'informatica uno degli obiettivi principali è quello di avvicinare gli studenti ai concetti del pensiero computazionale e in particolare al coding riducendo al minimo i fattori che rendono difficoltoso il lavoro soprattutto nella fase iniziale.

Come evidenziato in [19], un punto critico e centrale è relativo alla scelta del linguaggio di programmazione e del relativo ambiente di sviluppo allo scopo di ridurre lo sforzo dedicato all'acquisizione di dettagli tecnici e di focalizzare invece la concentrazione soprattutto sui principi della programmazione.

La scelta del linguaggio di programmazione, che in teoria dovrebbe essere solo lo strumento per esemplificare i concetti, ha in realtà una notevole influenza sulla struttura del corso e sulle competenze che gli studenti vanno ad acquisire.

In primo luogo le difficoltà che gli studenti incontrano inizialmente sono soprattutto legate alla sintassi specifica del linguaggio. Molti studi e proposte sono state fatte in relazione a questo aspetto. Se analizziamo dal punto di vista storico i corsi di introduzione alla programmazione si evidenzia una evoluzione nel campo dei linguaggi utilizzati.

- linguaggi grafici come i *diagrammi a blocchi*;

- linguaggi "artificiali" come ad esempio i vari *pseudolinguaggi* derivanti dalla "traduzione" nella lingua parlata di reali linguaggi di programmazione. Un esempio sono costrutti quali `'se ... allora ... altrimenti' (if`

`... then ... else`) con ulteriori semplificazioni sintattiche per quanto riguarda la suddivisione in blocchi del codice;

- linguaggi di programmazione nati a scopo didattico come ad esempio LOGO e Pascal;
- veri e propri linguaggi di programmazione professionali scelti di volta in volta in relazione alla loro diffusione nel mondo informatico (Basic, C, C++), anche se la misurazione della popolarità di un linguaggio è in realtà un argomento dibattuto e non facile da definire, o al fatto di proporre uno specifico paradigma di programmazione (Java) o caratterizzati da una sintassi più semplice (Python).

In alcuni casi, si pensi per esempio alla formazione scolastica in scuole superiori ad indirizzo informatico o a corsi universitari CS1, la scelta del linguaggio professionale è condizionata dall'utilizzo di questo in corsi di approfondimento successivi e questo va a discapito dell'introduzione graduale dei concetti principali della programmazione che vengono 'mischiati' con problematiche implementative specifiche dei linguaggi stessi.

È da notare anche, fortunatamente in casi sporadici, una sorta di 'sadismo' da parte di docenti *informatici di ere precedenti* i quali ritengono che anche i *nuovi informatici* debbano farsi le ossa e scontrasi con problemi legati a tecnologie hardware e software ormai obsolete.

La diffusione di un linguaggio può derivare inoltre dalle scelte politiche di vari governi nazionali nel supportare programmi educativi di alfabetizzazione informatica, ne è un esempio la popolarità di Scratch [22], un linguaggio di programmazione progettato per i bambini, la cui popolarità è nettamente cresciuta nell'ultimo periodo ponendolo al quattordicesimo posto nella classifica dei linguaggi di programmazione più popolari secondo l'indice TIOBE dell'ottobre 2017[1].

Se prendiamo in considerazione i linguaggi citati in precedenza notiamo che, se si eccettuano i diagrammi a blocchi e l'esempio di Scratch, questi sono tutti di tipo testuale e obbligano il programmatore a scontrarsi subito con sintassi più o meno complesse.

[1] www.tiobe.com/tiobe-index

### 2.2.1 Il dibattito

Il dibattico relativo alla scelta del linguaggio da utilizzare nei primi corsi di programmazione è un tema discusso da molto tempo, ciò è desumibile anche dalle citazioni e dai titoli degli articoli di questa breve rassegna:

- 1976:

  > The selection of languages for use as pedagogical aids in the teaching of computer science is still a big issue at most universities.

  *Selecting languages for pedagogical tools in the computer science curriculum* [23]

- 1979:

  > What is a good programming language for beginners?

  *First programming language: Consequences* [24]

- 1982:

  > With the diversity of high-level programming languages available, selecting the "right" one for a computer science curriculum or course can be a befuddling process.

  *Selecting the "right" programming language* [25]

- 1986:

  > Which computer languages should we be teaching our students - and why?

  *The future of computer languages: implications for education* [26]

- 1989:

  > There is increasing discussion about the primary programming language used for undergraduate courses in Computer Science.

*Never mind the language, what about the paradigm?* [27]

### 2.2.2 Le motivazioni

Ampio è anche il dibattito relativo alle motivazioni di tali scelte:

> The choice of an introductory programming language is probably the most significant of the factors that will shape the competency of the next generation of computer users

*A Brief History of Choosing First Programming Languages* [28]

> Many years of experience have taught us that there is no perfect language for CS1. The choice is often almost a matter of religious fervor, rather than any specific language-related feature(s)

*Cs1: perspectives on programming languages and the breadth-first approach* [29]

> "the most striking trend in the field of programming languages" in the 1980s had "been the rise of paradigms, of which the object-oriented paradigm is the best-known."

*The evolution of the programming languages course* [30]

Negli anni '70 si diffondono i linguaggi di programmazione ad alto livello e nelle università si inizia quindi a discutere e si deve decidere se continuare a insegnare la programmazione utilizzando i linguaggi assembly o se passare a questi linguaggi basati su un più alto livello di astrazione. Nel secondo caso si deve poi decidere se privilegiare linguaggi nati espressamente per la didattica (Pascal) o utilizzati nell'industria (COBOL, Fortran, Basic).

Per dare un esempio dell'evoluzione dei linguaggi utilizzati nei corsi universitari elenchiamo qui quelli proposti nel corso introduttivo alla programmazione al corso di laurea in Informatica dell'Università di Pisa:

- 1969-1977 Flow chart, CANE, Simulcane
- 1978-1982 AlgolW, PDP-11
- 1983-1995 Pascal
- 1996-2010 Java
- 2011- ML, C

## 2.3 La scelta del paradigma

Fra le varie tipologie di corsi abbiamo quindi approfondito l'analisi delle proposte che seguono il modello object-first in cui l'approccio iniziale alla soluzione di problemi concerne l'utilizzo di oggetti e analizza l'interazione fra questi prima ancora di introdurre concetti tipici della programmazione imperativa (selezione e iterazione).

Una delle esperienze che riteniamo più significative a riguardo è relativa a Bluej[2] che non è solo un ambiente di sviluppo didattico che mette a disposizione strumenti per interagire con gli oggetti in modo semplice e visuale, ma è lo strumento utilizzato per una metodologia didattica proposta dagli stessi autori e caratterizzata dal modello object-first [31] .

### 2.3.1 Object-first

La scelta dell'approccio object-first è dettata da due motivazioni principali: la prima valuta il fatto che oggi il paradigma ad oggetti caratterizza la maggior parte delle applicazioni informatiche, la seconda deriva dalla difficoltà mostrata dagli studenti già *esperti* in programmazione imperativa nell'utilizzare altri paradigmi di programmazione: il cosiddetto *paradigm switch*.

Dalla nostra esperienza didattica nell'analizzare codice prodotto dagli studenti per risolvere problemi di una certa complessità e dalla letteratura dedicata a questo argomento si evidenzia come molto spesso emergano comportamenti scorretti (*antipattern*

[2]www.bluej.org

*di programmazione*) derivanti soprattutto dal fatto che i concetti di progettazione e modellizzazione vengono normalmente introdotti in corsi universitari rivolti a studenti che hanno già sviluppato un loro approccio personale nell'affrontare i problemi. Da questa considerazione nasce la proposta di un approccio didattico Model-First in cui, anche se in modo semplificato, i concetti basilari di software engineering entrano a far parte dei corsi introduttivi.

Spesso accade che in questi corsi introduttivi si semplifichi eccessivamente il processo di programmazione al fine di renderlo accessibile agli studenti con la conseguenza di dare poco peso all'analisi, al design e alle procedure di testing [32].

Negli ultimi anni OOP è diventato uno dei paradigmi di programmazione più utilizzati [33] ed è largamente utilizzato sia a livello educativo che professionale, su questo fatto tutti, o quasi tutti, sono concordi, mentre il problema che viene spesso messo in discussione è relativo al momento in cui introdurre questo concetto all'interno dei corsi di programmazione.

Nella classificazione dei corsi introduttivi all'informatica [9] si individuano vari modelli, tra questi alcuni sono relativi al paradigma di programmazione adottato: *imperative-first, object-first, functional-first*.

Per molto tempo OOP è stato argomento di corsi avanzati e oggi il dibattito relativo al *paradigm shift* è ancora in corso, ma negli ultimi anni nei corsi introduttivi di informatica in molte università si è verificato un passaggio dal modello procedurale a quello object-oriented.

La metodologia object-first, come dicevamo, stabilisce che gli studenti, già nella prima parte dei corsi introduttivi alla programmazione, siano spinti a pensare in termini di oggetti e affrontino i concetti fondamentali del paradigma object-oriented (OO) quali l'incapsulamento, l'ereditarietà e il polimorfismo.

Imparare a programmare nello stile OO risulta difficile per molti studenti se questi hanno già avuto esperienze di programmazione in stile procedurale, ma in realtà la difficoltà non risiede nel paradigma ma nel passaggio da un paradigma ad un altro, quello che abbiamo definito come *paradigm switch*.

*"It is the switch that is difficult, not object-orientation"*

[33]

Non è tanto la difficoltà nell'apprendere i costrutti sintattici per definire classi o istanziare oggetti, ma ciò che viene a mancare è il corretto approccio mentale per utilizzare classi e oggetti in modo efficace.

Nonostante voci contrastanti che suggeriscono attenzione a proposito dei "miti dell'object-oriented" [34] molte esperienze mostrano come il livello di difficoltà che gli studenti incontrano nell'affrontare i concetti fondamentali della disciplina sia sostanziamente equiparabile in corsi che utilizzano il modello Imperative-First rispetto ad altri basati su quello object-first.

Un'analisi comparativa [35] mostra che gli studenti che hanno iniziato il loro percorso didattico partendo dalla programmazione ad oggetti danno risultati migliori quando si tratta di progettare software per la soluzione di problemi complessi.

Altri studi [36] [37] [38] presentano esperienze in cui l'approccio object-first ha effettivamente portato a notevoli miglioramenti nei risultati raggiunti dagli studenti diminuendo drasticamente la percentuale degli insuccessi.

### 2.3.2 Opinioni contrastanti

*"Your Code: OOP or POO?"*

– Jeff Atwood (Stack Overflow co-founder)

Non tutto il mondo scientifico, come spesso accade, si trova in accordo nell'apprezzare il paradigma di programmazione ad oggetti. Nel suo blog[3] Jeff Atwood afferma che spesso lo sviluppo di software object-oriented risulta essere una *tassa sulla produttività dei programmatori* e, nonostante la gestione degli oggetti risulti essere oggi la spina dorsale di ogni nuovo linguaggio di programmazione, questo paradigma in realtà comporti tanti problemi almeno quanti ne riesca a risolvere. In particolare la verbosità di alcuni linguaggi rende poco produttivo il lavoro di chi sviluppa software. In modo ironico Atwood definisce *Object Happiness* l'atteggiamento di chi sente la necessità di applicare i principi del design object-oriented a semplici, piccoli e banali progetti sprecando inutilmente tempo per creare inutili gerarchie di interfacce e classi

[3] blog.codinghorror.com/your-code-oop-or-poo

astratte. Pur non negando l'importanza di questo paradigma, mette quindi in guardia i programmatori a fare attenzione a quelle "sette" che egli definisce *OO True Beleavers*.

### 2.3.3 Design-First

Nell'ambito della discussione sulla metodologia didattica vari autori propongono il modello Design-First e pongono l'accento soprattutto su problematiche legate alla progettazione piuttosto che sugli aspetti sintattici dei linguaggi [39].

I fautori del modello Model-First affermano che prima della programmazione debba essere insegnata agli studenti la modellizzazione poiché quest'ultima non è legata a specifici aspetti tecnologici ed inoltre risulta utile per lo studio di varie discipline.

Nei corsi di *Ingegneria del software* si insegna che è possibile affrontare direttamente la codifica solo di piccoli problemi per i quali l'operazione di modellizzazione avviene in modo informale a livello mentale, ma ciò non risulta possibile se si iniziano ad affrontare problemi più complessi. Ma questi corsi normalmente non sono collocati nei primissimi anni dei percorsi di studio e questo fa sì che gli studenti, al momento in cui affrontano questi argomenti, abbiano già sviluppato un loro proprio modo di operare, spesso purtroppo, non corretto (*bad coding habits*).

#### 2.3.3.1 Unified Modeling Language

I diagrammi UML (Unified Modeling Language) sono comunemente utilizzati nei corsi introduttivi alla programmazione per insegnare il paradigma orientato agli oggetti. Sono disponibili molti software commerciali o di libero utilizzo che forniscono la possibilità di *disegnare* diagrammi UML, ma, purtroppo, la maggior parte di questi sono altamente professionali e troppo difficili da utilizzare da parte degli studenti in questo ambito [40]. C'è quindi la necessità di software espressamente progettato e realizzato con finalità didattiche che sia user-friendly anche a discapito di offrire un numero minore di funzionalità. Se gli studenti hanno a disposizione uno strumento che consente loro di esprimere facilmente la progettazione ad un livello di astrazione elevato allora cominciano a pensare e a codificare in termini di costrutti di

alto livello. Questo permette loro di concentrarsi soprattutto su temi di progettazione orientata agli oggetti e meno su problemi legati alla sintassi. Questo, come vedremo, è una delle motivazioni che ci hanno spinto a sviluppare il tool di progettazione Object Oriented Puzzle Programming.

## 2.4 L'ambiente didattico per lo sviluppo del software

Nelle varie esperienze descritte nella letteratura scientifica, si mettono in evidenza i vantaggi derivanti dall'utilizzo nei corsi introduttivi al coding di un ambiente di sviluppo realizzato specificamente per scopi educativi.

L'idea su cui abbiamo lavorato e che andremo a presentare ha come obiettivo quello di fornire metodi e strumenti per eliminare o quanto meno semplificare le difficoltà relative allo sviluppo di semplici applicazioni, difficoltà spesso accentuate dagli aspetti tecnici e sintattici presenti nell'uso dei più diffusi linguaggi di programmazione e ambienti di sviluppo.

Lo strumento didattico che proponiamo è orientato alla programmazione ad oggetti e implicitamente suggerisce un approccio object-first nel quale si individuano due momenti importanti [41] [42]

- l'utilizzo di oggetti: i primi approcci degli studenti con la programmazione sono relativi all'uso di oggetti già implementati
- la creazione di classi: gli studenti definiscono e implementano classi con l'obiettivo poi di istanziare e far interagire oggetti

Un esempio di ambiente di sviluppo che è di supporto a una metodologia didattica è dato da Bluej [31], un IDE didattico che favorisce l'utilizzo di oggetti anche a chi non possiede conoscenze di programmazione, ha un'interfaccia utente molto semplice e presenta visivamente gli oggetti e le loro caratteristiche oltre a fornire la possibilità di operare in modo interattivo con questi. La rappresentazione delle classi di un progetto e delle loro relazioni in un modello simile al class diagram UML è un ausilio alla progettazione; quello che manca in questo ambiente di sviluppo è uno strumento

che permetta di definire le caratteristiche di una classe (oggetti, attributi, metodi, costruttori, ...) in modo semplice attraverso un'interfaccia grafica e che permetta poi di generare l'ossatura del codice (nel nostro caso codice Java). Strumenti di questo tipo sono presenti, spesso come plug-in, in vari ambienti di sviluppo professionale (es. Eclipse), ma proprio per la natura professionale di questi IDE introducono una serie di aspetti fuorvianti per un primo approccio didattico.

## 2.5 Gaming e Computational Thinking

La critica che viene spesso mossa ai giochi dell'era digitale è che, rispetto ai giocattoli tradizionali, questi soffocherebbero la creatività dei bambini. Ma l'idea di base del gaming è che siano i ragazzi stessi a inventarsi il proprio videogioco.

I primi ambienti proposti per l'introduzione al coding sono caratterizzati da un aspetto ludico, i bambini giocano e vincere ogni sfida significa risolvere problemi, la scelta del gaming è quindi strategica perché consente di attirare l'attenzione anche dei più piccoli.

I primi problemi proposti sono semplici: evitare un ostacolo ed evitare di farsi catturare da uno dei personaggi "cattivi" della storia; ma già da questi primi approcci è necessario impegno per capire quale possa essere la possibile soluzione, si deve ragionare passo passo sul modo migliore per raggiungere l'obiettivo. Formulare la soluzione in realtà significa aver scritto, anche se inconsapevolmente, righe di codice informatico. Questo porta, anche i più piccoli, a pensare meglio e in modo creativo, stimola la loro curiosità attraverso quello che apparentemente può sembrare solo un gioco.

Nell'ampio dibattito relativo all'introduzione dei concetti principali del *pensiero computazionale* e del *coding* si inserisce la discussione relativa al contesto applicativo dei primi programmi sviluppati dai *giovani programmatori*. In questo ambito si presentano differenti opinioni [43] [44] relative alle varie tipologie di corsi introduttivi all'Informatica. In generale questi corsi, di qualsiasi livello scolastico, sono considerati molto difficili dagli studenti e spesso presentano alte percentuali di insuccesso. Nell'articolo *Learning Programming Patterns Using Games* [45] si afferma

che c'è, ancor oggi, la necessità di sviluppare nuovi metodi di insegnamento per favorire l'apprendimento della programmazione. Gli studenti che affrontano i primi passi in questo contesto trovano difficoltà legate soprattutto alla mancanza di capacità di astrazione e di creatività per definire la soluzione ai problemi.

Con il termine *gamification* si intende l'utilizzo di elementi mutuati dai giochi e da tecniche di game design in contesti esterni ai giochi. Negli ultimi anni il concetto di gamification è stato ampiamente descritto, ad esempio in [46, 47, 48]. In [47], gli autori riportano uno studio sistematico sull'applicazione della gamification all'istruzione.

La gamification è una tendenza emergente anche in realtà come il business, il marketing, l'ecologia e il benessere. Nella letteratura scientifica vengono presentate molte esperienze e sperimentazioni sull'utilizzo del gioco in contesto didattico [49], anche in corsi orientati al computational thinking e in particolare al coding [33].

Molti studenti percepiscono la scuola tradizionale come inefficace e noiosa: ma programmare è noioso? Sotto certi aspetti può risultare difficile, ma le difficoltà possono essere superate se si adottano adeguate strutture e metodologie per migliorare l'apprendimento. Ad esempio, molti strumenti sono progettati per facilitare lo realizzazione di giochi [50, 51, 52]. Il gaming fornisce un ambito all'interno del quale gli studenti si muovono con maggiore dimestichezza e interesse e questo favorisce il graduale superamento di queste difficoltà.

La realizzazione di un gioco permette di sviluppare la creatività e l'astrazione, due delle abilità più importanti su cui insistere nell'ambito di corsi all'interno dei curricula informatici. Il fatto che il "problema" da risolvere sia la realizzazione di un gioco risulta spesso più stimolante e accattivante se confrontato con problemi relativi ad ambiti differenti. Dal punto di vista educativo, l'utilizzo dei giochi nel processo di insegnamento-apprendimento costituisce quindi un elemento motivazionale che può aiutare il processo di costruzione della conoscenza in modo maggiormente personalizzato.

Vari autori [53] e vari docenti [54] [55] hanno formulato nuove proposte didattiche basate sullo sviluppo di giochi per cercare di porre rimedio alle difficoltà evidenziate dagli studenti nell'apprendimento dei concetti iniziali della programmazione e per permettere loro di affrontare in modo ludico problemi non certo banali.

Vassilev e Mutev [56] analizzano le difficoltà che si incontrano nell'insegnamento nei corsi universitari CS1 per gli studenti con curriculum informatico e propongono l'utilizzo di una libreria dedicata al gaming, da loro sviluppata, che permette di gestire oggetti virtuali che operano all'interno di un ambiente virtuale di gioco 3D. Da una prima analisi presentata dagli autori risulta che nello studio gli studenti risultano maggiormente motivati dall'uso della libreria e dallo sviluppo di applicazioni di gioco.

### 2.5.1 Game-First

Il dibattito sull'approccio object-early o object-late nell'insegnamento della programmazione è presentato in [33]. Dopo aver discusso il problema del *paradigm switch*, la conclusione, come abbiamo visto, è che imparare programmare utilizzando il paradigma orientato agli oggetti risulta piuttosto difficile dopo aver sviluppato vari programmi in stile procedurale.

Un punto di vista più radicale è quello di Leutenegger e Edgington [55] i quali affermano che una metodologia di successo per l'insegnamento della programmazione non deve essere basata su concetti di object-first o object-late ma piuttosto sull'approccio *game-first*.

In letteratura sono presenti varie ricerche inerenti ambienti che semplificano il processo di progettazione e realizzazione di giochi [56, 57, 58] e pongono l'accento soprattutto sul processo di sviluppo e apprendimento, piuttosto che sui prodotti, i giochi, effettivamente realizzati. La creazione di giochi non è quindi lo scopo, ma lo strumento per guidare gli utenti a imparare i concetti fondamentali della programmazione.

#### 2.5.1.1 Strategie

Le strategie relative all'insegnamento del coding sono collocabili in una gamma che ha come estremi: l'apprendimento intensivo (*intensive learning*), che è didattico e concentrato, e l'apprendimento estensivo (*extensive learning*), una sorta di processo che si verifica senza un obiettivo specifico. Troppo spesso troviamo, nelle aule sco-

lastiche e nei corsi universitari, esempi di apprendimento intensivo in cui le lezioni e i libri di testo analizzano completamente molti aspetti teorici, prima di scrivere la prima linea di codice. Diametricamente opposte sono le esperienze di apprendimento estensivo: in CodeCombat[4], ad esempio, esiste un ambiente di gioco in cui i giocatori devono scrivere il codice corretto per passare da un livello a quello successivo e questo incoraggia la necessità di apprendere, spronando i giocatori a diventare programmatori. Gli autori di CodeCombat sposano a pieno il concetto di apprendimento estensivo:

> *It is the computer game that teaches programming Players don't learn to be engineers by playing CodeCombat: they learn the more important foundational skills, like formal syntax, conditional logic, and variables*

La nostra strategia didattica, applicata per esempio nella progettazione dello stage che andremo poi a descrivere, si colloca in una posizione intermedia, adottando alcuni principi dell'extensive learning, ad esempio "code to stay alive", mutuato da "fight to be alive", che è il concetto principale di molti giochi. Proponiamo attività più pratiche, "giocando" con il codice, ma senza perdere di vista aspetti teorici fondamentali. A nostro parere, è necessario introdurre concetti teorici, ma allo stesso tempo metterli in pratica.

[4]codecombat.com

# Chapter 3

# Progetti per la diffusione del coding

*"Every student in every school should have the opportunity to learn computer science"*

– Code.org®.

## 3.1 Coding e Computational Thinking

Parliamo di *Computational Thinking* come approccio ai problemi e alla loro soluzione. Con il *coding* bambini e ragazzi sviluppano il pensiero computazionale e l'attitudine a risolvere problemi più o meno complessi. Non imparano solo a programmare ma programmano per apprendere. Questo approccio mette la programmazione al centro di un percorso dove l'apprendimento, già a partire dai primi anni scolastici, percorre strade nuove e fa parte di un più ampio progetto che mira ad abbattere le barriere dell'informatica e stimola un approccio mirato alla risoluzione dei problemi. L'aspetto più importante, in questo contesto, è che mediante i primi approcci alla programmazione i ragazzi imparano a raggiungere un obiettivo.

In qualche modo possiamo considerare il coding come palestra del pensiero

computazionale che va allenato già da piccoli e dai primi anni scolastici. *Giocare* con la programmazione è il modo migliore per sviluppare il pensiero computazionale.

Tutte le campagne volte alla diffusione del pensiero computazionale si stanno concentrando sul coding. I bambini *giocano* a programmare e in questo modo usano la logica per risolvere i problemi e sviluppano un processo logico-creativo che consente di scomporre un problema complesso in più parti per affrontarlo e risolverlo in modo più semplice. Anche i bambini sono quindi allenati a risolvere problemi "da grandi", e diventare soggetti attivi della tecnologia, creando un piccolo videogioco e realizzando le loro storie in pochissimo tempo.

## 3.2 Linguaggi e ambienti didattici di programmazione

Governi, istituzioni scolastiche pubbliche e private, associazioni ed emanazioni di fondazioni sono oggi impegnate in progetti per la diffusione attraverso il coding del pensiero computazionale.

Nuovi linguaggi e nuovi ambienti didattici vengono proposti per favorire il primo approccio alla programmazione. La quasi totalità dei progetti è rivolta a giovani di età scolare o prescolare quindi anche i nuovi linguaggi e gli ambienti di programmazione sono caratterizzati da semplicità di utilizzo e da una ambientazione ludica. Presentiamo qui una rassegna, non certo esaustiva, di quelli che riteniamo più importanti e che hanno influenzato i nostri progetti OOPP e codOWood.

### 3.2.1 Logo

Logo[1] può essere indubbiamente definito il progenitore dei linguaggi e degli ambienti didattici rivolti ai bambini per il loro primo contatto con la programmazione.

Ideato e realizzato a metà degli anni '60 al MIT da Seymour Papert [4] il linguaggio Logo, un dialetto del Lisp, è stato progettato come strumento per l'apprendimento basato su semplici istruzioni finalizzate a far muovere una tartaruga sullo schermo. Originariamente la tartaruga era un piccolo e semplice robot (fig. 3.1) che si muoveva

[1] el.media.mit.edu/logo-foundation

Figure 3.1: Logo - la tartaruga negli anni ’60

sul pavimento in base ai comandi ad esso impartiti dai bambini, poi, con la diffusione dei monitor, il linguaggio Logo divenne più accessibile e negli anni ottanta ne vennero realizzate versioni per personal computer (Apple II, Commodore 64, MSX e ZX Spectrum) utilizzate a scopi didattici, spesso per lo studio della geometria.

Nell’arco dei cinquant’anni dal suo debutto sono state create centinaia di versioni di Logo e il linguaggio è stato un punto di riferimento per le ricerche nel campo didattico che hanno portato alla realizzazione della maggior parte degli ambienti dedicati alla didattica del coding.

### 3.2.2 Scratch

Partendo da un progetto sviluppato nel 2003 dal gruppo Lifelong Kindergarten del MIT Media Lab, Scratch[2] è stato il modello di riferimento per molti altri linguaggi dedicati all’insegnamento del coding.

Si tratta di un linguaggio, basato sul block programming, che presenta un insieme di blocchi collegabili fra loro per gestire il comportamento dei personaggi all’interno

[2]scratch.mit.edu

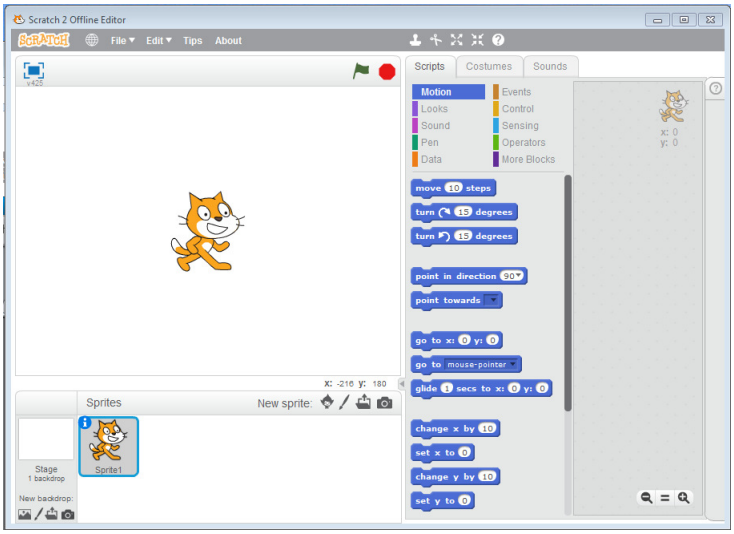



Figure 3.2: Scratch

di una storia (fig. 3.2).

Basato principalmente sul paradigma imperativo si è via via evoluto e, a partire dalla versione 2.0, c'è la possibilità di definire blocchi utente da utilizzare poi all'interno del progetto.

La diffusione di Scratch negli ultimi anni lo ha fatto salire fino al quattordicesimo posto nella classifica dei linguaggi di programmazione più popolari secondo l'indice TIOBE[3] dell'ottobre 2017.

I progetti realizzati con Scratch (storie interattive multimediali, giochi, animazioni) possono essere condivisi e sviluppati in collaborazione con altri utenti. La comunità per l'apprendimento collaborativo e creativo di Scratch oggi vanta ben più di 25 milioni di progetti condivisi e nella community ScretchEd[4] dedicata a docenti e studenti è disponibile una grande quantità di materiale utile per una didattica del coding nei vari ordini di scuola.

[3] www.tiobe.com/tiobe-index/
[4] scratched.gse.harvard.edu

### 3.2.3 Scratch Junior


Figure 3.3: Scratch Junior

Dalla esperienza di Scratch deriva il linguaggio per i più piccoli (dai 5 ai 7 anni) Scratch Junior[5] sempre basato sulla composizione di blocchi operativi che in questo caso sono più semplici e totalmente privi di testo.

I blocchi, con le loro icone esplicative, sono utilizzati per far muovere i personaggi sullo sfondo delle varie scene che compongono la storia animata dai bambini che, in questo modo, imparano a risolvere problemi, a creare progetti e ad esprimersi in modo creativo.

A differenza di Scratch che è un'applicazione multipiattaforma per personal computer, Scratch Junior è una app per dispositivi Android e iOS.

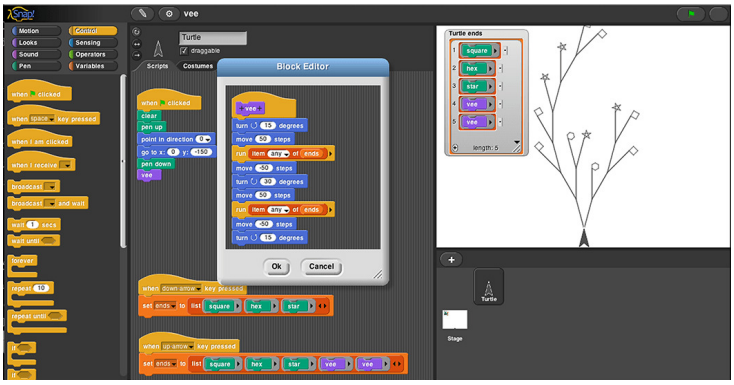



Figure 3.4: Snap!

### 3.2.4 Snap!

Snap![6], chiamato nelle prime versioni BYOB (Build Your Own Blocks) è un linguaggio di programmazione visuale ed è un'estensione di Scratch. Oltre alle funzionalità di Scratch permette infatti di definire nuovi blocchi e amplia la potenza del linguaggio che non ha più come target di riferimento bambini o ragazzi ma studenti delle scuole superiori e delle università.

Snap! è implementato in JavaScript e opera all'interno di un browser, non necessita quindi di procedure di installazione in locale.

### 3.2.5 Alice 3D

Alice[7] è una applicazione gratuita multipiattaforma che permette di creare animazioni tridimensionali e costruire storie in modo semplice mediante block programming.

Alice motiva l'apprendimento attraverso l'esplorazione creativa ed è progettato allo scopo di insegnare i principi della programmazione orientata agli oggetti. Sul sito del progetto è possibile consultare e scaricare materiale per l'utilizzo di questo linguaggio, sono stati inoltre pubblicati vari testi per la didattica del coding basati

[5] www.scratchjr.org
[6] snap.berkeley.edu
[7] www.alice.org

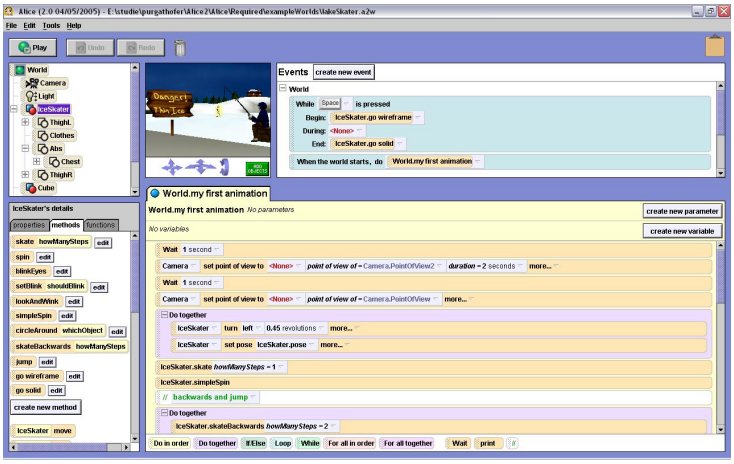


Figure 3.5: Alice

sull'utilizzo di Alice [59]. In letteratura troviamo molte sperimentazioni di questo linguaggio in ambito scolastico e fra queste vogliamo citare il testo di Daly [60].

### 3.2.6 BlueJ

BlueJ[8] è un software gratuito multipiattaforma ed è un ambiente didattico di sviluppo integrato per il linguaggio Java. Fa parte di un progetto più generale a supporto dell'insegnamento della programmazione orientata agli oggetti ed è correlato da un testo [31] per corsi CS1 basati sulla metodologia object-first.

Permette una interazione diretta con gli oggetti e con le classi e i concetti della programmazione object-oriented sono sono rappresentati graficamente (fig. 3.6) in modo da favorirne la comprensione da parte degli studenti.

### 3.2.7 Swift

Swift[9] è un linguaggio di programmazione creato da Apple per facilitare lo sviluppo di app per iOS, macOS, Apple TV, Apple Watch e Linux. E' un linguaggio potente

[8] www.bluej.org
[9] swift.org

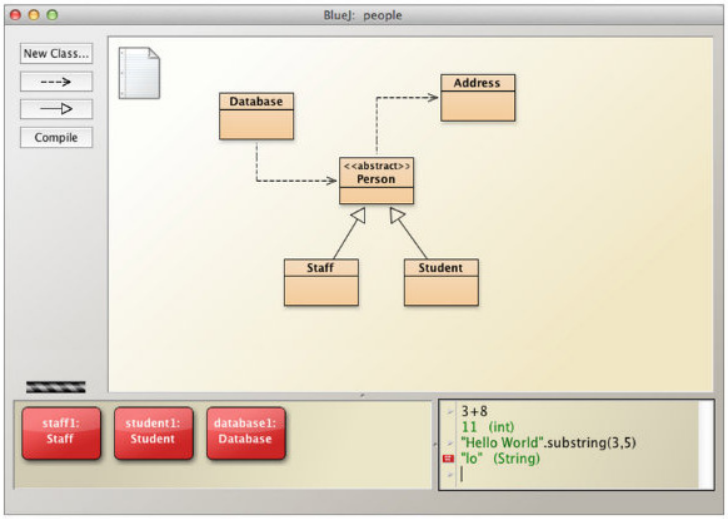



Figure 3.6: BlueJ

e semplice da usare che ha attirato l'interesse di scuole e università che lo hanno introdotto nei loro corsi di programmazione [61] e molti di questi corsi sono oggi presenti su iTunes U.

Swift è distribuito in licenza open source Apache 2.0 ed è stata creata una community per permettere agli utenti di contribuire allo sviluppo del progetto.

Apple ha inoltre realizzato Swift Playground, una app iPad per bambini che semplifica l'introduzione alla programmazione e rende l'apprendimento del linguaggio Swift interattivo e giocoso. Apple offre inoltre una libreria con lezioni di programmazione, un'interfaccia interattiva che incoraggia gli studenti, anche coloro i quali non hanno conoscenze pregresse di coding a esplorare l'uso di Swift inviando comandi, creando funzioni, imparando a usare loop, variabili, ecc. Questo favorisce progressivamente la loro acquisizione di competenze e la loro sicurezza nel gestire il codice.

Swift è un linguaggio di programmazione object-oriented testuale e, nonostante sia derivato da un linguaggio più complesso come Objective-C, è caratterizzato da una semplice sintassi.

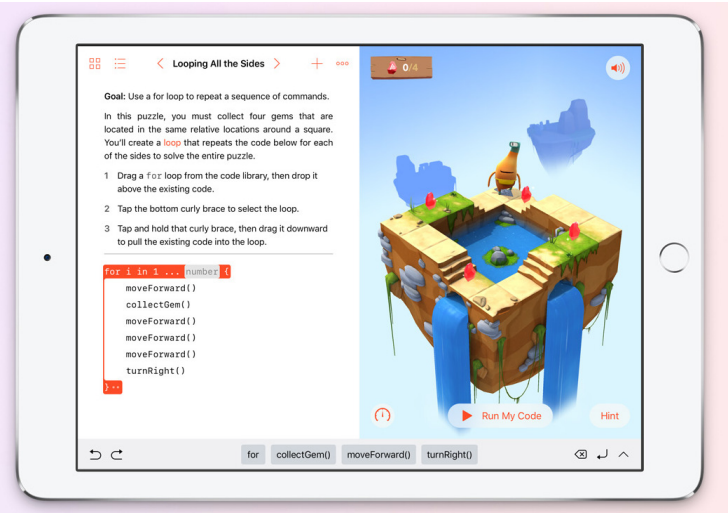



Figure 3.7: Swift Playgrounds

### 3.2.8 Hopscotch

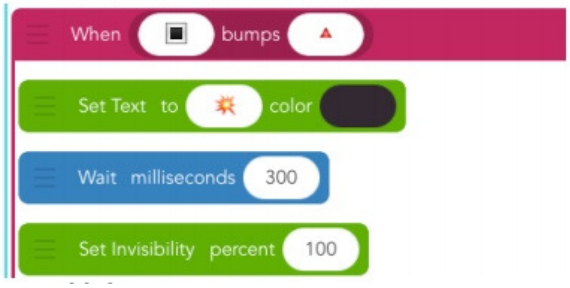



Figure 3.8: Esempio di programma Hopscotch

Con Hopscotch[10] è possibile creare giochi e animazioni trascinando e collegando fra loro nell'area di lavoro mattoncini colorati (fig. 3.8) che rappresentano "pezzi" di codice . Si tratta di un'app per iPhone e iPad che stimola la creatività dei bambini insegnando loro, al tempo stesso, i principi fondamentali della programmazione. È

[10]www.gethopscotch.com

indirizzata a ragazzi a partire dagli 8 anni anche se in realtà può essere utilizzata da bambini più piccoli, per i quali risulta una versione semplificata, e quindi più accessibile, rispetto al più noto linguaggio Scratch.

Analogamente a Scratch la caratteristica principale di questa app è l'interazione con i vari personaggi che vengono inseriti nel gioco e la possibilità di definire le azioni da intraprendere nel caso si verifichino vari eventi. La risposta agli eventi avviene ad esempio gestendo il tocco su di un personaggio o i segnali ricevuti dall'accelleratore del dispositivo e può comprendere spostamenti nell'area del gioco, modifica dell'aspetto del personaggio, emissione di suoni ecc.

### 3.2.9 Lightbot

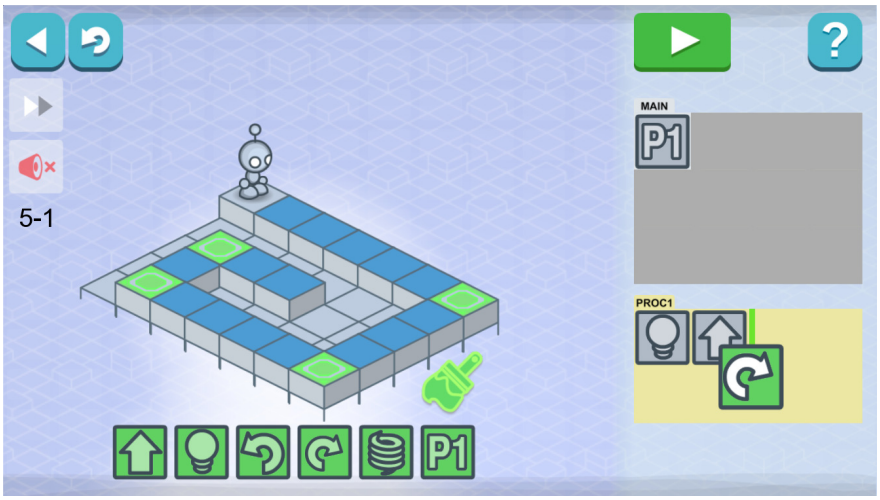



Figure 3.9: L'ambiente di Lighbot

Si tratta di una app[11] multipiattaforma per iOS e Android pensata per insegnare i fondamenti della programmazione a bambini di scuola elementare e media. L'obiettivo è far muovere un piccolo robot sulla scacchiera (fig. 3.8) impartendo i vari comandi assemblando sequenze di elementi grafici. In questo modo si riescono ad affrontare i

[11] lightbot.com

fondamenti della programmazione, come “procedure” e “loop” senza dover scrivere una sola riga di codice [62]. Si tratta di un prodotto commerciale distribuito con traduzioni in varie lingue dal costo molto contenuto che è stato usato da decine di migliaia di docenti in tutto il mondo.

### 3.2.10 Code Monster

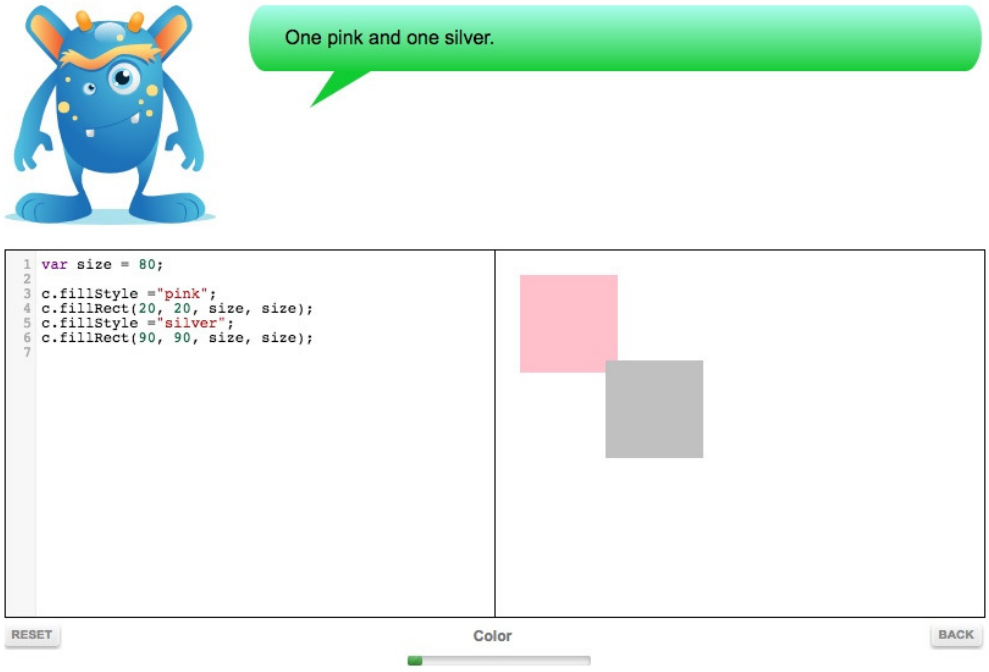



Figure 3.10: Code Monster

Code Monster, Code Maven e Game Maven[12] sono ambienti tutoriali interattivi rivolti a bambini dai 9 ai 14 anni (Code Monster) (fig. 3.10), ragazzi e adulti “curiosi” (Code Maven e Game Maven) (fig. 3.10) che permettono di “giocare” con il codice guidati da un avatar che suggerisce passo passo le varie istruzioni. Ogni interazione con il codice è immediatamente interpretata, eseguita e visualizzata attraverso

[12]www.crunchzilla.com

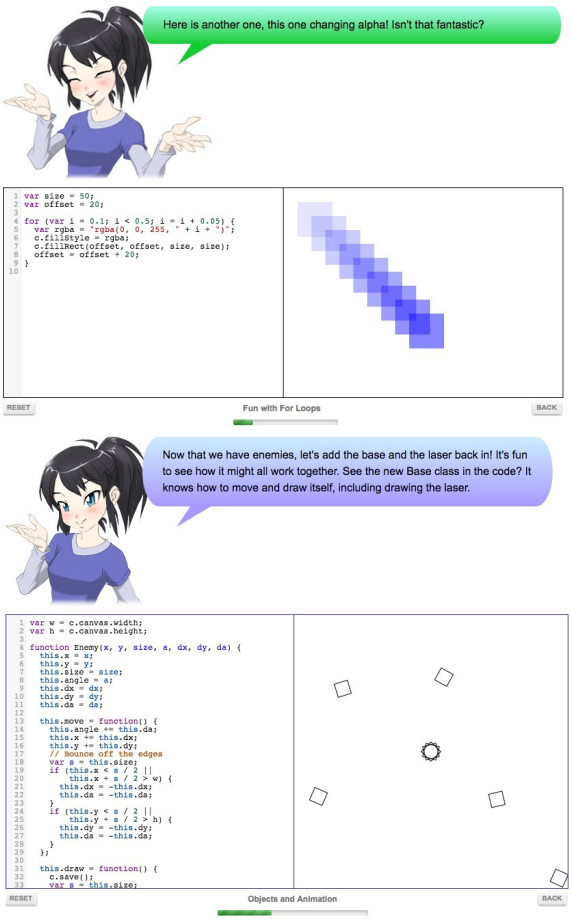



Figure 3.11: Code Maven e Game Maven

una bipartizione dello schermo. I giocatori/programmatori imparano passo passo a costruire grafici, a realizzare animazioni, frattali e semplici giochi. Non c'è nessuna descrizione preliminare del linguaggio che è basato sul paradigma object-oriented, le sue caratteristiche vengono apprese partendo dagli esempi e modificandoli di volta in volta per raggiungere i singoli obiettivi intermedi.

### 3.2.11 Kodable

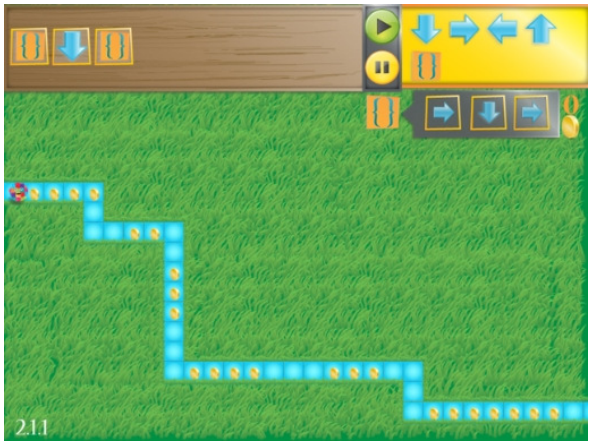



Figure 3.12: Kodable

Kodable[13] è una app per iPad rivolta ai bambini a partire dai dai 5 anni con l'obiettivo di far loro apprendere in modo ludico i concetti basilari della programmazione. In un labirinto virtuale (*technomaze*) (fig. 3.12) si muovono palline pelose (*fuzz*) guidate da comandi impartiti dai piccoli programmatori che dispongono blocchi operativi nell'area di scrittura del codice (*command center*) al fine di recuperare il maggior numero di monete e stelline e arrivare alla fine del percorso.

I blocchi operativi, il linguaggio di programmazione di Kodable, sono costituiti da frecce che definiscono il movimento (su, giù, sinistra e destra) e da piastrelle colorate (cui associare un comando), questi blocchi vengono trascinati (drag & drop) e messi in sequenza nel command center.

I vari livelli del gioco sono dedicati all'apprendimento dei concetti chiave della programmazione:

- le sequenze di comandi
- le condizioni semplici e composte

[13] www.kodable.com

- i cicli (la ripetizione di una porzione di codice)
- le funzioni (un insieme di comandi rappresentati da un blocco colorato che può essere poi riutilizzato)

Oltre alla app gratuita per uso individuale ne esiste una versione concepita per uso in ambito scolastico (Kodable Class) arricchita con funzioni di monitoraggio dei progressi e del livello raggiunto dai singoli alunni.

### 3.2.12 MIT App Inventor

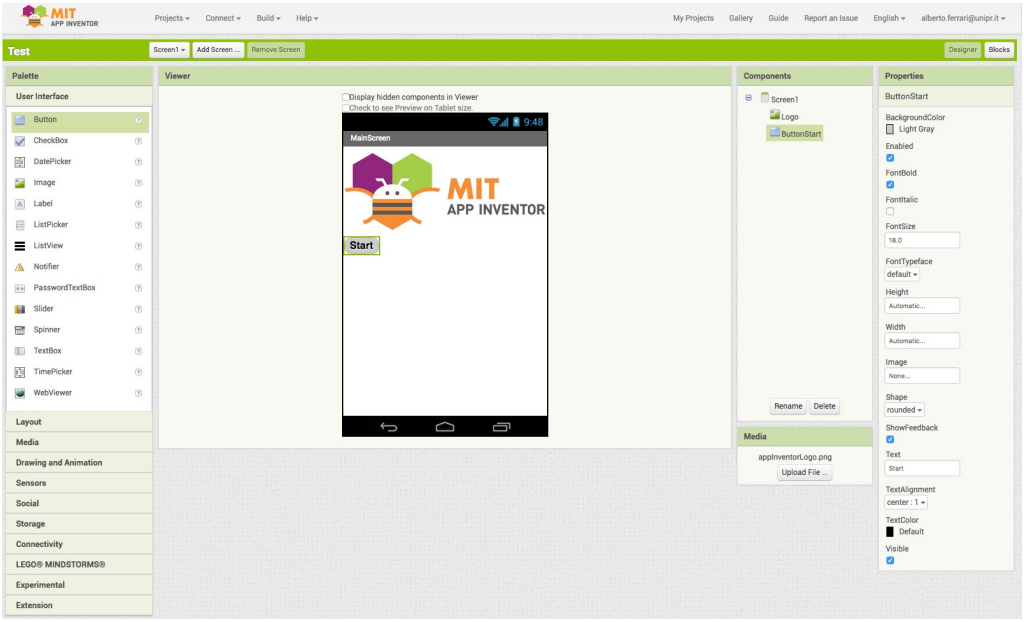



Figure 3.13: MIT App Inventor

App Inventor[14] è un ambiente di sviluppo che permette di realizzare applicazioni multimediali in ambiente Android ed è basato sul block programming. Creato da Google nel 2010 ora è gestito dal Massachusetts Institute of Technology.

[14]appinventor.mit.edu

L'ambiente di sviluppo (fig. 3.13) è dotato di un emulatore per testare direttamente le applicazioni che possono poi essere installate direttamente tramite WiFi o USB su periferica Android (smartphone o tablet).

L'utilizzo è molto semplice e intuitivo e l'utilizzo dei blocchi al posto di un linguaggio testuale facilita la creazione di applicazioni anche complesse in tempi significativamente inferiori se comparati agli ambienti di programmazione tradizionali.

La prima versione di App Inventor utilizzava per l'editor dei blocchi la libreria Java Open Blocks mentre le versioni più recenti sono applicazioni web in cui Open Blocks è stata sostituita dal framework Google Blockly e non necessita di alcuna installazione di componenti in locale.

La piattaforma utilizza i server di Google per il backup dei singoli progetti, per l'utilizzo di App Inventor è necessario quindi possedere un account Google.

Pur essendo basato sul block programming come molti ambienti dedicati alla didattica del coding per i giovani, questo strumento permette di sviluppare applicazioni anche complesse e non a caso è utilizzato anche in corsi universitari [63]

### 3.2.13 StarLogo

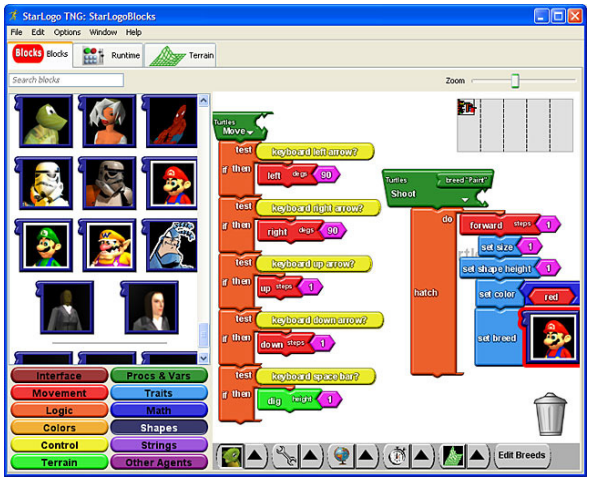



Figure 3.14: StarLogo TNG

StarLogo[15] è un'applicazione multipiattaforma che permette di creare giochi (fig. 3.14) e simulazioni tridimensionali ed è adatto, dal punto di vista didattico, per la comprensione di sistemi complessi.

StarLogo è un linguaggio di simulazione agent-based sviluppato presso il MIT Media Lab e fa parte del MIT Scheller Teacher Education Program.

Nel 2008 è stata rilasciata una nuova versione (StarLogo TNG - *The Next Generation*) che utilizza la grafica OpenGL e un linguaggio grafico basato su blocchi per aumentare la facilità d'uso e di apprendimento.

### 3.2.14 Google Blockly

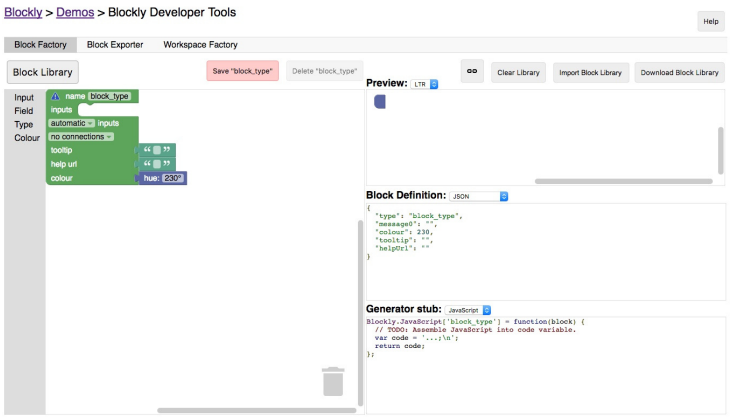



Figure 3.15: Google Blockly: Block Factory

Diversamente dai linguaggi presentati in precedenza Google Blockly[16] è una libreria JavaScript che mette a disposizione funzionalità per la creazione di blocchi visuali per applicazioni di block programming. Oltre alla versione web client-side che opera all'interno del browser sono disponibili anche versioni per dispositivi mobili (Android e iOS).

[15] education.mit.edu/portfolio_page/starlogo-tng/

[16] developers.google.com/blockly

Oltre a definire l'aspetto e i vincoli di collegamento fra i vari blocchi è possibile associare a ogni blocco il codice che questo andrà a generare (fig. 3.14). Il codice può essere definito per i linguaggi: JavaScript, Python, PHP e Dart.

Nato inizialmente con lo scopo di sostituire l'uso della libreria OpenBlocks in App Inventor, oggi è diventato quasi uno standard ed è utilizzato come base per molti progetti di block programming.

I tool di block programming *OOPP* e di tangible programming *codOWood* che abbiamo realizzato nella nostra università sono entrambi basati sulle API di Google Blockly.

## 3.3 Block Programming

Come si può evincere anche da questa rassegna di progetti, il tema del coding sta generando molto interesse e negli ultimi anni sono stati proposti diversi approcci per la diffusione del *Pensiero Computazionale*. In particolare, le esperienze più recenti relative all'introduzione alla programmazione sono state orientate prevalentemente ad un pubblico molto giovane. Fra i molti progetti che hanno come obiettivo quello di facilitare l'introduzione al coding possiamo citare il famoso progetto code.org [64], la piattaforma Raspberry Pi [65] e numerosi altri [51]. Una caratteristica condivisa da buona parte di questi progetti è l'utilizzo del *Block Programming* per semplificare il primo approccio alla programmazione, riducendo, e in qualche modo eliminando, le difficoltà di tipo sintattico.

Nell'approccio alla programmazione basato sul Block Programming vengono utilizzati editor visuali con cui gli utenti compongono fra loro blocchi operativi in modo da *scrivere* programmi in modo non testuale. Questi progetti sono orientati a una platea di giovanissimi o a persone in ogni caso non interessate a diventare professionisti dell'informatica.

Nei capitoli successivi verrà presentato il progetto OOPP che ha come target di riferimento gli studenti del primo anno di un percorso universitario in ambito scientifico che dovranno diventare esperti nel settore informatico. Vista la diversità del target, dovranno essere diverse le metodologie e gli strumenti utilizzati. L'idea

di un primo approccio al coding facilitato dall'assenza di problematiche sintattiche legate ai classici linguaggi di programmazione rimane però interessante. In relazione a questo aspetto abbiamo sperimentato la possibilità di espandere uno dei più diffusi strumenti di block programming (Google Blockly) per includere strumenti (blocchi) che permettano la programmazione ad oggetti.

Con la programmazione a blocchi è possibile programmare senza scrivere una sola riga di codice. È sufficiente spostare e ordinare in sequenza una serie di blocchi in una apposita area di lavoro, come se fossero i pezzi di un puzzle. A ogni blocco corrisponde un comando, una istruzione che non ha bisogno di essere digitata ma solo "incastrata" al blocco precedente. Negli ambienti basati sul Block Programming il processo di sviluppo di una applicazione è quindi basato sulla costruzione di puzzle che sono composti da blocchi che rappresentano gli elementi di base del *linguaggio*.

La programmazione a blocchi è probabilmente il modo più semplice e immediato per avvicinare i giovani al mondo del coding e rappresenta l'elemento iniziale del percorso di apprendimento della programmazione per tutti coloro che non hanno competenze specifiche in quanto è possibile ottenere il risultato desiderato, la soluzione algoritmica di un problema, senza scrivere una sola riga di codice.

Il successo di applicazioni quali Scratch, Snap, Blockly e App Inventor, solo per citarne alcune, mostra il grande interesse verso questo stile di programmazione. Il Block Programming è un campo di ricerca oggi di grande attualità e ambienti universitari, oltre a importanti aziende informatiche, dedicano molta attenzione a questo settore. A titolo di esempio citiamo Google Inc. che ha effettuato ingenti investimenti nel settore, tanto da divenire un punto di riferimento tra gli utilizzatori. Analogamente, prestigiose università internazionali, come il Massachusetts Institute of Technology di Boston, operano nel campo del Block Programming, del Coding e della diffusione del Computational Thinking.

Alcuni studi [66] riportano un atteggiamento negativo da parte di alcuni alunni della scuola primaria verso la programmazione dovuto soprattutto alle difficoltà in-

iziali incontrate nella scrittura del codice.

*"The loss of attraction to [computing] comes from our being unable to communicate the magic and beauty of the field. We need to create an appreciation for the elegance and power of what computing can do"*

– Peter Denning

Il Block Programming è spesso applicato in situazioni di gioco, ed è utilizzato in molti progetti per l'introduzione delle basi della programmazione, fra questi abbiamo visto ad esempio Scratch [67] e Alice [51]. Questi progetti propongono a bambini che non hanno precedenti esperienze di programmazione giochi educativi la cui soluzione necessita di un semplice procedimento algoritmico che viene espresso mediante la realizzazione di un puzzle di blocchi. Le esperienze di gioco e questi primi approcci al problem solving favoriscono poi il passaggio ai tradizionali linguaggi testuali per risolvere semplici problemi.

La maggior parte delle iniziative a carattere ludico che abbiamo citato sono rivolte a studenti della scuola primaria, ma esistono progetti rivolti ad un pubblico più adulto come, per esempio, AppInventor [63] sviluppato al Massachusetts Institute of Technology. Si tratta anche in questo caso, come abbiamo visto, di un ambiente basato sulla programmazione a blocchi che permette di sviluppare in modo semplice app multimediali che possono essere eseguite su dispositivi Android.

In ogni caso l'obiettivo comune, che caratterizza questi diversi progetti, non è di trasformare tutti in un professionisti della programmazione, ma di introdurre i concetti principali del problem solving.

L'idea fondante della programmazione a blocchi, che caratterizza quasi tutti i progetti precedentemente descritti, è quella di fornire un'interfaccia grafica con blocchi di diversi tipo. L'utente/programmatore può combinare i blocchi in vario modo con funzioni di drag & drop, in modo da formare un puzzle che rappresenta il procedimento di soluzione di un determinato problema. I blocchi rappresentano quindi gli elementi basilari del linguaggio e si differenziano per forma e colore per facilitarne l'identificazione e l'utilizzo.

### 3.3.1 Vantaggi della programmazione a blocchi

Le difficoltà incontrate da chi si avvicina alla programmazione per la prima volta, sono essenzialmente di due tipi: da una parte è necessario formalizzare le idee sulla soluzione algoritmica del problema, dall'altra è necessario rispettare la rigorosa sintassi del linguaggio di programmazione.

La programmazione a blocchi elimina la possibilità di introdurre errori sintattici e consente agli utenti di concentrarsi interamente sulla logica dei problemi assegnati e sulla loro soluzione. La connessione dei vari blocchi è infatti rigidamente vincolata dalla forma dei blocchi stessi, questo aspetto rappresenta i vincoli sintattici del linguaggio.

La struttura predefinita e non modificabile dei blocchi e delle loro interconnessioni previene quindi qualsiasi errore di tipo sintattico. Nei linguaggi text-based l'insieme delle parole chiave, la presenza di parentesi, la punteggiatura e altre caratteristiche sono, purtroppo, fonte di una serie di errori molto comuni riscontrabili soprattutto nel codice scritto dai neofiti della programmazione. La presenza di blocchi specifici rende più semplice la manipolazione di intere strutture sintattiche quali condizioni e cicli e "impone" una corretta indentazione al codice evitando i classici errori dovuti per esempio al non bilanciamento delle parentesi.

Anche se la maggior parte dei progetti educativi basati sul Block Programming è orientata a un pubblico molto giovane, cominciano ad apparire altri progetti rivolti verso gli studenti delle scuole superiori.

Nell'articolo di Weintrop e Wilensky [68] viene analizzato l'approccio degli studenti delle scuole superiori nei confronti del Block Programming. Vengono evidenziati i punti di forza e di debolezza confrontando ambienti basati sulla classica programmazione testuale con altri basati su quella a blocchi. Gli studenti identificano l'interazione di composizione mediante drag & drop e la comprensione dell'insieme dei componenti del linguaggio come elementi che determinano la facilità d'uso degli ambienti basati sul Block Programming. Preferiscono invece i più tradizionali ambienti text-based nel caso debbano sviluppare applicazioni più complesse.

Non tutti i pareri sono concordi sulla validità del block programming, nell'articolo *"Teaching objects-first in introductory computer science"* [69] viene espressa una

opinione critica che, pur confermando i vantaggi derivanti dall'utilizzo di questo stile di programmazione, ne evidenzia alcune limitazioni riscontrate soprattutto nella "scrittura" di algoritmi più complessi. In [70] gli autori presentano un ambiente di editing basato su uno stile "misto", *Frame-based programming*, che reintroduce l'utilizzo della tastiera superando l'approccio "mouse-centrico" tipico delle interfacce del block programming.

# Part II

# Proposte metodologiche e tool didattici per corsi CS1

*"Measuring programming progress by lines of code is like measuring aircraft building progress by weight."*

– Bill Gates.

Nel capitolo 4 presentiamo una esperienza di stage di introduzione al coding effettuato presso il dipartimento di Ingegneria e Architettura dell'Università di Parma. Lo stage, della durata di una settimana, ha portato gli studenti a realizzare una semplice ma funzionante implementazione di un videogame. La metodologia utilizzata, caratterizzata dai modelli object-early e game-first, è stata oggetto di un'analisi di gradimento basata sulle risposte a un questionario somministrato agli studenti al termine dello stage.

Nel capitolo 5 presentiamo *Object Oriented Puzzle Programming*, un tool didattico per la progettazione di applicazioni basate sul paradigma object-oriented. Il tool fonde i concetti del Block Programming con l'Object Oriented Design. Un primo prototipo è stato presentato nell'ambito di un corso di Istruzione e Formazione Tecnica Superiore per sviluppatori software ed è stato oggetto di analisi di gradimento basata sulle risposte a un questionario somministrato agli studenti al termine del corso.

# Chapter 4

# Gaming

*"The possibilities are infinite!"*

– Notch, Minecraft Developer.

## 4.1 Object-early e Gaming nella nostra esperienza di stage

Al dipartimento di Ingegneria e Architettura dell'Università di Parma negli ultimi anni è stato proposto uno stage della durata di 5 giorni agli studenti del quarto anno delle scuole medie superiori. L'obiettivo dello stage è quello di proporre un primo approccio al mondo informatico e alla programmazione anche a coloro i quali non possiedono nessuna competenza pregressa in questo settore.

Dopo una breve introduzione al problem solving viene proposto agli studenti lo sviluppo, per piccoli passi, di un videogame (Frogger, PacMan, Bubble Bobble ecc.) basato sul movimento di personaggi all'interno di un ambiente virtuale. Per la sua semplicità sintattica e per la facilità di utilizzo, come linguaggio di programmazione, è stato scelto Python [71] associato al framework PyGame [72].

Nell'ambito dello stage il processo di apprendimento è modellato su una sequenza di livelli di difficoltà crescenti. L'avanzamento di livello si ottiene attraverso

lo sviluppo di singole parti di un progetto, nel nostro caso un videogame dinamico basato su diversi personaggi che interagiscono fra loro in un'arena di gioco. Progredire nello sviluppo del gioco corrisponde quindi all'acquisire nuovi concetti e abilità nella programmazione.

Durante tutte le fasi di sviluppo agli studenti vengono forniti esempi e suggerimenti mirati per permettere loro di iniziare e poi progredire nella costruzione della loro applicazione.

La natura dei giochi proposti, che si basa su vari tipi di oggetti che si muovono nell'arena di gioco, permette di introdurre gli studenti alla progettazione e alla costruzione di una gerarchia di classi da cui istanziare i vari personaggi. L'incapsulamento, l'eredità e il polimorfismo sono intrinseci in questi tipi di gioco che, inoltre, si prestano ad uno sviluppo di progetti avanzati da parte degli studenti migliori.

I concetti elementari della programmazione e del paragima object-oriented vengono quindi introdotti parallelamente alle fasi di avanzamento del progetto con livelli di crescente difficoltà fino ad arrivare alla completa realizzazione di un gioco, semplice, ma pienamente funzionante.

La creazione di computer game è strettamente connessa ai concetti di oggetto ed evento: un gioco può essere facilmente visto come composto da oggetti con le loro proprietà e i loro comportamenti che agiscono in base ad una successione di eventi. Introdurre il concetto di ereditarietà risulta naturale nel momento in cui si devono specificare caratteristiche e comportamenti particolari di un personaggio in relazione a quelli di uno più generico e questo fatto porta, in modo naturale, gli studenti a comprendere le nozioni fondamentali del paradigma object-oriented.

Agli studenti viene fornito un semplice framework (Fig. 4.1) formato da poche decine di righe di codice, questo viene presentato e discusso con gli studenti stessi in modo da far loro comprenderne la struttura e il funzionamento. Il framework ruota intorno a due classi principali: Arena e Actor.

- **`Actor`** è essenzialmente un'interfaccia che deve essere implementata da tutti gli oggetti che fanno parte del gioco. Ogni personaggio deve fornire il metodo `move` che sarà attivato ad ogni iterazione e il metodo `collide` per gestirne il comportamento in caso di collisioni con altri oggetti.

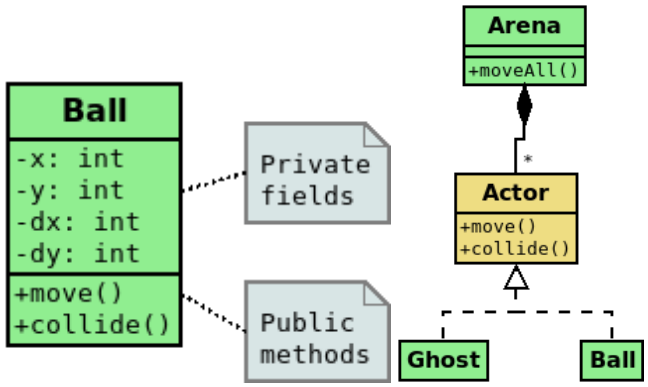



Figure 4.1: La prima classe presentata e il framework per la realizzazione del gioco

- **`Arena`** è fondamentalmente un contenitore di oggetti. Ha il compito di gestire tutti i personaggi del gioco secondo le politiche di turnazione in modo completamente agnostico rispetto al tipo dei vari oggetti. Quando l'applicazione principale attiva il metodo `move_all` della sua arena questa, a sua volta, attiva sequenzialmente ogni personaggio chiamandone il metodo `move`. Ad ogni turno viene inoltre eseguita una procedura per individuare le eventuali collisioni fra gli oggetti in movimento e, in caso avvenga una collisione, viene chiamato il metodo `collide` dei due oggetti che entrano in contatto.

### 4.1.1 L'approccio object-early

Il framework proposto ha il vantaggio di guidare gli studenti a utilizzare tecniche relativamente avanzate di programmazione, come spesso viene considerato il polimorfismo, in modo intuitivo. È facile, infatti, intuire che ogni personaggio si muove all'interno del gioco, ma che le varie tipologie di personaggi hanno un comportamento che può seguire una logica diversa da quella di altri.

È interessante far notare che non è tanto importante il fatto di fornire un framework comune predefinito, ma piuttosto quello di costruirlo insieme agli studenti e di svilupparlo in modo incrementale durante le fasi di sviluppo del gioco. Questo processo è semplificato anche da un modulo personalizzato che è compatibile con

l'API di PyGame [72, 53] ma esegue in modo trasparente il codice relativo alla grafica nel browser, tramite Brython[1]. Gli studenti, anche i meno esperti, possono iniziare a programmare visualizzando una forma sullo schermo e poi progredire verso compiti di crescente complessità. L'intero processo di apprendimento è proposto come una sequenza di livelli, ciascuno da completare prima di passare al livello successivo; tale progresso corrisponde all'acquisizione di nuovi concetti e abilità di programmazione.

1. **Livello 1 - Cicli e condizioni**, disegnare varie forme sullo schermo.
2. **Livello 2 - Liste**, memorizzare dati di input e, per esempio, disegnare un istogramma.
3. **Livello 3 - Funzioni**, gestire azioni periodiche in un semplice scenario e, per esempio, far rimbalzare una pallina in un riquadro.
4. **Livello 4 - Oggetti**, incapsulare i dati e gestire vari oggetti in base a una loro logica interna di movimento.
5. **Livello 5 - Polimorfismo**, gestire in modo omogeneo il movimento di differenti tipi di oggetti introducendo una nozione intuitiva del principio di sostituzione di Liskov.
6. **Livello 6 - Composizione**, iniziare la realizzazione di un gioco in cui interagiscono vari tipi di personaggi.
7. **Livello 7 - Ereditarietà**, creare un gioco specifico definendolo come sottoclasse di `Arena`. Gestire per esempio varie situazioni che portano alla fine del gioco o creare personaggi generici che hanno un comportamento comune.

I concetti del paradigma object-oriented possono essere introdotti gradualmente, come avanzamenti in un processo di apprendimento ludico, che in realtà corrisponde proprio allo sviluppo del gioco. In questo modo, l'utilità di questi diversi concetti diventa evidente al crescere della complessità dell'applicazione.

[1] brython.info

In base alla nostra esperienza, questo sviluppo può essere realizzato anche in tempi brevi, insieme agli studenti, fornendo loro suggerimenti e orientandoli nello sviluppo, ma preservando e incoraggiando la loro autonomia nel trovare soluzioni originali e partendo da un framework preesistente e da semplici esempi.

Nello stage estivo questo processo di sviluppo e apprendimento è concentrato in solo cinque giornate (40 ore in totale dando l'opportunità di accesso pomeridiano ai laboratori con l'assistenza di un tutor). Normalmente il primo giorno è dedicato ad esercizi su cicli e istruzioni condizionali, il secondo giorno a liste e funzioni, il terzo alla interazione con semplici oggetti, il quarto giorno al polimorfismo e alla composizione e infine il quinto giorno al concetto di ereditarietà. Inevitabilmente alcuni studenti necessitano di più tempo per completare alcuni dei livelli intermedi mentre altri sono tentati di anticipare i tempi; entrambe queste possibilità sono gestite mediante l'ausilio di tutor.

Gli studenti che iniziano lo stage senza alcuna conoscenza pregressa nel campo della programmazione necessitano, oltre a suggerimenti di natura tecnica, anche di essere incoraggiati soprattutto nelle fasi iniziali. Tuttavia essi si dimostrano per lo più abbastanza soddisfatti, in alcuni casi anche entusiasti, al termine delle giornate di stage. Ciò è dimostrato non solo dalle nostre impressioni dirette, ma anche dalle informazioni che ci ritornano dalle loro scuole e dai sondaggi che effettuiamo in modo sistematico fra gli studenti al termine dello stage.

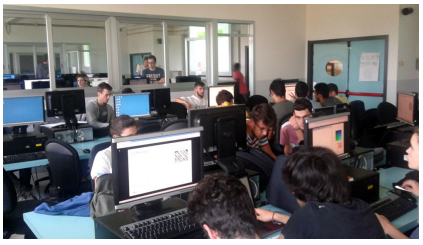
## 4.2 "The Space Invaders, in a week of code" - I risultati dell'esperienza

Pur avendo a disposizione un periodo di tempo limitato, circa 20 ore (con 20 ore facoltative pomeridiane in laboratorio), i risultati ottenuti, anche dagli studenti che non avevano in precedenza utilizzato nessun linguaggio di programmazione, sono stati buoni e gli studenti hanno mostrato in genere molto interesse e hanno partecipato attivamente sia alle lezioni teoriche che alle attività di laboratorio dove, in piccoli gruppi e sotto la guida di tutor, hanno realizzato la propria versione del gioco proposto.

### 4.2.1 Analisi dei risultati

Nello stage estivo del 2017 abbiamo ospitato presso il nostro dipartimento 115 studenti provenienti da varie scuole superiori. Fra questi, 15 studenti provenivano dal liceo classico o da scuole a carattere umanistico; 50 da licei scientifici (che purtroppo spesso non includono corsi specifici in informatica) e 50 da istituti tecnici. La figura 4.2 mostra la situazione abbastanza variegata dei diversi livelli di competenze iniziali.

L'obiettivo finale dello stage è stato la realizzazione del gioco Space Invaders. Gli studenti hanno inizialmente animato un singolo personaggio alieno proponendo soluzioni personali, in seguito hanno spostato dati e codice all'interno della struttura di una classe con l'obiettivo di creare un intero esercito di alieni. Infine, hanno reso conformi le loro classi a un'interfaccia astratta per i personaggi del gioco per poter gestire in modo omogeneo movimenti e interazioni e arrivare infine alla realizzazione di un'applicazione pienamente funzionante.



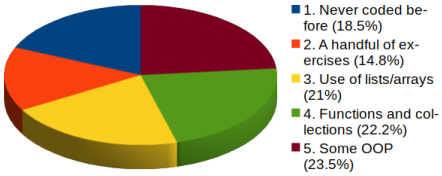



Figure 4.2: Studenti che hanno partecipato alle attività di stage e le loro competenze pregresse di programmazione

### 4.2.2 Le opinioni

Al termine dell'ultima giornata di stage è stato proposto agli studenti un questionario a risposta chiusa in cui veniva accettata una sola risposta per ogni domanda. In totale 81 studenti hanno accettato di rispondere, in modo anonimo, a tutte le domande del questionario che è stato proposto mediante Google Forms. Riportiamo qui i risultati più significativi.

La figura 4.3 evidenzia come OOP sia stato ritenuto l'argomento più difficile fra quelli trattati nello stage. È interessante notare che, per quanto riguarda la programmazione ad oggetti, gli studenti hanno valutato in egual misura il livello di difficoltà di tutti i suoi diversi aspetti senza particolari preferenze, ad eccezione dell'ereditarietà, ritenuta meno problematica nonostante sia stata trattata, anche se in modo non approfondito, solo nel corso dell'ultima giornata. Molti studenti hanno evidenziato problemi dovuto al fatto di confondere gli attributi degli oggetti con i parametri dei metodi e con le variabili locali. È inoltre interessante riscontrare che una fra le maggiori difficoltà sia risultata la gestione degli aspetti grafici, che in realtà fanno parte del dominio del problema proposto.

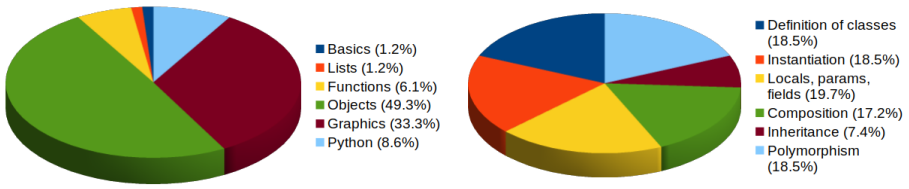



Figure 4.3: Le maggiori difficoltà in generale e in relazione a OOP

Una parte del questionario è relativa al tipo di esperienza che gli studenti avrebbero preferito affrontare in uno stage di questo tipo. Abbiamo chiesto loro se ritengono efficace affrontare e sviluppare un lungo progetto e in particolare un videogame. Questi argomenti sono stati messi in relazione con la possibilità di risolvere problemi più piccoli e di occuparsi di altri domini applicativi. Come mostra la figura 4.4, gli studenti hanno mostrato un alto gradimento per i progetti lunghi e per i videogame in particolare (fig. 4.5). Nessuno fra loro ha affermato di non apprezzare il dominio applicativo specifico dei computer game e il 91,5% ha dichiarato di essere motivato o fortemente motivato nello sviluppo di problemi di questo tipo. In generale, lo sviluppo di un progetto più lungo è stato trovato molto gratificante, o almeno interessante, dall'86,5% degli studenti.

La scelta del linguaggio utilizzato per introdurre i primi concetti di program-

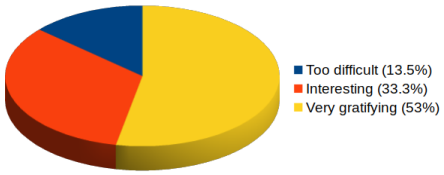



Figure 4.4: Motivazione nell'affrontare un unico grande progetto nello stage

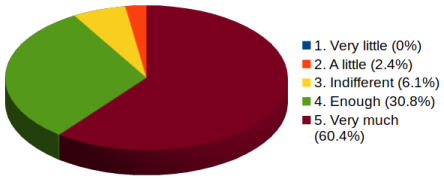



Figure 4.5: Motivazione nel realizzare un videogame

mazione e, in particolare, il paradigma object-oriented è importante. In generale, gli studenti hanno apprezzato la scelta di Python: solo il 16% preferirebbe infatti una linguaggio diverso per lo stage. L'aspetto ritenuto più vantaggioso di Python è dato dalla sua leggibilità e compattezza, queste caratteristiche sono state ritenute le più importanti dal 40,7% degli studenti. Altri studenti (19.8%) hanno apprezzato soprattutto la presenza di una shell interattiva in cui testare esempi di codice. La modalità di accesso agli attributi e l'utilizzo di `self` sono stati considerati invece i principali aspetti negativi di Python dal 37% degli studenti.

Abbiamo chiesto, infine, di valutare il loro livello di autonomia raggiunto al termine dello stage nella risoluzione dei problemi (figura 4.6) e il loro grado di soddisfazione in relazione a questa esperienza. Nella misurazione del grado di soddisfazione (figura 4.7) il valore 3 indica un livello di apprendimento più che sufficiente, i valori inferiori indicano situazioni ancora confuse mentre i valori alti indicano livelli buoni o ottimi. Le risposte sono indicate nella loro forma aggregata (a sinistra, in entrambe

le figure) e in relazione alle abilità iniziali di programmazioni degli studenti (a destra, in entrambe le figure). A nostro parere, i risultati sono abbastanza buoni e confermano le nostre prime impressioni di una generale soddisfazione e di entusiasmo da parte di vari studenti, molti dei quali erano, lo ricordiamo, alla loro prima esperienza di programmazione.

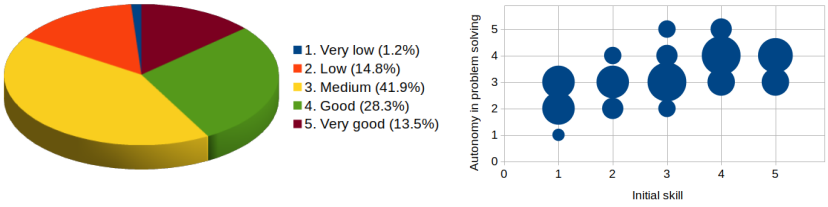



Figure 4.6: Autovalutazione dell'autonomia raggiunta alla fine dello stage nelle capacità di soluzione di problemi

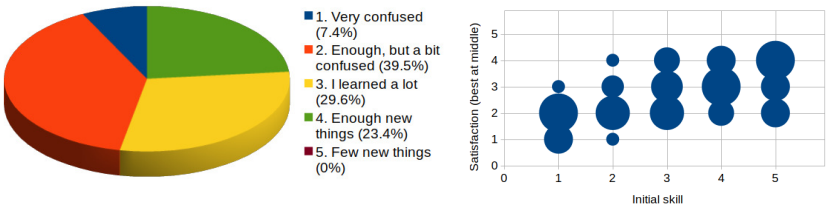



Figure 4.7: Livello di soddisfazione per i risultati ottenuti

### 4.2.3 I lavori prodotti

Le modalità di svolgimento dello stage hanno permesso a tutti gli studenti, anche quelli senza precedenti competenze in coding, di portare a compimento un progetto funzionante. Tuttavia, i programmi realizzati differiscono dal punto di vista delle caratteristiche dei personaggi, del loro modo di operare nell'arena di gioco e degli


Figure 4.8: Alcune immagini delle applicazioni sviluppate dagli studenti al termine dello stage.

aspetti multimediali del gioco stesso[2]. Durante lo stage, alcuni studenti hanno realizzato logiche di gioco più complesse e inserito nuovi personaggi, come ad esempio le *"Navi del Mistero"*, navicelle bonus che, se abbattute, permettono al giocatore di incrementare il suo punteggio più velocemente.

La programmazione di un'applicazione che realizza un semplice gioco è stata spesso riportata come una valida scelta per introdurre il pensiero computazionale e i principi del coding. Nella nostra esperienza, la programmazione di un gioco stimola gli studenti a affrontare le tipiche problematiche dello sviluppo di una generica applicazione applicando il paradigma orientato agli oggetti in modo graduale e favorisce il loro continuo coinvolgimento in tutte le fasi di sviluppo.

I personaggi di un gioco interattivo possono essere modellati come oggetti, con le loro proprietà e comportamenti per gestire i loro spostamenti e le loro reciproche interazioni. Inoltre diversi tipi di personaggi possono essere progettati partendo da un'interfaccia astratta comune realizzando poi successive implementazioni polimorfe.

Questo tipo di attività può essere paragonato a una sorta di processo di avanzamento all'interno di un gioco nel quale gli studenti devono fornire una loro personale soluzione a un determinato compito, prima di poter avanzare al livello successivo. L'analisi delle risposte degli studenti al sondaggio proposto dopo lo stage del 2017 ha per lo più confermato l'applicabilità e la validità di questa metodologia in un corso introduttivo della durata complessiva di una sola settimana.

[2] I progetti consegnati, nella loro forma originale, sono disponibili sul sito `http://www.ce.unipr.it/stage`

# Chapter 5

# Object Oriented Puzzle Programming

*"Watching my daughter Kate, and her middle school classmates,*
*struggle through a Java course using a commercial IDE was a painful experience*
*The sophistication of the tool added significant complexity to the task of learning."*

– James Gosling.

## 5.1 Un nuovo tool per Object Oriented Design basato su Block Programming

Lo stile di programmazione a blocchi introduce vari e significativi vantaggi rispetto alla programmazione testuale, non solo per i principianti ma per tutti i programmatori.

In un corso introduttivo al coding basato sul paradigma ad oggetti è importante focalizzare l'attenzione oltre che sugli aspetti strettamente legati al linguaggio di programmazione utilizzato anche, e soprattutto, sulla progettazione e modellizzazione delle applicazioni. Object Oriented Puzzle Programming è un ambiente di progettazione e sviluppo software che abbiamo realizzato con l'obiettivo di estendere le

caratteristiche del Block programming al paradigma object-oriented.

### 5.1.1 Object Oriented Puzzle Programming (OOPP)

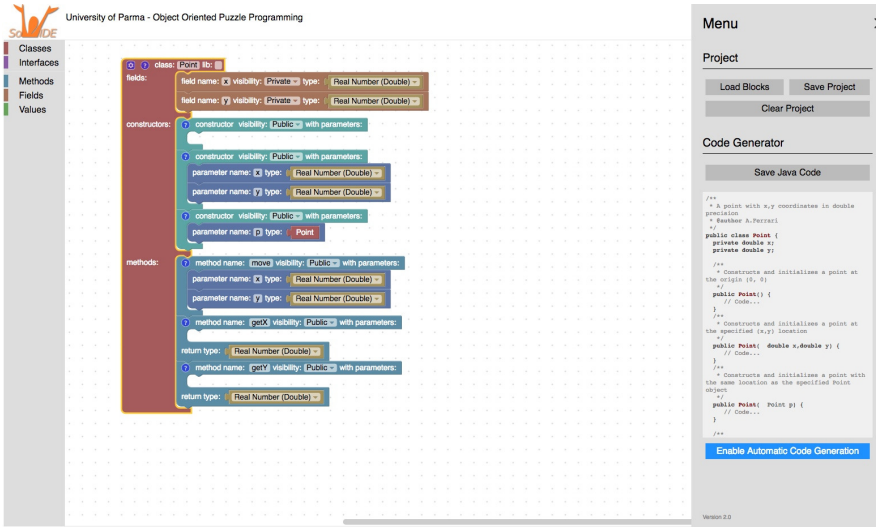



Figure 5.1: OOPP - workspace

OOPP è un ambiente di progettazione e sviluppo pensato principalmente per un utilizzo a scopo didattico, al fine di introdurre in modo semplice e intuitivo la progettazione e programmazione object-oriented. OOPP è uno strumento prototipale che consente a un utente/programmatore di progettare e creare semplici applicazioni object-oriented mediante la composizione di blocchi e di generare quindi in modo automatico il codice Java che sarà l'ossatura dell'applicazione che si vuole realizzare. Comprende un workspace (fig. 5.1) che permette di definire in modo visuale "blocchi" di codice, con relativi metodi, costruttori e attributi che poi andranno a generare automaticamente il relativo codice Java. Due tipi principali di blocchi sono disponibili: le classi e le interfacce, entrambi corredati di metodi, tipi, ereditarietà, visibilità ed eventuali valori nel caso di costanti per le interfacce.

### 5.1.2 Google Blockly API

Utilizzando l'Application Programming Interface (API) messa a disposizione da Google per il progetto Blockly [1] abbiamo realizzato un insieme di blocchi di alto livello che possono essere utilizzati per la programmazione orientata agli oggetti. L'ambiente visuale è interamente sviluppato in JavaScript ed è quindi eseguibile all'interno di un qualunque browser web, questo fatto riduce ulteriormente le difficoltà di utilizzo dell'applicazione che non necessita di nessuna installazione di software specifico. L'ambiente è caratterizzato da tre elementi fondamentali:

1. una toolbox che contiene i blocchi disponibili suddivisi in base al loro tipo
2. uno spazio di lavoro dove disporre e collegare fra loro i blocchi in modo da formare un puzzle/programma
3. un contenitore testuale per visualizzare il corrispondente codice generato automaticamente convertendo ogni blocco del puzzle in un insieme di istruzioni Java

Partendo dalle categorie standard di Blockly: blocchi logici, cicli, operatori matematici, liste, variabili ecc. abbiamo creato ulteriori blocchi per la programmazione orientata agli oggetti: interfacce, classi, costruttori, metodi, parametri. I nuovi blocchi sono sagomati in modo da permettere solo collegamenti che rispettino i vincoli sintattici.

La definizione di una nuova classe o di un'interfaccia rende immediatamente disponibile nella casella degli strumenti un nuovo blocco che rappresenta il nuovo tipo di dato che è poi possibile utilizzare nello sviluppo dell'applicazione. Per mantenere l'ambiente semplice e intuitivo, per quanto possibile, sono state introdotte solo le principali caratteristiche della programmazione orientata agli oggetti. Al momento, non è possibile definire esplicitamente classi interne e classi astratte. Le funzionalità presenti permettono altresì di mettere in pratica i fondamentali concetti della programmazione object-oriented: l'incapsulamento, l'ereditarietà e il polimorfismo. Per

[1] developers.google.com/blockly

la nostra esperienza didattica riteniamo che caratteristiche più avanzate, che possiamo trovate ad esempio nelle specifiche del linguaggio UML, non sono né necessarie né auspicabili in tale ambiente introduttivo.

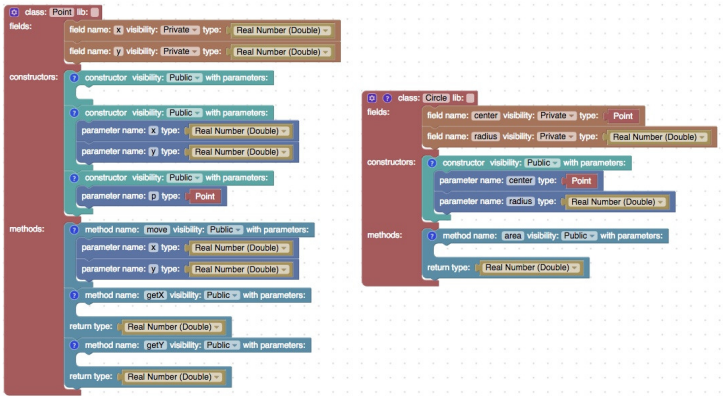



Figure 5.2: OOPP - un esempio di composizione

OOPP è pensato come strumento da utilizzare nelle fasi iniziali di un percorso didattico con finalità di progettazione object-oriented e quindi inadatto per applicazioni di una certa complessità, come gli stessi autori di Blockly affermano:

> *"Blockly is currently designed for creating relatively small scripts. ...*
> *Please do not attempt to maintain the Linux kernel using Blockly"*
>
> – Neil Fraser - Google.

La struttura dell'interfaccia permette all'utente programmatore di visualizzare e analizzare contemporaneamente il puzzle e il codice generato. Questa caratteristica risulta utile al fine di apprendere le caratteristiche sintattiche del linguaggio di programmazione (in questo primo prototipo Java), che possono essere utilizzate successivamente per risolvere problemi più complessi. Il codice generato può poi essere esportato in file java che possono essere utilizzati come struttura di base dell'applicazione da completare in altri ambienti di sviluppo. Il puzzle che rappresenta l'insieme delle interfacce e delle classi del progetto può essere salvato per

permettere successive modifiche ricaricandolo in OOPP.

Abbiamo inoltre sviluppato un tool che permette di importare in OOPP il file jar di una qualunque applicazione e quindi di visualizzarne come puzzle di blocchi l'insieme delle classi e delle interfacce con i loro attributi, costruttori e metodi favorendo così la comprensione della struttura dell'applicazione che si intende analizzare. Questa funzionalità rende anche disponibile l'insieme di oggetti e funzionalità presenti in una qualunque libreria, le classi e le interfacce vengono quindi importate e possono essere utilizzate come componenti delle nuove applicazioni che si vogliono sviluppare.

### 5.1.3 L'architettura di Google Blockly

Blockly è definito come *"... a library for building visual programming editors ..."* uno strumento per gli sviluppatori di applicazioni, molte sono infatti le applicazioni educative che lo utilizzano come base di partenza per lo sviluppo. In pratica si tratta di una libreria JavaScript client-side finalizzata alla creazione di linguaggi visuali basati sulla programmazione e blocchi. La libreria mette inoltre a disposizione varie funzionalità per definire gli ambienti (editor) per lo sviluppo delle applicazione block based.

Blockly è un progetto open source (Apache 2.0 License) sviluppato da Google e viene fornito con un gran numero di blocchi predefiniti, fra questi: blocchi di controllo, funzioni matematiche e così via. Blockly non è un linguaggio di programmazione, è un editor di programmazione visuale che opera all'interno di un browser e genera codice eseguibile in vari linguaggi di programmazione quali JavaScript, Python e altri ancora.

Oltre alla versione web based che sta alla base del nostro progetto OOPP, Google mette a disposizione API anche per la realizzazione di applicazioni block based in ambiente Android e iOS.

Nella versione web based, gli sviluppatori possono integrare in una qualunque pagina web un'interfaccia Blockly ridimensionabile popolandola di blocchi e fornendo un workspace che consente agli utenti (programmatori novizi) di scrivere programmi collegando fra loro i vari blocchi. Ogni oggetto visuale (blocco) rappresenta in realtà un oggetto di codice. Gli utenti possono selezionare i blocchi appartenenti a varie

categorie e collegarli fra loro per progettare una soluzione algoritmica per un problema e generare poi il programma corrispondente in un linguaggio di programmazione senza aver digitato una sola riga di codice. Una delle caratteristiche di Blockly, anzi uno dei suoi obiettivi principali è quello di generare codice leggibile.

Inoltre Google fornisce un insieme di funzioni di libreria (API) che consentono allo sviluppatore di creare nuovi blocchi. Questi nuovi blocchi possono poi essere utilizzati insieme a quelli predefiniti in applicazioni di varia natura. La creazione di un insieme di nuovi blocchi corrisponde alla creazione di un nuovo linguaggio, con la sua nuova sintassi e semantica. La sintassi è definita dalla struttura dei blocchi, dal colore, dalla forma e dalle connessioni possibili. Per gli utenti/programmatori ciò costituisce un vantaggio significativo: è infatti praticamente impossibile creare puzzle (cioè programmi) con errori di sintassi, in quanto per ogni blocco è possibile definire vincoli e modalità di connessione con gli altri. In pratica, Blockly fornisce una sorta di textarea dove è possibile scrivere solo codice sintatticamente corretto, la grande differenza rispetto ad altri ambienti di sviluppo è data dal fatto che il codice testuale è sostituito da blocchi visuali.

La semantica dei blocchi viene poi definita fornendo per ciascuno di essi il codice corrispondente che può essere scritto in qualsiasi linguaggio di programmazione. I più utilizzati sono JavaScript, Python, PHP e Dart, in quanto linguaggi interpretati che permettono una istantanea esecuzione del codice. Tuttavia, è anche possibile generare codice in linguaggi che vengono di solito compilate staticamente, ad esempio Java. La doppia rappresentazione del programma (fig. 5.3), sia come puzzle che come codice testuale, è molto interessante e stimolante in un ambiente educativo. Infatti, il programmatore novizio può confrontare le due versioni e identificare i loro limiti reciproci.

Anche i blocchi relativi alla programmazione object-oriented per OOPP sono stati sviluppati utilizzando le API di Blockly e utilizzando inizialmente BlockFactory (fig. 5.4) che è uno strumento molto utile per la costruzione dei nuovi blocchi mediante una semplice e intuitiva interfaccia grafica. BlockFactory permette di definire le funzionalità principali del nuovo blocco selezionandole fra quelle proposte in una casella degli strumenti. Durante la fase di creazione, viene mostrata un'anteprima del

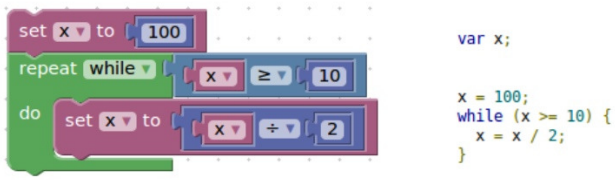



Figure 5.3: Puzzle programming e Text-based programming

nuovo blocco, il suo aspetto visivo e il codice JavaScript o JSON corrispondente.

Per ogni nuovo tipo di blocco è necessario definire il colore che ne faciliterà l'identificazione, il nome e vari altri attributi, oltre alla sua forma che stabilirà le possibili connessioni con altri blocchi. Nel caso di OOPP, gli utenti finali non sono tenuti a gestire questa fase, abbiamo infatti già creato i blocchi generici necessari alla programmazione object-oriented; questi blocchi sono disponibili per essere selezionati nella toolbox dell'ambiente di sviluppo e sono pronti per essere utilizzati per la realizzazione dei puzzle/programmi.

BlockFactory è di per sé un'applicazione block based, quindi, oltre ad essere, come abbiamo visto, uno strumento estremamente utile per la definizione di nuovi blocchi, può anche essere considerato come un esempio della potenzialità della programmazione block based.

Dopo aver realizzato con BlockFactory i blocchi principali di OOPP abbiamo analizzato il codice generato che è poi servito da modello per la creazione dei blocchi successivi che è avvenuta scrivendo direttamente il codice JavaScript.

### 5.1.4 OOPP: L'ambiente di lavoro

OOPP è una web application sviluppata in Javascript e interamente eseguita all'interno di un browser quindi fruibile da un qualunque server http ma anche eseguibile direttamente installando i file in ambiente locale. L'interfaccia è composta da una *toolbox* che mette a disposizione le varie categorie di blocchi che possono essere trascinati e collegati fra loro a formare un puzzle in un'area di lavoro. Un menu a scomparsa

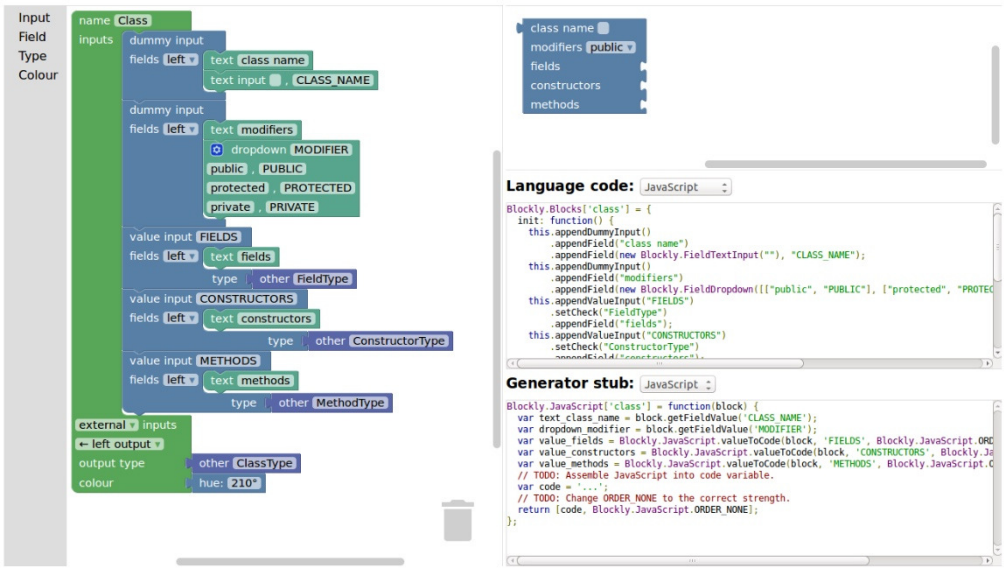



Figure 5.4: Block Factory: definizione del blocco classe

permette di attivare varie funzionalità inerenti al progetto corrente e alla generazione del codice.

## 5.2 Operare con OOPP

Il puzzle che rappresenta l'insieme delle interfacce e classi dell'applicazione è quindi formato da un insieme di blocchi di varie tipologie: oltre a blocchi di interfaccia e classe sono presenti i blocchi per gli attributi, i costruttori, i metodi, i parametri, i tipi di dato ecc.

Una delle funzionalità implementate permette di salvare il puzzle per poterlo utilizzare ed eventualmente modificare in un secondo momento. La funzione di salvataggio converte ogni blocco in un insieme di tag XML che ne descrivono tutte le caratteristiche.

La funzione di caricamento effettua poi la conversione in senso inverso e ricostru-

isce l'intero puzzle partendo da un file XML che segua le specifiche di descrizione dei singoli blocchi.

I tag definiscono per ogni blocco la tipologia, il colore, l'aspetto grafico e soprattutto le modalità e i vincoli di connessione con altri blocchi che descrivono gli aspetti sintattici del linguaggio.

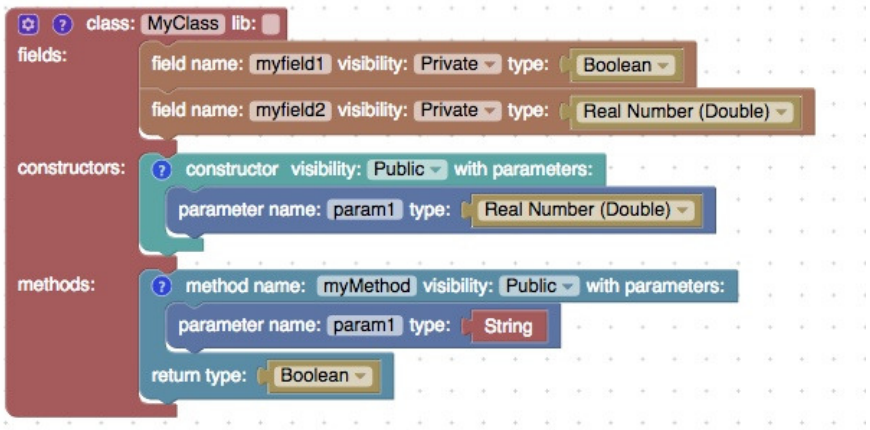



Figure 5.5: Puzzle di blocchi che rappresenta una semplice classe

L'insieme di blocchi che compongono la classe di esempio di figura 5.5 è convertito nel codice XML di figura 5.6.

Il caricamento da file XML di nuovi blocchi non comporta l'eliminazione di quelli già presenti nel workspace (che eventualmente possono essere precedente eliminati mediante la funzione *Clear Project*). La funzione è progettata in questo modo per permettere il caricamento di librerie in progetti preesistenti, in modo da poterne sfruttare i tipi di dato per la definizione di nuovi blocchi.

La definizione delle caratteristiche di ogni blocco è basata sulle specifiche definite dall API di Google Blockly [2] ed è memorizzata in file JavaScript presenti nella cartella blocks. I blocchi utilizzati nella nostra applicazione sono definiti nei file interfaces.js (interfacce, costanti e stringhe) e nel file classes.js (classi, tipi di dato primitivi, attributi, parametri, costruttori, metodi). I vincoli che stabiliscono quali blocchi sono

[2] developers.google.com/blockly/reference

```xml
<xml xmlns="http://www.w3.org/1999/xhtml">
  <block type="classes_set" id="129cfa90-4b3d-46fc-bd74-4d4a2ff083c9" x="12" y="12">
    <mutation implements="0" extends="0" package="0"></mutation>
    <field name="CLASS">MyClass</field>
    <field name="LIB">FALSE</field>
    <comment pinned="false" h="80" w="160">Describe this class... @author
    </comment>
    <statement name="FIELDS">
      <block type="classes_field" id="d38ba22c-d84d-46af-822f-beb40eb18221">
        <field name="NAME">myfield1</field>
        <field name="VISIBILITY">PRIVATE</field>
        <value name="TYPE">
          <block type="object_type" id="7da9e222-eed2-44d9-b642-441a7dfb9167">
            <field name="TYPE_NAME">BOOLEAN</field>
          </block>
        </value>
        <next>
          <block type="classes_field" id="e3ba24a4-c9bf-42e9-8ae0-7f08b1e5cbac">
            <field name="NAME">myfield2</field>
            <field name="VISIBILITY">PRIVATE</field>
            <value name="TYPE">
              <block type="object_type" id="aa2e7d2f-2df5-488c-ac1c-69c0cda999c0">
                <field name="TYPE_NAME">DOUBLE</field>
              </block>
            </value>
          </block>
        </next>
      </block>
    </statement>
    <statement name="CONSTRUCTORS">
      <block type="classes_constructor" id="f6b02c37-a198-43fe-a3a0-884b873f5bb6">
        <field name="VISIBILITY">PUBLIC</field>
        <comment pinned="false" h="80" w="160">Describe this constructor...</comment>
        <statement name="PARAM">
          <block type="classes_parameter" id="b199671c-da1e-49d7-b3eb-5bfa413ab619">
            <field name="NAME">param1</field>
            <value name="TYPE">
              <block type="object_type" id="b8c5356d-1e1e-424b-a575-7167f0cd7904">
                <field name="TYPE_NAME">DOUBLE</field>
              </block>
            </value>
          </block>
        </statement>
      </block>
    </statement>
    <statement name="METHODS">
      <block type="object_method_return" id="89744bc4-fadc-49a9-8b68-27dc31a3ac30">
        <field name="NAME">myMethod</field>
        <field name="VISIBILITY">PUBLIC</field>
        <comment pinned="false" h="80" w="160">Describe this method... @param param1 @return
        </comment>
        <statement name="PARAM">
          <block type="classes_parameter" id="440bc44a-3b9f-4bc2-b12e-4c0381d8355b">
            <field name="NAME">param1</field>
            <value name="TYPE">
              <block type="object_string" id="898405b6-aa21-4486-8d73-16c1fd3f7d81"></block>
            </value>
          </block>
        </statement>
        <value name="RET">
          <block type="object_type" id="faeff99a-7fbc-41a0-a7a9-fc722af76814">
            <field name="TYPE_NAME">BOOLEAN</field>
          </block>
        </value>
      </block>
    </statement>
  </block>
</xml>
```

Figure 5.6: Codice XML che rappresenta la classe di figura 5.5

collegabili ad altri permettono di attivare una funzione che separa, anche graficamente scollegandoli e spostandoli sull'area di lavoro, blocchi che risultano incompatibili.

Poichè rimane un obiettivo primario la semplicità di utilizzo dell'applicazione che è rivolta a neofiti della programmazione object-oriented, si è cercato di ridurre il numero delle informazioni che vengono visualizzate e gestite per ogni blocco. Solo i componenti principali sono proposti, per fare un esempio nel caso di definizione di una classe non appare inizialmente la possibilità di definire una superclasse o un insieme di interfacce da implementare. Questo favorisce lo studente che affronta i primi semplici problemi e non viene distratto dalla presenza di informazioni in quel momento non necessarie.

Le API mettono a disposizione la gestione dei *mutator* che permettono, in qualsiasi momento, di modificare la struttura di un blocco aggiungendo alcune caratteristiche fra quelle previste. Anche questa operazione di "mutazione" della forma di un blocco è fornita all'utente mediante block programming e risulta quindi estremamente semplice da utilizzare.

In figura (fig. 5.7) un esempio più complesso che prevede l'estensione da una superclasse e l'implementazione di interfacce oltre alla gestione dei package.

Opzionalmente è anche possibile associare ad ognuno dei blocchi principali (interfacce, classi, metodi ecc.) un commento testuale che verrà poi incluso nella generazione del codice.

### 5.2.1 Funzionalità orientate ad applicazioni più complesse

La funzione di zoom permette di visualizzare a vari livelli di ingrandimento il workspace e passare quindi da una visione globale che permette di avere un quadro generale della struttura delle classi ad una più specifica di una classe o interfaccia in modo da analizzarne la struttura interna ed eventualmente effettuare ampliamenti e/o modifiche.

Per ogni blocco è inoltre applicabile una funzione di "collapse" (fig.5.8) che nasconde buona parte delle informazioni lasciando in evidenza solo le principali e una di "expand" che riporta il blocco alla sua rappresentazione originale. Questa funzione è attivabile su tutti i blocchi del workspace o singolarmente su ognuno di

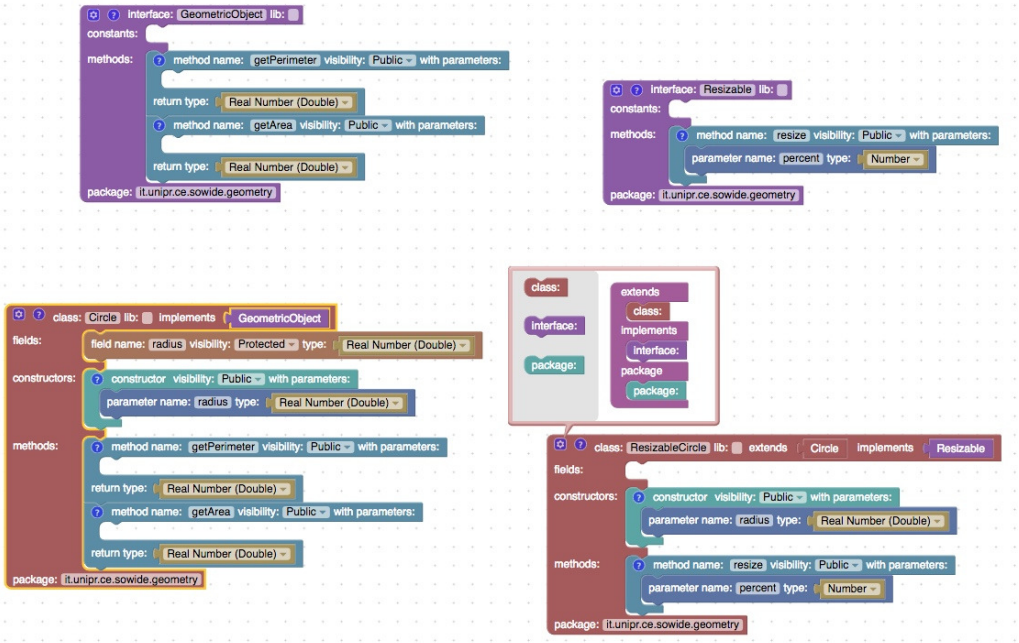



Figure 5.7: Classi, interfacce e package.

questi.

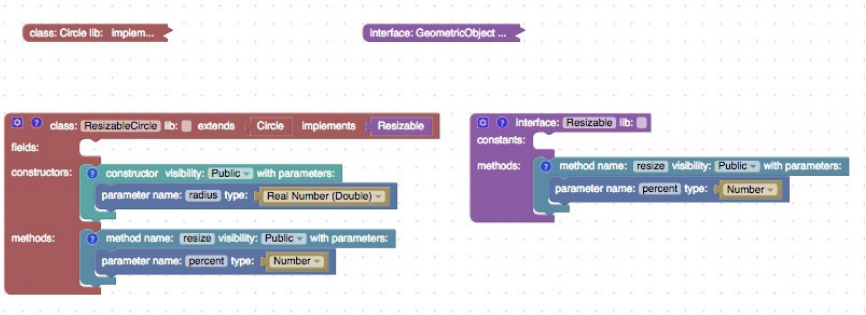



Figure 5.8: Blocchi “collapsed” ed “expanded”.

Nel caso di applicazioni di una certa complessità il codice generato in modo automatico può risultare piuttosto lungo, vista anche la verbosità intrinseca al linguaggio Java. Ogni blocco può essere facilmente *disabilitato* e successivamente *riabilitato*; poiché per i blocchi disabilitati non viene generato il codice si può sfruttare questa funzione per abilitare solo un sottoinsieme di blocchi per analizzarne più facilmente il codice o, in altri casi, per disabilitare temporaneamente blocchi che generano errori e quindi ottenere il codice anche per applicazioni non completamente sviluppate. Per le classi e le interfacce è possibile settare un apposito checkbox che sta a indicare che queste vengono fornite esternamente come librerie quindi i blocchi sono utilizzati solo come riferimento; di conseguenza come codice verrà generata una istruzione Java di `import`.

### 5.2.2 Generazione codice Java

Ogni volta in cui viene effettuata una modifica al puzzle dell’applicazione, inserendo, modificando o eliminando blocchi viene automaticamente generato e visualizzato il codice Java corrispondente. L’aggiornamento real time del codice ha la funzione, molto apprezzata dagli studenti, di evidenziare le caratteristiche sintattiche del linguaggio confrontandole con la più semplice struttura a blocchi. L’associazione blocco-codice è definita in file Javascript che, anche in questo caso, utilizzano le API di

Blockly per recuperare le varie informazioni interne al blocco e convertirle nel corrispondente codice Java. Per la nostra applicazione il codice di "conversione" è stato inserito nei file interfaces.js e classes.js e java.js presenti nella cartella blocks.

I vincoli di connessione dei blocchi eliminano la maggior parte degli errori sintattici, ma in ogni caso la funzione di generazione codice Java effettua una ulteriore analisi di correttezza, non solo sintattica, che fa sì che il codice generato sia sicuramente corretto. Questo è un aspetto molto importante in quanto l'applicazione è a scopo educativo e, come viene evidenziato da vari lavori in letteratura, gli errori sintattici o in genere dovuti a imprecisioni nel codice sono una delle principali cause di scoraggiamento per i neo programmatori.

A titolo di esempio citiamo una classica situazione di codice con errori anche se corretto dal punto di vista puramente sintattico: la mancanza di un metodo da parte di una classe che implementa un'interfaccia in cui questo metodo è specificato. Come visto in precedenza, mediante i mutator, è possibile dare un'organizzazione più complessa alle classi di una applicazione definendo per esempio una classe associata ad una specifica interfaccia.

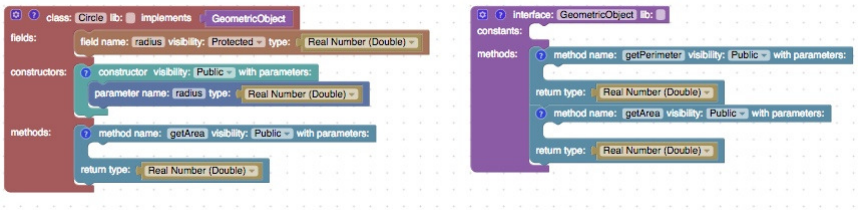



Figure 5.9: Esempio di errata implementazione di una interfaccia

In figura 5.9 vediamo un puzzle che rappresenta l'esempio proposto: la classe `Circle` implementa l'interfaccia `GeometricObject` ma non fornisce la rappresentazione del suo metodo `getPerimeter`. Dal punto di vista della sintassi dei blocchi non è violato alcun vincolo di collegamento ma, nel momento in cui tentiamo di ottenere il codice Java corrispondente, viene visualizzata una comunicazione

esplicativa del tipo di errore e il codice non viene generato. L'analisi viene fatta recuperando tutte le firme dei metodi definiti nelle interfacce implementate da una classe e confrontata con le firme dei metodi definiti nella classe stessa e solo in caso di corrispondenza viene generato il codice java. La funzione di generazione automatica del codice può essere eventualmente e temporaneamente disabilitata, questa opzione è suggerita per non rallentare la risposta dell'applicazione nel caso di progetti particolarmente complessi o prima del caricamento di grosse librerie.

### 5.2.3 Importazione da file jar

In associazione con l'ambiente di sviluppo è stata realizzata un'applicazione che permette di recuperare classi e interfacce con i relativi dettagli relativi da files .jar e genera blocchi compatibili con OOPP. Il file XML generato segue la sintassi di OOPP, è eventualmente modificabile e contiene tanti blocchi quante sono le classi e interfacce analizzate. Varie opzioni sono disponibili per fare in modo che i blocchi compaiano in modo *collapsed* o *expanded* al momento della visualizzazione nell'ambiente di lavoro o che classi e interfacce siano settate come *lib* e quindi non generino codice specifico ma una istruzione di `import`.

## 5.3 Sperimentazione

L'utilizzo di OOPP è stato sperimentato in un corso introduttivo alla programmazione nell'ambito di un più ampio corso di Istruzione e Formazione Tecnica Superiore per sviluppatori software. Il corso è stato incentrato su una metodologia object-early e preceduto da un breve ciclo di lezioni introduttive alla programmazione imperativa. L'obiettivo principale del corso è stato quello di introdurre i concetti della programmazione object-oriented e della progettazione di applicazioni software (OOD) mediante il linguaggio Java.

Al termine del corso è stato sottoposto agli studenti un questionario per valutare il tool OOPP in base alla loro esperienza d'uso. Più del 60% degli studenti (fig. 5.10) non aveva prima dell'inizio delle lezioni nessuna competenza di programmazione object-oriented e un ulteriore 16,7% non aveva mai sviluppato nessuna applicazione

object-oriented pur conoscendo dal punto di vista teorico il paradigma. Nella primis-

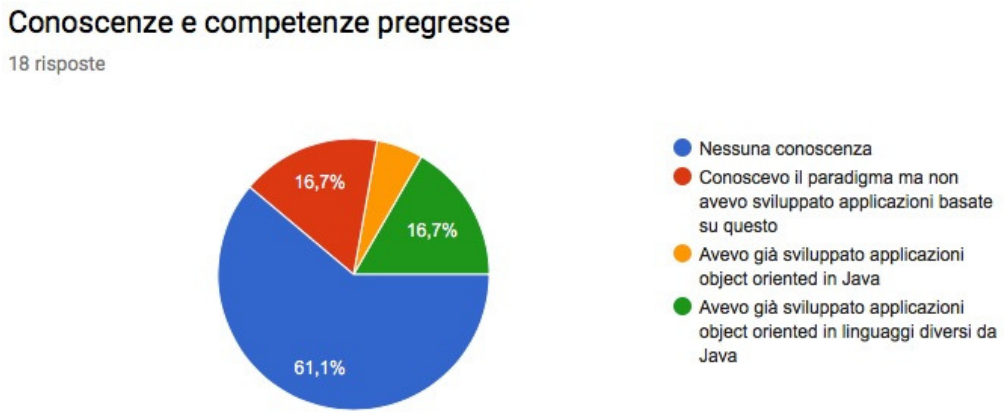



Figure 5.10: Competenze di programmazione pregresse

sima parte del corso le esercitazioni sono state caratterizzate dall'utilizzo e interazione di oggetti di classi predefinite prendendo spunto dall'approccio didattico presentato da Kölling [31] e utilizzando BlueJ [3] come ambiente di programmazione. Al momento della progettazione e realizzazione di classi e alla introduzione dei concetti di ereditarietà e polimorfismo è stato proposto agli studenti l'utilizzo del tool di progettazione OOPP. In una prima fase tutti gli studenti lo hanno utilizzato per progettare e generare la struttura delle classi per le loro applicazioni e la quasi totalità dgli studenti ha apprezzato la facilità d'uso del software (fig. 5.11) e anche fra i neofiti della programmazione object-oriented più del 90% (fig. 5.11) ha valutato semplice l'utilizzo di OOPP che viene ritenuto utile, almeno per la progettazione delle classi per i primi problemi proposti, dall'88,8% degli studenti. In particolare viene messo in evidenza il fatto che, operando mediante la composizione di blocchi, risulta più semplice sia la definizione dei vari componenti (attributi, costruttori, metodi) di una classe sia nel comprendere le strutture sintattiche mediante il codice generato in tempo reale durante la composizione del puzzle (fig. 5.13).

[3] www.bluej.org

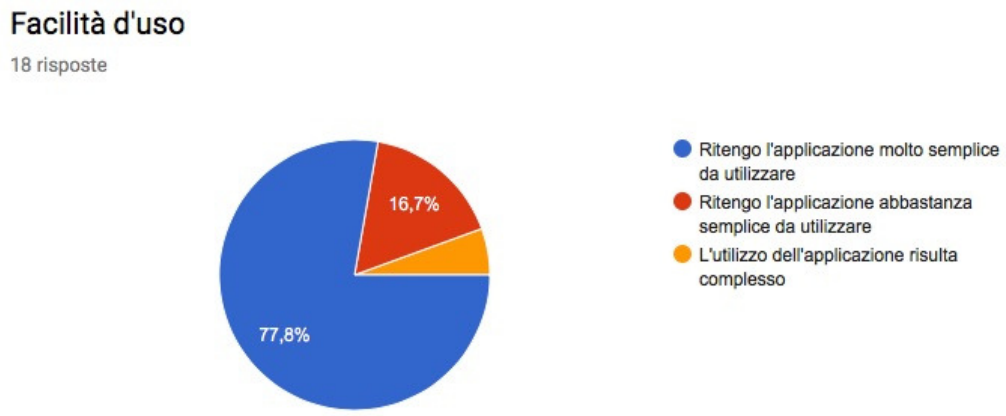



Figure 5.11: Facilità d'uso

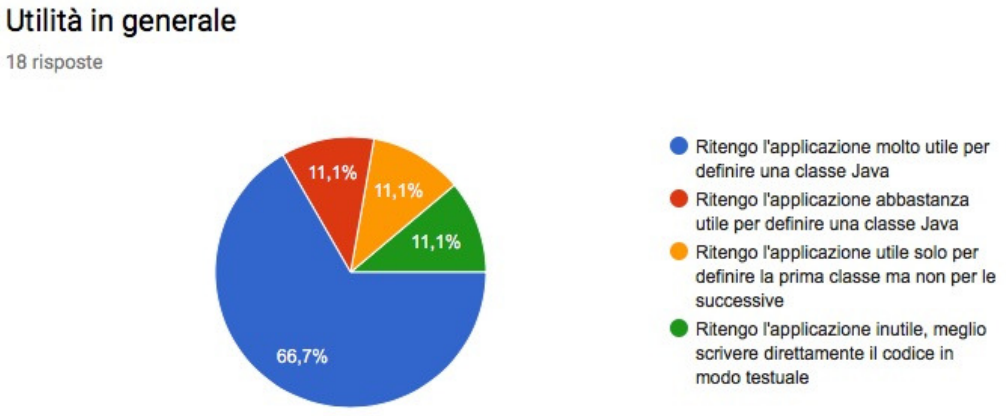



Figure 5.12: Utilità dell'applicazione

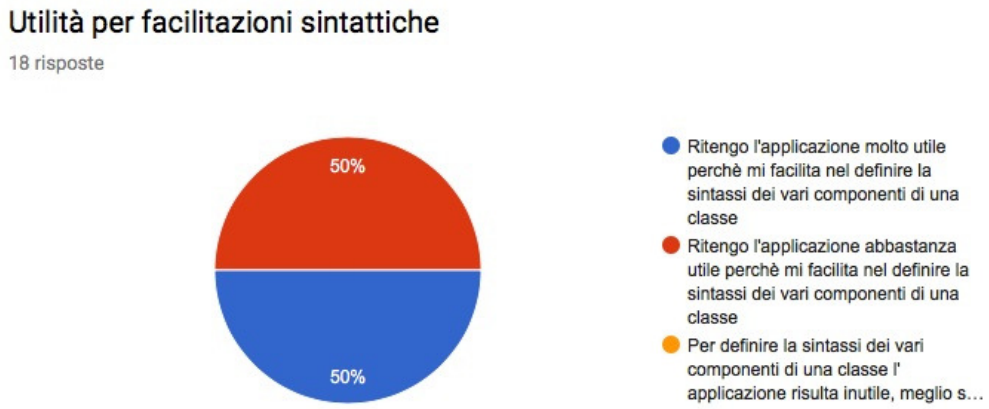



Figure 5.13: Facilitazioni sintattiche

## 5.4 Sviluppi futuri

Noi pensiamo anche in base ai risultati della prima sperimentazione che il nostro progetto possa essere utilizzato come ambiente iniziale per la modellazione e la progettazione di semplici applicazioni object-oriented in corsi CS1.

OOPP nasce come strumento per la programmazione object-oriented in generale e non rivolto ad un unico linguaggio anche se questo primo prototipo è basato sulla generazione di codice Java. La generazione automatica di codice in vari linguaggi potrà essere un valido aiuto per confrontare le differenze soprattutto, ma non solo, di natura sintattica.

L'applicazione è funzionante ed è stata sperimentata, come abbiamo visto, con successo relativamente alle sue funzionalità di design: la possibilità cioè di definire l'ossatura di una applicazione Java. Un possibile sviluppo futuro riguarda l'estensione dell'ambiente in modo da permettere la generazione di codice in linguaggi di programmazione che implementano il paradigma object-oriented. In particolare si è pensato di sviluppare una versione che permetta di "strutturare" applicazioni Python visto il crescente utilizzo di questo linguaggio in ambito didattico. A parte alcuni aggiustamenti nella struttura di alcuni blocchi, all'eliminazione delle interfacce e alla

possibilità di definire ereditarietà multipla occorre ovviamente ridefinire la parte di conversione blocco-codice, la struttura generale dell'applicazione rimane però quasi invariata.

Anche se non è stato l'obiettivo iniziale del progetto si può pensare ad un ampliamento di questo per permette lo sviluppo di una applicazione completa all'interno di OOPP. Per i costruttori e i metodi si rende quindi necessario fornire una implementazione utilizzando in parte i blocchi "operativi" presenti in altre applicazioni sviluppate con Blockly e definendo nuovi blocchi che permettano di istanziare oggetti e di operare con questi. Il puzzle complessivo risulterà ovviamente più ampio data la presenza dei blocchi "operativi" e quindi l'utilizzo sarà suggerito solo per applicazioni piuttosto semplici anche se le varie funzionalità di *zoom* e *collapse* possono permettere la gestione di un ambiente di lavoro anche "affollato" di blocchi.

# Part III

# Proposte metodologiche e tool didattici per corsi CS0

*"Whether you want to uncover the secrets of the universe, or you just want to pursue a career in the 21st century, basic computer programming is an essential skill to learn."*

– Stephen Hawkings

Nel capitolo 6 presentiamo una panoramica sull'evoluzione delle interfacce di comunicazione uomo-computer e in particolare su quelle legate alla programmazione. Analizziamo poi i vari progetti di sperimentazione delle interfacce di programmazione tangibili suddividendoli in base all'utilizzo di blocchi attivi o blocchi passivi. Il capitolo si conclude presentando varie sperimentazioni effettuate allo scopo di analizzare il rapporto degli studenti messi a confronto con le interfacce tangibili rispetto all'utilizzo dei tradizionali linguaggi e strumenti di programmazione.

Presentiamo poi nel capitolo 7 il progetto di un modello didattico basato su codOWood, un insieme di applicazioni per il Tangible Computer Programming che abbiamo sviluppato all'Università di Parma.

# Chapter 6

# Tangible User Interfaces

*"A picture is worth a thousand words. An interface is worth a thousand pictures."*

– Ben Shneiderman

## 6.1 Interfacce di comunicazione uomo-computer

Nel corso degli ultimi anni i computer sono diventati onnipresenti e sempre più persone hanno la necessità di interagire con questi. Non tutti hanno però familiarità con l'utilizzo delle nuove tecnologie e questo ha fatto nascere nel campo dell'informatica un nuovo ambito di ricerca che studia l'interazione uomo-computer (HCI - Human-Computer Interaction). L'obiettivo principale degli studi HCI è di abbattere le barriere e rendere il più possibile user friendly questo tipo di interazione.

L'evoluzione delle interfacce fra uomo e computer può essere classificata in 3 periodi principali [73]:

- Anni '50-'60
  - I computer sono utilizzati in modalità batch e in pratica non esiste interfaccia in quanto non c'è interattività

  - Come periferiche di input vengono utilizzati lettori di schede perforate e stampanti testuali come periferiche di output

- Dagli anni ’60 ai primi anni ’80
  - L’interazione con i sistemi di elaborazione e la programmazione stessa hanno preso la forma di una *colloquio* caratterizzato da linee di comando in un linguaggio formale dal forte rigore sintattico e semantico che in qualche modo si avvicina alla lingua inglese; si pensi a proposito ai sistemi operativi (Unix, DOS . . . ) e ai linguaggi di programmazione (COBOL, Fortran, Pascal, C . . . )
  - L’input/output avviane tramite terminali telescriventi (TTY) e terminali a schermo, per l’output si utilizzano stampanti testuali
- Anni ’80 fino ad oggi
  - Dal 1981 le workstation Xerox Star hanno proposto il primo sistema commerciale basato su interfacce grafiche (GUI) (fig. 6.1) rese poi popolari dal 1984 dai sistemi Macintosh e in seguito dai sistemi Windows.
  - Finestre, Icone, Menu e Mouse (WIMP - Window, Icon, Menu and Pointing device) sono diventati gli oggetti di base delle nuove interfacce e ancora oggi sono gli strumenti di interazione più utilizzati soprattutto dagli utenti meno esperti. A parte l’introduzione dei touch screen per l’input/output, la comparsa di nuovi widget, l’uso del colore e il miglioramento del realismo delle icone non si individuano sostanziali cambiamenti nelle interfacce moderne.

### 6.1.1 Interfacce post-WIMP

Le nuove proposte post-WIMP sono caratterizzate dal riconoscimento vocale, dal riconoscimento dei movimenti (gesture recognition) e dalle interfacce tangibili (TUI). Queste nuove modalità di interazione sono già in parte in uso e probabilmente diventeranno lo strumento di interfacciamento più utilizzato nei prossimi anni. Le interfacce

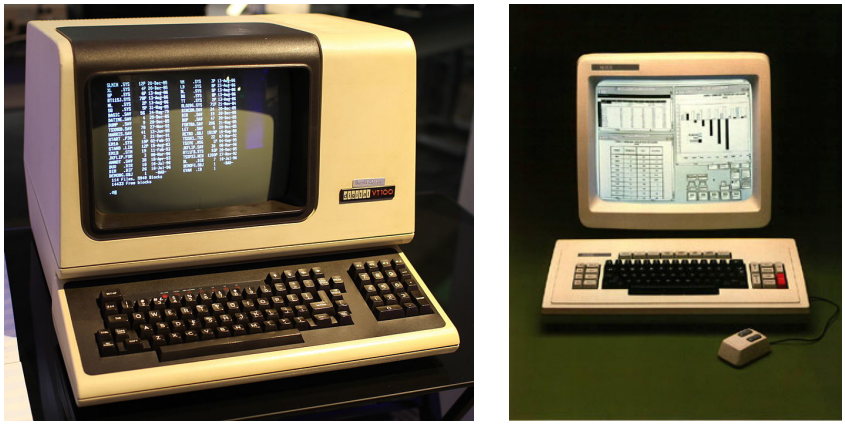


Figure 6.1: Interfaccia Testuale e Graphic User Interface

tangibili cambiano l'ottica del problema ribaltando la logica che fino ad ora si era posta l'obiettivo di arricchire il mondo virtuale del computer con oggetti con i quali interagire. Il nuovo approccio si basa invece sull'idea di arricchire gli oggetti del mondo reale con informazioni digitali gestibili dal computer.

Si parla quindi di "Tangible User Interface", termine coniato dal Tangible Media Group nel MIT Media Lab.

*"The goal is to change the "painted bits" of GUIs to "tangible bits," taking advantage of the richness of multimodal human senses and skills developed through our lifetime of interaction with the physical world."* [1]

Per quanto riguarda la HCI riportiamo qui la definizione data dallo Special Interest Group on Human-Computer Interaction:
*"Human-computer interaction is a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them."* [74]

HCI tratta sia gli aspetti tecnologici che quelli umani e studia come questi sono messi in relazione dalle interfacce di comunicazione. L'ambito di ricerca è molto ampio e analizza i vari aspetti dell'usabilità [75]:

[1] www.media.mit.edu/groups/tangible-media/overview

- la facilità di apprendimento (learnability)
  - l'utente è subito operativo
- l'efficienza (efficiency)
  - miglioramento della produttività
- la facilità di memorizzazione (memorability)
  - l'uso del sistema risulta facile anche dopo lunghi periodi di non utilizzo
- la robustezza agli errori (errors)
  - deve risultare "difficile" commettere errori
- il soddisfacimento (satisfaction)
  - il sistema deve essere "piacevole" da usare

## 6.2 Tangible Computer Programming

Lo studio sulle interazioni fra uomo e computer è, come abbiamo accennato, ampio ed è trattato in modo interdisciplinare da vari ambiti di ricerca: informatica, sociologia, psicologia e disegno industriale. Una trattazione approfondita esula quindi da questo testo il cui l'interesse è centrato sull'analisi delle interfacce fisiche nel campo della programmazione. Andremo quindi ad analizzare e proporremo poi un prototipo di ambiente per lo sviluppo di semplici applicazioni informatiche che si pone come obiettivo principale quello di facilitare l'interazione fra il programmatore e il computer mediate l'utilizzo di TUI.

Questo argomento di ricerca riscuote un sempre maggior interesse e sempre più aziende e laboratori di ricerca stanno impegnando risorse, per proporre strumenti che consentono di programmare mediante l'interazione diretta con oggetti fisici.

Il Tangible Computer Programming è legato strettamente al paradigma di programmazione a blocchi che, in questo caso, diventano blocchi "tangibili", ossia oggetti fisici.

Il termine “Tangible Programming Language” è stato coniato da Suzuki e Kato per descrivere AlgoBlocks [76], il primo linguaggio tangibile di programmazione presentato in ambito di ricerca sul collaborative problem solving. Ciò che lo rende differente dagli altri ambienti di programmazione è infatti la tangibilità della sua interfaccia: invece di manipolare oggetti virtuali visualizzati sullo schermo di un computer, gli utenti di AlgoBlock manipolano blocchi fisici disposti su di un tavolo [77].

### 6.2.1 Progetti di Tangible Computer Programming

In molti contesti educativi gli oggetti materiali con cui i bambini interagiscono giocano un ruolo importante nell'apprendimento di concetti matematici e scientifici. [78]

Fin dagli anni '60 sono stati creati un gran numero di linguaggi di programmazione volti ad eliminare o almeno a facilitare il processo di apprendimento sintattico della programmazione [44]. Nonostante esistano altri ostacoli concettuali [79] non inerenti la sintassi lo sforzo è sempre rivolto a rendere la programmazione sempre più accessibile a un numero sempre maggiore di persone.

Più recentemente una parte di questi studi si è concentrata sulla progettazione di linguaggi di programmazione tangible [80].

Presentiamo qui una rassegna di progetti e prodotti basati su Tangible User Interfaces:

- E-Block

  Basato su un precedente lavoro, T-Maze [81], E-Block [82] è un tool di programmazione pensato per bambini dai 5 ai 9 anni che comprende blocchi dotati di sensori che comunicano con il computer tramite microprocessori e una wireless box (figura 6.2). I simboli presenti sui blocchi permettono al bambino i comprendere il significato del blocco stesso.

  L'obiettivo finale è quello di fornire la sequenza di azioni necessarie per individuare la via di uscita da un labirinto. Il posizionamento di un blocco nella sequenza viene immediatamente controllato e accettato solo in caso si tratti di un'azione corretta. Il tasto di run permette poi al bambino di verificare se la

sequenza inserita porta effettivamente alla soluzione del problema, l'uscita dal labirinto.

Figure 6.2: E-Block

- Osmo (Apple) [2]
  Osmo è un accessorio per iPad pensato per la didattica. Si tratta di un sistema che include una base di supporto, uno specchio e varie tessere specifiche per un insieme di giochi disponibili (figura 6.3). Fra le varie applicazioni proposte una delle più interessanti è sicuramente *Coding Awbie* che presenta varie forme in legno che, composte fra loro, vanno a formare un puzzle che rappresenta l'insieme di azioni da far compiere ad Awbie, un personaggio virtuale che si muove sullo schermo.

  Le applicazioni di Osmo effettuano il riconoscimento dei pezzi dei vari giochi mediante riconoscimento ottico che in questo caso viene definito "Reflective Artificial Intelligence".

  Sempre basato sull'accessorio Osmo, Strawbies [83] ha una struttura analoga sempre basata sul riconoscimento ottico dei vari blocchi che in questo caso sono dotati di marcatori TopCode.

[2] www.playosmo.com

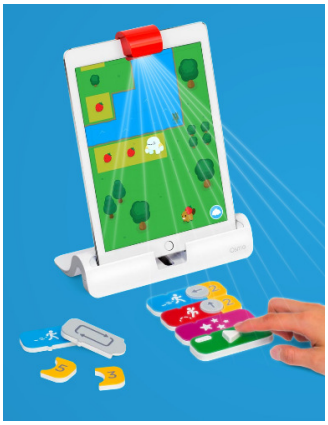


Figure 6.3: Osmo

- Tern (Tufts University HCI Lab) [3]
  Il linguaggio Tern [84] è stato progettato per essere un ambiente educativo che permette ai bambini di creare programmi collegando blocchi di legno per rendere più semplice e accattivante l'approccio al coding [85]. Con Tern (figura 6.4) è possibile creare programmi per robot come *LEGO Mindstorms RCX* o *iRobot Create*. Tern è presente in una mostra permanente presso il Boston Museum of Science [86] e viene inoltre utilizzato per il progetto Tangible Kindergarten presso il gruppo Tufts University Developmental Technologies. I blocchi di legno di Tern che possono rapprentare azioni, costrutti di controllo del flusso o parametri, non contengono elettronica embedded. Per il riconoscimento dei blocchi, su cui sono presenti specifici marker, Tern utilizza una webcam collegata a un computer per ottenere un'immagine del "programma", che poi viene convertito in codice digitale utilizzando la *TopCodes Computer Vision Library*.

- Electronic Blocks
  Il progetto Electronic Blocks [87] utilizza blocchi Lego (figura 6.5) progettati per consentire ai bambini di creare oggetti che possono compiere varie azioni.

[3] hci.cs.tufts.edu/tern


Figure 6.4: Tern

Sono presenti tre tipi di blocco: *sensor block* con funzionalità di input; *logic block* con funzionalità operative; *behavior block* con funzionalità di output.

In ogni blocco è incorporato un processore oltre ad altri dispositivi elettronici essenziali. Il numero relativamente limitato di blocchi rende semplice la sintassi ma la presenza di componenti elettroniche in ognuno di questi rende piuttosto alto il costo dell'intero sistema.

Figure 6.5: Electronic Blocks

- Cubetto (Primo Toys) [4]
  Cubetto è un piccolo robot (*Cubetto*) che, tramite una serie di comandi, può essere guidato nei suoi spostamenti.

  Il bambino viene messo al centro dell'esperienza formativa, deve infatti fornire le istruzioni per fare in modo che il piccolo robot ritorni a casa. Per fare questo ha a disposizione un piano con una serie di fori in cui posizionare vari tipi di blocchi in modo da dirigere il robottino durante il suo percorso (figura 6.6).

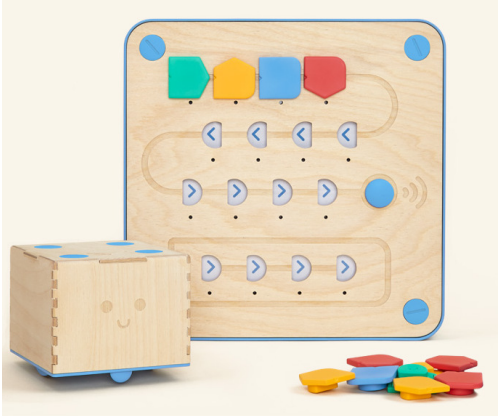


Figure 6.6: Cubetto

Nell'interfaccia di Cubetto è presente la linea Function che permette di creare una sequenza di operazioni che può essere richiamata e ripetuta più volte all'interno del percorso programmato. Creare funzioni, in questo caso piccole sequenze, è uno degli elementi fondamentali della programmazione ed è una parte molto importante nell'esperienza di apprendimento che il bambino fa con Cubetto.

Il kit di Cubetto è composto da 16 blocchi di comando, il cubo robot, l'interfaccia per programmare i suoi movimenti, una mappa sui cui farlo viaggiare e una serie di storie da seguire.

[4] www.primotoys.com

- Cubes Coding [5]
  Cubes Coding (figura 6.7) è una piattaforma di programmazione per la robotica. La versione tangibile consiste in 28 cubi più grandi che rappresentano 28 comandi differenti e 16 cubi più piccoli che rappresentano 16 diversi parametri. Gli utenti possono programmare un robot collegando fra loro in sequenza i cubi.

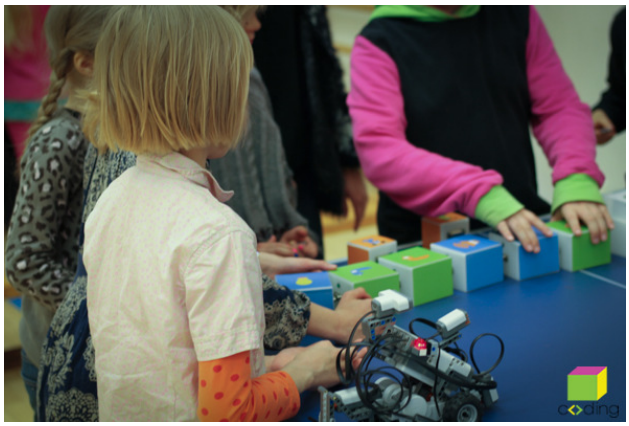


Figure 6.7: Cubes Coding

- Tangible Programming Bricks
  I Tangible Programming Bricks [88] sono mattoncini Lego che possono essere impilati per formare programmi (figura 6.8). Ogni blocco può comunicare con gli altri blocchi tramite trasmissione ad infrarossi. Impilando i vari blocchi gli utenti possono comunicare e impartire comandi a giocattoli e utensili di cucina quali, per esempio, forni a microonde. L'approccio alla programmazione è di tipo procedurale ma manca la possibilità di variare il flusso lineare con costrutti di programmazione quali condizioni e cicli.

- Quetzal
  Quetzal [89] è un linguaggio di programmazione tangible progettato per i

[5] www.cubescoding.com


Figure 6.8: Tangible Programming Bricks

bambini e per gli apprendisti programmatori che permette di controllare un robot *LEGO Mindstorms*. Consiste di oltre un centinaio di blocchetti (figura 6.9) collegabili fra loro. i vari blocchi rappresentano strutture di controllo di flusso, comandi e dati. I programmatori organizzano e collegano queste blocchetti per definire algoritmi che possono includere cicli e ramificazioni del flusso di esecuzione.


Figure 6.9: Quetzal

- TurTan
  TurTan (figura 6.10) è un sistema basato sulla geometria della tartaruga tipica di *LOGO*. Una videocamera cattura in real-time la posizione e il movimento di vari oggetti sull'area di lavoro e li converte in comandi per far muovere una tartaruga virtuale [90]. Il sistema di rilevamento è piuttosto complesso ed ha un costo piuttosto elevato.

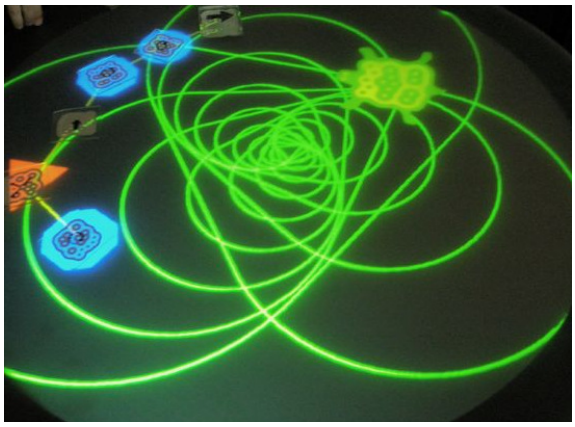


Figure 6.10: TurTan

- Project Blocks (Google) [6]
  Project Blocks (figura 6.11) è un progetto educativo realizzato in collaborazione fra Google Creative Lab, Google Research & Education Team, IDEO e il Transformative Learning Technologies Lab dell'Università di Stanford .

  Dato il proliferare di singoli progetti spesso proprietari e commercializzati ai fini di uso personale, la proposta di Project Blocks è quella di sviluppare una piattaforma comune che permetta di realizzare facilmente ambienti di tangible programming concentrando gli sforzi in direzione didattica piuttosto che tecnologica. Nell'idea del gruppo di ricerca Project Blocks dovrebbe rappresentare in ambito di tangible programming ciò che nel settore di block programming

[6] projectbloks.withgoogle.com

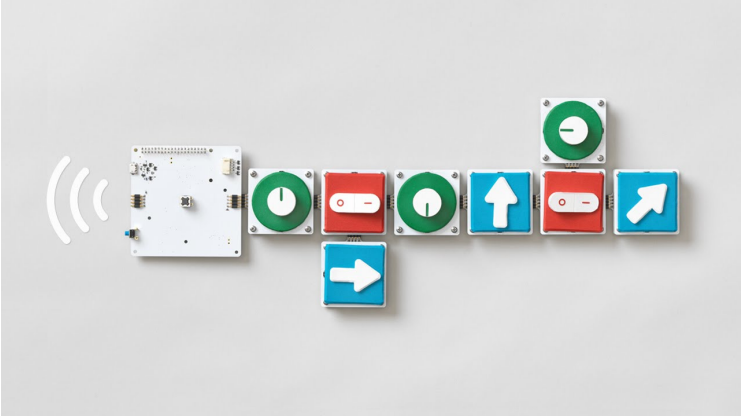


Figure 6.11: Project Blocks

è oggi Google Blockly: un piattaforma semplice e gratuita comune a nuovi progetti di ricerca.

Si tratta di una piattaforma hardware aperta che permette di arricchire i concetti legati al codice di una dimensione fisica, tangibile.

Così Google descrive il progetto: *'Project Blocks è un progetto di ricerca. Il nostro obiettivo è quello di creare una piattaforma hardware aperta per aiutare sviluppatori, designer e ricercatori a costruire una nuova generazione di esperienze di programmazione tangibili per i ragazzi.'*

Project Blocks è una piattaforma costituita da una serie di moduli che si possono combinare in maniere differenti per creare progetti di vario tipo.

Il blocco *Brain Board* ha la funzione di processore e alimentatore per gli altri moduli ed è basato sulla tecnologia di Raspberry Pi Zero; è dotato di altoparlanti, Wi-Fi e Bluetooth.

Il secondo modulo è composto dai pulsanti (*Pucks*) che servono a impartire vari tipi di semplici comandi ("accendi", "sposta", "ruota", "salta", "riproduci musica", ecc.).

I blocchi vanno posizionati sul modulo di base (*Base Board*) e devono essere

collegati uno all'altro per poter funzionare.

La strada scelta è quindi quella dei blocchi attivi, blocchi standard a basso costo dotati di tecnologia di comunicazione.

Con la piattaforma Blocks è possibile creare dei sistemi in grado di interfacciarsi con dispositivi come i Mirobot[7] o la piattaforma LEGO WeDO 2.0[8], per creare interfacce di controllo o piccoli sistemi di domotica.

L'intenzione di Google non è quella di commercializzare il prodotto ma di fornire una base di sviluppo a progetti di Tangible Programming. Il progetto è tuttora in fase di sviluppo.

## 6.3 Tipologie di blocchi

I sistemi di sviluppo basati sul tangible programming, di cui abbiamo presentato una, seppur non esaustiva, rassegna possono essere suddivisi in due macrocategorie in base alle caratteristiche degli oggetti fisici che vengono manipolati:

- blocchi attivi
- blocchi passivi

### 6.3.1 Blocchi attivi

La maggior parte dei progetti di Tangible Computer Programming, attualmente in commercio o in fase di sviluppo utilizza blocchi che contengono al loro interno componenti elettronici. Soprattutto i primi progetti degli anni '90 sono basati su blocchi attivi e ciò comporta un alto costo complessivo del sistema derivante proprio dalla presenza di queste componenti elettroniche. Nonostante la diminuzione negli anni del costo dei componenti elettronici, questo fatto ne limita la sperimentazione e l'utilizzo pensando soprattutto a un utilizzo per la diffusione del coding nei paesi in via di sviluppo.

[7] mime.co.uk

[8] elearning.legoeducation.com/wedo-2-0

La comunicazione fra i vari blocchi e il software di gestione è effettuata mediante WiFi o bluethooth e viene gestita direttamente da ogni singolo blocco o da un blocco specifico preposto alla trasmissione. In [91] troviamo una proposta originale, ma dal costo certamente elevato, in cui ogni blocco è costituito da uno smartphone (fig. 6.12).

Figure 6.12: Click - Physical coding for education

### 6.3.2 Blocchi passivi

Dopo i primi lavori basati su blocchi attivi vengono in seguito presentati e sperimentati progetti che eliminano ogni componente elettronica interna ai blocchi e sfruttano per il loro riconoscimento tag RFID passivi o algoritmi di image recognition.

Nel caso di image recognition, l'immagine dei blocchi e quindi la successiva elaborazione di riconoscimento vengono effettuate dal dispositivo centralizzato utilizzando in alcuni casi specifiche apparecchiature di ripresa o in altri le fotocamere ormai presenti in molti sistema di elaborazione.

Come vedremo il nostro progetto di ricerca sarà basato su blocchi passivi e sfrutterà sistemi di image recognition che non saranno però gestiti in maniera centralizzata ma da unità distribuite.

## 6.4 Analisi dei livelli di apprendimento in relazione alle User Interface

Gli studi che analizzano le diverse tipologie di User Interface [92] allo scopo di indagare i risultati di apprendimento raggiunti dagli studenti nel campo della programmazione sono pochi e spesso empirici. Fra questi possiamo citare Strawhacker e Bers [92] che descrivono una sperimentazione basata sul confronto fra tre UI (Tangible, Graphical e Hybrid) utilizzate per programmare i movimenti di un robot da parte di bambini di 4-6 anni. Si tratta di analizzare se e come lo stile dell'interfaccia (tangible o graphical) condiziona la capacità di apprendimento, da parte dei bambini, dei concetti di programmazione. In questo caso l'analisi qualitativa e quantitativa non porta però a significativi risultati.

Nella ricerca di Sapounidis e Demetriadis [93] in cui si mettono a confronto interfacce isomorfe grafiche e tangibile l'analisi quantitativa e qualitativa indica che l'interfaccia tangibile è stata ritenuta più attraente soprattutto per le ragazze, ed è risultata più piacevole e facile da usare per gli studenti più piccoli che erano meno esperti nell'uso del computer. Al contrario, per i più grandi (11-12 anni), gli oggetti tangibili, anche se più divertenti, non sono stati considerati come l'interfaccia utente più facile da utilizzare.

Altri studi relativi agli ambienti tangible mettono in evidenza l'attrattività e soprattutto la facilità d'uso di questi ambienti confrontandoli con altre modalità di interazione più tradizionali [94] [95]. In particolare l'utilizzo di oggetti fisici per interagire con i computer favorisce maggiormente il coinvolgimento dei bambini e stimola la loro attenzione e partecipazione [96].

# Chapter 7

# codOWood

Programmare il computer con piccoli pezzi di legno

*"When you're a carpenter making a beautiful chest of drawers, you're not going to use a piece of plywood on the back, even though it faces the wall and nobody will ever see it. You'll know it's there, so you're going to use a beautiful piece of wood on the back."*

– Steve Jobs

Il progetto codOWood è stato sviluppato a livello prototipale dal laboratorio di ricerca SoWIDE [1] dell'Università di Parma. Si tratta di una proposta didattica pensata per l'introduzione al coding in ambito di scuola primaria basata su un ambiente di programmazione tangible.

Il progetto utilizza tecnologie a basso costo ed è strutturato in modo da favorire il lavoro collaborativo. Le varie funzionalità sono state testate a livello prototipale fornendo risultati più che soddisfacenti.

Il nostro progetto di Tangible Computer Programming prevede blocchi passivi di materiale povero (legno) che vengono individuati mediante image recognition e "trasformati" in blocchi logici per applicazioni di Block Programming basate su Google Blockly.

[1] sowide.ce.unipr.it


Figure 7.1: Blocco fisico di codOWood

## 7.1 L'ambiente didattico di codOWood

### 7.1.1 Aule aumentate

L'ambiente didattico proposto per l'utilizzo di codOWood è caratterizzato da un'aula aumentata dalle tecnologie in cui è presente una postazione centralizzata di condivisione, e una serie di spazi in cui operano gli alunni suddivisi in piccoli gruppi (figura 7.2).

La struttura e le modalità di interazione fra i gruppi favoriscono l'apprendimento laboratoriale pur non necessitando di costose apparecchiature elettroniche e di strutture particolari. L'insieme delle tecnologie proposte è di basso costo e facilmente reperibile e il software è semplice da utilizzare; questi fattori sono importanti perché rendono il sistema sostenibile per la maggior parte delle realtà didattiche.

La postazione di condivisione è gestita dal docente che propone alla classe un problema scelto fra un insieme di temi in base al livello precedentemente raggiunto e alle competenze logiche degli studenti. Il problema consiste nel definire la sequenza di azioni che devono compiere personaggi che si muovono all'interno di un ambiente virtuale, in figura 7.3 vediamo un classico esempio di labirinto. L'ambiente e i personaggi sono visualizzati su uno schermo condiviso (LIM o videoproiettore) della postazione docente. Sempre sullo schermo condiviso verranno poi visualizzate le soluzioni proposte dai vari gruppi di studenti.

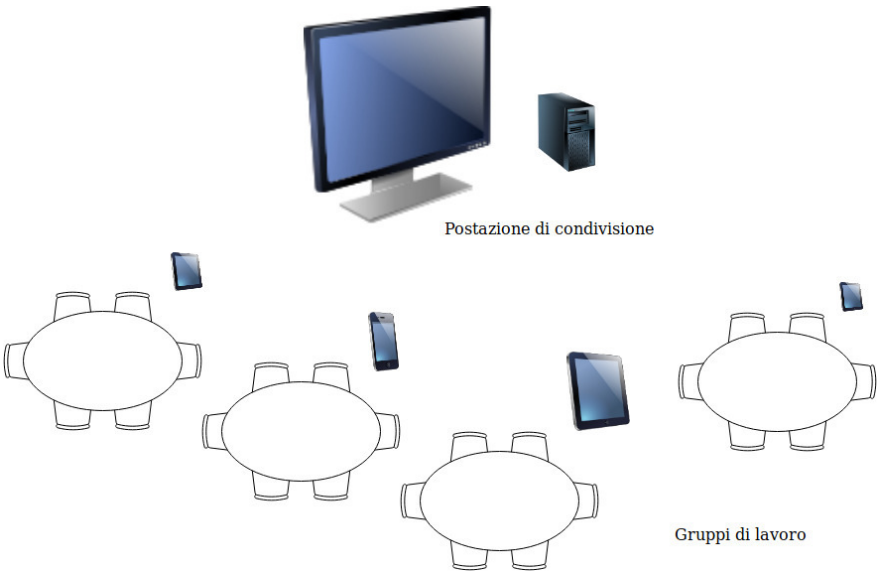



Figure 7.2: Ambiente didattico

Le postazioni per gli alunni prevedono una superficie su cui disporre e collegare fra loro i blocchi fisici e un dispositivo per acquisire l'immagine del puzzle-programma e per comunicarla alla postazione di condivisione; in figura 7.4 vediamo un esempio di algoritmo formato da blocchi concatenati.

### 7.1.2 Postazione di condivisione

Uno degli obiettivi che ci siamo posti già in fase di progettazione è stato il contenimento dei costi e la possibilità di utilizzare strumentazione spesso già presente negli ambienti scolastici. Per questa ragione l'implementazione di tutte le funzionalità della postazione di condivisione è stata effettuata su un single-board computer Raspberry Pi del costo complessivo di poche decine di Euro.

Questa postazione ha le funzionalità di server per l'intero sistema e si occupa della configurazione di una intranet per permettere la condivisione di informazioni con i dispositivi client utilizzati dagli alunni.

L'utilizzo di Raspberry Pi tuttavia non è necessario e il software server codOWood

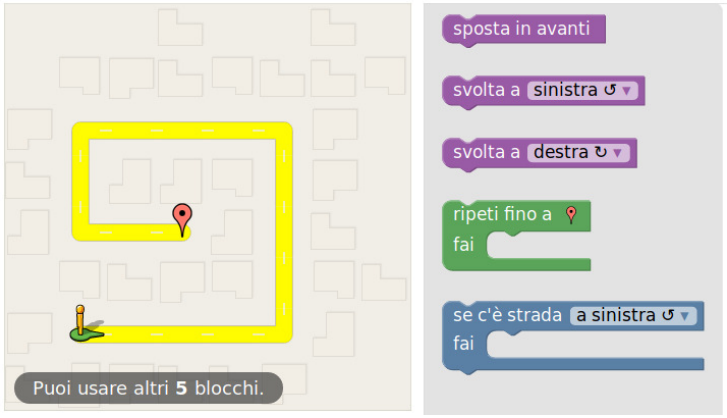



Figure 7.3: Un semplice esempio di problema

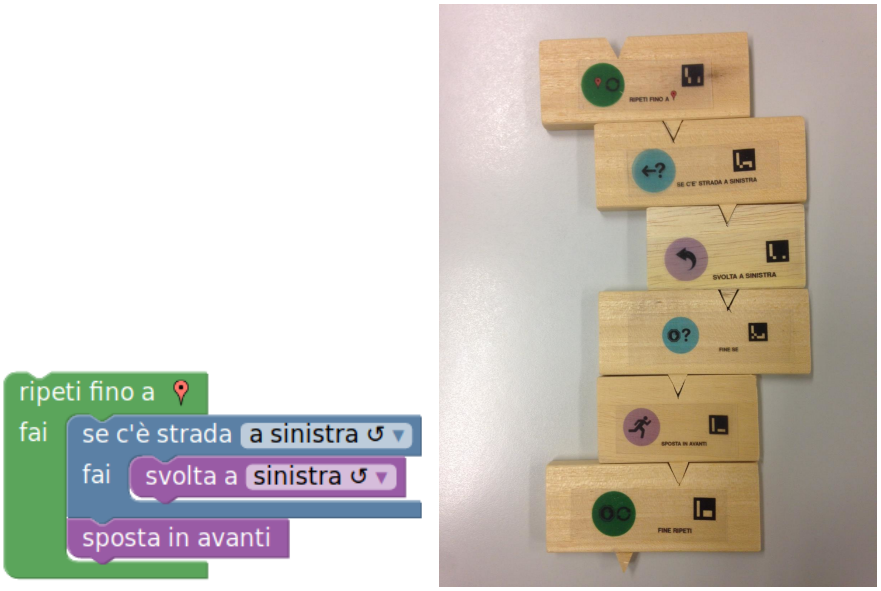



Figure 7.4: Un esempio di algoritmo

può essere installato su un qualsiasi computer eventualmente già presente in aula.

Per la visualizzazione dei problemi e delle varie proposte di soluzione è necessario uno schermo comune che può essere un qualsiasi monitor collegato al server, un videoprietttore o una lavagna interattiva multimediale (LIM).

### 7.1.3 Postazioni alunni

Gli alunni sono suddivisi in piccoli gruppi che operano indipendentemente ma in stretto contatto con il docente. Ogni gruppo ha ha disposizione un insieme di blocchi di legno che può disporre su uno spazio di lavoro (una qualunque superficie piana) in modo da realizzare un puzzle che rappresenta il flusso operativo e l'insieme dei comandi che portano alla soluzione del problema.

Ogni gruppo utilizza uno smartphone o tablet sul quale è installata l'app client di codOWood che riproduce l'ambiente virtuale del problema e implementa varie funzionalità:

- interazione con il server per la registrazione del gruppo di lavoro
- acquisizione dell'immagine del puzzle realizzato
- riconoscimento della struttura complessiva del puzzle e di ogni singolo blocco mediante una procedura di image recognition partendo dall'immagine acquisita e successiva conversione in codice operativo codOWood
- interazione con in server per l'invio della proposta di soluzione

## 7.2 Motivazioni alla base del progetto

### 7.2.1 Coding nella scuola primaria

Il grande sforzo oggi in atto per far sì che tutti i bambini abbiano la possibilità di imparare la programmazione nasce da concetto formulato da Seymour Papert oltre 50 anni fa: *i bambini sono gli insegnanti e le macchine diventano gli allievi* [97] [98].

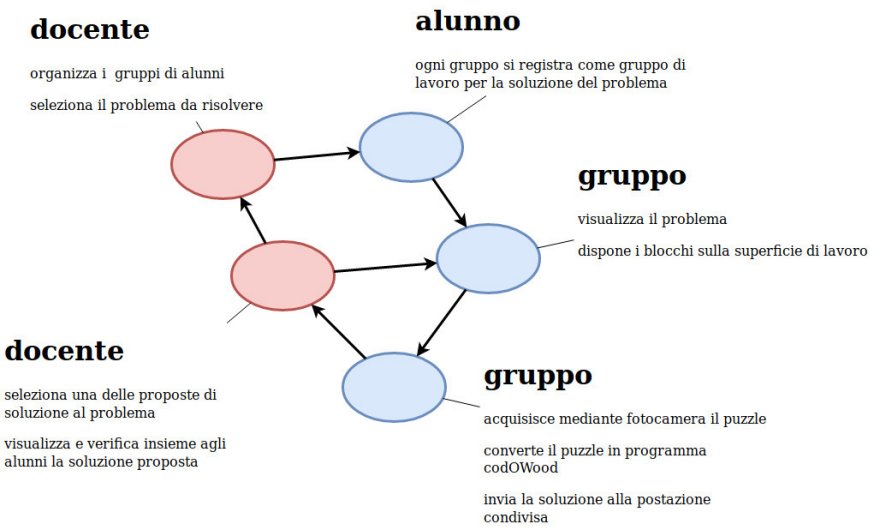



Figure 7.5: Funzionalità del sistema

Papert [4] e Resnick [99] hanno dimostrato che imparare i concetti fondamentali di programmazione comporta un cambiamento nel modo di pensare delle persone.

L'obiettivo di questi primi approcci al coding non è quello di creare professionisti della programmazione ma di far acquisire ai giovani un nuovo modo di pensare e di vedere il mondo, di padroneggiare la tecnologia e non esserne soltanto dei consumatori. La programmazione non è soltanto un job skill ma una literacy fondamentale che ognuno deve imparare nel 21 secolo. [100].

L'apprendimento dei concetti di base della programmazione è al posto al centro di un progetto più ampio che abbatte le barriere dell'informatica e stimola un approccio indirizzato alla risoluzione dei problemi. Con il coding bambini e ragazzi sviluppano il pensiero computazionale e l'attitudine a risolvere problemi più o meno complessi. Il coding favorisce il pensiero creativo e stimola la curiosità, attraverso il coding si "dialoga" con il computer, l'obiettivo non è formare una generazione di futuri programmatori, ma educare i più piccoli al pensiero computazionale, che è la capacità di risolvere problemi applicando la logica e ragionando sulla strategia migliore per

giungere alla soluzione.

Il nostro progetto è quindi pensato per bambini e bambine della scuola primaria anche se può facilmente essere rimodellato nella tipologia dei problemi e nella semantica dei blocchi per adattarlo a studenti dei primi anni di scuola secondaria.

### 7.2.2 Tangible Computer Programming: le motivazioni della scelta

I classici linguaggi di programmazione testuali sono in genere pensati per programmatori professionali e sono costituiti da termini e simboli spesso difficili da comprendere per i bambini. Le difficoltà di natura sintattica spesso impegnano i giovani programmatori che invece dovrebbero concentrarsi soprattutto sul procedimento di soluzione dei problemi.

A differenza di questi linguaggi in cui i programmi sono costituiti da sequenze di linee di testo, nel block programming le istruzioni sono rappresentate da blocchi che vengono collegati fra loro a formare un puzzle che rappresenta il programma.

Il Block Programming è il modo più semplice e immediato per avvicinarsi al mondo del coding e infatti è alla base della maggior parte dei progetti indirizzati ad un pubblico di giovani e giovanissimi. Dal punto di vista didattico, la programmazione a blocchi consente di apprendere le basi del coding, aiuta a sviluppare la logica, stimola la creatività ed educa al pensiero computazionale, a ragionare sui problemi e sul modo migliore per risolverli.

Scratch o Scratch Jr. per i più piccoli sono un esempio di strumenti divertenti e lo stesso si può dire dei problemi presenti nel sito code.org. Sono giochi, ma sono anche sfide a risolvere problemi più o meno complessi la cui soluzione è un vero e proprio programma anche se non sono visibili le righe di codice ma un insieme di blocchi colorati e di varia forma collegati fra loro a formare un puzzle. Si parla in questo caso di Graphical User Interfaces (GUI) poiché i semplici ambienti di sviluppo educativi si basano sul drag & drop di blocchi virtuali che vengono selezionati e disposti in modo da formare il puzzle-programma.

Se i blocchi-istruzione non sono più virtuali ma diventano oggetti fisici la programmazione avviene manipolandoli, spostandoli e incastrandoli in un ambiente reale,

in questo caso si parla allora di Tangible User Interfaces e di Tangible Computer Programming.

### 7.2.3 Sostegno all'interazione di gruppo: cooperative learning

Gli ambienti di sviluppo per i linguaggi di programmazione tangible permettono a piccoli gruppi di studenti di progettare e realizzare programmi in modo collaborativo. Questo risulta più difficile con i classici ambienti di programmazione basati sostanzialmente sull'utilizzo di mouse e/o tastiera, strumenti che possono essere utilizzati da un solo programmatore alla volta.

La possibilità di interagire a più mani sui blocchi disposti su un piano di lavoro favorisce quindi il lavoro di gruppo e l'interazione faccia a faccia fra gli studenti, favorisce inoltre lo sviluppo di una leadership distribuita all'interno di ogni gruppo. Tutti questi sono aspetti importanti che ritroviamo nel concetto di cooperative learning: ogni componente del gruppo può selezionare e posizionare i vari blocchi ma l'obiettivo, il puzzle finale, è comune e quindi risulta importante la relazione e l'interazione tra studente e studente.

L'obiettivo primario è ovviamente quello di introdurre i concetti del pensiero computazionale, ma questo modo di operare in gruppo educa anche ai comportamenti sociali e alle capacità comunicative, finalità certo di non minore importanza.

È importante inoltre sottolineare la figura del docente che ha un ruolo di guida che esercita monitorando e intervenendo nei lavori dei piccoli gruppi e nel gestire la presentazione e la discussione delle varie soluzioni proposte a tutta la classe.

## 7.3 Implementazione

### 7.3.1 Server

La postazione condivisa gestita dal docente è responsabile della gestione della comunicazione con le altre componenti del sistema. Come stabilito in fase di progetto deve rispondere a una serie di requisiti che riteniamo fondamentali per una effettiva spendibilità dell'intero sistema:

- utilizzo di hardware già presente nella sede didattica o in alternativa operabilità su hardware di basso costo,
- facilità di installazione e utilizzo sia in presenza di una rete preesistente sia in assenza di questa,
- possibilità di collegamento con ogni tipo di periferica di visualizzazione.

Sul server opera una web application scritta in Python che utilizza il framework open source Pyramid. L'applicazione configura automaticamente un rete locale wireless aperta alla comunicazione con i vari dispositivi client.
Nel caso sia presente nell'aula un computer per il docente, il software è di facile installazione e non presenta particolari requisiti hardware o di sistema software.
Nel caso invece in cui non sia già presente un computer o non si ritenga opportuno utilizzarlo, l'applicazione è stata installata e testata in tutte le sue funzionalità su un single-board computer Raspberry Pi con sistema operativo Raspbian. Questo hardware è facilmente reperibile e acquistabile a un costo complessivo di poche decine di Euro. La scelta di Raspberry Pi è dettata, oltre che per il basso costo, dal fatto che questo sistema è stato concepito proprio con l'intento di realizzare un dispositivo economico e utile per stimolare l'insegnamento di base dell'informatica e della programmazione nelle scuole. Tramite l'uscita per la periferica video del Raspberry è possibile poi collegare un dispositivo per la visualizzazione dei problemi e delle soluzioni proposte in modo da permettere la condivisione fra il docente e i vari gruppi di studenti .

### 7.3.2 Client

Ogni gruppo di lavoro dispone di un dispositivo che deve essere in grado di ottenere un'immagine del puzzle di blocchi, di convertirla in codice codOWood e di gestire la comunicazione con il server. Anche in questo caso abbiamo stabilito, in fase di progettazione, una serie di requisiti che hanno condizionato la scelta dell'hardware e del software:

- possibilità di utilizzo di hardware già disponibile o in ogni caso di basso costo,

- estrema semplicità di utilizzo da parte degli alunni.

Lo strumento ideale per visualizzare il puzzle proposto, catturare l'immagine del puzzle-programma ed eleborarla mediante algoritmi di image recognition è probabilmente un tablet. Purtroppo questo tipo di attrezzatura spesso non è presente nelle scuole.

I requisiti hardware richiesti per la nostra applicazione (risoluzione della fotocamera e potenza del processore) sono minimi quindi sarebbe sufficiente un tablet entry level, ma visto che ogni gruppo dovrebbe utilizzare un diverso dispositivo (la condivisione di pochi dispositivi renderebbe meno interattiva l'esperienza di programmazione) il costo complessivo supererebbe i limiti che ci siamo imposti.

L'applicazione è stata quindi testata e risulta funzionante su smartphone dotati di sistema operativo Android, al momento il più diffuso.

Gli smartphone non fanno normalmente parte della dotazione tecnologica delle scuole ma ormai quasi ogni famiglia (e spesso anche gli stessi alunni) dispone di dispositivi di questo tipo. Questo potrebbe essere quindi un momento per favorire l'utilizzo e l'integrazione con l'attività didattica dei propri dispositivi elettronici personali seguendo le politiche di BYOD (Bring Your Own Device) che oggi vengono proposte dai ministeri e dai vari enti che si occupano di didattica.

> "La scuola digitale, in collaborazione con le famiglie e gli enti locali, deve aprirsi al cosiddetto BYOD (Bring Your Own Device), ossia a politiche per cui l'utilizzo di dispositivi elettronici personali durante le attività didattiche sia possibile ed efficientemente integrato."
> *MIUR - Piano Nazionale Scuola Digitale* [101]

## 7.4 Object recognition

La scelta di utilizzare blocchi non dotati di elettronica interna rende necessario definire le procedure che permettono di riconoscere i vari tipi di blocchi e la loro collocazione all'interno del puzzle-programma.

Abbiamo testato varie soluzioni tutte basate su image recognition, il riconoscimento di oggetti (nel nostro caso i vari tipi di blocco) in base ad alcune caratteristiche quali forma, colore o la presenza di un marker simbolico. Il riconoscimento avviene su un'immagine del puzzle catturata dalla fotocamera del dispositivo e successivamente elaborata.

Requisiti generali e caratteristiche di un blocco:

- collegabilità con altri blocchi,
- possibilità di rappresentare il flusso di esecuzione del programma,
- identificabilità da parte dell'alunno,
- identificabilità da parte del SW da immagine fotografica,
- facilità di realizzazione e costo contenuto.

Presentiamo qui vari test che abbiamo effettuato su blocchi caratterizzati da differenti layout, e dalla presenza di tag simbolici di vario tipo. I test sono stati effettuati usando differenti ambienti software e varie modalità di analisi dell'immagine.

### 7.4.1 Ricerca dei margini del blocco e individuazione del colore

La struttura più semplice di blocco prevede il solo identificatore per l'alunno formato da una icona o da una icona e testo di commento (figura 7.6).

L'identificazione da parte del SW avviene analizzando l'immagine e riconoscendo i blocchi in base alle loro caratteristiche di layout e al loro colore.

E' stata sviluppata un'applicazione per testare la possibilità di utilizzo di blocchi di questo tipo. L'app opera su smartphone in ambiente Android e sfrutta le funzionalità messe a disposizione dalla libreria OpenCV.

OpenCV (Open Source Computer Vision Library) [2] è una libreria dedicata alla Computer Vision rilasciata sotto licenza BSD sia per uso accademico che commerciale. La libreria è stata progettata per applicazioni in tempo reale e garantisce una

[2] opencv.org

buona efficienza computazionale. Esiste anche un wrapper Java delle librerie OpenCV (OpenCV4Android [3]) che garantisce quasi tutte le sue funzionalità in ambiente Android.

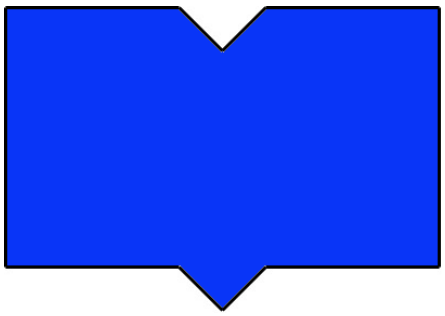


Figure 7.6: Blocco colorato

L'app applica in sequenza varie tecniche di image processing all'immagine originale allo scopo di estrapolare il maggior numero di informazioni. La fase di acquisizione dell'immagine genera un output appartenente allo spazio BGR che viene poi convertita in RGB, HSV e GrayScale per permettere le successive elaborazioni.

La ricerca dei contorni degli oggetti utilizza due tecniche differenti:

- utilizzo dell'algoritmo di Canny sull'immagine a scala di grigi opportunamente filtrata e della trasformata di Hough per estrapolare le linee,
- l'operazione di morfologia matematica di erosione per ottenere i contorni degli oggetti

Dall'immagine ottenuta si estrae un insieme di rettangoli che sono successivamente filtrati per eliminare quelli di dimensione non accettabile. Il risultato finale dell'estrazione dei rettangoli consente di associare ad ogni rettangolo un'area dell'immagine acquisita che contiene un singolo blocco/oggetto.

L'operazione di rilevazione del colore permette poi di riconoscere il tipo, e di conseguenza la semantica, di ogni blocco.

[3] opencv.org/platforms/android

Il riconoscimento del colore prevede il calcolo della luminosità media dell'immagine in spazio colore HSL e il calcolo del valore medio H in spazio HSV in un intorno di pixel localizzati al centro del blocco. Questo valore viene poi modificato in base alla luminosità media dell'immagine e confrontato con una serie di valori soglia, ottenuti sperimentalmente, che rappresentano i vari tipi di blocco (tabella 7.1).

Table 7.1: Valori di soglia per selezione colore

| | |
|---|---|
| H<=18 and H>=176 | rosso |
| H>18 and H<= 35 | giallo |
| H>35 and H<= 85 | verde |
| H>85 and H<= 107 | ciano |
| H>107 and H<=145 | blu |
| H>146 and H<=175 | magenta |

### 7.4.2 Ricerca del contorno del tag e individuazione del colore

Sempre utilizzando il colore dei blocchi per la definizione della loro semantica un ulteriore esperimento è stato fatto su blocchi caratterizzati dalla presenza di un cerchio centrale che contiene una icona (figura 7.7).

L'icona ha lo scopo di rendere evidente all'alunno il significato del blocco e non viene presa in considerazione del software di image recognition che si occupa solo del cerchio che la contiene.

Tramite la trasformata di Hough si identificano questi cerchi che rappresentano il centro di ogni blocco. In un intorno del cerchio viene poi calcolato l'istogramma colore e, in base ai valori di soglia precedentemente illustrati e alle trasformazioni legate alla luminosità dell'immagine, ad ogni blocco viene associata la relativa semantica.

Mettendo in atto i due procedimenti descritti sono state realizzate due applicazioni prototipali pienamente funzionanti in grado di onvertire il puzzle-programma in codice codOWood. Sono però emerse una serie di limitazioni dovute soprattutto all'analisi basata sul colore che è risultata sensibile alla luminosità della scena e dell'immagine catturata e che in ogni caso ha posto vincoli sul numero massimo di tipi di blocco

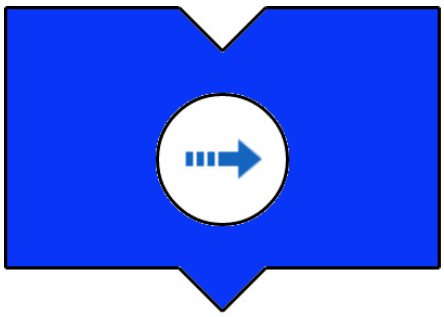


Figure 7.7: Blocco colorato con tag

individuabili. Questi prototipi sono quindi utilizzabili per applicazioni molto semplici che necessitano di poche tipologie di blocchi differenti e in ogni caso è richiesta una buona illuminazione della superficie in cui disporre gli oggetti per formare il puzzle-programma.

Per superare queste limitazioni si è preferito modificare il layout dei blocchi introducendo, oltre a una icona per l'utente, un marker per facilitare il riconoscimento da parte del software.

Sono stati presi in considerazione tre tipi di marker:

- QR Code
- ArUco
- TopCode

In questo caso in ogni blocco oltre all'icona utente è presente un marker che viene utilizzato dalla procedura di riconoscimento.

Prima di attivare la decodifica dei tag, differente per ogni tipo di marker, devono essere effettuate una serie di operazioni preliminari sull'immagine catturata dalla fotocamera. Un primo problema da risolvere è legato all'occupazione di memoria che si riscontra soprattutto in smartphone dotati di fotocamere con risoluzione elevata. In questo caso è preferibile ridurre mediante le impostazioni del dispositivo la risoluzione con cui catturare le immagini da elaborare. In ogni caso viene effettuato

un controllo sulla dimensione e nel caso sia troppo elevata è la stessa applicazione ad effettuare una diminuzione di risoluzione mantenendo inalterato il rapporto larghezza altezza. Oltre a risolvere in questo modo il problema relativo al superamento dei limiti di memoria questa operazione rende più veloce il processo successivo di elaborazione. Dopo il ridimensionamento avviene un controllo mediante lettura dei dati Exif sull'orientamento dell'immagine in quanto differenti dispositivi memorizzano le immagini con rotazioni di base differenti (90, 180 o 270 gradi) e, se necessario, viene effettuata una rotazione per ottenere l'immagine corretta.

Si procede poi all'analisi dell'immagine alla ricerca dei tag che rappresentano i vari blocchi del puzzle-programma.

### 7.4.3 QR Code

In ogni blocco oltre all'icona utente è presente un marker QR Code (figura 7.8).

I codici a barre bidimensionali QR Code (Quick Response Code) [4] sono composti da moduli neri disposti all'interno di uno schema di forma quadrata a sfondo bianco. Il numero di informazioni differenti rappresentabili mediante QR Code varia a seconda della versione utilizzata ma è in ogni caso più che sufficiente per definire le tipologie di blocco necessarie per nostro progetto.

L'uso della tecnologia QR è stato reso pubblico con licenza libera e questo tipo di tag è oggi molto diffuso nel campo dei dispositivi mobili. Varie applicazioni e librerie software per il riconoscimento di questi tag sono spesso già presenti negli smartphone in quanto preinstallati dai relativi produttori.

In ambiente Android è disponibile all'interno delle Mobile Vision API la Barcode Scanner API che permette di riconoscere e decodificare varie tipologie di barcode e fra questi i QR Code.

Dopo le procedure di eventuale ridimensionamento dell'immagine questa viene convertita in Bitmap Binario.

Una prima scansione restituisce una lista di valori risultanti della decodifica dei QRCode rilevati. Per ciascun marker oltre al suo codice si ottengono le coordinate

[4] www.qrcode.com


Figure 7.8: Blocco con QRCode

di 3 angoli, partendo da quello in basso a sinistra e procedendo poi in senso orario. Questi valori sono utilizzati per ritagliare la porzione significativa dell'immagine e per riallinearla.

Individuata la posizione dei QR Code all'interno dell'immagine si deduce il flusso sequenziale dell'algoritmo identificando l'indentazione dei blocchi. Nel caso di indentazione corretta (chiusura di ogni blocco di selezione o iterazione) si procede alla decodifica del QR Code e alla traduzione di ogni blocco in codice XML per codOWood (figura 7.9).

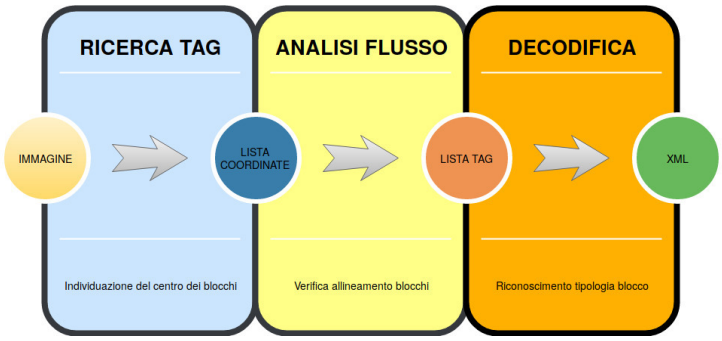



Figure 7.9: Flusso di decodifica

### 7.4.4 ArUco

In ogni blocco oltre all'icona utente è presente un marker ArUco (figura 7.10)

ArUco [102] è una libreria per la realtà aumentata interamente basata su OpenCV. Un Aruco marker è costituito da un quadrato con un ampio bordo nero e da una matrice binaria interna che determina il suo codice identificativo. La matrice interna è formata da 5 word composte da 5 bit che utilizzano come rappresentazione una variante del codice di Hamming. Il numero di informazioni differenti codificabili in un marker è 1024 (anche in questo caso più che sufficienti per rappresentare i vari tipi di blocchi di codOWood). In realtà il modulo Aruco fornisce diversi tipi di dizionari predefiniti che variano per dimensione della matrice interna e il numero dei marker utilizzabili. Per l'app Android è stato scelto il dizionario DICT_5X5_50, che ha una matrice di dimensione 5x5 e un numero di marker utilizzabili pari a 50.

La libreria è scritta in C++ ma ne esiste un porting JavaScript [5].

Per l'applicazione Android è stato utilizzato il modulo OpenCV4Android. Nella versione Android non sono però stati implementati tutti i mpduli di OpenCV e fra questi non risulta implementato il modulo per ArUco che è disponibile nella raccolta "contrib". Per poter utilizzare questi moduli extra, è necessario ricompilare i sorgenti C++ e utilizzare il tool NDK per integrarle con il resto dell'applicazione. Grazie al framework JNI è possibile lo scambio di dati fra codice Java e codice "nativo", nel nostro caso C++.

Dopo le prime fasi di trasformazione dell'immagine avviene la scansione che rileva gli identificatori e le coordinate dei 4 angoli di ciascun marker. Con queste informazioni viene identificato l'allineamento sequenziale del puzzle-programma e si procede alla decodifica e generazione del codice XML.

### 7.4.5 TopCode (Tangible Object Placement Codes)

In ogni blocco oltre all'icona utente è presente un marker TopCode (figura 7.11).

La libreria di computer vision TopCode [6] permette di identificare marcatori cir-

[5] github.com/jcmellado/js-aruco

[6] users.eecs.northwestern.edu/ mhorn/topcodes


Figure 7.10: Blocco con Aruco

colari bidimensionali in bianco e nero e di restituire per ognuno di questi il codice associato, la posizione, l'orientazione e il diametro.

La libreria, originariamente scritta in Java è ora disponibile anche in porting per il linguaggio C++ e JavaScript [7], permette di identificare 99 marcatori differenti. Associando a ogni tag TopCode un diverso tipo di blocco si ottiene un insieme di istruzioni più che sufficiente per risolvere ognuno dei semplici problemi che intendiamo proporre nell'ambiente di codOWood.

La libreria open source TopCode è stata sviluppata allo Human Computer Interaction Lab della Tufts University in Massachusetts.

L'immagine da analizzare deve essere inizialmente convertita in formato bitmap poi si procede a una prima scansione dell'immagine dall'alto verso il basso che restituisce una lista di TopCodes in ordine di rilevamento.

Ciascuno dei TopCodes presenti in lista fornisce diverse informazioni:

- il codice del marker
- le coordinate in pixel del marker all'interno dell'immagine
- il diametro, anche questo espresso in pixel
- l'orientamento angolare in radianti

[7] github.com/TIDAL-Lab/TopCodes

Se la lista risultante è vuota la procedura abortisce e l'immagine non viene riconosciuta come puzzle-programma, in caso contrario vengono eseguite le seguenti operazioni:

1. calcolo della media degli orientamenti dei TopCodes rilevati
2. rotazione dell'immagine in base al valore precedentemente ottenuto
3. seconda scansione dell'immagine con l'orientamento corretto
4. individuazione della coordinata x del TopCode che si trova più a destra e di quella che si trova più a sinistra e la coordinata y del TopCode che si trova più in alto e di quella che si trova più in basso
5. ritaglio dell'immagine in base ai valori massimi e minimi delle coordinate x e y dei TopCodes e in base alla media dei diametri
6. salvataggio della nuova immagine ottenuta sovrascrivendo quella originale
7. decodifica dei TopCodes e generazione del file XML


Figure 7.11: Blocco con TopCode

### 7.4.6 Confronto fra tipologie di marker

Sia con l'app Android che con la web application sono stati effettuati vari test di riconoscimento dei tag. I parametri per la valutazione sono stati la percentuale di

blocchi riconosciuti correttamente e il tempo necessario per la decodifica del puzzle in un programma codOWood.

Le prove sono state effettuate utilizzando 2 smartphone: uno di fascia bassa con processore poco performante e fotocamera con risoluzione modesta, e uno di fascia alta che ha prestazioni e qualità fotografica da top di gamma.

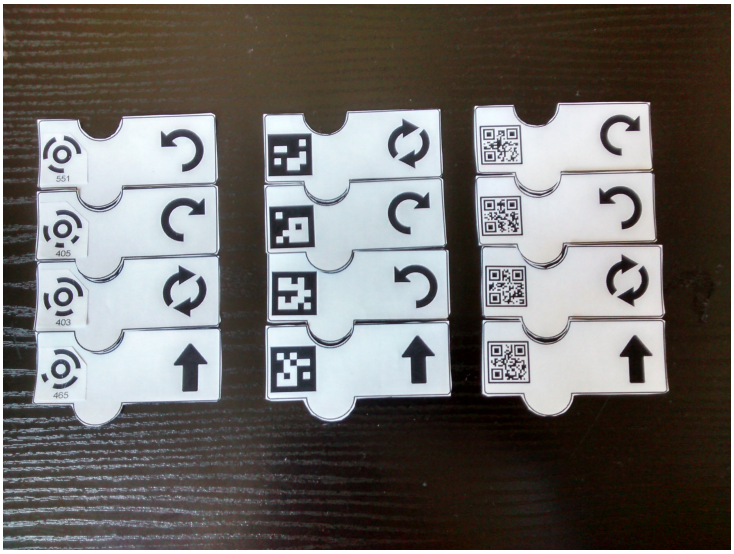


Figure 7.12: Primi test effettuati

Una prima analisi è stata effettuata su puzzle formati da un numero di blocchi ridotto (figura 7.12) che ha dato i seguenti risultati: per entrambi gli smartphone, si ha una precisione del 100% (tutti i codici presenti sono individuati) nella rilevazione dei TopCodes e degli ArUco Markers, mentre i QRCodes si sono dimostrati poco affidabili e spesso solo alcuni blocchi sono stati riconosciuti e decodificati correttamente.

Per quanto riguarda le prestazioni la decodifica degli ArUko Markers è stata pressochè immediata mentre per i TopCodes e QRCodes il tempo medio di elaborazione si aggira intorno ai pochi secondi.

In base ai primi risultati si è deciso di privilegiare l'affidabilità e di proseguire la sperimentazione sui TopCodes e ArUko Markers.

## 7.5 Tipi di blocchi - Il linguaggio codOWood

Per le prime attività di coding, seguendo un modello presente nella maggior parte dei corsi introduttivi, abbiamo pensato a una serie di problemi a difficoltà crescente. Nel prototipo di codOWood abbiamo seguito la sequenza di problemi presente nel modulo *maze* di Blockly Games [8]. Si tratta di una serie di situazioni in cui un personaggio deve eseguire le istruzioni che gli vengono fornite per raggiungere l'uscita da un labirinto.

Per risolvere i primi livelli sono sufficienti i comandi di avanzamento e di svolta che in codOWood corrispondono ai blocchi presenti in figura 7.13. Nel risolvere questo primi livelli gli alunni apprendono quindi il concetto di sequenza proponendo programmi composti da una sequenza di blocchi operativi.

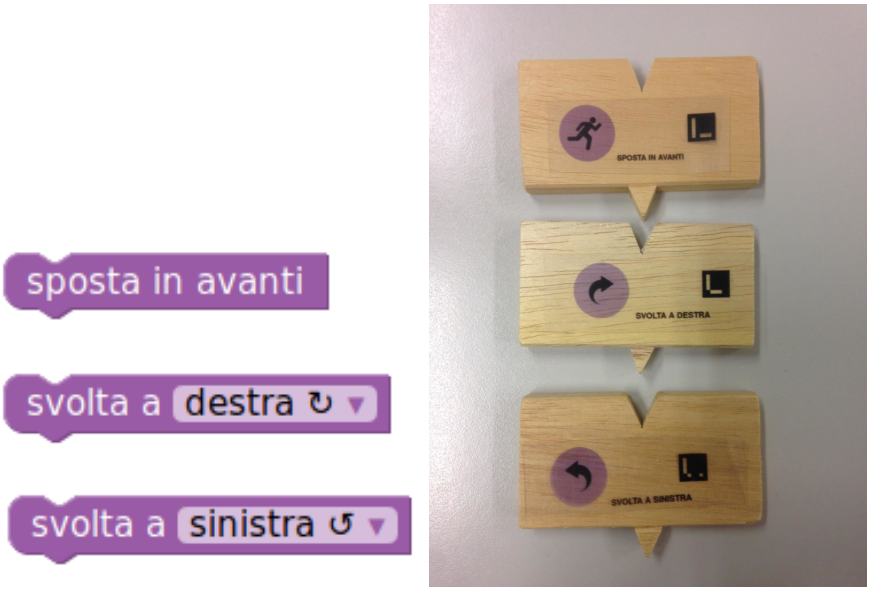



Figure 7.13: Blocchi operativi di Blockly e di codOWood

Il passaggio successivo è l'introduzione del concetto di ciclo tramite un blocco che definisce la ripetizione di una serie di comandi. Blockly Games propone un unico blocco di questo tipo (fig. 7.14) per ripetere una serie di azioni fino al raggiungimento

[8] https://blockly-games.appspot.com/maze

dell'obiettivo finale. I blocchi virtuali hanno la possibilità di variare il loro aspetto e, in questo caso, il blocco si espande per poter contenere un numero variabile di altri blocchi (i blocchi interni al ciclo). Questo aspetto "mutante" non può però essere

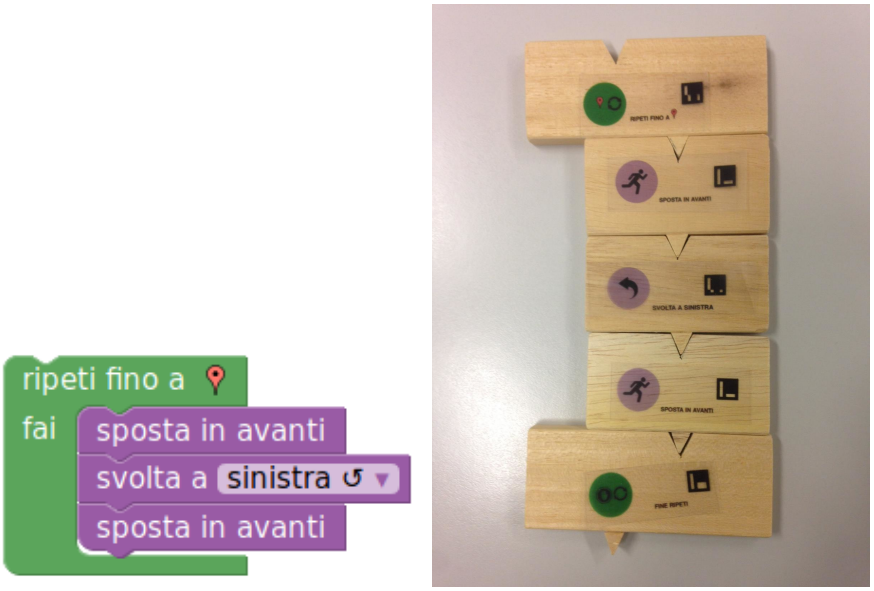



Figure 7.14: Ripetizione di azioni in Blockly e in codOWood

mantenuto nel caso di blocchi fisici. Il problema è stato quindi risolto introducendo un blocco di chiusura. Per rendere evidente la struttura complessiva del ciclo, oltre a colorare diversamente le icone dei vari tipi di blocchi, varia anche il punto di connessione tra un blocco e l'altro in modo da realizzare puzzle che ripropongono l'idea di indentazione comune a tutti i linguaggi di programmazione testuali. (fig. 7.14)

Un problema analogo relativo alla "rigidità" di struttura dei blocchi fisici lo abbiamo riscontrato in relazione alle strutture di selezione del flusso di esecuzione che sono necessarie per risolvere gli ultimi livelli e che introducono il concetto di selezione.

In figura 7.15 un esempio di algoritmo con blocchi di selezione in Blockly e in figura 7.16 l'equivalente in codOWood.

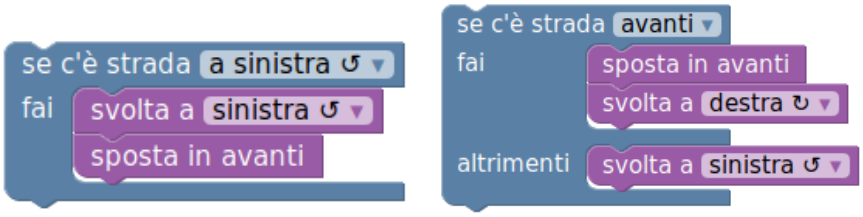



Figure 7.15: blocchi virtuali per la selezione

Figure 7.16: blocchi fisici per la selezione

Possiamo quindi riconoscere in codOWood tre tipologie di blocchi:

- blocchi operativi
- blocchi di ripetizione
- blocchi di selezione

## 7.6 Sviluppi futuri

Seppur a livello prototipale codOWood è pienamente funzionante ed è in atto una sperimentazione con tre classi di scuola elementare per confrontare il comportamento di gruppi di alunni che utilizzano blocchi virtuali con altri che utilizzano i blocchi tangibili per risolvere lo stesso problema.

Dal lato tecnologico del riconoscimento ottico dei blocchi stiamo lavorando nella direzione di blocchi senza marker di nessun tipo in cui il riconoscimento avviene tramite l'individuazione delle forme e dei colori delle icone. Questo permetterebbe di ottenere oggetti maggiormente user friendly ma è da verificare la qualità di riconoscimento mantenendo inalterati vincoli che ci siamo imposto relativi alla qualità dell'hardware e degli aspetti logisitici (luminosità non ottimale, inquadratura non perfettamente ortogonale e allineata al puzzle).

# Bibliography


[1] Jeannette M Wing. Computational thinking. *Communications of the ACM*, 49(3):33–35, 2006.

[2] National Research Council et al. *Report of a workshop on the scope and nature of computational thinking*. National Academies Press, 2010.

[3] National Research Council et al. *Report of a workshop on the pedagogical aspects of computational thinking*. National Academies Press, 2011.

[4] Seymour Papert. *Mindstorms: Children, computers, and powerful ideas*. Basic Books, Inc., 1980.

[5] Peter J. Denning. Remaining trouble spots with computational thinking. *Commun. ACM*, 60(6):33–39, May 2017. URL: `http://doi.acm.org/10.1145/2998438`, `doi:10.1145/2998438`.

[6] Donald E Knuth. Computer science and its relation to mathematics. *The American Mathematical Monthly*, 81(4):323–343, 1974.

[7] Peter J Denning. The profession of it beyond computational thinking. *Communications of the ACM*, 52(6):28–30, 2009.

[8] Peter J. Denning, Douglas E Comer, David Gries, Michael C. Mulder, Allen Tucker, A. Joe Turner, and Paul R Young. Computing as a discipline. *Computer*, 22(2):63–70, 1989.

[9] Strawman Draft. Computer science curricula 2013, 2013.

[10] Cédric Villani. *Birth of a Theorem: a mathematical adventure*. Farrar, Straus and Giroux, 2015.

[11] Google and Gallup. Images of computer science: perceptions among students, parents and educators in the us. 2015.

[12] Walter Gander, Antoine Petit, Gérard Berry, Barbara Demo, Jan Vahrenhold, Andrew McGettrick, Roger Boyle, Avi Mendelson, Chris Stephenson, Carlo Ghezzi, et al. Informatics education: Europe cannot afford to miss the boat. *ACM,[online] Available at: http://europe. acm. org/iereport/ie. html*, 2013.

[13] Steve Furber et al. Shut down or restart? the way forward for computing in uk schools. *The Royal Society, London*, 2012.

[14] Académie des sciences. L'enseignement de l'informatique en france - il est urgent de ne plus attendre. pages 1–34, 2013. URL: `http://www.academie-sciences.fr/pdf/rapport/rads_0513.pdf`.

[15] Megan Smith. Computer science for all. *The White House*, 2016.

[16] AICA. Informatica nei licei nel contesto della riforma della scuola. *Documenti di Mondo Digitale*, 2003.

[17] Enrico Nardelli and Giorgio Ventre. Introducing computational thinking in italian schools: A first report on "programma il futuro" project. In *INTED2015 Proceedings (9th International Technology, Education and Development Conference). IATED*, pages 7414–7421. Citeseer, 2015.

[18] Isabella Corradini, Michael Lodi, and Enrico Nardelli. Computational thinking in italian schools: Quantitative data and teachers' sentiment analysis after two years of. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*, pages 224–229. ACM, 2017.

[19] Alberto Barbero and G Barbara Demo. The art of programming in a technical institute after the italian secondary school reform. *Proceedings ISSEP 2011*, 2011.

[20] Matthew Hertz. What do cs1 and cs2 mean?: investigating differences in the early courses. In *Proceedings of the 41st ACM technical symposium on Computer science education*, pages 199–203. ACM, 2010.

[21] Joseph Bergin. Fourteen pedagogical patterns. 2000.

[22] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)*, 10(4):16, 2010.

[23] Carol Smith and Jon Rickman. Selecting languages for pedagogical tools in the computer science curriculum. In *ACM SIGCSE Bulletin*, volume 8, pages 39–47. ACM, 1976.

[24] RL Wexelblat. First programming language: Consequences (panel discussion). In *Proceedings of the 1979 annual conference*, page 259. ACM, 1979.

[25] Alan L Tharp. Selecting the "right" programming language. In *ACM SIGCSE Bulletin*, volume 14, pages 151–155. ACM, 1982.

[26] Naomi S Baron. The future of computer languages: implications for education. *ACM SIGCSE Bulletin*, 18:44–49, 1986.

[27] Paul A Luker. Never mind the language, what about the paradigm? In *ACM SIGCSE Bulletin*, volume 21, pages 252–256. ACM, 1989.

[28] L. Goosen. A brief history of choosing first programming languages. In *IFIP International Federation for Information Processing*, volume 269, pages 167–170, 2008. doi:10.1007/978-0-387-09657-5_11.

[29] Richard Close, Danny Kopec, and Jim Aman. Cs1: perspectives on programming languages and the breadth-first approach. In *Journal of Computing*

*Sciences in Colleges*, volume 15, pages 228–234. Consortium for Computing Sciences in Colleges, 2000.

[30] KN King. The evolution of the programming languages course. In *ACM SIGCSE Bulletin*, volume 24, pages 213–219. ACM, 1992.

[31] David J Barnes, Michael Kölling, and James Gosling. *Objects first with Java: A practical introduction using Bluej*. Pearson, 2017.

[32] Carl Alphonce and Phil Ventura. Object orientation in cs1-cs2 by design. *ACM SIGCSE Bulletin*, 34(3):70–74, 2002.

[33] Michael Kölling. The problem of teaching object-oriented programming, part 1: Languages. *Journal of Object-oriented programming*, 11(8):8–15, 1999.

[34] John Lewis. Myths about object-orientation and its pedagogy. In *ACM SIGCSE Bulletin*, volume 32, pages 245–249. ACM, 2000.

[35] Tamar Vilner, Ela Zur, and Judith Gal-Ezer. Fundamental concepts of cs1: procedural vs. object oriented paradigm-a case study. In *ACM SIGCSE Bulletin*, volume 39, pages 171–175. ACM, 2007.

[36] Jens Bennedsen and Michael E Caspersen. Teaching object-oriented programming-towards teaching a systematic programming process. In *Eighth Workshop on Pedagogies and Tools for the Teaching and Learning of Object Oriented Concepts. Affiliated with 18th European Conference on Object-Oriented Programming (ECOOP 2004)*, 2004.

[37] Kim B Bruce and Andrea Danyluk. Event-driven programming facilitates learning standard programming concepts. In *Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, pages 96–100. ACM, 2004.

[38] Stephen Cooper, Wanda Dann, and Randy Pausch. Teaching objects-first in introductory computer science. In *ACM SIGCSE Bulletin*, volume 35, pages 191–195. ACM, 2003.

[39] TM Rao, Sandeep Mitra, Roxanne Canosa, Sidney Marshall, and Thomas Bullinger. Problem stereotypes and solution frameworks: a design-first approach for the introductory computer science sequence. *Journal of Computing Sciences in Colleges*, 22(6):56–64, 2007.

[40] Eric Crahen, Carl Alphonce, and Phil Ventura. Quickuml: a beginner's uml tool. In *Companion of the 17th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 62–63. ACM, 2002.

[41] Albrecht Ehlert and Carsten Schulte. Empirical comparison of objects-first and objects-later. In *Proceedings of the fifth international workshop on Computing education research workshop*, pages 15–26. ACM, 2009.

[42] Jens Bennedsen and Carsten Schulte. What does objects-first mean?: An international study of teachers' perceptions of objects-first. In *Proceedings of the Seventh Baltic Sea Conference on Computing Education Research-Volume 88*, pages 21–29. Australian Computer Society, Inc., 2007.

[43] Ira Greenberg, Deepak Kumar, and Dianna Xu. Creative coding and visual portfolios for cs1. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pages 247–252. ACM, 2012.

[44] Caitlin Kelleher and Randy Pausch. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys (CSUR)*, 37(2):83–137, 2005.

[45] Alexis Vinícius de Aquino Leal and Deller James Ferreira. Learning programming patterns using games. *International Journal of Information and Communication Technology Education (IJICTE)*, 12(2):23–34, 2016.

[46] Hope Caton and Darrel Greenhill. Rewards and penalties: A gamification approach for increasing attendance and engagement in an undergraduate computing module. In *Gamification: Concepts, Methodologies, Tools, and Applications*, pages 1003–1014. IGI Global, 2015.

[47] Sebastian Deterding, Dan Dixon, Rilla Khaled, and Lennart Nacke. From game design elements to gamefulness: defining gamification. In *Proceedings of the 15th international academic MindTrek conference: Envisioning future media environments*, pages 9–15. ACM, 2011.

[48] Kate Howland and Judith Good. Learning to communicate computationally with flip: A bi-modal programming language for game creation. *Computers & Education*, 80:224–240, 2015.

[49] Lakshmi Prayaga, James W Coffey, and Karen Rasmussen. Strategies to teach game development across age groups. In *Design, Utilization, and Analysis of Simulations and Game-Based Educational Worlds*, pages 95–110. IGI Global, 2013.

[50] Mark Overmars. Teaching computer science through game design. *Computer*, 37(4):81–83, 2004.

[51] Stephen Cooper, Wanda Dann, and Randy Pausch. Alice: a 3-d tool for introductory programming concepts. In *Journal of Computing Sciences in Colleges*, volume 15, pages 107–116. Consortium for Computing Sciences in Colleges, 2000.

[52] Alexander Repenning. Agentsheets®: An interactive simulation environment with end-user programmable agents. *Interaction*, 2000.

[53] Albert Sweigart. *Making Games with Python & Pygame*. CreateSpace North Charleston, 2012.

[54] Ismar Frango Silveira, Pollyana Notargiacomo Mustaro, and Luciano Silva. Using computer games to teach design patterns and computer graphics in cs and it undergraduate courses: Some case studies. In *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*, volume 1, pages 490–499, 2007.

[55] Scott Leutenegger and Jeffrey Edgington. A games first approach to teaching introductory programming. *ACM SIGCSE Bulletin*, 39(1):115–118, 2007.

[56] Tzvetomir I Vassilev and Borislav I Mutev. An approach to teaching introductory programming using games. In *Proceedings of theInternational Conference on e-Learning*, volume 14, page 246, 2014.

[57] Thomas W Price and Tiffany Barnes. Comparing textual and block interfaces in a novice programming environment. In *Proceedings of the eleventh annual International Conference on International Computing Education Research*, pages 91–99. ACM, 2015.

[58] A Ferrari, A Poggi, and M Tomaiuolo. Object oriented puzzle programming. *Didattica Informatica-Didamatica 2016*, pages 1–10, 2016.

[59] Joel Adams. *Alice 3 in Action: Computing Through Animation*. Cengage Learning, 2014.

[60] Tebring Daly. *Influence of Alice 3: Reducing the hurdles to success in a CS1 programming course*. University of North Texas, 2013.

[61] Marcel Rebouças, Gustavo Pinto, Felipe Ebert, Weslley Torres, Alexander Serebrenik, and Fernando Castor. An empirical study on the usage of the swift programming language. In *Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on*, volume 1, pages 634–638. IEEE, 2016.

[62] Lindsey Ann Gouws, Karen Bradshaw, and Peter Wentworth. Computational thinking in educational activities: an evaluation of the educational game lightbot. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*, pages 10–15. ACM, 2013.

[63] David Wolber. App inventor and real-world motivation. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, pages 601–606. ACM, 2011.

[64] Cameron Wilson. Hour of code: we can solve the diversity problem in computer science. *ACM Inroads*, 5(4):22–22, 2014.

[65] Gareth Halfacree and Eben Upton. *Raspberry Pi user guide*. John Wiley & Sons, 2012.

[66] Bob Violino. Time to reboot. *Communication of the ACM*, 52(4):19–19, April 2009. URL: http://doi.acm.org/10.1145/1498765.1498774, doi:10.1145/1498765.1498774.

[67] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, et al. Scratch: programming for all. *Communications of the ACM*, 52(11):60–67, 2009.

[68] David Weintrop and Uri Wilensky. To block or not to block, that is the question: students' perceptions of blocks-based programming. In *Proceedings of the 14th International Conference on Interaction Design and Children*, pages 199–208. ACM, 2015.

[69] Neil CC Brown, Michael Kolling, and Amjad Altadmri. Position paper: Lack of keyboard support cripples block-based programming. In *Blocks and Beyond Workshop (Blocks and Beyond), 2015 IEEE*, pages 59–61. IEEE, 2015.

[70] Thomas W Price, Neil CC Brown, Dragan Lipovac, Tiffany Barnes, and Michael Kölling. Evaluation of a frame-based programming editor. In *ICER*, pages 33–42, 2016.

[71] Michael Dawson. *Python programming for the absolute beginner*. Cengage Learning, 2010.

[72] Vern Ceder and Nathan Yergler. Teaching programming with python and pygame. *Apresentado na PyCon*, 2003.

[73] Andries van Dam. Post-wimp user interfaces. *Commun. ACM*, 40(2):63–67, February 1997. URL: http://doi.acm.org/10.1145/253671.253708, doi:10.1145/253671.253708.

[74] Thomas T Hewett, Ronald Baecker, Stuart Card, Tom Carey, Jean Gasen, Marilyn Mantei, Gary Perlman, Gary Strong, and William Verplank. *ACM SIGCHI curricula for human-computer interaction*. ACM, 1992.

[75] Jakob Nielsen. Usability inspection methods. In *Conference companion on Human factors in computing systems*, pages 413–414. ACM, 1994.

[76] Hideyuki Suzuki and Hiroshi Kato. Algoblock: a tangible programming language, a tool for collaborative learning. In *Proceedings of 4th European Logo Conference*, pages 297–303, 1993.

[77] Hideyuki Suzuki and Hiroshi Kato. Interaction-level support for collaborative learning: Algoblock—an open programming language. In *The first international conference on Computer support for collaborative learning*, pages 349–355. L. Erlbaum Associates Inc., 1995.

[78] Mitchel Resnick, Fred Martin, Robert Berg, Rick Borovoy, Vanessa Colella, Kwin Kramer, and Brian Silverman. Digital manipulatives: new toys to think with. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 281–287. ACM Press/Addison-Wesley Publishing Co., 1998.

[79] Cyndi Rader, Cathy Brand, and Clayton Lewis. Degrees of comprehension: children's understanding of a visual programming environment. In *Proceedings of the ACM SIGCHI Conference on Human factors in computing systems*, pages 351–358. ACM, 1997.

[80] Timothy S McNerney. From turtles to tangible programming bricks: explorations in physical language design. *Personal and Ubiquitous Computing*, 8(5):326–337, 2004.

[81] Danli Wang, Cheng Zhang, and Hongan Wang. T-maze: a tangible programming tool for children. In *Proceedings of the 10th International Conference on Interaction Design and Children*, pages 127–135. ACM, 2011.

[82] Danli Wang, Yang Zhang, and Shengyong Chen. E-block: A tangible programming tool with graphical blocks. *Mathematical Problems in Engineering*, 2013, 2013.

[83] Felix Hu, Ariel Zekelman, Michael Horn, and Frances Judd. Strawbies: explorations in tangible programming. In *Proceedings of the 14th International Conference on Interaction Design and Children*, pages 410–413. ACM, 2015.

[84] Michael S Horn and Robert JK Jacob. Tangible programming in the classroom with tern. In *CHI'07 extended abstracts on Human factors in computing systems*, pages 1965–1970. ACM, 2007.

[85] Crouser RJ Horn, M and M Bers. Tangible interaction and learning: The case for a hybrid approach, 2015.

[86] Michael S Horn, Erin Treacy Solovey, and Robert JK Jacob. Tangible programming and informal science learning: making tuis work for museums. In *Proceedings of the 7th international conference on Interaction design and children*, pages 194–201. ACM, 2008.

[87] Peta Wyeth and Helen C Purchase. Tangible programming elements for young children. In *CHI'02 extended abstracts on Human factors in computing systems*, pages 774–775. ACM, 2002.

[88] Timothy S McNerney. *Tangible programming bricks: An approach to making programming accessible to everyone*. PhD thesis, Massachusetts Institute of Technology, 1999.

[89] Michael S Horn and Robert JK Jacob. Designing tangible programming languages for classroom use. In *Proceedings of the 1st international conference on Tangible and embedded interaction*, pages 159–162. ACM, 2007.

[90] Daniel Gallardo, Carles F Julia, and Sergi Jorda. Turtan: A tangible programming language for creative exploration. In *Horizontal Interactive Human*

*Computer Systems, 2008. TABLETOP 2008. 3rd IEEE International Workshop on*, pages 89–92. IEEE, 2008.

[91] Dixon Lo and Austin Lee. Click: Using smart devices for physical collaborative coding education. In *Proceedings of the TEI'16: Tenth International Conference on Tangible, Embedded, and Embodied Interaction*, pages 500–505. ACM, 2016.

[92] Amanda Strawhacker and Marina Bers. " i want my robot to look for food": Comparing kindergartner's programming comprehension using tangible, graphic, and hybrid user interfaces. *International Journal of Technology and Design Education*, 25(3):293, 2015.

[93] Theodosios Sapounidis and Stavros Demetriadis. Tangible versus graphical user interfaces for robot programming: exploring cross-age children's preferences. *Personal and ubiquitous computing*, 17(8):1775–1786, 2013.

[94] Michael S Horn, Erin Treacy Solovey, R Jordan Crouser, and Robert JK Jacob. Comparing the use of tangible and graphical programming languages for informal science education. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 975–984. ACM, 2009.

[95] Oren Zuckerman and Ayelet Gal-Oz. To tui or not to tui: Evaluating performance and preference in tangible vs. graphical user interfaces. *International Journal of Human-Computer Studies*, 71(7):803–820, 2013.

[96] Yuichi Itoh, Shintaro Akinobu, Hiroyasu Ichida, Ryoichi Watanabe, Yoshifumi Kitamura, and Fumio Kishino. Tsu. mi. ki: Stimulating children's creativity and imagination with interactive blocks. In *Creating, Connecting and Collaborating through Computing, 2004. Proceedings. Second International Conference on*, pages 62–70. IEEE, 2004.

[97] Seymour Papert. Teaching children thinking. *Programmed Learning and Educational Technology*, 9(5):245–255, 1972.

[98] Seymour Papert. Learn think to children. *L. A. relation*, 1972.

[99] Mitchel Resnick, Fred Martin, Randy Sargent, and Brian Silverman. Programmable bricks: Toys to think with. *IBM Systems journal*, 35(3.4):443–452, 1996.

[100] Seymour Papert and Cynthia Solomon. Twenty things to do with a computer. 1971.

[101] MIUR. Piano nazionale scuola digitale. *Ministero dell'Istruzione dell'Università e della Ricerca. Direttiva del 27 ottobre 2015, n. 851*, 2015. URL: http://www.istruzione.it/scuola_digitale/allegati/Materiali/pnsd-layout-30.10-WEB.pdf.

[102] S. Garrido-Jurado, R. Mu noz Salinas, F.J. Madrid-Cuevas, and M.J. Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280 – 2292, 2014. URL: http://www.sciencedirect.com/science/article/pii/S0031320314000235, doi:http://dx.doi.org/10.1016/j.patcog.2014.01.005.

# Acknowledgments

My gratitude goes to everybody who has contributed to this work in different ways.

A special thanks goes to:

- My Ph.D. advisor Michele Tomaiuolo who has made many direct contributions to this work and for the interesting discussions that we have during this time.
- Agostino Poggi, Monica Mordonini, Andrea Prati and Giulio Angiani with whom I have worked in these years.