Corso Formazione
Coding e Pensiero Computazionale
CRIAD @ Casa Bufalini

[modulo-02]

ELEMENTI DI PROGETTAZIONE DI MICROMONDI /1

OBIETTIVI DEL MODULO

- Introdurre un primo insieme di elementi utili per la progettazione e programmazione (coding) di micromondi, considerando esempi concreti in discipline specifiche e materie curricolari
- In particolare in questo primo insieme si considereranno concetti, meccanismi e tecniche che fanno riferimento alla parte più algoritmica dei micromondi.

CONCETTI GENERALI DI UN MICROMONDO

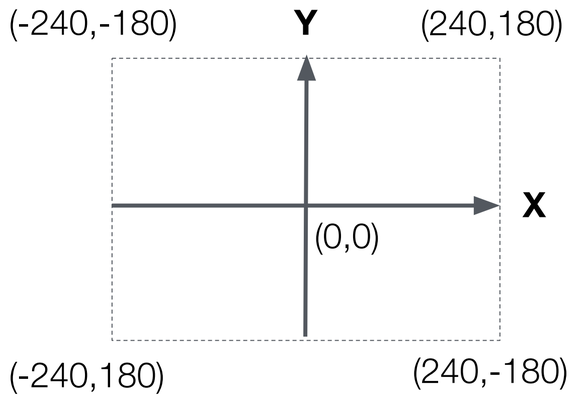
- Come pensare a un micromondo in senso astratto
  - ogni micromondo è inquadrabile nel modo più generale come un ***sistema*** in cui ci sono uno più ***attori*** in azione in un certo ***ambiente***
    - gli **attori** rappresentano i componenti autonomi del sistema che *agiscono* e *interagiscono* attivamente, svolgono un certo ruolo
    - l'**ambiente** rappresenta il contesto condiviso dagli attori
  - In Snap!
    - gli attori si chiamano "sprite"
    - l'ambiente si chiama "stage"

ATTORI

- Ogni attore ha una propria identità univoca e due aspetti principali
  - un **comportamento**
    - Il comportamento di un attore è definito da un ***copione***, che definisce in modo dettagliato cosa deve fare, quali istruzioni deve eseguire
    - in gergo informatico/coding, corrisponde al *programma* dell'attore
  - una **rappresentazione**
    - in Snap! ogni attore ha associata anche una rappresentazione visiva (un *costume*), che può cambiare durante l'azione
      - aspetto approfondito nel prossimo modulo

AMBIENTE

- Concettualmente rappresenta il *contesto* condiviso fra gli attori
  - la condivisione porta a forme di *interazione*: l'azione di un attore può avere effetti percepiti anche da un altro attore
  - come gli attori, anche l'ambiente può avere un proprio comportamento e rappresentazione
- In Snap!:
  - l'ambiente definisce un modello esplicito di ***spazio***
    - stage come piano cartesiano, dimensionabile a piacere
      - dimensioni di defaut: 480 x 360
    - attori con proprietà predefinte in relazione al piano cartesiano
      - **posizione**, **direzione**
      - blocchi con azioni al movimento, orientamento, …



SULL'USO DI METAFORE ANTROPOMORFICHE

- Strategia adottata anche in LOGO
  - usare metafore antropomorfiche come *livello di astrazione* utile per facilitare la progettazione e la programmazione
  - strategie
    - "personificazione"
    - "teatralizzazione"

IL COPIONE (SCRIPT)

- Modello generale di un copione
  - ogni copione è organizzato come una o più **sequenze** di istruzioni, da eseguire a fronte di un determinato **evento**
  - In Snap!
    - si usano **blocchi** per rappresentare sia le specifiche istruzioni, sia i tipi di eventi
      - la sequenza è data dalla catena di blocchi, agganciati l'un l'altro
        - in Snap! si usa il termine script per indicare la singola sequenza di blocchi
      - [Esempio Snap!] micromondo disegno geometrico - quadrato
    - l'insieme dei possibili blocchi è organizzato in categorie predefinite

IL CONCETTO DI "TEMPO DI ESECUZIONE" (RUN TIME)

- Un copione (programma) viene:
  - scritto dal *regista*/programmatore
  - eseguito dall'attore
- In gergo informatico, la fase in cui il sistema è in esecuzione e gli attori eseguono i propri copioni viene definito ***tempo di esecuzione***
  - "a tempo di esecuzione, l'attore esegue il copione", "il programma va in esecuzione”

ESECUZIONE PASSO PASSO

- Una funzionalità molto importante è la possibilità di eseguire passo passo il programma (*tracing* in gergo tecnico)
  - utile per trovare e rimuovere errori (fare *debugging*)
  - più in generale utile per capire e riflettere sulla struttura e funzionamento del programma, della strategia adottata
- In Snap! è una modalità di esecuzione attivabile mediante pulsante sulla barra in alto  e controllabile con tasti in alto a destra 

SEQUENZA E DECOMPOSIZIONE

- La sequenza costituisce una prima forma basilare di *decomposizione*
  - ovvero decomponiamo un comportamento articolato in una sequenza di comportamenti più semplici (istruzioni), in cui vale la regola per cui ogni istruzione può essere *eseguita* solo *dopo* l'esecuzione dell'istruzione precedente (se c'è, se non è la prima)

- Forma di decomposizione ***temporale***
  - poiché esprime una struttura in relazione al tempo
    - "prima fai questo, poi quello,..."

- In merito all'apprendimento - metacompetenza
  - capacità di saper rappresentare/organizzare strategie/comportamenti articolati, decomponendoli in passi più semplici
  - organizzazione temporale

ISTRUZIONI ED EVENTI

- **Istruzioni**
  - In generale ogni istruzione rappresenta **un'azione** che deve essere eseguita dall'attore
    - può essere un'azione semplice (o "atomica"), oppure una “macro-azione” o *procedura* a sua volta costituita da una sequenza di istruzioni
  - In Snap!
    - insieme di base predefinito di blocchi + possibilità di creare nuovi blocchi (come macro-azioni/procedure)
- **Eventi**
  - definiscono le condizioni verificate le quali deve essere eseguita la sequenza di istruzioni
  - In Snap! blocchi “quando” con forma specifica (non agganciabili)
    - trattati nel dettaglio nel prossimo modulo

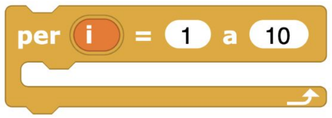
ISTRUZIONI DI CONTROLLO

- La categoria ***controllo*** è una categoria fondamentale, con istruzioni di base che in generale permettono di controllare o definire il flusso di esecuzione
  - definite anche “costrutti di controllo"
- Istruzioni (o costrutti) di controllo fondamentali
  - **iterazione**
    - permettono di eseguire ciclicamente un insieme di istruzioni
  - **selezione**
    - permettono di selezionare istruzioni da eseguire a fronte della valutazione di condizioni a tempo di esecuzione

COSTRUTTI DI ITERAZIONE

- Permette di ripetere un certo numero di volte una certa sequenza di istruzioni
  - In Snap! blocco “ripeti <N> volte”
    - [Esempio Snap!] quadrato con ripeti

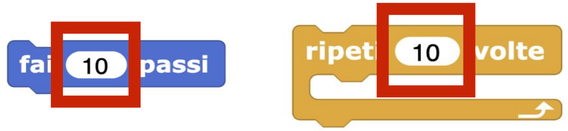
- Molteplici varianti
  - permettono di specificare criteri diversi per definire quante volte la sequenza deve essere o quali sono le condizioni per fermare la ripetizione



COSTRUTTI DI ITERAZIONE ANNIDATI

- La semplice combinazione dei costrutti di controllo di iterazione permette di costruire, realizzare comportamenti molto articolati e complicati
- Esempio:
  - Un ripeti dentro ad un altro ripeti
  - [Esempio Snap!] “Fiore”
- In gergo informatico: *ripeti annidati*

ISTRUZIONI (BLOCCHI) E PARAMETRI

- Il modello generale di un'istruzione (blocchi) prevede che ogni istruzione sia univocamente identificabile da un nome (o una descrizione) e abbia zero o più *parametri*
  - In Snap! 

- Ogni **parametro** ci permette di specificare un *valore*, utile o necessario per l'esecuzione dell'istruzione/blocco
  - Il **valore** può essere di *tipo* diverso: un numero, una lettera, una sequenza di lettere (ovvero una stringa).. ma anche un'immagine, un attore stesso...
  - Il **tipo** definisce proprietà e caratteristiche che accomunano tutti i valori di quel tipo (come insieme)

PARAMETRIZZAZIONE COME FORMA DI GENERALIZZAZIONE

- A livello concettuale la nozione di parametro ci permette di attuare un processo di *generalizzazione*
- Esempio blocco "fai <N> passi"
  - invece di avere una istruzione o blocco specifico per ogni valore di passo ("fai 1 passo", "fai 2 passi", "fai 3 passi"...), ne abbiamo solo uno “parametrizzato”, che può essere usato in generale

ATTORI E MEMORIA

- Per poter eseguire il copione, gli attori hanno a disposizione una *memoria* utile per *rappresentare* e tener traccia ("ricordare") una qualsiasi informazioni (come dato, conoscenza) che può essere utile o necessaria per eseguire il copione, per fare quello che deve fare
  - ogni attore ha sua memoria
  - in più esiste una "memoria condivisa"
- Il concetto fondamentale con cui possiamo sfruttare la memoria è il concetto di **variabile**

IL CONCETTO DI VARIABILE

- Nella forma più generale e astratta, le variabili ci permettono di *dare un nome alle cose*, ovvero di definire un'associazione fra un *rappresentazione simbolica* (es: un nome) e un certo *valore*
  - Esempi
    - i dati di un problema di matematica
      - "numero mele" ⟶ 5, "larghezza aquilone" ⟶ 20, "angolo lancio" ⟶ 45
    - informazioni che ho in merito ad un certo contesto
      - “nome del condottiero" ⟶ "Alessandro Magno”
      - “numero canti Divina Commedia" ⟶ 34
      - "temperatura città Cesena"⟶ 17
    - informazioni che fan parte della strategia per risolvere un problema
      - “numero città da visitare” ⟶ 20

IL CONCETTO DI VARIABILE

- Ogni **variabile** è rappresentata da un *nome* e dal *valore* associato
  - In Snap!
    - il nome può essere una descrizione con più parole 

    - il valore può essere di tipo diverso (numeri, stringhe, etc.) 

- Le variabili vengono definite con la stesura stessa copione, diventano un elemento fondamentale che permette di descrivere il copione stesso
  - In Snap! Categoria Variabili
    - ***creazione* nuova variabile**, *cancellazione* variabile esistente
    - variabili predefinite
- A tempo di esecuzione, l'insieme delle variabili definiscono la memoria dell'attore

VARIABILI - ASSEGNAMENTO E VALUTAZIONE

- Uso variabili - due aspetti principali:
  - **Assegnamento** e modifica
    - Data una variabile, possiamo assegnarle uno specifico valore (stabilendo l'associazione fra il nome e il valore) mediante opportune istruzioni/blocchi di assegnamento
    - In Snap! Categoria Variabili
      - "porta <Variabile> a <Valore>", "cambia <Variabile> di <Valore>"

  - **Valutazione**
    - Data una variabile, possiamo accedere/usare il valore ad essa associato indirettamente indicando il suo nome
      - in espressioni, nei parametri

- [Esempio Snap!] Quadrato con variabili

ESPRESSIONI CON VARIABILI

- I linguaggi di programmazione permettono in generale di scrivere ***espressioni*** come combinazioni di operatori e operandi, funzioni, e valori sia come costanti, sia in termini di variabili
  - per *funzione* possiamo intendere le funzioni matematiche, o più in generale una qualsiasi relazione che associa a un certo insieme di valori in ingresso (parametri di ingresso) uno specifico valore in uscita
    - es: radice_quadrata(16) 

, lunghezza_di("mondo") 

  - per *operatore* si intende una funzione che 1 o 2 valori in ingresso
    - con notazione *infissa* - 1+2 anziché +(1,2) 

- In Snap! categoria Operatori
  - operatori matematici, operatori che operano sulle stringhe, operatori che operano sui valori logici booleani.
  - [Esempio Snap!] quadrato con variabili ed espressioni

VARIABILI E INPUT

- Le variabili sono necessarie per poter rappresentare e tenere traccia di dati/informazioni inserite in **input** dall'utente
  - riferimento ai micromondi/applicazioni interattive
  - argomento trattato del dettaglio nel prossimo modulo
- In merito, sono disponibili istruzioni/blocchi per interagire con l'utente, in particolare per permettere ad un utente di inserire in ingresso a tempo di esecuzione una certa informazione/dato
  - Il valore inserito viene associato ad una variabile
- In Snap!
  - blocco "chiedi" nella categoria Sensori e variabile "risposta"
  - [Esempio Snap!] quadrato con lunghezza lato in input

RAPPRESENTARE CONDIZIONI CON VARIABILI E USO COSTRUTTO DI SELEZIONE

- Mediante espressioni con variabili possiamo rappresentare ***condizioni***
  - *espressioni condizionali* espressioni il cui valore può essere vero o falso
    - In Snap!: Categoria Operatori - blocchi di forma esagonale (incluse le costanti vero e falso)

- Il costrutto di selezione ci permette di compiere azioni diverse a seconda di una condizione, ovvero del valore di un'espressione condizionale
  - In Snap! blocchi “se <Cond>” 

  - [Esempio Snap!] quadrato completo con controllo input

CONDIZIONI CON VARIABILI E USO COSTRUTTO DI ITERAZIONE

- Le espressioni condizionali possono essere usate nei costrutti di iterazione ("ripeti") quando non è nota a priori il numero di volte per le quali deve essere ripetuta la sequenza di istruzioni
  - tuttavia è possibile identificare la condizione che specifica, ad esempio, fino a quando deve esserci la ripetizione
- In Snap! blocchi “ripeti fino a quando <Cond>”

  - [Esempio Snap!] quadrato completo con controllo input con ripeti

VARIABILI… CHE VARIANO

- Le variabili sono spesso usate per rappresentare entità o una grandezza con un valore che varia nel tempo, con l'esecuzione del programma
- Un utilizzo frequente è nei cicli / costrutti di iterazione (es: ripeti)
  - ad ogni iterazione possiamo avere variabili che assumono un valore diverso opportunamente aggiornando quello precedente
  - [Esempio Snap!] spirale ottenuta variando la lunghezza del lato

VARIABILI E OUTPUT

- Oltre ad avere istruzioni/blocchi che permettono di ottenere un'informazione in input (ingresso) dall'utente, dualmente abbiamo istruzioni/blocchi per produrre messaggi in output (uscita)
  - anche questo argomento verrà trattato nel dettaglio nel prossimo modul
- I messaggi in uscita possono includere valori contenuti nelle variabili
- In Snap!
  - alcuni blocchi predefiniti
    - "dire <Frase>" (Categoria Aspetto), blocco "scrivi <Frase>" (Categoria Penna)
  - [Esempio Snap!] Spirale con in output la lunghezza lato corrente

VARIABILI E PROBLEM SOLVING

- Dato un problema o progetto, le variabili ci permettono di rappresentare
  - sia i dati/informazioni di partenza, di cui conosciamo il valore
  - sia *ciò che dobbiamo determinare*, di cui verrà determinato con l'esecuzione il valore
  - sia tutto ciò che può essere utile rappresentare nel processo di risoluzione
    - "variabili di appoggio"
- Buona pratica
  - scegliere nomi significativi per le variabili

VARIABILI E ASTRAZIONE, GENERALIZZAZIONE

- Le variabili ci permettono di definire strategie/algoritmi/ragionamenti facendo riferimento alle grandezze/informazioni in termini di nomi/simboli e non direttamente dei valori
- Coinvolti due principi in relazione al pensiero computazionale e all'apprendimento
  - principio di *astrazione*
    - esprimiamo strategie e ragionamenti astraendo dai valori concreti
  - principio di *generalizzazione*
    - esprimiamo strategie e ragionamenti che valgono in generale, per tutti i valori concreti che rappresentiamo in modo simbolico con le variabili

ESEMPI E DISCUSSIONE

- [Esempio Snap!] micromondo problema di matematica
- [Esempio Snap!] micromondo parole
- [Esempio Snap!] Algoritmico | calcolo somma di tutti i valori in un intervallo {<N1>…<N2>}

GESTIRE LA COMPLESSITA' DI PROBLEMI, DI PROGRAMMI

- Metodi propri del pensiero computazionale
  - **Decomposizione**
  - **Astrazione**
  - **Generalizzazione**

DECOMPOSIZIONE

- *Capacità di scomporre un problema in sottoproblemi più semplici, risolvere questi separatamente e quindi trovare la soluzione complessiva componendo i risultati parziali*
  - cruciale per affrontare la complessità dei problemi
  - facilita il debugging, la gestione degli errori
  - supportare l'indeterminatezza gestita
- Aspetto metodologico
  - procedere incrementalmente verso la risoluzione di un problema o la creazione di un sistema
  - sia top-down (pianificazione), sia bottom-up (bricolage)

ASTRAZIONE

- *Capacità di ridurre la complessità nascondendo dettagli irrilevanti e focalizzando i concetti/elementi principali per il problema che si deve risolvere o progetto da sviluppare*
  - l'insieme dei concetti definisce il *livello di astrazione* a cui ci collochiamo
  - stretta relazione con il linguaggio con cui specifichiamo il micromondo
- Aspetto metodologico
  - supporto al procedere incrementale mediante la definizione di più livelli di astrazione, gerarchici
  - muoversi in verticale per livelli: zoom in/out, raffinamento/astrazione

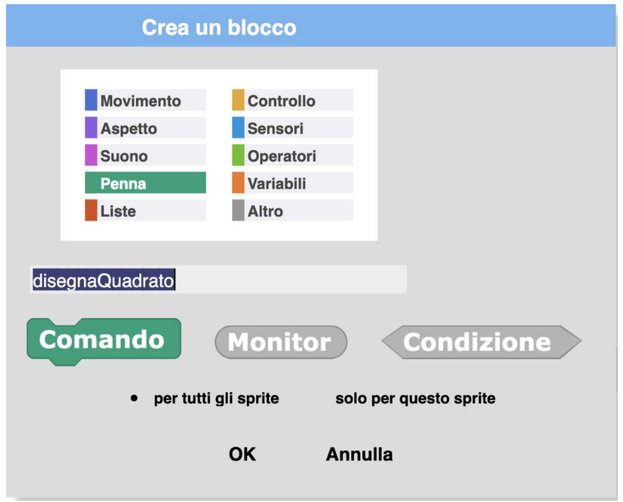
GENERALIZZAZIONE

- *Capacità di identificare soluzioni o approcci di progettazione utili non solo per una specifica istanza di problema o costruzione da fare, ma per tutte le istanze relative ad una classe/insieme/categoria*
- Aspetto metodologico
  - riuso concettuale e pratico di strategie, programmi
  - supportare nello sviluppo incrementale momenti di fattorizzazione fino all'identificazione di nuovi livelli di astrazione
  - ragionamento induttivo

MECCANISMO DI SUPPORTO IN SNAP!: CREAZIONE NUOVI BLOCCHI

- Possibilità di **creare nuovi blocchi** definendo nome e comportamento associato al blocco
  - il comportamento sempre definito in modo uniforme e procedurale come sequenza di istruzioni (blocchi)
    - tale sequenza definisce il corpo della procedura/nuovo blocco (chiamata sempre script)
  - [Esempio Snap!] micromondo geometrico con nuovo blocco non parametrizzato

Crea un blocco



NUOVI BLOCCHI COME "PROCEDURE"

- In gergo informatico (e in LOGO) il nuovo blocco è una ***procedura***
  - *programmazione procedurale*
  - Due aspetti distinti:
    - **definizione** della procedura
    - **chiamata/invocazione** della procedura
- Meccanismo importante dal punto di vista metodologico
  - decomposizione, astrazione e generalizzazione (a seguire)

CREAZIONE NUOVI BLOCCHI - DECOMPOSIZIONE

- Supporto per **decomposizione *procedurale*, *gerarchica***
  - decomponiamo un problema in sottoproblemi e definiamo una specifica procedura/blocco che risolva ogni singolo sottoproblema e quindi definiamo la procedura principale che chiama le specifiche **procedure**, nella sequenza giusta
  - in gergo informatico: *programmazione procedurale*
- Vantaggi in termini di
  - "modularità", comprensione del codice, identificazione degli errori
  - riuso: la specifica procedura/blocco definita per risolvere uno specifico problema la possiamo "riusare" in tutti i contesti in cui può essere utile

CREAZIONE NUOVI BLOCCHI - ASTRAZIONE

- Supporto per astrazione
  - ogni nuova **procedura** o blocco permette di essere usata (chiamata) astraendo da come sia fatta internamente, da quale sia la specifica sequenza di istruzioni che ne costituiscono il corpo (implementazione)
    - rende visibile il *cosa di vuole fare* e nasconde il *come*
- Questo ci permette di definire un micromondo del *giusto livello di astrazione* rispetto agli obiettivi che abbiamo
  - portando in primo piano concetti che ci interessano e invece nascondendo concetti e aspetti che non sono rilevanti (per il livello di astrazione scelto)

DEFINIZIONE DI NUOVI BLOCCHI CON PARAMETRI

- Un aspetto molto importante è la possibilità di specificare anche *parametri* nella definizione di una nuova procedura o blocco
  - questo consente di estendere l'utilità la nuova procedura/blocco in tutti i casi definiti dai valori concreti usati per per i parametri
  - nella definizione della procedura/blocco, i parametri sono rappresentati come variabili che possono essere riferite all'interno del corpo della procedura/blocco
    - [Esempio Snap!] micromondo geometrico con nuovo blocco parametrizzato
- Forma di generalizzazione

PROCEDURE E VARIABILI LOCALI

- Quando si definiscono nuove procedure/blocchi, è frequente la necessità o utilità di definire variabili che sono usate solo nel corpo/script della procedura
  - sono variabili "locali" allo script, che rimangono in memoria per il solo tempo di esecuzione del nuovo blocco
  - In Snap!
    - le variabili locali sono definibili mediante un blocco 

nella categoria Variabili
    - [Esempio Snap!] procedura (nuovo blocco) disegna spirale

SCEGLIERE I BLOCCHI PER DEFINIRE IL LIVELLO DI ASTRAZIONE DI BASE

- In Snap! c’è la possibilità non solo di *creare*, ma anche di **nascondere blocchi esistenti**
  - questo consente di allestire micromondo come ambiente di apprendimento più opportuno scegliendo i concetti chiave di base
  - ovvero, in generale: *scegliere il livello di astrazione a cui collocare l’attività di problem solving o di progettazione degli studenti*

ESEMPI E DISCUSSIONE

- [Esempio Snap!] esempi precedenti rivisti con la definizione di nuovi blocchi
- [Esempio Snap!] micromondo racconto strutturato