Lezione 1.2:
Rilevare e risolvere problemi nel codice di programmazione

Introduzione

In questo materiale ci concentreremo su un'abilità fondamentale per ogni programmatore: il **debugging**, ovvero l'arte di trovare e correggere errori nel codice. Anche i programmatori più esperti commettono errori regolarmente - la differenza è che sanno come trovarli e risolverli efficacemente.

Obiettivi di apprendimento

Al termine di questa lezione, sarai in grado di:

- Identificare i tipi più comuni di errori nella programmazione
- Usare tecniche sistematiche per localizzare errori nel codice
- Correggere errori in sequenze di istruzioni
- Analizzare perché un programma non produce l'output previsto
- Ottimizzare il codice per migliorare i tempi di esecuzione

Indice dei Contenuti

1. Tipi di errori nella programmazione

Gli errori nei programmi possono essere classificati in tre categorie principali:

1.1 Errori di sintassi

Sono errori nella "grammatica" del linguaggio di programmazione. In Scratch, questi errori sono rari grazie all'interfaccia a blocchi, ma possono verificarsi quando:

- Mancano blocchi essenziali (come il blocco iniziale "Quando si clicca sulla bandiera verde")
- I blocchi non sono correttamente connessi
- Le strutture nidificate non sono completate correttamente

1.2 Errori logici

Sono errori nel ragionamento o nella logica del programma. Il codice viene eseguito senza errori visibili, ma non fa ciò che dovrebbe:

- Condizioni errate in un blocco "Se"
- Ordine sbagliato delle istruzioni
- Calcoli matematici errati
- Valori iniziali inappropriate

1.3 Errori di esecuzione

Si verificano durante l'esecuzione del programma:

- Divisione per zero
- Accesso a variabili non inizializzate
- Cicli infiniti che causano il blocco del programma

by: Ulises Gabriel Miranda – mail@ulises.it

2. Identificare i problemi nel codice

2.1 Osservare i sintomi

Il primo passo per risolvere un problema è identificare esattamente cosa non funziona:

- Il programma si blocca?
- Il programma funziona ma produce risultati errati?
- Alcune parti del programma funzionano mentre altre no?
- Il programma è troppo lento?

Lo strumento **SCRATCH** per il debugging analitico:

https://scratch.mit.edu/download

2.2 Tecniche di diagnosi

2.2.1 Esecuzione passo-passo

In Scratch puoi eseguire il codice un blocco alla volta per vedere dove si verifica il problema:

1. Fai clic destro sul blocco di codice
2. Seleziona "Esegui passo-passo"

2.2.2 Usa blocchi "Dire"

Inserisci blocchi "Dire" per visualizzare il valore delle variabili in punti critici del programma:

Quando si clicca sulla bandiera verde

Imposta [contatore] a [0]

Ripeti 10 volte

Cambia [contatore] di [1]

by: Ulises Gabriel Miranda – mail@ulises.it

Dire (unione di "Contatore = " e (contatore)) per [1] secondi

Fine

2.2.3 Isola il problema

Se il programma è complesso, prova a disattivare temporaneamente alcune parti per isolare quella problematica:

- Commenta (disattiva) parti del codice
- Testa una sezione alla volta
- Semplifica il programma fino a quando funziona, poi aggiungi complessità

2.2.4 Monitoraggio delle variabili

Usa il monitoraggio delle variabili per vedere come cambiano durante l'esecuzione:

1. Fai clic destro su una variabile nello script
2. Seleziona "Mostra variabile"

3. Errori comuni e come risolverli

3.1 Problemi di sequenza

Esempio di problema: Un personaggio deve prima saltare e poi cambiare costume, ma le azioni avvengono nell'ordine sbagliato.

Soluzione: Verifica l'ordine dei blocchi e riorganizzali correttamente:

Codice errato

Quando si clicca sulla bandiera verde

by: Ulises Gabriel Miranda – mail@ulises.it

Cambia costume a [costume2]

Cambia y di [50]

Attendi [1] secondi

Cambia y di [-50]

Codice corretto

Quando si clicca sulla bandiera verde

Cambia y di [50]

Cambia costume a [costume2]

Attendi [1] secondi

Cambia y di [-50]

3.2 Problemi con le condizioni

Esempio di problema: Un personaggio dovrebbe muoversi solo quando non tocca un ostacolo, ma continua a muoversi anche quando lo tocca.

Soluzione: Verifica la condizione e correggila:

Codice errato

Quando si clicca sulla bandiera verde

Per sempre

  Se <non <sta toccando [bordo]?>> allora

    Fai [10] passi

  Fine

Fine

Codice corretto

Quando si clicca sulla bandiera verde

by: Ulises Gabriel Miranda – mail@ulises.it

```
Per sempre
  Se <non <sta toccando [ostacolo]?>> allora
    Fai [10] passi
  Fine
Fine
```

3.3 Problemi con i cicli

Esempio di problema: Un programma che dovrebbe terminare dopo 10 iterazioni continua indefinitamente.

Soluzione: Verifica il tipo di ciclo e le condizioni di uscita:

Codice errato

```
Quando si clicca sulla bandiera verde
Imposta [contatore] a [0]
Per sempre
  Cambia [contatore] di [1]
  Se <(contatore) = [10]> allora
    Dire [Finito!]
  Fine
Fine
```

Codice corretto

```
Quando si clicca sulla bandiera verde
Imposta [contatore] a [0]
Ripeti fino a quando <(contatore) = [10]>
  Cambia [contatore] di [1]
```


by: Ulises Gabriel Miranda – mail@ulises.it

Fine

Dire [Finito!]

3.4 Problemi di sincronizzazione

Esempio di problema: Due sprite devono muoversi contemporaneamente, ma uno parte prima dell'altro.

Soluzione: Usa messaggi per sincronizzare le azioni:

Sprite 1

Quando si clicca sulla bandiera verde

Attendi [2] secondi

Invia a tutti [inizia] e attendi

Sprite 2

Quando ricevo [inizia]

Fai [10] passi

4. Ottimizzare programmi lenti

A volte il programma funziona, ma è troppo lento o inefficiente. Ecco alcune tecniche per migliorare le prestazioni:

4.1 Ridurre operazioni ridondanti

Esempio di problema: Un programma che calcola ripetutamente lo stesso valore.

Soluzione: Calcola il valore una volta e memorizzalo:

Codice inefficiente

Quando si clicca sulla bandiera verde

Per sempre

Imposta [area] a ((lunghezza) * (larghezza))

by: Ulises Gabriel Miranda – mail@ulises.it

```
    Imposta [perimetro] a ((lunghezza) * [2] + (larghezza) * [2])
    Attendi [0.1] secondi
  Fine

# Codice ottimizzato
  Quando si clicca sulla bandiera verde
  Imposta [area] a ((lunghezza) * (larghezza))
  Imposta [perimetro] a ((lunghezza) * [2] + (larghezza) * [2])
  Per sempre
    Se <(lunghezza) ≠ [ultimo_lung]> allora
      Imposta [ultimo_lung] a (lunghezza)
      Imposta [area] a ((lunghezza) * (larghezza))
      Imposta [perimetro] a ((lunghezza) * [2] + (larghezza) * [2])
    Fine
    Se <(larghezza) ≠ [ultimo_larg]> allora
      Imposta [ultimo_larg] a (larghezza)
      Imposta [area] a ((lunghezza) * (larghezza))
      Imposta [perimetro] a ((lunghezza) * [2] + (larghezza) * [2])
    Fine
    Attendi [0.1] secondi
  Fine
```

4.2 Ottimizzare i cicli

Esempio di problema: Un ciclo che esegue troppe iterazioni.

Soluzione: Riduci il numero di iterazioni o usa un approccio diverso:

```
# Codice inefficiente
```


by: Ulises Gabriel Miranda – mail@ulises.it

```
Quando si clicca sulla bandiera verde
Ripeti [1000] volte
  Cambia [x] di [0.1]
Fine
```

Codice ottimizzato

```
Quando si clicca sulla bandiera verde
Cambia [x] di [100]
```

4.3 Limitare gli effetti grafici

I calcoli grafici intensivi possono rallentare il programma:

Codice che rallenta il programma

```
Quando si clicca sulla bandiera verde
Per sempre
  Cambia effetto [fantasma] di [1]
  Cambia effetto [luminosità] di [1]
  Cambia effetto [mosaico] di [1]
  Ruota di [1] gradi
Fine
```

Codice ottimizzato

```
Quando si clicca sulla bandiera verde
Per sempre
  Se <(timer) > [0.1]> allora
    Cambia effetto [fantasma] di [10]
    Cambia effetto [luminosità] di [10]
```

Cambia effetto [mosaico] di [10]

Ruota di [10] gradi

Riavvia timer

Fine

Fine

by: Ulises Gabriel Miranda – mail@ulises.it

5. Studio di casi pratici

Caso 1: Il labirinto che non funziona

Problema: Hai creato un gioco del labirinto, ma il personaggio attraversa i muri invece di fermarsi.

Analisi:

1. Il codice per il movimento del personaggio non controlla correttamente le collisioni
2. Il colore del muro potrebbe non essere esattamente quello specificato nel blocco "sta toccando il colore"

Soluzione:

```
# Codice corretto per il movimento
Quando si clicca sulla bandiera verde
Per sempre
  Se <tasto [freccia su] premuto?> allora
    Punta in direzione [0]
    Se <non <sta toccando il colore [#000000]?>> allora
      Fai [5] passi
    Altrimenti
      Fai [-5] passi
    Fine
  Fine
  # Ripeti per le altre direzioni
Fine
```


by: Ulises Gabriel Miranda – mail@ulises.it

Caso 2: Il contatore che non si aggiorna

Problema: Hai un gioco in cui raccogli monete, ma il punteggio non aumenta quando le tocchi.

Analisi:

1. Il codice per rilevare la collisione potrebbe essere nel posto sbagliato
2. La variabile punteggio potrebbe non essere incrementata correttamente
3. La moneta potrebbe scomparire prima che il punteggio venga aggiornato

Soluzione:

```
# Nel codice dello sprite moneta
Quando vengo clonato
Mostra
Per sempre
  Se <sta toccando [giocatore]?> allora
    Cambia [punteggio] di [1]
    Elimina questo clone
  Fine
Fine
```

6. Attività pratica di debugging

Esercizio 1: Trova e correggi gli errori

Ecco un programma con alcuni errori. Individua e correggi i problemi:

```
# Programma con errori
    Quando si clicca sulla bandiera verde
    Imposta [punteggio] a [0]
    Per sempre
```


by: Ulises Gabriel Miranda – mail@ulises.it

```
    Se <tasto [freccia su] premuto> allora
      Punta verso [0] gradi
      Fai [10] passi
    Fine
    Se <sta toccando [moneta]> allora
      Elimina questo clone
      Cambia [punteggio] di [1]
    Fine
  Fine
```

Esercizio 2: Ottimizza il programma

Questo programma funziona, ma è inefficiente. Come lo ottimizzeresti?

```
# Programma inefficiente
    Quando si clicca sulla bandiera verde
    Per sempre
      Ripeti [360] volte
        Ruota di [1] gradi
        Fai [1] passi
        Se <sta toccando [bordo]?> allora
          Rimbalza
        Fine
      Fine
    Fine
```


by: Ulises Gabriel Miranda – mail@ulises.it

Conclusione

In questa lezione abbiamo esplorato le tecniche per identificare e risolvere problemi nel codice. Ricorda che il debugging è una parte normale e importante del processo di programmazione. Anche i programmatori più esperti devono dedicare tempo a trovare e correggere errori.

Con la pratica, diventerai sempre più abile nel riconoscere pattern di errori comuni e nel trovare soluzioni efficienti. Non scoraggiarti quando incontri problemi - sono opportunità per imparare e migliorare le tue capacità di risoluzione dei problemi!

Bibliografia e risorse

Libri in italiano

- Camuso, A. (2018). *Debugging: L'arte di trovare e correggere gli errori nel codice*. Apogeo.
- Dix, P. (2019). *Pensiero computazionale e debugging*. Erickson.
- Stilli, G. (2020). *Scratch 3.0: Guida pratica con soluzioni ai problemi comuni*. Edizioni LSWR.

Risorse online in italiano

- Guida al debugging in Scratch - Forum ufficiale di Scratch
- Corso di debugging - Programma il Futuro - Lezioni dedicate al debugging
- CodeMotion Kids - Debugging - Risorse didattiche gratuite
- BugBuster - Impara a correggere gli errori - Piattaforma educativa italiana

Video tutorial in italiano

- Debugging in Scratch - Alessandro Bogliolo
- Come trovare errori nei tuoi programmi - CodeWeek Italia
- Tecniche di debugging per bambini - RaiScuola

Strumenti online

- Scratch Debug It! puzzles - Esercizi specifici per praticare il debugging (interfaccia in inglese ma utilizzabile in italiano)
- Esempi pratici: PONG – download e analisi in diverse versioni https://scratch.mit.edu/search/projects?q=pong
- Scratch Debugging Cards - Schede didattiche in italiano
- Esempio di debugging interattivo - Sito con esempi interattivi

Comunità di supporto

- Forum Scratch Italia - Sezione italiana del forum ufficiale di Scratch
- Gruppi Facebook "Coding a scuola" - Comunità di insegnanti e appassionati
- Italia Programma - Comunità italiana dedicata all'insegnamento della programmazione

Allegato 1: Pong tramite Scratch

Esercitazione pratica:

Mini-Pong: Un gioco semplice da creare con Scratch

Descrizione del progetto

In questo tutorial creeremo una versione semplificata del classico gioco Pong: una pallina rimbalzerà sullo schermo e il giocatore dovrà impedirle di toccare il bordo inferiore muovendo una racchetta con il mouse.

Materiali necessari

- Computer con accesso a Internet
- Account Scratch (opzionale, puoi usare Scratch anche senza registrarti)
- Accesso al sito web di Scratch: scratch.mit.edu

Tempo di realizzazione

Circa 20-30 minuti

Istruzioni passo-passo

1. Prepara il progetto

1. Vai su scratch.mit.edu e clicca su "Crea" (o se preferisci, accedi al tuo account)
2. Elimina lo sprite del gatto facendo clic destro su di esso e selezionando "Elimina"

2. Crea la racchetta

1. Clicca sull'icona "Scegli uno Sprite" in basso a destra
2. Cerca "paddle" o "racchetta" (o disegnala tu stesso nella sezione "Disegna")
3. Seleziona la racchetta che preferisci

Programma la racchetta:

Aggiungi questo script alla racchetta:

quando si clicca su [bandiera verde]

vai a x: (0) y: (-150)

per sempre

vai a x: (posizione x del mouse) y: (-150)

fine

3. Crea la pallina

1. Clicca sull'icona "Scegli uno Sprite" in basso a destra
2. Cerca "ball" o "palla" (o crea una semplice palla rotonda usando l'editor di disegno)
3. Seleziona una palla

Programma la pallina:

Aggiungi questo script alla pallina:

```
quando si clicca su [bandiera verde]
vai a x: (0) y: (0)
punta in direzione (45)
ripeti fino a quando <sta toccando [bordo v]?>
 fai (10) passi
 rimbalza quando tocchi il bordo
 se <sta toccando [Racchetta v]?> allora
  punta in direzione ((180) - (direzione))
  cambia [punteggio v] di (1)
 fine
fine
dire [Game Over!] per (2) secondi
```

4. Aggiungi un punteggio

1. Clicca sul pulsante "Variabili" nella sezione degli script
2. Clicca su "Crea una Variabile"
3. Chiama la variabile "punteggio" e clicca su OK
4. Assicurati che la casella accanto alla variabile sia selezionata (così sarà visibile sullo stage)

by: Ulises Gabriel Miranda – mail@ulises.it

Aggiungi questo script allo Stage:

quando si clicca su [bandiera verde]

porta [punteggio v] a (0)

5. Testa il gioco

1. Clicca sulla bandiera verde per avviare il gioco
2. Muovi il mouse per controllare la racchetta
3. Cerca di far rimbalzare la pallina sulla racchetta il più a lungo possibile
4. Il gioco termina quando la pallina tocca il bordo inferiore

Sfide aggiuntive

Se vuoi rendere il gioco più interessante, prova ad aggiungere queste funzionalità:

Sfida 1: Aumenta la velocità

Aggiungi questo codice alla pallina, subito dopo il blocco "cambia [punteggio] di (1)":

cambia [velocità v] di (0.5)

Non dimenticare di creare la variabile "velocità" e di impostarla a 10 all'inizio del gioco. Poi sostituisci il blocco "fai (10) passi" con "fai (velocità) passi".

Sfida 2: Aggiungi un suono

Aggiungi un suono quando la pallina colpisce la racchetta:

1. Seleziona la pallina
2. Vai alla scheda "Suoni"
3. Aggiungi un nuovo suono (per esempio "pop")
4. Nel codice, dopo il blocco "punta in direzione ((180) - (direzione))", aggiungi:

produci suono [pop v]

Sfida 3: Aggiungi un conteggio delle vite

1. Crea una nuova variabile chiamata "vite"
2. Imposta le vite a 3 all'inizio del gioco
3. Invece di terminare il gioco immediatamente, quando la pallina tocca il bordo inferiore:

- Riduci le vite di 1
- Riporta la pallina al centro
- Continua a giocare finché le vite non sono esaurite

Aspetti educativi

Con questo semplice progetto, imparerai:

- Come controllare gli sprite con il mouse
- Come rilevare le collisioni
- Come usare le variabili per il punteggio
- Come utilizzare i cicli e le condizioni

Risoluzione dei problemi

Se la pallina rimbalza in modo strano:

- Verifica che la racchetta sia posizionata correttamente
- Assicurati che la direzione iniziale della pallina sia impostata a 45 gradi
- Controlla che il codice di rimbalzo funzioni correttamente

Se la pallina attraversa la racchetta:

- Prova a ridurre la velocità della pallina
- Assicurati che il codice rilevi correttamente quando la pallina tocca la racchetta

Estensioni creative

Quando hai completato il gioco base, prova a:

- Aggiungere uno sfondo colorato
- Creare diversi livelli di difficoltà
- Aggiungere effetti speciali come particelle quando la pallina colpisce la racchetta
- Aggiungere un punteggio massimo che viene salvato tra le partite

by: Ulises Gabriel Miranda – mail@ulises.it