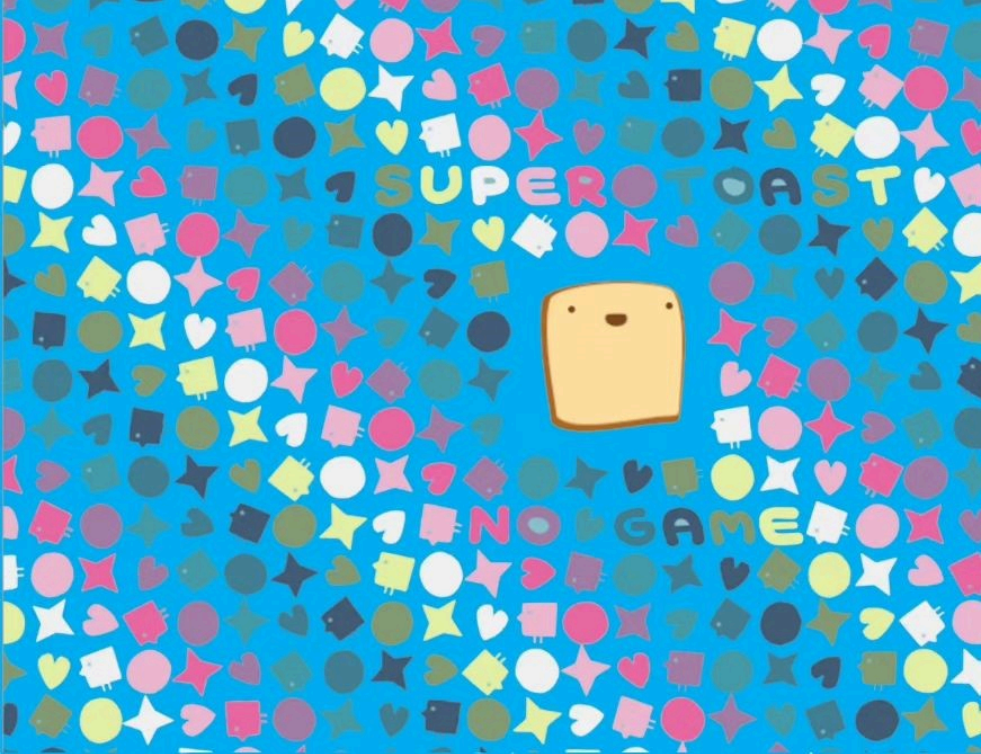


# LOVE 2D

Un piccolo corso d'introduzione (si spera)

by Ganzio Bello



# Giuseppe Lantieri

## Introduzione a Love2D

Chi non ha mai programmato potrà provare a creare un videogioco insieme ad altrx, scrivendo codice in LUA. L'obiettivo è fornire strumenti e metodi a chi non mastica troppa informatica.

Ciao a tuttə, mi  
presento!

Ganzio Bello

Programmatore da 4 anni

Studente Universitario

Boh facciamo -3 a una laurea se il  
trasferimento va bene

instagram => @ganzio.bello

# Grazie per avermi rinvitato!

## Sono felicissimo di essere di nuovo qui! Altri 5 mandati si !

Vi riassumo l'esperienza con un grafico e un piccolo riassunto di quello che hanno detto i miei amici



# Piccola Dichiarazione di Intenti

Questo non è e non vuole essere un *corso esaustivo*.

L'obiettivo è divertire per un'oretta incuriosendo il più possibile e darvi l'incentivo a provare il framework. Se volete buttarvi nella creazione di videogiochi spetta a voi, non esiste il corso giusto che potrà sostituire volontà ed iniziativa.

Io qui parlo di basi e faccio battute brutte.

Detto questo spero che possa piacervi lo stesso e che ci possiamo divertire insieme.

# Iniziamo!

# Cos'è LOVE 2D?




## Probabilmente quello che non ti aspetti !

LOVE 2D è un framework che permette di creare videogiochi 2D grazie a del codice scritto in LUA.

È un abbastanza apprezzato dalla community, considerate che è stato usato per creare il recente gioco Balatro.

Può perfino essere usato da neofiti che vogliono iniziare a comprendere le basi.

### Punti Forti:

-  Text-based - Tutto dal testo nessuna interfaccia
-  OpenSource - Ottima etica e una community vastissima
-  Multi-Piattaforma - Dal web al computer arrivando perfino alle console.

*Nessuna limitazione se non data dalla bravura*

# Come mai scegliere LOVE 2D?

## Perché è divertente

Per sviluppare un videogioco ci sono tantissime alternative tra cui scegliere.

Sicuramente se vi siete interessati all'argomento conoscerete i 3 grandi nomi in questo momento:

- Unity
- Unreal
- Godot

Ma oltre a loro ci sono Engine e framework molto usati come GameMaker o PyGame o RPG Maker. Nessuno di questi strumenti è definitivo o indispensabile per fare un buon gioco, ognuno di essi eccelle in qualcosa e pecca in altro.

# Come mi diverto con LOVE 2D

## Mi piace reinventare la ruota

Come accennato prima LOVE 2D è un framework, un banco di lavoro con degli strumenti.

Offre tanta libertà al costruttore e non impone molti limiti nella creazione.

Questo è sia un grande vantaggio, sia un estremo svantaggio.

Infatti non tutti vogliono gestire, fin dall'inizio, tutte le complessità che un gioco può creare, ma per progetti abbastanza semplici la libertà lasciata allo sviluppatore gli permette una maggiore crescita.

Io adoro questo lato e trovo molto divertente cercare di riprodurre cose viste in altri framework o inventare roba inutile.



# Perchè LUA e non Python ?

## Installa python su

È proprio vero Python è un linguaggio unico nel suo genere che piace alla maggioranza delle persone, io non incluso, ma tolto che impararne uno non escluda l'altro, Lua ha dei validi motivi per non essere da meno.

Innanzitutto Lua è:

- Già facilmente eseguibile da qualsiasi sistema operativo
- Molto semplice da leggere, capire e apprendere





# Questo corso doveva essere per novizi

## Quindi in caso ci siano, leggere queste slide

Quando avevo scritto la mail di presentazione, la mia idea era di fare un piccolo corso a chiunque. Programmatori e non. Perciò, prima di iniziare con gli argomenti più complessi, un rapido ripasso di concetti fondamentali nella programmazione:

- Le Variabili
- Le Funzioni
- Le Tabelle e Gli Oggetti
- I Controlli
- I Cicli

# I primi concetti

## Variabili

Cos'è una variabile? Fissiamo il significato di variabile come quella cosa che porta dentro di sé un valore che può variare nel tempo.

Per gli informatici le costanti sono solo un caso speciale delle variabili.

Quindi tutto ciò che deve essere memorizzato deve essere contenuto in una variabile.

```
VariabileX = 10;  
PIPP0 = 20;  
topolino = 30;  
VariabileX = VariabileX + PIPPO;  
  
topolino == VariabileX -- True
```

Quello che si può osservare è che in Lua è molto facile dichiarare una variabile: basta scrivere il suo nome e assegnargli un valore.

# I primi concetti

## Funzione

```
function NomeFunzione(argomentoFunzione1, argomentoFunzione2, ....)  
    return risultatoFunzione  
end
```

Questa è una funzione. In programmazione ogni codice è composto da funzioni e da variabili.

Le funzioni rappresentano il nostro modo di esprimere le operazioni che vogliamo.

Se non avete mai programmato è un concetto particolare da imparare ma con degli esempi sarà più facile, la funzione di somma appare così:

```
function Somma(a,b)  
    return a+b  
end
```

# I primi concetti

## Liste e Oggetti

In Lua le parentesi graffe inserite dopo il nome di una variabile possono significare sia una lista numerata di variabili, sia roba molto più complessa e difficile da definire in maniera precisa come un oggetto.

Mi spiego peggio:

- Una lista:

```
listaSpesa = {"banane", "pesche", "pane", "vino"}
```

- Un Oggetto:

```
coloriCasa = {  
    tetto = "rosso",  
    cucina = "gialla",  
    porta = "marrone"  
}
```

Avevi detto che era semplice!

Come faccio a capire cosa è uno o l'altro?

Ok, capisco che l'esempio di prima possa confondere.

Ma non sono proprio identici e difficilmente scambierete mai uno per l'altro.

Senza contesto possono sembrare uguali e confondersi facilmente, ma fidatevi che succederà di rado (o almeno si spera).

# I primi concetti

## I Controlli

La parola riservata "if", ci permette di esprimere condizioni che si devono soddisfare prima di effettuare determinato codice:

```
if condizione1 then  
  
elseif condizione2 then  
  
else  
  
end
```

# I primi concetti

## I Cicli

I cicli nella programmazione sono essenziali, poichè ci permettono di esprimere più facilmente certe operazioni. Oggi vedremo in particolare due tipi di ciclo:

- **for** semplice:

```
for indice = N_INIZIALE, N_TOTALE, INCREMENTO do  
  
    end
```

- **for** di una lista:

```
for indice, value in ipairs(Lista) do  
  
    end
```

# Prepariamoci ad iniziare

## Mettiamo le mani nelle installazioni

Iniziamo installando tutto ciò che è necessario:

- Love 2D
- VS Code, potete scegliere il vostro editor preferito

Tutto qui o per lo meno tutto qui quello necessario. Ora iniziamo a settare tutto quello che può essere utile

- Git, perché prima o poi vi servirà o lo vorrete
- Template, un template per VSCode



# Perciò tutto pronto per iniziare

## Iniziamo a programmare

L'esempio che vi porto oggi è come si può creare un clone di Flappy Bird.

Prima di iniziare a scrivere qualsiasi riga di codice, dobbiamo avere ben in mente il gioco che vogliamo realizzare:

- Le meccaniche
- La grafica
- I suoni
- I menù

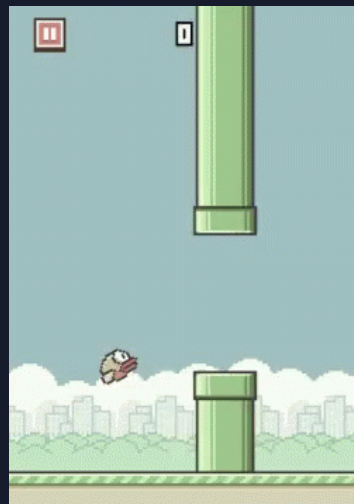


# Per capire cosa intendo con meccaniche:

In Flappy Bird abbiamo un personaggio, posizionato a sinistra dello schermo, spinto giù da una forza invisibile, che sale in alto ogni volta che premiamo un pulsante.

Scomponiamo il problema in domande da singola soluzione:

- Come facciamo a farlo apparire a schermo?
- Come facciamo a farlo interagire con i nostri comandi?
- Come facciamo i tubi?



# Disegno

## Far apparire le cose a schermo

Per disegnare qualsiasi cosa a schermo in LOVE, devo conoscere le sue coordinate e la sua apparenza.

Le coordinate servono per collocarlo all'interno dello schermo in una determinata posizione, LOVE, infatti, sfrutta un piano cartesiano con asse delle x che va da sinistra a destra e un'asse delle y che va dall'alto al basso del nostro schermo.



# Disegno

## Far apparire le cose a schermo

Una volta capito dove vogliamo far apparire una determinata cosa, possiamo sfruttare diverse funzioni per disegnare quello che vogliamo:

```
-- Disegnare un cerchio con un' area colorata
love.graphics.circle("fill", x, y, raggio )

-- Disegnare un rettangolo con solo il perimetro colorato
love.graphics.rectangle("line", x, y, larghezza, altezza)

-- Disegnare un' immagine o roba più complessa
love.graphics.draw(Immagine, x, y, angolo)
```

# Logica

Far fare cose alle cose

Qui ci vengono in soccorso i concetti accennati prima: Variabili e Funzioni.

# Variabili

Usate per immagazzinare informazioni.

In generale per affrontare problemi come:

- Stato di un elemento
- Costanti da dover decidere

# Funzioni

Usate per apportare delle modifiche.

In generale conviene scrivere una funzione ogni volta che sappiamo che del codice si ripeterà tante volte.

Conviene, inoltre, che non siano troppe lunghe o troppo corte

Detto questo, dovremmo avere tutte le conoscenze necessarie per procedere ad aprire il file main.lua

# Primi passi

Apriamo il file chiamato main.lua e all'interno troveremo:

```
function love.load()
end

function love.update()
    print("Hello!")
end

function love.draw()
end
```



# In realtà c'è questa parte omessa

```
local IS_DEBUG = os.getenv("LOCAL_LUA_DEBUGGER_VSCODE") == "1" and arg[2] == "debug"
if IS_DEBUG then
    require("lldebugger").start()

    function love.errorhandler(msg)
        error(msg, 2)
    end
end
```

Questo codice è qui per fornirci una migliore esperienza di debug, ma visto che in questo momento non ci interessa andare così a fondo su certi argomenti, lo ignoreremo.

# Le prime funzioni di LOVE 2D

## Le tre sorelle

```
function love.load()  
end
```

```
function love.update()  
    print("Hello!")  
end
```

```
function love.draw()  
end
```



# Le prime funzioni di LOVE 2D

## Le tre sorelle

Per aiutarci a descrivere il nostro gioco al meglio, LOVE ci offre tre funzioni che saranno sicuramente indispensabili.

- Load viene richiamata quando il nostro gioco viene avviato, qui dovremo scrivere tutte le nostre variabili che ci servono nel gioco.
- Update è dove avverranno la maggior parte degli aggiornamenti alle nostre variabili.
- Draw è la funzione che si occupa di disegnare a schermo ogni singolo frame

Queste funzioni vengono richiamate dall'avvio del gioco in questo ordine:

load-> update-> draw-> update-> draw-> update-> draw-> update-> draw-> ...

Il fatto che draw venga richiamata ogni tot secondi creerà il nostro frame rate finale.



# Ma quindi questo volatile?

```
function love.load()
  Bird = {
    x = 100,
    y = WHeight / 2,
    width = 10,
    height = 10,
  };
end

function love.update(dt)
  ApplyGravity(dt)
end

function love.draw()
  love.graphics.setColor(255, 255, 255, 1);
  love.graphics.rectangle("fill", Bird.x, Bird.y, Bird.width, Bird.height);
end

function ApplyGravity(dt)
end
```

# Bello ma come funziona la Gravità?

A mele! Ovvio no?

Possiamo vedere la Gravità come uno spostamento verso il basso a velocità costante. Quindi in parole povere ad ogni frame la y del nostro volatile diventerà sempre più grande.

Quando sposti verso l'alto =>



CARLO STAI  
CARLO STAI A TERRA PORCA



# Bello ma come funziona la Gravità?

## Riportiamo a terra Carlo

```
function ApplyGravity(dt)
    WWidth, WHeight = love.graphics.getDimensions()

    if Bird.y < (WHeight - Bird.height) then
        Bird.y = Bird.y + SPEED * dt;
    end
    if Bird.y > (WHeight - Bird.height) then
        Bird.y = (WHeight - Bird.height);
    end
end
```

# Ok ora i tubi

```
function DetectCollision()
  for _, value in ipairs(Pipes) do
    if Bird.x ≥ value.x and Bird.x + Bird.width ≤ value.x + value.width and
       Bird.y ≥ value.y and Bird.y + Bird.height ≤ value.y + value.height or
       Bird.x ≥ value.x and Bird.x + Bird.width ≤ value.x + value.width and
       Bird.y ≥ (value.height + value.gap) and
       Bird.y + Bird.height ≤ ((value.height + value.gap) + (WHeight - (value.height + value.gap)))
    then
      print("Hai preso il tubo!");
    end
  end
end

function love.draw()
  -- codice precedente
  for _, value in ipairs(Pipes) do
    love.graphics.rectangle("fill", value.x, value.y, value.width, value.height);
    love.graphics.rectangle("fill", value.x, (value.height + value.gap), value.width,
      (WHeight - (value.height + value.gap)));
  end
end
```

# E ora, per ultimo, i comandi

```
function love.keypressed(key)
  if (key == "space") then
    if (GameState == 1) then
      Bird.y = Bird.y - JUMP;
      love.audio.play(SwooshSFX);
    end
  end
  if key == "left" then
    if (Bird.x > 0) then
      Bird.x = Bird.x - JUMP;
    end
  end
  if key == "right" then
    if Bird.x < (love.graphics.getWidth() - Bird.width) then
      Bird.x = Bird.x + JUMP;
    end
  end
end
```





è così abbiamo fatto  
pure gli input



E ora il gioco è finito!



e ora il gioco è finito?



Menù

Interfacce

Punteggi

Audio

Perciò, passiamo alla repository preparata in precedenza

<https://github.com/giuseppe-lantieri/loving-birds>

## Link utili a chi vuole approfondire:

- [Libro introduttivo](#)
- [Wiki ufficiale](#)
- [Mio amore !\[\]\(a22ba4e13c745edbf29e51af246c4c12\_img.jpg\) Challacade !\[\]\(33b18af9a4b997eb52666cfeb3c44157\_img.jpg\)](#)
- [Lista di cose Awesome](#)