



Pygame Zero & Polars

La Battaglia Finale – Harry vs Voldemort

Approfondimento su Data Science e Game Logic

 III Liceo Scientifico Biella - Scienze Applicate

 Python Biella Group



Obiettivi Didattici Avanzati

In questo progetto non ci limiteremo a muovere sprite, ma impareremo a:

1. **Gestione Dati Esterna:** Separare il codice dai dati (incantesimi) usando file CSV/Excel.
2. **Motore di Calcolo:** Usare **Polars** per interrogare velocemente le tabelle.
3. **State Machine:** Gestire il passaggio tra turno del giocatore, animazione e turno della CPU.
4. **Feedback Visivo:** Creare animazioni che comunichino al giocatore cosa sta succedendo (danni, fallimenti, vittorie).
5. **User Interface (UI):** Disegnare menu dinamici basati su coordinate matematiche.



Polars: Gestire il "Libro degli Incantesimi"

Invece di scrivere centinaia di righe per ogni incantesimo, usiamo un database:

```
import polars as pl

incantesimi_df = pl.read_csv("spells.csv")

def ottieni_opzioni(personaggio):
    return incantesimi_df.filter(pl.col("character") == personaggio)
```

Perché Polars? È una libreria di Data Science estremamente veloce. Qui la usiamo per trattare il gioco come un sistema basato su dati: se vogliamo aggiungere nuovi incantesimi, modifichiamo il file Excel senza toccare una riga di codice Python!



La Gestione dei Turni (State Control)

Il gioco non avviene tutto insieme. Abbiamo bisogno di variabili che controllino il "flusso":

```
attesa_input = True   # Il gioco aspetta che tu clicchi un incantesimo  
gioco_attivo = True   # La partita è in corso o qualcuno è stato sconfitto?
```

- **Turno Harry:** `attesa_input` è `True`. Il mouse è attivo.
- **Animazione:** Appena clicchi, `attesa_input` diventa `False`. Il gioco elabora l'effetto e mostra i flash.
- **Turno Voldemort:** Dopo l'animazione, il computer sceglie una mossa e si torna alla fase di animazione.

P
BG



Barre della Vita: L'illusione del Movimento

Perché i punti vita non calano istantaneamente? Usiamo due variabili diverse:

```
punti_vita_harry = 100          # Il valore "reale" (logica)
display_punti_vita_harry = 100 # Il valore "disegnato" (grafica)
```

In `update()`, se il valore visualizzato è maggiore di quello reale, lo facciamo scendere lentamente:

```
if display_punti_vita_harry > punti_vita_harry:
    display_punti_vita_harry -= 1 # Crea l'effetto "barra che scorre"
```

REMEMBER: Questo trucco rende il gioco molto più professionale e "smooth" (fluida) agli occhi del giocatore.

PBG Logica di Combattimento: Danno e Precisione

Ogni incantesimo ha una probabilità di successo. Come la gestiamo?

```
# random.random() genera un numero tra 0.0 e 1.0
successo = random.random() < precisione

if successo:
    punti_vita_difensore -= danno
    messaggio = f"{attaccante} colpisce con {incantesimo}!"
else:
    messaggio = f"{incantesimo} è fallito!"
```

Matematica del gioco: Se un incantesimo ha precisione 0.8 (80%), abbiamo l'80% di probabilità che il numero generato sia minore di 0.8. È il modo più semplice per implementare la "fortuna" nei videogiochi.



UI Design: La Griglia 2x2

Dobbiamo disporre 4 tasti. Invece di scrivere 4 posizioni manuali, usiamo la matematica:

```
for i in range(len(opzioni_correnti)):
    # i può essere 0, 1, 2, 3
    x = 40 + (i % 2) * 380 # Resto della divisione: alterna tra 0 e 1 (colonne)
    y = 440 + (i // 2) * 60 # Divisione intera: 0,0 per i primi due, 1,1 per gli altri (righe)

    screen.draw.rect(Rect((x, y), (350, 50)), "white")
```

PRO TIP: Questa formula è universale. Cambiando il divisore (es. % 3) puoi creare griglie di qualsiasi dimensione (3x3, 4x4, ecc.) senza riscrivere il codice.



Animazioni con `animate()` e `clock`

Per rendere il duello "magico", usiamo gli effetti speciali di Pygame Zero:






1. **Sbalzo (Bounce):** Usiamo `animate()` per far scattare lo sprite in avanti quando attacca.
2. **Flash:** Usiamo `clock.schedule_unique()` per far sparire e riapparire lo sprite velocemente quando subisce danni.

```
def flash_danno(sprite):  
    # Cambiamo l'opacità: 0 = invisibile, 255 = visibile  
    for i in range(3):  
        clock.schedule_unique(lambda: setattr(sprite, "opacity", 0), i * 0.4)  
        clock.schedule_unique(lambda: setattr(sprite, "opacity", 255), i * 0.4 + 0.2)
```



Struttura del Progetto

Assicurati che i tuoi file siano organizzati così per evitare errori `FileNotFoundException`:

-  `progetto_duello/`
-  `harry_potter.py` (Il tuo codice)
-  `spells.csv` (Il database degli incantesimi)
-  `images/` (Tutte le immagini `.png`)
- `harry.png`, `voldemort.png`, `hogwarts_bg.png` ...
-  `sounds/` (Facoltativo: file `.wav` o `.ogg` per i colpi)



Sfide per voi (Esercitazione)

Prova a modificare il codice per aggiungere queste funzionalità:

1. **Critici:** Se un incantesimo colpisce, c'è una piccola probabilità (es. 10%) che faccia il doppio dei danni.
2. **Stamina:** Ogni incantesimo consuma "Energia Magica". Se finisce, devi saltare un turno per ricaricare.
3. **Colori Dinamici:** Cambia il colore della barra della vita in base alla percentuale (Verde > 50%, Giallo < 50%, Rosso < 20%).
4. **Logica Voldemort:** Rendi Voldemort più intelligente; invece di scegliere a caso, fagli usare una cura se ha poca vita!



Conclusione

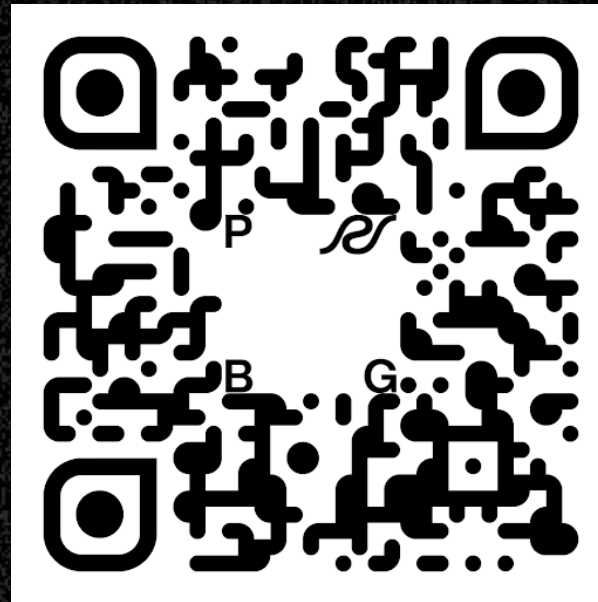
Il codice non è solo una lista di istruzioni, ma un insieme di **sistemi** (Dati, Grafica, Logica) che comunicano tra loro.

L'uso di strumenti come **Polars** ci permette di pensare come veri sviluppatori di giochi moderni, dove il contenuto (gli incantesimi) è separato dal motore di gioco.

Bacchette pronte... al lavoro!



Grazie per l'attenzione...



"C'è sempre qualcosa da imparare per migliorarci e crescere...insieme!"