Simon says…

Press blue !

Welcome to Simon Says

Hello ladies and gentlemen, today we're here to present our latest project,
welcome to Team Simon Says.
After a whole month of development, hopefully, it will have been worth the wait.

You may be wondering... what is Simon Says? (SS for short.)
SS, is a very silly and fun patterns game.
Unlike the IRL version here we cannot enslave people to make them do what we want, however this version (Go touch some grass.) is a pattern based game developed entirely in python.

Okay but... what is SS about?

In our world of SS, the program will start off giving the player a pattern to follow which gets progressively faster and longer as the game goes.

With eight different color the player has to press each color along with a “Gaynnaro” on top of each color.

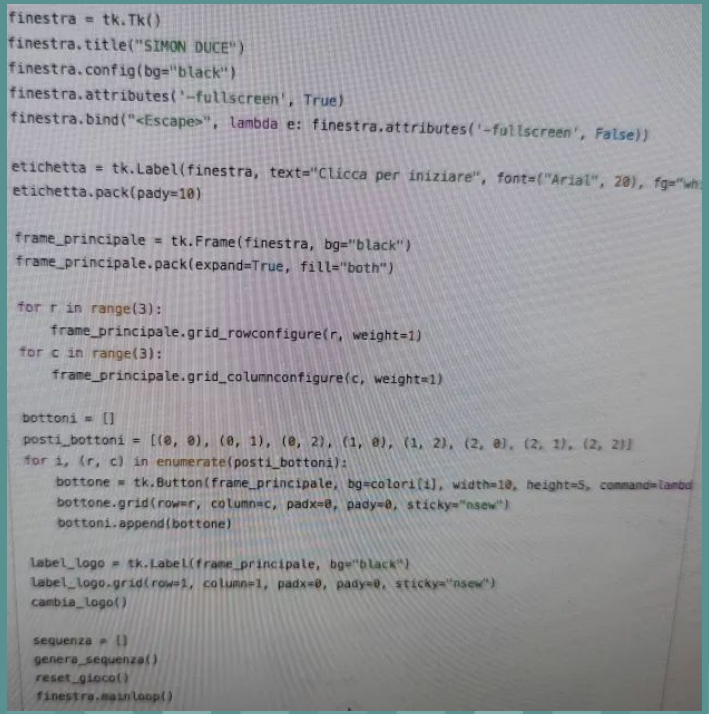
Upon the first sequence of 3 colors the game will continue the sequence and increase it, each time scoring a point.

Committing a mistake will result in losing a point, upon reaching 0, and therefore losing.

Let’s dive in the first part.

This is the biggest part of the entire program, which operates all of the functions in the game allowing us a functional creation.

The `tk.Tk()` function initializes the main window using the Tkinter library, displaying the game in fullscreen for appropriate experience, however it is also able to rescale to smaller windows if done so by the user.
A `Label` widget at the top displays the game status (e.g., "Click to start", score, or game over messages).
A central `Frame` contains a 3x3 grid (tic tac toe style) layout with 8 colored buttons placed around the center.
The center cell holds the logo, which updates each round with a random “Gaynnaro”.
Button clicks are linked to the `al_click_colore()` function, allowing the user to press a button after the game initializes a sequence.
The game initializes with:
A new random sequence (`genera_sequenza()`).
The GUI updates (`reset_gioco()`), and the game loop begins.
The window remains active with `finestra.mainloop()`, which keeps looking out for user interactions (clicks, key presses).

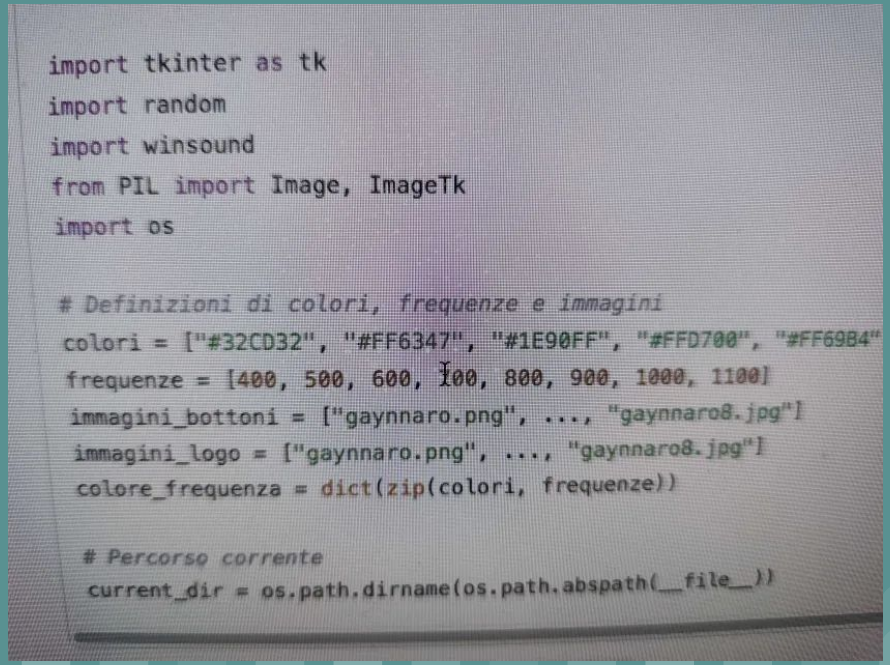


```
finestra = tk.Tk()
finestra.title("SIMON DUCE")
finestra.config(bg="black")
finestra.attributes('-fullscreen', True)
finestra.bind("<Escape>", lambda e: finestra.attributes('-fullscreen', False))

etichetta = tk.Label(finestra, text="Clicca per iniziare", font=("Arial", 20), fg="wh
etichetta.pack(pady=10)

frame_principale = tk.Frame(finestra, bg="black")
frame_principale.pack(expand=True, fill="both")

for r in range(3):
    frame_principale.grid_rowconfigure(r, weight=1)
for c in range(3):
    frame_principale.grid_columnconfigure(c, weight=1)

bottoni = []
posti_bottoni = [(0, 0), (0, 1), (0, 2), (1, 0), (1, 2), (2, 0), (2, 1), (2, 2)]
for i, (r, c) in enumerate(posti_bottoni):
    bottone = tk.Button(frame_principale, bg=colori[i], width=10, height=5, command=lambd
    bottone.grid(row=r, column=c, padx=0, pady=0, sticky="nsew")
    bottoni.append(bottone)

label_logo = tk.Label(frame_principale, bg="black")
label_logo.grid(row=1, column=1, padx=0, pady=0, sticky="nsew")
cambia_logo()

sequenza = []
genera_sequenza()
reset_gioco()
finestra.mainloop()
```

Perchè funziona?

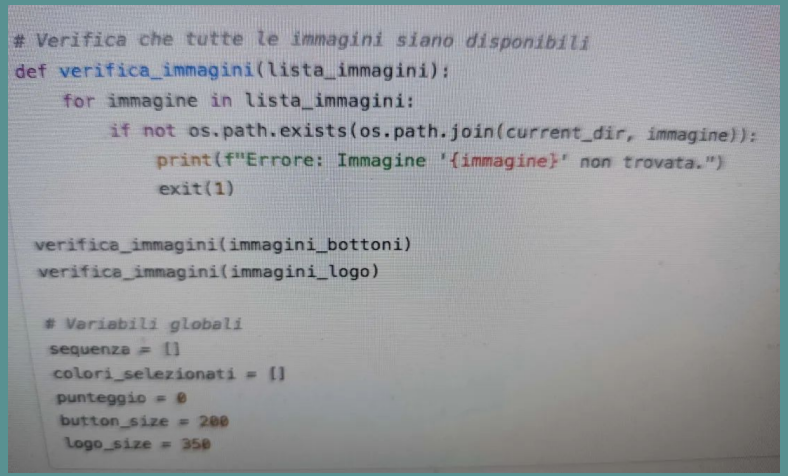
Il codice è capace di funzionare grazie a questa prima parte di codice la quale include tutte le librerie utilizzate, suoni, immagini e colori.

Tkinter - GUI
Random - Generazione valore casuale
winsound - Utilizzo di suoni windows
PIL - Gestione immagini
os - Gestione percorso file



```
import tkinter as tk
import random
import winsound
from PIL import Image, ImageTk
import os

# Definizioni di colori, frequenze e immagini
colori = ["#32CD32", "#FF6347", "#1E90FF", "#FFD700", "#FF69B4"
frequenze = [400, 500, 600, 700, 800, 900, 1000, 1100]
immagini_bottoni = ["gaynnaro.png", ..., "gaynnaro8.jpg"]
immagini_logo = ["gaynnaro.png", ..., "gaynnaro8.jpg"]
colore_frequenza = dict(zip(colori, frequenze))

# Percorso corrente
current_dir = os.path.dirname(os.path.abspath(__file__))
```

Informazioni del programma



```
# Verifica che tutte le immagini siano disponibili
def verifica_immagini(lista_immagini):
    for immagine in lista_immagini:
        if not os.path.exists(os.path.join(current_dir, immagine)):
            print(f"Errore: Immagine '{immagine}' non trovata.")
            exit(1)

verifica_immagini(immagini_bottoni)
verifica_immagini(immagini_logo)

# Variabili globali
sequenza = []
colori_selezionati = []
punteggio = 0
button_size = 200
logo_size = 350
```

Questa parte di codice prima di tutto controlla che tutte le immagini necessarie siano presenti, in caso non lo fossero apparirà un messaggio di errore.

E poi contiene anche le variabili che contengono i dati di gioco.

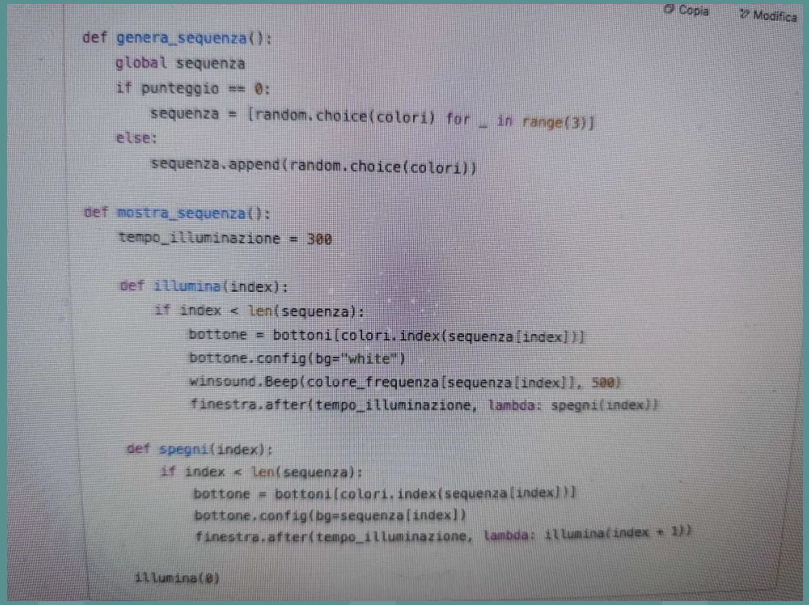
Sequenza - Array con le frequenze dei suoni.
Colori_Selezionati - Array di stringhe dei colori dei pulsanti.
punteggio - Lo score del player.
button_size - Dimensioni dei pulsanti da premere.
logo_size - Dimensione del logo.

E la sequenza?

Questa parte del codice ha il compito di generare la sequenza di colori di SS.
`genera_sequenza()`
Se il punteggio è 0, Il giocatore perde e viene generata una nuova frequenza.
Ad ogni punto guadagnato, viene aggiunto un nuovo colore alla sequenza esistente, rendendo il gioco progressivamente più difficile.

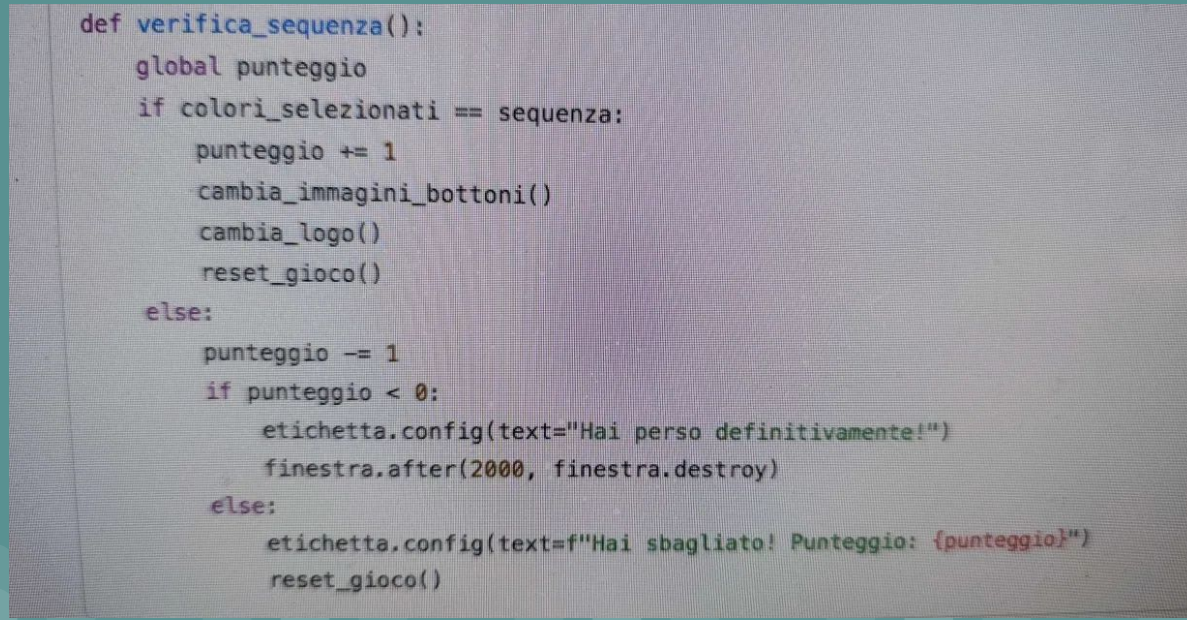
`mostra_sequenza()`
Visualizza i colori uno alla volta, con un effetto lampeggiante: Il bottone si accende (diventa bianco) e viene emesso un beep corrispondente al colore.

L'effetto si ottiene con la funzione `after()` di Tkinter, che gestisce il tempo di attesa tra un colore e l'altro.
Questa funzione mostra la sequenza che il giocatore dovrà poi ripetere cliccando i bottoni.



Come funziona la sequenza?

Questa funzione controlla se la sequenza di colori cliccata dall'utente corrisponde a quella mostrata dal gioco.
verifica_sequenza()
Se il giocatore ha cliccato i colori nell'ordine corretto Il punteggio aumenta.
Le immagini dei bottoni cambiano.
Il logo centrale viene aggiornato e parte una nuova sequenza, più lunga.
Se l'utente sbaglia Il punteggio diminuisce di 1, e se malauguratamente il punteggio scende sotto zero, il gioco termina e la finestra si chiude.
Altrimenti, il gioco riparte con una nuova sequenza e viene mostrato il punteggio aggiornato.

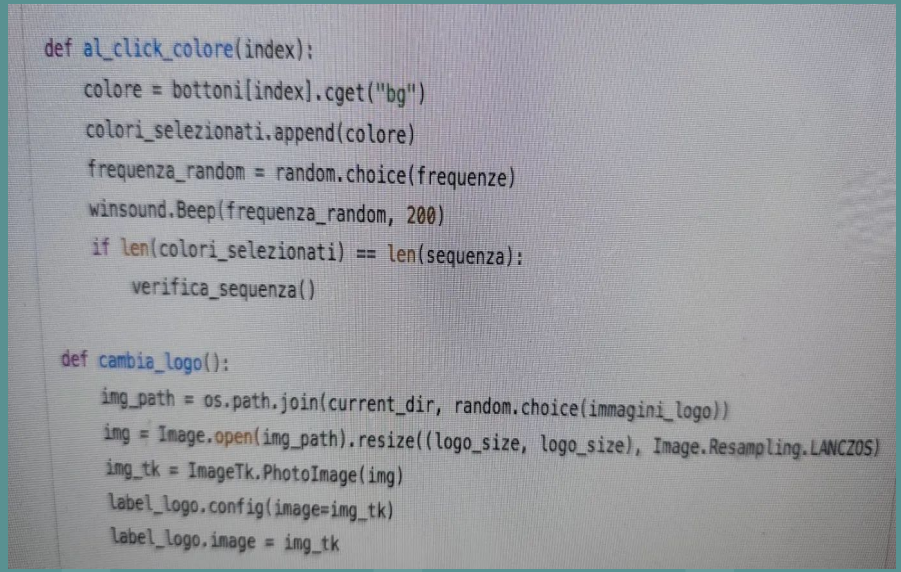


```
def verifica_sequenza():
    global punteggio
    if colori_selezionati == sequenza:
        punteggio += 1
        cambia_immagini_bottoni()
        cambia_logo()
        reset_gioco()
    else:
        punteggio -= 1
        if punteggio < 0:
            etichetta.config(text="Hai perso definitivamente!")
            finestra.after(2000, finestra.destroy)
        else:
            etichetta.config(text=f"Hai sbagliato! Punteggio: {punteggio}")
            reset_gioco()
```

Interazioni col gioco

In questa parte si gestisce l'interazione dell'utente e l'aggiornamento del logo.
al_click_colore(index)
Viene eseguita ogni volta che il giocatore clicca un bottone, Il colore selezionato viene memorizzato e poi viene emesso un suono casuale per dare feedback all'utente.
Quando sono stati cliccati tutti i colori della sequenza, si avvia il controllo con la funzione verifica_sequenza().

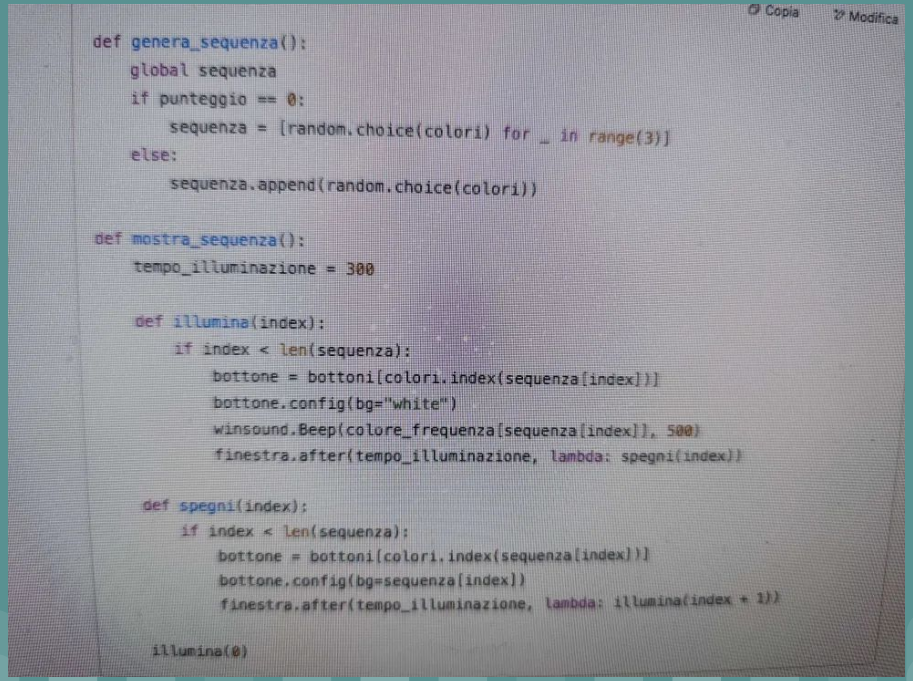
cambia_logo()
Seleziona casualmente un'immagine dalla lista.
Ridimensiona l'immagine e la mostra nel logo centrale del gioco.
L'immagine viene aggiornata ad ogni risposta corretta, rendendo l'esperienza più coinvolgente.



```
def al_click_colore(index):
    colore = bottoni[index].cget("bg")
    colori_selezionati.append(colore)
    frequenza_random = random.choice(frequenze)
    winsound.Beep(frequenza_random, 200)
    if len(colori_selezionati) == len(sequenza):
        verifica_sequenza()

def cambia_logo():
    img_path = os.path.join(current_dir, random.choice(immagini_logo))
    img = Image.open(img_path).resize((logo_size, logo_size), Image.Resampling.LANCZOS)
    img_tk = ImageTk.PhotoImage(img)
    label_logo.config(image=img_tk)
    label_logo.image = img_tk
```

Cambio immagini e reset.

Questa parte di gioco è ciò che va ad aggiornare ogni frequenza.
`cambia_immagini_bottoni()`
Prende tutte le immagini disponibili e le mescola casualmente.
Assegna un'immagine diversa a ciascun bottone.
Le immagini hanno la capacità di ridimensionarsi al bottone a cui vengono associate.
`reset_gioco()`
Svuota la lista dei colori selezionati dall'utente.
Genera una nuova sequenza (`genera_sequenza()`) e aggiorna il punteggio, poi dopo 1 secondo, mostra la nuova sequenza (`mostra_sequenza()`).
Questa funzione permette al gioco di proseguire dopo ogni risposta.



The End.

Leader - Salvato Michele
Slaves - Santaniello Gennaro
Matrone Francesco Pio