```
código base

# Paso 1: Instalar dependencias
!pip install -q scikit-learn pandas
# Paso 2: Código del juego con IA y widgets
import random
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from IPython.display import display, clear_output
import ipywidgets as widgets

# --------------------------
# ENTRENAR MODELO DE IA
# --------------------------
def entrenar_modelo_ia():
    try:
        datos = pd.read_csv("datos_partidas.csv")
        X = datos[["vida_jugador", "vida_enemigo"]]
        y = datos["accion"]
        modelo = DecisionTreeClassifier()
        modelo.fit(X, y)
        return modelo
    except:
        return None

# --------------------------
# GUARDAR DATOS DE PARTIDAS
# --------------------------
def guardar_dato_partida(vida_jugador, vida_enemigo, accion):
    fila = pd.DataFrame([[vida_jugador, vida_enemigo, accion]],
columns=["vida_jugador", "vida_enemigo", "accion"])
    try:
        archivo = pd.read_csv("datos_partidas.csv")
        archivo = pd.concat([archivo, fila], ignore_index=True)
    except:
        archivo = fila
    archivo.to_csv("datos_partidas.csv", index=False)

# --------------------------
# CLASES DE PERSONAJE
# --------------------------
class Personaje:
    def __init__(self, nombre, vida, ataque):
```

```
        self.nombre = nombre
        self.vida = vida
        self.ataque = ataque

    def atacar(self, objetivo):
        danio = random.randint(self.ataque // 2, self.ataque)
        objetivo.recibir_danio(danio)
        return f"{self.nombre} ataca a {objetivo.nombre} y le inflige
{danio} puntos de daño."

    def recibir_danio(self, danio):
        self.vida -= danio
        if self.vida <= 0:
            return f"{self.nombre} ha sido derrotado."
        return ""

class Brujo(Personaje):
    def __init__(self, nombre):
        super().__init__(nombre, vida=70, ataque=10)
        self.mana = 100

    def lanzar_hechizo(self, objetivo):
        if self.mana >= 20:
            danio_magico = random.randint(15, 25)
            objetivo.recibir_danio(danio_magico)
            self.mana -= 20
            return f"{self.nombre} lanza un hechizo a {objetivo.nombre}
y le inflige {danio_magico} puntos de daño mágico."
        else:
            return f"{self.nombre} no tiene suficiente maná para lanzar
un hechizo."

# --------------------------
# DECISIÓN DE LA IA
# --------------------------
def decision_ia_enemigo(enemigo, objetivo, modelo_ia):
    if modelo_ia:
        entrada = pd.DataFrame([[objetivo.vida, enemigo.vida]],
columns=["vida_jugador", "vida_enemigo"])
        accion = modelo_ia.predict(entrada)[0]
    else:
        accion = random.choice(["atacar", "esperar"])
```

```
    if accion == "atacar":
        resultado = enemigo.atacar(objetivo)
    else:
        resultado = f"{enemigo.nombre} decide esperar este turno."

    return accion, resultado

# --------------------------
# COMBATE EN COLAB
# --------------------------
modelo_ia = entrenar_modelo_ia()
brujo = Brujo("Gandalf")
enemigo = Personaje("Orco Maligno", vida=100, ataque=12)

salida = widgets.Output()
boton_atacar = widgets.Button(description="Atacar")
boton_hechizo = widgets.Button(description="Lanzar Hechizo")
boton_reiniciar = widgets.Button(description="Reiniciar Juego")

def mostrar_estado():
    with salida:
        print(f"\n{brujo.nombre}: {brujo.vida} HP - {brujo.mana} MP")
        print(f"{enemigo.nombre}: {enemigo.vida} HP")

def turno_jugador(accion):
    with salida:
        clear_output()
        if brujo.vida <= 0 or enemigo.vida <= 0:
            print("La partida ha terminado. Reinicia para volver a
jugar.")
            return

        if accion == "atacar":
            resultado = brujo.atacar(enemigo)
        elif accion == "hechizo":
            resultado = brujo.lanzar_hechizo(enemigo)
        else:
            resultado = "Acción no válida."

        print(f"\nTurno de {brujo.nombre}:\n{resultado}")

        if enemigo.vida <= 0:
            print(f"\n{enemigo.nombre} ha sido derrotado. ¡Ganaste!")
```

```
            return

        accion_enemigo, resultado_ia = decision_ia_enemigo(enemigo,
brujo, modelo_ia)
        print(f"\nTurno de {enemigo.nombre}:\n{resultado_ia}")
        guardar_dato_partida(brujo.vida, enemigo.vida, accion_enemigo)

        if brujo.vida <= 0:
            print(f"\n{brujo.nombre} ha sido derrotado. ¡Perdiste!")

        mostrar_estado()

def reiniciar_juego(b):
    global brujo, enemigo, modelo_ia
    brujo = Brujo("Gandalf")
    enemigo = Personaje("Orco Maligno", vida=100, ataque=12)
    modelo_ia = entrenar_modelo_ia()
    with salida:
        clear_output()
        print("--- Juego Reiniciado ---")
        mostrar_estado()

def click_atacar(b):
    turno_jugador("atacar")

def click_hechizo(b):
    turno_jugador("hechizo")

# Asignar funciones a botones
boton_atacar.on_click(click_atacar)
boton_hechizo.on_click(click_hechizo)
boton_reiniciar.on_click(reiniciar_juego)

# Mostrar controles
display(widgets.HBox([boton_atacar, boton_hechizo, boton_reiniciar]))
display(salida)

# Estado inicial
with salida:
    print("--- Bienvenido al Combate ---")
    mostrar_estado()
```