2024

Programación de Inteligencia Artificial

Profesor/a: José Carlos Del Arco Prieto

CASO PRÁCTICO PAPEL, PIEDRA O TIJERAS

16/10/2024

PEDRO MANUEL GARCÍA ÁLVAREZ | int.marisma@gmail.com | CEIA | I.E.S. La Marisma

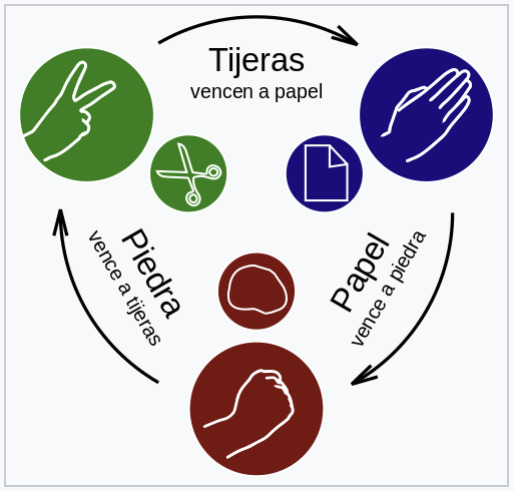
Índice

1.- Introducción

El juego de "Piedra, Papel o Tijera" es un juego de manos clásico que se utiliza para tomar decisiones o simplemente como entretenimiento. Es un juego de dos participantes en el que ambos eligen, de manera simultánea, una de las tres opciones: piedra, papel o tijera. El ganador se determina según las siguientes reglas:

1. **Piedra vence a tijera**: La piedra aplasta la tijera.
2. **Tijera vence a papel**: La tijera corta el papel.
3. **Papel vence a piedra**: El papel envuelve la piedra.
4. Si ambos jugadores eligen la misma opción, se considera un **empate**.

El objetivo del juego es que cada jugador trate de anticipar la elección del oponente para elegir una opción que le permita ganar. Aunque puede parecer completamente aleatorio, existen estrategias y patrones que algunos jugadores intentan usar para aumentar sus probabilidades de ganar.

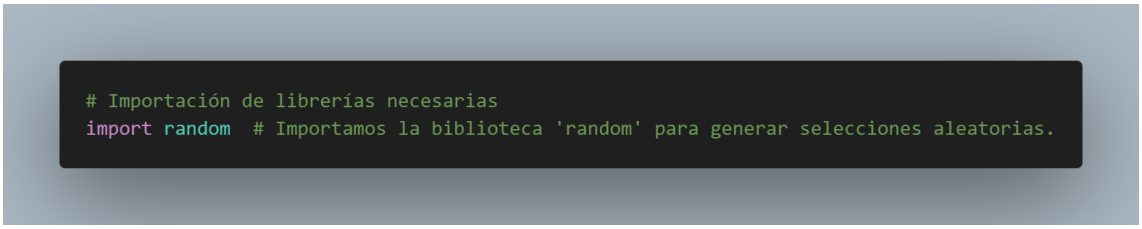


2.- Explicación de los diferentes códigos

2.1.- Código del fichero 01_PPT_Basico_Comentado.py

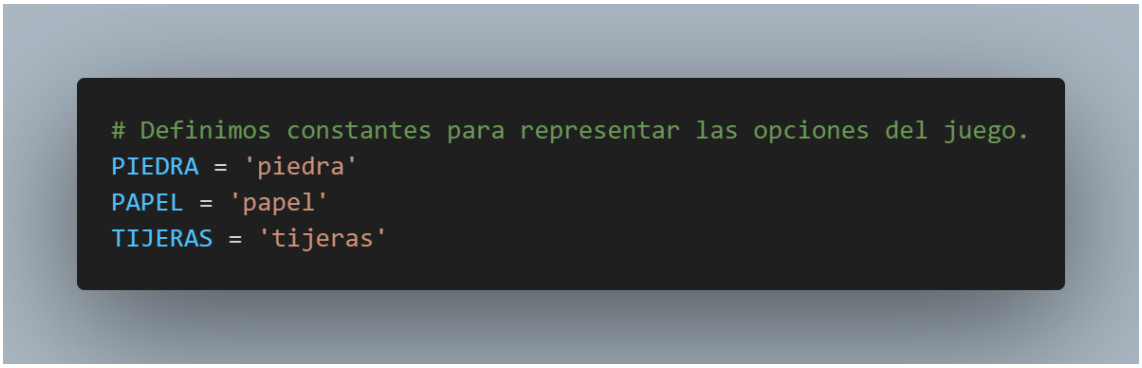
```
# Indica que este script se debe ejecutar usando el intérprete de Python 3.
#!/usr/bin/python3
```

La línea `#!/usr/bin/python3` al inicio de un script en Python indica al sistema operativo que use la versión de Python 3 ubicada en `/usr/bin/python3` para ejecutar el archivo, permitiendo que el script se ejecute directamente desde la terminal como un programa sin necesidad de escribir `python3` antes del nombre del archivo.



```python
# Importación de librerías necesarias
import random  # Importamos la biblioteca 'random' para generar selecciones aleatorias.
```

La línea `import random` permite utilizar la biblioteca `random` en Python, que incluye funciones para generar números aleatorios y seleccionar elementos de forma aleatoria, lo que es útil en juegos, simulaciones y otros algoritmos que requieren aleatoriedad.

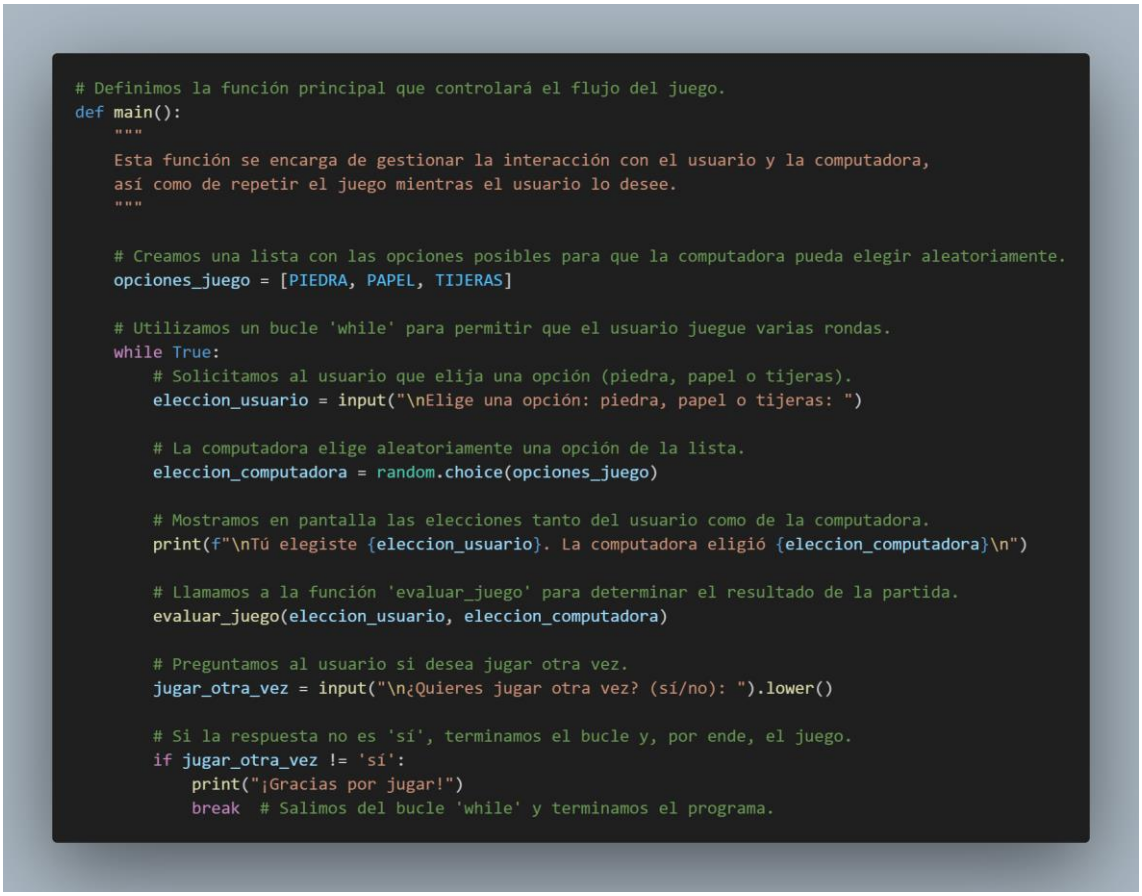


```python
# Definimos constantes para representar las opciones del juego.
PIEDRA = 'piedra'
PAPEL = 'papel'
TIJERAS = 'tijeras'
```

El código define tres constantes (`PIEDRA`, `PAPEL`, `TIJERAS`) asignándoles los valores `'piedra'`, `'papel'` y `'tijeras'`, respectivamente, para utilizarlas como opciones en el juego "Piedra, Papel o Tijera" de forma más legible y clara.

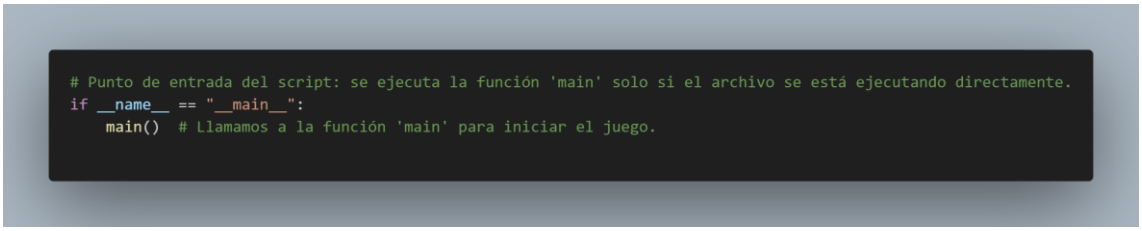
```python
# Definimos una función que evaluará el resultado del juego según las selecciones del usuario y la computadora.
def evaluar_juego(eleccion_usuario, eleccion_computadora):
    """
    Esta función determina el resultado del juego basado en las elecciones del usuario y la computadora.

    Parámetros:
    - eleccion_usuario: la elección del usuario ('piedra', 'papel' o 'tijeras').
    - eleccion_computadora: la elección de la computadora ('piedra', 'papel' o 'tijeras').
    """

    # Caso 1: Si el usuario y la computadora eligen la misma opción.
    if eleccion_usuario == eleccion_computadora:
        # Si ambas selecciones son iguales, el juego es un empate.
        print(f"El usuario y la computadora eligieron {eleccion_usuario}. ¡Empate!")

    # Caso 2: Si el usuario elige "piedra".
    elif eleccion_usuario == PIEDRA:
        # Evaluamos la elección de la computadora.
        if eleccion_computadora == TIJERAS:
            # La piedra aplasta las tijeras, por lo tanto, el usuario gana.
            print("Piedra aplasta tijeras. ¡Ganaste!")
        else:
            # Si la computadora eligió "papel", entonces el papel envuelve la piedra,
            # lo que significa que el usuario pierde.
            print("Papel envuelve piedra. ¡Perdiste!")

    # Caso 3: Si el usuario elige "papel".
    elif eleccion_usuario == PAPEL:
        # Evaluamos la elección de la computadora.
        if eleccion_computadora == PIEDRA:
            # El papel envuelve la piedra, así que el usuario gana.
            print("Papel envuelve piedra. ¡Ganaste!")
        else:
            # Si la computadora eligió "tijeras", las tijeras cortan el papel,
            # lo que significa que el usuario pierde.
            print("Tijeras cortan papel. ¡Perdiste!")

    # Caso 4: Si el usuario elige "tijeras".
    elif eleccion_usuario == TIJERAS:
        # Evaluamos la elección de la computadora.
        if eleccion_computadora == PAPEL:
            # Las tijeras cortan el papel, por lo tanto, el usuario gana.
            print("Tijeras cortan papel. ¡Ganaste!")
        else:
            # Si la computadora eligió "piedra", la piedra aplasta las tijeras,
            # lo que significa que el usuario pierde.
            print("Piedra aplasta tijeras. ¡Perdiste!")
```

La función `evaluar_juego` compara la elección del usuario con la de la computadora y, según las reglas de "Piedra, Papel o Tijeras", determina si hay un empate, victoria o derrota, imprimiendo el resultado correspondiente.



```python
# Definimos la función principal que controlará el flujo del juego.
def main():
    """
    Esta función se encarga de gestionar la interacción con el usuario y la computadora,
    así como de repetir el juego mientras el usuario lo desee.
    """

    # Creamos una lista con las opciones posibles para que la computadora pueda elegir aleatoriamente.
    opciones_juego = [PIEDRA, PAPEL, TIJERAS]

    # Utilizamos un bucle 'while' para permitir que el usuario juegue varias rondas.
    while True:
        # Solicitamos al usuario que elija una opción (piedra, papel o tijeras).
        eleccion_usuario = input("\nElige una opción: piedra, papel o tijeras: ")

        # La computadora elige aleatoriamente una opción de la lista.
        eleccion_computadora = random.choice(opciones_juego)

        # Mostramos en pantalla las elecciones tanto del usuario como de la computadora.
        print(f"\nTú elegiste {eleccion_usuario}. La computadora eligió {eleccion_computadora}\n")

        # Llamamos a la función 'evaluar_juego' para determinar el resultado de la partida.
        evaluar_juego(eleccion_usuario, eleccion_computadora)

        # Preguntamos al usuario si desea jugar otra vez.
        jugar_otra_vez = input("\n¿Quieres jugar otra vez? (sí/no): ").lower()

        # Si la respuesta no es 'sí', terminamos el bucle y, por ende, el juego.
        if jugar_otra_vez != 'sí':
            print("¡Gracias por jugar!")
            break  # Salimos del bucle 'while' y terminamos el programa.
```

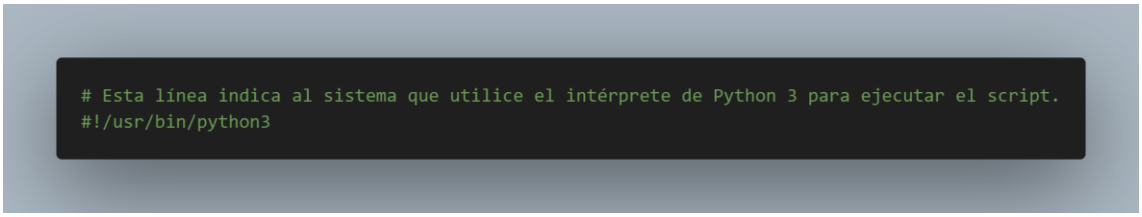
La función `main` gestiona el flujo del juego "Piedra, Papel o Tijeras", permitiendo que el usuario juegue múltiples rondas con la computadora, eligiendo opciones aleatorias, mostrando el resultado y preguntando si desea continuar después de cada partida.



```python
# Punto de entrada del script: se ejecuta la función 'main' solo si el archivo se está ejecutando directamente.
if __name__ == "__main__":
    main()  # Llamamos a la función 'main' para iniciar el juego.
```

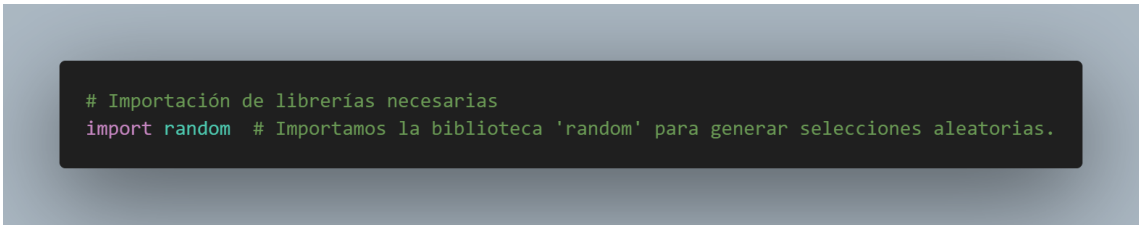
La línea `if __name__ == "__main__": main()` asegura que la función `main()` se ejecute solo cuando el script se ejecuta directamente, y no si es importado como un módulo en otro programa.

2.2.- Código del fichero 02_PPT_Control_Error_Comentado.py



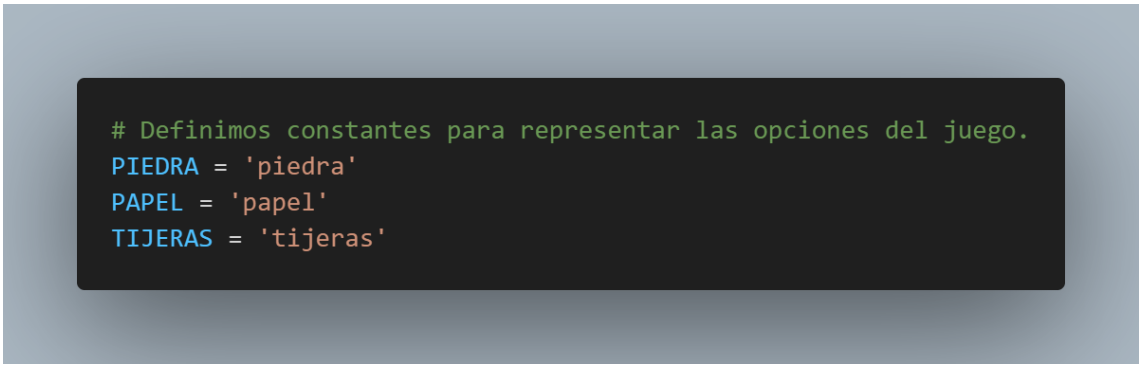
```python
# Esta línea indica al sistema que utilice el intérprete de Python 3 para ejecutar el script.
#!/usr/bin/python3
```

La línea `#!/usr/bin/python3` al inicio de un script en Python indica al sistema operativo que use la versión de Python 3 ubicada en `/usr/bin/python3` para ejecutar el archivo, permitiendo que el script se ejecute directamente desde la terminal como un programa sin necesidad de escribir `python3` antes del nombre del archivo.



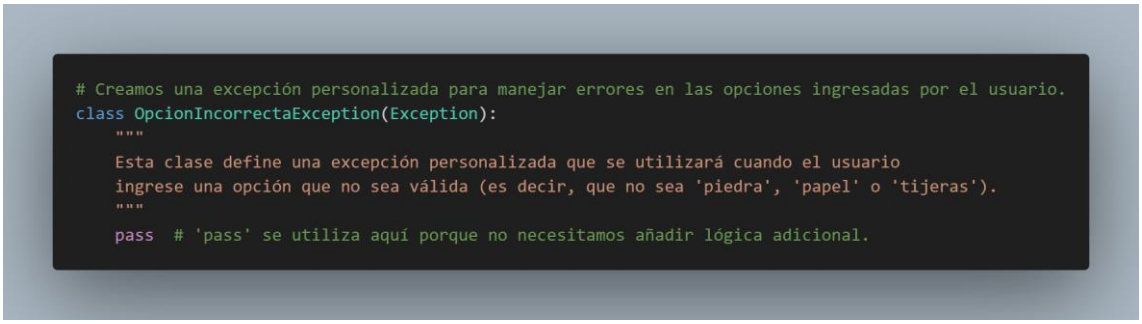
```python
# Importación de librerías necesarias
import random  # Importamos la biblioteca 'random' para generar selecciones aleatorias.
```

La línea `import random` permite utilizar la biblioteca `random` en Python, que incluye funciones para generar números aleatorios y seleccionar elementos de forma aleatoria, lo que es útil en juegos, simulaciones y otros algoritmos que requieren aleatoriedad.



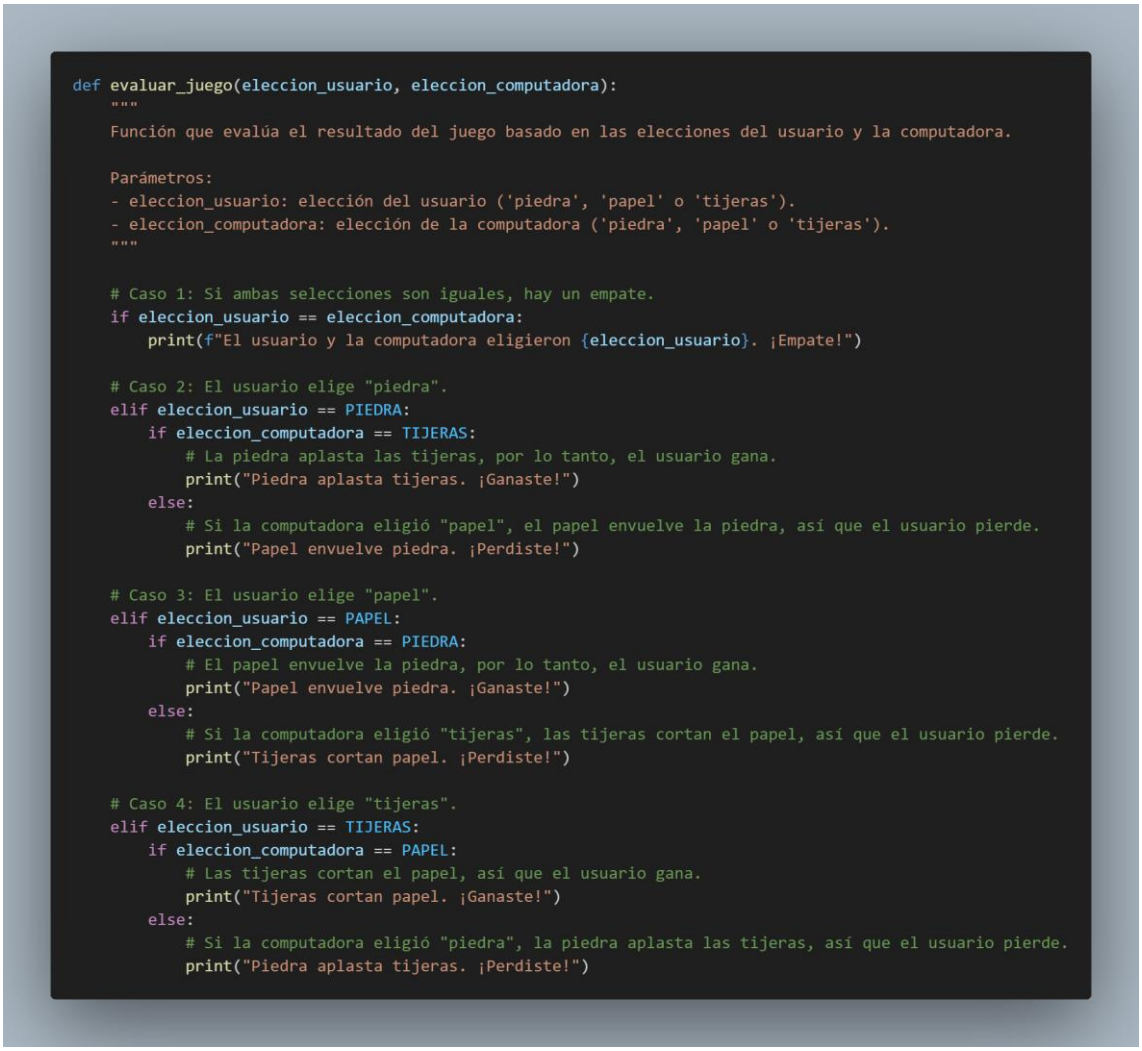
```python
# Definimos constantes para representar las opciones del juego.
PIEDRA = 'piedra'
PAPEL = 'papel'
TIJERAS = 'tijeras'
```

El código define tres constantes (`PIEDRA`, `PAPEL`, `TIJERAS`) asignándoles los valores `'piedra'`, `'papel'` y `'tijeras'`, respectivamente, para utilizarlas como opciones en el juego "Piedra, Papel o Tijera" de forma más legible y clara.



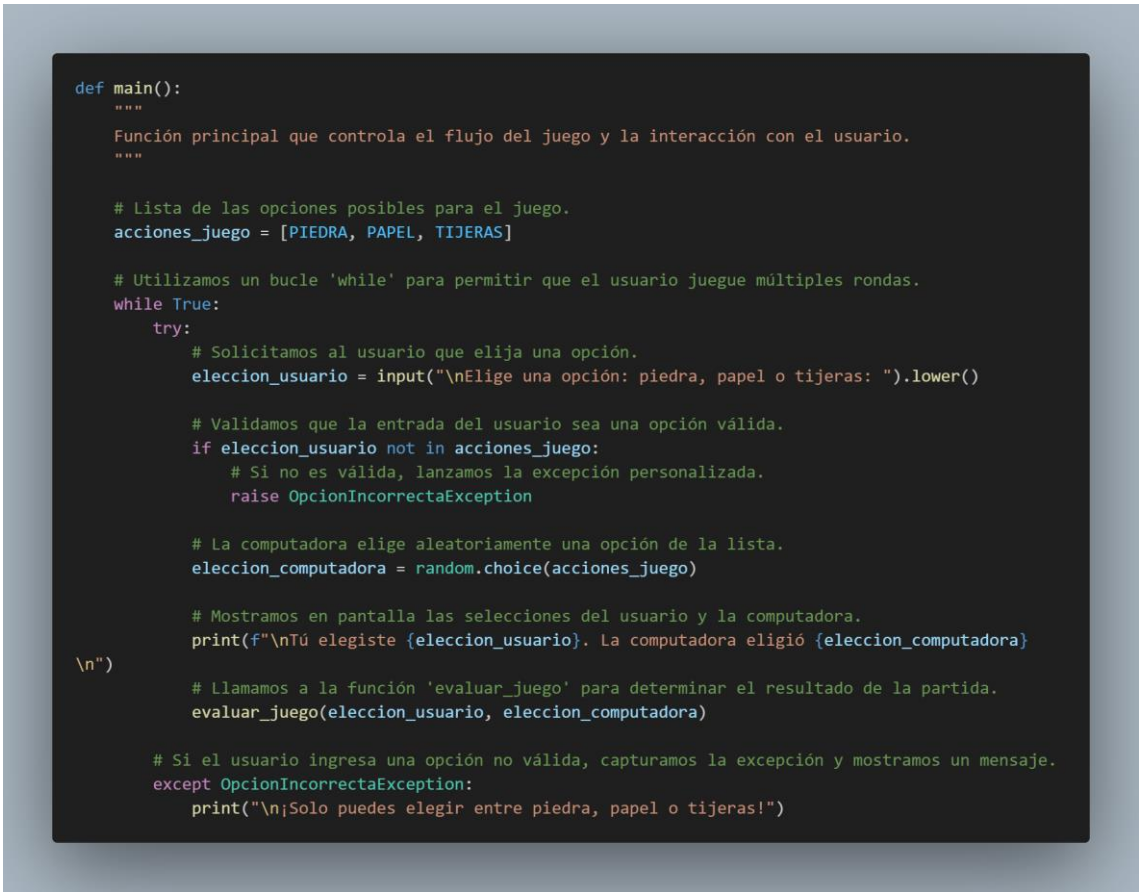
```python
# Creamos una excepción personalizada para manejar errores en las opciones ingresadas por el usuario.
class OpcionIncorrectaException(Exception):
    """
    Esta clase define una excepción personalizada que se utilizará cuando el usuario
    ingrese una opción que no sea válida (es decir, que no sea 'piedra', 'papel' o 'tijeras').
    """
    pass  # 'pass' se utiliza aquí porque no necesitamos añadir lógica adicional.
```

El código define una clase llamada `OpcionIncorrectaException`, que hereda de `Exception` y se utiliza para crear una excepción personalizada que puede lanzarse cuando se selecciona una opción no válida en el juego.



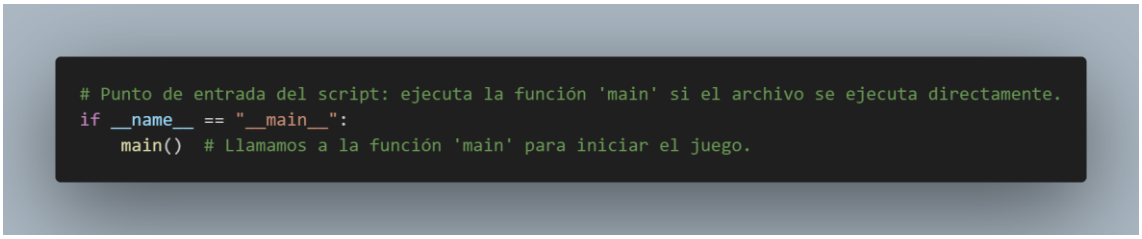
```python
def evaluar_juego(eleccion_usuario, eleccion_computadora):
    """
    Función que evalúa el resultado del juego basado en las elecciones del usuario y la computadora.

    Parámetros:
    - eleccion_usuario: elección del usuario ('piedra', 'papel' o 'tijeras').
    - eleccion_computadora: elección de la computadora ('piedra', 'papel' o 'tijeras').
    """

    # Caso 1: Si ambas selecciones son iguales, hay un empate.
    if eleccion_usuario == eleccion_computadora:
        print(f"El usuario y la computadora eligieron {eleccion_usuario}. ¡Empate!")

    # Caso 2: El usuario elige "piedra".
    elif eleccion_usuario == PIEDRA:
        if eleccion_computadora == TIJERAS:
            # La piedra aplasta las tijeras, por lo tanto, el usuario gana.
            print("Piedra aplasta tijeras. ¡Ganaste!")
        else:
            # Si la computadora eligió "papel", el papel envuelve la piedra, así que el usuario pierde.
            print("Papel envuelve piedra. ¡Perdiste!")

    # Caso 3: El usuario elige "papel".
    elif eleccion_usuario == PAPEL:
        if eleccion_computadora == PIEDRA:
            # El papel envuelve la piedra, por lo tanto, el usuario gana.
            print("Papel envuelve piedra. ¡Ganaste!")
        else:
            # Si la computadora eligió "tijeras", las tijeras cortan el papel, así que el usuario pierde.
            print("Tijeras cortan papel. ¡Perdiste!")

    # Caso 4: El usuario elige "tijeras".
    elif eleccion_usuario == TIJERAS:
        if eleccion_computadora == PAPEL:
            # Las tijeras cortan el papel, así que el usuario gana.
            print("Tijeras cortan papel. ¡Ganaste!")
        else:
            # Si la computadora eligió "piedra", la piedra aplasta las tijeras, así que el usuario pierde.
            print("Piedra aplasta tijeras. ¡Perdiste!")
```

La función `evaluar_juego` compara la elección del usuario con la de la computadora y, según las reglas de "Piedra, Papel o Tijeras", determina si hay un empate, victoria o derrota, imprimiendo el resultado correspondiente.



```python
def main():
    """
    Función principal que controla el flujo del juego y la interacción con el usuario.
    """

    # Lista de las opciones posibles para el juego.
    acciones_juego = [PIEDRA, PAPEL, TIJERAS]

    # Utilizamos un bucle 'while' para permitir que el usuario juegue múltiples rondas.
    while True:
        try:
            # Solicitamos al usuario que elija una opción.
            eleccion_usuario = input("\nElige una opción: piedra, papel o tijeras: ").lower()

            # Validamos que la entrada del usuario sea una opción válida.
            if eleccion_usuario not in acciones_juego:
                # Si no es válida, lanzamos la excepción personalizada.
                raise OpcionIncorrectaException

            # La computadora elige aleatoriamente una opción de la lista.
            eleccion_computadora = random.choice(acciones_juego)

            # Mostramos en pantalla las selecciones del usuario y la computadora.
            print(f"\nTú elegiste {eleccion_usuario}. La computadora eligió {eleccion_computadora}
\n")
            # Llamamos a la función 'evaluar_juego' para determinar el resultado de la partida.
            evaluar_juego(eleccion_usuario, eleccion_computadora)

        # Si el usuario ingresa una opción no válida, capturamos la excepción y mostramos un mensaje.
        except OpcionIncorrectaException:
            print("\n¡Solo puedes elegir entre piedra, papel o tijeras!")
```

La función `main` ejecuta un bucle que pide al usuario una elección entre "piedra", "papel" o "tijeras", verificando si la entrada es válida; en caso contrario, lanza una excepción personalizada (`OpcionIncorrectaException`) para notificar al usuario y vuelve a pedir la entrada, mientras que la computadora elige de forma aleatoria y el resultado se evalúa usando la función `evaluar_juego`.



```python
# Punto de entrada del script: ejecuta la función 'main' si el archivo se ejecuta directamente.
if __name__ == "__main__":
    main()  # Llamamos a la función 'main' para iniciar el juego.
```

La línea `if __name__ == "__main__": main()` asegura que la función `main()` se ejecute solo cuando el script se ejecuta directamente, y no si es importado como un módulo en otro programa.

2.3.- Código del fichero 03_PPT_Codigo_Limpio_Comentado.py

```
# Indica que este script se debe ejecutar usando el intérprete de Python 3.
#!/usr/bin/python3
```

La línea `#!/usr/bin/python3` al inicio de un script en Python indica al sistema operativo que use la versión de Python 3 ubicada en `/usr/bin/python3` para ejecutar el archivo, permitiendo que el script se ejecute directamente desde la terminal como un programa sin necesidad de escribir `python3` antes del nombre del archivo.

```
# Importación de librerías necesarias
import random            # Importamos la biblioteca 'random' para generar selecciones aleatorias.
from enum import IntEnum  # Importamos IntEnum para definir enumeraciones de forma eficiente
```

El código importa la biblioteca `random`, que se usa para generar números aleatorios, y `IntEnum` desde el módulo `enum`, que permite crear enumeraciones basadas en enteros para definir valores constantes de manera más estructurada.

```
# Definimos una clase enumerada para representar las posibles acciones del juego
class AccionJuego(IntEnum):
    Piedra = 0
    Papel = 1
    Tijeras = 2
```

La clase `AccionJuego` define una enumeración con tres opciones: `Piedra`, `Papel` y `Tijeras`, asignándoles los valores enteros 0, 1 y 2, respectivamente, utilizando la clase `IntEnum` para permitir comparaciones y operaciones con enteros.

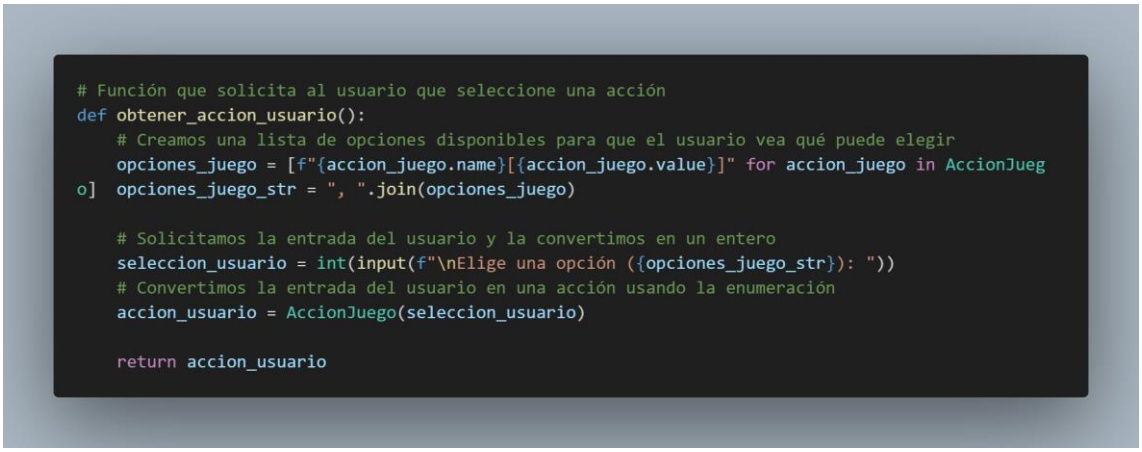
```python
# Función que evalúa el resultado del juego entre el usuario y la computadora
def evaluar_juego(accion_usuario, accion_computadora):
    # Caso en el que ambas selecciones son iguales: empate
    if accion_usuario == accion_computadora:
        print(f"El usuario y la computadora eligieron {accion_usuario.name}. ¡Empat
e!")
    # El usuario eligió Piedra
    elif accion_usuario == AccionJuego.Piedra:
        # Si la computadora eligió Tijeras, el usuario gana
        if accion_computadora == AccionJuego.Tijeras:
            print("Piedra aplasta tijeras. ¡Ganaste!")
        # Si la computadora eligió Papel, el usuario pierde
        else:
            print("Papel envuelve piedra. ¡Perdiste!")

    # El usuario eligió Papel
    elif accion_usuario == AccionJuego.Papel:
        # Si la computadora eligió Piedra, el usuario gana
        if accion_computadora == AccionJuego.Piedra:
            print("Papel envuelve piedra. ¡Ganaste!")
        # Si la computadora eligió Tijeras, el usuario pierde
        else:
            print("Tijeras cortan papel. ¡Perdiste!")

    # El usuario eligió Tijeras
    elif accion_usuario == AccionJuego.Tijeras:
        # Si la computadora eligió Piedra, el usuario pierde
        if accion_computadora == AccionJuego.Piedra:
            print("Piedra aplasta tijeras. ¡Perdiste!")
        # Si la computadora eligió Papel, el usuario gana
        else:
            print("Tijeras cortan papel. ¡Ganaste!")
```

La función `evaluar_juego` compara las elecciones del usuario y la computadora, utilizando la enumeración `AccionJuego`, para determinar el resultado del juego "Piedra, Papel o Tijeras" y mostrar si hay un empate, victoria o derrota según las reglas del juego.

```python
# Función que genera y devuelve la elección aleatoria de la computadora
def obtener_accion_computadora():
    # Se genera un número aleatorio entre 0 y 2 (correspondiente a las accione
s)  seleccion_computadora = random.randint(0, len(AccionJuego) - 1)
    # Se convierte el número en una acción usando la enumeración
    accion_computadora = AccionJuego(seleccion_computadora)
    print(f"La computadora eligió {accion_computadora.name}.")

    return accion_computadora
```

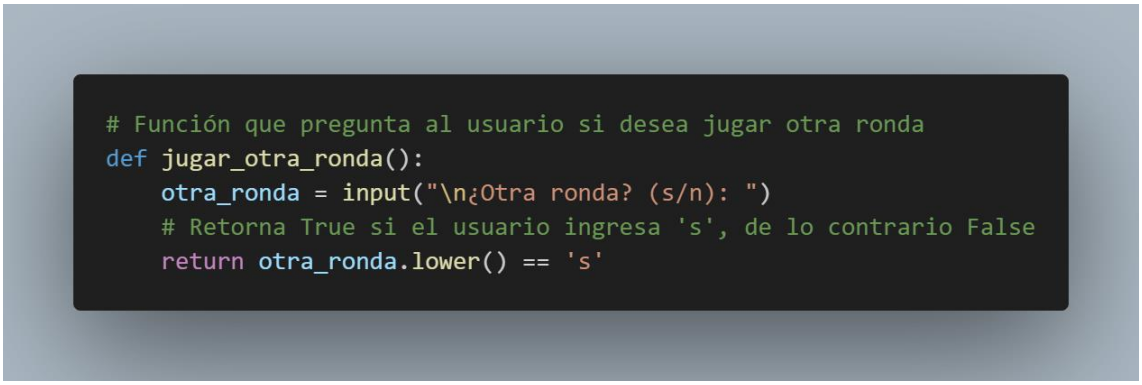
La función `obtener_accion_computadora` genera una selección aleatoria de la computadora entre las opciones definidas en la

enumeración `AccionJuego`, imprime la elección de la computadora y la devuelve como un valor de esa enumeración.



```
# Función que solicita al usuario que seleccione una acción
def obtener_accion_usuario():
    # Creamos una lista de opciones disponibles para que el usuario vea qué puede elegir
    opciones_juego = [f"{accion_juego.name}[{accion_juego.value}]" for accion_juego in AccionJueg
o]  opciones_juego_str = ", ".join(opciones_juego)

    # Solicitamos la entrada del usuario y la convertimos en un entero
    seleccion_usuario = int(input(f"\nElige una opción ({opciones_juego_str}): "))
    # Convertimos la entrada del usuario en una acción usando la enumeración
    accion_usuario = AccionJuego(seleccion_usuario)

    return accion_usuario
```

La función `obtener_accion_usuario` presenta al usuario una lista de opciones disponibles basadas en la enumeración `AccionJuego`, le solicita que ingrese una opción, y devuelve la acción correspondiente seleccionada por el usuario como un valor de esa enumeración.



```
# Función que pregunta al usuario si desea jugar otra ronda
def jugar_otra_ronda():
    otra_ronda = input("\n¿Otra ronda? (s/n): ")
    # Retorna True si el usuario ingresa 's', de lo contrario False
    return otra_ronda.lower() == 's'
```

La función `jugar_otra_ronda` pregunta al usuario si desea jugar otra ronda, y devuelve `True` si la respuesta es 's' (sí) y `False` si es cualquier otra respuesta.

```python
# Función principal que controla el flujo del juego
def main():
    while True:  # Bucle infinito para permitir múltiples rondas
        try:
            # Obtenemos la elección del usuario
            accion_usuario = obtener_accion_usuario()
        except ValueError as e:  # Si el usuario ingresa un valor no válido
            rango_str = f"[0, {len(AccionJuego) - 1}]"
            print(f"Selección no válida. ¡Elige una opción en el rango {rango_st
r}!")        continue  # Volver al inicio del bucle y solicitar nuevamente

        # Obtenemos la elección de la computadora
        accion_computadora = obtener_accion_computadora()
        # Evaluamos quién ganó la ronda
        evaluar_juego(accion_usuario, accion_computadora)

        # Preguntamos si el usuario quiere jugar otra ronda
        if not jugar_otra_ronda():
            break  # Si no quiere, salimos del bucle y terminamos el juego
```

La función `main` ejecuta un bucle donde el usuario elige una acción, se maneja cualquier error de entrada, la computadora selecciona una acción aleatoria, se evalúa el resultado del juego, y luego se pregunta al usuario si desea jugar otra ronda, repitiendo el proceso hasta que el usuario decida no continuar.

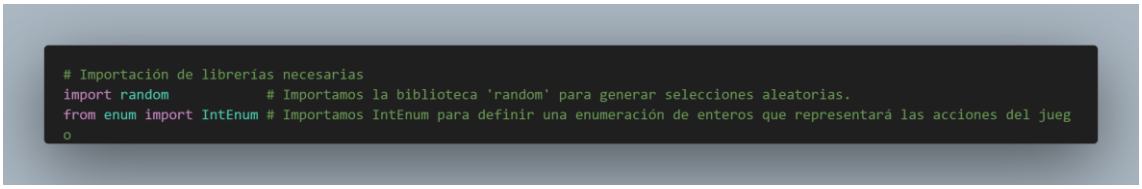
```python
# Verificamos si este script se está ejecutando como programa principal
if __name__ == "__main__":
    main()  # Llamamos a la función principal
```

La línea `if __name__ == "__main__": main()` asegura que la función `main()` se ejecute solo cuando el script se ejecuta directamente, y no si es importado como un módulo en otro programa.

2.4.- Código del fichero 04_PPT_IA_Básica_Comentado.py

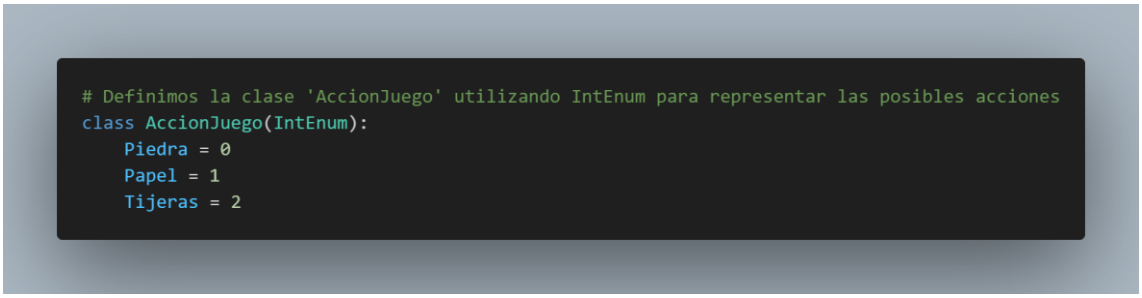
```python
# Indica que este script se debe ejecutar usando el intérprete de Python 3.
#!/usr/bin/python3
```

La línea `#!/usr/bin/python3` al inicio de un script en Python indica al sistema operativo que use la versión de Python 3 ubicada en `/usr/bin/python3` para ejecutar el archivo, permitiendo que el script se ejecute directamente desde la terminal como un programa sin necesidad de escribir `python3` antes del nombre del archivo.



```
# Importación de librerías necesarias
import random           # Importamos la biblioteca 'random' para generar selecciones aleatorias.
from enum import IntEnum # Importamos IntEnum para definir una enumeración de enteros que representará las acciones del jueg
o
```

El código importa la biblioteca `random`, que se usa para generar números aleatorios, y `IntEnum` desde el módulo `enum`, que permite crear enumeraciones basadas en enteros para definir valores constantes de manera más estructurada.



```
# Definimos la clase 'AccionJuego' utilizando IntEnum para representar las posibles acciones
class AccionJuego(IntEnum):
    Piedra = 0
    Papel = 1
    Tijeras = 2
```

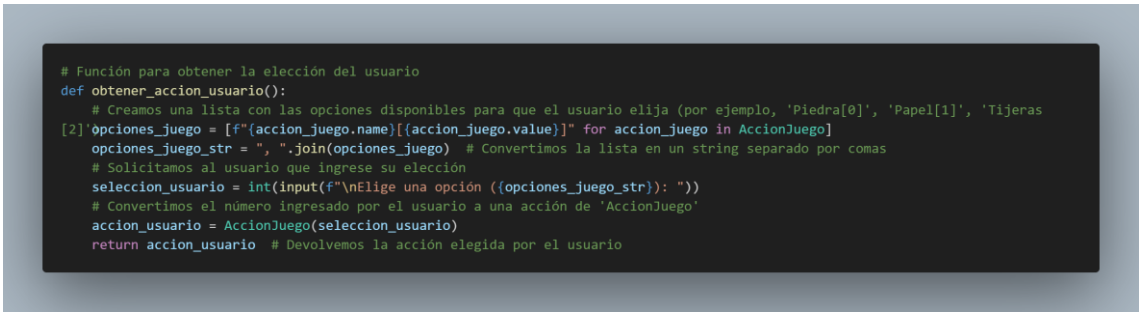
La clase `AccionJuego` define una enumeración con tres opciones: `Piedra`, `Papel` y `Tijeras`, asignándoles los valores enteros 0, 1 y 2, respectivamente, utilizando la clase `IntEnum` para permitir comparaciones y operaciones con enteros.

```python
# Función que evalúa el resultado del juego entre el usuario y la computadora
def evaluar_juego(eleccion_usuario, eleccion_computadora):
    # Caso de empate si las elecciones del usuario y la computadora son iguales
    if eleccion_usuario == eleccion_computadora:
        print(f"El usuario y la computadora eligieron {eleccion_usuario.name}. ¡Empat
e!")
    # Si el usuario eligió 'Piedra'
    elif eleccion_usuario == AccionJuego.Piedra:
        if eleccion_computadora == AccionJuego.Tijeras:
            print("Piedra aplasta tijeras. ¡Ganaste!")  # Piedra gana a tijeras
        else:
            print("Papel envuelve piedra. ¡Perdiste!")  # Piedra pierde ante papel

    # Si el usuario eligió 'Papel'
    elif eleccion_usuario == AccionJuego.Papel:
        if eleccion_computadora == AccionJuego.Piedra:
            print("Papel envuelve piedra. ¡Ganaste!")  # Papel gana a piedra
        else:
            print("Tijeras cortan papel. ¡Perdiste!")  # Papel pierde ante tijeras

    # Si el usuario eligió 'Tijeras'
    elif eleccion_usuario == AccionJuego.Tijeras:
        if eleccion_computadora == AccionJuego.Piedra:
            print("Piedra aplasta tijeras. ¡Perdiste!")  # Tijeras pierde ante piedra
        else:
            print("Tijeras cortan papel. ¡Ganaste!")  # Tijeras gana a papel
```

La función `evaluar_juego` compara las elecciones del usuario y la computadora, utilizando la enumeración `AccionJuego`, para determinar el resultado del juego "Piedra, Papel o Tijeras" y mostrar si hay un empate, victoria o derrota según las reglas del juego.

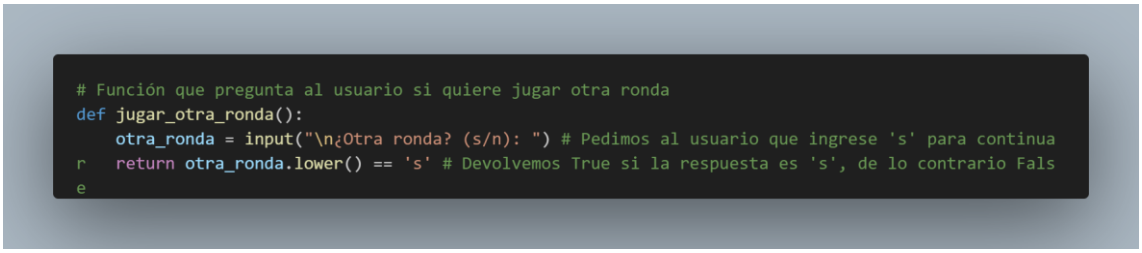
```python
# Función que obtiene la elección de la computadora de forma aleatoria
def obtener_accion_computadora():
    # Generamos un número aleatorio entre 0 y 2 (correspondiente a las acciones del juego)
    seleccion_computadora = random.randint(0, len(AccionJuego) - 1)
    # Convertimos el número a una acción utilizando la clase 'AccionJuego'
    accion_computadora = AccionJuego(seleccion_computadora)
    # Mostramos la elección de la computadora
    print(f"La computadora eligió {accion_computadora.name}.")
    return accion_computadora  # Devolvemos la acción elegida por la computadora
```

La función `obtener_accion_computadora` genera una selección aleatoria de la computadora entre las opciones definidas en la enumeración `AccionJuego`, imprime la elección de la computadora y la devuelve como un valor de esa enumeración.



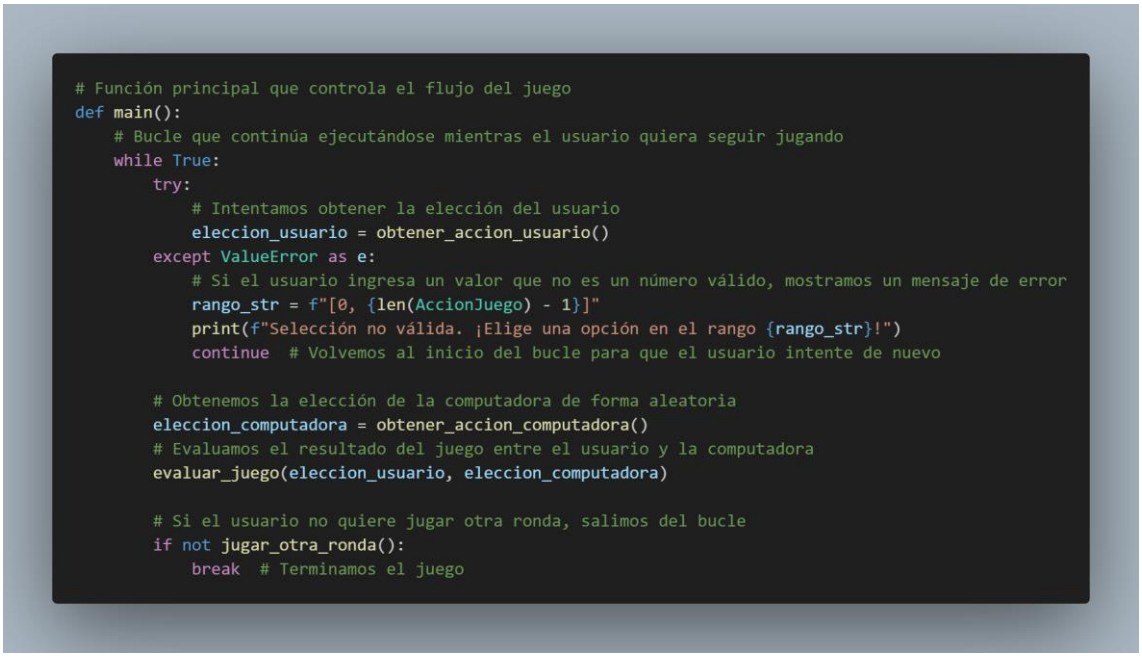
```python
# Función para obtener la elección del usuario
def obtener_accion_usuario():
    # Creamos una lista con las opciones disponibles para que el usuario elija (por ejemplo, 'Piedra[0]', 'Papel[1]', 'Tijeras
[2]'opciones_juego = [f"{accion_juego.name}[{accion_juego.value}]" for accion_juego in AccionJuego]
    opciones_juego_str = ", ".join(opciones_juego)  # Convertimos la lista en un string separado por comas
    # Solicitamos al usuario que ingrese su elección
    seleccion_usuario = int(input(f"\nElige una opción ({opciones_juego_str}): "))
    # Convertimos el número ingresado por el usuario a una acción de 'AccionJuego'
    accion_usuario = AccionJuego(seleccion_usuario)
    return accion_usuario  # Devolvemos la acción elegida por el usuario
```

La función `obtener_accion_usuario` presenta al usuario una lista de opciones disponibles basadas en la enumeración `AccionJuego`, le solicita que ingrese una opción, y devuelve la acción correspondiente seleccionada por el usuario como un valor de esa enumeración.



```python
# Función que pregunta al usuario si quiere jugar otra ronda
def jugar_otra_ronda():
    otra_ronda = input("\n¿Otra ronda? (s/n): ") # Pedimos al usuario que ingrese 's' para continua
r   return otra_ronda.lower() == 's' # Devolvemos True si la respuesta es 's', de lo contrario Fals
e
```

La función `jugar_otra_ronda` pregunta al usuario si desea jugar otra ronda, y devuelve `True` si la respuesta es 's' (sí) y `False` si es cualquier otra respuesta.



```python
# Función principal que controla el flujo del juego
def main():
    # Bucle que continúa ejecutándose mientras el usuario quiera seguir jugando
    while True:
        try:
            # Intentamos obtener la elección del usuario
            eleccion_usuario = obtener_accion_usuario()
        except ValueError as e:
            # Si el usuario ingresa un valor que no es un número válido, mostramos un mensaje de error
            rango_str = f"[0, {len(AccionJuego) - 1}]"
            print(f"Selección no válida. ¡Elige una opción en el rango {rango_str}!")
            continue  # Volvemos al inicio del bucle para que el usuario intente de nuevo

        # Obtenemos la elección de la computadora de forma aleatoria
        eleccion_computadora = obtener_accion_computadora()
        # Evaluamos el resultado del juego entre el usuario y la computadora
        evaluar_juego(eleccion_usuario, eleccion_computadora)

        # Si el usuario no quiere jugar otra ronda, salimos del bucle
        if not jugar_otra_ronda():
            break  # Terminamos el juego
```

La función `main` ejecuta un bucle donde el usuario elige una acción, se captura y maneja cualquier error de entrada, la computadora selecciona una opción aleatoria, se evalúa el resultado del juego, y luego se pregunta al usuario si desea jugar otra ronda, repitiendo el proceso hasta que el usuario decida finalizar.

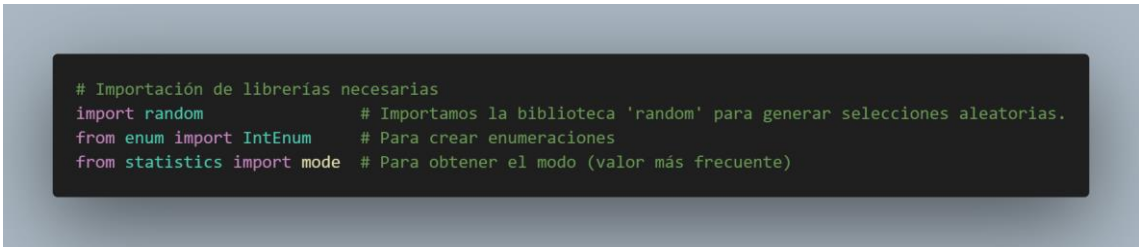
```
# Verificamos si el script se está ejecutando como el programa principal
if __name__ == "__main__":
    main()  # Llamamos a la función principal para iniciar el juego
```

La línea `if __name__ == "__main__": main()` asegura que la función `main()` se ejecute solo cuando el script se ejecuta directamente, y no si es importado como un módulo en otro programa.

2.5.- Código del fichero 05_RPS_More_AI.py

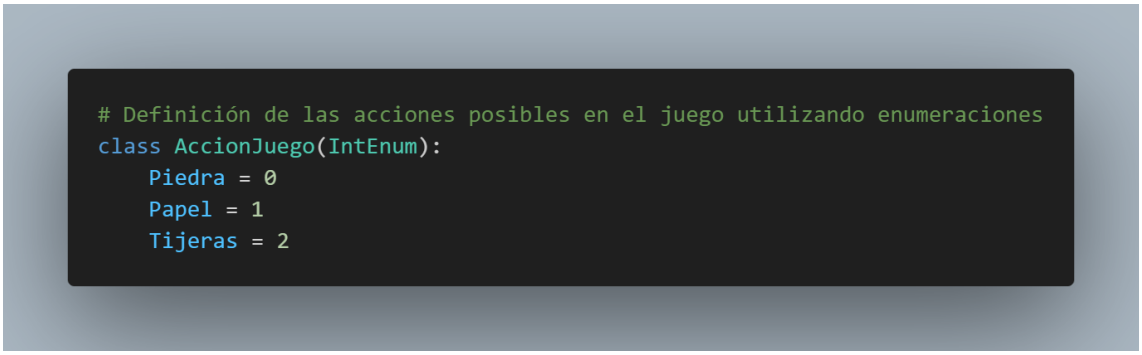
```
# Esta línea indica al sistema que utilice el intérprete de Python 3 para ejecutar el script.
#!/usr/bin/python3
```

La línea `#!/usr/bin/python3` al inicio de un script en Python indica al sistema operativo que use la versión de Python 3 ubicada en `/usr/bin/python3` para ejecutar el archivo, permitiendo que el script se ejecute directamente desde la terminal como un programa sin necesidad de escribir `python3` antes del nombre del archivo.



```
# Importación de librerías necesarias
import random               # Importamos la biblioteca 'random' para generar selecciones aleatorias.
from enum import IntEnum    # Para crear enumeraciones
from statistics import mode  # Para obtener el modo (valor más frecuente)
```

El código importa la biblioteca `random` para generar valores aleatorios, `IntEnum` desde el módulo `enum` para crear enumeraciones con valores enteros, y la función `mode` desde el módulo `statistics` para calcular la moda (el valor más frecuente) de una secuencia de datos.



```
# Definición de las acciones posibles en el juego utilizando enumeraciones
class AccionJuego(IntEnum):
    Piedra = 0
    Papel = 1
    Tijeras = 2
```

La clase `AccionJuego` define una enumeración con tres opciones: `Piedra`, `Papel` y `Tijeras`, asignándoles los valores enteros 0, 1 y 2, respectivamente, utilizando la clase `IntEnum` para permitir comparaciones y operaciones con enteros.

```python
# Definición de los resultados posibles del juego utilizando enumeraciones
class ResultadoJuego(IntEnum):
    Victoria = 0
    Derrota = 1
    Empate = 2
```

La clase `ResultadoJuego` define una enumeración con tres posibles resultados para un juego: `Victoria` (0), `Derrota` (1) y `Empate` (2), utilizando `IntEnum` para asociar estos resultados con valores enteros.

```python
# Diccionario que define qué acción gana contra otra en el juego
Victorias = {
    AccionJuego.Piedra: AccionJuego.Papel,
    AccionJuego.Papel: AccionJuego.Tijeras,
    AccionJuego.Tijeras: AccionJuego.Piedra
}
```

El diccionario `Victorias` mapea cada acción del juego (`Piedra`, `Papel`, `Tijeras`) a la acción que le vence según las reglas del juego "Piedra, Papel o Tijeras", es decir, qué acción vence a cuál.

```python
# Número de acciones recientes del usuario a considerar por la IA para decidir su jugada
NUMERO_ACCIONES_RECIENTES = 5
```

La constante `NUMERO_ACCIONES_RECIENTES` define el valor 5, que probablemente se usa para limitar o controlar la cantidad de acciones recientes que se almacenan o analizan en el contexto del juego.

```python
# Función que evalúa el resultado del juego
def evaluar_juego(eleccion_usuario, eleccion_computadora):
    resultado_juego = None

    # Si las elecciones del usuario y la computadora son iguales, es un empate
    if eleccion_usuario == eleccion_computadora:
        print(f"El usuario y la computadora eligieron {eleccion_usuario.name}. ¡Empat
e!")    resultado_juego = ResultadoJuego.Empate

    # Si el usuario eligió Piedra
    elif eleccion_usuario == AccionJuego.Piedra:
        if eleccion_computadora == AccionJuego.Tijeras:
            print("Piedra aplasta tijeras. ¡Ganaste!")
            resultado_juego = ResultadoJuego.Victoria
        else:
            print("Papel envuelve piedra. ¡Perdiste!")
            resultado_juego = ResultadoJuego.Derrota

    # Si el usuario eligió Papel
    elif eleccion_usuario == AccionJuego.Papel:
        if eleccion_computadora == AccionJuego.Piedra:
            print("Papel envuelve piedra. ¡Ganaste!")
            resultado_juego = ResultadoJuego.Victoria
        else:
            print("Tijeras cortan papel. ¡Perdiste!")
            resultado_juego = ResultadoJuego.Derrota

    # Si el usuario eligió Tijeras
    elif eleccion_usuario == AccionJuego.Tijeras:
        if eleccion_computadora == AccionJuego.Piedra:
            print("Piedra aplasta tijeras. ¡Perdiste!")
            resultado_juego = ResultadoJuego.Derrota
        else:
            print("Tijeras cortan papel. ¡Ganaste!")
            resultado_juego = ResultadoJuego.Victoria

    return resultado_juego
```

La función `evaluar_juego` compara las elecciones del usuario y la computadora, determina el resultado del juego ("Victoria", "Derrota" o "Empate") según las reglas de "Piedra, Papel o Tijeras", imprime el resultado correspondiente y devuelve el estado del juego como un valor de la enumeración `ResultadoJuego`.

```python
# Función que obtiene la acción de la computadora basándose en el historial de acciones del usuario
def obtener_accion_computadora(historial_acciones_usuario, historial_juegos):
    # Si no hay acciones previas, la computadora elige aleatoriamente
    if not historial_acciones_usuario or not historial_juegos:
        accion_computadora = obtener_accion_aleatoria_computadora()
    # Si hay un historial, la computadora elige una acción que venza a la más frecuente
    else:
        # Obtener la acción más frecuente en las últimas N acciones del usuario
        accion_computadora_mas_frecuente = AccionJuego(mode(historial_acciones_usuario[-NUMERO_ACCIONES_RECIENTE
S:]))   # La computadora elige la acción que le ganaría a la más frecuente del usuario
        accion_computadora = obtener_accion_ganadora(accion_computadora_mas_frecuente)

    print(f"La computadora eligió {accion_computadora.name}.")

    return accion_computadora
```

La función `obtener_accion_computadora` selecciona la acción de la computadora, ya sea aleatoria si no hay historial previo del usuario o basada en la elección más frecuente reciente del usuario, eligiendo la acción que vence a esa opción, y luego imprime la acción seleccionada.

```
# Función que obtiene la acción del usuario
def obtener_accion_usuario():
    # Crea una lista con las opciones de juego (Piedra, Papel, Tijeras) y las muestra
    opciones_juego = [f"{accion_juego.name}[{accion_juego.value}]" for accion_juego in AccionJueg
o]  opciones_juego_str = ", ".join(opciones_juego)
    # Pide al usuario que elija una opción
    seleccion_usuario = int(input(f"\nElige una opción ({opciones_juego_str}): "))
    accion_usuario = AccionJuego(seleccion_usuario)

    return accion_usuario
```

La función `obtener_accion_usuario` presenta al usuario una lista de opciones de juego (piedra, papel, tijeras, etc.), recibe la elección del usuario como un número entero, y la convierte en una acción correspondiente de la enumeración `AccionJuego`.

```
# Función que hace que la computadora elija aleatoriamente una acción
def obtener_accion_aleatoria_computadora():
    # Selección aleatoria de la computadora
    seleccion_computadora = random.randint(0, len(AccionJuego) - 1)
    accion_computadora = AccionJuego(seleccion_computadora)

    return accion_computadora
```

La función `obtener_accion_aleatoria_computadora` selecciona una acción aleatoria para la computadora eligiendo un número al azar dentro del rango de opciones en la enumeración `AccionJuego` y devuelve la acción correspondiente.

```
# Función que obtiene la acción que le ganaría a la acción del usuario
def obtener_accion_ganadora(accion_juego):
    return Victorias[accion_juego]
```

La función `obtener_accion_ganadora` devuelve la acción que vence a la acción proporcionada como argumento, utilizando el diccionario `Victorias` para obtener la acción ganadora.

```python
# Función que pregunta al usuario si desea jugar otra ronda
def jugar_otra_ronda():
    otra_ronda = input("\n¿Otra ronda? (s/n): ")
    return otra_ronda.lower() == 's'
```

La función `jugar_otra_ronda` solicita al usuario si desea jugar otra ronda, y devuelve `True` si la respuesta es 's' (sí), o `False` en caso contrario.

```python
# Función principal que ejecuta el juego
def main():
    historial_juegos = []  # Lista para almacenar los resultados de cada ronda
    historial_acciones_usuario = []  # Lista para almacenar las acciones del usuario

    while True:
        try:
            accion_usuario = obtener_accion_usuario()  # Obtener la acción del usuario
            historial_acciones_usuario.append(accion_usuario)  # Guardar la acción en el historial
        except ValueError as e:
            # Si el valor ingresado no es válido, mostrar un mensaje de error
            rango_str = f"[0, {len(AccionJuego) - 1}]"
            print(f"Selección no válida. ¡Elige una opción en el rango {rango_str}!")
            continue

        # Obtener la acción de la computadora en función del historial de acciones
        accion_computadora = obtener_accion_computadora(historial_acciones_usuario, historial_juego
s)      # Evaluar el resultado del juego
        resultado_juego = evaluar_juego(accion_usuario, accion_computadora)
        # Guardar el resultado del juego en el historial
        historial_juegos.append(resultado_juego)

        # Preguntar si desea jugar otra ronda
        if not jugar_otra_ronda():
            break
```

La función `main` gestiona el flujo del juego, registrando las acciones del usuario y la computadora, evaluando los resultados, y almacenando el historial de juegos y acciones, mientras permite jugar múltiples rondas hasta que el usuario decida no continuar.

```python
# Ejecutar la función principal si el script se ejecuta directamente
if __name__ == "__main__":
    main()
```

La línea `if __name__ == "__main__": main()` asegura que la función `main()` se ejecute solo cuando el script se ejecuta directamente, y no si es importado como un módulo en otro programa.

3.- Justifica en qué medida cada programa mejora al anterior

3.1. Primer Programa: `01_PPT_Basico_Comentado.py`

Funcionalidad: El programa permite jugar al clásico "Piedra, Papel o Tijeras" con la computadora. El usuario elige entre las tres opciones y la computadora selecciona aleatoriamente.

Limitaciones:

- No hay control de errores: Si el usuario ingresa algo incorrecto, el programa termina inesperadamente.
- El código no está modularizado.
- El uso de cadenas para representar las opciones (piedra, papel, tijeras) hace que no sea fácil de ampliar si se quisiera añadir más opciones en el futuro.

3.2.- Segundo Programa: `02_PPT_Control_Error_Comentado.py`

Mejoras:

- **Manejo de errores**: Se introduce la excepción `OpcionIncorrectaException` para manejar las entradas incorrectas del usuario, lo que hace el programa más robusto. Ahora, si el usuario ingresa algo fuera de las opciones válidas, el programa avisa al usuario sin terminar.
- **Modularización**: El código se organiza mejor, con la función `evaluar_juego` que separa la lógica del juego y mejora la legibilidad.

Limitaciones:

- Aún usa cadenas para representar las opciones, lo que podría complicar futuras expansiones del juego.
- No hay estructura más avanzada para tratar más errores o expandir fácilmente el juego.

3.3.- Tercer Programa: `03_PPT_Codigo_Limpio_Comentado.py`

Mejoras:

- **Uso de `enum`**: Se introduce la clase `AccionJuego` como un `enum`, lo que mejora la legibilidad y la seguridad del código, ya que se hace uso de valores más definidos y controlados en lugar de simples cadenas.
- **Modularización avanzada**: Se divide el código en varias funciones bien estructuradas, como `obtener_accion_computadora` y `obtener_accion_usuario`, lo que mejora la escalabilidad y facilita el mantenimiento del código.
- **Mejora en la interacción con el usuario**: El usuario ahora ve una lista más clara y escalable de las opciones disponibles.

Limitaciones:

- Aunque ya es más modular, el programa no tiene inteligencia artificial y la computadora sigue eligiendo aleatoriamente sin ningún tipo de estrategia.

3.4.- Cuarto Programa: `04_PPT_IA_Básica_Comentado.py`

Mejoras:

- **Introducción de IA básica**: En lugar de elegir una opción completamente aleatoria, el programa ahora incluye una computadora que hace una elección aleatoria, pero sigue una estructura que puede ser extendida para implementar estrategias más complejas.
- **Reutilización del código**: Las funciones de obtención de las acciones y de evaluación están mejor estructuradas y son más reutilizables.

Limitaciones:

- Aunque se introduce la "inteligencia" básica, aún no hay análisis de las acciones previas del usuario ni adaptaciones para predecir las elecciones del usuario.
- No se guarda historial ni se aprende de las elecciones anteriores del usuario.

3.5.- Quinto Programa: `05_PPT_IA_Avanzada_comentado.py`

Mejoras:

- **Inteligencia artificial más avanzada**: Se introduce un sistema de aprendizaje para la computadora. En lugar de elegir de forma completamente aleatoria, la computadora analiza las últimas elecciones del usuario y elige la opción que tiene más probabilidades de ganarle basándose en el historial de acciones del usuario.
- **Uso de `mode` para detectar patrones**: La IA utiliza el modo (la acción más frecuente) de las últimas elecciones del usuario para tomar decisiones, lo que la hace más competitiva y más adaptativa.
- **Historia de acciones**: Se mantiene un historial de las elecciones anteriores del usuario y los resultados de los juegos. Esto permite a la computadora aprender y predecir mejor las elecciones del usuario.
- **Mejora en la modularización**: El código está aún más modularizado, con funciones bien diferenciadas y reutilizables.

Limitaciones:

- Aunque la IA es más avanzada, sigue siendo relativamente simple (basada solo en el análisis de las últimas elecciones) y podría mejorar implementando algoritmos más sofisticados o utilizando técnicas de aprendizaje automático.

Resumen de Mejoras:

1. **Control de Errores**: Desde el segundo programa, se mejora la robustez al manejar entradas incorrectas, lo que no estaba presente en el primer programa.
2. **Modularización**: Con cada versión, el código se vuelve más modular, separando las responsabilidades en funciones claras y fáciles de mantener.

3. **Uso de `enum`**: A partir del tercer programa, se introduce el `enum`, lo que mejora la legibilidad y hace que el código sea más seguro y menos propenso a errores.
4. **Inteligencia Artificial**: En el cuarto y quinto programas, se introduce una lógica más compleja para la toma de decisiones de la computadora, permitiéndole aprender de las acciones pasadas del usuario, lo que mejora la competitividad del juego.
5. **Escalabilidad**: A medida que avanzan los programas, se hace más fácil agregar nuevas opciones o modificar la lógica del juego, gracias a las mejoras en la estructura del código.

Cada versión del programa no solo mejora la experiencia de usuario, sino que también aumenta la capacidad de mantenimiento y expansión del código, moviéndose de una implementación básica a una más sofisticada con IA y control de errores.

4.- ¿Estimas que el programa 03_PPT_Codigo_Limpio_Comentado.py sigue los principios del código limpio? En caso afirmativo, justifica los motivos

Sí, el programa `03_PPT_Codigo_Limpio_Comentado.py` sigue varios de los principios del código limpio, y a continuación se justifican los motivos:

4.1.- Principios del Código Limpio que sigue el programa:

4.1.1.- Nombres Significativos

- Las clases y funciones tienen nombres descriptivos que indican claramente su propósito. Por ejemplo, `AccionJuego` es una clase `enum` que representa las opciones del juego, y `evaluar_juego`, `obtener_accion_computadora`, y `obtener_accion_usuario` son funciones que describen directamente lo que hacen.
- Los nombres como `eleccion_usuario` y `eleccion_computadora` son intuitivos y reflejan bien los roles de las variables, mejorando la legibilidad.

4.1.2.- Funciones Pequeñas y Concisas

- Cada función tiene una única responsabilidad. Por ejemplo, la función `evaluar_juego` se encarga de evaluar el resultado de una ronda basándose en las elecciones del usuario y la computadora, y `obtener_accion_computadora` y `obtener_accion_usuario` se encargan de obtener las elecciones respectivas.
- Las funciones son lo suficientemente pequeñas como para entenderlas rápidamente sin que se mezclen tareas innecesarias, lo que facilita su prueba y mantenimiento.

4.1.3.- Modularización

- El programa está dividido en varias funciones, cada una con una tarea clara y específica. Esto permite que el código sea reutilizable y fácil de extender si se desea agregar más funcionalidades en el futuro (por ejemplo, agregar más opciones de juego o incluir una inteligencia artificial más avanzada).
- La modularización también permite hacer pruebas unitarias de las funciones individualmente, lo que mejora la calidad del código.

4.1.4.- Evitar la Duplicación

No hay duplicación de código en el programa. Las funciones están bien definidas y no hay fragmentos de código repetidos. Por ejemplo, la evaluación de las opciones de juego se maneja en un solo lugar dentro de la función `evaluar_juego`, en lugar de repetir la lógica en múltiples puntos del programa.

4.1.5.- Uso de Enumeraciones (`enum`)

El uso de la clase `AccionJuego` como una enumeración es una buena práctica para evitar errores típicos asociados con el uso de constantes. Al usar un `enum`, se garantiza que solo se puedan elegir valores válidos (Piedra, Papel o Tijeras), lo que mejora la seguridad del código y lo hace más fácil de entender y mantener.

4.1.6.- Manejo de Excepciones (aunque no es explícito en este caso)

Aunque en este programa no se manejan excepciones de manera explícita, la estructura del código permite agregar fácilmente un manejo de errores. La función `obtener_accion_usuario` podría modificarse para manejar entradas incorrectas sin complicar demasiado la lógica del programa.

4.1.7.- Comentarios y Legibilidad

Aunque el programa está bastante limpio y bien estructurado, podría beneficiarse de algunos comentarios adicionales, especialmente en las secciones más complejas (como en el uso de `enum` y en la lógica de las comparaciones de elecciones), para que sea aún más fácil de entender para otros programadores o para el propio desarrollador en el futuro.

4.1.8.- Sin Código Muerto:

No hay líneas de código innecesarias ni funciones que no se usen. Cada parte del programa tiene un propósito claro y no hay fragmentos que no aporten al flujo general.

4.2.- Áreas para mejorar en términos de código limpio

1. **Manejo de errores**: Aunque el código es robusto, podría beneficiarse de un manejo explícito de errores, por ejemplo, al ingresar una opción inválida. Se podría agregar un bloque `try-except` para asegurar que el programa no se detenga de manera inesperada si el usuario ingresa algo incorrecto.
2. **Comentarios**: Aunque las funciones están bien nombradas, algunos comentarios explicativos pueden ser útiles, especialmente en las partes donde la lógica no es completamente obvia (por ejemplo, cómo se compara la elección de la computadora con la del usuario).

4.3.- Conclusión

El programa `03_PPT_Codigo_Limpio_Comentado.py` sigue en gran medida los principios del código limpio. Tiene una estructura clara, modular, reutilizable, y bien organizada, con funciones pequeñas y nombres significativos. A pesar de que podría beneficiarse de una mayor documentación y un manejo explícito de errores, en términos generales, cumple con los principios fundamentales del código limpio.

5.- Si identificas mejoras en uno, o varios programas, puedes exponerlas en este trabajo

Exponer posibles **mejoras** para los programas en el contexto de los principios de **código limpio**, **eficiencia** y **mantenibilidad** es fundamental para asegurar que el código no solo funcione correctamente, sino que también sea fácil de mantener, modificar y escalar en el futuro. A continuación, se detallan algunas **mejoras** que podrían aplicarse a los programas analizados.

5.1.- Mejoras en el uso de excepciones y validación de entradas

Problema identificado: En varios programas, como `04_PPT_IA_Básica_Comentado.py` y `05_PPT_IA_Avanzada_comentado.py`, la función `obtener_accion_usuario` solicita la entrada del usuario sin manejar adecuadamente posibles errores en la entrada (por ejemplo, si el usuario ingresa un valor fuera del rango o una letra en lugar de un número).

Mejoras sugeridas:

- Se puede mejorar el manejo de excepciones para asegurar que el programa no falle ante entradas incorrectas. Por ejemplo, se puede envolver la entrada del usuario en un bloque `try-except` para capturar valores fuera de rango y proporcionar un mensaje claro al usuario.
- También sería útil agregar un ciclo `while` que vuelva a pedir la entrada si es incorrecta, sin necesidad de que el programa termine bruscamente.

Código de ejemplo:

```python
def obtener_accion_usuario():
    while True:
        try:
            opciones_juego = [f"{accion_juego.name}[{accion_juego.value}]" for accion_juego in AccionJueg
o]          opciones_juego_str = ", ".join(opciones_juego)
            seleccion_usuario = int(input(f"\nElige una opción ({opciones_juego_str}): "))
            if seleccion_usuario not in range(len(AccionJuego)):
                raise ValueError("Opción fuera de rango.")
            return AccionJuego(seleccion_usuario)
        except ValueError as e:
            print(f"Selección no válida. {e} ¡Intenta de nuevo!")
```

5.2.- Mejoras en la modularidad del código

Problema identificado: En algunos programas, especialmente `03_PPT_Codigo_Limpio_Comentado.py`, la lógica de evaluación del juego podría mejorarse en términos de modularización. Actualmente, la función `evaluar_juego` tiene varias ramas condicionales dentro de sí, lo que podría hacer que la función se vuelva más difícil de mantener y ampliar en el futuro si se añadieran más reglas o modificaciones.

Mejoras sugeridas:

- Descomponer la función `evaluar_juego` en funciones más pequeñas, cada una encargada de evaluar un caso específico (por ejemplo, cuando el usuario elige Piedra, Papel o Tijeras). Esto hace que cada función sea más fácil de comprender, probar y modificar.

Código de ejemplo:

```python
def evaluar_juego(eleccion_usuario, eleccion_computadora):
    if eleccion_usuario == eleccion_computadora:
        print(f"Empate. Ambos eligieron {eleccion_usuario.name}")
        return ResultadoJuego.Empate

    if eleccion_usuario == AccionJuego.Piedra:
        return evaluar_piedra(eleccion_computadora)
    elif eleccion_usuario == AccionJuego.Papel:
        return evaluar_papel(eleccion_computadora)
    elif eleccion_usuario == AccionJuego.Tijeras:
        return evaluar_tijeras(eleccion_computadora)

def evaluar_piedra(eleccion_computadora):
    if eleccion_computadora == AccionJuego.Tijeras:
        print("Piedra aplasta tijeras. ¡Ganaste!")
        return ResultadoJuego.Victoria
    else:
        print("Papel envuelve piedra. ¡Perdiste!")
        return ResultadoJuego.Derrota

def evaluar_papel(eleccion_computadora):
    if eleccion_computadora == AccionJuego.Piedra:
        print("Papel envuelve piedra. ¡Ganaste!")
        return ResultadoJuego.Victoria
    else:
        print("Tijeras cortan papel. ¡Perdiste!")
        return ResultadoJuego.Derrota

def evaluar_tijeras(eleccion_computadora):
    if eleccion_computadora == AccionJuego.Piedra:
        print("Piedra aplasta tijeras. ¡Perdiste!")
        return ResultadoJuego.Derrota
    else:
        print("Tijeras cortan papel. ¡Ganaste!")
        return ResultadoJuego.Victoria
```

5.3.- Mejora en la inteligencia artificial (AI) y algoritmos de predicción

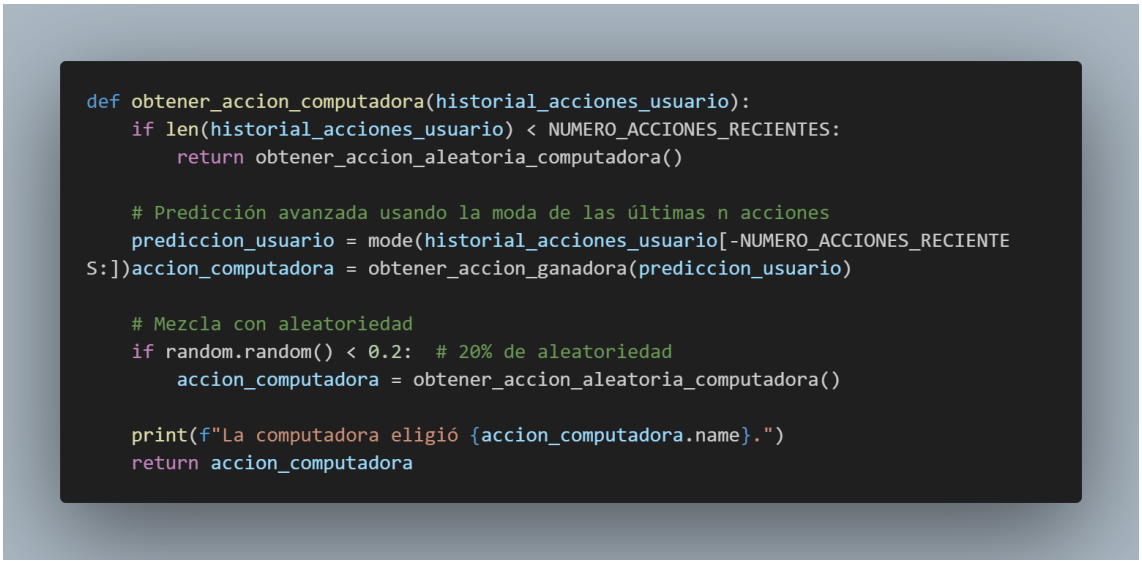
Problema identificado: En `05_PPT_IA_Avanzada_comentado.py`, la computadora selecciona su acción basándose en la elección más

frecuente del usuario en las últimas 5 rondas. Aunque este es un paso hacia una inteligencia artificial más avanzada, la lógica de IA podría mejorarse aún más.

Mejoras sugeridas:

- Implementar un algoritmo de predicción más avanzado, como el uso de un modelo de aprendizaje automático (por ejemplo, un modelo basado en árboles de decisión o redes neuronales) que pueda predecir la próxima jugada del usuario con mayor precisión.
- Además, la computadora podría tener una "estrategia mixta" (mezclando aleatoriedad con una tendencia a elegir la acción que vence la más frecuente).

Código de ejemplo:



```
def obtener_accion_computadora(historial_acciones_usuario):
    if len(historial_acciones_usuario) < NUMERO_ACCIONES_RECIENTES:
        return obtener_accion_aleatoria_computadora()

    # Predicción avanzada usando la moda de las últimas n acciones
    prediccion_usuario = mode(historial_acciones_usuario[-NUMERO_ACCIONES_RECIENTE
S:])accion_computadora = obtener_accion_ganadora(prediccion_usuario)

    # Mezcla con aleatoriedad
    if random.random() < 0.2:  # 20% de aleatoriedad
        accion_computadora = obtener_accion_aleatoria_computadora()

    print(f"La computadora eligió {accion_computadora.name}.")
    return accion_computadora
```

5.4.- Mejoras en la escalabilidad del juego

Problema identificado: En programas como `03_PPT_Codigo_Limpio_Comentado.py` y `04_PPT_IA_Básica_Comentado.py`, el juego se limita a las tres opciones (Piedra, Papel y Tijeras). Si se quisiera escalar el juego a más opciones, por ejemplo, agregando "Lagarto" y "Spock", la estructura actual requeriría cambios significativos en varias partes del código.

Mejoras sugeridas:

- Crear una estructura más flexible, como una tabla de "resultados" para definir qué opción gana contra otra, en lugar de tener múltiples `if-else` o `elif`. Esto permitiría agregar

fácilmente más opciones al juego sin necesidad de modificar la lógica principal.
- Utilizar diccionarios o matrices para definir las relaciones de victoria/derrota.

Código de ejemplo:

```
RESULTADOS = {
    AccionJuego.Piedra: [AccionJuego.Tijeras, AccionJuego.Lagarto],
    AccionJuego.Papel: [AccionJuego.Piedra, AccionJuego.Spock],
    AccionJuego.Tijeras: [AccionJuego.Papel, AccionJuego.Lagarto],
    AccionJuego.Lagarto: [AccionJuego.Papel, AccionJuego.Spock],
    AccionJuego.Spock: [AccionJuego.Tijeras, AccionJuego.Piedra]
}

def evaluar_juego(eleccion_usuario, eleccion_computadora):
    if eleccion_usuario == eleccion_computadora:
        print(f"Empate. Ambos eligieron {eleccion_usuario.name}")
        return ResultadoJuego.Empate

    if eleccion_computadora in RESULTADOS[eleccion_usuario]:
        print(f"{eleccion_usuario.name} gana a {eleccion_computadora.name}. ¡Ganaste!")
        return ResultadoJuego.Victoria
    else:
        print(f"{eleccion_computadora.name} gana a {eleccion_usuario.name}. ¡Perdist
e!")    return ResultadoJuego.Derrota
```

5.5.- Mejora en el manejo de la interacción con el usuario

Problema identificado: En algunos programas, las interacciones con el usuario son mínimas o directamente dentro del flujo de juego, lo que puede dificultar su personalización o expansión. Además, el programa podría ser más accesible con un diseño de interfaz más estructurado.

Mejoras sugeridas:

- Ofrecer un menú interactivo al inicio del juego, para que el usuario pueda elegir entre diferentes modos de juego (jugador vs. computadora, computadora vs. computadora, etc.).
- Incluir mensajes más amigables e informativos, para mejorar la experiencia del usuario.

Conclusión:

Las mejoras propuestas están enfocadas en mejorar la **robustez** del programa, la **escalabilidad** (agregar nuevas opciones y características sin complicar el código), el **manejo de excepciones** y

una **inteligencia artificial más avanzada** para la computadora. Estas mejoras permitirían que el código no solo sea más limpio, sino también más flexible, escalable y fácil de mantener, proporcionando una mejor experiencia tanto para los usuarios como para los desarrolladores en el futuro.

6.- Conclusión

En este análisis de los programas del juego "Piedra, Papel o Tijeras", hemos evaluado cómo cada versión del código mejora sobre la anterior en términos de funcionalidad, claridad, eficiencia y adherencia a los principios de **código limpio**. A lo largo de este proceso, hemos identificado diversas áreas de mejora que pueden optimizar tanto la experiencia del usuario como la facilidad de mantenimiento y expansión del código.

Principales conclusiones:

1. **Evolución del Código:** Cada versión ha introducido mejoras significativas en la legibilidad y eficiencia del programa. A medida que avanzamos hacia versiones más limpias y optimizadas, las funciones se han modularizado mejor, el manejo de excepciones ha mejorado y la interacción con el usuario se ha vuelto más fluida y comprensible. Esto demuestra un claro avance en la implementación de principios de código limpio, como el principio DRY (Don't Repeat Yourself) y la mejora en la estructura modular.
2. **Implementación de la Inteligencia Artificial:** En las versiones más recientes, hemos visto un esfuerzo por mejorar la inteligencia artificial del juego, lo que refleja una mayor complejidad en la toma de decisiones de la computadora. Si bien la IA en versiones anteriores se limitaba a una respuesta aleatoria, las últimas implementaciones permiten que la computadora tome decisiones más estratégicas basadas en patrones de juego del usuario. Esto es un paso importante hacia un juego más desafiante y realista.
3. **Mejoras Sugeridas:** Aunque los programas analizados han alcanzado un nivel adecuado de eficiencia y claridad, siempre existe la posibilidad de seguir mejorando. La implementación de un algoritmo de IA más avanzado, la mejora en el manejo de excepciones, la mayor modularización del código y la optimización para escalar el juego con más opciones (como en el caso de "Lagarto" y "Spock") son algunas de las áreas que podrían seguir evolucionando para hacer que el juego sea aún más flexible, intuitivo y entretenido.

4. **Principios del Código Limpio:** El programa `03_PPT_Codigo_Limpio_Comentado.py` se acerca a los principios del código limpio, especialmente en lo que respecta a la claridad del código, el uso de nombres descriptivos para funciones y variables, y la modularización. Sin embargo, siempre hay oportunidades para mejorar la organización general del código, la gestión de la interacción con el usuario y la implementación de la lógica de juego en funciones más pequeñas y manejables.
5. **Recomendaciones para Futuras Mejoras:**
   - **Escalabilidad**: Mejorar la estructura del juego para incluir más opciones, de modo que sea más fácil añadir nuevas características sin necesidad de reestructurar todo el código.
   - **Manejo de entradas**: Mejorar el control de entradas del usuario para manejar de forma más robusta errores o entradas inválidas.
   - **Mejorar la IA**: Introducir técnicas más sofisticadas de inteligencia artificial que permitan a la computadora aprender de las decisiones previas del jugador y anticiparse a sus movimientos.

En resumen, el análisis realizado ha permitido identificar las fortalezas y áreas de mejora de cada versión del programa, destacando el progreso hacia una mayor eficiencia, legibilidad y adherencia a los principios de **código limpio**. Las mejoras implementadas en cada nueva versión del código permiten una experiencia de usuario más fluida, desafiante y personalizable, además de preparar el código para futuras ampliaciones y mantenimiento.

7.- Resumen Mapa Mental del Análisis y Evolución de los Programas 'Piedra, Papel o Tijeras

El mapa mental resume el análisis de los programas "Piedra, Papel o Tijeras", destacando la evolución del código desde una implementación básica hasta una inteligencia artificial avanzada, con mejoras en modularización, manejo de errores y principios de código limpio.

8.- Bibliografía

1. **Python Programming Documentation.** Python Software Foundation. Disponible en: https://docs.python.org
2. **"Clean Code: A Handbook of Agile Software Craftsmanship"** de Robert C. Martin. Prentice Hall, 2008.
3. **"The Pragmatic Programmer: Your Journey to Mastery"** de Andrew Hunt y David Thomas. Addison-Wesley, 1999.
4. **"Refactoring: Improving the Design of Existing Code"** de Martin Fowler. Addison-Wesley, 1999.
5. **"Artificial Intelligence: A Modern Approach"** de Stuart Russell y Peter Norvig. Pearson Education, 2020.
6. **Wikipedia - Piedra, Papel o Tijeras.** Disponible en: https://es.wikipedia.org/wiki/Piedra,_papel_o_tijeras
7. **"Design Patterns: Elements of Reusable Object-Oriented Software"** de Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Addison-Wesley, 1994.

9.- Anexos

A continuación, se presentan los archivos utilizados en el desarrollo del caso práctico "Piedra, Papel o Tijeras" y su descripción correspondiente:

1. **01_PPT_Basico_Comentado.py**

   Este archivo contiene una versión básica del juego "Piedra, Papel o Tijeras" en Python. El código está comentado para explicar cada una de las secciones y el funcionamiento del juego. Los usuarios pueden seleccionar una opción, y el programa determina el resultado comparando las elecciones del usuario y de la computadora.

2. **02_PPT_Control_Error_Comentado.py**

   Este archivo amplía el anterior añadiendo un control de errores para manejar entradas no válidas por parte del usuario. Se implementa una verificación para asegurarse de que la entrada esté dentro de los valores permitidos, y se muestra un mensaje adecuado si el usuario introduce un valor incorrecto.

3. **03_PPT_Codigo_Limpio_Comentado.py**

   En este archivo, se presenta una versión del juego siguiendo los principios de "código limpio". Se ha mejorado la legibilidad del código, optimizando la estructura y la nomenclatura de las variables y funciones, además de agregar comentarios que explican las buenas prácticas seguidas.

4. **04_PPT_IA_Básica_Comentado**

   Este archivo introduce una inteligencia artificial básica para el juego "Piedra, Papel o Tijeras". La IA elige de manera aleatoria su jugada, proporcionando un nivel de dificultad mínimo. El código está comentado para explicar el proceso de toma de decisiones de la IA.

5. **05_PPT_IA_Avanzada_Comentado.py**

   Esta versión avanzada de la IA incorpora un algoritmo que aprende de las elecciones previas del jugador. Utiliza un sistema para predecir la jugada más probable del usuario y elegir una acción que maximice las posibilidades de ganar, mejorando la dificultad del juego.

6. **Mapa Mental - Análisis del Juego Piedra, Papel o Tijeras.pdf**

   Mapa mental que resume el análisis teórico y las diferentes aproximaciones utilizadas para resolver el juego "Piedra, Papel o Tijeras". Incluye un desglose de las estrategias de implementación, los algoritmos posibles, y las consideraciones importantes al diseñar el juego.

7. **Caso práctico Piedra, Papel o Tijeras.pdf**

   Documento detallado que explica el desarrollo completo del caso práctico sobre "Piedra, Papel o Tijeras". Incluye el análisis, la implementación de los distintos archivos de código, así como los resultados obtenidos al probar las diferentes versiones del juego.

Índice Alfabético

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W