Solución Desafío -Tipos de métodos

Requerimiento 1

Archivo personaje.py

1. Crear clase Personaje

```
class Personaje():
```

2. Agregar constructor. El constructor debe recibir por parámetro el nombre (obligatorio), y asignarlo al atributo de instancia. También debe asignar el valor 1 al nivel y el valor 0 a la experiencia.

```
class Personaje():

    def __init__(self, nombre):
        self.nombre = nombre
        self.nivel = 1
        self.experiencia = 0
```

3. (A continuación del código anterior) Agregar propiedad que define método getter de estado, retornando un string con nombre, nivel y experiencia.

```
    @property
    def estado(self):
        return (f"NOMBRE: {self.nombre}"
                f"NIVEL: {self.nivel}"
                f"EXP: {self.experiencia}")
```

4. (A continuación del código anterior) Agregar el método que define setter de estado, recibiendo la experiencia por parámetro. Dentro del método, crear variable temporal que almacena el resultado de la experiencia actual más la ingresada.

```
    @estado.setter
    def estado(self, exp: int):
        tmp_exp = self.experiencia + exp
```

5. (A continuación, dentro del método anterior) Mientras la variable temporal sea mayor o igual a 100, sumar 1 al nivel de la instancia, y restar 100 a la variable temporal

```
while tmp_exp >= 100:
    self.nivel += 1
    tmp_exp -= 100
```

6. (A continuación, terminado el while anterior) Mientras la variable temporal sea negativa, si el nivel es mayor a 1, restar 1 al nivel de la instancia, y agregar 100 a la variable temporal. Si el nivel no es mayor a 1, solo asignar 0 a la variable temporal.

```
while tmp_exp < 0:
    if self.nivel > 1:
        tmp_exp = 100 + tmp_exp
        self.nivel -= 1
    else:
        tmp_exp = 0
```

7. (A continuación, terminando el while anterior) Asignar el valor de la variable temporal al atributo experiencia de la instancia.

```
self.experiencia = tmp_exp
```

8. (A nivel de la clase) Sobrecargar método `__lt__`, considerando que la instancia actual será menor a la otra si su nivel es menor.

```
def __lt__(self, other) -> bool:
    return self.nivel < other.nivel
```

9. (A nivel de la clase) Sobrecargar método `__gt__`, considerando que la instancia actual será mayor a la otra si su nivel es mayor.

```
def __gt__(self, other) -> bool:
    return self.nivel > other.nivel
```

10. (A nivel de la clase) Opcional. Sobrecargar el método `__eq__`, considerando que dos instancias son iguales si tienen el mismo nivel.

```
def __eq__(self, other) -> bool:
    return self.nivel == other.nivel
```

11. (A nivel de clase) Opcional. Crear método de instancia que retorna probabilidad de ganar un ataque de la instancia actual versus otra.

```
def get_probabilidad_ganar(self, other) -> float:
    if self < other:
        return 0.33
    elif self > other:
        return 0.66
    else:
        return 0.5
```

12. (A nivel de clase) Opcional. Crear método estático que muestra menú con chance de ganar (ingresada por parámetro) y retorna opción del usuario.

```
@staticmethod
def mostrar_dialogo_opcion(probabilidad_ganar):
    return int(input(
        f"\nCon tu nivel actual, tienes {probabilidad_ganar * 100}% "
        "de probabilidades de ganarle al Orco.\n"
        "\nSi ganas, ganarás 50 puntos de experiencia y el orco perderá 30. \n"
        "Si pierdes, perderás 30 puntos de experiencia y el orco ganará 50.\n"
        "\n¿Qué deseas hacer?\n"
        "1. Atacar\n"
        "2. Huir\n"
    ))
```

Requerimiento 2

Archivo juego.py

1. Importar clase Personaje y módulo random.

```
from personaje import Personaje
import random
```

2. Dar saludo a usuario y almacenar el nombre del personaje.

```
print("¡Bienvenido a Gran Fantasía!")
nombre = input("Por favor indique nombre de su personaje:\n")
```

3. Crear personaje con el nombre entregado, y mostrar estado.

```
p = Personaje(nombre)
print(p.estado)
```

4. Crear personaje orco y almacenar probabilidad del personaje de ganar un ataque al orco.

```
print("\n¡Oh no!, ¡Ha aparecido un Orco!")
o = Personaje("Orco")
probabilidad_ganar = p.get_probabilidad_ganar(o)
```

5. Almacenar opción del jugador, e iniciar ciclo mientras la opción sea 1.

```
opcion_orco = Personaje.mostrar_dialogo_opcion(probabilidad_ganar)

while opcion_orco == 1:
```

6. Dentro del ciclo, almacenar el resultado del ataque.

```
    resultado = "G" if random.uniform(0,1) < probabilidad_ganar else "P"
```

7. Si el resultado es ganar, informar al usuario y actualizar estados, asignando 50 puntos al del jugador, y restando 30 al del orco.

```
if resultado == "G":
    print(
        "\n¡Le has ganado al orco, felicidades!\n"
        "¡Recibirás 50 puntos de experiencia!\n"
    )
    p.estado = 50
    o.estado = -30
```

8. En caso contrario, informar al usuario y actualizar estados, asignando 50 puntos al orco, y restando 30 al jugador.

```
else:
    print(
        "\n¡Oh no! ¡El orco te ha ganado!\n"
        "¡Has perdido 30 puntos de experiencia!\n"
    )
    p.estado = -30
    o.estado = 50
```

9. Mostrar estados actualizados en pantalla.

```
print(p.estado)
print(o.estado)
```

10. Actualizar probabilidad de ganar ataque y almacenar nueva opción de jugador.

```
probabilidad_ganar = p.get_probabilidad_ganar(o)
opcion_orco = Personaje.mostrar_dialogo_opcion(probabilidad_ganar)
```

11. (Fuera del ciclo) En caso de huida, mostrar mensaje correspondiente.

```
print("\n¡Has huido! El orco ha quedado atrás.")
```