Análisis de Balance del Juego 'Duelo de Titanes' + Reequilibrio (Hollow Knight)

Reporte generado en ChatGPT (versión PDF).

1) Parámetros para causar desbalance y que gane siempre el Jugador 2

El juego original crea los atributos de cada jugador de forma aleatoria dentro de un rango. Para forzar que el Jugador 2 gane la mayoría de las veces, basta con modificar sus atributos al momento de crearlo. Los tres atributos más influyentes en este diseño son:

- Velocidad: determina quién ataca primero en cada ronda.

- Armadura: reduce el daño recibido, porque el daño se calcula como (fuerza*0.7 + inteligencia*0.3) - armadura*0.5.

- Salud: le da más margen de supervivencia en las 3 rondas.

Si al Jugador 2 le damos +40 de salud, +5 de armadura y +10 de velocidad, conseguimos un sesgo claro. Ataca primero, recibe menos daño y tarda más en morir. En un sistema de solo 3 rondas, eso es suficiente para que domine.

Código de creación de Jugador 2 desbalanceado (para Colab):

```
class Jugador:
    def __init__(self, nombre):
        self.nombre = nombre
        self.fuerza = random.randint(5, 15)
        self.salud = random.randint(80, 120)
        self.inteligencia = random.randint(5, 15)
        self.armadura = random.randint(5, 15)
        self.velocidad = random.randint(5, 15)

        # Desbalance a favor del Jugador 2
        if self.nombre == "Jugador 2":
            self.salud += 40
            self.armadura += 5
            self.velocidad += 10
```

2) Cómo medir o comprobar si hay desbalance sutil

Con una sola partida no se ve el desbalance. Hay que ejecutar muchas peleas con las mismas reglas y contar cuántas veces gana cada jugador. Si un jugador supera de forma consistente el 55–60% de las victorias en un sistema que se supone justo, hay indicios de desbalance.

Pasos:

1. Ejecutar el combate 1000 veces.

2. Registrar quién ganó cada vez.

3. Calcular porcentajes.

4. Graficar o mostrar una tabla.

```
def jugar_una_vez():
    # ... crear jugadores y pelear 3 rondas ...
    if jugador1.salud > jugador2.salud:
        return "J1"
    elif jugador2.salud > jugador1.salud:
        return "J2"
    else:
        return "EMP"

def simular(n=1000):
    conteo = {"J1": 0, "J2": 0, "EMP": 0}
    for _ in range(n):
        ganador = jugar_una_vez()
        conteo[ganador] += 1
    return conteo

resultados = simular(1000)
print(resultados)
```

Interpretación: si obtienes algo como {'J1': 420, 'J2': 540, 'EMP': 40}, ya puedes decir que hay una ligera ventaja para el Jugador 2. Si el Jugador 2 llega a 650–700 victorias de 1000, es un desbalance claro.

3) ¿Qué ajuste llevaría siempre a un empate?

La forma más simple de forzar empates en este diseño es eliminar las diferencias iniciales y el orden de ataque. Es decir: dar los mismos atributos a ambos jugadores y hacer que siempre ataquen los dos en cada ronda. Si tienen la misma salud, el mismo daño y el mismo número de rondas, terminarán empatando.

```
# Forzar igualdad
for pj in (jugador1, jugador2):
    pj.salud = 100
    pj.fuerza = 12
    pj.inteligencia = 10
    pj.armadura = 10
    pj.velocidad = 10

# Rondas: ambos atacan
for _ in range(3):
    jugador1.atacar(jugador2)
    jugador2.atacar(jugador1)
```

Tabla de ejemplo de 3 escenarios (1000 partidas simuladas)

Escenario	J1 gana	J2 gana	Empates	Comentario
Base (aleatorio)	≈ 350-450	≈ 350-450	bajo	más o menos balanceado
Sesgo a J2	bajo	alto (600+)	bajo	desbalance claro
Empate forzado	0	0	1000	reglas simétricas

Análisis de Juego: Hollow Knight

Descripción del juego y su sistema de progresión / combate

Hollow Knight es un metroidvania 2D de exploración no lineal. El avance del jugador no depende de subir de nivel, sino de obtener habilidades y herramientas que abren nuevas rutas (manto, garra de Mantis, aguijón onírico, etc.). El combate es de corto alcance, muy basado en timing, posicionamiento y gestión del recurso 'alma', que sirve tanto para lanzar hechizos como para curarse.

La progresión se expresa en tres ejes: (1) mejoras al aguijón (más daño), (2) aumento de máscaras de vida, (3) equipamiento de amuletos que modifican el estilo de juego (daño, curación, movilidad, alcance, invocaciones).

Problema de balance identificado

Cuando el jugador acumula varias mejoras ofensivas (aguijón mejorado + amuletos de daño) algunos jefes intermedios se derrotan demasiado rápido. Esto 'aplana' el pico de dificultad que el juego intenta mantener en ciertas zonas. No es un bug, pero sí un desbalance suave: el juego recompensa tanto la exploración que permite al jugador saltarse el reto planeado.

Propuesta de modificación o reequilibrio

1) Escalonar el coste de amuletos ofensivos para evitar llevar muchos a la vez.

2) Aumentar ligeramente la vida/fases de jefes si el jugador llega muy mejorado.

3) Añadir una leve resistencia a daño repetitivo en jefes clave.

Justificación de mejora de la jugabilidad

Con estos ajustes no se rompe la fantasía de poder del jugador (sigue sintiéndose más fuerte conforme avanza), pero se protege el ritmo del juego y el valor de las bossfights. El jugador debe elegir: daño, movilidad o curación; esto genera decisiones significativas y mantiene el desafío.