

Sistema de Puntuación y JSON en Tetris

Lógica del Juego y Persistencia de Datos

Carlos Alberto Bello Rosas

Universidad Autónoma de Zacatecas
Ingeniería en Software

14 de diciembre de 2025

Contenido de la Exposición

- 1 Lógica de Puntuación
- 2 Implementación en Python
- 3 Modelo Matemático
- 4 Persistencia con JSON
- 5 Conclusión

Sistema de Puntuación del Tetris

Filosofía del Diseño

El sistema no es lineal. Busca premiar la **habilidad** y el **riesgo**:

- **Estrategia**: Limpiar 4 líneas (Tetris) vale mucho más que 4 sencillas.
- **Dificultad**: A mayor nivel (velocidad), mayor recompensa.

Tabla de Puntos Base (Estándar NES)

Líneas	Acción	Puntos Base
1	Single	40
2	Double	100
3	Triple	300
4	Tetris	1200

1. Definición de Constantes

```
1 # Valores base seg n la normativa cl sica
2 SCORE_VALUES = {
3     1: 40,      # Single
4     2: 100,     # Double
5     3: 300,     # Triple
6     4: 1200    # Tetris
7 }
8
9 # Configuraci n de dificultad
10 BASE_SPEED = 500      # ms iniciales
11 SPEED_DECREMENT = 40 # Reducc i n por nivel
12 LEVEL_UP_LINES = 10   # L neas para subir nivel
```

Observación Matemática

Hacer un *Tetris* (1200 pts) es **30 veces más valioso** que una línea simple (40 pts).

2. Algoritmo de Actualización

```
1 def update_score_and_level(lines_cleared):
2     global score, level, lines_cleared_count
3
4     if lines_cleared == 0:
5         return
6
7     # F RMULA PRINCIPAL: Base * (Nivel + 1)
8     # Nota: Muchos sistemas usan (Nivel + 1) para que el Nivel 0 cuente
9     base_points = SCORE_VALUES.get(lines_cleared, 0)
10    score += base_points * level
11
12    # L gica de subida de nivel
13    lines_cleared_count += lines_cleared
14    if lines_cleared_count >= LEVEL_UP_LINES:
15        level += 1
16        lines_cleared_count -= LEVEL_UP_LINES # Residuo se mantiene
17        increase_speed() # Aumenta la dificultad
18
19    update_gui_labels() # Refresca la vista
```

Fórmulas del Sistema

Para calcular el puntaje total S tras un movimiento:

Fórmula de Puntuación

$$S_{nuevo} = S_{actual} + (P_{base} \times N_{actual})$$

Donde $P_{base} \in \{40, 100, 300, 1200\}$

Ejemplo Práctico: Nivel 5

El jugador realiza un **Tetris** en el **Nivel 5**.

- Base: 1200 puntos.
- Multiplicador: 5.
- Cálculo: $1200 \times 5 = \mathbf{6,000}$ puntos en un solo movimiento.

Escenario:

- 1 Inicia Nivel 1.
- 2 Hace un *Tetris* (4 líneas).
- 3 Hace un *Double* (2 líneas).
- 4 Hace otro *Tetris* (4 líneas).

Total líneas: $4 + 2 + 4 = 10 \rightarrow$ ¡Sube a Nivel 2!

Cálculo de Puntos:

- 1 $1200 \times 1 = 1200$
- 2 $100 \times 1 = 100$
- 3 $1200 \times 1 = 1200$

Total: 2,500 pts

¿Por qué JSON?

Usamos JSON (*JavaScript Object Notation*) para guardar el **High Score**.

Estructura del Archivo

Archivo: `highscore.json`

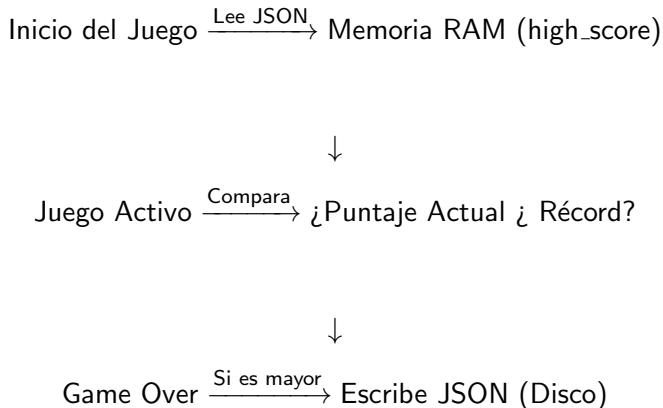
```
{  
    "high_score": 15600  
}
```

- **Legible:** Texto plano fácil de depurar.
- **Ligero:** Solo guarda lo esencial.
- **Nativo:** Python tiene la librería 'json' integrada.

Manejo de Archivos (I/O)

```
1 import json
2
3 def save_high_score(new_score):
4     """Guarda solo si supera el r cord anterior"""
5     current_record = load_high_score()
6
7     if new_score > current_record:
8         data = {"high_score": new_score}
9         try:
10             with open("highscore.json", "w") as f:
11                 json.dump(data, f)
12                 print(" Nuevo R cord Guardado!")
13         except IOError:
14             print("Error al guardar archivo.")
```

Flujo de Datos (Data Flow)



Eficiencia

Solo accedemos al disco duro al **inicio** y al **final** del juego, evitando lag durante la partida.

- ❶ **Escalabilidad:** El sistema de puntuación escala con la dificultad (Nivel).
- ❷ **Incentivos:** El diseño matemático fuerza al jugador a tomar riesgos (buscar el Tetris).
- ❸ **Persistencia:** JSON ofrece una solución robusta y portátil para guardar datos entre sesiones.

¿Preguntas?