Desarrollo de un Clon de Tetris: Lógica, Matemáticas e Interfaz Gráfica

Proyecto Final

Carlos Alberto Bello Rosas

Ingeniería en Software
Materia: Introducción a la Programación y Laboratorio
Universidad Autónoma de Zacatecas

14 de diciembre de 2025

Introducción y Objetivos

- **Objetivo:** Aplicar los conceptos fundamentales de la programación estructurada y orientada a eventos para recrear el clásico juego *Tetris*.
- **Tecnologías:**
  - Lenguaje: **Python** (por su legibilidad y manejo de estructuras de datos).
  - Librería Gráfica: **Tkinter** (para el renderizado del lienzo y manejo de inputs).
- **Enfoque:** El proyecto no solo es visual, sino que implementa lógica matemática vectorial para el movimiento y matrices para el estado del juego.

Modelo Matemático: El Tablero como Matriz

Para la computadora, el juego no son "bloques", son números.

El Tablero (T): Se modela como una matriz de dimensiones 20×10.

$$T = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & \cdots & 1 \end{bmatrix}_{20 \times 10}$$

Donde $T_{i,j} = 0$ representa un espacio vacío y $T_{i,j} \neq 0$ un bloque ocupado.

Física del Juego: Vectores de Traslación

Cada pieza activa tiene una posición definida por un vector de coordenadas $\vec{P}$:

$$\vec{P} = \begin{bmatrix} x \\ y \end{bmatrix}$$

El movimiento se calcula mediante **suma de vectores**:

Movimiento Lateral:

$$\vec{P}_{nuevo} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} \Delta x \\ 0 \end{bmatrix}$$

Donde $\Delta x \in \{-1, 1\}$.

Gravedad:

$$\vec{P}_{nuevo} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

El eje Y crece hacia abajo en computación gráfica.

Implementación: Diccionarios y Estado

En lugar de usar objetos complejos, utilicé estructuras de datos eficientes de Python. La pieza actual es un *diccionario*:

```
current_piece = {
    'shape_index': 3,  # Identificador de la forma (T, L, Z ...)
    'rotation': 0,     # Estado de rotacion (0-3)
    'x': 5,            # Coordenada vectorial X
    'y': 0             # Coordenada vectorial Y
}

```

Esto permite acceder y modificar el estado del juego en tiempo constante $O(1)$ durante el bucle principal.

Rotación: Transformación vs. Pre-cálculo

Matemáticamente, rotar una pieza implica una **Matriz de Rotación** R_{90°:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Optimización de Ingeniería: Para evitar calcular multiplicaciones de matrices en cada frame (costoso), implementé un sistema de *Look-up Tables* (Tablas de búsqueda). Las rotaciones están pre-calculadas en una lista tridimensional constante.

Ventaja: Reducción drástica del uso de CPU.

Algoritmo de Colisiones

La función crítica `check_collision` determina la validez de un estado futuro mediante álgebra de conjuntos.

Un movimiento es inválido si:

1. $x < 0$ o $x \geq Ancho$ (Límites laterales).
2. $y \geq Alto$ (Suelo).
3. $Tablero[y][x] \neq 0$ (Intersección con piezas existentes).

Además, implementé **"Wall Kicks"**: Si una pieza rota y choca con la pared, el algoritmo la empuja vectorialmente hacia adentro para permitir la jugada, mejorando la experiencia de usuario.

Arquitectura: El Game Loop

Tkinter es basado en eventos, por lo que creé un bucle recursivo usando `.after()` para simular el tiempo real.

```
def game_loop(canvas):
    if game_over_flag or is_paused: return

    # 1. Aplicar vector de gravedad (0, 1)
    # 2. Verificar colisiones
    # 3. Renderizar cambios en el Canvas

    # Recursividad programada para controlar la velocidad
    window.after(get_game_speed(), game_loop, canvas)

```

Conclusiones y Aprendizaje

Resultados

Se logró una implementación funcional completa con sistema de puntuación, niveles, "pieza fantasmaz guardado de récords (JSON).

Aprendizajes clave:

- La importancia del **Álgebra Lineal** en el desarrollo de simulaciones y gráficos 2D.
- Manejo de **estructuras de datos** (matrices y diccionarios) para representar estados complejos.
- Diseño de algoritmos eficientes para la detección de colisiones en tiempo real.

¿Preguntas?