Proyecto Tetris en Python

Implementación Técnica y Lógica del Juego

Carlos Alberto Bello Rosas

Universidad Autónoma de Zacatecas
Ingeniería en Software

14 de diciembre de 2025

Guía para el expositor: Comienza saludando y presentándote.

Contenido de la Presentación

¿Qué es este proyecto?

Objetivo Principal

Desarrollar un clon completo del juego Tetris implementando:

- **Lenguaje**: Python 3
- **Interfaz**: Tkinter (GUI nativa)
- **Paradigma**: Programación Estructurada y Modular

Características Técnicas

- Lógica de matrices para el tablero (Grid System).
- Persistencia de datos (JSON) para High Scores.
- Algoritmos de detección de colisiones y rotación (SRS).

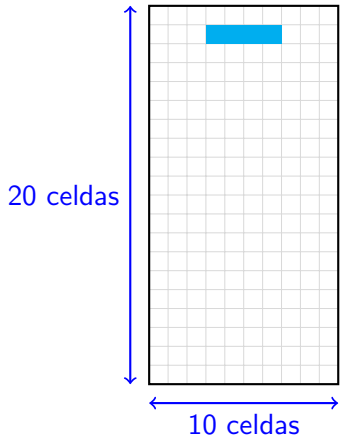
Arquitectura del Tablero (Grid)

Dimensiones Estándar

- **Matriz Lógica**: 10 columnas $\times$ 20 filas.
- **Resolución**: 300px $\times$ 600px.
- **Celda**: 30px $\times$ 30px.

¿Por qué 10x20?

Es el estándar de la *Tetris Guideline*. Permite suficiente espacio para maniobrar piezas de 4 bloques de ancho (Tetrominos) con márgenes de error.



Implementación: Constantes

```
# --- Constantes del Tablero ---
BOARD_WIDTH = 10      # Unidades l gicas
BOARD_HEIGHT = 20
SQUARE_SIZE = 30      # P xeles por bloque

# C lculo din mico de la ventana
GAME_WIDTH = BOARD_WIDTH * SQUARE_SIZE    # 300px
GAME_HEIGHT = BOARD_HEIGHT * SQUARE_SIZE  # 600px

# Dimensiones de la interfaz
SIDE_PANEL_WIDTH = 200
WINDOW_WIDTH = GAME_WIDTH + SIDE_PANEL_WIDTH
```

Algoritmo de Puntuación

Utilizamos un sistema exponencial basado en el nivel actual:

Líneas	**Tipo**	**Base**	**Fórmula**
1	Single	40	$40 \times (Nivel + 1)$
2	Double	100	$100 \times (Nivel + 1)$
3	Triple	300	$300 \times (Nivel + 1)$
4	**Tetris**	**1200**	$1200 \times (Nivel + 1)$

Incentivo al Jugador

El sistema premia el riesgo: hacer un "Tetris"(4 líneas) vale 30 veces más que una línea simple.

Código: Cálculo de Puntos

```python
def update_score_and_level(lines_cleared):
    global score, level, lines_cleared_count

    if lines_cleared == 0: return

    # 1. Obtener puntos base del diccionario
    # SCORE_VALUES = {1: 40, 2: 100, 3: 300, 4: 1200}
    base_points = SCORE_VALUES.get(lines_cleared, 0)

    # 2. Aplicar multiplicador de nivel
    score += base_points * level

    # 3. Verificar subida de nivel (cada 10 l neas)
    lines_cleared_count += lines_cleared
    if lines_cleared_count >= 10:
        level += 1
        lines_cleared_count -= 10 # Mantiene el residuo
        increase_speed() # Aumenta dificultad
```

Manejo de High Scores con JSON

Estructura JSON

```
{
    "high_score": 15600,
    "last_player": "Carlos"
}
```

- Formato ligero y estándar.
- Fácil de leer/editar para depuración.

```python
import json

def load_high_score():
    try:
        with open("data.json", "r") as f:
            data = json.load(f)
            return data.get("high_score",
                0)
    except FileNotFoundError:
        return 0 # Si es la primera vez
```

Matriz de Colisiones

Para detectar choques, verificamos 3 condiciones antes de mover una pieza:

```
def check_collision(shape, x, y):
    for row_idx, row in enumerate(shape):
        for col_idx, cell in enumerate(row):
            if cell != 0: # Si la celda de la pieza es s lida
                # Coordenadas en el tablero global
                board_x = x + col_idx
                board_y = y + row_idx

                # 1. L mites laterales
                if board_x < 0 or board_x >= BOARD_WIDTH: return True
                # 2. L mite inferior (suelo)
                if board_y >= BOARD_HEIGHT: return True
                # 3. Colisi n con piezas existentes
                if board_state[board_y][board_x] != 0: return True
    return False
```

Rotación y "Wall Kicks"

La rotación no es solo visual, implica transponer la matriz de la pieza.

```
def rotate_piece():
    # Algoritmo de Wall Kick simplificado
    new_rotation = calculate_next_rotation()

    # Intenta rotar en posici n actual (0,0)
    if not check_collision(new_rotation, x, y):
        apply_rotation()
        return

    # Si choca, intenta mover la pieza (Kick)
    # Izquierda, Derecha, Arriba
    for kick in [(-1,0), (1,0), (0,-1)]:
        if not check_collision(new_rotation, x+kick[0], y+kick[1]):
            apply_rotation_with_kick(kick)
            return
```

Resumen Técnico

Logros

- Motor de juego funcional en Python.
- Implementación correcta de matrices.
- Persistencia de datos exitosa.

Características Avanzadas

- Sistema Hold (guardar pieza).
- Hard Drop (caída instantánea).
- Ghost Piece (sombra guía).
- Algoritmo SRS (Super Rotation System).

¡Gracias!

¿Preguntas?