Tetris en Python con Tkinter

Carlos Alberto Bello R.

November 20, 2025

Contenido

1. Introducción
2. Constantes y Configuración
3. Variables de Estado
4. Funciones del Juego
5. Bucle del Juego y Dibujo
6. Configuración e Inicio
7. Resumen

Introducción

- Implementación del clásico juego Tetris en Python.
- Uso de Tkinter para la interfaz gráfica.
- Programación estructurada.
- Persistencia de datos (récords en JSON).
- Control con teclado.

Constantes del Juego

```
1 import tkinter as tk
2 import random
3
4 # Constantes del juego
5 BOARD_WIDTH = 10     # Ancho del tablero
6 BOARD_HEIGHT = 20    # Alto del tablero
7 SQUARE_SIZE = 30     # Tama o de cada bloque
```

- `BOARD_WIDTH`: Número de bloques horizontales.
- `BOARD_HEIGHT`: Número de bloques verticales.
- `SQUARE_SIZE`: Tamaño en píxeles de cada bloque.

Colores de las Piezas

```
PIECE_COLORS = [
    "#1a1a1a",  # 0 - Vac o (negro)
    "#00f0f0",  # 1 - I (Cian)
    "#f0f000",  # 2 - O (Amarillo)
    "#a000f0",  # 3 - T (Morado)
    "#00f000",  # 4 - S (Verde)
    "#f00000",  # 5 - Z (Rojo)
    "#0000f0",  # 6 - J (Azul)
    "#f0a000"   # 7 - L (Naranja)
]
```

- Cada índice corresponde a un tipo de pieza.
- El índice 0 es el color de fondo (vacío).

Formas de las Piezas (I)

```
PIECE_SHAPES = [
    [], # 0. Vac o
    # Pieza I (2 rotaciones)
    [
        [[0,0,0,0], [1,1,1,1], [0,0,0,0], [0,0,0,0]],
        [[0,1,0,0], [0,1,0,0], [0,1,0,0], [0,1,0,0]]
    ],
    # Pieza O (1 rotaci n)
    [[[2,2], [2,2]]],
    # Pieza T (4 rotaciones)
    [
        [[0,3,0], [3,3,3], [0,0,0]],
        [[0,3,0], [0,3,3], [0,3,0]],
        [[0,0,0], [3,3,3], [0,3,0]],
        [[0,3,0], [3,3,0], [0,3,0]]
    ],
    # ... (otras piezas)
]
```

Variables de Estado Global

```
1 board_state = []
2 current_piece = None
3 game_over_flag = False
```

- `board_state`: Matriz que representa el tablero.
- `current_piece`: Diccionario con la pieza actual.
- `game_over_flag`: Indica si el juego ha terminado.

Creación del Tablero Vacío

```
1 def create_empty_board():
2     """Crea un tablero vac o"""
3     return [[0 for _ in range(BOARD_WIDTH)]
4             for _ in range(BOARD_HEIGHT)]
```

- Crea una matriz de BOARD_HEIGHT × BOARD_WIDTH.
- Inicializada con 0 (vacío).

Pieza Aleatoria

```python
def get_random_piece():
    """Devuelve una pieza aleatoria simple"""
    shape_index = random.randint(1, 3)  # Solo I, O y T para empezar
    return {
        'shape_index': shape_index,
        'rotation': 0,
        'x': BOARD_WIDTH // 2 - 2,
        'y': 0
    }
```

- shape_index: Tipo de pieza (1-3).
- rotation: Rotación inicial (0).
- x, y: Posición inicial (centro arriba).

Detección de Colisiones

```python
def check_collision(shape, x, y):
    for y_offset, row in enumerate(shape):
        for x_offset, cell_value in enumerate(row):
            if cell_value != 0:
                board_x = x + x_offset
                board_y = y + y_offset
                # Verificar bordes
                if board_x < 0 or board_x >= BOARD_WIDTH:
                    return True
                if board_y >= BOARD_HEIGHT:
                    return True
                # Verificar otras piezas
                if board_y >= 0 and board_state[board_y][
    board_x] != 0:
                    return True
    return False
```

Fijar Pieza en el Tablero

```
def fix_piece():
    """Fija la pieza actual en el tablero"""
    global board_state
    shape = get_current_shape()
    piece_x = current_piece['x']
    piece_y = current_piece['y']
    color_index = current_piece['shape_index']

    for y, row in enumerate(shape):
        for x, cell_value in enumerate(row):
            if cell_value != 0 and piece_y + y >= 0:
                board_state[piece_y + y][piece_x + x] =
    color_index
```

Crear Nueva Pieza

```
def create_new_piece():
    global current_piece, game_over_flag
    current_piece = get_random_piece()
    # Comprobar si la nueva pieza causa game over
    if check_collision(get_current_shape(),
                       current_piece['x'],
                       current_piece['y']):
        game_over_flag = True
        canvas.create_text(...)  # Muestra "GAME OVER"
```

Bucle Principal del Juego

```
def game_loop():
    if not game_over_flag:
        # Mover la pieza hacia abajo
        if current_piece:
            new_y = current_piece['y'] + 1
            if not check_collision(get_current_shape(),
                                   current_piece['x'], new_y):
                current_piece['y'] = new_y
            else:
                fix_piece()
                create_new_piece()
        draw_board()
    window.after(500, game_loop)  # Velocidad fija
```

Dibujo del Tablero

```
def draw_board():
    canvas.delete("all")  # Limpia la pantalla
    # Dibuja las piezas fijadas
    for y in range(BOARD_HEIGHT):
        for x in range(BOARD_WIDTH):
            if board_state[y][x] != 0:
                color = PIECE_COLORS[board_state[y][x]]
                # Dibuja rect ngulo en (x, y) con color
    # Dibuja la pieza actual
    if current_piece and not game_over_flag:
        shape = get_current_shape()
        color = PIECE_COLORS[current_piece['shape_index']]
        # Dibuja cada bloque de la pieza actual
```

Configuración de la Ventana

```
def setup_game():
    global window, canvas, board_state
    window = tk.Tk()
    window.title("Tetris - Primera Etapa")
    canvas = tk.Canvas(
        window,
        width=BOARD_WIDTH * SQUARE_SIZE,
        height=BOARD_HEIGHT * SQUARE_SIZE,
        bg="#1a1a1a"
    )
    canvas.pack()
    board_state = create_empty_board()
    create_new_piece()
    game_loop()
    window.mainloop()
```

Punto de Entrada

```
1 if __name__ == "__main__":
2     setup_game()
```

- El código se ejecuta solo si es el archivo principal.
- Llama a `setup_game()` para iniciar el juego.

Resumen y Aprendizajes

- **Programación estructurada**: Uso de funciones modulares.
- **Interfaz gráfica**: Tkinter y Canvas para dibujar.
- **Lógica de juego**: Colisiones, movimientos, estados.
- **Persistencia de datos**: Guardado de récords en JSON.
- **Manejo de eventos**: Teclado para controlar piezas.