📋 GUÍA DE OPTIMIZACIÓN DE RENDERIZADO - PrimerJuego2D

🎯 Objetivo

Mejorar el rendimiento del juego escalando imágenes UNA SOLA VEZ al cargarlas (en lugar de escalarlas en cada frame), manteniendo la nitidez del pixel art.

📊 PASO 1: Sistema de Medición de Rendimiento (Debug)

1.1 Modificar `PanelJuego.java`

Agregar variables de clase:

```
// Después de la línea: public UI ui = new UI(this);

// Sistema de medición de rendimiento
private boolean checkDrawTime = true; // Cambiar a false para
        producción
private long drawStart;
private long drawTime;
```

Modificar el método `paintComponent`:

```
public void paintComponent(Graphics g) {
    super.paintComponent(g);

    // ⏱️ INICIO DE MEDICIÓN
    drawStart = System.nanoTime();

    Graphics2D g2 = (Graphics2D) g;

    // tiles
```

```
        tileManager.draw(g2);

        // objetos
        for (int i = 0; i < objs.length; i++) {
            if (objs[i] != null) {
                objs[i].draw(g2, this);
            }
        }

        // jugador
        jugador.draw(g2);

        // UI (HUD) - ¡SIEMPRE AL FINAL!
        ui.draw(g2);

        // ⏱️ FIN DE MEDICIÓN Y VISUALIZACIÓN
        drawTime = System.nanoTime() - drawStart;

        if (checkDrawTime) {
            g2.setColor(Color.WHITE);
            g2.drawString("Draw Time: " + drawTime + " ns", 10, 400);
            g2.drawString("Draw Time: " + (drawTime / 1000000.0) + "
             ms", 10, 420);
        }

        g2.dispose();
    }
```

Resultado esperado: Verás el tiempo de renderizado en pantalla. Después de la optimización, debería reducirse significativamente.

🛠️ PASO 2: Crear la Clase `UtilityTool`

2.1 Crear archivo `UtilityTool.java` en el paquete `main`

Ruta: `src/main/UtilityTool.java`

```
package main;

import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.image.BufferedImage;
```

```java
/**
 * Herramienta de utilidad para el escalado optimizado de
       imágenes.
 * Usa
       interpolación NEAREST_NEIGHBOR para mantener la nitidez
       del pixel art.
 */
public class UtilityTool {


    /**
     * Escala una imagen a un tamaño específico manteniendo la
        nitidez del pixel art.
     *
     * @param original - Imagen original a escalar
     * @param width - Ancho deseado
     * @param height - Alto deseado
     * @return BufferedImage escalada
     */
    public BufferedImage scaleImage(BufferedImage original, int
        width, int height) {

        // 1. Crear nueva imagen en blanco del tamaño deseado
        BufferedImage scaledImage = new BufferedImage(width,
        height, original.getType());

        // 2. Obtener el contexto gráfico
        Graphics2D g2 = scaledImage.createGraphics();

        // 3. 🔑 PASO CRÍTICO: Configurar interpolación para pixel
        art
        // Esto evita el efecto borroso en imágenes pequeñas
        escaladas
        g2.setRenderingHint(RenderingHints.KEY_INTERPOLATION,

        RenderingHints.VALUE_INTERPOLATION_NEAREST_NEIGHBOR);

        // 4. Dibujar la imagen original en el nuevo tamaño
        g2.drawImage(original, 0, 0, width, height, null);

        // 5. Liberar recursos
        g2.dispose();

        // 6. Retornar imagen escalada
        return scaledImage;
    }
}
```

🗺️ PASO 3: Refactorizar `TileManager`

3.1 Modificar `TileManager.java`

Agregar instancia de `UtilityTool` como variable de clase:

```
// Después de: public int mapaPorNumeroTile[][];

private UtilityTool uTool = new UtilityTool();
```

Crear método privado `setup`:

```
/**
 * Configura un tile: carga su imagen, la escala una sola vez y
        configura colisión.
 *
 * @param indice - Índice del tile en el array
 * @param rutaImagen - Ruta al recurso de imagen
 * @param colision - true si el tile tiene colisión
 */
private void setup(int indice, String rutaImagen, boolean
        colision) {
    try {
        // 1. Cargar imagen original (pequeña)
        BufferedImage imagenOriginal =
        ImageIO.read(getClass().getResource(rutaImagen));

        // 2. Escalar UNA SOLA VEZ usando UtilityTool
        tiles[indice].imagen = uTool.scaleImage(imagenOriginal,
        pj.tamanioTile, pj.tamanioTile);

        // 3. Configurar colisión
        tiles[indice].colision = colision;

        // 4. La imagenOriginal será eliminada por el Garbage
        Collector automáticamente

    } catch (IOException e) {
        System.err.println("Error al cargar imagen: " +
        rutaImagen);
        e.printStackTrace();
    }
}
```

Refactorizar método `getImagenTile`:

```java
public void getImagenTile(String rutaTiles) {
    try {
        InputStream is =
        getClass().getResourceAsStream(rutaTiles);
        BufferedReader br = new BufferedReader(new
        InputStreamReader(is));

        // Contar tiles
        int numTiles = 0;
        String linea;
        while ((linea = br.readLine()) != null) {
            if (!linea.trim().isEmpty()) {
                numTiles++;
            }
        }
        br.close();
        tiles = new Tile[numTiles];

        // Reiniciar stream
        is = getClass().getResourceAsStream(rutaTiles);
        br = new BufferedReader(new InputStreamReader(is));

        int indice = 0;
        while ((linea = br.readLine()) != null) {
            if (!linea.trim().isEmpty()) {
                String[] parametros = linea.split(";");

                if (parametros.length >= 2) {
                    tiles[indice] = new Tile();

                    // 🔄 CAMBIO PRINCIPAL: Usar el método setup
                    boolean tieneColision =
        parametros[1].trim().equals("1");
                    setup(indice, parametros[0], tieneColision);

                    indice++;
                }
            }
        }
        br.close();

    } catch (IOException e) {
        System.err.println("Error al leer rutaTiles: " +
        rutaTiles);
        e.printStackTrace();
    } catch (Exception e) {
```

```
            System.err.println("Error general en getImagenTile:");
            e.printStackTrace();
        }
    }
```

Modificar método `draw` (ELIMINAR parámetros de tamaño):

```
    public void draw(Graphics2D g2) {
        int worldCol = 0;
        int worldFila = 0;

        while (worldCol < pj.maxWorldcol && worldFila <
            pj.maxWorldfilas) {

            int numeroTile = mapaPorNumeroTile[worldCol][worldFila];

            int worldX = worldCol * pj.tamanioTile;
            int worldY = worldFila * pj.tamanioTile;
            int screenX = worldX - pj.jugador.worldx +
            pj.jugador.screenX;
            int screenY = worldY - pj.jugador.worldy +
            pj.jugador.screeny;

            // Culling de tiles fuera de pantalla
            if (worldX + pj.tamanioTile > pj.jugador.worldx -
            pj.jugador.screenX &&
                worldX - pj.tamanioTile < pj.jugador.worldx +
            pj.jugador.screenX &&
                worldY + pj.tamanioTile > pj.jugador.worldy -
            pj.jugador.screeny &&
                worldY - pj.tamanioTile < pj.jugador.worldy +
            pj.jugador.screeny) {

                // ✅ OPTIMIZACIÓN: Sin parámetros de tamaño, la
            imagen ya está escalada
                g2.drawImage(tiles[numeroTile].imagen, screenX,
            screenY, null);
            }

            worldCol++;
            if (worldCol == pj.maxWorldcol) {
                worldCol = 0;
                worldFila++;
            }
        }
    }
```

🎮 PASO 4: Refactorizar Clase `Jugador`

4.1 Modificar `Jugador.java`

Agregar instancia de `UtilityTool` como variable de clase:

```
// Después de: keyHandler kh;

private UtilityTool uTool = new UtilityTool();
```

Refactorizar método `getImagenDelJugador`:

```
public void getImagenDelJugador() {
    try {
        // Cargar imágenes originales y escalarlas UNA SOLA VEZ
        arriba1 = uTool.scaleImage(
            ImageIO.read(getClass().getResourceAsStream("/jugador/
        arriba_0001.png")),
            pj.tamanioTile, pj.tamanioTile
        );
        arriba2 = uTool.scaleImage(
            ImageIO.read(getClass().getResourceAsStream("/jugador/
        arriba_0002.png")),
            pj.tamanioTile, pj.tamanioTile
        );
        abajo1 = uTool.scaleImage(
            ImageIO.read(getClass().getResourceAsStream("/jugador/
        abajo_0001.png")),
            pj.tamanioTile, pj.tamanioTile
        );
        abajo2 = uTool.scaleImage(
            ImageIO.read(getClass().getResourceAsStream("/jugador/
        abajo_0002.png")),
            pj.tamanioTile, pj.tamanioTile
        );
        derecha1 = uTool.scaleImage(
            ImageIO.read(getClass().getResourceAsStream("/jugador/
        derecha_0001.png")),
            pj.tamanioTile, pj.tamanioTile
        );
        derecha2 = uTool.scaleImage(
            ImageIO.read(getClass().getResourceAsStream("/jugador/
        derecha_0002.png")),
            pj.tamanioTile, pj.tamanioTile
        );
        derecha3 = uTool.scaleImage(
```

```
            ImageIO.read(getClass().getResourceAsStream("/jugador/
        derecha_0003.png")),
            pj.tamanioTile, pj.tamanioTile
        );
        izquierda1 = uTool.scaleImage(
            ImageIO.read(getClass().getResourceAsStream("/jugador/
        izquierda_0001.png")),
            pj.tamanioTile, pj.tamanioTile
        );
        izquierda2 = uTool.scaleImage(
            ImageIO.read(getClass().getResourceAsStream("/jugador/
        izquierda_0002.png")),
            pj.tamanioTile, pj.tamanioTile
        );
        izquierda3 = uTool.scaleImage(
            ImageIO.read(getClass().getResourceAsStream("/jugador/
        izquierda_0003.png")),
            pj.tamanioTile, pj.tamanioTile
        );

    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Modificar método `draw` (ELIMINAR parámetros de tamaño):

```
public void draw(Graphics2D g2) {

    BufferedImage imagen = null;

    switch (direccion) {
    case "arriba":
        if (numeroSpites == 1) imagen = arriba1;
        if (numeroSpites == 2) imagen = arriba2;
        if (numeroSpites == 3) imagen = arriba2;
        break;

    case "abajo":
        if (numeroSpites == 1) imagen = abajo1;
        if (numeroSpites == 2) imagen = abajo2;
        if (numeroSpites == 3) imagen = abajo2;
        break;

    case "izquierda":
        if (numeroSpites == 1) imagen = izquierda1;
```

```
            if (numeroSpites == 2) imagen = izquierda3;
            if (numeroSpites == 3) imagen = izquierda2;
            break;

        case "derecha":
            if (numeroSpites == 1) imagen = derecha1;
            if (numeroSpites == 2) imagen = derecha3;
            if (numeroSpites == 3) imagen = derecha2;
            break;
        }

        // ✅ OPTIMIZACIÓN: Sin parámetros de tamaño
        g2.drawImage(imagen, screenX, screeny, null);

        // Hitbox debug
        if (debug) {
            g2.setColor(Color.RED);
            g2.drawRect(screenX + AreaSolida.x, screeny +
            AreaSolida.y,
                        AreaSolida.width, AreaSolida.height);
        }
    }
```

📦 PASO 5: Refactorizar Objetos

5.1 Modificar `superObjeto.java`

Agregar instancia de `UtilityTool`:

```
package objetos;

import java.awt.Graphics2D;
import java.awt.Rectangle;
import java.awt.image.BufferedImage;

import main.PanelJuego;
import main.UtilityTool; // ← NUEVO IMPORT

public class superObjeto {

    public BufferedImage imagen;
    public String nombre;
    public boolean colision;
```

```
    public int worldX, worldY;

    // Hitbox por defecto del tamaño del tile
    public Rectangle AreaSolida = new Rectangle(0, 0, 48, 48);
    public int AreaSolidaDefaultX = 0;
    public int AreaSolidaDefaultY = 0;

    // ✅ NUEVA instancia de UtilityTool
    protected UtilityTool uTool = new UtilityTool();

    public void draw(Graphics2D g2, PanelJuego pj) {

        int screenX = worldX - pj.jugador.worldx +
        pj.jugador.screenX;
        int screenY = worldY - pj.jugador.worldy +
        pj.jugador.screeny;

        // Culling de objetos fuera de pantalla
        if (worldX + pj.tamanioTile > pj.jugador.worldx -
        pj.jugador.screenX &&
            worldX - pj.tamanioTile < pj.jugador.worldx +
        pj.jugador.screenX &&
            worldY + pj.tamanioTile > pj.jugador.worldy -
        pj.jugador.screeny &&
            worldY - pj.tamanioTile < pj.jugador.worldy +
        pj.jugador.screeny) {

            // ✅ OPTIMIZACIÓN: Sin parámetros de tamaño
            g2.drawImage(imagen, screenX, screenY, null);
        }
    }
}
```

5.2 Modificar todos los objetos hijos

Ejemplo: `OBJ_llave.java`

```
package objetos;

import java.io.IOException;
import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;

public class OBJ_llave extends superObjeto {

    public OBJ_llave(int tamanioTile) { // ← NUEVO PARÁMETRO
```

```
        nombre = "llave";
        try {
            // 1. Cargar imagen original
            BufferedImage imagenOriginal =
        ImageIO.read(getClass().getResource("/objetos/
        llave.png"));

            // 2. ✅ Escalar UNA SOLA VEZ
            imagen = uTool.scaleImage(imagenOriginal,
        tamanioTile, tamanioTile);

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Aplicar el mismo patrón a: - `OBJ_puerta.java` - `OBJ_cofre.java` - `OBJ_botas.java`

Ejemplo genérico:

```
public OBJ_[NombreObjeto](int tamanioTile) {
    nombre = "[nombreObjeto]";
    try {
        BufferedImage imagenOriginal =
        ImageIO.read(getClass().getResource("/objetos/
        [archivo].png"));
        imagen = uTool.scaleImage(imagenOriginal, tamanioTile,
        tamanioTile);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

5.3 Actualizar `AssetSetter.java`

Debes pasar el parámetro `tamanioTile` al crear objetos:

```
// ANTES:
pj.objs[0] = new OBJ_llave();

// DESPUÉS:
pj.objs[0] = new OBJ_llave(pj.tamanioTile);
```

✅ PASO 6: Verificación y Pruebas

6.1 Checklist de Implementación

- [ ] ✅ Clase `UtilityTool` creada en `main/`
- [ ] ✅ `PanelJuego`: Sistema de medición agregado
- [ ] ✅ `TileManager`: Método `setup()` creado y usado
- [ ] ✅ `TileManager.draw()`: Parámetros de tamaño eliminados
- [ ] ✅ `Jugador`: Escalado en `getImagenDelJugador()`
- [ ] ✅ `Jugador.draw()`: Parámetros de tamaño eliminados
- [ ] ✅ `superObjeto`: `UtilityTool` agregado
- [ ] ✅ `superObjeto.draw()`: Parámetros de tamaño eliminados
- [ ] ✅ `OBJ_llave`, `OBJ_puerta`, `OBJ_cofre`, `OBJ_botas`: Constructores modificados
- [ ] ✅ `AssetSetter`: Parámetro `tamanioTile` agregado a instancias

6.2 Compilar y Ejecutar

```
# Compilar (desde la raíz del proyecto)
javac -d bin src/**/*.java

# Ejecutar
java -cp bin main.Main
```

6.3 Verificar Resultados

1. **Medición de rendimiento:** Observa el valor de “Draw Time” en pantalla
    - **Antes:** ~500,000 - 1,000,000 ns (~0.5-1 ms)
    - **Después:** ~100,000 - 300,000 ns (~0.1-0.3 ms)
    - **Mejora esperada:** 50-70% de reducción

2. **Calidad visual:** Los sprites deben verse **nítidos y pixelados** (no borrosos)

3. **Funcionamiento:** Movimiento, colisiones y recolección de objetos deben funcionar igual

---

📈 Resultados Esperados

Mejoras Técnicas:

- **Reducción del 50-70%** en tiempo de renderizado
- **Menor uso de CPU** durante el juego
- **FPS más estables** sin caídas

Mejoras Visuales:

- **Pixel art nítido** sin efecto borroso
- **Bordes definidos** en tiles y sprites

Gestión de Memoria:

- Las imágenes originales se descartan automáticamente
- Solo se mantienen en memoria las versiones escaladas
- Menor presión en el Garbage Collector

🔧 Troubleshooting

Problema: “Cannot find symbol: UtilityTool”

Solución: Verifica que el archivo esté en `src/main/UtilityTool.java` y que hayas importado `import main.UtilityTool;`

Problema: Imágenes se ven borrosas

Solución: Asegúrate de usar `RenderingHints.VALUE_INTERPOLATION_NEAREST_NEIGHBOR` en `UtilityTool`

Problema: NullPointerException al dibujar

Solución: Verifica que todas las imágenes se estén escalando correctamente en los constructores

Problema: Objetos no aparecen en pantalla

Solución: Asegúrate de pasar `pj.tamanioTile` al crear objetos en `AssetSetter`

📚 Conceptos Técnicos

¿Por qué es más rápido?

Antes: `ImageIO.read()` → guardar pequeña → `g2.drawImage(img, x, y, 64, 64, null)` [escalar en cada frame]

Después: `ImageIO.read()` → `uTool.scaleImage()` → guardar grande → `g2.drawImage(img, x, y, null)` [ya está escalada]

Interpolación NEAREST_NEIGHBOR

- **Bilinear/Bicubic:** Suaviza píxeles = borroso
- **Nearest Neighbor:** Copia píxeles exactos = nítido para pixel art

Gestión de Memoria

Las variables locales (como `imagenOriginal` en `setup()`) son elegibles para recolección de basura al salir del método, liberando memoria automáticamente.

🎓 Autor de la Optimización

Basado en técnicas de optimización de renderizado para juegos 2D en Java, adaptado específicamente para **PrimerJuego2D**.

Fecha de creación: 22 de diciembre de 2025