Guía de Implementación: Depuración y Mecánicas de Movimiento

Descripción General

Esta guía documenta tres implementaciones importantes en el proyecto de videojuego 2D: 1. **Visualización del área de colisión** (hitbox) para depuración 2. **Reset automático del sprite** al detenerse 3. **Movimiento basado en cuadrícula** (tile/grid based movement)

---

1. Visualización del Área de Colisión (Hitbox)

Objetivo

Facilitar la depuración mostrando visualmente el rectángulo que representa el área sólida del jugador en pantalla.

Implementación en Java

En la clase `Jugador`, dentro del método `draw()`:

```
public void draw(Graphics2D g2) {
    // Dibujar el sprite del jugador
    BufferedImage image = null;
    // ... código para seleccionar sprite según dirección ...

    g2.drawImage(image, screenX, screenY, gp.tamañoFinal,
        gp.tamañoFinal, null);

    // DEPURACIÓN: Dibujar hitbox
    g2.setColor(Color.RED);
    g2.drawRect(
```

```
            screenX + solidArea.x,     // Posición X en pantalla +
            offset del área sólida
            screenY + solidArea.y,     // Posición Y en pantalla +
            offset del área sólida
            solidArea.width,           // Ancho del área sólida
            solidArea.height           // Alto del área sólida
        );
}
```

Lógica de Coordenadas

- **Posición absoluta en pantalla:** `screenX + solidArea.x`, `screenY + solidArea.y`
- **Dimensiones:** Se toman directamente de `solidArea.width` y `solidArea.height`
- **Color:** Rojo para alta visibilidad durante debugging

Cuándo Usar

Esta visualización debe activarse solo durante el desarrollo. Se puede controlar con una variable booleana:

```
boolean debug = true; // Cambiar a false para producción

if (debug) {
    g2.setColor(Color.RED);
    g2.drawRect(screenX + solidArea.x, screenY + solidArea.y,
                solidArea.width, solidArea.height);
}
```

---

2. Reset del Sprite al Detenerse

Problema Detectado

El personaje quedaba “congelado” en un frame intermedio de la animación de caminata al soltar las teclas, resultando en una apariencia poco natural.

Solución

Implementar un sistema de retardo para volver suavemente al sprite de reposo (sprite 1).

Implementación en Java

En la clase `Jugador`, dentro del método `update()`:

```
public class Jugador extends Entidad {
    int standCounter = 0;  // Nueva variable de clase

    public void update() {
        if (keyH.upPressed || keyH.downPressed ||
            keyH.leftPressed || keyH.rightPressed) {

            // Lógica de movimiento y dirección
            // ... código existente ...

            // Animación de sprites durante movimiento
            spriteCounter++;
            if (spriteCounter > 12) {
                if (spriteNum == 1) {
                    spriteNum = 2;
                } else if (spriteNum == 2) {
                    spriteNum = 1;
                }
                spriteCounter = 0;
            }

        } else {
            // NUEVO: Lógica de retorno suave al sprite de reposo
            standCounter++;

            if (standCounter == 20) {  // Retardo de 20 frames
        (~0.33 segundos a 60fps)
                spriteNum = 1;         // Volver al sprite neutral
                standCounter = 0;       // Reiniciar contador
            }
        }
    }
}
```

Lógica del Contador de Retardo

1. **Incremento constante:** El contador aumenta en cada frame cuando no hay input
2. **Threshold:** Al alcanzar 20 frames (ajustable según preferencia)
3. **Reset visual:** Se fuerza `spriteNum = 1` (posición neutral)

4. **Reinicio:** El contador vuelve a 0 para futuros ciclos

Beneficio

Evita el “flickeo” o cambio brusco, creando una transición más natural y profesional.

---

3. Movimiento Basado en Cuadrícula (Tile-Based)

Concepto

Cambiar de movimiento libre (píxel a píxel) a movimiento rígido por baldosas, similar a: - Pokémon (Game Boy / Game Boy Advance) - Final Fantasy I-VI - Juegos RPG clásicos

El personaje siempre se mueve exactamente la distancia de una baldosa completa (48 píxeles).

Ajustes Necesarios

1. Modificar el Área Sólida

Reducir `solidArea` para evitar atascos en los bordes:

```
// En el constructor de Jugador
solidArea = new Rectangle();
solidArea.x = 1;          // 1 píxel de margen
solidArea.y = 1;          // 1 píxel de margen
solidArea.width = 46;     // 48 - 2 = 46 píxeles
solidArea.height = 46;    // 48 - 2 = 46 píxeles
```

2. Agregar Variables de Estado

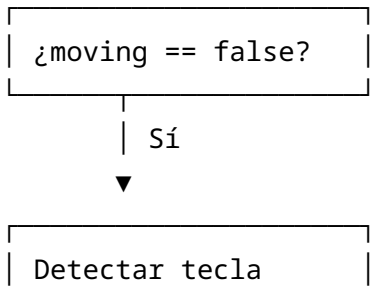
```
public class Jugador extends Entidad {
    boolean moving = false;        // Estado de movimiento activo
    int pixelCounter = 0;          // Píxeles recorridos en el
        movimiento actual
    int tileSize = 48;             // Tamaño de la baldosa
}
```

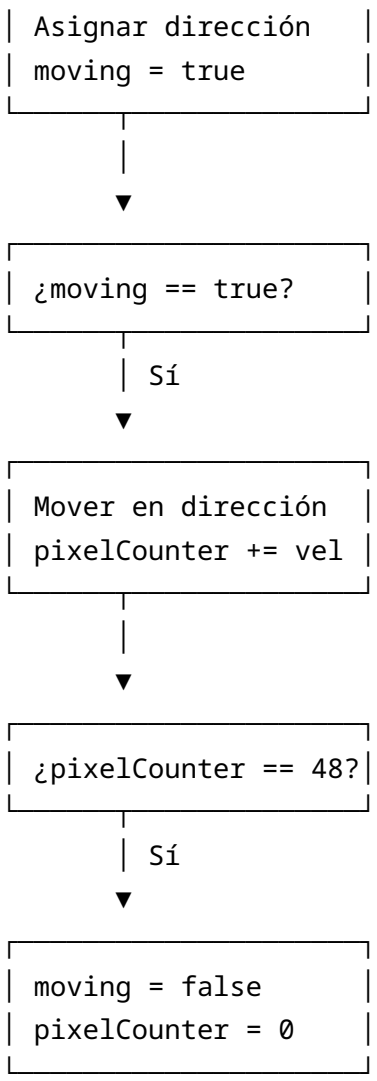
Implementación Completa

```
public void update() {

    // FASE 1: Detección de Input (solo si no está en movimiento)
    if (moving == false) {
        if (keyH.upPressed) {
            direccion = "arriba";
            moving = true;        // Iniciar movimiento
        }
        else if (keyH.downPressed) {
            direccion = "abajo";
            moving = true;
        }
        else if (keyH.leftPressed) {
            direccion = "izquierda";
            moving = true;
        }
        else if (keyH.rightPressed) {
            direccion = "derecha";
            moving = true;
        }
    }

    // FASE 2: Ejecución del Movimiento
    if (moving == true) {

        // Detección de colisiones
        colisionOn = false;
        gp.cChecker.checkTile(this);

        // Mover si no hay colisión
        if (colisionOn == false) {
            switch(direccion) {
                case "arriba":
                    worldY -= velocidad;
                    break;
                case "abajo":
                    worldY += velocidad;
                    break;
                case "izquierda":
                    worldX -= velocidad;
                    break;
                case "derecha":
```

```
                    worldX += velocidad;
                    break;
                }
            }

            // Incrementar contador de píxeles
            pixelCounter += velocidad;

            // FASE 3: Finalización del Movimiento
            if (pixelCounter == tileSize) {  // 48 píxeles completados
                moving = false;               // Permitir nuevo input
                pixelCounter = 0;             // Resetear contador
            }

            // Animación de sprites
            spriteCounter++;
            if (spriteCounter > 12) {
                if (spriteNum == 1) {
                    spriteNum = 2;
                } else if (spriteNum == 2) {
                    spriteNum = 1;
                }
                spriteCounter = 0;
            }

        } else {
            // Lógica de sprite en reposo (ver sección 2)
            standCounter++;
            if (standCounter == 20) {
                spriteNum = 1;
                standCounter = 0;
            }
        }
    }
```

Diagrama de Flujo Lógico





Ventajas del Sistema

1. **Control preciso:** El jugador siempre está alineado a la cuadrícula
2. **Prevención de glitches:** No hay posiciones intermedias problemáticas
3. **Facilita diseño de niveles:** Los obstáculos y objetos se alinean perfectamente
4. **Estilo retro:** Sensación nostálgica de RPGs clásicos

Consideraciones Importantes

- **Velocidad:** Debe ser divisor exacto del tamaño de baldosa (ej: 1, 2, 3, 4, 6, 8, 12, etc. para 48)
- **Colisiones:** Se verifican durante todo el movimiento, no solo al inicio
- **Input buffering:** Este sistema no permite cambios de dirección hasta completar el movimiento actual

---

Integración Completa

Estas tres características trabajan juntas para crear una experiencia de juego pulida:

1. **Hitbox visible** → Depurar problemas de colisión durante el desarrollo
2. **Sprite de reposo** → Animaciones más naturales y profesionales
3. **Movimiento por cuadrícula** → Gameplay preciso y predecible

Referencias Temporales (Video)

- Visualización hitbox: 00:56
- Sistema de sprite en reposo: 02:28
- Ajuste de solidArea: 06:20
- Lógica de movimiento por tiles: 07:09

Guía creada el 22 de diciembre de 2025