🎮 Guía de Transformación: De Juego de Búsqueda a Action RPG

Introducción

Esta guía documenta la transición del **PrimerJuego2D** desde un juego simple de recolección de llaves hacia un **Action RPG** completo. Los cambios fundamentales incluyen:

- Sistema de estados del juego (Game State)
- Sistema de pausa
- Reestructuración de la interfaz de usuario (UI)
- Limpieza y refactorización del código

> **Nota importante:** Usamos variables descriptivas como `playState` en lugar de "números mágicos" (1, 2) directamente en la lógica. Esto mejora la mantenibilidad y evita errores futuros.

📋 Índice

---

1. Implementación de Estados del Juego

📍 Archivo: `src/main/PanelJuego.java`

El concepto central es que el juego puede estar en diferentes "estados" (jugando, pausado, menú, diálogo, etc.) y el comportamiento debe variar según el estado actual.

1.1 Agregar Variables de Estado

Ubicación: Después de las declaraciones de variables existentes (alrededor de la línea 55)

Busca la sección donde están declaradas las variables de instancia:

```
// Sistema de sonido
Sound musica = new Sound();
Sound efectoSonido = new Sound();

// Interfaz de Usuario (HUD)
public UI ui = new UI(this);
```

Agregar después:

```
// ========================================
// SISTEMA DE ESTADOS DEL JUEGO (Game State)
// ========================================
// Constantes para identificar estados (evita "números mágicos")
public final int tituloState = 0;   // Pantalla de título
public final int playState = 1;     // Jugando
public final int pauseState = 2;    // Pausado
public final int dialogoState = 3;  // En diálogo (futuro)

// Estado actual del juego
public int gameState;
```

1.2 Inicializar el Estado en `setupJuego()`

Ubicación: Método `setupJuego()` (alrededor de la línea 65)

Estado actual:

```
public void setupJuego() {
    aSetter.setObjetct();
    reproducirMusicaFondo(0);
}
```

Modificar a:

```
/**
 * Configura el estado inicial del juego (coloca objetos, NPCs,
       etc).
 */
public void setupJuego() {
```

```
    aSetter.setObjetct();
    reproducirMusicaFondo(0);

    // Iniciar en estado de juego
    gameState = playState;
}
```

1.3 Modificar el Método `update()`

Ubicación: Método `update()` (alrededor de la línea 120)

Estado actual:

```
public void update() {
    jugador.update();
}
```

Modificar a:

```
/**
 * Actualiza la lógica del juego según el estado actual.
 */
public void update() {
    if (gameState == playState) {
        // ESTADO: JUGANDO
        jugador.update();
        // Aquí irán actualizaciones de NPCs, enemigos, etc.

    } else if (gameState == pauseState) {
        // ESTADO: PAUSADO
        // No se actualiza nada - el juego se "congela"
    }
}
```

🧠 ¿Por qué usar constantes?

```
// ❌ MAL - Números mágicos (difícil de mantener)
if (gameState == 1) { ... }

// ✅ BIEN - Variables descriptivas (fácil de entender)
if (gameState == playState) { ... }
```

---

2. Sistema de Pausa

📍 Archivo: `src/main/keyHandler.java`

2.1 Agregar Referencia a PanelJuego

El `keyHandler` necesita acceso a `PanelJuego` para modificar el estado del juego.

Estado actual:

```
public class keyHandler implements KeyListener {

    public boolean arribaPres, abajoPres, izqPres, drchPres;
```

Modificar a:

```
public class keyHandler implements KeyListener {

    // Referencia al panel principal del juego
    PanelJuego pj;

    public boolean arribaPres, abajoPres, izqPres, drchPres;

    /**
     * Constructor que recibe la referencia al PanelJuego.
     * Necesario para modificar el estado del juego desde aquí.
     */
    public keyHandler(PanelJuego pj) {
        this.pj = pj;
    }
```

2.2 Agregar Lógica de Pausa en `keyPressed()`

Ubicación: Dentro del método `keyPressed()`, después de los controles de movimiento

Estado actual:

```
@Override
public void keyPressed(KeyEvent e) {
    int keycode = e.getKeyCode();

    if (keycode == KeyEvent.VK_W) {
```

```
            arribaPres = true;
        }
        if (keycode == KeyEvent.VK_S) {
            abajoPres = true;
        }
        if (keycode == KeyEvent.VK_A) {
            izqPres = true;
        }
        if (keycode == KeyEvent.VK_D) {
            drchPres = true;
        }
    }
```

Modificar a:

```
    @Override
    public void keyPressed(KeyEvent e) {
        int keycode = e.getKeyCode();

        // ========================================
        // ESTADO: JUGANDO (playState)
        // ========================================
        if (pj.gameState == pj.playState) {

            // Controles de movimiento
            if (keycode == KeyEvent.VK_W) {
                arribaPres = true;
            }
            if (keycode == KeyEvent.VK_S) {
                abajoPres = true;
            }
            if (keycode == KeyEvent.VK_A) {
                izqPres = true;
            }
            if (keycode == KeyEvent.VK_D) {
                drchPres = true;
            }

            // Tecla P - Pausar juego
            if (keycode == KeyEvent.VK_P) {
                pj.gameState = pj.pauseState;
            }
        }
        // ========================================
        // ESTADO: PAUSADO (pauseState)
```

```
    // ========================================
    else if (pj.gameState == pj.pauseState) {

        // Tecla P - Reanudar juego
        if (keycode == KeyEvent.VK_P) {
            pj.gameState = pj.playState;
        }
    }
}
```

2.3 Actualizar la Creación del keyHandler en PanelJuego

Ubicación: `src/main/PanelJuego.java` (alrededor de la línea 41)

Estado actual:

```
// lsitener
keyHandler kh = new keyHandler();
```

Modificar a:

```
// listener (manejador de teclas)
public keyHandler kh = new keyHandler(this);
```

> **Importante:** Ahora `keyHandler` necesita recibir `this` (la instancia de `PanelJuego`) en su constructor.

---

3. Gestión de Interfaz de Usuario

📍 Archivo: `src/main/UI.java`

La UI debe responder al estado del juego, mostrando diferentes pantallas según el contexto.

3.1 Eliminar Variables del Juego Anterior

Ubicación: Variables de clase (líneas 15-30)

Eliminar estas variables y mantener solo las necesarias:

```
// === ELIMINAR ESTAS LÍNEAS ===
BufferedImage imagenLlave;
public boolean mensajeActivo = false;
```

```
public String mensaje = "";
int contadorMensaje = 0;
public boolean juegoTerminado = false;
double tiempoJuego;
DecimalFormat formatoDecimal = new DecimalFormat("#0.00");
```

Nueva estructura de variables:

```
public class UI {

    PanelJuego pj;
    Graphics2D g2;

    // Fuentes del juego
    Font arial_40;
    Font arial_80B;

    public UI(PanelJuego pj) {
        this.pj = pj;
        arial_40 = new Font("Arial", Font.PLAIN, 40);
        arial_80B = new Font("Arial", Font.BOLD, 80);
    }
```

3.2 Reestructurar el Método `draw()`

Estado actual del método `draw()`:

```
public void draw(Graphics2D g2) {
    if (juegoTerminado == true) {
        dibujarPantallaFin(g2);
    } else {
        dibujarHUD(g2);
    }
}
```

Modificar a:

```
/**
 * Método principal de dibujado.
 * Dibuja diferentes pantallas según el estado del juego.
 */
public void draw(Graphics2D g2) {

    this.g2 = g2;
```

```
        g2.setFont(arial_40);
        g2.setColor(Color.WHITE);

        // Dibujar según el estado actual
        if (pj.gameState == pj.playState) {
            // Estado: Jugando - Dibujar HUD del juego
            // (Por ahora vacío, se llenará con stats del Action RPG)

        } else if (pj.gameState == pj.pauseState) {
            // Estado: Pausado - Dibujar pantalla de pausa
            dibujarPantallaPausa();
        }
    }
```

3.3 Crear el Método `dibujarPantallaPausa()`

Agregar este nuevo método:

```
    /**
     * Dibuja la pantalla de pausa con el texto "PAUSED" centrado.
     */
    public void dibujarPantallaPausa() {

        g2.setFont(arial_80B);
        g2.setColor(Color.WHITE);

        String texto = "PAUSED";

        int x = obtenerXCentrado(texto);
        int y = pj.altoPantalla / 2;

        g2.drawString(texto, x, y);
    }
```

3.4 Simplificar el Método `obtenerXCentrado()`

El método existente ya está bien, pero lo modificamos para usar el `g2` de instancia:

Estado actual:

```
    public int obtenerXCentrado(String texto, Graphics2D g2) {
        int longitudTexto = (int)
            g2.getFontMetrics().getStringBounds(texto, g2).getWidth();
```

```
    return (pj.anchoPantalla / 2) - (longitudTexto / 2);
}
```

Modificar a:

```
/**
 * Calcula la coordenada X para centrar un texto en pantalla.
 * @param texto - El texto a centrar
 * @return La coordenada X donde debe dibujarse
 */
public int obtenerXCentrado(String texto) {
    int longitudTexto = (int)
        g2.getFontMetrics().getStringBounds(texto, g2).getWidth();
    return (pj.anchoPantalla / 2) - (longitudTexto / 2);
}
```

3.5 Versión Final Limpia de UI.java

```
package main;

import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics2D;

public class UI {

    PanelJuego pj;
    Graphics2D g2;

    // Fuentes del juego
    Font arial_40;
    Font arial_80B;

    public UI(PanelJuego pj) {
        this.pj = pj;
        arial_40 = new Font("Arial", Font.PLAIN, 40);
        arial_80B = new Font("Arial", Font.BOLD, 80);
    }

    public void draw(Graphics2D g2) {

        this.g2 = g2;

        g2.setFont(arial_40);
        g2.setColor(Color.WHITE);
```

```
        if (pj.gameState == pj.playState) {
            // HUD del juego (vacío por ahora)

        } else if (pj.gameState == pj.pauseState) {
            dibujarPantallaPausa();
        }
    }

    public void dibujarPantallaPausa() {

        g2.setFont(arial_80B);
        g2.setColor(Color.WHITE);

        String texto = "PAUSED";

        int x = obtenerXCentrado(texto);
        int y = pj.altoPantalla / 2;

        g2.drawString(texto, x, y);
    }

    public int obtenerXCentrado(String texto) {
        int longitudTexto = (int)
        g2.getFontMetrics().getStringBounds(texto, g2).getWidth();
        return (pj.anchoPantalla / 2) - (longitudTexto / 2);
    }
}
```

---

4. Limpieza de Código

📍 Archivo: `src/entidad/Jugador.java`

4.1 Eliminar el Contador de Llaves

Estado actual (alrededor de la línea 27):

```
public int numeroLlaves = 0; // Contador de llaves recolectadas
```

Acción: Eliminar esta línea (ya no necesitamos el sistema de llaves).

4.2 Eliminar/Simplificar el Método `recogerObjeto()`

Estado actual:

```
public void recogerObjeto(int index) {
    if (index != 999) {
        String nombreObjeto = pj.objs[index].nombre;
        switch (nombreObjeto) {
        case "llave":
            pj.playSE(1);
            numeroLlaves++;
            pj.objs[index] = null;
            pj.ui.mostrarMensaje("encontraste 1 llave");
            break;
        case "puerta":
            if (numeroLlaves > 0) {
                pj.playSE(3);
                pj.objs[index] = null;
                numeroLlaves--;
                pj.ui.mostrarMensaje("Llaves restantes: " +
        numeroLlaves);
            } else {
                System.out.println("oe busca las llaves ps");
            }
            break;
        case "cofre":
            pj.playSE(4);
            pj.objs[index] = null;
            pj.ui.juegoTerminado = true;
            pj.stopMusic();
            break;
        case "botas":
            pj.playSE(2);
            vel += 4;
            pj.objs[index] = null;
            pj.ui.mostrarMensaje("¡Velocidad aumentada
        loquito !");
            break;
        }
    }
}
```

Simplificar a (base para el Action RPG):

```
/**
 * Maneja la interacción con objetos del mundo.
```

```
     * @param index - índice del objeto en el array pj.obj[]
     */
    public void recogerObjeto(int index) {
        // Por ahora vacío - se implementará el nuevo sistema de
            objetos del Action RPG
    }
```

4.3 Limpiar las Referencias en `update()`

En el método `update()`, mantener la llamada a `recogerObjeto()` pero el método estará vacío:

```
// Verificar colisión con objetos
int objIndex = pj.dColisiones.checkObjeto(this, true);
recogerObjeto(objIndex);
```

📍 Archivo: `src/main/AssetSetter.java`

4.4 Limpiar la Colocación de Objetos

Estado actual:

```
public void setObjetct() {
    pj.objs[0] = new OBJ_llave(pj.tamanioTile);
    pj.objs[0].worldX = 24 * pj.tamanioTile;
    pj.objs[0].worldY = 12 * pj.tamanioTile;

    // ... más objetos ...
}
```

Limpiar a:

```
package main;

/**
 * Encargada de colocar los objetos, NPCs y enemigos en el mapa
        del juego.
 */
public class AssetSetter {
    PanelJuego pj;

    public AssetSetter(PanelJuego pj) {
        this.pj = pj;
    }
```

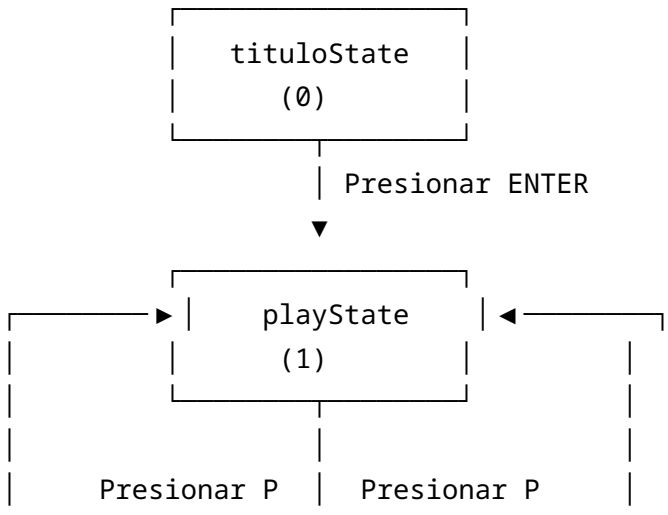
```
    /**
     * Instancia y posiciona los objetos en el mapa.
     * (Vacío por ahora - se llenará con objetos del Action RPG)
     */
    public void setObjetct() {
        // Los objetos del Action RPG se agregarán aquí
    }
}
```

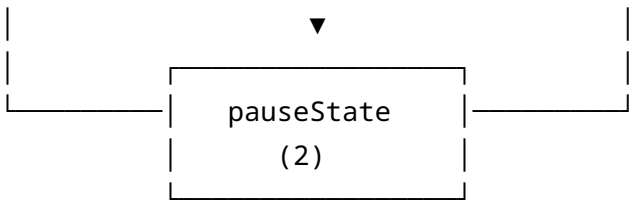
---

5. Resumen de Archivos Modificados

Archivo	Cambios Principales
`PanelJuego.java`	+ Variables de estado (`gameState`, `playState`, etc.) + Lógica condicional en `update()`
`keyHandler.java`	+ Constructor con `PanelJuego` + Lógica de pausa (tecla P)
`UI.java`	- Eliminar variables del juego anterior + Método `dibujarPantallaPausa()` + Lógica condicional en `draw()`
`Jugador.java`	- Eliminar `numeroLlaves` - Limpiar `recogerObjeto()`
`AssetSetter.java`	- Eliminar colocación de objetos viejos

🎯 Diagrama del Sistema de Estados





⚠️ Errores Comunes

Error 1: NullPointerException en keyHandler

```
java.lang.NullPointerException: Cannot read field "gameState"
```

Causa: El `keyHandler` se crea antes que `PanelJuego` termine de inicializarse. **Solución:** Asegúrate que la línea `keyHandler kh = new keyHandler(this);` esté DESPUÉS de las declaraciones de estado.

Error 2: El juego no pausa

Causa: La referencia `pj` en `keyHandler` es `null`. **Solución:** Verifica que el constructor reciba correctamente la instancia de `PanelJuego`.

Error 3: Pantalla negra al pausar

Causa: El método `paintComponent()` no dibuja cuando está pausado. **Solución:** Asegúrate que `paintComponent()` siga dibujando los tiles y la UI (solo `update()` se congela).

📝 Próximos Pasos (Action RPG)

Con el sistema de estados implementado, los siguientes pasos típicos serían:

1. **Sistema de combate** - Atacar, defender, recibir daño
2. **Sistema de stats** - HP, MP, ATK, DEF
3. **Sistema de inventario** - Objetos equipables
4. **NPCs y diálogos** - Usando `dialogoState`
5. **Enemigos con IA** - Patrullaje, persecución, ataque
6. **Sistema de niveles** - Experiencia y subida de nivel

🔧 Comandos de Prueba

Para verificar que todo funciona:

1. Ejecuta el juego
2. El jugador debe poder moverse con WASD
3. Presiona **P** → El juego debe pausarse (texto “PAUSED” centrado)
4. Presiona **P** de nuevo → El juego debe reanudarse
5. El jugador debe poder moverse nuevamente

> **Autor:** Guía generada para el proyecto PrimerJuego2D **Versión:** 1.0 - Sistema de Estados y Pausa **Fecha:** Diciembre 2025