# 🎮 Guía de Implementación: Sistema de NPCs (Personajes No Jugadores)

> **Objetivo:** Crear un "Viejo" (Old Man) que se mueva por el mapa e interactúe con el jugador.
>
> **Basado en:** Tutorial de RyiSnow - Video

---

## 📋 Estado Actual del Proyecto

Antes de implementar NPCs, así está estructurado tu proyecto:

### Clases Existentes:

| Clase | Paquete | Descripción |
|---|---|---|
| `Entidad.java` | entidad | Clase base con atributos comunes (posición, sprites, colisión) |
| `Jugador.java` | entidad | Extiende Entidad, maneja input y movimiento del jugador |
| `PanelJuego.java` | main | Panel principal, bucle de juego, renderizado |
| `detectorColisiones.java` | main | Sistema de colisiones (tiles y objetos) |
| `AssetSetter.java` | main | Coloca objetos en el mapa |
| `UtilityTool.java` | main | Utilidades para escalar imágenes |

### Variables Importantes de tu código actual:

```
// En Entidad.java
public int worldx, worldy;           // Posición en el mundo
public int vel;                       // Velocidad
public String direccion;              // "arriba", "abajo",
        "izquierda", "derecha"
public boolean hayColision = false;   // Flag de colisión

// En PanelJuego.java
public final int tamanioTile = 64;    // Tamaño de tile (32 * 2)
public Jugador jugador;               // Instancia del jugador
public superObjeto[] objs;            // Array de objetos
```

---

# 🔧 FASE 1: Refactorizar la Clase `Entidad`

## 1.1 Agregar `PanelJuego` al Constructor de `Entidad`

**Archivo:** `src/entidad/Entidad.java`

**¿Por qué?** Para que todas las entidades (Jugador, NPCs, monstruos) tengan acceso al panel de juego sin declararlo repetidamente.

```
package entidad;

import java.awt.Graphics2D;
import java.awt.Rectangle;
import java.awt.image.BufferedImage;
import java.io.IOException;
import javax.imageio.ImageIO;
import main.PanelJuego;
import main.UtilityTool;

public class Entidad {

    // ✅ NUEVO: Referencia al panel de juego
    public PanelJuego pj;

    public int worldx, worldy;
    public int vel;

    public BufferedImage arriba1, arriba2, abajo1, abajo2,
                         izquierda1, izquierda2, izquierda3,
```

```
                    derecha1, derecha2, derecha3;
public String direccion;

public int contadorSpites = 0;
public int numeroSpites = 1;

public Rectangle AreaSolida;
public int AreaSolidaDefaultX;
public int AreaSolidaDefaultY;

public boolean hayColision = false;

// ✅ NUEVO: Contador para bloquear cambios de acción (para IA
    de NPCs)
public int actionLockCounter = 0;

// ✅ NUEVO: Constructor que recibe PanelJuego
public Entidad(PanelJuego pj) {
    this.pj = pj;
}

// ✅ NUEVO: Método genérico para cargar y escalar imágenes
public BufferedImage setup(String rutaImagen) {
    UtilityTool uTool = new UtilityTool();
    BufferedImage imagen = null;

    try {
        imagen =
    ImageIO.read(getClass().getResourceAsStream(rutaImagen));
        imagen = uTool.escalarImagen(imagen, pj.tamanioTile,
    pj.tamanioTile);
    } catch (IOException e) {
        e.printStackTrace();
    }

    return imagen;
}

// ✅ NUEVO: Método que cada NPC sobreescribirá con su IA
public void setAction() {
    // Vacío - las subclases lo implementarán
}

// ✅ NUEVO: Método update para NPCs (el jugador tiene su
    propia lógica)
public void update() {
```

```
        setAction(); // Ejecutar IA del NPC

        // Detección de colisiones
        hayColision = false;
        pj.dColisiones.chektile(this);
        pj.dColisiones.checkObjeto(this, false);
        pj.dColisiones.checkJugador(this);  // Nuevo método que
        crearemos

        // Si no hay colisión, mover
        if (hayColision == false) {
            switch (direccion) {
            case "arriba":
                worldy -= vel;
                break;
            case "abajo":
                worldy += vel;
                break;
            case "izquierda":
                worldx -= vel;
                break;
            case "derecha":
                worldx += vel;
                break;
            }
        }

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

    // ✅ NUEVO: Método draw para NPCs
    public void draw(Graphics2D g2) {

        BufferedImage imagen = null;
```

```
            // Calcular posición en pantalla relativa al jugador
            int screenX = worldx - pj.jugador.worldx +
            pj.jugador.screenX;
            int screenY = worldy - pj.jugador.worldy +
            pj.jugador.screeny;

            // Solo dibujar si está en el área visible
            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) {

                switch (direccion) {
                case "arriba":
                    imagen = (numeroSpites == 1) ? arriba1 : arriba2;
                    break;
                case "abajo":
                    imagen = (numeroSpites == 1) ? abajo1 : abajo2;
                    break;
                case "izquierda":
                    imagen = (numeroSpites == 1) ? izquierda1 :
            izquierda2;
                    break;
                case "derecha":
                    imagen = (numeroSpites == 1) ? derecha1 :
            derecha2;
                    break;
                }

                g2.drawImage(imagen, screenX, screenY, null);
            }
        }
    }
```

---

## 1.2 Actualizar `Jugador.java`

**Archivo:** `src/entidad/Jugador.java`

**Cambios:** 1. Eliminar la declaración de `PanelJuego pj` (ahora viene de `Entidad`) 2. Llamar a `super(pj)` en el constructor 3. Usar el método `setup()` para cargar imágenes (opcional, pero recomendado)

```java
package entidad;

import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.Rectangle;
import java.awt.image.BufferedImage;
import java.io.IOException;
import javax.imageio.ImageIO;
import main.PanelJuego;
import main.UtilityTool;
import main.keyHandler;

public class Jugador extends Entidad {

    // ❌ ELIMINAR ESTA LÍNEA: PanelJuego pj;  (ya viene de Entidad)
    keyHandler kh;

    public final int screenX;
    public final int screeny;

    // ... resto de variables igual ...

    public Jugador(PanelJuego pj, keyHandler kh) {
        super(pj);  // ✅ NUEVO: Llamar al constructor padre
        this.kh = kh;

        // ... resto del constructor igual ...
    }

    // ✅ OPCIONAL: Refactorizar getImagenDelJugador() usando setup()
    public void getImagenDelJugador() {
        arriba1 = setup("/jugador/arriba_0001.png");
        arriba2 = setup("/jugador/arriba_0002.png");
        abajo1 = setup("/jugador/abajo_0001.png");
        abajo2 = setup("/jugador/abajo_0002.png");
        derecha1 = setup("/jugador/derecha_0001.png");
        derecha2 = setup("/jugador/derecha_0002.png");
        derecha3 = setup("/jugador/derecha_0003.png");
        izquierda1 = setup("/jugador/izquierda_0001.png");
        izquierda2 = setup("/jugador/izquierda_0002.png");
        izquierda3 = setup("/jugador/izquierda_0003.png");
    }
```

```
    // ... resto de métodos igual ...
}
```

---

# 🧙 FASE 2: Crear la Clase `NPC_Viejo`

## 2.1 Crear el Archivo

**Archivo nuevo:** `src/entidad/NPC_Viejo.java`

```
package entidad;

import java.util.Random;
import main.PanelJuego;

/**
 * NPC "Viejo" - Un personaje que camina aleatoriamente por el
       mapa.
 */
public class NPC_Viejo extends Entidad {

    public NPC_Viejo(PanelJuego pj) {
        super(pj);

        direccion = "abajo";
        vel = 1;  // Más lento que el jugador (que tiene vel = 4)

        getImagenNPC();
        setAreaSolida();
    }

    /**
     * Configura el área de colisión del NPC.
     */
    public void setAreaSolida() {
        AreaSolida = new java.awt.Rectangle();
        AreaSolida.x = 8;
        AreaSolida.y = 16;
        AreaSolida.width = 48;
        AreaSolida.height = 48;

        AreaSolidaDefaultX = AreaSolida.x;
        AreaSolidaDefaultY = AreaSolida.y;
    }
```

```java
/**
 * Carga las imágenes del viejo usando el método heredado
    setup().
 */
public void getImagenNPC() {
    // NOTA: Debes tener las imágenes en res/npc/viejo/
    arriba1 = setup("/npc/viejo/arriba1.png");
    arriba2 = setup("/npc/viejo/arriba2.png");
    abajo1 = setup("/npc/viejo/abajo1.png");
    abajo2 = setup("/npc/viejo/abajo2.png");
    izquierda1 = setup("/npc/viejo/izquierda1.png");
    izquierda2 = setup("/npc/viejo/izquierda2.png");
    derecha1 = setup("/npc/viejo/derecha1.png");
    derecha2 = setup("/npc/viejo/derecha2.png");
}

/**
 * IA del NPC: Camina aleatoriamente cada 2 segundos (120
    frames).
 */
@Override
public void setAction() {

    actionLockCounter++;

    // Solo cambiar dirección cada 120 frames (~2 segundos a
    60 FPS)
    if (actionLockCounter == 120) {

        Random random = new Random();
        int i = random.nextInt(100) + 1;  // Número entre 1 y
    100

        if (i <= 25) {
            direccion = "arriba";
        } else if (i > 25 && i <= 50) {
            direccion = "abajo";
        } else if (i > 50 && i <= 75) {
            direccion = "izquierda";
        } else {
            direccion = "derecha";
        }

        actionLockCounter = 0;  // Reiniciar contador
    }
```

```
    }
}
```

### 2.2 Estructura de Carpetas para Imágenes del NPC

Crea esta estructura en tu carpeta `res/`:

```
res/
└── npc/
    └── viejo/
        ├── arriba1.png
        ├── arriba2.png
        ├── abajo1.png
        ├── abajo2.png
        ├── izquierda1.png
        ├── izquierda2.png
        ├── derecha1.png
        └── derecha2.png
```

> **Tip:** Puedes usar sprites de `res/jugador/` temporalmente para probar.

---

## 🎯 FASE 3: Integrar NPCs en `PanelJuego`

### 3.1 Declarar el Array de NPCs

**Archivo:** `src/main/PanelJuego.java`

**Agregar después de la línea `public superObjeto[] objs = new superObjeto[15];`:**

```
// Array de NPCs
public Entidad[] npc = new Entidad[10];
```

**No olvides el import:**

```
import entidad.Entidad;
```

---

### 3.2 Modificar `setupJuego()`

```
public void setupJuego() {
    aSetter.setObjetct();
```

```
    aSetter.setNPC();  // ✅ NUEVO: Colocar NPCs
    reproducirMusicaFondo(0);
    gameState = playState;
}
```

---

## 3.3 Modificar `update()`

```
public void update() {
    if (gameState == playState) {
        // Estado: jugando
        jugador.update();

        // ✅ NUEVO: Actualizar NPCs
        for (int i = 0; i < npc.length; i++) {
            if (npc[i] != null) {
                npc[i].update();
            }
        }
    } else {
        // Estado: en pausa
    }
}
```

---

## 3.4 Modificar `paintComponent()`

### Agregar después de dibujar objetos y ANTES de dibujar al jugador:

```
// tiles
tileManager.draw(g2);

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

// ✅ NUEVO: NPCs
for (int i = 0; i < npc.length; i++) {
    if (npc[i] != null) {
        npc[i].draw(g2);
    }
```

```
}

// jugador
jugador.draw(g2);

// UI (HUD)
ui.draw(g2);
```

---

## 🔧 FASE 4: Configurar NPCs en `AssetSetter`

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

```
package main;

import entidad.NPC_Viejo;
import objetos.OBJ_llave;
import objetos.OBJ_puerta;
import objetos.OBJ_botas;
import objetos.OBJ_cofre;

public class AssetSetter {
    PanelJuego pj;

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

    public void setObjetct() {
        // Los objetos se agregan aquí
    }

    // ✅ NUEVO: Método para colocar NPCs
    public void setNPC() {
        pj.npc[0] = new NPC_Viejo(pj);
        pj.npc[0].worldx = pj.tamanioTile * 21;  // Columna 21
        pj.npc[0].worldy = pj.tamanioTile * 21;  // Fila 21

        // Puedes agregar más NPCs:
        // pj.npc[1] = new NPC_Viejo(pj);
        // pj.npc[1].worldx = pj.tamanioTile * 25;
        // pj.npc[1].worldy = pj.tamanioTile * 18;
```

```
    }
}
```

---

# 💥 FASE 5: Sistema de Colisiones para NPCs

## 5.1 Agregar Métodos a `detectorColisiones.java`

**Archivo:** `src/main/detectorColisiones.java`

**Agregar estos métodos al final de la clase:**

```java
/**
 * Detecta colisión entre una entidad y un array de entidades
       objetivo (NPCs).
 *
 * @param entidad - La entidad que se mueve (normalmente el
       jugador)
 * @param objetivo - Array de entidades objetivo (NPCs)
 * @return índice de la entidad golpeada, o 999 si no hay colisión
 */
public int checkEntidad(Entidad entidad, Entidad[] objetivo) {
    int indice = 999;

    for (int i = 0; i < objetivo.length; i++) {
        if (objetivo[i] != null) {

            // Obtener posición del área sólida de la entidad
            entidad.AreaSolida.x = entidad.worldx +
        entidad.AreaSolida.x;
            entidad.AreaSolida.y = entidad.worldy +
        entidad.AreaSolida.y;

            // Obtener posición del área sólida del objetivo
            objetivo[i].AreaSolida.x = objetivo[i].worldx +
        objetivo[i].AreaSolida.x;
            objetivo[i].AreaSolida.y = objetivo[i].worldy +
        objetivo[i].AreaSolida.y;

            // Predecir próxima posición según dirección
            switch (entidad.direccion) {
            case "arriba":
                entidad.AreaSolida.y -= entidad.vel;
                break;
            case "abajo":
                entidad.AreaSolida.y += entidad.vel;
```

```
                break;
            case "izquierda":
                entidad.AreaSolida.x -= entidad.vel;
                break;
            case "derecha":
                entidad.AreaSolida.x += entidad.vel;
                break;
            }

            // Verificar intersección
            if
        (entidad.AreaSolida.intersects(objetivo[i].AreaSolida)) {
                if (objetivo[i] != entidad) {  // Evitar colisión
        consigo mismo
                    entidad.hayColision = true;
                    indice = i;
                }
            }

            // Resetear posiciones
            entidad.AreaSolida.x = entidad.AreaSolidaDefaultX;
            entidad.AreaSolida.y = entidad.AreaSolidaDefaultY;
            objetivo[i].AreaSolida.x =
        objetivo[i].AreaSolidaDefaultX;
            objetivo[i].AreaSolida.y =
        objetivo[i].AreaSolidaDefaultY;
        }
    }

    return indice;
}

/**
 * Detecta si un NPC colisiona con el jugador.
 * Se usa para que los NPCs no atraviesen al jugador.
 *
 * @param entidad - La entidad NPC que se mueve
 */
public void checkJugador(Entidad entidad) {

    // Obtener posición del área sólida de la entidad
    entidad.AreaSolida.x = entidad.worldx + entidad.AreaSolida.x;
    entidad.AreaSolida.y = entidad.worldy + entidad.AreaSolida.y;

    // Obtener posición del área sólida del jugador
```

```
        pj.jugador.AreaSolida.x = pj.jugador.worldx +
            pj.jugador.AreaSolida.x;
        pj.jugador.AreaSolida.y = pj.jugador.worldy +
            pj.jugador.AreaSolida.y;

        // Predecir próxima posición según dirección
        switch (entidad.direccion) {
        case "arriba":
            entidad.AreaSolida.y -= entidad.vel;
            break;
        case "abajo":
            entidad.AreaSolida.y += entidad.vel;
            break;
        case "izquierda":
            entidad.AreaSolida.x -= entidad.vel;
            break;
        case "derecha":
            entidad.AreaSolida.x += entidad.vel;
            break;
        }

        // Verificar intersección
        if (entidad.AreaSolida.intersects(pj.jugador.AreaSolida)) {
            entidad.hayColision = true;
        }

        // Resetear posiciones
        entidad.AreaSolida.x = entidad.AreaSolidaDefaultX;
        entidad.AreaSolida.y = entidad.AreaSolidaDefaultY;
        pj.jugador.AreaSolida.x = pj.jugador.AreaSolidaDefaultX;
        pj.jugador.AreaSolida.y = pj.jugador.AreaSolidaDefaultY;
    }
```

---

## 5.2 Actualizar `Jugador.java` para detectar colisión con NPCs

**Archivo:** `src/entidad/Jugador.java`

**Dentro del método `update()`, agregar después de `int objIndex = pj.dColisiones.checkObjeto(this, true);`:**

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

```java
// ✅ NUEVO: Verificar colisión con NPCs
int npcIndex = pj.dColisiones.checkEntidad(this, pj.npc);
interactuarConNPC(npcIndex);
```

**Agregar este método en `Jugador.java`:**

```java
/**
 * Maneja la interacción con un NPC.
 * @param index - índice del NPC en el array pj.npc[]
 */
public void interactuarConNPC(int index) {
    if (index != 999) {
        // Por ahora solo detectamos la colisión
        // Aquí se implementará el sistema de diálogo
        System.out.println("¡Tocaste al NPC " + index + "!");
    }
}
```

---

# 📝 Resumen de Archivos Modificados/Creados

| Archivo | Acción | Descripción |
|---|---|---|
| `Entidad.java` | MODIFICAR | Agregar PanelJuego, constructor, setup(), setAction(), update(), draw() |
| `Jugador.java` | MODIFICAR | Usar super(pj), agregar checkEntidad, interactuarConNPC() |
| `NPC_Viejo.java` | CREAR | Nueva clase para el NPC viejo |
| `PanelJuego.java` | MODIFICAR | Agregar array npc[], actualizar/dibujar NPCs |
| `AssetSetter.java` | MODIFICAR | Agregar método setNPC() |
| `detectorColisiones.java` | MODIFICAR | Agregar checkEntidad(), checkJugador() |

---

# ✅ Checklist de Implementación

- [ ] 1. Refactorizar `Entidad.java` con constructor y métodos genéricos
- [ ] 2. Actualizar `Jugador.java` para usar `super(pj)`
- [ ] 3. Crear clase `NPC_Viejo.java`
- [ ] 4. Agregar sprites del NPC en `res/npc/viejo/`
- [ ] 5. Agregar array `npc[]` en `PanelJuego.java`
- [ ] 6. Modificar `setupJuego()` para llamar `setNPC()`
- [ ] 7. Actualizar y dibujar NPCs en el game loop
- [ ] 8. Crear método `setNPC()` en `AssetSetter.java`
- [ ] 9. Agregar `checkEntidad()` en `detectorColisiones.java`
- [ ] 10. Agregar `checkJugador()` en `detectorColisiones.java`
- [ ] 11. Probar colisiones entre Jugador y NPC
- [ ] 12. Probar que el NPC no atraviese paredes ni al jugador

---

# 🎮 Resultado Esperado

Al completar esta guía tendrás:

1. ✅ Un viejo caminando aleatoriamente por el mapa
2. ✅ El viejo se detiene al chocar con paredes
3. ✅ El viejo se detiene al chocar con el jugador
4. ✅ El jugador se detiene al chocar con el viejo
5. ✅ Sistema preparado para agregar más NPCs fácilmente

---

## ➡️ Siguiente Paso: Sistema de Diálogos

Una vez que los NPCs funcionen, el siguiente paso es implementar: - Estado de juego `dialogoState` - Array de diálogos en `NPC_Viejo` - Método `speak()` para iniciar conversación - Renderizado de cuadro de diálogo en `UI.java`

---

*Guía creada el 4 de enero de 2026 Basada en el tutorial de RyiSnow sobre NPCs*