# Explicación de la Clase ColisionesPanel

La clase `ColisionesPanel` es responsable de gestionar la detección de colisiones en el juego basándose en una imagen de colisión. Se utiliza el canal Alfa del PNG para determinar si una zona es transitable o no, aplicando el método Pixel-Perfect Collision Detection.

## 1. Definición de la Clase

Ubicación: `juegoprog.escenarios.ColisionesPanel`

Extiende: `JPanel` (permite que el sistema de colisiones se integre en la interfaz gráfica)

## 2. Explicación Detallada del Código

### Atributos

```java  
private BufferedImage colisionesImg;  
```

Este atributo almacena la imagen que representa las colisiones en el escenario. Las áreas opacas en la imagen representan obstáculos, mientras que las transparentes indican zonas transitables.

### Constructor

```java  
public ColisionesPanel() {  
 setOpaque(false);  
 cargarImagenCollision();  
}  
```

✔️ `setOpaque(false);` permite que el `JPanel` sea transparente y no cubra el escenario. Luego, `cargarImagenCollision();` se ejecuta para cargar la imagen de colisión en memoria.

### Método cargarImagenCollision()

```java  
private void cargarImagenCollision() { ... }  
```

Este método se encarga de cargar la imagen de colisión desde los recursos del juego y convertirla a un formato que soporte transparencia (`ARGB`).

✔️ \*\*Se carga la imagen desde los recursos\*\*:

```java  
InputStream input = getClass().getClassLoader().getResourceAsStream("escenarios/colision\_distrito\_sombrio.png");  
```

Se usa `getResourceAsStream()` para encontrar la imagen dentro de los archivos del juego.

✔️ \*\*Se convierte la imagen al formato ARGB\*\* (con soporte de transparencia):

```java  
colisionesImg = new BufferedImage(imagenOriginal.getWidth(), imagenOriginal.getHeight(), BufferedImage.TYPE\_INT\_ARGB);  
```

Esto garantiza que la imagen tenga canal Alfa para detectar colisiones correctamente.

### Método hayColision()

```java  
public boolean hayColision(int x, int y, int offsetX, int offsetY) { ... }  
```

Este método determina si un punto del mapa es transitable o no, basándose en la opacidad del píxel en la imagen de colisión.

✔️ \*\*Se calcula la posición en la imagen de colisión\*\*:

```java  
int colisionX = (x + offsetX);  
int colisionY = (y + offsetY);  
```

Se suman los `offsetX` y `offsetY` para considerar el desplazamiento del escenario.

✔️ \*\*Se obtiene el valor del píxel en esa posición\*\*:

```java  
int pixel = colisionesImg.getRGB(colisionX, colisionY);  
int alpha = (pixel >> 24) & 0xff;  
```

La variable `alpha` almacena la transparencia del píxel. Si es `0`, es transparente y transitable; si es mayor que `0`, hay colisión.

✔️ \*\*Devuelve `true` si hay colisión, `false` si es transitable\*\*.

### Método getImagenColision()

```java  
public BufferedImage getImagenColision() {  
 return colisionesImg;  
}  
```

Devuelve la imagen de colisión para su uso en depuración o ajustes visuales.

### Método paintComponent()

```java  
@Override  
protected void paintComponent(Graphics g) { ... }  
```

Este método dibuja la capa de colisiones en pantalla, útil para depuración.

✔️ \*\*Se asegura que la imagen se dibuje correctamente\*\*:

```java  
g.drawImage(colisionesImg, -offsetX, -offsetY, this);  
```

Esto ajusta la imagen de colisiones para alinearla con el desplazamiento del mapa.

### Método actualizarOffset()

```java  
public void actualizarOffset(int offsetX, int offsetY) { ... }  
```

Este método actualiza la posición del mapa de colisiones cuando el escenario se desplaza. Llama a `repaint();` para refrescar la imagen.

## 3. Ejemplo de Uso

Para usar la detección de colisiones en el juego, podemos hacer lo siguiente:

```java  
if (colisionesPanel.hayColision(personaje.getX(), personaje.getY(), offsetX, offsetY)) {  
 System.out.println("❌ Colisión detectada!");  
}  
```

Esto permite verificar si el personaje está tocando un obstáculo en la imagen de colisión.

## 4. Conclusión

La clase `ColisionesPanel` implementa una detección de colisiones basada en imágenes, permitiendo un control preciso de las áreas transitables y obstáculos. Su integración con el `JPanel` permite actualizar las colisiones en tiempo real y sincronizarlas con el movimiento del escenario.