

Java 3D: Ejemplo Ladrillos

Francisco Velasco Anguita


Dpto. Lenguajes y Sistemas Informáticos
Universidad de Granada

Sistemas Gráficos

Grado en Ingeniería Informática
Curso 2016-2017



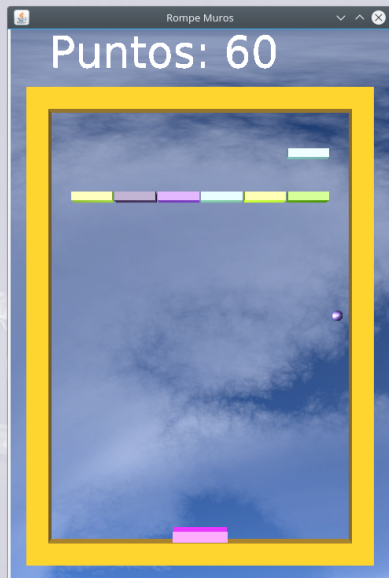
Contenidos

- 
- 1 **Objetivos**
 - 2 **Introducción**
 - 3 **Grafo de escena**
 - 4 **Diagrama de clases**
 - 5 **Implementación**

Objetivos

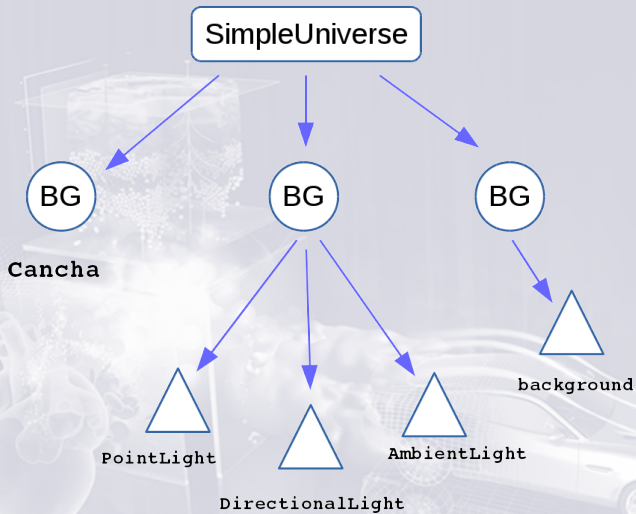
- Conocer un ejemplo de desarrollo de un Sistema Gráfico
 - ▶ Basado en grafos de escena
 - ▶ Diseño e implementación
- Seguir conociendo y aprendiendo Java 3D

Introducción



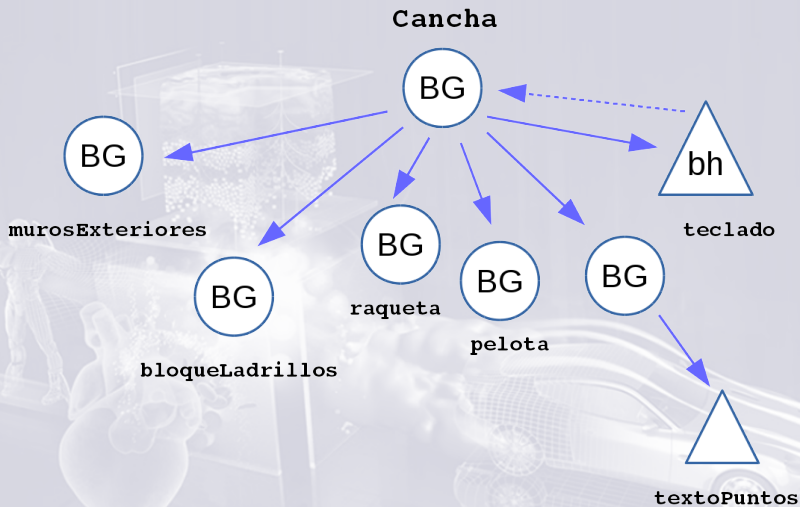
Grafo de escena

Raiz



Grafo de escena

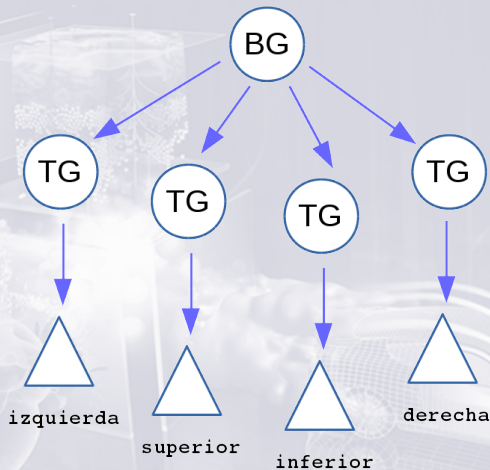
Rama *Cancha*



Grafo de escena

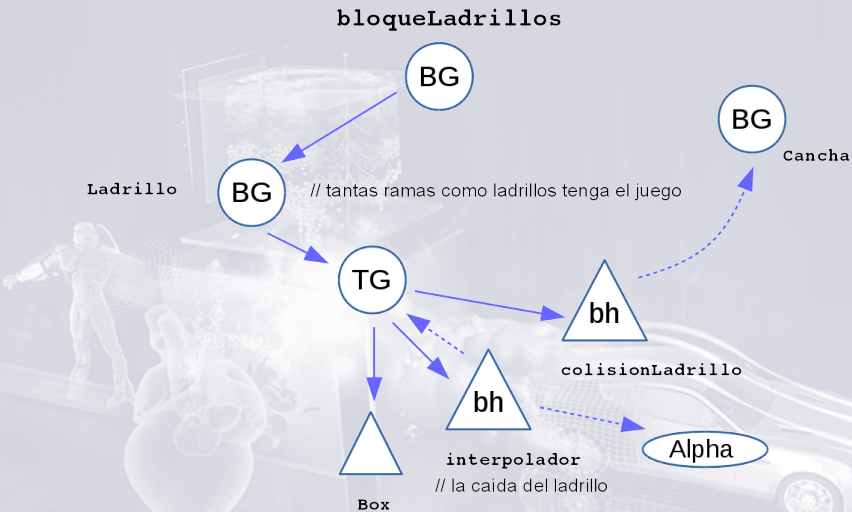
Rama *murosExteriores*

`murosExteriores`



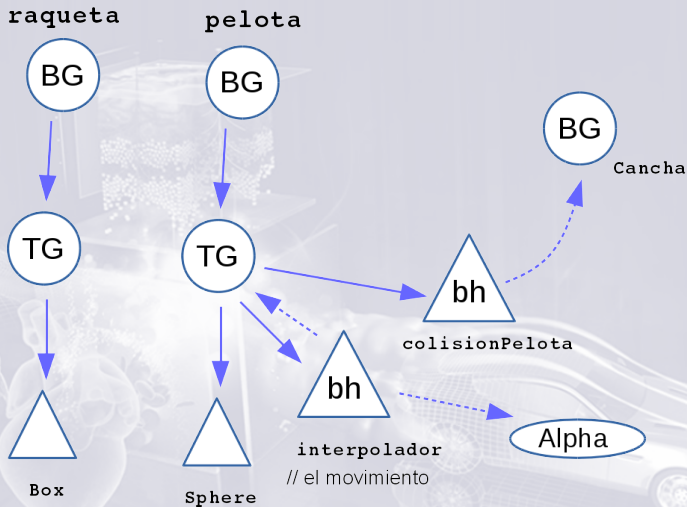
Grafo de escena

Rama *bloqueLadrillos*



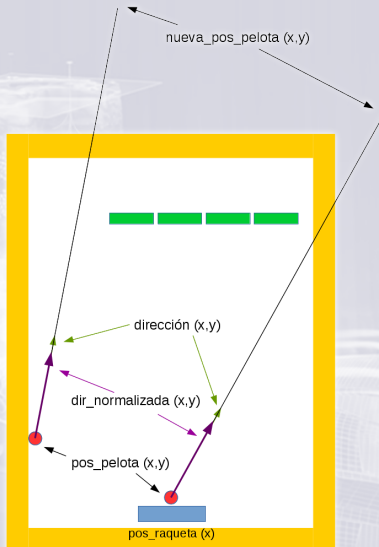
Grafo de escena

Ramas *raqueta* y *pelota*



Gestión del movimiento

$nueva_pos_pelota = pos_pelota + dir_normalizada * constante$



Gestión de las colisiones

Preparar las figuras colisionables

- Al crear las figuras colisionables se le asigna una etiqueta que las identifica

Colisiones: Etiquetado de figuras colisionables

```
Box muroSuperior = new Box ( . . . );  
muroSuperior.setUserData ( Colliders.PARED_SUPERIOR );
```

- Cuando se produzca la colisión, se consulta la etiqueta

Colisiones: Identificado de la figura colisionada

```
Colliders tipoFigura = ( Colliders ) figuraColisionada.getUserData ( );
```

Gestor de colisiones de Java 3D

- Se crea una condición de activación asociada a la figura a controlar

Colisiones: Condición de activación

```
WakeupOnCollisionEntry condicion = new WakeupOnCollisionEntry  
    (pelota.getPrimitive(), WakeupOnCollisionEntry.USE_BOUNDS);  
this.setSchedulingBounds(pelota.getPrimitive().getBounds());
```

- Cuando se produce la colisión, se consulta la geometría colisionada

Colisiones: Consulta de la geometría colisionada

```
Primitive figuraColisionada =  
    (Primitive) condicion.getTriggeringPath().getObject().getParent();  
pelota.colision(figuraColisionada);
```

Diagrama de clases

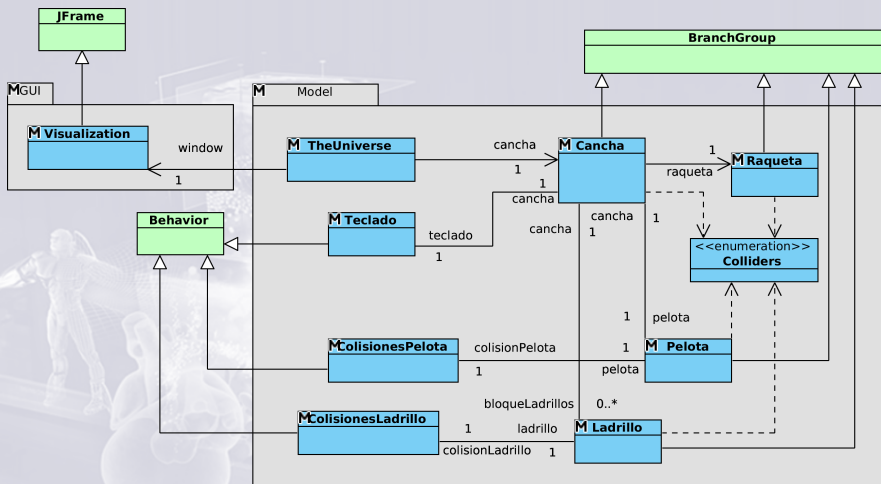
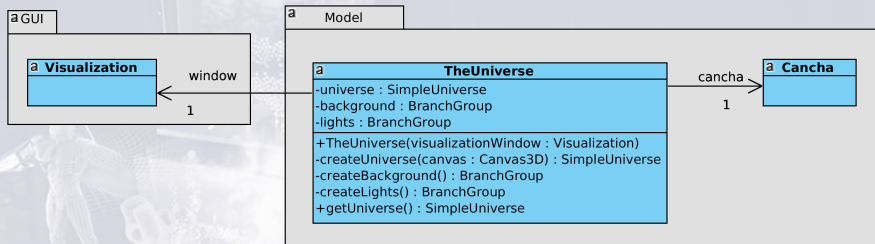


Diagrama de clases

Clase TheUniverse



Powered By Visual Paradigms Community Edition

Diagrama de clases

Clase Cancha

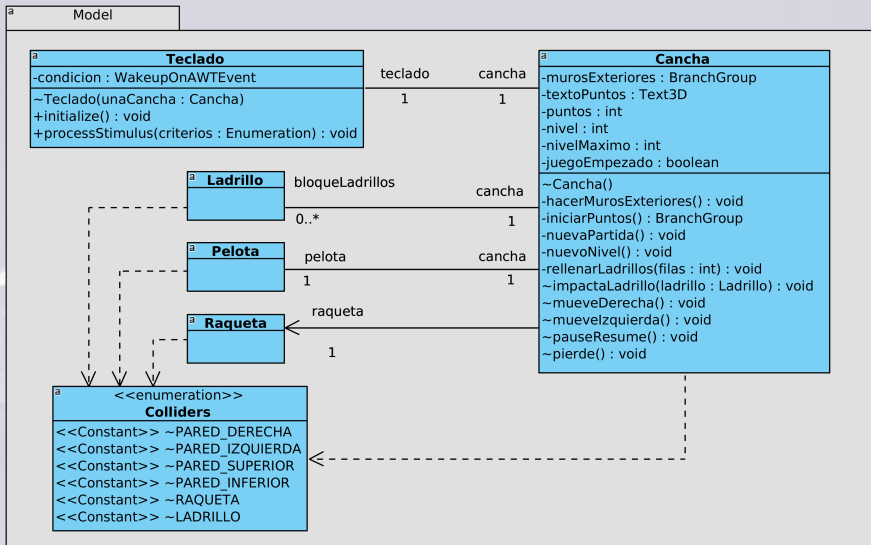


Diagrama de clases

Clase Raqueta

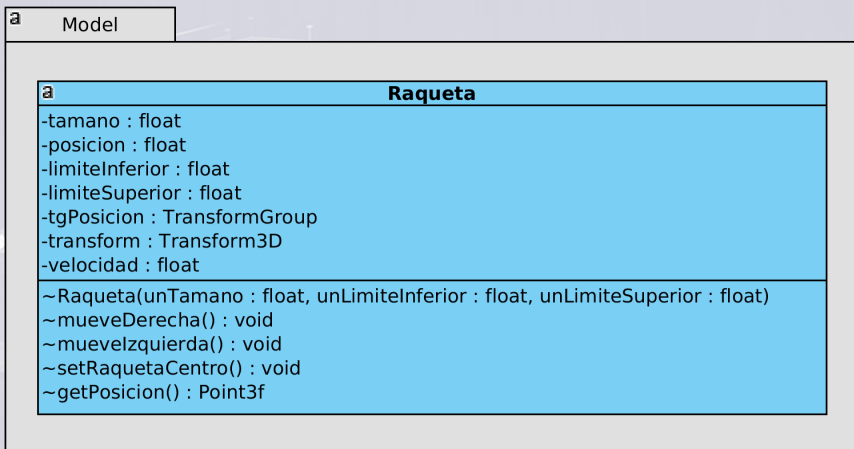


Diagrama de clases

Clase Pelota

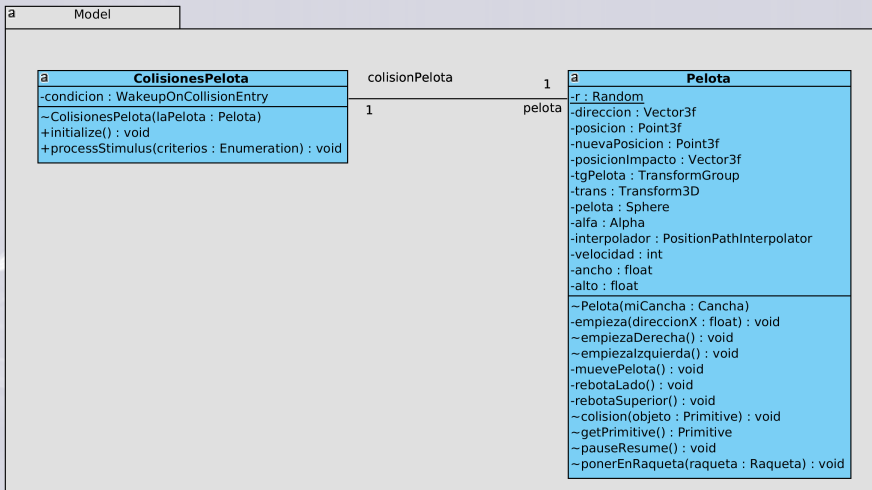
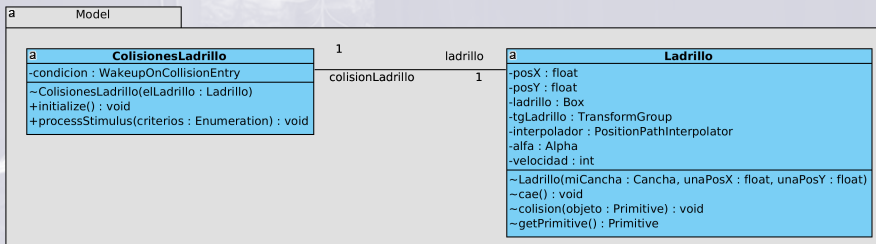


Diagrama de clases

Clase Ladrillo



Rendered By Visual Paradigm Community Edition

Implementación

- En esta presentación se incluye sólo lo más significativo
- Se proporciona el código completo, con comentarios

Programa principal: `main`

Ladrillos.java: `main`

```
public static void main(String[] args) {  
    // Se obtiene la configuración gráfica del sistema y se crea el  
    // Canvas3D que va a mostrar la imagen  
    Canvas3D canvas = new Canvas3D (SimpleUniverse.  
        getPreferredConfiguration());  
  
    // Se construye la ventana que mostrará el juego  
    Visualization visualizationWindow = new Visualization (canvas);  
  
    // Se crea el universo, incluye una vista para ese canvas  
    TheUniverse universe = new TheUniverse (visualizationWindow);  
  
    // Se hace la aplicación visible  
    visualizationWindow.setVisible(true);  
}
```

Clase TheUniverse

Constructor

TheUniverse: Constructor

```
public TheUniverse (Visualization visualizationWindow) {  
    // Atributos de referencia  
    window = visualizationWindow;  
    // Se crea el universo y la rama de la vista con ese canvas  
    universe = createUniverse(window.getCanvas());  
  
    // Se crea la rama del fondo y se cuelga al universo  
    background = createBackground();  
    universe.addBranchGraph(background);  
  
    // Se crean y se añaden luces  
    lights = createLights();  
    universe.addBranchGraph(lights);  
  
    // Se crea la rama de la escena y se cuelga al universo  
    cancha = new Cancha();  
    universe.addBranchGraph(cancha);  
}
```

Clase TheUniverse

Método createUniverse

TheUniverse.java: createUniverse (fragmento)

```
// El punto desde donde se mira, el punto a donde se mira
// y el ángulo de visión se han establecido para que se vean
// los muros y los puntos.

// La transformación de vista, dónde se está, a dónde se mira, Vup
TransformGroup viewTransformGroup = viewingPlatform.
    getViewPlatformTransform();
Transform3D viewTransform3D = new Transform3D();
viewTransform3D.lookAt (new Point3d (7,11,50),
                       new Point3d (7,11,0), new Vector3d (0,1,0));
viewTransform3D.invert();
viewTransformGroup.setTransform (viewTransform3D);

// Se establece el angulo de vision a 20 grados
Viewer viewer = new Viewer (canvas);
View view = viewer.getView();
view.setFieldOfView (Math.toRadians(20));
```

Clase TheUniverse

Método createBackground

TheUniverse.java: createBackground

```
private BranchGroup createBackground () {  
    // Se crea el objeto para la imagen de fondo y  
    // se le asigna un área de influencia  
    Background backgroundNode = new Background (  
        new TextureLoader ("imgs/back.jpg", null).getImage());  
    backgroundNode.setApplicationBounds (  
        new BoundingSphere (new Point3d (7.0, 10.0, 0.0), 100.0));  
  
    // Se crea un BranchGroup para devolver el Background  
    // Al SimpleUniverse solo se le pueden colgar BranchGroup  
    BranchGroup bgBackground = new BranchGroup();  
    bgBackground.addChild(backgroundNode);  
    return bgBackground;  
}
```

Clase TheUniverse

Método createLights

TheUniverse.java: createLights (fragmento)

```
private BranchGroup createLights () {
    // Se van a poner 3 luces, una ambiente, una direccional y una
    // puntual
    [ . . . ]
    // Se crea la luz ambiente, todas las luces deben tener su área de
    // influencia
    Light aLight = new AmbientLight (new Color3f (0.9f, 0.9f, 0.9f));
    aLight.setInfluencingBounds (influencingBound);
    aLight.setEnable(true);
    lightsNode.addChild(aLight);

    // Se crea una luz direccional
    aLight = new DirectionalLight (new Color3f (0.7f, 0.7f, 0.7f),
                                   new Vector3f (1.0f, -1.0f, -4.0f));
    [ . . . ]
    // Se crea una luz puntual
    PointLight pLight = new PointLight ();
    pLight.setPosition (7f, 10f, 0f);
    [ . . . ]
```


Clase Cancha

Constructor

Cancha: Constructor (1)

(fragmento)

```
Cancha (float ancho, float alto) {  
    [ . . . ]  
    // Se implementa el grafo de escena  
    // Se construyen ramas y se cuelgan de this  
    hacerMurosExteriores();  
    this.addChild(murosExteriores);  
  
    // Al nodo se le deben poder añadir y quitar ladrillos estando vivo  
    // Se le dan las capacidades para ello  
    bloqueLadrillos = new BranchGroup();  
    bloqueLadrillos.setCapability (BranchGroup.ALLOW_CHILDREN_WRITE);  
    bloqueLadrillos.setCapability (BranchGroup.ALLOW_CHILDREN_EXTEND);  
    this.addChild(bloqueLadrillos);  
  
    // Se crea y cuelgan la Raqueta, la Pelota  
    // y el texto de la puntuación  
    [ . . . ]  
}
```

Clase Cancha

Constructor

Cancha: Constructor (y 2)

(fragmento)

```
[ . . . ]

// Los comportamientos requieren de un área de influencia
// La detección y procesamiento de las pulsaciones de teclado se
// realiza con un nodo comportamiento

teclado = new Teclado (this);
teclado.setSchedulingBounds(new BoundingSphere (
    new Point3d (ancho/2, alto/2, 0.0), ancho+alto));
this.addChild(teclado);

// Se rellenan los ladrillos para el primer nivel
nuevoNivel();
}
```

Clase Cancha

Método hacerMurosExteriores

Cancha: hacerMurosExteriores (1) (fragmento)

```
private void hacerMurosExteriores () {  
    Box superior, inferior, derecha, izquierda;  
    murosExteriores = new BranchGroup();  
  
    // Se define un material lambertiano para los muros  
    Appearance app = new Appearance ();  
    app.setMaterial(new Material (  
        new Color3f (0.20f, 0.20f, 0.20f), // Color ambiental  
        new Color3f (0.00f, 0.00f, 0.00f), // Color emisivo  
        new Color3f (0.49f, 0.34f, 0.00f), // Color difuso  
        new Color3f (0.89f, 0.79f, 0.00f), // Color especular  
        17.0f ));  
  
    [ . . . ]  
}
```

Clase Cancha

Método `hacerMurosExteriores`

Cancha: `hacerMurosExteriores` (y 2) (fragmento)

```
[ . . . ]
```

```
// Se crean las cajas de los muros del tablero
```

```
// OJO, las cajas se crean al doble de las dimensiones indicadas
superior = new Box(ancho/2,0.5f,0.5f,Primitive.GENERATE_NORMALS,app);
```

```
// Se le asigna la etiqueta para la detección de colisiones
superior.setUserData(Colliders.PARED_SUPERIOR);
```

```
// Se posiciona y se cuelga de su nodo padre
Transform3D posicion = new Transform3D ();
posicion.setTranslation (new Vector3f (ancho/2, alto+0.5f, 0));
TransformGroup tgSuperior = new TransformGroup (posicion);
tgSuperior.addChild(superior);
murosExteriores.addChild (tgSuperior);
```

```
// De manera similar con los otros 3 muros
[ . . . ]
```

Clase Cancha

Método iniciarPuntos

Cancha: iniciarPuntos

```
private BranchGroup iniciarPuntos() {
    puntos = 0;
    // El objeto del tipo de letra
    Font3D fuente = new Font3D (new Font ("Helvetica", Font.PLAIN, 2),
        new FontExtrusion());
    // La geometría del texto
    textoPuntos = new Text3D (fuente, "Puntos: " + puntos, new Point3f
        (0, alto+2, 0));
    // La capacidad para poder actualizarlo
    textoPuntos.setCapability(Text3D.ALLOW_STRING_WRITE);

    // La rama de la que cuelga el texto
    BranchGroup bgPuntos = new BranchGroup();
    bgPuntos.addChild (new Shape3D (textoPuntos));
    // Esta geometría no es colisionable
    bgPuntos.setCollidable (false);
    return bgPuntos;
}
```

Clase Cancha

Método rellenarLadrillos

Cancha: rellenarLadrillos

```
private void rellenarLadrillos (int filas) {  
    totalLadrillos = 0;  
    Ladrillo l;  
    int ladrillosPorFila = (int) ancho/2 - 1;  
    for (int i = 0; i < ladrillosPorFila; i++)  
        for (int j = 0; j < filas; j++) {  
            // La última fila se rellena completa  
            // En las anteriores no se ponen todos los posibles  
            if (r.nextBoolean() || j == (filas-1)) {  
                l = new Ladrillo (this,  
                    i*2f + ancho/2-ladrillosPorFila + 1, -j*2f + alto-2);  
                bloqueLadrillos.addChild(l);  
                totalLadrillos++;  
            }  
        }  
}
```

Clase Cancha

Métodos `mueveDerecha` y `pauseResume`

Cancha: `mueveDerecha` y `pauseResume`

```
void mueveDerecha () {  
    if (!juegoEmpezado) {  
        juegoEmpezado = true;  
        pelota.empiezaDerecha();  
    }  
    raqueta.mueveDerecha();  
}
```

// Similar para el movimiento a la izquierda

```
void pauseResume () {  
    pelota.pauseResume();  
}
```

Clase Cancha

Método impactaLadrillo

Cancha: impactaLadrillo

```
void impactaLadrillo (Ladrillo ladrillo) {
    ladrillo.cae();
    totalLadrillos--;
    puntos += nivel*10;
    textoPuntos.setString ("Puntos: " + puntos);
    if (totalLadrillos <= 0) // Se completa un nivel
        if (nivel == nivelMaximo) { // Se completa la partida
            if (JOptionPane.showConfirmDialog(null,
                "H A S G A N A D O\n ¿Deseas empezar una nueva partida?",
                "Rompe Muros", JOptionPane.YES_NO_OPTION)
                == JOptionPane.YES_OPTION) {
                nuevaPartida();
            } else System.exit(0); // No se desea una nueva partida
        } else { // Se avanza al siguiente nivel
            nuevoNivel ();
            JOptionPane.showMessageDialog(null, "Avanzas de Nivel",
                "Rompe Muros", JOptionPane.INFORMATION_MESSAGE);
        }
    }
}
```


Clase Cancha

Métodos nuevoNivel, nuevaPartida y pierde

Cancha: nuevoNivel, nuevaPartida y pierde

```
private void nuevoNivel () {
    pelota.ponerEnRaqueta (raqueta);
    if (bloqueLadrillos.numChildren() > 0)
        bloqueLadrillos.removeAllChildren();
    nivel++;
    rellenarLadrillos (nivel);
}

private void nuevaPartida () {
    juegoEmpezado = false; nivel = 0; puntos = 0;
    textoPuntos.setString ("Puntos: " + puntos);
    raqueta.setRaquetaCentro();
    nuevoNivel();
}

void pierde () {
    if (JOptionPane.showConfirmDialog(null,
        "H A S   P E R D I D O \n ¿Deseas empezar una nueva partida?",
        "Rompe Muros", JOptionPane.YES_NO_OPTION)
        == JOptionPane.YES_OPTION) {
        nuevaPartida();
    } else System.exit(0);
}
```

Clase Raqueta

Constructor

Raqueta: Constructor

(fragmento)

```

Raqueta (float unTamano, float unLimiteInferior ,
        float unLimiteSuperior) {

    [ . . . ]

    Appearance app = new RandomColor ();
    raqueta = new Box (tamano/2f, 0.25f, 0.5f,
        Primitive.GENERATE_NORMALS, app);
    raqueta.setUserData(Colliders.RAQUETA);
    posicion = (limiteSuperior + limiteInferior) / 2;
    transform = new Transform3D();
    tgPosicion = new TransformGroup ();
    // Para poder moverla hay que darle las capacidades adecuadas
    tgPosicion.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
    tgPosicion.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
    tgPosicion.addChild(raqueta);
    setRaquetaCentro();
    this.addChild(tgPosicion);
}

```

Clase Raqueta

Métodos mueveIzquierda y setRaquetaCentro

Raqueta: mueveIzquierda y setRaquetaCentro

```
void mueveIzquierda () {  
    if ((posicion - velocidad) > (limiteInferior + tamano/2)) {  
        posicion -= velocidad;  
        // Se calcula la nueva posicion y se actualiza la transformación  
        transform.set (new Vector3f (posicion, 0.3f, 0));  
        tgPosicion.setTransform(transform);  
    }  
}  
  
void setRaquetaCentro () {  
    posicion = (limiteInferior + limiteSuperior) / 2;  
    transform.set (new Vector3f (posicion, 0.3f, 0));  
    tgPosicion.setTransform(transform);  
}
```

Clase Pelota

Constructor

Pelota: Constructor (1)

(fragmento)

```

Pelota (Cancha miCancha) {
    [ . . . ]
    // El movimiento se realiza entre una posición origen
    // y una posición destino
    posicion = new Point3f();
    direccion = new Vector3f();
    nuevaPosicion = new Point3f();
    posicionImpacto = new Vector3f();
    // Se crea la rama (geometría) en el grafo para la pelota
    Appearance app = new RandomColor();
    pelota = new Sphere (0.25f, Primitive.GENERATE_NORMALS, app);
    trans = new Transform3D();
    tgPelota = new TransformGroup (trans);
    tgPelota.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
    tgPelota.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
    tgPelota.addChild(pelota);
    this.addChild(tgPelota);
    [ . . . ]

```

Clase Pelota

Constructor

Pelota: Constructor (y 2)

(fragmento)

```
[ . . . ]
// El alpha y el interpolador para los movimientos
velocidad = 100; // Es tiempo, a valor más bajo, pelota más rápida
alfa = new Alpha (1,(long)(ancho+alto)*velocidad);
alfa.pause();
puntos = new Point3f[2];
puntos[0] = posicion;
puntos[1] = nuevaPosicion;
float knots[] = {0.0f, 1.0f};
interpolador = new PositionPathInterpolator (alfa, tgPelota, trans,
    knots, puntos);
interpolador.setSchedulingBounds(
    new BoundingSphere (new Point3d (0,0,0), 10));
tgPelota.addChild(interpolador);
// El comportamiento para las colisiones
colisionPelota = new ColisionesPelota(this);
colisionPelota.setEnable(false);
tgPelota.addChild(colisionPelota);
}
```

Clase Pelota

Métodos para el movimiento

Pelota: empiezaIzquierda, empieza y muevePelota

```
void empiezaIzquierda () {
    float direccionX = -r.nextFloat() - 0.5f; // Aleatorio en [-1.5, -0.5]
    empieza (direccionX);
}
private void empieza (float direccionX) {
    direccion = new Vector3f (direccionX, 1, 0);
    direccion.normalize(); // Para no afectar a la velocidad
    muevePelota();
    colisionPelota.setEnabled (true);
}
private void muevePelota () {
    nuevaPosicion.setX (posicion.x + direccion.x * (ancho+alto));
    nuevaPosicion.setY (posicion.y + direccion.y * (ancho+alto));
    interpolador.setPosition(0, posicion);
    interpolador.setPosition(1, nuevaPosicion);
    alfa.resume();
    alfa.setStartTime(System.currentTimeMillis());
}
```

Clase Pelota

Método colision

Pelota: colision

```
void colision (Primitive objeto) {
    alfa.pause();
    Colliders tipoObjeto = (Colliders) objeto.getUserData();
    if (tipoObjeto != null) {
        switch (tipoObjeto) {
            case PARED_DERECHA : case PARED_IZQUIERDA :
                rebotaLado(); break;
            case LADRILLO :
                // Para llegar al BG del Ladrillo desde el Box
                cancha.impactaLadrillo((Ladrillo)
                    objeto.getParent().getParent());
            case PARED_SUPERIOR : case RAQUETA :
                rebotaSuperior(); break;
            case PARED_INFERIOR :
                cancha.pierde (); break;
        }
    }
    alfa.resume();
}
```

Clase Pelota

Método rebotaLado

Pelota: rebotaLado

```
private void rebotaLado () {  
    tgPelota.getTransform(trans);  
    trans.get(posicionImpacto);  
    posicion.set (posicionImpacto);  
    // Las esquinas se tratan de manera especial  
    if (posicion.y < 1.5 || posicion.y > alto-1) {  
        direccion.y *= -1.05f;  
        direccion.normalize();  
    }  
    direccion.x *= -1;  
    muevePelota();  
}  
  
// rebotaSuperior es similar
```


Clase Ladrillo

Constructor

Ladrillo: Constructor (1)

(fragmento)

```
Ladrillo (Cancha miCancha, float unaPosX, float unaPosY) {
    Point3f puntos[];
    // Parte de la rama (geometría)
    cancha = miCancha;
    posX = unaPosX;
    posY = unaPosY;
    Appearance app = new RandomColor();
    ladrillo = new Box (0.95f, 0.2f, 0.5f,
        Primitive.GENERATE_NORMALS, app);
    ladrillo.setUserData(Colliders.LADRILLO);
    Transform3D transform = new Transform3D();
    transform.set(new Vector3f(posX, posY, 0f));
    tgLadrillo = new TransformGroup(transform);
    tgLadrillo.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
    tgLadrillo.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
    tgLadrillo.addChild(ladrillo);
    this.addChild(tgLadrillo);
    [ . . . ]
}
```

Clase Ladrillo

Constructor

Ladrillo: Constructor (y 2)

(fragmento)

```
[ . . . ]
// Movimiento y colisiones
velocidad = 1500;  alfa = new Alpha (1, velocidad);  alfa.pause();
puntos = new Point3f[2];
puntos[0] = new Point3f (posX, posY, 0);
puntos[1] = new Point3f (posX, -1.5f, 0);
float knots[] = {0.0f, 1.0f};
interpolador = new PositionPathInterpolator (alfa, tgLadrillo,
    transform, knots, puntos);
interpolador.setSchedulingBounds(
    new BoundingSphere (new Point3d (0,0,0), 10));
interpolador.setEnable(false);
tgLadrillo.addChild(interpolador);
colisionLadrillo = new ColisionesLadrillo(this);
colisionLadrillo.setEnable(false);
tgLadrillo.addChild(colisionLadrillo);
// Se le permite separarse del BranchGroup que los agrupa
this.setCapability(BranchGroup.ALLOW_DETACH);
}
```

Clase Ladrillo

Caída y Colisión

Ladrillo: cae y colision

```
void cae () {  
    ladrillo.setUserData(null);  
    alfa.resume();  
    alfa.setStartTime(System.currentTimeMillis());  
    interpolador.setEnable(true);  
    colisionLadrillo.setEnable(true);  
}  
  
void colision (Primitive objeto) {  
    Colliders tipoObjeto = (Colliders) objeto.getUserData();  
    if (tipoObjeto != null) {  
        switch (tipoObjeto) {  
            case RAQUETA :  
                cancha.pierde();  
                break;  
        }  
    }  
}
```

Comportamientos

Clase Teclado

Teclado: Condición de Activación y Procesamiento

```
// Se activa el comportamiento pulsando una tecla
private WakeupOnAWTEvent condicion =
    new WakeupOnAWTEvent (KeyEvent.KEY_PRESSED);
[ . . . ]
// Una vez se tiene en la variable  tecla  la pulsación del usuario
//   Se le evían mensajes a la  cancha  según proceda
switch (tecla.getKeyCode ()) {
    // Diversos casos
    case VK_RIGHT :
        cancha.mueveDerecha();
        break;
    case VK_LEFT :
        cancha.muevelzquierda();
        break;
    [ . . . ]
}
// Se establece de nuevo la condición de respuesta
wakeupOn (condicion);
[ . . . ]
```

Comportamientos

Clase ColisionesPelota

ColisionesPelota: Condición de Activación y Procesamiento

```
// Se activa cuando la pelota colisione con otra geometría
condicion = new WakeupOnCollisionEntry (laPelota.getPrimitive(),
    WakeupOnCollisionEntry.USE_BOUNDS);
// El área de influencia debe ajustarse a la pelota
this.setSchedulingBounds(pelota.getPrimitive().getBounds());
[ . . . ]

// Se recupera primitiva que ha colisionado y se le envía
// al objeto pelota para que lo procese
WakeupOnCollisionEntry disparo = (WakeupOnCollisionEntry)
    criterios.nextElement();
Primitive objeto = (Primitive) disparo.getTriggeringPath().
    getObject().getParent();
pelota.colision (objeto);
// Se debe activar de nuevo la condición del comportamiento
wakeupOn(condicion);
[ . . . ]
```

Comportamientos

Clase ColisionesLadrillo

ColisionesLadrillo: Condición de Activación y Procesamiento

```
// Se activa cuando el ladrillo colisione con otra geometría
condicion = new WakeupOnCollisionEntry (ladrillo.getPrimitive(),
    WakeupOnCollisionEntry.USE_BOUNDS);
// El área de influencia debe ajustarse al ladrillo
this.setSchedulingBounds(ladrillo.getPrimitive().getBounds());
[ . . . ]

// Se recupera primitiva que ha colisionado y se le envía
// al objeto ladrillo para que lo procese
WakeupOnCollisionEntry disparo = (WakeupOnCollisionEntry)
    criterios.nextElement();
Primitive objeto = (Primitive) disparo.getTriggeringPath().
    getObject().getParent();
ladrillo.colision (objeto);
// Se debe activar de nuevo la condición del comportamiento
wakeupOn(condicion);
[ . . . ]
```

Java 3D: Ejemplo Ladrillos

Francisco Velasco Anguita

Dpto. Lenguajes y Sistemas Informáticos
Universidad de Granada

Sistemas Gráficos

Grado en Ingeniería Informática
Curso 2016-2017

