```
/*
 * Seek.java
 *
 * Use the Java 3D API to write a simple video game -
 * Find the hidden treasure.
 *
 * written by mike slattery - april 2000
 * ( mikes (a) mscs.mu.edu )
 * Modified to try and make door and ball
 *  behaviors CPU speed independent, mcs - april 2002
 * Door switched to PositionInterpolator, mcs - nov 2003
 */

import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import java.util.Enumeration;
import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.universe.*;
import javax.media.j3d.*;
import javax.vecmath.*;
import com.sun.j3d.utils.behaviors.keyboard.*;

class Wall extends Box
{
        static final Color3f black = new Color3f(0.0f, 0.0f, 0.0f);
        static final Color3f white = new Color3f(1.0f, 1.0f, 1.0f);

        Wall(float xw, float zw, float h)
        {
           // Construct a wall object with x and y dimensions of
           // xw and zw respectively and height h
           //
           super(xw/2.0f, h/2.0f, zw/2.0f, null);
           Appearance app = new Appearance();
           Color3f color = new Color3f(0.0f, 1.0f, 0.0f);
           app.setMaterial(new Material(color, black, color, white, 80.0f));
           setAppearance(app);
           // Now, setup userData tags to recognize sides
           getShape(Box.FRONT).setUserData(new Integer(Seek.ZWALL));
           getShape(Box.BACK).setUserData(new Integer(Seek.ZWALL));
           getShape(Box.LEFT).setUserData(new Integer(Seek.XWALL));
           getShape(Box.RIGHT).setUserData(new Integer(Seek.XWALL));
        }

        Wall getNode()
        {
                // the wall is its own top-level node
                return this;
        }
}

class Door
{
        TransformGroup lift;
        static final Color3f black = new Color3f(0.0f, 0.0f, 0.0f);
        static final Color3f white = new Color3f(1.0f, 1.0f, 1.0f);

        Door(float xw, float zw, float h)
        {
           Wall w = new Wall(xw, zw, h);
           Appearance app = new Appearance();
           Color3f color = new Color3f(0.2f, 0.4f, 0.9f);
           app.setMaterial(new Material(color, black, color, white, 80.0f));
           w.setAppearance(app);
           Transform3D t = new Transform3D();
           t.setTranslation(new Vector3f(0.0f, 1.1f, 0.0f));
           lift = new TransformGroup(t);
           lift.addChild(w);
           // Allow the door to move
           lift.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
           lift.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
           // We want a PositionInterpolator that will
           // move the door along the y-axis, so we rotate
           // about the z-axis to move the x-axis to where
           // the y-axis is.
                   Transform3D rot_to_y = new Transform3D();
                   rot_to_y.rotZ(Math.PI/2.0);
                   // The PositionInterpolator will change the value
                   // of the lift transform, but initially we leave out
                   // the Alpha (so the interpolator doesn't do anything)
           PositionInterpolator pi = new PositionInterpolator(null, lift,
                                                                   rot_to
_y, 1.1f, 2.8f);
           DoorBehavior raise = new DoorBehavior(w.getShape(Box.FRONT), pi);
           // We need to make sure that both of these behaviors have
           // scheduling bounds and are attached to the scene graph
           pi.setSchedulingBounds(new BoundingSphere(new Point3d(), 100.0));
           raise.setSchedulingBounds(new BoundingSphere(new Point3d(), 100.0));
           lift.addChild(pi);
           lift.addChild(raise);
        }

        TransformGroup getNode()
        {
                return lift;
        }

        class DoorBehavior extends Behavior
        {
                // This behavior starts the PositionInterpolator
                // (which actually opens the door) when something
                // hits the door.
                PositionInterpolator opener;
                Shape3D shape;

                DoorBehavior(Shape3D s, PositionInterpolator pi)
                {
                        opener = pi;
                        shape = s;
                }

                public void initialize()
                {
                        // We don't do anything until a collision:
                        wakeupOn(new WakeupOnCollisionEntry(shape));
                }

                public void processStimulus(Enumeration criteria)
                {
                  // Set up an Alpha to control the opener
                  long currTime = getView().getCurrentFrameStartTime();
                  Alpha clock = new Alpha(1, 2000L);
                  clock.setStartTime(currTime);
                  opener.setAlpha(clock);
                }
                }
}
```

```
  class Building
  {
     BranchGroup objRoot;

     Building()
     {
       /*
        * Construct a simple building of 3 walls and a front door
        * which opens when hit by anything
        */
        objRoot = new BranchGroup();

        Seek.addAt(objRoot, new Wall(3.0f, 0.5f, 2.0f),
                                new Vector3f(0.0f, 1.0f, -0.75f));
        Seek.addAt(objRoot, new Wall(0.5f, 2.0f, 2.0f),
                                new Vector3f(-1.75f, 1.0f, 0.0f));
        Seek.addAt(objRoot, new Wall(0.5f, 2.0f, 2.0f),
                                new Vector3f(1.75f, 1.0f, 0.0f));

        Seek.addAt(objRoot, (new Door(2.8f, 0.5f, 1.8f)).getNode(),
                                new Vector3f(0.0f, 0.0f, 0.75f));
     }

     BranchGroup getNode()
     {
       return objRoot;
     }

  }

class Ball
{
   Vector3f velocity;   //Current velocity of the ball
   TransformGroup posTG; //A transform to the current position
   TransformGroup vpTrans; //Game viewpoint transformation

   Ball(TransformGroup vp)
   {
              // Save the Viewpoint to use when launching the ball
              vpTrans = vp;
              posTG = new TransformGroup();
              // Initially hide the ball under the floor
              setPosition(new Vector3f(0.0f, -2.0f, 0.0f));
       // Enable the TRANSFORM_WRITE capability so that
              // our behavior code can modify it at runtime.  Add it to the
              // root of the subgraph.
       posTG.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
       posTG.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);

       // Create a sphere, add it to the scene graph
       Appearance appear = new Appearance();
       Material material = new Material();
       material.setShininess(50.0f);
       material.setDiffuseColor(0.0f, 0.0f, 0.9f);
       appear.setMaterial(material);
       Sphere s = new Sphere(0.2f, appear);
       posTG.addChild(s);

       // Create a new Behavior object to move the ball
       BallMoveBehavior drop = new BallMoveBehavior();

              // a bounding sphere specifies a region a behavior is active
       // create a sphere centered at the origin with radius of 100
       // (I really don't know what a reasonable radius is here)
       BoundingSphere bounds = new BoundingSphere(new Point3d(),100.0);
       drop.setSchedulingBounds(bounds);
       posTG.addChild(drop);

       // Set collision behavior too
       CollisionDetector cd = new CollisionDetector(s.getShape());
       cd.setSchedulingBounds(bounds);
       posTG.addChild(cd);
   }

   void setPosition(Vector3f loc)
   {
     // Move the Ball to the given position
     Transform3D t = new Transform3D();

     t.setTranslation(loc);
     posTG.setTransform(t);
   }

   TransformGroup getNode()
   {
     return posTG;
   }

       class BallMoveBehavior extends Behavior
       {
               Transform3D transform = new Transform3D();
               Vector3f translate = new Vector3f();
               WakeupOnElapsedFrames stim = new WakeupOnElapsedFrames(0);
               WakeupOnAWTEvent keyStim = new WakeupOnAWTEvent(KeyEvent.KEY_PRESSED);
               boolean ball_fired = false;
               int counter = 0;
               // Keep frame times
               long prevTime, currTime;
               // Create a temp vector for scaled versions of
               // velocity and gravity
               Vector3f scaled = new Vector3f();

               // Move the ball under the influence of gravity.
               // The position is stored in the posTG TransformGroup,
               // velocity is in the vector velocity and acceleration
               // is specified by Seek.gravity.

               BallMoveBehavior()
               {
               }

               public void initialize()
               {
                       // We don't do anything until the F key is pressed:
                       wakeupOn(keyStim);
               }

               public void processStimulus(Enumeration criteria)
               {
                 WakeupCriterion wakeup;
                 AWTEvent[] event;
                 float t;

                 if (!ball_fired)
                 {
                   boolean fire = false;

                   // Check for F key pressed
                   while (criteria.hasMoreElements())
                   {
```

```
        wakeup = (WakeupCriterion) criteria.nextElement();

          event = ((WakeupOnAWTEvent)wakeup).getAWTEvent();
          for (int i=0; i<event.length; i++)
          {
            int key = ((KeyEvent)event[i]).getKeyCode();
            if (key == KeyEvent.VK_F)
                fire = true;
          }
      }
      if (fire)
      {
          // Do stuff to launch ball
          // and set trigger for next frame
          vpTrans.getTransform(transform);
          Matrix3f rotMat = new Matrix3f();
          transform.get(rotMat, translate);
          // Now, reuse transform to adjust the ball's position
          posTG.getTransform(transform);
          transform.setTranslation(translate);
          posTG.setTransform(transform);
          // Then, use rotMat to setup the velocity
          // This is change per second (and is scaled
          // appropriately when used)
          velocity = new Vector3f(0.0f, 0.8f, -0.5f);
          rotMat.transform(velocity);

          ball_fired = true;
          // This code uses a frame counter to measure
          // the lifetime of the ball.  That means that
          // balls live longer on slower machines (fewer
          // frames per second).  Probably total elapsed
          // time would be a better criterion.
          counter = 1000;
          prevTime = getView().getCurrentFrameStartTime();
          wakeupOn(stim);
      }
      else
      {
          // reset key trigger
          wakeupOn(keyStim);
      }
    }
    else
    {
          // Get elapsed time
          currTime = getView().getCurrentFrameStartTime();
                          t = (currTime-prevTime)/1000.0f;
                          prevTime = currTime;
      // Do basic ball movement
      posTG.getTransform(transform);
      transform.get(translate);
      scaled.scale(t,velocity);
      translate.add(scaled);
      transform.setTranslation(translate);
      posTG.setTransform(transform);
      // Let gravity act
      scaled.scale(t,Seek.gravity);
      velocity.add(scaled);
      counter--;
      // Add in check for below floor, just to
      // deal with falling through floor problem
      // (was: if (counter>0) )
      if ((counter>0)&&(translate.y>0.0f))
        wakeupOn(stim);
                      else
                      {
                        // Reset ball to under floor
                        setPosition(new Vector3f(0.0f, -2.0f, 0.0f));

                        ball_fired = false;
                        wakeupOn(keyStim);
                      }
                    }
                  }
        }

class CollisionDetector extends Behavior {

  // Bounce the ball off of any object which it
  // hits.  We use the userData fields to store information
  // about the wall, door, and floor orientations (to bounce
  // properly).  If no userData is available, we default to
  // reversing the z direction.

  private WakeupOnCollisionEntry wEnter;

  public CollisionDetector(Shape3D s) {
      wEnter = new WakeupOnCollisionEntry(s);
  }

  public void initialize() {
      wakeupOn(wEnter);
  }

  public void processStimulus(Enumeration criteria) {
    SceneGraphPath path;
    WakeupCriterion wakeup;
    int type=Seek.ZWALL;

    while (criteria.hasMoreElements())
    {
       wakeup = (WakeupCriterion) criteria.nextElement();
       if (wakeup instanceof WakeupOnCollisionEntry)
       {
           // Find out what we hit and lookup any userData
           //
           path = ((WakeupOnCollisionEntry)wakeup).getTriggeringPath();
           Node node = path.getObject();
           Integer user = (Integer)(node.getUserData());
           if (user != null)
             type = user.intValue();
           if (type == Seek.FLOOR)
           {
             velocity.y = -0.9f * velocity.y;
           }
           else if (type == Seek.XWALL)
           {
             velocity.x = -0.9f * velocity.x;
           }
           else
           {
             velocity.z = -0.9f * velocity.z;
           }
       }
    }
    wakeupOn(wEnter);
  }
}
```

```
}

  class SpinningCube
  {
    // Create a spinning, tilted ColorCube
    TransformGroup objRotate;

      SpinningCube()
      {
        // rotate object has composited transformation matrix
        Transform3D rotate = new Transform3D();
        Transform3D tempRotate = new Transform3D();

        rotate.rotX(Math.PI/4.0d);
        tempRotate.rotY(Math.PI/5.0d);
        rotate.mul(tempRotate);

        objRotate = new TransformGroup(rotate);

        // Create the transform group node and initialize it to the
        // identity.  Enable the TRANSFORM_WRITE capability so that
        // our behavior code can modify it at runtime.  Add it to the
        // root of the subgraph.
        TransformGroup objSpin = new TransformGroup();
        objSpin.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);

        objRotate.addChild(objSpin);

        // Create a simple shape leaf node, add it to the scene graph.
        // ColorCube is a Convenience Utility class
        objSpin.addChild(new ColorCube(0.1));

        // Create a new Behavior object that will perform the desired
        // operation on the specified transform object and add it into
        // the scene graph.
        Transform3D yAxis = new Transform3D();
        Alpha rotationAlpha = new Alpha(-1, 4000);

        RotationInterpolator rotator =
            new RotationInterpolator(rotationAlpha, objSpin, yAxis,
                                     0.0f, (float) Math.PI*2.0f);

        // a bounding sphere specifies a region a behavior is active
        // create a sphere centered at the origin with radius of 1
        BoundingSphere bounds = new BoundingSphere();
        rotator.setSchedulingBounds(bounds);
        objSpin.addChild(rotator);
      }

      TransformGroup getNode()
      {
        return objRotate;
      }
  }


public class Seek extends Applet {
                // Gravity is change of velocity per second (and is
                // scaled appropriately when used)
        public static final Vector3f gravity = new Vector3f(0.0f, -1.0f, 0.0f);
        public static final int FLOOR=1;
        public static final int XWALL=2;
        public static final int ZWALL=3;
        SimpleUniverse simpleU=null;


  public static void addAt(Group root, Node n, Vector3f loc)
  {
    // This is a general utility method for doing translations
    // It's here because I didn't know where else to put it.
    //
    // Add the Node n to the Group root using a translation
    // to place the new group at the loccation specified by loc
    Transform3D t = new Transform3D();
    t.setTranslation(loc);
    TransformGroup TG = new TransformGroup(t);
    TG.addChild(n);
    root.addChild(TG);
  }

  public BranchGroup createSceneGraph() {
     // Create the root of the branch graph
     BranchGroup objRoot = new BranchGroup();

     AmbientLight lightA = new AmbientLight();
     BoundingSphere bs = new BoundingSphere(new Point3d(0.0,0.0,0.0),100.0);
     lightA.setInfluencingBounds(bs);
     objRoot.addChild(lightA);

     DirectionalLight lightD = new DirectionalLight();
     lightD.setInfluencingBounds(bs);
     Vector3f direction = new Vector3f(-1.0f, -1.0f, -0.5f);
     direction.normalize();
     lightD.setDirection(direction);
     lightD.setColor(new Color3f(0.8f, 0.8f, 0.8f));
     objRoot.addChild(lightD);

     // Set the background (essentially the "sky" in
     // the game) to yellow
     //
     Background bg = new Background();
     bg.setColor(1.0f, 1.0f, 0.0f); // yellow
     bg.setApplicationBounds(new BoundingSphere());
     objRoot.addChild(bg);

     // Build a big flat box for the land.
     Color3f black = new Color3f(0.0f, 0.0f, 0.0f);
     Color3f white = new Color3f(1.0f, 1.0f, 1.0f);
     Appearance appL = new Appearance();
     Color3f colorL = new Color3f(0.8f, 0.0f, 0.0f);
     appL.setMaterial(new Material(colorL, black, colorL, white, 80.0f));
     Box land = new Box(30.0f, 0.5f, 30.0f, appL);
     // Set type indicator for floor
     (land.getShape(Box.TOP)).setUserData(new Integer(FLOOR));
     addAt(objRoot, land, new Vector3f(0.0f, -0.5f, 0.0f));

     // Create a ball and add it to the scene graph
     Ball ball = new Ball(simpleU.getViewingPlatform().getViewPlatformTransform());
     objRoot.addChild(ball.getNode());

     // Put some buildings into the scene
     addAt(objRoot, (new Building()).getNode(), new Vector3f(2.5f, 0.0f, -4.25f));
     addAt(objRoot, (new Building()).getNode(), new Vector3f(-3.0f, 0.0f, -5.0f));
     addAt(objRoot, (new Building()).getNode(), new Vector3f(7.0f, 0.0f, -9.0f));

     // Add a ColorCube (the "prize")
     // Place it in one of the three buildings at random
     //
     Vector3f spinLoc;
```

```
        double choice = Math.random();
        if (choice < 0.333)
                spinLoc = new Vector3f(2.5f, 1.0f, -4.25f);
        else if (choice < 0.667)
                spinLoc = new Vector3f(-3.0f, 1.0f, -5.0f);
        else
                spinLoc = new Vector3f(7.0f, 1.0f, -9.0f);

        addAt(objRoot, (new SpinningCube()).getNode(), spinLoc);

                // This will move the ViewPlatform back a bit so the
                // objects in the scene can be viewed.
        TransformGroup vpTrans = simpleU.getViewingPlatform().getViewPlatformTransform
();
        Vector3f translate = new Vector3f(0.0f, 0.4f, 4.0f);
        Transform3D T3D = new Transform3D();
        T3D.setTranslation(translate);
        vpTrans.setTransform(T3D);

        //...and, set up a KeyNavigator to let us move about
        //
        KeyNavigatorBehavior keyNav = new KeyNavigatorBehavior(vpTrans);
        keyNav.setSchedulingBounds(new BoundingSphere(new Point3d(), 1000.0));
        objRoot.addChild(keyNav);

        return objRoot;
    } // end of CreateSceneGraph method of Seek

    public Seek() {
        setLayout(new BorderLayout());
        Canvas3D canvas3D = new Canvas3D(SimpleUniverse.getPreferredConfiguration());
        add("Center", canvas3D);

        // SimpleUniverse is a Convenience Utility class
        simpleU = new SimpleUniverse(canvas3D);

        BranchGroup scene = createSceneGraph();
        scene.compile();

        simpleU.addBranchGraph(scene);
    }

    //  The following allows this to be run as an application
    //  as well as an applet

    public static void main(String[] args) {
        Frame frame = new MainFrame(new Seek(), 256, 256);
    } // end of main (method of Seek)

} // end of class Seek
```