A Provisional Patent Application

for the

Spike Interface Embodied Virtual Environment

David Wallace Croft

2008 Jun 10 Tue

Description

The Spike Interface Embodied Virtual Environment (SIEVE) is a method for experimenting with cognitive models embodied in a virtual reality environment. Examples of an experimental cognitive model include a biologically realistic neuronal network or a set of rules for interaction behaviors. SIEVE is novel in that the only inputs from the virtual reality environment to the cognitive model and the only outputs from the cognitive model to the virtual reality environment are in the form of simulated spikes, the action potentials that propagate information along a nerve axon.

In biology, the amplitude and duration of one spike tends to be indistinguishable from another. The information that is carried by the spike is communicated solely by the timing of its arrival at the interface of the transmitting axon and the receiver. SIEVE indicates the existence of a spike by the value of a bit, either a one or a zero, sampled periodically during a phase of the simulation loop. A bit array can be used to store the states of a collection of simulated axons such as those in a sensory or an effector nerve bundle.

The phases of the simulation loop of SIEVE are illustrated in Figure 1. The circles at the top and bottom of the figure represent the beginning and ending of the simulation loop, respectively. In the first phase, “Render Virtual Environment”, the current state of the modeled virtual reality environment is rendered. An example of this includes rendering a 3D scene graph into a pixel buffer. Note that a virtual reality environment could include non-visual components that are rendered such as audio or haptics.

The second phase of the loop is “Convert Rendering to Sensory Spikes”. An example of this would be to convert the red, green, and blue components of the rendered pixels into simulated optic nerve spikes. The simulation serves as an artificial retina by mapping the color intensity values into ones or zeros in an array of bits representing the current state of a nerve bundle. Note that the sensory spikes represented in SIEVE can include visual, auditory, olfactory, and other senses as all are transduced into nerve impulses on their way toward the central nervous system.

The third phase is “Process Sensory Spikes / Generate Effector Spikes”. While sensory spikes travel along afferent nerves toward the central nervous system for cognitive processing, effector spikes travel along efferent nerves from the central nervous system toward effectors such as simulated muscles that

manipulate the virtual reality environment. This phase is performed by the simulated cognitive model which acts as a plug-in within the SIEVE framework. This permits the experimenter to test different cognitive models with the same "spikes in / spikes out" interface to the virtual reality environment.

The fourth phase is to "Convert Effector Spikes to Body Effects". An example of this would be a spike stimulating a muscle motor unit to effect a contraction of the muscle and a force on a limb. Another example is the opening or closing of a pore in response to nerve stimulation. This can be simulated by reading the bit array representing the effector spikes and converting the values into corresponding forces on the virtual body. By virtual body, I refer to a simulated body for the cognitive model that is located and oriented within the virtual reality environment by which the cognitive model can move within and interact with the virtual reality environment.

The fifth phase of the simulation loop is to "Update Virtual Environment". In this final stage of the loop, the state of the virtual reality environment is updated incrementally. This includes converting the forces on the body into accelerations which update the velocities of objects within the virtual reality environment. For example, effector spikes generating limb muscle contraction forces can change the orientation and position of the embodied cognitive model within the virtual reality environment. Updates can include interactions between the embodied cognitive model and other objects. These changes to state of the virtual reality environment are then rendered in the next loop of the simulation.

Implementation

My initial implementation of SIEVE, CroftSoft SIEVE, was a 3D graphics simulation written in the computer programming language Java. The source code listing for the application-specific sections of CroftSoft SIEVE is attached to this document. The attached code depends on a general-purpose reusable code library, the CroftSoft Code Library, which is distributed separately under the terms of an Open Source license as project "CroftSoft" from the public source code repository SourceForge.net. The attached CroftSoft SIEVE code is proprietary and provides an implementation of the invention; the CroftSoft Code Library on which it depends is published and does not provide an implementation of the invention by itself. The attached source code listing for CroftSoft SIEVE is sufficient, with or without the CroftSoft Code Library, to instruct an individual with expertise in the Java programming language in how to implement the invention.

Figure 2 shows a screenshot of CroftSoft SIEVE. At the top of the screenshot is a number displaying the average number of frames (animations) per second rendered as part of the simulation loop. In this case it is approximately the same as the computer screen refresh rate, 60 Hertz.

Below the average frame rate on the screenshot is a rendering of a multi-colored spinning cube which exists in the virtual reality environment. The rendering is from the position and the orientation of the viewpoint of the virtual body of the embodied cognitive model. Three of the faces of the spinning cube are within view: the cyan, the magenta, and the blue colored faces. This depicts the initial "Render Virtual Environment" phase of the simulation loop.

Below the spinning cube is the same cube rendered three more times. Each of the three renderings provides a depiction of the virtual reality environment as decomposed into just the red, green, or blue color components separately. As can be seen, the cyan color of the cube face decomposes into green and blue but no red components and the magenta decomposes into red and blue but no green components. This depicts an intermediate step in the "Convert Rendering to Sensory Spikes" phase of the simulation loop.

The bottom row of the screenshot shows a subsequent conversion of the red, green, and blue pixels into sensory spikes. Each dot represents an action potential on the axon of a simulated optic nerve. The probability of a spike at a particular position was based on the intensity of the decomposed color at a given pixel position. This simulates the conversion of light falling on long, medium, and short wavelength retinal receptor cells into optic nerve impulses. This depicts the final step in the "Convert Rendering to Sensory Spikes" phase of the simulation loop in which a color pixel buffer is converted into a sensory spike bit array.

Figure 3 shows a screenshot of CroftSoft SIEVE when the viewpoint of the virtual body of the embodied cognitive model is at a greater distance from the spinning cube. In the "Process Sensory Spikes / Generate Effector Spikes" phase of the simulation loop, the cognitive model that I provided counted the number of red spikes in the sensory spike bit array and compared them to the number of blue spikes. If the number of red spikes was greater than the number of blue spikes, the cognitive model would set a bit on the effector spike bit array to a one to indicate that the virtual body should move forward. If the number of blue spikes exceeded the number of red spikes, a different effector bit is set to move the virtual body backward away from the cube. I implemented it here in this first example using simple rules but the same behavior can also be modeled as a two neuron winner-take-all neuronal network circuit.

In the "Convert Effector Spikes to Body Effects" phase of the simulation loop, I convert the effector spikes into forces on the virtual body. The conversion here is from one and zeros in the effector spike bit array into simulated magnitudes and directions of forces. In this implementation, I limited the directions of the forces to just one degree of freedom, either toward or away from the spinning cube.

In the final "Update Virtual Environment" phase of the simulation loop, I convert the net force on the virtual body into acceleration which then affects the velocity and position of the virtual body. I also update other aspects of the virtual reality environment such as rotating the spinning cube and enforcing position boundary constraints. When the updated position of the virtual body and rotation of the spinning cube are rendered again at the beginning of the next simulation loop, the animation will provide the appearance of a cognitive model exhibiting a behavior in which it moves in the virtual reality environment toward or away from the spinning cube depending on which colors are currently showing.

This demonstrates the SIEVE invention including all phases of the simulation loop in that a cognitive model interacts with a virtual reality environment using sensory spikes as the sole inputs and effector spikes as the sole outputs. The SIEVE simulation framework renders a virtual reality environment and converts it into sensory spike inputs provided to the cognitive model; it converts the generated effector spikes into movement of the embodied cognitive model in the updated virtual reality environment via translation of the viewpoint.

Applications

The SIEVE invention has applications in cognitive modeling, robotics, and neural-machine interfaces.

In cognitive modeling, it is difficult to determine how the operation of low-level neuronal network circuits of the brain result in high-level behaviors of the body interacting with the environment. By enforcing a discipline of restricting all inputs and outputs to and from the cognitive model to be in the form of spikes, SIEVE filters out experimental models that are too abstract to be biologically plausible. This is analogous to the reason that many artificial intelligence (AI) researchers prefer to test their theories using robotics. It is more difficult, and therefore more significant, to demonstrate AI behavior

in a robot processing noisy pixels captured by a camera than to write a simulation in which the inputs are provided as symbolic tokens.

Many robotics AI researchers do depend on simulated virtual reality environments during the initial training phase before field validation. SIEVE provides such an environment for those researchers who seek to implement AI robotics based on reverse engineering of the brain, specifically neuronal network models. By neuronal network models, I refer to those which process inputs and outputs as spikes as opposed to traditional artificial neural networks (ANNs) such as multilayer perceptrons.

Neural-machine interfaces are becoming more common as the field of neuroprosthetics advances. Examples of neural-machine interfaces currently in medical use or in the research lab include cochlear implants, artificial retinas, and artificial limbs controlled by signals from cortical electrodes. A common feature of these devices is that they convert electrical signals to or from nerve impulses. In SIEVE, a cognitive model interacts with a virtual reality environment via spike inputs and outputs. Using neural-machine interface devices and SIEVE, a person can interact with a virtual reality environment in place of the cognitive model.

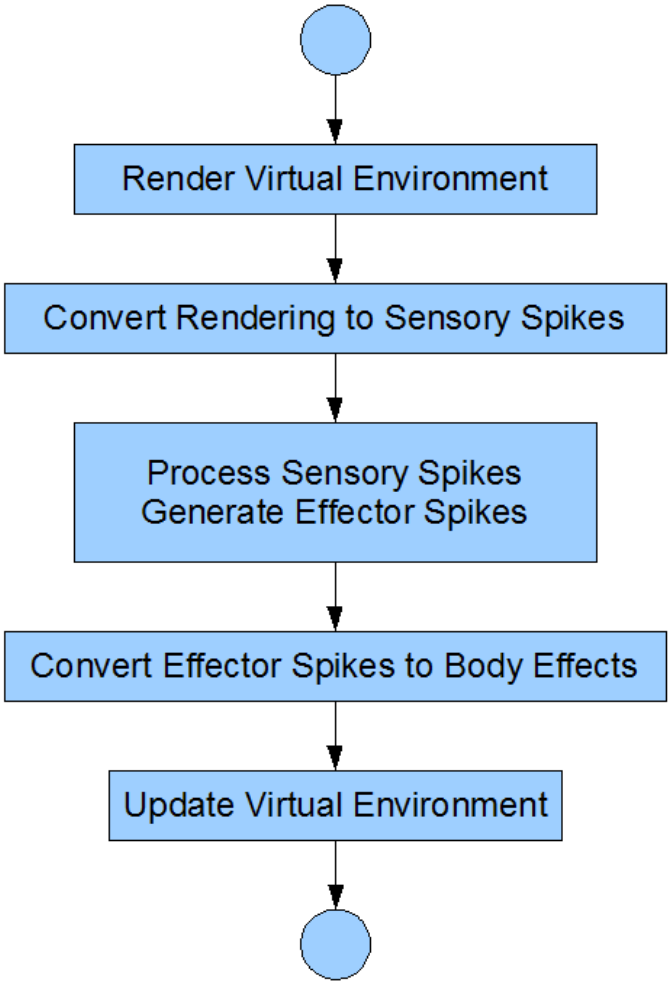

Figure 1: Phases of the SIEVE simulation loop.

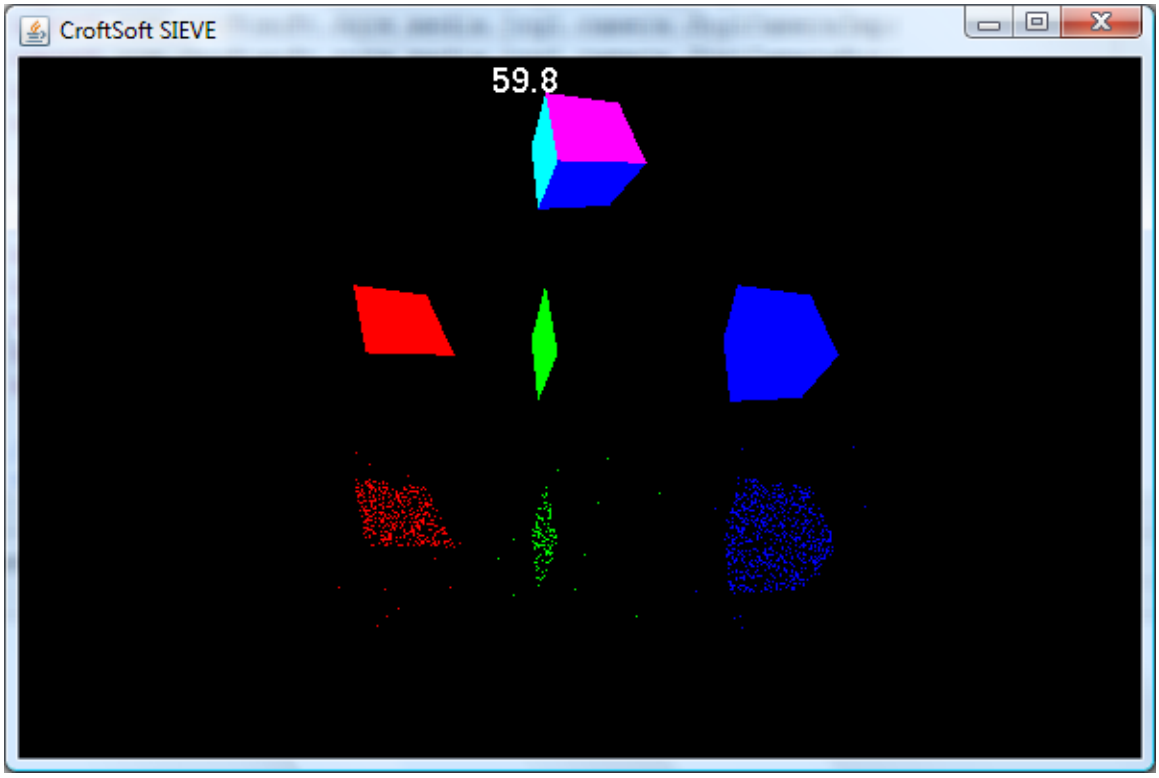

Figure 2. Screenshot of CroftSoft SIEVE.

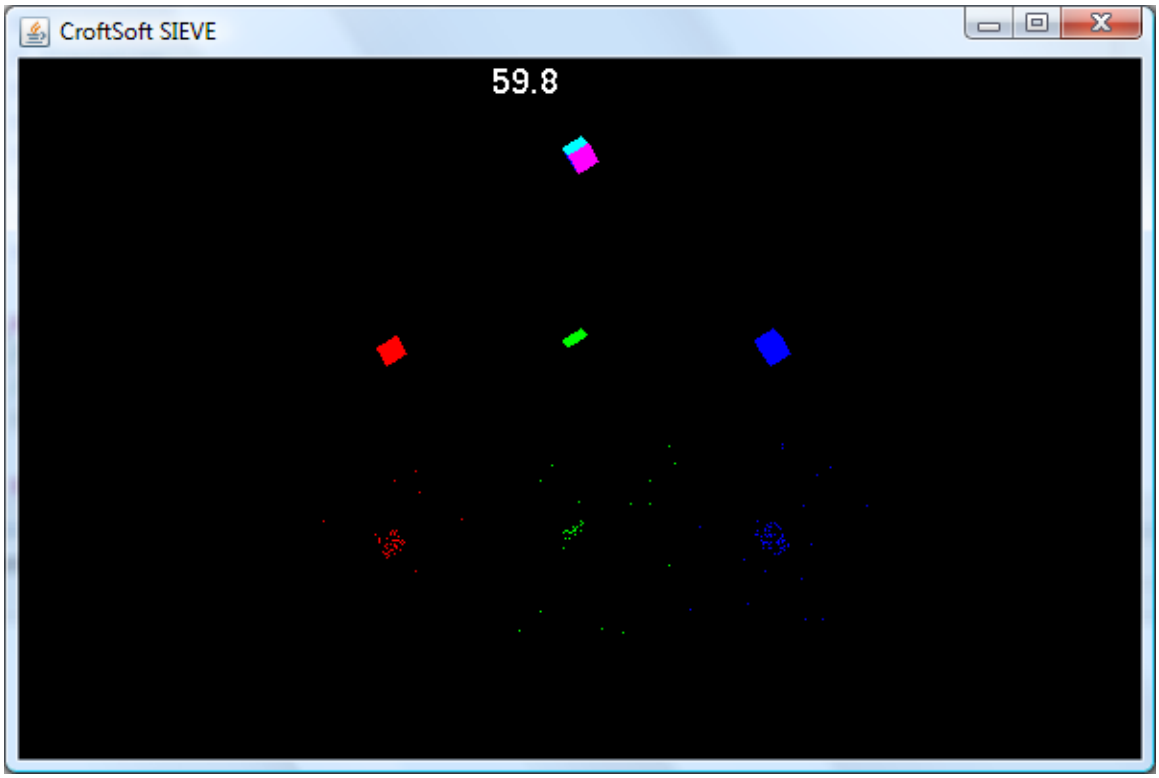

Figure 3. Spinning cube from a distance.

SieveMain.java

```
package com.croftsoft.apps.sieve;

import java.awt.*;
import javax.swing.*;

import com.croftsoft.core.gui.LifecycleWindowListener;
import com.croftsoft.core.lang.lifecycle.Lifecycle;
import com.croftsoft.core.lang.lifecycle.LifecycleLib;
import com.croftsoft.core.lang.lifecycle.Updatable;
import com.croftsoft.core.util.loop.EventQueueUpdateLoop;
import com.croftsoft.core.util.loop.Looper;
import com.croftsoft.core.util.loop.NanoTimeLoopGovernor;
import com.croftsoft.core.util.mail.FlipMail;

import com.croftsoft.apps.sieve.imp.SieveConfigImp;
import com.croftsoft.apps.sieve.imp.SieveMindContextImp;
import com.croftsoft.apps.sieve.imp.SieveModelImp;
import com.croftsoft.apps.sieve.mind.SieveMindImp;

/*********************************************************************
* Main.
*
* Launches the application within a framework.
*
* @version
*   $Id: SieveMain.java 232 2008-05-18 20:41:49Z root $
* @since
*   2008-02-18
* @author
*   <a href="http://www.CroftSoft.com/">David Wallace Croft</a>
*********************************************************************/

public final class  SieveMain
  implements Lifecycle
/////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////
{

private static final String  DEFAULT_SIEVE_MIND_CLASS_NAME
  = SieveMindImp.class.getCanonicalName ( );

//

private final SieveConfig      sieveConfig;

private final SieveModelImp    sieveModelImp;

private final SieveController  sieveController;

private final SieveView        sieveView;

private final SieveMind        sieveMind;

private final Looper           looper;

////////////////////////////////////////////////////////////////////////
// public static methods
////////////////////////////////////////////////////////////////////////

public static void  main ( final String [ ]  args )
////////////////////////////////////////////////////////////////////////
{
  final SieveMain  sieveMain = new SieveMain ( args );

  final JFrame  jFrame = new JFrame (
    sieveMain.sieveConfig.getFrameTitle ( ) );

  sieveMain.setContentPane ( jFrame.getContentPane ( ) );

  // The Frame is the framework.

  LifecycleWindowListener.launchFrameAsDesktopApp (
```

```
    jFrame,
    sieveMain,
    sieveMain.sieveConfig.getFrameSize ( ),
    sieveMain.sieveConfig.getShutdownConfirmationPrompt ( ) );
}

////////////////////////////////////////////////////////////////////////
// constructor methods
////////////////////////////////////////////////////////////////////////

public  SieveMain ( final String [ ]  args )
////////////////////////////////////////////////////////////////////////
{
  sieveConfig = SieveConfigImp.load ( args );

  System.out.println ( "\n" + sieveConfig.getInfo ( ) );

  final FlipMail<SieveMessage>  flipMail
    = new FlipMail<SieveMessage> ( );

  sieveModelImp = new SieveModelImp (
    sieveConfig,
    flipMail );

  sieveController = new SieveController (
    flipMail,
    sieveModelImp.getSieveEffectorImp ( ) );

  sieveView = new SieveView (
    flipMail,
    sieveModelImp,
    sieveModelImp.getSensorSpikeData ( ) );

  sieveView.addKeyListener ( sieveController );

  sieveView.addMouseListener ( sieveController );
```

```
String  sieveMindClassName = DEFAULT_SIEVE_MIND_CLASS_NAME;

if ( ( args != null    )
  && ( args.length > 0 ) )
{
  sieveMindClassName = args [ 0 ];
}

try
{
  final Class<?>  sieveMindClass
    = Class.forName ( sieveMindClassName );

  sieveMind = ( SieveMind ) sieveMindClass.newInstance ( );
}
catch ( final Exception  ex )
{
  throw ( IllegalArgumentException )
    new IllegalArgumentException ( ).initCause ( ex );
}

sieveMind.setSieveMindContext (
  new SieveMindContextImp (
    sieveModelImp.getEffectorSpikeData ( ),
    sieveModelImp.getSensorSpikeData   ( ) ) );

final Updatable [ ]  updatables = new Updatable [ ] {
  flipMail,
  sieveModelImp,
  sieveView,
  sieveController,
  sieveMind };

looper = new Looper (
  new EventQueueUpdateLoop ( updatables ), // loopable
  new NanoTimeLoopGovernor ( sieveConfig.getUpdateRate ( ) ),
  null, // exceptionHandler
```

```
    sieveConfig.getThreadName ( ),
    Thread.MIN_PRIORITY,
    true ); // useDaemonThread
}

////////////////////////////////////////////////////////////////////////
// accessor methods
////////////////////////////////////////////////////////////////////////

public SieveConfig  getSieveConfig ( )
////////////////////////////////////////////////////////////////////////
{
  return sieveConfig;
}

////////////////////////////////////////////////////////////////////////
// mutator methods
////////////////////////////////////////////////////////////////////////

public void  setContentPane ( final Container  contentPane )
////////////////////////////////////////////////////////////////////////
{
  sieveView.setContentPane ( contentPane );
}

////////////////////////////////////////////////////////////////////////
// interface Lifecycle methods
////////////////////////////////////////////////////////////////////////

public void  init ( )
////////////////////////////////////////////////////////////////////////
{
  LifecycleLib.init ( sieveView, sieveMind, looper );
}

public void  start ( )
////////////////////////////////////////////////////////////////////////
```

```
{
  LifecycleLib.start ( sieveController, looper );
}

public void  stop ( )
//////////////////////////////////////////////////////////////////////////
{
  LifecycleLib.stop ( looper );
}

public void  destroy ( )
//////////////////////////////////////////////////////////////////////////
{
  LifecycleLib.destroy ( sieveMind, looper );
}

//////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////
}
```

SieveConfig.java

```
package com.croftsoft.apps.sieve;

import java.awt.*;

/*********************************************************************
* Configuration interface.
*
* @version
*   $Id: SieveConfig.java 232 2008-05-18 20:41:49Z root $
* @since
*   2008-02-18
* @author
*   <a href="http://www.CroftSoft.com/">David Wallace Croft</a>
*********************************************************************/

public interface  SieveConfig
//////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////
{

String     getInfo ( );

Dimension  getFrameSize ( );

String     getFrameTitle ( );

String     getShutdownConfirmationPrompt ( );

String     getThreadName ( );

double     getUpdateRate ( );

//////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////
}
```

SieveController.java

```
package com.croftsoft.apps.sieve;

import java.awt.event.*;
import java.util.*;

import com.croftsoft.core.gui.controller.NilController;
import com.croftsoft.core.lang.NullArgumentException;
import com.croftsoft.core.lang.lifecycle.Startable;
import com.croftsoft.core.util.mail.Mail;

import com.croftsoft.apps.sieve.body.SieveEffectorImp;
import com.croftsoft.apps.sieve.body.SieveEffectorImp.Effector;

/*********************************************************************
* Controller.
*
* Modifies the Model based on user input.
*
* @version
*   $Id: SieveController.java 228 2008-05-17 01:01:30Z root $
* @since
*   2008-02-18
* @author
*   <a href="http://www.CroftSoft.com/">David Wallace Croft</a>
*********************************************************************/

public final class  SieveController
  extends NilController
  implements Startable
//////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////
{

private final Mail<SieveMessage>  mail;
```

```
private final SieveEffectorImp    sieveEffectorImp;

private final BitSet              bitSet;

//

private boolean  mouseClicked;

////////////////////////////////////////////////////////////////////////
// constructor method
////////////////////////////////////////////////////////////////////////

public  SieveController (
  final Mail<SieveMessage>  mail,
  final SieveEffectorImp    sieveEffectorImp )
////////////////////////////////////////////////////////////////////////
{
  NullArgumentException.checkArgs (
    this.mail             = mail,
    this.sieveEffectorImp = sieveEffectorImp );

  bitSet = new BitSet ( Effector.values ( ).length );
}

////////////////////////////////////////////////////////////////////////
// listener methods
////////////////////////////////////////////////////////////////////////

@Override
public void  keyPressed  ( final KeyEvent  keyEvent )
////////////////////////////////////////////////////////////////////////
{
  processKeyEvent ( keyEvent, true );
}

@Override
public void  keyReleased  ( final KeyEvent  keyEvent )
```

```
////////////////////////////////////////////////////////////////////////
{
  processKeyEvent ( keyEvent, false );
}

@Override
public void  mouseClicked ( final MouseEvent  mouseEvent )
////////////////////////////////////////////////////////////////////////
{
  mouseClicked = true;
}

@Override
public void  mouseMoved ( final MouseEvent  mouseEvent )
////////////////////////////////////////////////////////////////////////
{
  //mouseMoved = true;
}

////////////////////////////////////////////////////////////////////////
// lifecycle methods
////////////////////////////////////////////////////////////////////////

public void  start ( )
////////////////////////////////////////////////////////////////////////
{
  mouseClicked = false;
}

@Override
public void  update ( )
////////////////////////////////////////////////////////////////////////
{
  if ( !bitSet.isEmpty ( ) )
  {
    sieveEffectorImp.setSpiking ( bitSet );
  }
```

```
  if ( mouseClicked )
  {
    mouseClicked = !mail.offer (
      SieveMessage.TOGGLE_PAUSE_REQUEST_INSTANCE );
  }
}

////////////////////////////////////////////////////////////////////////
// private methods
////////////////////////////////////////////////////////////////////////

private Effector  getEffector ( final KeyEvent  keyEvent )
////////////////////////////////////////////////////////////////////////
{
  final int  keyCode = keyEvent.getKeyCode ( );

  final boolean  shiftDown = keyEvent.isShiftDown ( );

  switch ( keyCode )
  {
    case KeyEvent.VK_A:

      return Effector.LEFT;

    case KeyEvent.VK_D:

      return Effector.RIGHT;

    case KeyEvent.VK_W:

      return Effector.FORWARD;

    case KeyEvent.VK_S:

      return Effector.BACKWARD;
```

```
case KeyEvent.VK_UP:

  if ( shiftDown )
  {
    return Effector.PITCH_DOWN;
  }

  return Effector.UP;

case KeyEvent.VK_DOWN:

  if ( shiftDown )
  {
    return Effector.PITCH_UP;
  }

  return Effector.DOWN;

case KeyEvent.VK_LEFT:

  if ( shiftDown )
  {
    return Effector.ROLL_LEFT;
  }

  return Effector.YAW_LEFT;

case KeyEvent.VK_RIGHT:

  if ( shiftDown )
  {
    return Effector.ROLL_RIGHT;
  }

  return Effector.YAW_RIGHT;

default:
```

```
      return null;
  }
}

private void  processKeyEvent (
  final KeyEvent  keyEvent,
  final boolean   keyPressed )
//////////////////////////////////////////////////////////////////////
{
  final Effector  effector = getEffector ( keyEvent );

  if ( effector != null )
  {
    bitSet.set (
      sieveEffectorImp.getOffset ( effector ),
      keyPressed );
  }
}

//////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////
}
```

SieveMessage.java

```
package com.croftsoft.apps.sieve;

import com.croftsoft.core.lang.NullException;

/*********************************************************************
* Jogl enumerated type message.
*
* Use to pass messages between the model, view, and controller.
*
* @version
*   $Id: SieveMessage.java 232 2008-05-18 20:41:49Z root $
* @since
*   2008-02-18
* @author
*   <a href="http://www.CroftSoft.com/">David Wallace Croft</a>
*********************************************************************/

public class  SieveMessage
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
{

public enum  Type
{
  TOGGLE_PAUSE_EVENT,
  TOGGLE_PAUSE_REQUEST,
}

public static SieveMessage
  TOGGLE_PAUSE_EVENT_INSTANCE
    = new SieveMessage ( Type.TOGGLE_PAUSE_EVENT ),
  TOGGLE_PAUSE_REQUEST_INSTANCE
    = new SieveMessage ( Type.TOGGLE_PAUSE_REQUEST );

//
```

```
private final Type  type;

////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////

public Type  getType ( ) { return type; }

////////////////////////////////////////////////////////////////////////
// protected methods
////////////////////////////////////////////////////////////////////////

protected  SieveMessage ( final Type  type )
////////////////////////////////////////////////////////////////////////
{
  NullException.check ( this.type = type );
}

////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
}
```

SieveMind.java

```
package com.croftsoft.apps.sieve;

import com.croftsoft.core.lang.lifecycle.Destroyable;
import com.croftsoft.core.lang.lifecycle.Initializable;
import com.croftsoft.core.lang.lifecycle.Updatable;

/*********************************************************************
* Cognitive model.
*
* Processes incoming sensory spikes and generates outgoing effector
* spikes.
*
* @version
*   $Id: SieveMind.java 237 2008-05-23 20:46:18Z root $
* @since
*   2008-04-11
* @author
*   <a href="http://www.CroftSoft.com/">David Wallace Croft</a>
*********************************************************************/

public interface  SieveMind
  extends Initializable, Destroyable, Updatable
//////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////
{

void  setSieveMindContext ( SieveMindContext  sieveMindContext );

//////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////
}
```

SieveMindContext.java

```
package com.croftsoft.apps.sieve;

/*********************************************************************
* Interface to the framework for a SieveMind.
*
* @version
*   $Id: SieveMindContext.java 237 2008-05-23 20:46:18Z root $
* @since
*   2008-04-11
* @author
*   <a href="http://www.CroftSoft.com/">David Wallace Croft</a>
*********************************************************************/

public interface  SieveMindContext
//////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////
{

String [ ]  getEffectorLabels  ( );

int         getEffectorOffset  ( String  effectorLabel );

int         getEffectorLength  ( String  effectorLabel );

String [ ]  getSensorLabels    ( );

int         getSensorOffset    ( String  sensorLabel );

int         getSensorLength    ( String  sensorLabel );

boolean     isSensorSpiking    ( int  index );

void        setEffectorSpiking ( int  index );

//////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////
}
```

SieveModel.java

```
package com.croftsoft.apps.sieve;

import com.croftsoft.core.math.axis.AxisAngle;
import com.croftsoft.core.media.jogl.render.JoglSpinCube;

/*********************************************************************
* SIEVE model accessor interface.
*
* Read-only access to model state.
*
* @version
*   $Id: SieveModel.java 232 2008-05-18 20:41:49Z root $
* @since
*   2008-02-18
* @author
*   <a href="http://www.CroftSoft.com/">David Wallace Croft</a>
*********************************************************************/

public interface  SieveModel
//////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////
{

enum  EnumInt
{
  OPTIC_HEIGHT,
  OPTIC_WIDTH,
}

int  get ( EnumInt  enumInt );

AxisAngle  getOrientation ( );

double  getPositionX ( );
```

```
double  getPositionY ( );

double  getPositionZ ( );

JoglSpinCube.Model  getJoglSpinCubeModel ( );

boolean  isSimulationRunning ( );

//////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////
}
```

SieveRenderer.java

```
package com.croftsoft.apps.sieve;

import java.nio.*;

import javax.media.opengl.GL;
import javax.media.opengl.glu.GLU;

import com.croftsoft.apps.sieve.SieveModel.EnumInt;
import com.croftsoft.apps.sieve.body.SieveSensor;
import com.croftsoft.core.lang.NullException;
import com.croftsoft.core.math.axis.AxisAngle;
import com.croftsoft.core.media.jogl.JoglLib;
import com.croftsoft.core.media.jogl.JoglRenderer;
import com.croftsoft.core.media.jogl.render.JoglFrameRate;
import com.croftsoft.core.media.jogl.render.JoglSpinCube;

/*********************************************************************
* ComponentAnimator.
*
* @version
*   $Date: 2008-05-16 21:45:41 -0500 (Fri, 16 May 2008) $
* @since
*   2008-02-18
* @author
*   <a href="http://www.CroftSoft.com/">David Wallace Croft</a>
*********************************************************************/

public final class  SieveRenderer
  implements JoglRenderer
/////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////
{

private final SieveModel        sieveModel;
```

```
private final SieveSpikeData    sensorSpikeData;

private final JoglFrameRate     joglFrameRate;

private final JoglSpinCube      joglSpinCube;

private final JoglRenderer [ ]  joglRenderers;

private final int
  opticWidth,
  opticHeight,
  opticOffset;

//

private int
  width,
  height,
  sceneOffsetX,
  sceneOffsetY;

private ByteBuffer
  byteBuffer,
  redByteBuffer,
  greenByteBuffer,
  blueByteBuffer;

////////////////////////////////////////////////////////////////////////
// constructor methods
////////////////////////////////////////////////////////////////////////

/**********************************************************************
* Main constructor.
**********************************************************************/
public  SieveRenderer (
  final SieveModel      sieveModel,
  final SieveSpikeData  sensorSpikeData )
```

```
////////////////////////////////////////////////////////////////////////
{
  NullException.check (
    this.sieveModel      = sieveModel,
    this.sensorSpikeData = sensorSpikeData );

  opticWidth  = sieveModel.get ( EnumInt.OPTIC_WIDTH );

  opticHeight = sieveModel.get ( EnumInt.OPTIC_HEIGHT );

  opticOffset = sensorSpikeData.getOffset ( SieveSensor.LABEL_OPTIC );

  joglFrameRate = new JoglFrameRate ( );

  joglRenderers = new JoglRenderer [ ] {
    joglSpinCube
      = new JoglSpinCube ( sieveModel.getJoglSpinCubeModel ( ) ),
    joglFrameRate };
}

////////////////////////////////////////////////////////////////////////
// interface GLEventListener methods
////////////////////////////////////////////////////////////////////////

public void init ( final GL  gl )
////////////////////////////////////////////////////////////////////////
{
  JoglLib.printInfo ( gl );

  gl.glClearColor ( 0f, 0f, 0f, 0f );

  for ( final JoglRenderer  joglRenderer : joglRenderers )
  {
    joglRenderer.init ( gl );
  }

  JoglLib.checkError ( gl );
```

```
}

public void  destroy ( final GL  gl )
//////////////////////////////////////////////////////////////////////
{
  for ( final JoglRenderer  joglRenderer : joglRenderers )
  {
    joglRenderer.destroy ( gl );
  }

  JoglLib.checkError ( gl );
}

public void  setBounds (
  final GL   gl,
  final int  x,
  final int  y,
  final int  width,
  final int  height )
//////////////////////////////////////////////////////////////////////
{
  this.width = width;

  this.height = height;

  sceneOffsetX = ( width - opticWidth ) / 2;

  sceneOffsetY = height - opticHeight;

  gl.glViewport (
    sceneOffsetX,
    sceneOffsetY,
    opticWidth,
    opticHeight );

  gl.glMatrixMode ( GL.GL_PROJECTION );
```

```
  gl.glLoadIdentity ( );

  new GLU ( ).gluPerspective (
    45.0,
    ( double ) opticWidth / ( double ) opticHeight,
    1,
    100 );

  final int  byteCount = 4 * opticWidth * opticHeight;

  byteBuffer      = ByteBuffer.allocateDirect ( byteCount );

  redByteBuffer   = ByteBuffer.allocateDirect ( byteCount );

  greenByteBuffer = ByteBuffer.allocateDirect ( byteCount );

  blueByteBuffer  = ByteBuffer.allocateDirect ( byteCount );

  for ( final JoglRenderer  joglRenderer : joglRenderers )
  {
    joglRenderer.setBounds ( gl, x, y, opticWidth, opticHeight );
  }

  joglFrameRate.setOffsetX ( 4 );

  joglFrameRate.setOffsetY ( opticHeight - 18 );

  JoglLib.checkError ( gl );
}

public void  render ( final GL  gl )
////////////////////////////////////////////////////////////////////////
{
  gl.glClear (
      GL.GL_COLOR_BUFFER_BIT
    | GL.GL_DEPTH_BUFFER_BIT );
```

```
gl.glMatrixMode ( GL.GL_MODELVIEW );

gl.glLoadIdentity ( );

final AxisAngle  axisAngle = sieveModel.getOrientation ( );

gl.glRotated (
  -axisAngle.getDegrees ( ),
  axisAngle.getX ( ),
  axisAngle.getY ( ),
  axisAngle.getZ ( ) );

gl.glTranslated (
  -sieveModel.getPositionX ( ),
  -sieveModel.getPositionY ( ),
  -sieveModel.getPositionZ ( ) );

joglSpinCube.render ( gl );

gl.glFinish ( );

gl.glReadPixels (
  sceneOffsetX,
  sceneOffsetY,
  opticWidth,
  opticHeight,
  GL.GL_RGBA,
  GL.GL_UNSIGNED_BYTE,
  byteBuffer );

byteBuffer.rewind ( );

redByteBuffer.put ( byteBuffer ).flip ( );

byteBuffer.rewind ( );

greenByteBuffer.put ( byteBuffer ).flip ( );
```

```
byteBuffer.rewind ( );

blueByteBuffer.put ( byteBuffer ).flip ( );

byteBuffer.rewind ( );

for ( int  i = 0; i < opticWidth * opticHeight; i++ )
{
  redByteBuffer  .put ( 4 * i + 1, ( byte ) 0 );

  redByteBuffer  .put ( 4 * i + 2, ( byte ) 0 );

  greenByteBuffer.put ( 4 * i + 0, ( byte ) 0 );

  greenByteBuffer.put ( 4 * i + 2, ( byte ) 0 );

  blueByteBuffer .put ( 4 * i + 0, ( byte ) 0 );

  blueByteBuffer .put ( 4 * i + 1, ( byte ) 0 );
}

final int  offsetX = ( width - 3 * opticWidth ) / 2;

int  offsetY = height - 2 * opticHeight;

gl.glWindowPos2d ( offsetX, offsetY );

gl.glDrawPixels (
  opticWidth,
  opticHeight,
  GL.GL_RGBA,
  GL.GL_UNSIGNED_BYTE,
  redByteBuffer );

gl.glWindowPos2d ( offsetX + opticWidth, offsetY );
```

```
gl.glDrawPixels (
  opticWidth,
  opticHeight,
  GL.GL_RGBA,
  GL.GL_UNSIGNED_BYTE,
  greenByteBuffer );

gl.glWindowPos2d ( offsetX + 2 * opticWidth, offsetY );

gl.glDrawPixels (
  opticWidth,
  opticHeight,
  GL.GL_RGBA,
  GL.GL_UNSIGNED_BYTE,
  blueByteBuffer );

for ( int  i = 0; i < opticWidth * opticHeight; i++ )
{
  final int  rgbOffset = opticOffset + 3 * i;

  byte  redByte, greenByte, blueByte;

  boolean  redSpike, greenSpike, blueSpike;

  if ( sieveModel.isSimulationRunning ( ) )
  {
    redByte   = redByteBuffer  .get ( 4 * i + 0 );

    greenByte = greenByteBuffer.get ( 4 * i + 1 );

    blueByte  = blueByteBuffer .get ( 4 * i + 2 );

    redSpike   = spike ( redByte   );

    greenSpike = spike ( greenByte );

    blueSpike  = spike ( blueByte  );
```

```
    if ( redSpike )
    {
      sensorSpikeData.setSpiking ( rgbOffset + 0 );
    }

    if ( greenSpike )
    {
      sensorSpikeData.setSpiking ( rgbOffset + 1 );
    }

    if ( blueSpike )
    {
      sensorSpikeData.setSpiking ( rgbOffset + 2 );
    }
  }
  else
  {
    redSpike   = sensorSpikeData.isSpiking ( rgbOffset + 0 );

    greenSpike = sensorSpikeData.isSpiking ( rgbOffset + 1 );

    blueSpike  = sensorSpikeData.isSpiking ( rgbOffset + 2 );
  }

  redByte   = redSpike   ? ( byte ) 255 : ( byte ) 0;

  greenByte = greenSpike ? ( byte ) 255 : ( byte ) 0;

  blueByte  = blueSpike  ? ( byte ) 255 : ( byte ) 0;

  redByteBuffer  .put ( 4 * i + 0, redByte   );

  greenByteBuffer.put ( 4 * i + 1, greenByte );

  blueByteBuffer .put ( 4 * i + 2, blueByte  );
}
```

```
  offsetY = offsetY - opticHeight;

  gl.glWindowPos2d ( offsetX, offsetY );

  gl.glDrawPixels (
    opticWidth,
    opticHeight,
    GL.GL_RGBA,
    GL.GL_UNSIGNED_BYTE,
    redByteBuffer );

  gl.glWindowPos2d ( offsetX + opticWidth, offsetY );

  gl.glDrawPixels (
    opticWidth,
    opticHeight,
    GL.GL_RGBA,
    GL.GL_UNSIGNED_BYTE,
    greenByteBuffer );

  gl.glWindowPos2d ( offsetX + 2 * opticWidth, offsetY );

  gl.glDrawPixels (
    opticWidth,
    opticHeight,
    GL.GL_RGBA,
    GL.GL_UNSIGNED_BYTE,
    blueByteBuffer );

  joglFrameRate.render ( gl );

  JoglLib.checkError ( gl );
}

////////////////////////////////////////////////////////////////////////
// private methods
```

```
////////////////////////////////////////////////////////////////////////

private static boolean  spike ( final byte  colorByte )
////////////////////////////////////////////////////////////////////////
{
  final int  colorInt = 0xFF & colorByte;

  final double  intensity = colorInt / 255.0;

  final double  spikeProbability  = intensity / 4 + 0.001;

  return Math.random ( ) <= spikeProbability;
}

////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
}
```

SieveSpikeData.java

```
package com.croftsoft.apps.sieve;

/*********************************************************************
* Common interface for effector and sensor spike data.
*
* @version
*   $Id: SieveSpikeData.java 237 2008-05-23 20:46:18Z root $
* @since
*   2008-04-11
* @author
*   <a href="http://www.CroftSoft.com/">David Wallace Croft</a>
*********************************************************************/

public interface  SieveSpikeData
//////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////
{

String [ ]  getLabels  ( );

int         getOffset  ( String  label );

int         getLength  ( String  label );

boolean     isSpiking  ( int  index );

void        setSpiking ( int  index );

//////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////
}
```

SieveView.java

```
package com.croftsoft.apps.sieve;

import java.awt.*;
import java.awt.event.*;

import javax.media.opengl.*;

import com.croftsoft.core.lang.NullException;
import com.croftsoft.core.lang.lifecycle.Initializable;
import com.croftsoft.core.lang.lifecycle.Updatable;
import com.croftsoft.core.media.jogl.JoglAdapter;
import com.croftsoft.core.util.mail.Mail;

/*********************************************************************
* SIEVE view.
*
* @version
*   $Id: SieveView.java 238 2008-05-23 20:55:51Z root $
* @since
*   2008-02-18
* @author
*   <a href="http://www.CroftSoft.com/">David Wallace Croft</a>
*********************************************************************/

public final class  SieveView
  implements Initializable, Updatable
///////////////////////////////////////////////////////////////////////
///////////////////////////////////////////////////////////////////////
{

private final Mail<SieveMessage>  mail;

private final GLCanvas            glCanvas;

///////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////

public  SieveView (
  final Mail<SieveMessage>  mail,
  final SieveModel          sieveModel,
  final SieveSpikeData      sensorSpikeData )
////////////////////////////////////////////////////////////////////////
{
  NullException.check (
    this.mail = mail,
    sieveModel );

  final GLCapabilities  glCapabilities = new GLCapabilities ( );

  glCanvas = new GLCanvas ( glCapabilities );

  glCanvas.addGLEventListener (
    new JoglAdapter (
      new SieveRenderer ( sieveModel, sensorSpikeData ) ) );
}

////////////////////////////////////////////////////////////////////////
// mutator methods
////////////////////////////////////////////////////////////////////////

public void  addKeyListener ( final KeyListener  keyListener )
////////////////////////////////////////////////////////////////////////
{
  glCanvas.addKeyListener ( keyListener );
}

public void  addMouseListener ( final MouseListener  mouseListener )
////////////////////////////////////////////////////////////////////////
{
  glCanvas.addMouseListener ( mouseListener );
}
```

```
public void  setContentPane ( final Container  contentPane )
//////////////////////////////////////////////////////////////////////////
{
  contentPane.setLayout ( new BorderLayout ( ) );

  contentPane.add ( glCanvas, BorderLayout.CENTER );
}

//////////////////////////////////////////////////////////////////////////
// lifecycle methods
//////////////////////////////////////////////////////////////////////////

public void  init ( )
//////////////////////////////////////////////////////////////////////////
{
  glCanvas.requestFocus ( );
}

public void  update ( )
//////////////////////////////////////////////////////////////////////////
{
  final int  size = mail.size ( );

  for ( int  i = 0; i < size; i++ )
  {
    final SieveMessage  sieveMessage = mail.get ( i );

    final SieveMessage.Type  type = sieveMessage.getType ( );

    switch ( type )
    {
      case TOGGLE_PAUSE_EVENT:

        Toolkit.getDefaultToolkit ( ).beep ( );

        break;
```

```
      default:

        // ignore
    }
  }

  glCanvas.display ( );
}

////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
}
```

SieveEffector.java

```
package com.croftsoft.apps.sieve.body;

import com.croftsoft.apps.sieve.SieveSpikeData;

/*********************************************************************
* Exemplar effector spike data interface.
*
* @version
*   $Id: SieveEffector.java 237 2008-05-23 20:46:18Z root $
* @since
*   2008-04-11
* @author
*   <a href="http://www.CroftSoft.com/">David Wallace Croft</a>
*********************************************************************/

public interface  SieveEffector
  extends SieveSpikeData
//////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////
{

public static final String  EFFECTOR_LABEL_PREFIX
  = SieveEffector.class.getCanonicalName ( ) + ".";

public static final String
  LABEL_BACKWARD = EFFECTOR_LABEL_PREFIX + "BACKWARD",
  LABEL_FORWARD  = EFFECTOR_LABEL_PREFIX + "FORWARD";

//////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////
}
```

SieveEffectorImp.java

```
package com.croftsoft.apps.sieve.body;

import java.util.*;

/*********************************************************************
* Implementation of interface SieveEffector.
*
* @version
*   $Id: SieveEffectorImp.java 237 2008-05-23 20:46:18Z root $
* @since
*   2008-04-04
* @author
*   <a href="http://www.CroftSoft.com/">David Wallace Croft</a>
*********************************************************************/

public final class  SieveEffectorImp
  implements SieveEffector
//////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////
{

public enum  Effector
{
  BACKWARD,
  DOWN,
  FORWARD,
  LEFT,
  PITCH_DOWN,
  PITCH_UP,
  RIGHT,
  ROLL_LEFT,
  ROLL_RIGHT,
  UP,
  YAW_LEFT,
  YAW_RIGHT
```

```
}

private static final int  EFFECTOR_LENGTH = Effector.values ( ).length;

private static final String
  EFFECTOR_LABEL_PREFIX = SieveEffector.EFFECTOR_LABEL_PREFIX;

private static final int
  EFFECTOR_LABEL_PREFIX_LENGTH = EFFECTOR_LABEL_PREFIX.length ( );

//

private final BitSet  bitSet;

////////////////////////////////////////////////////////////////////////
// static methods
////////////////////////////////////////////////////////////////////////

public static String [ ]  getEffectorLabels ( )
////////////////////////////////////////////////////////////////////////
{
  final List<String>  labelList = new ArrayList<String> ( );

  final Effector [ ]  effectorArray = Effector.values ( );

  for ( final Effector  effector : effectorArray )
  {
    labelList.add ( EFFECTOR_LABEL_PREFIX + effector.name ( ) );
  }

  return labelList.toArray ( new String [ 0 ] );
}

public static Effector  getEffector ( final String  fullLabel )
////////////////////////////////////////////////////////////////////////
{
  if ( !fullLabel.startsWith ( EFFECTOR_LABEL_PREFIX ) )
```

```
  {
    throw new IllegalArgumentException ( );
  }

  return Effector.valueOf (
    fullLabel.substring ( EFFECTOR_LABEL_PREFIX_LENGTH ) );
}

////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////

public  SieveEffectorImp ( )
////////////////////////////////////////////////////////////////////////
{
  bitSet = new BitSet ( EFFECTOR_LENGTH );
}

////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////

public String [ ]  getLabels ( )
////////////////////////////////////////////////////////////////////////
{
  return getEffectorLabels ( );
}

public int  getLength ( String  effectorFullLabel )
////////////////////////////////////////////////////////////////////////
{
  return getLength ( getEffector ( effectorFullLabel ) );
}

public int  getLength (
  @SuppressWarnings ( "unused" )
  final Effector  effector )
////////////////////////////////////////////////////////////////////////
{
```

```
  return 1;
}

public int  getOffset ( String  effectorFullLabel )
//////////////////////////////////////////////////////////////////////////
{
  return getOffset ( getEffector ( effectorFullLabel ) );
}

public int  getOffset ( final Effector  effector )
//////////////////////////////////////////////////////////////////////////
{
  return effector.ordinal ( );
}

public boolean  isSpiking ( final int  index )
//////////////////////////////////////////////////////////////////////////
{
  return bitSet.get ( index );
}

//////////////////////////////////////////////////////////////////////////
// mutator methods
//////////////////////////////////////////////////////////////////////////

public void  clear ( )
//////////////////////////////////////////////////////////////////////////
{
  bitSet.clear ( );
}

public void  setSpiking ( final BitSet  spikingBitSet )
//////////////////////////////////////////////////////////////////////////
{
  bitSet.or ( spikingBitSet );
}
```

```
public void  setSpiking ( final int  index )
////////////////////////////////////////////////////////////////////////
{
  bitSet.set ( index );
}

////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
}
```

SieveSensor.java

```
package com.croftsoft.apps.sieve.body;

import com.croftsoft.apps.sieve.SieveSpikeData;

/*********************************************************************
* Exemplar sensor spike data interface.
*
* @version
*   $Id: SieveSensor.java 237 2008-05-23 20:46:18Z root $
* @since
*   2008-04-11
* @author
*   <a href="http://www.CroftSoft.com/">David Wallace Croft</a>
*********************************************************************/

public interface  SieveSensor
  extends SieveSpikeData
//////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////
{

public static final String  SENSOR_LABEL_PREFIX
  = SieveSensor.class.getCanonicalName ( ) + ".";

public static final String
  LABEL_OPTIC = SENSOR_LABEL_PREFIX + "OPTIC";

//////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////
}
```

SieveSensorImp.java

```
package com.croftsoft.apps.sieve.body;

import java.util.*;

/*********************************************************************
* Implementation.
*
* @version
*   $Id: SieveSensorImp.java 237 2008-05-23 20:46:18Z root $
* @since
*   2008-04-04
* @author
*   <a href="http://www.CroftSoft.com/">David Wallace Croft</a>
*********************************************************************/

public final class  SieveSensorImp
  implements SieveSensor
//////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////
{

public enum  Sensor
{
  OPTIC,
}

private static final int
  SENSOR_LABEL_PREFIX_LENGTH = SENSOR_LABEL_PREFIX.length ( );

private static final int
  OPTIC_WIDTH     = 100,
  OPTIC_HEIGHT    = 100,
  OPTIC_LENGTH    = 3 * OPTIC_WIDTH * OPTIC_HEIGHT,
  OPTIC_OFFSET    = 0,
  SENSOR_LENGTH   = OPTIC_LENGTH;
```

```
//

private final BitSet  bitSet;

////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////

public static String  getSensorLabel ( final Sensor  sensor )
////////////////////////////////////////////////////////////////////////
{
  return SENSOR_LABEL_PREFIX + sensor.name ( );
}

public static String [ ]  getSensorLabels ( )
////////////////////////////////////////////////////////////////////////
{
  final List<String>  labelList = new ArrayList<String> ( );

  final Sensor [ ]  sensorArray = Sensor.values ( );

  for ( final Sensor  sensor : sensorArray )
  {
    labelList.add ( getSensorLabel ( sensor ) );
  }

  return labelList.toArray ( new String [ 0 ] );
}

public static Sensor  getSensor ( final String  fullLabel )
////////////////////////////////////////////////////////////////////////
{
  if ( !fullLabel.startsWith ( SENSOR_LABEL_PREFIX ) )
  {
    throw new IllegalArgumentException ( );
  }
```

```
  return Sensor.valueOf (
    fullLabel.substring ( SENSOR_LABEL_PREFIX_LENGTH ) );
}

////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////

public  SieveSensorImp ( )
////////////////////////////////////////////////////////////////////////
{
  bitSet = new BitSet ( SENSOR_LENGTH );
}

////////////////////////////////////////////////////////////////////////
// interface accessor methods
////////////////////////////////////////////////////////////////////////

public String [ ]  getLabels ( )
////////////////////////////////////////////////////////////////////////
{
  return getSensorLabels ( );
}

public int  getLength ( String  sensorFullLabel )
////////////////////////////////////////////////////////////////////////
{
  return getLength ( getSensor ( sensorFullLabel ) );
}

public int  getOffset ( String  sensorFullLabel )
////////////////////////////////////////////////////////////////////////
{
  return getOffset ( getSensor ( sensorFullLabel ) );
}

public int  getLength ( final Sensor  sensor )
////////////////////////////////////////////////////////////////////////
```

```
{
  switch ( sensor )
  {
    case OPTIC:

      return OPTIC_LENGTH;

    default:

      throw new IllegalArgumentException ( sensor.name ( ) );
  }
}

public int  getOffset ( final Sensor  sensor )
//////////////////////////////////////////////////////////////////////////
{
  switch ( sensor )
  {
    case OPTIC:

      return OPTIC_OFFSET;

    default:

      throw new IllegalArgumentException ( sensor.name ( ) );
  }
}

public int  getOpticHeight  ( ) { return OPTIC_HEIGHT;  }

public int  getOpticWidth   ( ) { return OPTIC_WIDTH;   }

public boolean  isSpiking ( final int  index )
//////////////////////////////////////////////////////////////////////////
{
  return bitSet.get ( index );
}
```

```
//////////////////////////////////////////////////////////////////////////
// mutator methods
//////////////////////////////////////////////////////////////////////////

public void  clear ( )
//////////////////////////////////////////////////////////////////////////
{
  bitSet.clear ( );
}

public void  setSpiking ( final BitSet  spikingBitSet )
//////////////////////////////////////////////////////////////////////////
{
  bitSet.or ( spikingBitSet );
}

public void  setSpiking ( final int  index )
//////////////////////////////////////////////////////////////////////////
{
  bitSet.set ( index );
}

public void  setSpiking (
  final int      index,
  final boolean  spiking )
//////////////////////////////////////////////////////////////////////////
{
  bitSet.set ( index, spiking );
}

//////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////
}
```

SieveConfigImp.java

```
package com.croftsoft.apps.sieve.imp;

import java.awt.*;

import com.croftsoft.apps.sieve.SieveConfig;
import com.croftsoft.core.CroftSoftConstants;

/*********************************************************************
* SIEVE configuration.
*
* Can be modified to be persistent.
*
* @version
*   $Id: SieveConfigImp.java 239 2008-05-23 21:01:14Z root $
* @since
*   2008-02-18
* @author
*   <a href="http://www.CroftSoft.com/">David Wallace Croft</a>
*********************************************************************/

public final class  SieveConfigImp
  implements SieveConfig
//////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////
{

private static final String  VERSION
  = "$Date: 2008-05-23 16:01:14 -0500 (Fri, 23 May 2008) $";

private static final String  TITLE
  = "CroftSoft SIEVE";

private static final String  INFO
  = TITLE + "\n"
  + "Version " + VERSION + "\n"
```

```
    + CroftSoftConstants.COPYRIGHT + "\n"
    + CroftSoftConstants.DEFAULT_LICENSE + "\n"
    + CroftSoftConstants.HOME_PAGE + "\n";

  private static final int
    FRAME_WIDTH  = 600,
    FRAME_HEIGHT = 400;

  private static final double  UPDATE_RATE = Double.POSITIVE_INFINITY;

  private static final Color
    BACKGROUND_COLOR = Color.WHITE,
    FOREGROUND_COLOR = Color.BLACK;

  private static final String
    SHUTDOWN_CONFIRMATION_PROMPT = "Exit " + TITLE + "?";

  private static final Cursor  CURSOR
    = new Cursor ( Cursor.CROSSHAIR_CURSOR );

  private static final Font    FONT
    = new Font ( "Arioso", Font.BOLD, 20 );

  ////////////////////////////////////////////////////////////////////////
  ////////////////////////////////////////////////////////////////////////

  public static SieveConfigImp  load (
    @SuppressWarnings ( "unused" )
    final String [ ]  args )
  ////////////////////////////////////////////////////////////////////////
  {
    return new SieveConfigImp ( );
  }

  ////////////////////////////////////////////////////////////////////////
  ////////////////////////////////////////////////////////////////////////
```

```
public Color   getBackgroundColor ( ) { return BACKGROUND_COLOR; }

public Cursor  getCursor ( ) { return CURSOR; }

public Color   getForegroundColor ( ) { return FOREGROUND_COLOR; }

public String  getInfo ( ) { return INFO; }

public Font    getFont ( ) { return FONT; }

public Dimension  getFrameSize ( )
  { return new Dimension ( FRAME_WIDTH, FRAME_HEIGHT ); }

public String  getFrameTitle ( ) { return TITLE; }

public String  getShutdownConfirmationPrompt ( )
  { return SHUTDOWN_CONFIRMATION_PROMPT; }

public String  getThreadName ( ) { return TITLE; }

public double  getUpdateRate ( ) { return UPDATE_RATE; }

////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
}
```

SieveMindContextImp.java

```
package com.croftsoft.apps.sieve.imp;

import com.croftsoft.core.lang.NullArgumentException;

import com.croftsoft.apps.sieve.SieveMindContext;
import com.croftsoft.apps.sieve.SieveSpikeData;

/*********************************************************************
* SieveMindContext implementation.
*
* @version
*   $Id: SieveMindContextImp.java 237 2008-05-23 20:46:18Z root $
* @since
*   2008-04-11
* @author
*   <a href="http://www.CroftSoft.com/">David Wallace Croft</a>
*********************************************************************/

public final class  SieveMindContextImp
  implements SieveMindContext
//////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////
{

private final SieveSpikeData  effectorSpikeData;

private final SieveSpikeData  sensorSpikeData;

//////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////

public  SieveMindContextImp (
  final SieveSpikeData  effectorSpikeData,
  final SieveSpikeData  sensorSpikeData )
//////////////////////////////////////////////////////////////////////
```

```
{
  NullArgumentException.checkArgs (
    this.effectorSpikeData = effectorSpikeData,
    this.sensorSpikeData   = sensorSpikeData );
}

////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////

public String [ ]  getEffectorLabels ( )
////////////////////////////////////////////////////////////////////////
{
  return effectorSpikeData.getLabels ( );
}

public String [ ]  getSensorLabels ( )
////////////////////////////////////////////////////////////////////////
{
  return sensorSpikeData.getLabels ( );
}

public int  getEffectorLength ( String  effectorLabel )
////////////////////////////////////////////////////////////////////////
{
  return effectorSpikeData.getLength ( effectorLabel );
}

public int  getEffectorOffset ( String  effectorLabel )
////////////////////////////////////////////////////////////////////////
{
  return effectorSpikeData.getOffset ( effectorLabel );
}

public int  getSensorLength ( String  sensorLabel )
////////////////////////////////////////////////////////////////////////
{
  return sensorSpikeData.getLength ( sensorLabel );
```

```
}

public int  getSensorOffset ( String  sensorLabel )
//////////////////////////////////////////////////////////////////////////
{
  return sensorSpikeData.getOffset ( sensorLabel );
}

public boolean  isSensorSpiking ( int  index )
//////////////////////////////////////////////////////////////////////////
{
  return sensorSpikeData.isSpiking ( index );
}

public void  setEffectorSpiking ( int  index )
//////////////////////////////////////////////////////////////////////////
{
  effectorSpikeData.setSpiking ( index );
}

//////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////
}
```

SieveModelImp.java

```
package com.croftsoft.apps.sieve.imp;

import java.util.*;

import com.croftsoft.core.lang.NullArgumentException;
import com.croftsoft.core.lang.lifecycle.Updatable;
import com.croftsoft.core.math.MathLib;
import com.croftsoft.core.math.axis.AxisAngle;
import com.croftsoft.core.media.jogl.camera.JoglCamera;
import com.croftsoft.core.media.jogl.camera.JoglCameraImp;
import com.croftsoft.core.media.jogl.camera.JoglCameraMut;
import com.croftsoft.core.media.jogl.render.JoglSpinCube;
import com.croftsoft.core.util.mail.Mail;

import com.croftsoft.apps.sieve.SieveConfig;
import com.croftsoft.apps.sieve.SieveMessage;
import com.croftsoft.apps.sieve.SieveModel;
import com.croftsoft.apps.sieve.SieveSpikeData;
import com.croftsoft.apps.sieve.body.SieveEffectorImp;
import com.croftsoft.apps.sieve.body.SieveSensorImp;
import com.croftsoft.apps.sieve.body.SieveEffectorImp.Effector;

/*********************************************************************
* Model.
*
* Maintains program state.
*
* @version
*   $Id: SieveModelImp.java 235 2008-05-18 22:10:22Z root $
* @since
*   2008-02-18
* @author
*   <a href="http://www.CroftSoft.com/">David Wallace Croft</a>
*********************************************************************/
```

```
public final class  SieveModelImp
  implements SieveModel, Updatable
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
{

private static final double
  Z_INIT            = 7,
  X_MAX             = 20,
  X_MIN             = -20,
  Y_MAX             = 20,
  Y_MIN             = -20,
  Z_MAX             = 20,
  Z_MIN             = 3,
  RESISTANCE_FACTOR = 0.9,
  ROTATE_FACTOR     = 0.2,
  TRANSLATE_FACTOR  = 0.005;

// private final instance variables

private final Mail<SieveMessage>  mail;

private final SieveSensorImp      sieveSensorImp;

private final SieveEffectorImp    sieveEffectorImp;

// model state instance variables

private final Map<EnumInt, Integer>  intMap;

private final JoglCameraMut          joglCameraMut;

private final JoglSpinCube.ModelImp  joglSpinCubeModelImp;

//

private boolean  simulationRunning;
```

```
private double
  linearVelocityX,
  linearVelocityY,
  linearVelocityZ,
  angularVelocityX,
  angularVelocityY,
  angularVelocityZ;

////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////

public  SieveModelImp (
  final SieveConfig         sieveConfig,
  final Mail<SieveMessage>  mail )
////////////////////////////////////////////////////////////////////////
{
  NullArgumentException.checkArgs (
    sieveConfig,
    this.mail = mail );

  sieveEffectorImp = new SieveEffectorImp ( );

  sieveSensorImp = new SieveSensorImp   ( );

  joglCameraMut = new JoglCameraImp ( );

  joglCameraMut.setZ ( Z_INIT );

  intMap = new HashMap<EnumInt, Integer> ( );

  set ( EnumInt.OPTIC_HEIGHT, sieveSensorImp.getOpticHeight ( ) );

  set ( EnumInt.OPTIC_WIDTH , sieveSensorImp.getOpticWidth  ( ) );

  joglSpinCubeModelImp = new JoglSpinCube.ModelImp ( );
```

```
  simulationRunning = true;
}

////////////////////////////////////////////////////////////////////////
// interface SieveModel accessor methods
////////////////////////////////////////////////////////////////////////

public int  get ( final EnumInt  enumInt )
////////////////////////////////////////////////////////////////////////
{
  final Integer  i = intMap.get ( enumInt );

  return i == null ? 0 : i.intValue ( );
}

public AxisAngle  getOrientation ( )
////////////////////////////////////////////////////////////////////////
{
  return joglCameraMut.getAxisAngle ( );
}

public double  getPositionX ( )
////////////////////////////////////////////////////////////////////////
{
  return joglCameraMut.getX ( );
}

public double  getPositionY ( )
////////////////////////////////////////////////////////////////////////
{
  return joglCameraMut.getY ( );
}

public double  getPositionZ ( )
////////////////////////////////////////////////////////////////////////
{
  return joglCameraMut.getZ ( );
```

```
}

public JoglSpinCube.Model  getJoglSpinCubeModel ( )
////////////////////////////////////////////////////////////////////////
{
  return joglSpinCubeModelImp;
}

public boolean  isSimulationRunning ( )
////////////////////////////////////////////////////////////////////////
{
  return simulationRunning;
}

////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////

public SieveSpikeData    getEffectorSpikeData ( )
                           { return sieveEffectorImp; }

public SieveSpikeData    getSensorSpikeData ( )
                           { return sieveSensorImp; }

public SieveEffectorImp  getSieveEffectorImp ( )
                           { return sieveEffectorImp; }

public SieveSensorImp    getSieveSensorImp   ( )
                           { return sieveSensorImp;   }

////////////////////////////////////////////////////////////////////////
// lifecycle methods
////////////////////////////////////////////////////////////////////////

public void  update ( )
////////////////////////////////////////////////////////////////////////
{
  final int  size = mail.size ( );
```

```
  for ( int  i = 0; i < size; i++ )
  {
    final SieveMessage  sieveMessage = mail.get ( i );

    final SieveMessage.Type  type = sieveMessage.getType ( );

    switch ( type )
    {
      case TOGGLE_PAUSE_REQUEST:

        simulationRunning = !simulationRunning;

        mail.offer ( SieveMessage.TOGGLE_PAUSE_EVENT_INSTANCE );

        break;

      default:

        // ignore
    }
  }

  if ( simulationRunning )
  {
    joglSpinCubeModelImp.update ( );

    updateEffectorPhysics ( );

    clipToBounds ( );

    sieveEffectorImp.clear ( );

    sieveSensorImp  .clear ( );
  }
}
```

```
////////////////////////////////////////////////////////////////////////
// private methods
////////////////////////////////////////////////////////////////////////

private void  set (
  final EnumInt  enumInt,
  final int      value )
////////////////////////////////////////////////////////////////////////
{
  intMap.put ( enumInt, new Integer ( value ) );
}

private void  updateEffectorPhysics ( )
////////////////////////////////////////////////////////////////////////
{
  int
    linearForceX = 0,
    linearForceY = 0,
    linearForceZ = 0,
    torqueForceX = 0,
    torqueForceY = 0,
    torqueForceZ = 0;

  for ( final Effector  effector : Effector.values (  ) )
  {
    if ( sieveEffectorImp.isSpiking (
      sieveEffectorImp.getOffset ( effector ) ) )
    {
      switch ( effector )
      {
        case BACKWARD:

          linearForceZ++;

          break;

        case DOWN:
```

```
  linearForceY--;
  break;
case FORWARD:
  linearForceZ--;
  break;
case LEFT:
  linearForceX--;
  break;
case PITCH_DOWN:
  torqueForceX--;
  break;
case PITCH_UP:
  torqueForceX++;
  break;
case RIGHT:
  linearForceX++;
  break;
case ROLL_LEFT:
```

```
        torqueForceZ++;
        break;
      case ROLL_RIGHT:
        torqueForceZ--;
        break;
      case UP:
        linearForceY++;
        break;
      case YAW_LEFT:
        torqueForceY++;
        break;
      case YAW_RIGHT:
        torqueForceY--;
        break;
      default:
        throw new RuntimeException ( );
    }
  }
}

// force = mass * acceleration
// acceleration = force / mass
```

```
// velocity1 = velocity0 + acceleration * delta_time
// For unit mass and unit delta_time, velocity1 = velocity0 + force

linearVelocityX  += TRANSLATE_FACTOR * linearForceX;

linearVelocityY  += TRANSLATE_FACTOR * linearForceY;

linearVelocityZ  += TRANSLATE_FACTOR * linearForceZ;

angularVelocityX += ROTATE_FACTOR * torqueForceX;

angularVelocityY += ROTATE_FACTOR * torqueForceY;

angularVelocityZ += ROTATE_FACTOR * torqueForceZ;

// The resistance factor is between zero and one.
// It models resisted motion in a medium such as air or water.

linearVelocityX  *= RESISTANCE_FACTOR;

linearVelocityY  *= RESISTANCE_FACTOR;

linearVelocityZ  *= RESISTANCE_FACTOR;

angularVelocityX *= RESISTANCE_FACTOR;

angularVelocityY *= RESISTANCE_FACTOR;

angularVelocityZ *= RESISTANCE_FACTOR;

translate ( JoglCamera.Axis.X, linearVelocityX );

translate ( JoglCamera.Axis.Y, linearVelocityY );

translate ( JoglCamera.Axis.Z, linearVelocityZ );

rotate ( JoglCamera.Axis.X, angularVelocityX );
```

```
  rotate ( JoglCamera.Axis.Y, angularVelocityY );

  rotate ( JoglCamera.Axis.Z, angularVelocityZ );
}

private void  rotate (
  final JoglCamera.Axis  axis,
  final double           angularVelocity )
//////////////////////////////////////////////////////////////////////
{
  if ( angularVelocity == 0 )
  {
    return;
  }

  joglCameraMut.rotate ( axis, angularVelocity );
}

private void  translate (
  final JoglCamera.Axis  axis,
  final double           linearVelocity )
//////////////////////////////////////////////////////////////////////
{
  if ( linearVelocity == 0 )
  {
    return;
  }

  joglCameraMut.translate ( axis, linearVelocity );
}

private void  clipToBounds ( )
//////////////////////////////////////////////////////////////////////
{
  final double  x = joglCameraMut.getX ( );
```

```
  final double  y = joglCameraMut.getY ( );

  final double  z = joglCameraMut.getZ ( );

  final double  clipX = MathLib.clip ( x, X_MIN, X_MAX );

  final double  clipY = MathLib.clip ( y, Y_MIN, Y_MAX );

  final double  clipZ = MathLib.clip ( z, Z_MIN, Z_MAX );

  if ( clipX != x )
  {
    joglCameraMut.setX ( clipX );

    linearVelocityX = 0;
  }

  if ( clipY != y )
  {
    joglCameraMut.setY ( clipY );

    linearVelocityY = 0;
  }

  if ( clipZ != z )
  {
    joglCameraMut.setZ ( clipZ );

    linearVelocityZ = 0;
  }
}

//////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////
}
```

SieveMindImp.java

```
package com.croftsoft.apps.sieve.mind;

import com.croftsoft.apps.sieve.SieveMind;
import com.croftsoft.apps.sieve.SieveMindContext;
import com.croftsoft.apps.sieve.body.SieveEffector;
import com.croftsoft.apps.sieve.body.SieveSensor;

/*********************************************************************
* Exemplar SieveMind implementation.
*
* @version
*   $Id: SieveMindImp.java 220 2008-04-27 20:46:05Z root $
* @since
*   2008-03-15
* @author
*   <a href="http://www.CroftSoft.com/">David Wallace Croft</a>
*********************************************************************/

public final class  SieveMindImp
  implements SieveMind
//////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////
{

private SieveMindContext  sieveMindContext;

private int
  effectorBackwardOffset,
  effectorForwardOffset,
  opticLength,
  opticOffset;

//////////////////////////////////////////////////////////////////////
// interface SieveMindContext methods
//////////////////////////////////////////////////////////////////////
```

```
public void  setSieveMindContext (
  final SieveMindContext  sieveMindContext )
//////////////////////////////////////////////////////////////////////////
{
  this.sieveMindContext = sieveMindContext;
}

public void  init ( )
//////////////////////////////////////////////////////////////////////////
{
  final String [ ]  effectorLabels
    = sieveMindContext.getEffectorLabels ( );

  final String [ ]  sensorLabels
    = sieveMindContext.getSensorLabels ( );

  for ( final String  effectorLabel : effectorLabels )
  {
    System.out.println ( effectorLabel );
  }

  for ( final String  sensorLabel : sensorLabels )
  {
    System.out.println ( sensorLabel );
  }

  effectorBackwardOffset = sieveMindContext.getEffectorOffset (
    SieveEffector.LABEL_BACKWARD );

  effectorForwardOffset = sieveMindContext.getEffectorOffset (
    SieveEffector.LABEL_FORWARD );

  opticLength
    = sieveMindContext.getSensorLength ( SieveSensor.LABEL_OPTIC );

  opticOffset
```

```
    = sieveMindContext.getSensorOffset ( SieveSensor.LABEL_OPTIC );
}

public void  update ( )
////////////////////////////////////////////////////////////////////////
{
  int  redSpikes  = 0;

  int  blueSpikes = 0;

  for ( int  i = 0; i < opticLength; i += 3 )
  {
    if ( sieveMindContext.isSensorSpiking ( opticOffset + i + 0 ) )
    {
      redSpikes++;
    }

    if ( sieveMindContext.isSensorSpiking ( opticOffset + i + 2 ) )
    {
      blueSpikes++;
    }
  }

  if ( redSpikes >= blueSpikes )
  {
    sieveMindContext.setEffectorSpiking ( effectorForwardOffset );
  }

  if ( redSpikes <= blueSpikes )
  {
    sieveMindContext.setEffectorSpiking ( effectorBackwardOffset );
  }
}

public void  destroy ( )
////////////////////////////////////////////////////////////////////////
{
```

```
  // empty
}

////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
}
```