A JOTSA Example

Steven Robbins

November, 1996 last updated January, 1998

Technical Report CS-96-13

Abstract. This technical report is an introduction to JOTSA.

Division of Computer Science
The University of Texas at San Antonio
San Antonio, TX 78249

1 An introduction to JOTSA

JOTSA is a Java package for animation that is based on the path-transition paradigm [1]. JOTSA uses a `JotsaAnimationApplet` class that extends the Java `Applet` class. To write a simple JOTSA animation, you create an instance of a class that extends the class `JotsaAnimationApplet`. The resulting applet can be run using an applet viewer or a browser.

You must specify what will remain fixed on the display and what will move. The fixed part is put into a Java `Image` called `JotsaBackGround` and each moving part is specified by an instance of the JOTSA class `JotsaAnimationObject`. In the simplest case a `JotsaAnimationObject` consists of an image to display, a path along which the image is to move, a speed of movement, a level number and a key. The level number determines the order in which the moving objects are displayed and therefore which objects hide others from view. The key is used to remove the object.

JOTSA manages the collection of `JotsaAnimationObjects` in a list sorted by increasing level number. To display moving objects, you need to perform the following steps for each object.

1. Create an instance of the class `JotsaAnimationObject`, setting the initial position, a level number and a key.

2. Specify the image to be displayed.

3. Specify the path along which it is to move.

4. Specify the speed at which it is to move.

5. Insert the object in the collection.

6. Activate the object.

The rest of this report develops two examples that cover the minimal information needed to write an animation using JOTSA. More details can be found in another technical report.

1.1 A Simple Example

Example 1 shows a simple use of JOTSA. The applet displays a small window and a single button marked `Do It`. When the button is pushed, a small red disk appears in the upper left portion of the window and the button label is changed to `Move It`. When the button is pushed a second time, the object moves diagonally down the window growing larger as it moves. The button label is reset to `Do It`.

```
/*
   <Applet code = "jtest1.simple"
           width = 200 height = 200>
   </applet>
 */
package jtest1;

import java.awt.*;
import java.awt.image.*;
import java.applet.*;
import jotsa.*;

public class simple extends JotsaAnimationApplet {
   Button doit;
   JotsaAnimationObject obj;
   Color C = Color.red;
   int start_x = 20;
   int start_y = 10;
   int end_x = 135;
   int end_y = 135;
   int start_size = 20;
   int end_size = 30;
   int move_time = 5000;

   public void init() {
      super.init();
      setLayout(new BorderLayout());
      add("South",doit = new Button("Do It"));
      add("Center",JotsaDefaultCanvas);
      validate();
      JotsaInitImages();
      JotsaWriteBackgroundString("A simple JOTSA example",20,80,Color.blue);
      JotsaWriteBackgroundString("using JOTSA "+JotsaVersion,20,100,Color.blue);
      obj = new JotsaAnimationObject(start_x,start_y,this);
      obj.SetFillOval(start_size,start_size,C);
      obj.SetPositionCentered();
      obj.PathCreateAlongLine(start_x,start_y,end_x,end_y);
      obj.SetSizeLinear(start_size,end_size);
   }

   public boolean action(Event e,Object arg) {
      if ("Do It".equals(arg)) {
         System.out.println("Do It Pushed");
         doit.setLabel("Move It");
         JotsaRemoveAllObjects();
         obj.Deactivate();
         JotsaInsertObject(obj);
         JotsaForceRedisplay();
         return true;
      }
      if ("Move It".equals(arg)) {
         System.out.println("Move It Pushed");
         doit.setLabel("Do It");
         obj.TimesSet(move_time);
         obj.ActivateDelay();
         JotsaForceRedisplay();
         return true;
      }
      return super.action(e,arg);
   }
}
```

Example 1: A simple JOTSA animation.

1.2 The Applet

The user-written JOTSA applet extends the JOTSA class `JotsaAnimationApplet`. Its `init` method should call `super.init()` to initialize the various data structures that are needed. In the simplest cases all of the drawing is done in an instance of the `JotsaCanvas` class called `JotsaDefaultCanvas`. The canvas should be inserted into the layout of the applet. A call to `validate()` is necessary to make sure that the canvas has had its size set. Then a call to

```
   JotsaInitImages();
```

sets up the images JOTSA uses for buffering.

1.3 Creating a Background Image

The background image contains the part of the display that does not change. You can draw a string to it using

```
   public JotsaAnimationObject JotsaWriteBackgroundString(String str,
             int x, int y, Color C);
```

You specify the string, the position of the start of the string and its color. In the example, a blue string is drawn. You can also draw a line in the background using:

```
   public JotsaAnimationObject JotsaWriteBackgroundLine(int x1,
             int y1, int x2, int y2, Color C);
```

1.4 Creating a `JotsaAnimationObject`

A `JotsaAnimationObject` is created with a constructor that is given the initial x and y coordinates, a level number, a key and the applet. For example:

```
    obj = new JotsaAnimationObject(start_x,start_y,level,key,this);
```

As in the example, the level and key can be omitted and they will be assigned automatically. The coordinates `(start_x,start_y)` gives the position of the object before it starts moving. Objects are drawn on the screen in order of increasing level number, which is an integer. Objects with higher level number are drawn on top of those with lower level number. All objects are drawn on top of the background. The order for objects with the same level number is arbitrary. You can change the level number of a `JotsaAnimationObject` using the `set_level` method of the object, for example:

```
    obj.set_level(current_level++);
```

However, the level can only be set before the object is inserted in the list of objects to be displayed. The `key` is used to remove the object from the list.

If you do not want to bother with level numbers or keys, Jotsa can take care of them for you. Just omit the level number and key from every constructor if `JotsaAnimationObject` and the next available level number and key will be assigned. This is done in Example 1.

```
obj = new JotsaAnimationObject(start_x,start_y,this);
```

1.5 Setting the Image to be Displayed

JOTSA supports several methods of associating an image with a `JotsaAnimationObject` and only the simplest of these are described here. You can provide your own custom image to a `JotsaAnimationObject` by creating an object of type `Image` and passing it to the `JotsaAnimationObject` for display. Use:

```
public void SetImage(Image im);
```

For example:

```
obj.SetImage(myimage);
```

Alternatively, the `JotsaAnimationObject` can draw an image chosen from a set of supported object types. Some of these are set with the following methods from the class `JotsaAnimationObject`.

Simple Shapes:

```
public void SetDrawOval(int width, int height, Color C);
public void SetFillOval(int width, int height, Color C);
public void SetDrawRect(int width, int height, Color C);
public void SetFillRect(int width, int height, Color C);
public void SetDrawArc(int width, int height, int startang,
                         int arcang, Color C);
public void SetDrawSector(int width, int height, int startang,
                         int arcang, Color C);
public void SetFillSector(int width, int height, int startang,
                         int arcang, Color C);
public void SetDrawRegularPoly(double r, int n, Color C);
public void SetFillRegularPoly(double r, int n, Color C);
public void SetDrawPoly(int xpos[], int ypos[], Color C);
public void SetFillPoly(int xpos[], int ypos[], Color C);
public void SetDrawString(String str, Color C);
public void SetDrawStrings(String str, Color C);
```

The `width` and `height` have the same meaning as in the corresponding Java methods. You can set the color to use for the image. Example 1 creates a disk that is 20 pixels wide and 20 pixel high. For arcs and sectors the `startang` is the starting angle in degrees with 0 degrees at the 3 o'clock position. The `arcang` is the number of degrees in the arc and angles increase in the counterclockwise direction.

For some objects the position can be specified in several ways. For rectangles and ovals the default position refers to the upper right corner of the (bounding rectangle of the) object. For arcs, sectors, and polygons it refers to the center of the object. For strings it refers to the lower left corner of the object. For ovals, rectangles and strings you can change the meaning of the position to refer to either the center of the object or the right side of the object using one of the following methods from `JotsaAnimationObject`:

```
public void SetPositionCentered();
public void SetPositionRightJustified();
```

For example, to create a red circle of diameter 30 pixels with center at x = 50 and y = 70:

```
obj = new JotsaAnimationObject(50,70,this);
obj.SetDrawOval(30,30,Color.red);
obj.SetPositionCentered();
```

For right justified ovals and rectangles the position refers to the upper right corner of the (bounding rectangle of the) object. For strings it refers to the lower right corner of the string.

Strings:

You can specify the properties of a string or you can use the default ones. The following methods can be used to change the string properties:

```
public void ResetFont(int fontsize);
public void ResetFont(String fname, int fstyle, int fsize);
public void ResetString(String str);
public void ResetColorString(Color C);
```

The first of these changes only the size of the font while the second changes the font name and style. For example, to create a blue string, "This is my string" which is centered at x=100 and y=200 in an italic Times Roman font of size 30, you could use the following:

```
obj = new JotsaAnimationObject(100,200,this);
obj.SetDrawString("This is my string");
obj.SetPositionCentered();
obj.ResetFont("Times",Font.ITALIC,30);
```

Any object other than a string or line can have a string displayed at its center by using `ResetString` to associate a string with the object. For example, the following specifies an oval with width 50 and height 30 with the upper left corner at x=100 and y=200. The color of the oval is blue and the green string "Hi" in a Times Roman font of size 10 is in the center of the oval.

```
obj = new JotsaAnimationObject(100,200,this);
obj.SetDrawOval(50,30,Color.blue);
obj.ResetString("Hi");
obj.ResetColorString(Color.green);
obj.ResetFont("Times",Font.PLAIN,10);
```

Lines: Lines are specified differently to allow each end of the line to be moved independently. A line is specified with:

```
public void SetDrawLine(JotsaAnimationObject l1, int l1_x, int l1_y,
                        JotsaAnimationObject l2, int l2_x, int l2_y,
                        Color C);
```

This draws a line between the two given `JotsaAnimationObjects`. The integer parameters specify the position of the endpoints of the line relative to the position of the objects. `l1_x` is the x offset of the first endpoint of the line from the position of the object `l1`. If the objects `l1` and `l2` have their images set to `null` with `set_image(null)`, or they are not inserted in the collection of objects to be drawn, they will not be drawn. The purpose of these two objects is to control the endpoints of the line. Each end of the line can have an optional arrow either drawn with two lines or with a solid triangle. The methods to set these are:

```
public void SetFrontArrow(int width, int height, Color C);
public void SetFrontArrowSolid(int width, int height, Color C);
public void SetRearArrow(int width, int height, Color C);
public void SetRearArrowSolid(int width, int height, Color C);
```

The front of the line refers to the end of line linked to the first object and the rear is the end linked to the second object.
Multi-Line Strings: You can specify a multi-line string by including line terminators in the string.For example, a 3-line string can be specified with:

```
obj.SetDrawString("This is line one\nThis is line two\nLine three");
```

By default, the position of a multi-line string refers to the lower left corner of the first line of the string. If the multi-line string is centered, the position refers to the center of the string. If it is right justified, the x-position is the right end of the longest string and the y-position is the bottom of the first line.

1.6 Setting the Path of the Image

The path of the image can be set in one of several ways. In each case a method from the `JotsaAnimationObject` class is used.

a) Specifying the points on the path
You can calculate the coordinates of the path and pass two arrays to the image object giving the path. The length of the `x` array determines the length of the path.

```
public void PathSet(int x[], int y[]);
```

b) Smooth motion along a straight line:
You can have the animation object calculate the path along a line with given endpoints using

```
public void PathCreateAlongLine(start_x,start_y,end_x,end_y);
```

In the example, the path is set to be a straight line between the points (20,10) and (120,120).

c) Controlled motion along a polygon:
You can also specify a polygon along which the object is to move. You can do this with

```
public void PathPolySet(int x1[], int y1[]);
```

The two arrays represent the x and y coordinates of the vertices of the polygon.

d) Motion linked to another object:
Lastly, you can link the object to another animation object so that the two objects move together. You specify the object to link to and the offset from that object's coordinates.

```
public void PathSetLinked(animation_object ani, int x, int y);
```

1.7 Changing the Object Along the Path

After the path has been set, the properties of the object can be set to change along the path. Some of the methods available to do this are:

```
public void SetSizeLinear(int dimenx_start, int dimenx_end,
                            int dimeny_start, int dimeny_end);
```

which sets the x-dimension to vary linearly from `dimenx_start` to `dimenx_end` and similarly sets the y-dimension to vary.

```
public void SetScaleLinear(int scx_start, int scx_end,
                            int scx_start, int scx_end);
```

which is similar to the above but also affects objects linked to this one.

```
public void SetAnglesLinear(int a_start, int a_end);
```

which causes polygons to rotate along their path from an angle of `a_start` degrees to `a_end` degrees.

```
public void SetStartAnglesLinear(int a_start, int a_end);
```

which causes the start angles of arcs to vary.

```
public void SetArcAnglesLinear(int a_start, int a_end);
```

which causes the angle of arcs to vary.

```
public void SetColorLinear(Color color_start, Color color_end);
```

which causes the color of the object to vary. The RGB values are each varied linearly along the path. This is mainly useful when two of the component colors do not change or all components are equal, giving shades of gray.

```
public void SetImages(Image img[]);
```

which when used with `set_image` causes different images to be displayed along the path.

1.8 Setting the Speed

The speed can be set by giving the time in milliseconds to traverse from one end of a path to the other. This is set with the following method.

```
TimesSet(int tm);
```

Example 1 uses this method. Alternatively, you can give the time it takes to move between consecutive points on the path.

```
TimesFrameSet(double tm);
```

Again the time is given in milliseconds but a floating point value is used because this value is likely to be small. If the object is linked to another, the time need not be set.

For polygonal paths you can set the time for each segment of the polynomial when the path is created:

```
public void PathPolySet(int x1[], int y1[], long t[]);
```

where the `t` array gives the time in milliseconds to traverse each segment of the polygon. You can also have the object stop at each vertex before continuing along the next segment with:

```
public void PathPolySet(int x1[], int y1[], long t[], long td[]);
```

where the additional array gives the time to pause at each vertex.

The time can also be changed using the `JotsaSetRate` method as described in Section 1.12.

1.9 Putting the Object in the Collection

The object can be put in the collection of objects to be displayed with:

```
public void JotsaInsertObject(animation_object obj);
```

This will cause the image to be displayed at its initial position. The object can be removed with:

```
public void JotsaRemoveObject(int key);
```

where `key` is the key value used when the object was created. You can remove all objects with:

```
public void JotsaRemoveAllObjects();
```

In Example 1, a press of `Do It` removes all JOTSA objects and then inserts one object after its times have been set.

1.10 Activating the Object

The object will start moving when it is activated:

```
public void Activate();
```

Alternatively, you can use

```
public void ActivateDelay();
```

which causes the object to start moving only after the screen is updated. In Example 1, the object is activated when the button is pushed for a second time. It is also desirable to execute the following method from the `JotsaAnimationApplet` class:

```
public void JotsaForceRedisplay();
```

This is discussed in Section 1.12.

1.11 The `paint` Method

JOTSA animations use double buffering. The basic paint method looks as follows:

```
public void paint(Graphics g) {
   JotsaPaintStart();
   JotsaPaintLocal(last_display_time);
   JotsaDisplayCanvases();
   JotsaPaintEnd();
}
```

The `JotsaPaintStart` and `JotsaPaintEnd` methods just record the time so that profiling can be done. `JotsaPaintLocal` can be overridden so that some preprocessing can be done before the painting takes place. A use of this is shown in Example 2. `JotsaDisplayCanvases` calls `repaint` for each canvas.

Each canvas, including the default one, has its own `paint` method which looks like this:

```
public void paint(Graphics g) {
   if (ap.JotsaPaintCanvas(g,this)) return;
   PaintStart();
   if (scaled_win == null) PaintNorm(g);
   else PaintScaled(g);
   PaintEnd();
}
```

`JotsaPaintCanvas` by default returns false, but it can be overridden in the main applet to allow the user to rewrite the paint method for any canvas. `PaintStart` and `PaintEnd` just record timing information. Most canvases are not scaled and so call `PaintNorm` to do the painting.

Except for some synchronization, `PaintNorm` looks like:

```
public void PaintNorm(Graphics g) {
   long timenow;
   timenow = ap.JotsaLastDisplayTime();
   UpdateNorm(timenow);
   DrawImage(g);
}
```

The `timenow` is just the virtual time at which the last `paint` from the main applet was called. `UpdateNorm` updates a temporary image based on the given virtual time and `DrawImage` copies this image to the screen.

1.12 How the redisplay works

The start method of `JotsaAnimationApplet` starts a `JotsaMasterAnimationThread` that loops alternately sleeping and calling the `JotsaAnimationApplet`'s `paint` method. Its default sleep time is 1 second, but this can be changed with

```
public void JotsaSetDefaultSleepTime(int millisec);
```

Each `JotsaAnimationObject` has a default update time. The minimum of these is used by the `JotsaMasterAnimationThread` for its next sleep value. It ignores those with update time of -1. The default is used when there are no objects to be displayed, they all have value -1, or the minimum is greater than the default. If the default sleep time is used, the `JotsaMasterAnimationThread` calls `repaint_full(1)`. Otherwise it calls `repaint_clip(1)`. By overriding these methods you can use clipping to update the display when objects are moving.

A `JotsaAnimationObject` sets its update time to half of the time to move one pixel. It resets it to -1 when it is done moving.

Because the `JotsaMasterAnimationThead` has a long default sleep time of 1 second, the object may not seem to start moving when `activate` is called. You can wake up the `JotsaMasterAnimationThread` with the `JotsaForceRedisplay()` method. However, it may take up to 100 ms for this to wake it up. This is the reason for the `activate_delay()` method. This will activate the object the next time the `paint` method is called.

Alternatively you could also decrease the default sleep time or decrease the wake up time of the `JotsaMasterAnimationThread` with

```
public void JotsaSetWakeUpDelay(int millisec);
```

JOTSA uses real time for all movements but it allows this time to be scaled to slow down or speed up the display. You can change the rate at which the animation runs by using

```
public void JotsaSetRate(double r);
```

The default rate is 1.0. To slow down the display by a factor of 2, use: `JotsaSetRate(0.5)`. You can also temporarily stop and restart the time with the methods:

```
public void JotsaStopTime();
public void JotsaRestartTime();
```

1.13 Looping

An object can be set to repeat its path by putting it in looping mode. When it is in this mode, it restarts its path after the path is completed. The following methods in the `JotsaAnimationObject` class set and clear looping mode.

```
public void PathSetLoop();
public void PathClearLoop();
```

1.14 Linking

Objects can be linked together in several ways. We have already seen that the endpoints of a line object must be linked to two other objects.

We also saw that the position of an object can be linked to that of another. The following methods are used to link and unlink objects.

```
public void PathSetLinked(animation_object ani, int x, int y);
public void PathClearLinked();
```

A third type of linking synchronizes the path of two objects so that they start and stop at the same time. This is called index linking and can be set and cleared with the following methods:

```
public void PathSetIndexLinked(animation_object ani);
public void PathClearIndexLinked();
```

1.15 Widgets

JOTSA has a number of widgets that can be useful. The `JotsaSlider` widget is a slider for representing integer values. It is essentially a Java Scrollbar with a string giving the value and three control buttons. The `Half` button halves the maximum value and the `Double` button doubles it. Consecutive activations of the `Round` button will round the slider value round to the nearest 10, 100, 1000, or 10000. The `JotsaSliderf` widget is similar but is for use with double values.

1.16 A More Complicated Animation

This section describes the animation `jtest2.example` which appears at the end of this section as Example 2. Clicking and holding down a mouse button causes an object of some type and with random color to appear. Drag the mouse. When the button is released the object will start to move to the new location.

Twenty-four buttons and four choice widgets appear at the bottom of the screen. `Start` and `Stop` start and stop time. `Select Next` selects one of the objects on the screen. A selected object is shown by having it bounce around. `Select None` causes no objects to be selected. `Select All` causes the selected object and all things linked to it to bounce. The `Quit` button will exit the applet if it was started from the appletviewer.

The `Link` button causes the selected object to be linked to the next object created. The `Link Index` button causes the selected object to be index linked to the next object created. The `Link Level` button creates a string object to be linked to the selected object when the mouse is clicked. The string represents the level number of the selected item. The `Line` button attaches a line between the last two objects created when the mouse is clicked. The `Loop` button causes the selected object to be in loop mode and the `UnLoop` button will turn off looping for the selected object.

The `Popup` button produces a new popup window and the `Scaled` button creates a new scaled window. The `Color` button changes the color of the selected item. The `Remove` button removes the selected object. The `Palette` button displays the color palette used. The `Reset Time` sets the virtual time back to zero. All objects are redisplayed starting at their time of creation.

The `Resize` button will resize the canvas if the appletviewer window was resized. The two `Test` buttons currently do not do anything. The `Time -` button decrements the time by 100 milliseconds and is best used when the time has been stopped. The `Time +` button increments the time by 100 milliseconds. The `Controls` button pops up a window containing sliders for controling various parameters. This is explained below.

The `Show Popups` choice allows you to choose one of the created popup windows to be the active one. It also pops up the chosen window if it was hidden. The `Show Scaled` choice allows you to pop up any of the scaled windows. The next choice allows you to select one of 13 types of images to draw when the mouse is pushed. The default is to draw an oval. The last choice, `Set Mode`, allows you to modify how the images are displayed.

The types of objects illustrated by this example include:

Random Image	Select one of the following at random.
Draw Oval	The outline of an oval in a random color.
Fill Oval	A solid oval in a random color.
Draw Rectangle	The outline of a rectangle in a random color.
Fill Rectangle	A solid rectangle in a random color.
Draw Polygon	The outline of a regular polygon in a random color.
Fill Polygon	A solid regular polygon in a random color.
Draw Arc	The arc of an oval in a random color.
Draw Sector	The outline of a sector of an oval in a random color.
Fill Sector	A solid sector of a circle in a random color.
Draw String	A text string containing the characters: *A String*.
Draw Strings	A multi-line string.

GIF Image	A small image read in from a GIF file.
Gif Images	A sequence of images read from GIF files illustrating cartoon animation.

Properties of the above objects can be set with the `Set Mode` choice. Possibilities include:

Position Norm	Use the normal positioning of the object. For rectangles and ovals the position is the upper left corner. For arcs, sectors, and polygons it is the center of the object. For strings it is the lower left of the first line of the string.
Position Centered	For rectangles, ovals, and strings use the center of the object as its position. Use the normal positioning of the object.
Position Right Justified	For rectangles, ovals, and strings use the right edge as its position.
Random Border	Draw a 1-pixel wide border in a random color around filled rectangles, ovals, polygons and sectors.
Black Border	Draw a 1-pixel wide black border around filled rectangles, ovals, polygons and sectors.
No Border	Do not draw a border.
Fixed Size	The object remains fixed in size as it moves.
Vary Size	The object varies in size linearly as it moves. This does not affect the GIF images.
Fixed Color	The color of the object does not change as it moves.
Vary Color	The color of the object changes somewhat smoothly between green and red as it moves.
Front Line Norm	No arrow at the start of the line. The front is at the first object to which the line is linked.
Front Line Arrow	An arrow is drawn in a random color at the front of the line.
Front Line Solid	An solid arrow is drawn in a random color at the front of the line.
Rear Line Norm	No arrow at the tail of the line. The rear is at the second object to which the line is linked.
Rear Line Arrow	An arrow is drawn in a random color at the rear of the line.
Rear Line Solid	An solid arrow is drawn in a random color at the rear of the line.

Fixed Start Angle	Arcs and wedges are drawn with a fixed start angle.
Vary Start Angle	Arcs and wedges are drawn with a start angle varying linearly from 0 to 360 degrees.
Fixed Arc Angle	Arcs and wedges are drawn with a fixed angle.
Vary Arc Angle	Arcs and wedges are drawn with an angle varying linearly.
Do Not Rotate	Polygons, Arcs, and Sectors do not rotate as they move.
Rotate	Polygons, Arcs, and Sectors rotate as they move.
Moving Images	This refers to the **GIF Images** type. It is the default in which the images move to the point at which the mouse is released.
Fixed Images	This refers to the **GIF Images** type. They do not move to the position at which the mouse was released.
Use String	Draw a string of a random color in the center of everything except strings.
Use Strings	Draw a multi-line string of a random color in the center of everything except strings.
No String	Do not draw a string inside objects.

For each of these the active state is shown with a leading asterisk.

Pushing the `Controls` button brings up a popup window with 10 integer sliders and one floating point slider. Each slider controls a parameter of the animation. Each slider consists of a line containing a description of the parameter and its current value. The `Double` and `Half` buttons modify the maximum value of the slider. Consecutive activations of the `Round` button will round the integer slider value round to the nearest 10, 100, 1000, or 10000 or floating point slider values to the nearest hundreth, tenth, or integer.

The integer sliders do not take affect until the `Apply` button is pushed. Until then, the value show appears with an asterisk. The `Reset` button will return the slider to the last applied value.

The parameters that can be changed by the `Controls` window are:

width	The width of most displayed items.
end width	The ending width of most displayed items if they are set to vary their size as they move.
height	The height of most displayed items.
end height	The ending height of most displayed items if they are set to vary their size as they move.
start angle	The starting angle of arcs and sectors.
end angle	The ending start angle of arcs and sectors if they vary their

	start angle as they move.
arc angle	The angle of arcs and sectors.
end arc angle	The ending angle of arcs and sectors if they vary their angle as they move.
sides	The number of sides of regular polygons.
Move Time	The number of milliseconds it takes to move an object.
Time Rate	The rate at which virtual time moves compared to real time.

The `init` method determines the bounds and sets up the layout. It then calls the method `JotsaInitImages()`. It also sets up the background strings and initializes certain variables for use with the buttons. Lastly, it gets some GIF files and creates images from them.

The method `get_random_color` selects a color from a standard color map of 125 colors. It uses `find_color_index` to determine what the current color is. `set_object_image` picks one of 13 types of images to display.

The method `JotsaChangeParameters` is overridden. It is called when one of the Jotsa sliders changes. In this example it is used to update the rate at which time runs.

The `JotsaPaintLocal` method is overridden so that the string of the `timeinfo` object contains the virtual time in seconds at the time that `paint` is executed. This value is displayed at the top of the window.

The `make_colored_object` and `make_colored_headings` methods are used to display the color palette.

The `handleEvent` method is overwritten to handle mouse events. The `MOUSE_DOWN` event sets the variables `start_x` and `start_y` to the position of the pointer and then shows the position on the status line.

If the `link_level_flag` is set (the `Link Level` button was pushed) and an object was selected, a new string animation object is created and linked to the selected object so that it is displayed at that object's coordinates.

If the last object created was not a line (`last_line_flag` is false), the previous two objects created which were not lines are saved for use with the `Line` button. It then creates an `JotsaAnimationObject` called `obj` with that start position. Both the level number and the key are `current level` and this variable is then incremented. If the `Line` button was pushed it creates a line between the last two saved objects. Otherwise it calls `set_object_image()` to choose a random image for `obj` to use and inserts the object in the collection of objects to be displayed.

If the `Link` button had been pushed and an object is selected, that selected object is linked to the newly created one with offsets of 10 and 15.

If the `Link Index` button has been pushed (`link_flag` is true) and an object has been selected (`selected`$\geq$ 0), it links the selected object to the newly created one.

The `MOUSE_UP` event sets the variables `end_x` and `end_y` to the position of the pointer and gets the move time, `move_time`, from the speed control slider. If the `Line` button was pushed, it sets the object to be a line linked between the previous two objects.

If the `Link Level` button was pushed (`link_level_flag` is true), or if the `Line` button was pushed (`line_flag` is true) there is nothing more to do.

Otherwise it fixes up the end point in case the mouse was dragged out of the window and shows the positions for the move on the status line. It creates a path for the object with `path_create_along_line()` using the positions at which the mouse button was pushed and released for the endpoints and sets the move time for the object. If the object was not an image, the size is varied between 20 and 40 as it moves. The object is then activated when the display is next updated.

The method `action` is overridden to handle the button events. First the choices are handled.

The `Start` and `Stop` buttons use `JotsaRestartTime` and `JotsaStopTime`.

The `selected` variable keeps track of which object is selected. Each object has an index in the collection of displayed objects and this variable holds that index. If it is less than 0, no object is selected. The method `JotsaGetObjectAt` is used to get one of the displayed objects from its index. A selected object will shake when displayed. The method `SetShake(r)` will make an object jiggle by `r` pixels when it is displayed. `SetShake(0)` will turn off the jiggling.

The `Select Next` button first checks to see that there is at least one object. If so, it checks to see if an object was already selected so that it can be deselected. If so, that object's shake value is set to 0 with `SetShake`. The number of displayed objects can be gotten with `JotsaNumObjects` and the `selected` variable is incremented modulo this value. The selected object is then obtained and its shake value is set to 10. The method `clear_shake_all` prevents objects linked to this one from shaking also. The `Select All` button causes objects linked to the selected object to shake also. The `Select None` button causes no object to be selected. The `Time -` button uses `JotsaAddToVirtualTime` to subtract 100 milliseconds from the virtual time.

The `Link` button just sets the `link_flag` so that if an object is selected, it will be linked to the next object created when a mouse button is pushed.

The `Link Index` button just sets the `link_index_flag` so that when the next object is created, the selected object will be index linked to it. This causes the selected object to move along its path in time with the new object so that they both start and stop at the same time.

The `Link Level` button just sets the `link_level_flag` so that if there as a selected object, a new string object will be created showing the level of the selected object when the mouse button is pushed.

The `Line` button sets the `line_flag` if there are at least two objects and the last object created was not a line. If these conditions are satisfied, the next mouse button causes a line to be drawn between the last two objects created. The `Loop` button causes the selected object to loop in its path so that when it gets to the end of its path it will start over again. The `Unloop` button causes the selected object to stop looping.

The `Popup` button creates a new popup window. Each popup window has its own list of objects to be displayed. You can have objects displayed in a popup window by passing an extra parameter to `JotsaInsertObject`. This is what the `pwin` variable is used for. It represents the active popup window. When the main window is active, `pwin` is `null` and in this case `JotsaInsertObject` uses the main window. The main window is active by default and can be made active by selecting the choice `Show Popups`. Choosing another popup window from this choice makes that popup the active window. Creating a new popup window with `Popup` also makes that the active window. If a popup window is active, a green line of text is displayed in this window. The example manages this by having each popup window have a background line indicating that the screen in active. If the window is not active, this line is hidden by using `SetInhibitDisplay`.

The `Scaled` button causes a new window to be created which shows the same thing as the original, but which can be panned and zoomed. You can create any number of scaled windows but each one slows down the display. The `Show Scaled` button will cause all hidden scaled windows to be displayed.

The `Color` button will change the color of the selected object if it is not an image. The colors will loop through the 125 colors used in this example. The `Palette` button will display the color palette used for creating the objects. 125 colors are used which are obtained by using RGB values selected from 0, 102, 153, 204, and 255. Each rectangle displayed is a separate JOTSA object.

The `Remove` button removes the selected object from those currently displayed. The object is not actually removed, but its `RemoveTime` is set so that it will not be displayed if the current virtual time is greater than the time it was removed. The `Reset Time` button sets the virtual time to 0, and all created objects disappear and then reappear at their creation time.

References

[1] J. T. Stasko, “The Path-Transition Paradigm: A practical methodology for adding animation to program interfaces,” Journal of Visual Languages and Computing, 1, pp. 213–236, 1990

The Control Panel Object

```
package jtest2;

import java.awt.*;
import jotsa.*;

   public class ControlPanel extends Frame {
      JotsaAnimationApplet ap;
      public JotsaSlider start_angle_control;
      public JotsaSlider arc_angle_control;
      public JotsaSlider end_angle_control;
      public JotsaSlider end_arc_angle_control;
      public JotsaSliderf time_control;
      public JotsaSlider start_width_control;
      public JotsaSlider end_width_control;
      public JotsaSlider start_height_control;
      public JotsaSlider end_height_control;
      public JotsaSlider num_sides_control;
      public JotsaSlider speed_control;
      public final int KEY_START_ANGLE = 1;
      public final int KEY_ARC_ANGLE = 2;
      public final int KEY_END_ANGLE = 3;
      public final int KEY_END_ARC_ANGLE = 4;
      public final int KEY_START_WIDTH = 5;
      public final int KEY_END_WIDTH = 6;
      public final int KEY_START_HEIGHT = 7;
      public final int KEY_END_HEIGHT = 8;
      public final int KEY_NUM_SIDES = 9;
      public final int KEY_MOVE_TIME = 10;
      private final int format_type = 1;

      public ControlPanel(String str, int init_startwidth, int init_startheight,
                          int init_endwidth, int init_endheight,
                          int init_startangle, int init_arcangle,
                          int init_endangle, int init_endarcangle,
                          int init_numsides, int init_move_time,
                          int width,JotsaAnimationApplet ap) {
         super(str);
         setLayout(new GridLayout(12,1));
         resize(width,300);
         time_control = new JotsaSliderf(1000,0,2000,1000,200,"Time Rate ",
            format_type,ap);
         speed_control = new JotsaSlider(init_move_time,0,2*init_move_time,1,
               width,"Move Time ",true,format_type,KEY_MOVE_TIME,ap);
         start_angle_control = new JotsaSlider(init_startangle,0,360,1,width,
              "start angle: ",true,format_type,KEY_START_ANGLE,ap);
         end_angle_control = new JotsaSlider(init_endangle,0,360,1,width,
              "end angle: ",true,format_type,KEY_END_ANGLE,ap);
         arc_angle_control = new JotsaSlider(init_arcangle,0,360,1,width,
              "arc angle: ",true,format_type,KEY_ARC_ANGLE,ap);
         end_arc_angle_control = new JotsaSlider(init_endarcangle,0,360,1,width,
              "end arc angle: ",true,format_type,KEY_END_ARC_ANGLE,ap);
         start_width_control = new JotsaSlider(init_startwidth,1,200,1,width,
              "width: ",true,format_type,KEY_START_WIDTH,ap);
         end_width_control = new JotsaSlider(init_endwidth,1,200,1,width,
              "end width: ",true,format_type,KEY_END_WIDTH,ap);
         start_height_control = new JotsaSlider(init_startheight,1,200,1,width,
              "height: ",true,format_type,KEY_START_HEIGHT,ap);
         end_height_control = new JotsaSlider(init_endheight,1,200,1,width,
              "end height: ",true,format_type,KEY_END_HEIGHT,ap);
         num_sides_control = new JotsaSlider(init_numsides,3,10,1,width,
              "sides: ",true,format_type,KEY_NUM_SIDES,ap);
         this.ap = ap;
         add(start_width_control);
         add(end_width_control);
         add(start_height_control);
         add(end_height_control);
         add(start_angle_control);
         add(end_angle_control);
         add(arc_angle_control);
         add(end_arc_angle_control);
         add(num_sides_control);
         add(speed_control);
         add(time_control);
         add(new Button("Close"));
      }

      public boolean action(Event e, Object arg) {
         if ("Close".equals(arg))
            hide();
         return true;
      }

   }
```

The applet for Example 2

```
/*
   <Applet code = "jtest2.example"
           width = 600 height = 400>
   </applet>
 */

package jtest2;

import java.awt.*;
import java.awt.image.*;
import java.applet.*;
import java.net.*;
import jotsa.*;

public class example extends JotsaAnimationApplet {

   Image timage;
   Graphics tGC;
   int start_x;
   int start_y;
   int end_x;
   int end_y;
   int width;
   int height;
   int[] colorlist={0, 102, 153, 204, 255};
   boolean link_flag;
   boolean line_flag;
   boolean last_line_flag;
   boolean link_index_flag;
   boolean link_level_flag;
   JotsaAnimationObject obj;
   int selected;
   JotsaAnimationObject selected_object;
   JotsaAnimationObject oldobj;
   JotsaAnimationObject oldobj1;
   JotsaAnimationObject lastobj;
   JotsaCanvas can;
   JotsaPopupWindow pwin;
   JotsaScaledWindow swin;
   String backstr1;
   String backstr2;
   int scaled_count = 0;
   int popup_count = 0;
   Image external_image;
   Image[] ext_images;
   JotsaAnimationObject active_obj;
   Choice Imagetype;
   Choice Modetype;
   int image_type = 2;
   int poly_x[];
   int poly_y[];
   int poly_sides = 5;
   int start_angle = 0;
   int arc_angle = 45;
   int end_angle = 180;
   int end_arc_angle = 135;
   int move_time = 5000;
   boolean vary_images_flag;
   boolean arc_flag;
   boolean poly_flag;
   boolean vary_size;
   boolean vary_arc_angle;
   boolean vary_start_angle;
   boolean rotate_flag;
   boolean vary_color;
   boolean fixed_images_flag;
   boolean include_string_flag = false;
   boolean include_strings_flag = false;
   int front_line_type;
   int rear_line_type;
   final int NUM_EXT_IMAGES=11;
   final int REPEAT_EXT_IMAGES=4;
   int[] fixed_path_x;
   int[] fixed_path_y;
   MediaTracker tracker;
   JotsaAnimationObject timeinfo;
   int link_off_x = 10;
   int link_off_y = 15;
   boolean position_centered_flag = true;
   boolean position_right_justified_flag = false;
   int border_color_type = 0;
   int image_width = 50;
   int image_height = 50;
   int image_end_width = 200;
   int image_end_height = 200;
   Panel ModeTypePanel;
   ControlPanel CP;
   URL backURL;

   public void init() {
       super.init();
       width = bounds().width;
       selected = -1;
       height = setup_layout();
       JotsaInitImages();
       Imagetype.select("Draw Oval");
       backstr1=
          "Illustrates some Jotsa features, version 2.25 running JOTSA version "+
          JotsaVersionMajor+"."+JotsaVersionMinor;
       backstr2 = "Hold down a mouse button, drag it, and let go";
       JotsaWriteBackgroundString(backstr1,20,20,new Color(255,0,0));
       JotsaWriteBackgroundString(backstr2,20,40,new Color(0,0,255));
       active_obj =
          JotsaWriteBackgroundString("This screen is active",
             20,60,new Color(0,255,0));
       JotsaDefaultCanvas.SetRelatedInfo(active_obj);
       can = JotsaDefaultCanvas;
       timeinfo = new JotsaAnimationObject(width-120,20,-1,-1,this);
       timeinfo.SetDrawString("",Color.blue);
       JotsaInsertObject(timeinfo);
       link_flag = false;
       line_flag = false;
       last_line_flag = false;
       link_index_flag = false;
       link_level_flag = false;
       front_line_type = 0;
       rear_line_type = 0;
       vary_size = false;
       vary_start_angle = false;
       vary_arc_angle = false;
       rotate_flag = false;
       vary_color = false;
       fixed_images_flag = false;
       include_string_flag = false;
       include_strings_flag = false;
       poly_x = new int[4];
       poly_y = new int[4];
       poly_x[0] = -50;
       poly_y[0] = -25;
       poly_x[1] = poly_x[0]+100;
       poly_y[1] = poly_y[0];
       poly_x[2] = poly_x[1];
       poly_y[2] = poly_y[1] + 50;
       poly_x[3] = poly_x[0];
       poly_y[3] = poly_y[2];
       fixed_path_x = new int[NUM_EXT_IMAGES*REPEAT_EXT_IMAGES];
       fixed_path_y = new int[NUM_EXT_IMAGES*REPEAT_EXT_IMAGES];
       ext_images = new Image[NUM_EXT_IMAGES*REPEAT_EXT_IMAGES];
       external_image = getImage(getDocumentBase(),"testim.gif");
       ext_images[0] = getImage(getDocumentBase(),"T1.gif");
       ext_images[1] = getImage(getDocumentBase(),"T2.gif");
       ext_images[2] = getImage(getDocumentBase(),"T3.gif");
       ext_images[3] = getImage(getDocumentBase(),"T4.gif");
       ext_images[4] = getImage(getDocumentBase(),"T5.gif");
       ext_images[5] = getImage(getDocumentBase(),"T6.gif");
       ext_images[6] = getImage(getDocumentBase(),"T7.gif");
       ext_images[7] = ext_images[5];
       ext_images[8] = ext_images[4];
       ext_images[9] = ext_images[3];
       ext_images[10] = ext_images[2];
       ext_images[11] = ext_images[1];
       for (int i=1;i<REPEAT_EXT_IMAGES;i++)
          for (int j=0;j<NUM_EXT_IMAGES;j++)
             ext_images[i*NUM_EXT_IMAGES+j] = ext_images[j];
       for (int i=0;i<NUM_EXT_IMAGES;i++)
          if (ext_images[i]==null)
             System.out.println("image "+i+" could not be found");
       tracker = new MediaTracker(this);
       tracker.addImage(ext_images[0],0);
       tracker.addImage(ext_images[1],1);
       tracker.addImage(ext_images[2],2);
       tracker.addImage(ext_images[3],3);
       tracker.addImage(ext_images[4],4);
       tracker.addImage(ext_images[5],5);
       tracker.addImage(ext_images[6],6);
       tracker.statusAll(true);      // Attempt to start loading
       show_tracker_status();
       CP = new ControlPanel("Controls",image_width,image_height,
                              image_end_width,image_end_height,
                              start_angle,arc_angle,
                              end_angle,end_arc_angle,poly_sides,move_time,
                              640,this);
       JotsaForceRedisplay();
       try {
          backURL = new URL((getDocumentBase()),"index.html");
       }
       catch (MalformedURLException e) {
         System.out.println("Cannot form URL in this context");
         backURL = null;
      }

   }

   private void show_tracker_status() {
      int stat;
      stat = tracker.statusAll(true);
      if (stat == MediaTracker.LOADING)
```

```
      System.out.println("Media Loading");
   else if (stat == MediaTracker.ABORTED)
      System.out.println("Media Aborted");
   else if (stat == MediaTracker.ERRORED)
      System.out.println("Media Error");
   else if (stat == MediaTracker.COMPLETE)
      System.out.println("Media Complete");
   else
      System.out.println("Media Unknown Status");
}

private int setup_layout() {
    setLayout(new BorderLayout());
    Panel p = new Panel();
    Panel q = new Panel();
    Panel r = new Panel();
    Panel s = new Panel();
    Panel q4 = new Panel();
    Panel t = new Panel();
    ModeTypePanel = new Panel();
    Imagetype = new Choice();
    Modetype = new Choice();
    Imagetype.addItem("Random Image");
    Imagetype.addItem("Draw Oval");
    Imagetype.addItem("Fill Oval");
    Imagetype.addItem("Draw Rectangle");
    Imagetype.addItem("Fill Rectangle");
    Imagetype.addItem("Draw Polygon");
    Imagetype.addItem("Fill Polygon");
    Imagetype.addItem("Draw Arc");
    Imagetype.addItem("Draw Sector");
    Imagetype.addItem("Fill Sector");
    Imagetype.addItem("Pie Chart");
    Imagetype.addItem("Draw String");
    Imagetype.addItem("Draw Strings");
    Imagetype.addItem("GIF Image");
    Imagetype.addItem("GIF Images");
    set_mode_entries();
    p.setLayout(new GridLayout(5,1));
    q.setLayout(new GridLayout(1,6));
    r.setLayout(new GridLayout(1,6));
    s.setLayout(new GridLayout(1,6));
    q4.setLayout(new GridLayout(1,6));
    t.setLayout(new GridLayout(1,4));
    ModeTypePanel.setLayout(new GridLayout(1,1));
    ModeTypePanel.add(Modetype);
    q.add(new Button("Start"));
    q.add(new Button("Stop"));
    q.add(new Button("Select Next"));
    q.add(new Button("Select All"));
    q.add(new Button("Select None"));
    q.add(new Button("Quit"));
    r.add(new Button("Link"));
    r.add(new Button("Link Index"));
    r.add(new Button("Link Level"));
    r.add(new Button("Line"));
    r.add(new Button("Loop"));
    r.add(new Button("UnLoop"));
    s.add(new Button("Popup"));
    s.add(new Button("Scaled"));
    s.add(new Button("Color"));
    s.add(new Button("Palette"));
    s.add(new Button("Remove"));
    s.add(new Button("Reset Time"));
    q4.add(new Button("Resize"));
    q4.add(new Button("Test1"));
    q4.add(new Button("Test2"));
    q4.add(new Button("Time -"));
    q4.add(new Button("Time +"));
    q4.add(new Button("Controls"));
    t.add(JotsaShowPopupWindows);
    t.add(JotsaShowScaledWindows);
    t.add(Imagetype);
    t.add(ModeTypePanel);
    p.add(q);
    p.add(r);
    p.add(s);
    p.add(q4);
    p.add(t);
    add("Center",JotsaDefaultCanvas);
    add("South",p);
    validate();
    return p.location().y;
}

void reset_mode_entries() {
   ModeTypePanel.removeAll();
   Modetype = new Choice();
   set_mode_entries();
   ModeTypePanel.add(Modetype);
}

void set_mode_entries() {
    Modetype.addItem("Set Mode");
    if (position_centered_flag) {
       Modetype.addItem("   Position Norm");
       Modetype.addItem("* Position Centered");
       Modetype.addItem("   Position Right Justified");
    }
    else if (position_right_justified_flag) {
       Modetype.addItem("   Position Norm");
       Modetype.addItem("   Position Centered");
       Modetype.addItem("* Position Right Justified");
    }
    else {
       Modetype.addItem("* Position Norm");
       Modetype.addItem("   Position Centered");
       Modetype.addItem("   Position Right Justified");
    }
    if (border_color_type == 0) {
       Modetype.addItem("   Random Border");
       Modetype.addItem("   Black Border");
       Modetype.addItem("* No Border");
    }
    else if (border_color_type == 1) {
       Modetype.addItem("* Random Border");
       Modetype.addItem("   Black Border");
       Modetype.addItem("   No Border");
    }
    else {
       Modetype.addItem("   Random Border");
       Modetype.addItem("* Black Border");
       Modetype.addItem("   No Border");
    }
    if (vary_size) {
       Modetype.addItem("   Fixed Size");
       Modetype.addItem("* Vary Size");
    }
    else {
       Modetype.addItem("* Fixed Size");
       Modetype.addItem("   Vary Size");
    }
    if (vary_color) {
       Modetype.addItem("   Fixed Color");
       Modetype.addItem("* Vary Color");
    }
    else {
       Modetype.addItem("* Fixed Color");
       Modetype.addItem("   Vary Color");
    }
    if (front_line_type == 0) {
       Modetype.addItem("* Front Line Norm");
       Modetype.addItem("   Front Line Arrow");
       Modetype.addItem("   Front Line Solid");
    }
    else if (front_line_type == 1) {
       Modetype.addItem("   Front Line Norm");
       Modetype.addItem("* Front Line Arrow");
       Modetype.addItem("   Front Line Solid");
    }
    else {
       Modetype.addItem("   Front Line Norm");
       Modetype.addItem("   Front Line Arrow");
       Modetype.addItem("* Front Line Solid");
    }
    if (rear_line_type == 0) {
       Modetype.addItem("* Rear Line Norm");
       Modetype.addItem("   Rear Line Arrow");
       Modetype.addItem("   Rear Line Solid");
    }
    else if (rear_line_type == 1) {
       Modetype.addItem("   Rear Line Norm");
       Modetype.addItem("* Rear Line Arrow");
       Modetype.addItem("   Rear Line Solid");
    }
    else {
       Modetype.addItem("   Rear Line Norm");
       Modetype.addItem("   Rear Line Arrow");
       Modetype.addItem("* Rear Line Solid");
    }
    if (vary_start_angle) {
       Modetype.addItem("   Fixed Start Angle");
       Modetype.addItem("* Vary Start Angle");
    }
    else {
       Modetype.addItem("* Fixed Start Angle");
       Modetype.addItem("   Vary Start Angle");
    }
    if (vary_arc_angle) {
       Modetype.addItem("   Fixed Arc Angle");
       Modetype.addItem("* Vary Arc Angle");
    }
    else {
       Modetype.addItem("* Fixed Arc Angle");
       Modetype.addItem("   Vary Arc Angle");
    }
    if (rotate_flag) {
       Modetype.addItem("   Do Not Rotate");
       Modetype.addItem("* Rotate");
```

```
    }
    else {
       Modetype.addItem("* Do Not Rotate");
       Modetype.addItem("   Rotate");
    }
    if (fixed_images_flag) {
       Modetype.addItem("   Moving Images");
       Modetype.addItem("* Fixed Images");
    }
    else {
       Modetype.addItem("* Moving Images");
       Modetype.addItem("   Fixed Images");
    }
    if (include_string_flag) {
       Modetype.addItem("* Use String");
       Modetype.addItem("   Use Strings");
       Modetype.addItem("   No String");
    }
    else if (include_strings_flag) {
       Modetype.addItem("   Use String");
       Modetype.addItem("* Use Strings");
       Modetype.addItem("   No String");
    }
    else {
       Modetype.addItem("   Use String");
       Modetype.addItem("   Use Strings");
       Modetype.addItem("* No String");
    }
}

String coordstr(int x, int y) {
   return "("+x+","+y+")";
}

Color get_random_color() {
   int cred,cgreen,cblue;

   cred = colorlist[(int)(5*Math.random())];
   cgreen = colorlist[(int)(5*Math.random())];
   cblue = colorlist[(int)(5*Math.random())];
   return new Color(cred,cgreen,cblue);
}

int find_color_index(int val) {
   for (int i=0;i<colorlist.length;i++)
      if (colorlist[i]==val) return i;
   return -1;
}

Color get_next_color(Color C) {
   int red_index;
   int green_index;
   int blue_index;
   red_index = find_color_index(C.getRed());
   green_index = find_color_index(C.getGreen());
   blue_index = find_color_index(C.getBlue());
   if ( red_index < 0) return new Color(0,0,0);
   if ( green_index < 0) return new Color(0,0,0);
   if ( blue_index < 0) return new Color(0,0,0);
   red_index++;
   if (red_index >= colorlist.length) {
      red_index = 0;
      green_index++;
   }
   if (green_index >= colorlist.length) {
      green_index = 0;
      blue_index++;
   }
   if (blue_index >= colorlist.length) {
      blue_index = 0;
   }
   return new Color(colorlist[red_index],colorlist[green_index],
                    colorlist[blue_index]);
}

void randomize() {
   for (int i=0;i<10;i++)
      Math.random();
}

Color get_border_color() {
   if (border_color_type == 1)
      return get_random_color();
   return Color.black;
}

void set_object_image(JotsaAnimationObject ob) {
   int tp;
   tp = image_type;
   arc_flag = false;
   poly_flag = false;
   vary_images_flag = false;
   if (image_type == -1) {
      randomize();
      tp = (int)(17*Math.random());
   }
   if (tp == 0)
      obj.SetImage(external_image);
   else if (tp == 1)
      obj.SetFillOval(image_width,image_height,get_random_color());
   else if (tp == 2)
      obj.SetDrawOval(image_width,image_height,get_random_color());
   else if (tp == 3) {
      obj.SetDrawString("A String",get_random_color());
      obj.ResetFont(15);
   }
   else if (tp == 4)
      obj.SetDrawRect(image_width,image_height,get_random_color());
   else if (tp == 5)
      obj.SetFillRect(image_width,image_height,get_random_color());
   else if (tp == 6) {
      obj.SetDrawString("Line 1\n Line 2\n  Longer Line 3\n"+
          "  Line 4\n   Line 5",get_random_color());
      obj.ResetFont(10);
      obj.SetHighlighted(1,2,get_random_color());
   }
   else if (tp == 7) {
      obj.SetDrawSector(image_width,image_height,start_angle,arc_angle,
                        get_random_color());
      arc_flag = true;
   }
   else if (tp == 8) {
      obj.SetDrawRegularPoly((double)image_width,
                             poly_sides,get_random_color());
      poly_flag = true;
   }
   else if (tp == 9) {
      obj.SetFillRegularPoly((double)image_width,
                             poly_sides,get_random_color());
      poly_flag = true;
   }
   else if (tp == 10) {
      obj.SetImage(ext_images[0]);
      vary_images_flag = true;
   }
   else if (tp == 11) {
      obj.SetDrawArc(image_width,image_height,start_angle,arc_angle,
                     get_random_color());
      arc_flag = true;
   }
   else if (tp == 12) {
      obj.SetFillSector(image_width,image_height,start_angle,arc_angle,
                        get_random_color());
      arc_flag = true;
   }
   else if (tp == 13) {
      set_object_pie_chart(obj);
   }
   if (position_centered_flag)
      obj.SetPositionCentered();
   if (position_right_justified_flag)
      obj.SetPositionRightJustified();
   if (border_color_type > 0)
      obj.SetBorderColor(get_border_color());
   if ( (tp!=3) && (tp!=10) )
      if (include_string_flag) {
         obj.ResetString("Abc");
         obj.ResetFont("Times",Font.PLAIN,20);
         obj.ResetColorString(get_random_color());
      }
      if (include_strings_flag) {
         obj.ResetString("Line 1\n Line 2\n  Longer Line 3\n"+
          "  Line 4\n   Line 5");
         obj.ResetFont("Times",Font.PLAIN,8);
         obj.ResetColorString(get_random_color());
         obj.SetHighlighted(1,2,get_random_color());
      }
}

void set_object_pie_chart(JotsaAnimationObject obj) {
   double[] fra;
   Color[] cols;
   String[] strs;
   fra = new double[poly_sides];
   cols = new Color[poly_sides];
   strs = new String[poly_sides];
   for (int i=0;i<poly_sides;i++) {
      fra[i] = 1.0/poly_sides;
      cols[i] = get_random_color();
      strs[i] = ""+i;
   }
   obj.SetFillPiechart(image_width,image_height,start_angle,fra,cols);
   obj.SetPiechartStrings(strs);
   obj.ResetColorString(Color.black);
   obj.ResetString("");
   obj.SetSectorStringPosition(0.67);
   arc_flag = true;
}
```

```
public void JotsaChangeParameters() {
   JotsaSetRate(CP.time_control.value()/1000.0);
   JotsaForceRedisplay();
}

public void JotsaChangeParameters(int key, Object obj) {
   if (key == CP.KEY_START_ANGLE) {
      start_angle = ((Integer)obj).intValue();
      JotsaForceRedisplay();
   }
   if (key == CP.KEY_ARC_ANGLE) {
      arc_angle = ((Integer)obj).intValue();
      JotsaForceRedisplay();
   }
   if (key == CP.KEY_END_ANGLE) {
      end_angle = ((Integer)obj).intValue();
      JotsaForceRedisplay();
   }
   if (key == CP.KEY_END_ARC_ANGLE) {
      end_arc_angle = ((Integer)obj).intValue();
      JotsaForceRedisplay();
   }
   if (key == CP.KEY_START_WIDTH) {
      image_width = ((Integer)obj).intValue();
      JotsaForceRedisplay();
   }
   if (key == CP.KEY_END_WIDTH) {
      image_end_width = ((Integer)obj).intValue();
      JotsaForceRedisplay();
   }
   if (key == CP.KEY_START_HEIGHT) {
      image_height = ((Integer)obj).intValue();
      JotsaForceRedisplay();
   }
   if (key == CP.KEY_END_HEIGHT) {
      image_end_height = ((Integer)obj).intValue();
      JotsaForceRedisplay();
   }
   if (key == CP.KEY_NUM_SIDES) {
      poly_sides = ((Integer)obj).intValue();
      JotsaForceRedisplay();
   }
   if (key == CP.KEY_MOVE_TIME) {
      move_time = ((Integer)obj).intValue();
      JotsaForceRedisplay();
   }
}

public void JotsaPaintLocal(long ldt) {
   timeinfo.ResetString(""+ldt/1000.0);
}


public void make_colored_object(int cr, int cg, int cb) {
   int start_x;
   int start_y;
   int cred;
   int cgreen;
   int cblue;
   int level;
   start_x = 100+15*cr + 100*cb;
   start_y = 120+30*cg;
   level = JotsaNextLevel();
```

```
      obj = new JotsaAnimationObject(start_x,start_y,level,level,this);
      cred = colorlist[cr];
      cgreen = colorlist[cg];
      cblue = colorlist[cb];
      obj.SetFillRect(10,20,new Color(cred,cgreen,cblue));
      obj.SetPositionCentered();
      JotsaInsertObject(obj,can);
   }

   public void make_colored_headings() {
      int i;
      int level;
      for (i=0;i<5;i++) {
         level = JotsaNextLevel();
         obj = new JotsaAnimationObject(130+100*i,80,level,level,this);
         obj.SetDrawString("blue = "+colorlist[i],new Color(0,0,0));
         obj.SetPositionCentered();
         JotsaInsertObject(obj,can);
         level = JotsaNextLevel();
         obj = new JotsaAnimationObject(10,125+30*i,level,level,this);
         obj.SetDrawString("green = "+colorlist[i],new Color(0,0,0));
         JotsaInsertObject(obj);
      }
   }

   public void change_to_canvas(JotsaCanvas c) {
      if (c == null) return;
      if (c == can) return;
      if (can != null)
         System.out.println("Changing canvases from "+can.GetName()+" to "+
                            c.GetName());
      else
         System.out.println("Changing canvases to "+ c.GetName());
      deactivate_canvas(can);
      activate_canvas(c);
      can = c;
   }


//   public boolean handleEvent(Event e) {
   public boolean JotsaHandleCanvasEvent(Event e, JotsaCanvas canv) {
      int level;
      if (e.id == Event.MOUSE_DOWN) {
         change_to_canvas(canv);
         start_x = e.x;
         start_y = e.y;
         showStatus("Start position set to "+coordstr(start_x,start_y));
         if (link_level_flag)
            if (selected < 0)
               link_level_flag = false;
         if (link_level_flag) {
            level = JotsaNextLevel();
            obj = new JotsaAnimationObject(start_x,start_y,level,level,this);
            obj.SetDrawString(""+selected_object.GetLevel(),
                get_random_color());
            obj.SetPositionCentered();
            obj.PathSetLinked(selected_object,0,0);
            JotsaInsertObject(obj,can);
            JotsaForceRedisplay();
            return true;
         }
         if (!last_line_flag) {
            oldobj1 = oldobj;
            oldobj = lastobj;
         }
         last_line_flag = false;
         level = JotsaNextLevel();
         obj = new JotsaAnimationObject(start_x,start_y,level,level,this);
         if (line_flag) {
            if ((oldobj == null) || (oldobj1 == null) )
               System.out.println("Not enough objects for line");
            else {
               obj.SetDrawLine(oldobj,0,0,oldobj1,0,0,get_random_color());
               if (front_line_type == 1)
                  obj.SetFrontArrow(10,20,get_random_color());
               else if (front_line_type == 2)
                  obj.SetFrontArrowSolid(10,20,get_random_color());
               if (rear_line_type == 1)
                  obj.SetRearArrow(5,10,get_random_color());
               else if (rear_line_type == 2)
                  obj.SetRearArrowSolid(5,10,get_random_color());
               last_line_flag = true;
               showStatus("Creating a line between two other objects");
            }
         }
         else
            set_object_image(obj);
         if ( (link_flag) && (selected >= 0) ){
            selected_object.PathSetLinked(obj,link_off_x,link_off_y);
            link_flag = false;
         }
         JotsaInsertObject(obj,can);
         if ( (link_index_flag) && (selected >= 0) ) {
            selected_object.PathSetIndexLinked(obj);
            link_index_flag = false;
         }
         JotsaForceRedisplay();
      }
      else if (e.id == Event.MOUSE_UP) {
         end_x = e.x;
         end_y = e.y;
//         move_time = speed_control.value();
         if (link_level_flag || line_flag) {
            line_flag = false;
            link_level_flag = false;
            return true;
         }
         else {
            lastobj = obj;
            if (end_x < 0) end_x = 0;
            if (end_y < 0) end_y = 0;
            if (end_x > width) end_x = width;
            if (end_y > height) end_y = height;
            if (vary_images_flag && fixed_images_flag) {
               for (int i=0;i<NUM_EXT_IMAGES*REPEAT_EXT_IMAGES;i++) {
                  fixed_path_x[i]=start_x;
                  fixed_path_y[i]=start_y;
               }
               showStatus("Images fixed at "+coordstr(start_x,start_y));
               obj.PathSet(fixed_path_x,fixed_path_y);
               obj.TimesSetFrame(((double)move_time)/fixed_path_x.length);
            }
            else {
               showStatus("Moving object from "+coordstr(start_x,start_y)+
               " to "+coordstr(end_x,end_y)+" in "+move_time+" milliseconds");
               obj.PathCreateAlongLine(start_x,start_y,end_x,end_y);
               obj.TimesSet(move_time);
            }
            if ( (obj.GetImageType() != 0) && vary_size )
               obj.SetSizeLinear(image_width,image_end_width,
                                 image_height,image_end_height);
            if ( (obj.GetImageType() != 0) && vary_color )
               obj.SetColorLinear(Color.green,Color.red);
            if (poly_flag && rotate_flag)
               obj.SetAnglesLinear(0,720);
            if (vary_images_flag) {
               obj.SetImages(ext_images);
               if (position_centered_flag)
                  obj.SetPositionCentered();
               if (position_right_justified_flag)
                  obj.SetPositionRightJustified();
            }
            if (arc_flag) {
               if (vary_start_angle)
                  obj.SetStartAnglesLinear(start_angle,end_angle);
               if (vary_arc_angle)
                  obj.SetArcAnglesLinear(arc_angle,end_arc_angle);
            }
         }
         obj.ActivateDelay();
         JotsaForceRedisplay();
      }
      return super.handleEvent(e);
   }

   private void deactivate_canvas(JotsaCanvas c) {
      if (c == null) return;
      if (c.GetRelatedInfo() == null) return;
System.out.println("Deactivating Canvas :"+c.GetName());
      c.GetRelatedInfo().SetInhibitDisplay();
      c.RecreateBackground();
      JotsaForceRedisplay();
}

private void activate_canvas(JotsaCanvas c) {
      if (c == null) return;
      if (c.GetRelatedInfo() == null) return;
System.out.println("Activating Canvas :"+c.GetName());
      c.GetRelatedInfo().ClearInhibitDisplay();
      c.RecreateBackground();
      JotsaForceRedisplay();
}

public boolean action(Event e, Object arg) {
      JotsaAnimationObject tempanim;
      Color old_color;
      Color new_color;
      int dim[];
//System.out.println("jtest2 action entered");
      if (e.target instanceof Choice) {
         String choice = (String)arg;
         if (choice.equals("Random Image")) {
            image_type = -1;
            return true;
         }
         else if (choice.equals("GIF Image")) {
            image_type = 0;
            return true;
         }
```

```
else if (choice.equals("Fill Oval")) {
   image_type = 1;
   return true;
}
else if (choice.equals("Draw Oval")) {
   image_type = 2;
   return true;
}
else if (choice.equals("Draw String")) {
   image_type = 3;
   return true;
}
else if (choice.equals("Draw Rectangle")) {
   image_type = 4;
   return true;
}
else if (choice.equals("Fill Rectangle")) {
   image_type = 5;
   return true;
}
else if (choice.equals("Draw Strings")) {
   image_type = 6;
   return true;
}
else if (choice.equals("Draw Sector")) {
   image_type = 7;
   return true;
}
else if (choice.equals("Draw Polygon")) {
   image_type = 8;
   return true;
}
else if (choice.equals("Fill Polygon")) {
   image_type = 9;
   return true;
}
else if (choice.equals("GIF Images")) {
   show_tracker_status();
   if (tracker.statusAll(true) != MediaTracker.COMPLETE) {
      System.out.println(
         "Image loading not yet complete, try again later");
      image_type = -1;
      Imagetype.select("Random Image");
      return true;
   }
   image_type = 10;
   return true;
}
else if (choice.equals("Draw Arc")) {
   image_type = 11;
   return true;
}
else if (choice.equals("Fill Sector")) {
   image_type = 12;
   return true;
}
else if (choice.equals("Pie Chart")) {
   image_type = 13;
   return true;
}
else if (choice.equals("   Position Norm")) {
   position_centered_flag = false;
   position_right_justified_flag = false;
   reset_mode_entries();
   return true;
}
else if (choice.equals("   Position Centered")) {
   position_centered_flag = true;
   position_right_justified_flag = false;
   reset_mode_entries();
   return true;
}
else if (choice.equals("   Position Right Justified")) {
   position_centered_flag = false;
   position_right_justified_flag = true;
   reset_mode_entries();
   return true;
}
else if (choice.equals("   Random Border")) {
   border_color_type = 1;
   reset_mode_entries();
   return true;
}
else if (choice.equals("   Black Border")) {
   border_color_type = 2;
   reset_mode_entries();
   return true;
}
else if (choice.equals("   No Border")) {
   border_color_type = 0;
   reset_mode_entries();
   return true;
}
else if (choice.equals("   Fixed Size")) {
   vary_size = false;
   reset_mode_entries();
   return true;
}
else if (choice.equals("   Vary Size")) {
   vary_size = true;
   reset_mode_entries();
   return true;
}
else if (choice.equals("   Fixed Color")) {
   vary_color = false;
   reset_mode_entries();
   return true;
}
else if (choice.equals("   Vary Color")) {
   vary_color = true;
   reset_mode_entries();
   return true;
}
else if (choice.equals("   Front Line Norm")) {
   front_line_type = 0;
   reset_mode_entries();
   return true;
}
else if (choice.equals("   Front Line Arrow")) {
   front_line_type = 1;
   reset_mode_entries();
   return true;
}
else if (choice.equals("   Front Line Solid")) {
   front_line_type = 2;
   reset_mode_entries();
   return true;
}
else if (choice.equals("   Rear Line Norm")) {
   rear_line_type = 0;
   reset_mode_entries();
   return true;
}
else if (choice.equals("   Rear Line Arrow")) {
   rear_line_type = 1;
   reset_mode_entries();
   return true;
}
else if (choice.equals("   Rear Line Solid")) {
   rear_line_type = 2;
   reset_mode_entries();
   return true;
}
else if (choice.equals("   Fixed Start Angle")) {
   vary_start_angle = false;
   reset_mode_entries();
   return true;
}
else if (choice.equals("   Vary Start Angle")) {
   vary_start_angle = true;
   reset_mode_entries();
   return true;
}
else if (choice.equals("   Fixed Arc Angle")) {
   vary_arc_angle = false;
   reset_mode_entries();
   return true;
}
else if (choice.equals("   Vary Arc Angle")) {
   vary_arc_angle = true;
   reset_mode_entries();
   return true;
}
else if (choice.equals("   Do Not Rotate")) {
   rotate_flag = false;
   reset_mode_entries();
   return true;
}
else if (choice.equals("   Rotate")) {
   rotate_flag = true;
   reset_mode_entries();
   return true;
}
else if (choice.equals("   Moving Images")) {
   fixed_images_flag = false;
   reset_mode_entries();
   return true;
}
else if (choice.equals("   Fixed Images")) {
   fixed_images_flag = true;
   reset_mode_entries();
   return true;
}
else if (choice.equals("   Use Strings")) {
   include_string_flag = false;
   include_strings_flag = true;
   reset_mode_entries();
   return true;
}
```

```
   else if (choice.equals("   Use String")) {
      include_string_flag = true;
      include_strings_flag = false;
      reset_mode_entries();
      return true;
   }
   else if (choice.equals("   No String")) {
      include_string_flag = false;
      include_strings_flag = false;
      reset_mode_entries();
      return true;
   }
   else if (choice.substring(0,1).equals("*")) {
      reset_mode_entries();
      return true;
   }
}
if ("Start".equals(arg)) {
   System.out.println("Start Pushed");
   JotsaRestartTime();
   return true;
}
if ("Stop".equals(arg)) {
   System.out.println("Stop Pushed");
   JotsaStopTime();
   return true;
}
if ("Select Next".equals(arg)) {
   System.out.println("Select Pushed");
   try_log_line("Select Next Pushed");
   if (JotsaNumObjects() < 1) return true;
   if (selected >= 0) {
      tempanim = JotsaGetObjectAt(selected);
      tempanim.SetShake(0);
   }
   selected++;
   if (JotsaNumObjects() <= selected)
      selected = 0;
   tempanim = JotsaGetObjectAt(selected);
   tempanim.SetShake(10);
   tempanim.ClearShakeAll();
   selected_object = tempanim;
   return true;
}
if ("Select All".equals(arg)) {
   System.out.println("Select All Pushed");
   if (selected < 0)
      return true;
   tempanim = JotsaGetObjectAt(selected);
   tempanim.SetShakeAll();
   return true;
}
if ("Select None".equals(arg)) {
   System.out.println("Select None Pushed");
   if (selected < 0) return true;
   tempanim = JotsaGetObjectAt(selected);
   selected = -1;
   tempanim.SetShake(0);
   return true;
}
if ("Link".equals(arg)) {
   System.out.println("Link Pushed");
   link_flag = true;
```

```
         return true;
      }
      if ("Line".equals(arg)) {
         System.out.println("Line Pushed");
         if (JotsaNumObjects() < 2) {
            System.out.println("At least 2 objects must be created first");
            return true;
         }
         if (last_line_flag) {
            System.out.println("Already have a line there");
            return true;
         }
         line_flag = true;
         return true;
      }
      if ("Link Index".equals(arg)) {
         System.out.println("Link Index Pushed");
         try_log_line("Link Index Pushed");
         link_index_flag = true;
         return true;
      }
      if ("Loop".equals(arg)) {
         System.out.println("Loop Pushed");
         try_log_line("Loop Pushed");
         if (selected >= 0) {
            selected_object.PathSetLoop();
         }
         return true;
      }
      if ("UnLoop".equals(arg)) {
         System.out.println("UnLoop Pushed");
         if (selected >= 0) {
            selected_object.PathClearLoop();
         }
         return true;
      }
      if ("Time +".equals(arg)) {
         System.out.println("Time Up Pushed");
//          JotsaDefaultCanvas.SetResizeable(true);
         JotsaAddToVirtualTime(100);
         return true;
      }
      if ("Time -".equals(arg)) {
         System.out.println("Time Down Pushed");
//          JotsaDefaultCanvas.SetResizeable(false);
         JotsaAddToVirtualTime(-100);
         return true;
      }
      if ("Link Level".equals(arg)) {
         System.out.println("Link Level Pushed");
         link_level_flag = true;
         return true;
      }
      if ("Reset Time".equals(arg)) {
         System.out.println("Reset Time Pushed");
         JotsaClearVirtualTime();
         return true;
      }
      if ("Color".equals(arg)) {
         System.out.println("Color Pushed");
         if (selected < 0) return true;
         old_color = selected_object.GetColor();
         new_color = get_next_color(old_color);
```

```
   selected_object.ResetColor(new_color);
   selected_object.ResetColorString(get_next_color(new_color));
   return true;
}
if ("Remove".equals(arg)) {
   System.out.println("Remove Pushed");
   if (selected < 0) return true;
   selected_object.SetRemoveTime();
   return true;
}
if ("Palette".equals(arg)) {
   System.out.println("Palette Pushed");
   for (int i=0;i<5;i++)
      for (int j=0;j<5;j++)
         for (int k=0;k<5;k++)
      make_colored_object(k,j,i);
      make_colored_headings();
   return true;
}
if ("Popup".equals(arg)) {
   System.out.println("Popup Pushed");
   popup_count++;
   if (can == null)
      System.out.println("Canvas is null");
   else
      System.out.println("Canvas is: "+can.GetName());
   deactivate_canvas(can);
   can = JotsaMakePopupWindow("Popup "+popup_count,400,300).can;
   tempanim = can.WriteBackgroundString("This is the active popup",
        20,50,Color.green);
   can.SetRelatedInfo(tempanim);
   activate_canvas(can);
   return true;
}
if ("Scaled".equals(arg)) {
   System.out.println("Scaled Pushed");
   scaled_count++;
   swin = JotsaMakeScaledWindow("Scaled "+scaled_count,400,300);
   return true;
}
if ("Resize".equals(arg)) {
   System.out.println("Resize Pushed");
   JotsaDefaultCanvas.ResetSize();
   JotsaForceRedisplay();
   return true;
}
if ("Controls".equals(arg)) {
   System.out.println("Controls Pushed");
   CP.show();
   return true;
}
if ("Quit".equals(arg)) {
   System.out.println("Quit Pushed");
   try {System.exit(0);}
   catch (SecurityException e1) {
      System.out.println("Not allowed to exit from this implementation");
      if (backURL != null) {
         System.out.println("Trying to transfer to "+backURL);
         getAppletContext().showDocument(backURL);
      }
   }
}
return super.action(e,arg);
```

```
   }

   public void JotsaPopupChosen(JotsaPopupWindow win) {
      deactivate_canvas(can);
      activate_canvas(win.can);
   }

   public void JotsaAllPopupsChosen() {
      deactivate_canvas(can);
      can = null;
   }

   void try_log_line(String str) {
      if (JotsaLog == null) return;
      if (JotsaLog.IsOpen())
         JotsaLog.WriteLineDate(str);
   }

   public void JotsaCanvasResized(int oldx, int oldy, int newx, int newy,
                                  JotsaCanvas can) {
//      System.out.println("Canvas size changed from "+oldx+" "+oldy+" to "+
//           newx+" "+newy);
      resize(bounds().width+newx-oldx,bounds().height+newy-oldy);
   }



}
```