Using the Objectdraw Library in Grace

Kim Bruce

1 Introduction

The purpose of this document is to explain how to use the *objectdraw* library in Grace[1]. We assume the reader is familiar with Grace. While not necessary, it will be helpful if the user is already familiar with the objectdraw library in Java. A brief description is available at http://eventfuljava.cs.williams.edu/library.html. More detailed information is available in the text Java: An eventful Approach [2]. A quick summary of commands is also available at http://www.cs.pomona.edu/~kim/CSC051F12/handouts/quickreference.pdf.

2 Installation

To run the objectdraw library, the user must have GTK+ installed on their computer as well as X11 (XQuartz on the Mac). Information on installing GTK+ is available at http://www.gtk.org/download/index.php. *Warning: This takes some time and may require assistance from a knowledgable systems manager.* A version of X11 for the Mac, called XQuartz, is available at http://xquartz.macosforge.org/landing/.

The user must also have installed Grace, including grace-gtk. Information on installing this is available at http://gracelang.org/applications/minigrace/. The files `objectdraw.grace` and `math.grace` should be included in the directory `grace-gdk`. These files can be found at http://www.cs.pomona.edu/~kim/GraceStuff/objectdraw.grace and http://www.cs.pomona.edu/~kim/GraceStuff/math.grace.

Programs using objectdraw should also be located in that directory.

3 Overview of Objectdraw Library

The objectdraw library allows users to create applications using graphics built on the GDK+ library, though the interface completely hides the GDK primitives.

The user creates graphic objects including rectangles, ovals, lines, and text to label them on a canvas in a window. The main program will be defined in an object that is of type GraphicApplication. The program is written by inheriting from the class MkGraphicApplication. Figure 1 is an example of a simple program using the objectdraw library.

When the object is initialized a blank window with the title "Simple Objectdraw Demo" will be created. Each time the user presses the mouse button down and then drags a red rectangle will be created and will follow the mouse until the mouse button is released. The following is a brief explanation of the code in the program.

Every object that corresponds to a graphics program should inherit from objectdraw's MkGraphicApplication class, providing the width and height of the window.

The first item in the object should always be:

```
canvas.doSetUp(self,window)
```

```
import "objectdraw" as od

// Applet that draws rect when clicked -- dragging moves it.
// class Clicker. size(width':Number,height':Number)
def Clicker: od.GraphicApplication = object {
    inherits od.MkGraphicApplication.size(400,400)

    // required to wire up the application to respond to mouse events
    canvas.doSetUp(self,window)

    // set title of window displaying graphics (self shouldn't be necessary)
    self.windowTitle:= "Simple Objectdraw Demo"

    // item to be moved
    var cloth: od.Resizable2DObject

    // When the user presses mouse, create new rectangle at that point
    // to be moved by dragging mouse
    method onMousePress(mousePoint:od.Location)->Done{
        cloth := od.FilledRect.at(mousePoint)size(100,100)on(canvas)
        cloth.color := od.red
    }

    // When mouse is dragged, rectangle follows it
    method onMouseDrag(mousePoint:od.Location)->Done{
        cloth.moveTo(mousePoint)
    }

    // required to pop up window and start graphics
    startGraphics
}
```

Figure 1: Program using objectdraw to drag rectangles in a window

Executing this method request will wire up the application to receive mouse events.

After that you can declare whatever objects are needed for your application (in this case only a variable cloth representing red squares to be dragged). The method onMousePress is executed when the mouse button is depressed, while onMouseDrag is executed repeatedly while the mouse is dragged (with the button depressed).

For this program, when the mouse is depressed and new red square is created at the location of the mouse. When the mouse is dragged the last red square created is moved to the new location of the mouse (continuously during the dragging).

The command startGraphics must be executed at the end of the object definition in order to create the window and start the graphics.

4 Graphics Classes

While the library has a number of types and classes, we will only describe here the ones likely to be of interest to end users of the objectdraw library. The classes will generate both framed (outlined) and filled (solid) rectangles and ellipses as well as straight lines and text items that can be put on the canvas. The type and the associated classes for generating two-dimensional objects that can be resized are shown in Appendix A. All four of these classes of objects respond to the same messages, though they represent different kinds of shapes.

The objectdraw library also supports lines. The type and class for lines are shown in Appendix B. Finally, the type and class for generating text items on the screen are given in Appendix C. Graphics items may be in any color. The type, class, and pre-defined color constants are available in Appendix D.

5 Responding to Mouse Events

The objectdraw library is designed to be used in interactive programs. Thus it provides methods that will be called by the system to respond to mouse events. Methods to respond to mouse presses, releases, clicks, moves, drags, and entering or leaving the window are supported. The methods listed below are defined in class MkGraphicApplication. They should be overridden as desired in any object the inherits from MkGraphicsApplication.

The methods to respond to these events are given in Appendix E. The appropriate methods will be called by the system each time the corresponding mouse event occurs. The mousePoint parameter will be supplied by the system and corresponds to the location of the mouse when the event happened. Users should be aware that a mouse click generates three events: a press event, a release event, and then a click event.

6 Summary

The objectdraw library has been ported from Java to Grace. The types, classes, and methods in Grace are similar to those in Java, but take advantage of features in Grace like multi-part method names and methods that look like assignment statements.

A few sample programs using the objectdraw library in Grace are available in http://www.cs.pomona.edu/~kim/GraceStuff/GraceObjectdrawSamplePrograms/.

The main thing missing from the objectdraw library at this point is features that will support animations. Because Grace does not currently support true parallelism and concurrency, these will be supported by adding timer events to trigger actions. These will be added to Grace in the near future.

References

[1] A. P. Black, K. B. Bruce, M. Homer, J. Noble, A. Ruskin, and R. Yannow. Seeking Grace: A new object-oriented language for novices. In *ACM Conference on Computer Science Education (SIGCSE)*, 2013.

[2] K. Bruce, A. Danyluk, and T. Murtagh. *Java: An Eventful Approach*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2005.

A Type for Two-Dimensional Objects and Classes to Generate Them

```
// Two−dimensional objects that can also be resized
type Resizable2DObject = Graphic2DObject & {
    // location of object on screen
    x −> Number
    y −> Number
    location −> Location

    // Dimensions of object
    width−>Number
    height−>Number

    // Change dimensions of object
    setSize(width:Number,height:Number)−>Done
    width:=(width:Number)−>Done
    height:=(height:Number)−>Done

    // return the color of this object
    color−>GColor

    // set the color of this object to c
    color:=(c:GColor)−>Done

    // Determine if object is whown on screen
    isVisible :=(_:Boolean)−>Done

    // Is this object visible on the screen?
    isVisible −>Boolean

    // move this object to newLocn
    moveTo(newLocn:Location)−>Done

    // move this object dx to the right and dy down
    moveBy(dx:Number,dy:Number)−>Done

    // Does this object contain locn
    contains(locn:Location)−>Boolean

    // Does other overlap with this object
    overlaps(other:Graphic2DObject)−>Boolean

    // Send this object up one layer on the screen
    sendForward−>Done

    // send this object down one layer on the screen
    sendBackward −> Done

    // send this object to the top layer on the screen
    sendToFront −> Done

    // send this object to the bottom layer on the screen
    sendToBack −> Done
```

```
    // Return a string representation of the object
    asString -> String    // dimensions of object

    // The canvas this object is part of !! confidential
    canvas->Canvas

    // Add this object to canvas c
    addToCanvas(c:Canvas)->Done

    // Remove this object from its canvas
    removeFromCanvas->Done
}

// class to generate framed rectangle at (x',y') with size width' x height'
// created on canvas'
class FramedRect.at(location':Location)size(width':Number,height':Number)
                    on(canvas':Canvas) -> Resizable2DObject

// class to generate filled rectangle at (x',y') with size width' x height'
// created on canvas'
class FilledRect.at(location':Location)size(width':Number,height':Number)
                    on(canvas':Canvas) -> Resizable2DObject

// class to generate framed oval at (x',y') with size width' x height'
// created on canvas'
class FramedOval.at(location':Location)size(width':Number,height':Number)
                    on(canvas':Canvas) -> Resizable2DObject

// class to generate filled oval at (x',y') with size width' x height'
// created on canvas'
class FilledOval.at(location':Location)size(width':Number,height':Number)
                    on(canvas':Canvas) -> Resizable2DObject
```

B Type for Lines and Class to Generate Lines

```
type LinearObject = GraphicObject & {
    // start and end of line
    start –> Location
    end –> Location

    // set start and end of line
    start :=( start ': Location) –> Done
    end:=(end':Location) –> Done
    setEndPoints(start ': Location,end':Location) –> Done

    // location of start of line on screen
    x –> Number
    y –> Number
    location –> Location

    // Is this object visible on the screen?
    isVisible –>Boolean

    // Determine if object is whown on screen
    isVisible :=(_:Boolean)–>Done

    // Add this object to canvas c
    addToCanvas(c:Canvas)–>Done

    // Remove this object from its canvas
    removeFromCanvas–>Done

    // The canvas this object is part of !! confidential
    canvas–>Canvas

    // move this object to newLocn
    moveTo(newLocn:Location)–>Done

    // move this object dx to the right and dy down
    moveBy(dx:Number,dy:Number)–>Done

    // set the color of this object to c
    color:=(c:GColor)–>Done

    // return the color of this object
    color–>GColor

    // Send this object up one layer on the screen
    sendForward–>Done

    // send this object down one layer on the screen
    sendBackward –> Done

    // send this object to the top layer on the screen
    sendToFront –> Done
```

```
    // send this object to the bottom layer on the screen
    sendToBack -> Done

    // Return a string representation of the object
    asString -> String

// class to generate a line from start' to end' on canvas'
class Line.from(start ': Location)to(end':Location)on(canvas':Canvas)
            -> LinearObject
```

C Type for Text Items and Class to Generate Them

```
type Textual = {
    // location of object on screen
    x -> Number
    y -> Number
    location -> Location

    // Is this object visible on the screen?
    isVisible ->Boolean

    // Determine if object is whown on screen
    isVisible :=(_:Boolean)->Done

    // Add this object to canvas c
    addToCanvas(c:Canvas)->Done

    // Remove this object from its canvas
    removeFromCanvas->Done

    // The canvas this object is part of !! confidential
    canvas->Canvas

    // move this object to newLocn
    moveTo(newLocn:Location)->Done

    // move this object dx to the right and dy down
    moveBy(dx:Number,dy:Number)->Done

    // set the color of this object to c
    color:=(c:GColor)->Done

    // return the color of this object
    color->GColor

    // Send this object up one layer on the screen
    sendForward->Done

    // send this object down one layer on the screen
    sendBackward -> Done

    // send this object to the top layer on the screen
    sendToFront -> Done

    // send this object to the bottom layer on the screen
    sendToBack -> Done

    // Return a string representation of the object
    asString -> String

    // current contents of the text
    contents -> String
```

```
    // Update the contents of the text item
    contents:=(s:String) -> Done

    // font size of text showing in item
    fontSize -> Number

    // update font size
    fontSize:=(size:Number) -> Done
}

// class to generate text
class Text.at(location ':Location)with(contents':String)on(canvas':Canvas)
                                -> Textual
```

D Colors in Objectdraw

```
type GColor = {
    red –> Number
    green –> Number
    blue –> Number

    asString –> String
}

// Simple color class
class MkColor.r(r').g(g').b(b') –> GColor

// predefined colors in objectdraw
def white:GColor = MkColor.r(255)g(255)b(255)
def black:GColor = MkColor.r(0)g(0)b(0)
def green:GColor = MkColor.r(0)g(255)b(0)
def red:GColor = MkColor.r(255)g(0)b(0)
def gray:GColor = MkColor.r(60)g(60)b(60)
def blue:GColor = MkColor.r(0)g(0)b(255)
def cyan:GColor = MkColor.r(0)g(255)b(255)
def magenta:GColor = MkColor.r(255)g(0)b(255)
def yellow:GColor = MkColor.r(255)g(255)b(0)
```

E Methods for Responding to Mouse Events

Methods listed below should be overridden as needed in objects that inherit from MkGraphicApplication
.

```
// response to mouse click at mousePoint
onMouseClick(mousePoint:Location)->Done

// response to mouse click at mousePoint
onMousePress(mousePoint:Location)->Done

// response to mouse release at mousePoint
onMouseRelease(mousePoint:Location)->Done

// response to mouse move at mousePoint
onMouseMove(mousePoint:Location)->Done

// response to mouse click at mousePoint
onMouseDrag(mousePoint:Location)->Done

// response to mouse entry to window at mousePoint
// currently not available for use
onMouseEnter(mousePoint:Location)->Done

// response to mouse exit of window at mousePoint
// currently not available for use
onMouseExit(mousePoint:Location)->Done
```