MorphicDraw

Stephan J.C. Eggermont, Sensus

April 28, 2015

MorphicDraw is a drawing application demonstrating some of the power of Morphic. Morphic is a powerful graphics environment, used in Self, Squeak, Cuis and Pharo. In an iterative and incremental process we'll build up an application that supports drawing connected figures.

1 A Morphic Application

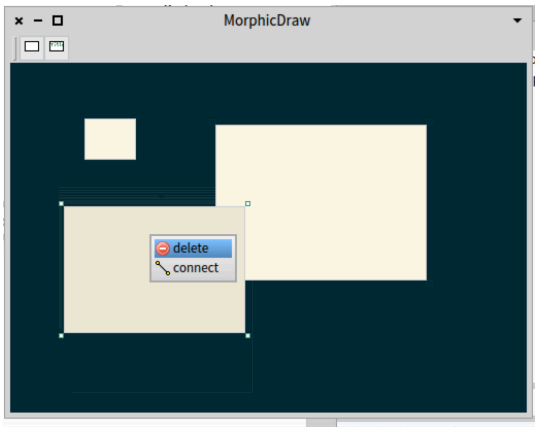

Figure 1: A first iteration of the main window of MorphicDraw

The first iteration (Figure 1) shows an application window with a toolbar and a drawing area. In the drawing area there are three graphical shapes, one of which is selected. A context menu for the selected shape shows options to delete it and to connect it.

1.1 An application with a window

Add a class that represents the application. It has instance variables for the different parts.

```
Object subclass: #MorphicDraw
    instanceVariableNames: 'window tools dock'
```

```
    classVariableNames: ''
    category: 'MorphicDraw-Model'
```

Creating a window in Morphic is simple. Open a workspace and DoIt

```
StandardWindow new openInWorld
```

This creates a window and opens it on the screen. It already has default behaviour for closing and resizing, and a default title. All graphical elements in Morphic are subclasses of Morph, and the World is a container for all of them. Opening a Morph in the world positions it and makes it visible. An alternative to opening it directly is to add it to the (mouse) cursor. In Morphic this is called the hand. DoIt:

```
StandardWindow new openInHand
```

The window is then positioned by clicking.
The MorphicDraw application uses the first, but needs to change the window title and default size.

```
MorphicDraw>>createWindow
    window := StandardWindow new
        setLabel: 'MorphicDraw';
        extent: 400@400;
        yourself.
```

StandardWindow is part of PolyMorph. PolyMorph extends Morphic with theme-ability. PolyMorph widgets often strictly adhere to the theme, ignoring the Morphic-level setters for colors and borders defined in their superclassses. PolyMorph makes the Window responsible for adding predefined user interface widgets to the application Window. For that it uses the TEasilyThemed trait. It adds a lot (163 in my current image) of convenience methods.
In Morphic, a toolbar in a window has buttons on it. This iteration of MorphicDraw uses two buttons to be able to create two different graphical shapes.

```
MorphicDraw>>createNewCardButton
    ^ window
        newButtonFor: self
        getState: nil
        action: #newCard
        arguments: nil
        getEnabled: nil
        labelForm: MDIcons default cardIcon
        help: 'New Card' translated
```

The help text is shown when hovering the mouse over the button. The button is always enabled, and sends the #newCard message without any arguments to self when it is pressed. It has no state-dependent behaviour or shape. The icon for the button is provided by MDIcons default cardIcon (see the Icons section, p. 4).
The button for the other shape is similar:

```
MorphicDraw>>createNewRectangleButton
    ^ window
        newButtonFor: self
        getState: nil
        action: #newRectangle
        arguments: nil
        getEnabled: nil
        labelForm: MDIcons default rectangleIcon
        help: 'New Rectangle' translated
```

The newCard and newRectangle are implemented by creating a new morph of the right kind and opening it in the hand. Both MDShape is a subclass of BorderedMorph and MDCard a subclass of MDShape.

```
MorphicDraw>>newCard
    ^MDCard new openInHand


MorphicDraw>>newRectangle
    ^MDShape new openInHand
```

A toolbar in Morphic consists of two parts, a ToolDockingBar and a Toolbar. The Toolbar is added to the dock. A DockingBarMorph sticks to one edge of the Morph that it is added to.

```
MorphicDraw>>createToolBar
   tools := window newToolbar: (Array with: self createNewRectangleButton with: self createNewCardBut
    dock := window newToolDockingBar.
    dock addMorph: tools
```

This allows the creation and opening of the window

```
MorphicDraw>>open
     self createWindow.
     self createToolBar.
     window
         addMorph: dock
        fullFrame: ((0@0 corner: 1@0) asLayoutFrame bottomOffset: dock minExtent y);
        addMorph: MDPanel new
         fullFrame: ((0@0 corner: 1@1) asLayoutFrame topOffset: dock minExtent y).
    ^window openInWorld.
```

Some morphs automatically lay out their submorphs when they are added, others need to add an explicit layout strategy. The toolbar adds its toolbar buttons from left to right without a space. The toolbar dock by default takes up the top edge of the morph it is added to. For a window that is the area with the drag bar, title and window icons. A LayoutFrame that should fit the whole contents area of the window is created using a rectangle from 0@0 to 1@1.

```
(0@0 corner: 1@1) asLayoutFrame.
```

The left half would be (0@0 corner: 0.5@1). Space can be left at a side by using the bottom/top/left/rightOffset. The dock knows its minimum height

```
dock minExtent y
```

so uses a layout frame based on the top line, extending it at its bottom by the dock height. The MDPanel is a PasteUpMorph subclass containing all the drawing shapes.
At the class side add a method to open the application

```
MorphicDraw>>open
    ^self new open
```

2 Shapes and PasteUpMorph

The basic graphics entity in Morphic is a Morph. In a workspace, DoIt

```
Morph new openInWorld
```

This opens a small blue rectangle on the screen. This can be dragged around on the screen. That is functionality that can be reused. World is a subclass of PasteUpMorph. The default PasteUpMorph is another small rectangle, this time light green.

```
PasteUpMorph new openInWorld
```

After dragging a Morph onto a PasteUpMorph, it behaves as a canvas containing the Morph. Let's use a subclass of PasteUpMorph as the application canvas panel.

```
PasteUpMorph subclass: #MDPanel
    instanceVariableNames: ''
    classVariableNames: ''
    category: 'MorphicDraw-Model'
```

3 Icons

The current icons in Pharo are bitmap icons. Athens makes it possible to replace them by SVG, vector based icons. PolyMorph adds the ThemeIcons class to make it easy to manage an applications' icons. In this class a number of utility functions are defined to load and save icons in png format.
Create the png icons in an external program (these were created with Gimp), store them in a directory. Create a subclass of ThemeIcons

```
ThemeIcons subclass: #MDIcons
    instanceVariableNames: ''
    classVariableNames: ''
    category: 'MorphicDraw-Model'
```

Add two methods containing the names for small- and normal sized icons

```
MDIcons>>normalSizeNames
    "Answer the names of the normal icons"
    ^#('rectangle' 'connector' 'card')

MDIcons>>smallSizeNames
    "Answer the names of the small icons. None"
    ^#()
```

Provide a default instance of this class. At the class side, add a class instance variable and a lazy accessor

```
MDIcons class
    instanceVariableNames: 'default'

MDIcons class>>default
    ^default ifNil: [ ^self new ]
```

Now add the icons using a utility method. In a workspace, DoIt with directory replaced by the fully qualified path name of the directory containing the icon png files:

```
MDIcons default createIconMethodsFromDirectory: directory
```

This adds two methods for each png file, one with a base64 encoded contents and one icon form accessor. Form provides asMorph, so the icons can be tested by DoIt

```
MDIcons default connectorIcon asMorph openInWorld
```

This opens an ImageMorph with the icon in the World (Fig. 2). By selecting it with shift-alt, its halos are shown. With the cross halo the icon can be deleted.

Figure 2: ImageMorph with icon and halos

By having the resource contents in a method, the normal version control can be used and there is no need for specific resource management. This works well when the resources are small and can be manipulated in the image, like png icons.

4 Colors

The responsibility for the colors used in MorphicDraw is given to the MDColors class.

```
Object subclass: #MDColors
    instanceVariableNames: ''
    classVariableNames: ''
    category: 'MorphicDraw-Model'
```

At the moment there is no need to dynamically change these colors, so add an indicator that the colors are all defined class-side.

```
MDColors>>seeClassSide
```

At the class side, the colors for a Solarize color scheme are added

```
MDColors class>>base0
    ^Color fromHexString: '839496'

MDColors class>>base00
    ^Color fromHexString: '657b83'

MDColors class>>base01
    ^Color fromHexString: '586e75'
,,,
MDColors class>>red
    ^Color fromHexString: 'dc322f'

MDColors class>>violet
    ^Color fromHexString: '6c71c4'

MDColors class>>yellow
    ^Color fromHexString: 'b58900'
```

5 Toolbar

6 Connecting

7 Selection and resizing

8 Loading the code

The code can be found on www.smalltalkhub.com, in the repository StephanEggermont/MorphicDraw Open the Monticello Browser. Add a new repository of type smalltalkhub.com. The owner is StephanEggermont, the project is MorphicDraw. User and password are only needed when you

want to commit changes to the repository. Open the repository and load the latest version of MorphicDraw.