How to create a java color picker app

Note: This is an entry in the Asha Touch Competition 2012Q3

This article explains how I created my color picker app (http://store.ovi.com/content/177628) for touch and non-touch devices with attractive elements and Nokia UI library.

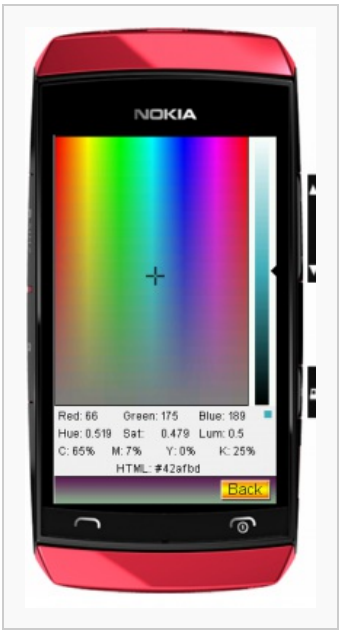
Introduction

A colour picker allows a user to select a colour from a palette or image and display its components using one of the main representation formats. Colour pickers can be a very important utility for graphic designers, webmasters, programmers, photography lovers etc.

This app allows you both to enjoy a full color palette similar to the windows paint application as well as selecting a color pixel from an image taken from camera or from the gallery folder of the phone. All of the color values are automatically translated into RGC, HSL, CMYK & HTML formats.

App screenshots are shown below:

main screen



palette screen

palette screen

palette screen

(C) Copyright Nokia 2013. All rights reserved.

Source code

The first image is the app main menu. It showcases all the different options that are available in the app - it can be achieved with either canvas based drawing of your own components (buttons etc), with images & simple drawing or you can use the native LCDUI form & components. In this case I used buttons.

The next screen is the palette screen, notice that we have 2 different kind of palette to fully control all the possible color values. One palette controls the luminance value and the other palette controls the Hue and Saturation values.

Here is my class for the luminance palette:

```
package com.future.rgbselector.ui.controls;

import java.io.IOException;

import javax.microedition.lcdui.Canvas;
import javax.microedition.lcdui.Graphics;
import javax.microedition.lcdui.Image;

import com.future.rgbselector.PaletteScreen;
import com.future.utils.JarDynamicResourceLoader;
import com.future.utils.UserInputListener;
import com.future.utils.graphics.GFX;
import com.future.utils.graphics.PaintableArea;
import com.future.utils.ui.controls.Dimensionable;

public class LumSlider extends Dimensionable implements PaintableArea, UserInputListener {

 private PaletteScreen parent;
 private Image pointerImg;
 private int pointerX, pointerY;
 private double hue, sat, lum;
 private int[] rgb;
 private int rgbX, rgbY, rgbH, rgbW;
 private boolean isFocused, paintFocus;
 private int jump;

 private static final int BORDER_WIDTH = 1;

 public LumSlider(PaletteScreen parent)
 {
  this.parent = parent;
  jump = 1;
  hue = PaletteScreen.INIT_HUE;
  sat = PaletteScreen.INIT_SAT;
  lum = PaletteScreen.INIT_LUM;
  try
  {
   pointerImg = JarDynamicResourceLoader.getImage("pointer.png");
  } catch (IOException e)
  {
   e.printStackTrace();
  }
 }

 public void paint(Graphics g)
 {
  int orgColor = g.getColor();
  g.setColor(0xFFFFFFFF);
  g.fillRect(x, y, width, height);
  if (paintFocus)
  {
   if (isFocused)
    g.setColor(0xFFFF0000);
   else
    g.setColor(0xFFF0F0F0);
   g.drawRect(x, y, width, height);
  }
  g.drawImage(pointerImg, pointerX, pointerY, Graphics.LEFT | Graphics.TOP);
  g.drawRGB(rgb, 0, rgbW, rgbX, rgbY, rgbW, rgbH, true);
  g.setColor(orgColor);
 }

 public void setPaintFocus(boolean paint)
 {
  paintFocus = paint;
 }

 public void updateHueSat(double hue, double sat)
 {
  this.hue = hue;
  this.sat = sat;
  updateRGB();
 }

 private void updateRGB()
 {
  int yIndexArray;
  double lum;
  for (int _y = 0; _y < rgbH; _y++)
  {
   yIndexArray = _y * rgbW;
   lum = 1 - (double) _y / rgbH;
   for (int _x = 0; _x < rgbW; _x++)
   {
    rgb[yIndexArray + _x] = HSL_TO_RGB(hue, sat, lum);
   }
  }
 }
                     public static int HSL_TO_RGB(double H, double S, double L)
 {
  int R, G, B;
  double a, b;
  if (S == 0) // HSL from 0 to 1
  {
   R = (int) (L * 255); // RGB results from 0 to 255
   G = (int) (L * 255);
   B = (int) (L * 255);
  } else
  {
   if (L < 0.5)
    b = L * (1 + S);
```


(C) Copyright Nokia 2013. All rights reserved.

```
    else
     b = (L + S) - (S * L);

    a = 2 * L - b;

    R = roundInt(255 * Hue_2_RGB(a, b, H + (1 / 3d)));
    G = roundInt(255 * Hue_2_RGB(a, b, H));
    B = roundInt(255 * Hue_2_RGB(a, b, H - (1 / 3d)));
   }
   return 0xFF000000 | (R << 16) | (G << 8) | B;
  }

  public void setPos(int _x, int _y)
  {
   super.setPos(_x, _y);
   setRgbPosDimen();
  }

  public void setDimension(int _width, int _height)
  {
   super.setDimension(_width, _height);
   setRgbPosDimen();
  }

  private void setRgbPosDimen()
  {
   rgbY = y + BORDER_WIDTH;
   rgbH = height - (BORDER_WIDTH * 2);
   rgbW = width / 2;
   rgbX = x + rgbW / 2;
   if (rgbH > 0 && rgbW > 0)
    rgb = new int[rgbH * rgbW];
   pointerX = x + width - pointerImg.getWidth();
   pointerY = (int) (rgbY + (lum * rgbH) - pointerImg.getHeight() / 2);
  }

  public void keyReleased(int keycode, int gameAction)
  {
   jump = 1;
  }

  public void keyPressed(int keycode, int gameAction)
  {
   switch (gameAction)
   {
   case Canvas.UP:
    lum -= ((double) jump / rgbH);
    if (lum < 0)
     lum = 0;
    break;
   case Canvas.DOWN:
    lum += ((double) jump / 255);
    if (lum > 1)
     lum = 1;
    break;
   case Canvas.LEFT:
   case Canvas.FIRE:
    setFocused(false);
    break;
   }
   pointerY = (int) (rgbY + rgbH * lum) - pointerImg.getHeight() / 2;
   parent.updateLuminance(1 - lum);
  }

  public void keyRepeated(int keycode, int gameAction)
  {
   if (jump < 8)
    jump *= 2;
   keyPressed(keycode, gameAction);

  }

  public void pointerPressed(int x, int y)
  {
   if (y >= rgbY + rgbH)
    y = rgbY + rgbH - 1;
   lum = (double) y / rgbH;
   pointerY = (int) (rgbY + (lum * rgbH) - pointerImg.getHeight() / 2);
   parent.updateLuminance(1 - lum);
  }

  public void pointerDragged(int x, int y)
  {
   pointerPressed(x, y);
  }

  public void pointerReleased(int x, int y)
  {

  }

  public void setFocused(boolean focused)
  {
   isFocused = focused;
  }

  public boolean isFocused()
  {
   return isFocused;
  }
 }
```

You can see that I'm implementing some of my internal interfaces and extending other class (those are just simple x,y,w,h basic class and `UserInteraction` is there to make sure i implement both Pointers and Keys methods) the look of the luminance is changed when the Hue & Saturation values change (this is just for the effect to the user to understand what the actual color will look like in the selected luminance) and in my `public void paint(Graphics g)` method is where I draw the actual luminance components which is mainly the class `int[] rgb` array.

The second palette class is:

```
package com.future.rgbselector.ui.controls;

import java.io.IOException;

import javax.microedition.lcdui.Canvas;
```


(C) Copyright Nokia 2013. All rights reserved.

```
import javax.microedition.lcdui.Graphics;
import javax.microedition.lcdui.Image;

import com.future.rgbselector.PaletteScreen;
import com.future.utils.JarDynamicResourceLoader;
import com.future.utils.UserInputListener;
import com.future.utils.graphics.GFX;
import com.future.utils.graphics.PaintableArea;
import com.future.utils.ui.controls.Dimensionable;

public class ColorPalette extends Dimensionable implements PaintableArea, UserInputListener {

 private PaletteScreen parent;
 private int cursorX, cursorY;
 private int[] rgb;
 private boolean isFocused, paintFocus;
 private Image cursorImg;
 int rgbHeight;
 int rgbWidth;
 private int jump;
 private double fixedLum;
 private int noSatFixedR, noSatFixedG, noSatFixedB;

 private static final int BORDER_WIDTH = 1;

 public ColorPalette(PaletteScreen parent)
 {
  this.parent = parent;
  jump = 1;
  try
  {
   cursorImg = JarDynamicResourceLoader.getImage("cursor.png");
  } catch (IOException e)
  {
   e.printStackTrace();
  }
  fixedLum = PaletteScreen.INIT_LUM;
  noSatFixedR = (int) (fixedLum * 255);
  noSatFixedG = (int) (fixedLum * 255);
  noSatFixedB = (int) (fixedLum * 255);
 }

 public void paint(Graphics g)
 {
  if (paintFocus)
  {
   int orgColor = g.getColor();
   if (isFocused)
    g.setColor(0xFFFF0000);
   else
    g.setColor(0xFFF0F0F0);
   g.drawRect(x, y, width - 1, height - 1);
   g.setColor(orgColor);
  }
  g.drawRGB(rgb, 0, rgbWidth, x + BORDER_WIDTH, y + BORDER_WIDTH, rgbWidth, rgbHeight, false);
  if (cursorImg != null)
   g.drawImage(cursorImg, cursorX - cursorImg.getWidth() / 2, cursorY - cursorImg.getHeight() / 2,
     Graphics.TOP | Graphics.LEFT);
 }

 public void setPaintFocus(boolean paint)
 {
  paintFocus = paint;
 }

 private static int roundInt(double a)
 {
  return (int) (a+0.5);
 }

 public int HSL_TO_RGB(double H, double S, double a, double b)
 {
  int R, G, B;
  if (S == 0) // HSL from 0 to 1
  {
   R = noSatFixedR; // RGB results from 0 to 255
   G = noSatFixedG;
   B = noSatFixedB;
  } else
  {

   R = roundInt(255 * Hue_2_RGB(a, b, H + (1 / 3d)));
   G = roundInt(255 * Hue_2_RGB(a, b, H));
   B = roundInt(255 * Hue_2_RGB(a, b, H - (1 / 3d)));
  }
  return 0xFF000000 | (R << 16) | (G << 8) | B;
 }


                     public static double Hue_2_RGB(double v1, double v2, double vH) // Function Hue_2_RGB
 {
  if (vH < 0)
   vH += 1;
  if (vH > 1)
   vH -= 1;
  if ((6 * vH) < 1)
   return (v1 + (v2 - v1) * 6 * vH);
  if ((2 * vH) < 1)
   return (v2);
  if ((3 * vH) < 2)
   return (v1 + (v2 - v1) * ((2 / 3d) - vH) * 6);
  return (v1);
 }
 public void setDimension(int _width, int _height)
 {
  super.setDimension(_width, _height);
  rgbHeight = this.height - BORDER_WIDTH * 2;
  rgbWidth = this.width - BORDER_WIDTH * 2;
  rgb = new int[rgbHeight * rgbWidth];
  int yIndexArray;
  double a, b;
  double[] tempH = new double[rgbWidth];
  for (int _x = 0; _x < rgbWidth; ++_x)
   tempH[_x] = (double) _x / rgbWidth;
  double[] tempS = new double[rgbHeight];
  for (int _y = 0; _y < rgbHeight; ++_y)
   tempS[_y] = 1 - (double) _y / rgbHeight;
  for (int _y = 0; _y < rgbHeight; ++_y)
  {
   yIndexArray = _y * rgbWidth;
```


(C) Copyright Nokia 2013. All rights reserved.

```
    if (fixedLum < 0.5)
     b = fixedLum * (1 + tempS[_y]);
    else
     b = (fixedLum + tempS[_y]) - (tempS[_y] * fixedLum);

    a = 2 * fixedLum - b;
    for (int _x = 0; _x < rgbWidth; ++_x)
    {
     rgb[yIndexArray + _x] = HSL_TO_RGB(tempH[_x], tempS[_y], a, b);
    }
   }
   pointerPressed(x + cursorImg.getWidth() / 2, y + cursorImg.getHeight() / 2);
  }

  public void pointerPressed(int x, int y)
  {
   if (y > this.y + height - BORDER_WIDTH)
    y -= BORDER_WIDTH;
   else if (y < this.y + BORDER_WIDTH)
    y += BORDER_WIDTH;
   if (x > this.x + width - BORDER_WIDTH)
    x -= BORDER_WIDTH;
   else if (x < this.x + BORDER_WIDTH)
    x += BORDER_WIDTH;
   cursorX = x;
   cursorY = y;
   double hue = (cursorX - (this.x + BORDER_WIDTH)) / (double) rgbWidth;
   double sat = (1 - (cursorY - (this.y + BORDER_WIDTH)) / (double) rgbHeight);
   parent.updateHueSat(hue, sat);
  }

  public void pointerDragged(int x, int y)
  {
   pointerPressed(x, y);
  }

  public void pointerReleased(int x, int y)
  {

  }

  public void keyReleased(int keycode, int gameAction)
  {
   jump = 1;
  }

  public void keyPressed(int keycode, int gameAction)
  {
   switch (gameAction)
   {
   case Canvas.UP:
    cursorY -= jump;
    if (cursorY < y + BORDER_WIDTH)
     cursorY = y + BORDER_WIDTH;
    break;
   case Canvas.DOWN:
    cursorY += jump;
    if (cursorY > y + height - BORDER_WIDTH)
     cursorY = y + height - BORDER_WIDTH;
    break;
   case Canvas.LEFT:
    cursorX -= jump;
    if (cursorX < x + BORDER_WIDTH)
     cursorX = x + BORDER_WIDTH;
    break;
   case Canvas.RIGHT:
    cursorX += jump;
    if (cursorX > x + width - BORDER_WIDTH)
    {
     cursorX = x + width - BORDER_WIDTH;
     setFocused(false);
    }
    break;
   case Canvas.FIRE:
    setFocused(false);
    break;
   }
   double hue = (cursorX - (x + BORDER_WIDTH)) / (double) rgbWidth;
   double sat = (1 - (cursorY - (y + BORDER_WIDTH)) / (double) rgbHeight);
   parent.updateHueSat(hue, sat);
  }

  public void keyRepeated(int keycode, int gameAction)
  {
   if (jump < 8)
    jump *= 2;
   keyPressed(keycode, gameAction);
  }

  public void setFocused(boolean focused)
  {
   isFocused = focused;
  }

  public boolean isFocused()
  {
   return isFocused;
  }
 }
```

Since the graphics object allows us to draw RGB arrays we need to convert our H,S,L values to an RGB format to store in our array and that is some of the math you might see in these classes.

Similary the user might be interested in detecting the color value of a pretty flower while he is outside so I added the option to also let the user pick a pixel color value from an image (either from phone gallery or from the camera snapshot)

For that i need to add a gallery like screen to show thumbs of all the phone gallery images, this is the third screen, I wont go into very many details here but basically you need to check the phone gallery image folders this can be detected using the following `system.getProperty("fileconn.dir.memorycard")`, `System.getProperty("fileconn.dir.photos")` & `System.getProperty("fileconn.dir.photos.name")` then you just go over all the files in the gallery folder and create a thumbnail version of the image (since Java ME doesnt have a image resize function) you need to write your own image resizing method or use the following.

(C) Copyright Nokia 2013. All rights reserved.

```
public Image resize(Image img, int a, int b) // a - destination width, b - destination height
    {
        if(a == img.getWidth() && b == img.getHeight())
            return img;
        if(a <= 0 || b <= 0)
            return img;
        int ai[] = new int[img.getWidth()*img.getHeight()];
 img.getRGB(ai, 0, img.getWidth(), 0, 0, img.getWidth(), img.getHeight());
        int ai1[] = new int[a*b];
        int i = a;
        int j = img.getWidth();
        int m;
        int k1 = (int)(((double)(m = img.getHeight()) / (double)b) * 1024D);
        int l1 = (int)(((double)j / (double)a) * 1024D);
        int i2 = 0 - i;
        for(int k = 0; k < b; k++)
        {
            int l = (k1 * k >> 10) * j;
            i2 += i;
            int j1 = 0 - l1;
            for(m = 0; m < a; m++)
            {
                int i1 = (j1 += l1) >> 10;
                ai1[i2 + m] = ai[l + i1];
            }

        }

        return Image.createRGBImage(ai1, a, b, true);
    }
```

In the fourth image you can see the screen after selecing an image from gallery or from camera (it leads to the same screen), as you can see I've added zooming option to add percision to the pixel selection which is done with scaling a specific section of the image with the above function.

You can move the image around with joystick or either by pressing its edges to move in that direction or by flicking it around with Nokia's `com.nokia.mid.ui.gestures.GestureListener` class and `FrameAnimatorListener` to indicate how the image should be moved.

```
package com.future.rgbselector;

import javax.microedition.lcdui.game.GameCanvas;

public class GestureAnimatorListener implements com.nokia.mid.ui.frameanimator.FrameAnimatorListener, com.nokia.mid.ui.gestures.Gest
 private com.nokia.mid.ui.frameanimator.FrameAnimator animator;
 private CameraScreen screen;
 private GameCanvas canvas;

 public GestureAnimatorListener(CameraScreen screen,GameCanvas canvas)
 {
  com.nokia.mid.ui.gestures.GestureRegistrationManager.setListener(canvas, this);
  animator = new com.nokia.mid.ui.frameanimator.FrameAnimator();
  this.canvas = canvas;
  this.screen = screen;
 }

 public void deRegister()
 {
  com.nokia.mid.ui.gestures.GestureRegistrationManager.unregisterAll(canvas);
 }

 public void updatePosDimen(int x, int y, int width, int height)
 {
  com.nokia.mid.ui.gestures.GestureInteractiveZone gestureZone = new com.nokia.mid.ui.gestures.GestureInteractiveZone(com.nokia.mid.
  gestureZone.setRectangle(x, y, width, height);
  com.nokia.mid.ui.gestures.GestureRegistrationManager.unregisterAll(canvas);
  com.nokia.mid.ui.gestures.GestureRegistrationManager.register(canvas, gestureZone);
 }

 public void animate(com.nokia.mid.ui.frameanimator.FrameAnimator animator, int x, int y, short delta, short deltaX, short deltaY,
   boolean lastFrame)
 {
  screen.animate(x,y,delta,deltaX,deltaY,lastFrame);

 }

 public void gestureAction(Object container, com.nokia.mid.ui.gestures.GestureInteractiveZone gestureInteractiveZone,
   com.nokia.mid.ui.gestures.GestureEvent gestureEvent)
 {
  switch (gestureEvent.getType())
  {
  case com.nokia.mid.ui.gestures.GestureInteractiveZone.GESTURE_TAP:
   break;
  case com.nokia.mid.ui.gestures.GestureInteractiveZone.GESTURE_LONG_PRESS:
   break;
  case com.nokia.mid.ui.gestures.GestureInteractiveZone.GESTURE_LONG_PRESS_REPEATED:
   break;
  case com.nokia.mid.ui.gestures.GestureInteractiveZone.GESTURE_DRAG:
   break;
  case com.nokia.mid.ui.gestures.GestureInteractiveZone.GESTURE_DROP:
   break;
  case com.nokia.mid.ui.gestures.GestureInteractiveZone.GESTURE_FLICK:
   if (animator.isRegistered())
    animator.unregister();
   if (animator.register(gestureEvent.getStartX(), gestureEvent.getStartY(), (short)15, (short)50, this))
    animator.kineticScroll(gestureEvent.getFlickSpeed(), com.nokia.mid.ui.frameanimator.FrameAnimator.FRAME_ANIMATOR_FREE_ANGLE,
     com.nokia.mid.ui.frameanimator.FrameAnimator.FRAME_ANIMATOR_FRICTION_LOW, gestureEvent.getFlickDirection());
   break;
  }
 }
}
```

This is the class I use to register and listen to gesture as well as passing the needed animation flicking movment to my own class (`CameraClass` in this code).

At the bottom of the screen you can see all the different pixel RGB, HSL etc format as well as a larger square section with that exact color.

Summary

(C) Copyright Nokia 2013. All rights reserved.

I hope this article will help you in creating your future app.

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.