library lw\_color\_picker;

import 'dart:math';

import 'package:flutter/material.dart';

/\*

This file was inspired by fuyumi's flutter color picker @ https://github.com/mchome/flutter\_colorpicker

\*/

class LWColorPicker extends StatefulWidget{

LWColorPicker({

this.callback,

this.hasKeyboard,

this.width: 300.0,

this.heightToWidthRatio: .69,

}

);

final ValueChanged<Color> callback;

final ValueChanged<bool> hasKeyboard;

final double width;

final double heightToWidthRatio;

@override

State<StatefulWidget> createState() => LWColorPickerState();

}

class LWColorPickerState extends State<LWColorPicker> {

FocusNode \_focusR = new FocusNode();

FocusNode \_focusG = new FocusNode();

FocusNode \_focusB = new FocusNode();

double hue;

double saturation;

double value;

Color getCurrentColor(){

return HSVColor.fromAHSV(1.0, hue, saturation, value).toColor();

}

void \_callback(){

if(widget.callback != null){

widget.callback(getCurrentColor());

}

}

void setCurrentColor(Color color){

HSVColor next = HSVColor.fromColor(color);

hue = next.hue;

saturation = next.saturation;

value = next.value;

}

int \_validateColorInput(String input){

bool stringError = false;

switch(input.toUpperCase()){

case "F":

case "FUL":

return 255;

case "Z":

case "ZRO":

case "B":

case "BLK":

case "OUT":

return 0;

case "DON":

return 69;

default:

break;

}

int val = int.parse(

input,

onError: (error){

return 0;

});

if(stringError){

return 0;

}

if(val > 255){

val = 255;

} else if(val < 0){

val = 0;

}

return val;

}

void \_focusChanged(){

bool retVal = false;

if(\_focusR.hasFocus){

retVal = true;

}

if(\_focusG.hasFocus){

retVal = true;

}

if(\_focusB.hasFocus){

retVal = true;

}

if(widget.hasKeyboard != null){

widget.hasKeyboard(retVal);

}

}

@override

initState() {

super.initState();

HSVColor color = HSVColor.fromColor(Colors.purple);

\_focusR.addListener(\_focusChanged);

\_focusG.addListener(\_focusChanged);

\_focusB.addListener(\_focusChanged);

hue = color.hue;

saturation = color.saturation;

value = 1.0;

}

@override

Widget build(BuildContext context) {

double width = widget.width;

double height = width \* widget.heightToWidthRatio;

TextEditingController \_controllerR = new TextEditingController();

TextEditingController \_controllerG = new TextEditingController();

TextEditingController \_controllerB = new TextEditingController();

return Column(

children: <Widget>[

Container(

width: width,

height: height,

child: GestureDetector(

onPanStart: (DragStartDetails details){

},

onPanUpdate: (DragUpdateDetails details){

RenderBox box = context.findRenderObject();

Offset localOffset = box.globalToLocal(details.globalPosition);

setState(() {

this.hue = ((localOffset.dx.clamp(0.0, width) / width) \* 300.0);

this.saturation = 1 - localOffset.dy.clamp(0.0, height) / height;

\_callback();

});

},

child: CustomPaint(

size: Size(width, height),

painter: ColorPainter(

hue: this.hue,

saturation: this.saturation,

value: this.value

),

),

),

),

Container(

width: width,

child: Slider(

onChanged: (val){

setState(() {

this.value = val;

\_callback();

});

},

value: this.value,

),

),

Container(

width: width,

child: Row(

children: <Widget>[

Expanded(

child: Text

(

"R",

textAlign: TextAlign.center,

),

),

Expanded(

child: Text

(

"G",

textAlign: TextAlign.center,

),

),

Expanded(

child: Text

(

"B",

textAlign: TextAlign.center,

),

),

],

)

),

Container(

width: width,

child: Row(

children: <Widget>[

Expanded(

child: TextField(

controller: \_controllerR,

focusNode: \_focusR,

decoration: InputDecoration(

border: InputBorder.none,

hintText: getCurrentColor().red.toString()

),

keyboardType: TextInputType.number,

textAlign: TextAlign.center,

onChanged: (val){

if(val.length >= 3){

setState(() {

setCurrentColor(getCurrentColor().withRed(\_validateColorInput(val)));

\_callback();

\_controllerR.clear();

});

}

},

onSubmitted: (val){

setState(() {

setCurrentColor(getCurrentColor().withRed(\_validateColorInput(val)));

\_callback();

\_controllerR.clear();

});

},

),

),

Expanded(

child: TextField(

controller: \_controllerG,

focusNode: \_focusG,

decoration: InputDecoration(

border: InputBorder.none,

hintText: getCurrentColor().green.toString()

),

keyboardType: TextInputType.number,

textAlign: TextAlign.center,

onChanged: (val){

if(val.length >= 3){

setState(() {

setCurrentColor(getCurrentColor().withGreen(\_validateColorInput(val)));

\_callback();

\_controllerG.clear();

});

}

},

onSubmitted: (val){

setState(() {

setCurrentColor(getCurrentColor().withGreen(\_validateColorInput(val)));

\_callback();

\_controllerG.clear();

});

},

),

),

Expanded(

child: TextField(

controller: \_controllerB,

focusNode: \_focusB,

decoration: InputDecoration(

border: InputBorder.none,

hintText: getCurrentColor().blue.toString(),

),

keyboardType: TextInputType.number,

textAlign: TextAlign.center,

onChanged: (val){

if(val.length >= 3){

setState(() {

setCurrentColor(getCurrentColor().withBlue(\_validateColorInput(val)));

\_callback();

\_controllerB.clear();

});

}

},

onSubmitted: (val){

setState(() {

setCurrentColor(getCurrentColor().withBlue(\_validateColorInput(val)));

\_callback();

\_controllerB.clear();

});

},

),

)

],

),

)

],

);

}

}

class ColorPainter extends CustomPainter {

ColorPainter({

this.hue: 0.0,

this.saturation: 1.0,

this.value: 1.0,

});

double hue;

double saturation;

double value;

@override

paint(Canvas canvas, Size size) {

Rect rect = Offset.zero & size;

Gradient gradientBW = LinearGradient(

begin: Alignment.topCenter,

end: Alignment.bottomCenter,

colors: [

Color.fromARGB(255, 0, 0, 0),

HSVColor.fromAHSV(1.0, 0.0, 0.0, value).toColor(),

],

);

Gradient gradientColor = LinearGradient(

colors: [

HSVColor.fromAHSV(1.0, 0.0, 1.0, value).toColor(),

HSVColor.fromAHSV(1.0, 45.0, 1.0, value).toColor(),

HSVColor.fromAHSV(1.0, 90.0, 1.0, value).toColor(),

HSVColor.fromAHSV(1.0, 135.0, 1.0, value).toColor(),

HSVColor.fromAHSV(1.0, 180.0, 1.0, value).toColor(),

HSVColor.fromAHSV(1.0, 225.0, 1.0, value).toColor(),

HSVColor.fromAHSV(1.0, 270.0, 1.0, value).toColor(),

HSVColor.fromAHSV(1.0, 315.0, 1.0, value).toColor(),

],

);

canvas.drawRect(

rect,

Paint()

..shader = gradientColor.createShader(rect));

canvas.drawRect(

rect,

Paint()

..shader = gradientBW.createShader(rect)

..blendMode = BlendMode.lighten);

canvas.drawCircle(

Offset(size.width \* (hue / 300.0), size.height \* (1 - saturation)),

size.height \* 0.04,

Paint()

..color = HSVColor.fromAHSV(1.0, hue, saturation, (1-pow(value, 0.3))).toColor()

..strokeWidth = 3.0

..style = PaintingStyle.stroke);

}

@override

bool shouldRepaint(CustomPainter oldDelegate) => true;

}