二次元コードの機能拡充に関する研究

A study on the Advanced Two-Dimensional Code

サムレットウィット ダムリ

Damri Samretwit

平成 26 年 3 月 6 日

福岡工業大学大学院工学研究科　情報通信工学専攻

Fukuoka Institute of Technology,　Graduate School of Information and Communication Engineering

現在，インターネットや携帯電話の普及に伴い QR コードが世界中に広まっており，Web サービスのアクセスなど各種用途に QR コードが使われている．　二次元コードの代表である QR コードは白黒 2 値のドット表示であるためデザイン性が良くないので，ドットをカラー化して見栄えを良くする方法が用いられる．この時，ドットの色を変更したり見栄えを良くするためにイメージを重畳したりするとイメージ損傷になり QR コードの読み取り特性に影響を及ぼす．また，紙に印刷した QR コードやディスプレイに表示した QR コードの場合には，携帯電話やスマートフォンなどのカメラで撮影する際の照明条件および表示条件によって，QR コードの読み取り特性が異なる．

本論文では，通常の QR コードの機能拡充として静止画などのイメージを重畳し白黒 2 値からカラー化二次元コードを作成するカラー化 QR エディタを構成するとともに，QR リーダとして現状のスマートフォンなどに用いられている QR デコーダをそのまま用いた場合に，カラー化する際のパラメータによってどの程度正常に認識できるのか実験により読み取り特性を評価する．

1. QR コードの概要

図 1．1 QR コード

QR コード[1] （QR コード® は，株式会社デンソーウェーブの登録商標である）は二次元コード[2]の一種であり，Quick Response コードの略である．その名前の通り，高速で読み取れることを目的にデンソーウェーブ（開発当時は株式会社デンソーの一部門）が開発したものである．QR コードは，縦，横二次元方向に白と

黒の明暗パターン情報を持つことで，一次元のバーコードよりも記録できる情報量を飛躍的に増加させたコードである．

2. QR コードのカラー化の基本ルール

カラー化ドット絵付二次元コードエディタの構成[3]にあたり，以下の２つのルールを考慮して QR コードのカラー化処理を行う．

2.1 ドット絵を挿入する領域に関するルール

QR コードでは，形式情報及び型番情報を除くシンボルの符号化領域は，データを符号化したときの 0 と 1 の組み合わせで右下部から各ドットの上方向に配置され上端に達すると 1 ドット左にずれて下方向にデータが配置され，これを順次繰り返して絵柄が決定される．型番を大きめに選ぶと冗長性が増加し，実データの入っていない冗長部が左側の誤り訂正コード部を除いた中央部に埋め草コード部の領域となる．この埋め草コード部は，埋め込むデータを保持したまま自由に操作することが可能である．

2.2“暗”ドットと”明”ドットの識別ルール

萩原の文献[7]によると，「符号化と復号では要となる処理に差異がある」であり，これによれば，「復号器は即応符号を構成するドットとして“暗い色“は“暗“ドットと認識され，“明るい色“は“明“ドットに認識される」．従って，各ドットのカラーの輝度を算出し，輝度が基準値以下であれば“暗“ドット，基準値以上であれば“明“と認識される．

3．カラーQREditor の概要

3．1 カラー二次元コードの作成

通常の QR コードをカラー化（RGB）するにあたって，そのカラーの輝度値 Y をグレー変換により算出し，各ドットの値が 90%以上の場合には“明”と判定し，50%以下の場合には“暗”と判定することとしている．ただし，輝度値の値がこれらの間の 50～90％の輝度値の場合は読み取りに時間がかかったり，“明”と“暗”の読み取りエラーが生ずる可能性がある．

輝度値 Y としては，カラーの RGB 値からグレー変換して下記の(1)式から算出する．

$$Y = 0.299 \times R + 0.587 \times G + 0.114 \times B \qquad (1)$$

$$0 < R,\ G,\ B < 255$$

以下では，各色の色相(H:Hue)，彩度(S:Saturation)および明度(B:Brightness)の値によって上記 50％や 90%の値がどのように変化するのか，また色相を連続的に変化させた時，輝度値が 50％以下の時，90％以上の時，どのような読み取り特性になるのか実験的に求め評価する．

3．2 カラーQREditor の機能追加

カラーQR エディタのユーザインタフェースの基本部はほぼ白黒 QREditor[8]のままで，カラー化のボタンを追加してドットの描き方を編集できるようにした．このプログラムの特徴は，各ドットの色を個別にカラー化でき，最初に現れたエディタの画面では白黒のドットを設定している．図 3.1 に，カラー化のドットを描いて作成したカラードット絵二次元コードエディタを示す．“画”のボタンをクリックするとパソコン内に保存している画像を選択し，画像を組み込むことができる．キーボードの矢印（↑↓→←）により読み出した画像を自由的に移動することができる．また，“大”は画像を大きくするボタンであり，“小”は画像を小さくする

ボタンであり，拡大・縮小の編集処理を行う．さらに，画像のアルファチャンネルの値の透明度を変化するリストにより 0%から 100%を選択し，画像を透明化することができる．

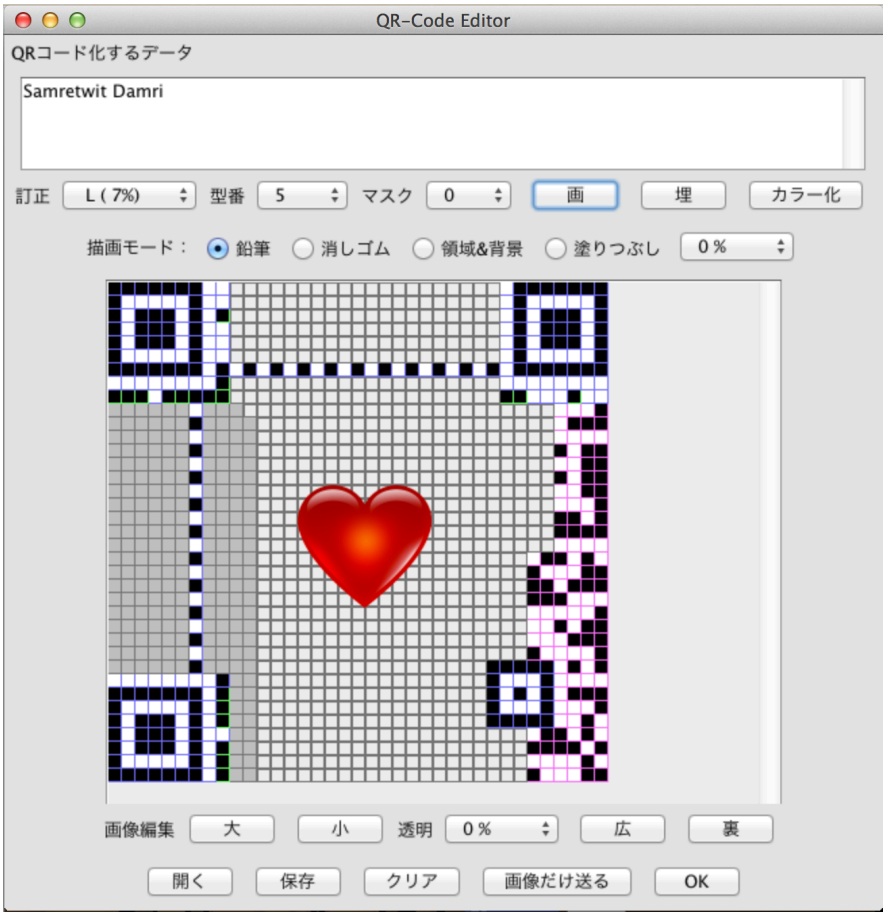

図 3.1 カラーQREditor のドット編集の画面

3．3 カラーバー

カラーバーは，選択できる色を RGB 値や HSB 値により設定する．画面の下では選ばれた色の RGB 値を表示している．さらにその選んだ色はどのくらいの明るさかを示している．

カラー調整を RGB のバーで自由に設定できるが，輝度値の値によっては “明”か“暗”の設定を判断できない色もあるので，確実に選択できる色として設定された輝度値の以下の輝度値は“暗”（式（１）で 50%以下），設定された輝度値以上の輝度値は“明”と設定する（式（１）で 90%以上）こととした．これらの中間の値（50-90%）では携帯電話やスマートフォンに内蔵している QR デコーダのアルゴリズムにより“明”または“暗”の判定が難しい場合もある．図 3．2 に輝度値を設定するためのリストを表示している．図 3.3 にアルファチャネルの透明度と色を識別するための色の輝度値の値（最大 100%）を表示している．

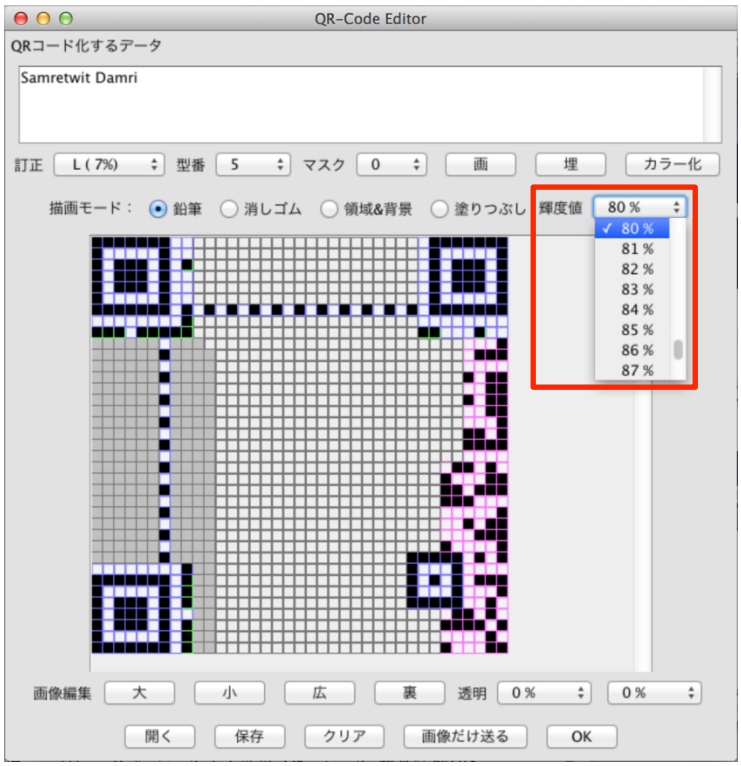

図 3.2 輝度値の設定のためのリスト

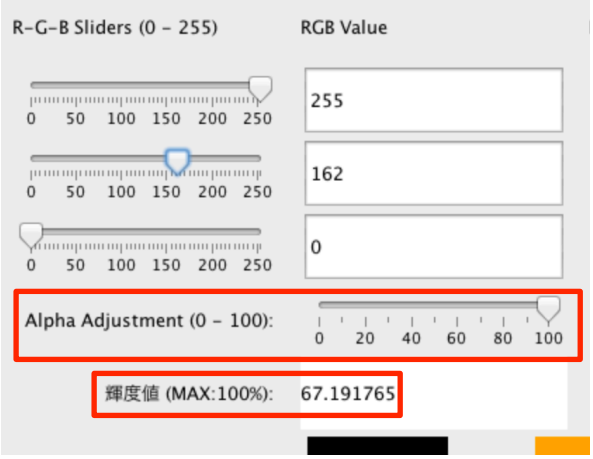

図 3.3 色の輝度値の表示

４．実験とその結果

4．1 実験環境

1　表示装置

実験時に利用した，表示装置は以下の通りである．

1)　MacBook pro(13 インチ)

2　実験期間

実験を行った期間は，以下の通りである．

1)　2013 年 7 月～2014 年 1 月

3　環境条件

実験時に利用した，環境条件である．

1) 通常照明(蛍光灯)

4 環境モニタリング条件

実験時に利用した，環境モニタリング条件である．

1) 80%輝度

5 QR 読み取り装置（QR リーダ）

実験時に利用した，QR 読み取り装置である．

1) iPhone 5

6 画像編集プログラム

実験時に利用した画像編集プログラムである．

1) Gimp 2．8．0

7 読み取りプログラム（QR デコーダ）

実験時に利用したカラー二次元コードを読み取るプログラムである．

1) i-nigma[4]

2) QRReader for iPhone[5]

3) Best Barcode Scanner[6]

8 符号化データ

実験を行った際の符号化データである．

1) 英字 15 文字

9 型番(バージョン)

実験を行った際の型番(バージョン)である．

1) 5

10 誤り訂正レベル

実験を行った際の誤り訂正レベルである．

1) L，M，Q，H のうちの 1 レベル

本研究では，二次元コードを開発するため，各種カラー化評価実験を行って誤り訂正能力を確認するとともに色の 3 属性を用いドットの色の認識特性を評価した．その得られた結果を利用して QREditor のソフトを改造してカラー化し，カラーQREditor を開発した．その実験の内容を以下に説明する．

4．2 カラー二次元コードの読み取り特性

4．2．1 実験条件

QR コードのカラー化に当たっては，カラー二次元コードエディタ [2] を用いる．作成した二次元コードに 15 文字のアルファベットと記号からなる URL を入力し，8 ビットバイトモードで二次元コードを作成する．バージョンは 5（37X37 モジュール）とし，誤り訂正レベルは L とした．また，QR リーダとしては，iPhone4 のスマートフォンに，i-nigma，QRReader for iPhone および Best Barcode Scanner の 3 種類を用いた．作成した二次元コードを MacBook Pro に表示し，通常の蛍光灯下で読み取り実験を行った．

4．2．2 実験結果

（1） 赤，青，緑に対する輝度値の変化特性

赤(R)，緑(G)，青(B)のそれぞれの色に対して，読み取りの限界値である 50%の輝度値および 90%の輝度値がどのように変化するかを求めた．それぞれの特性を図 4．1，図 4．2，図 4．3 図 4．4，図 4．5，図 4．6 に示す．

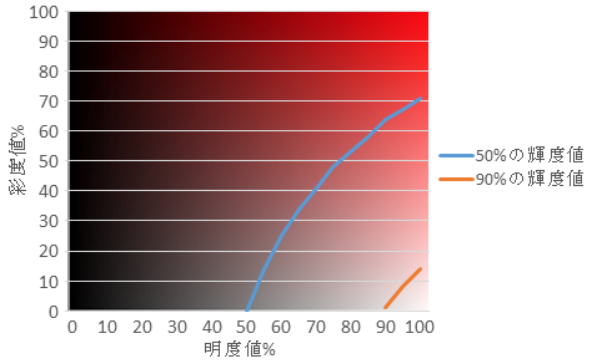

図 4．1　赤色に対する輝度値の変化

図 4．1 は，赤色に対して彩度値を 0~100%変化させた場合の 50%および 90%の輝度値の変化を求めたものである．彩度値が 0%の時は確かにそれぞれ 50%，90%であるが，彩度値が 0%から大きくなると明度値も少しずつ値が大きくなる．

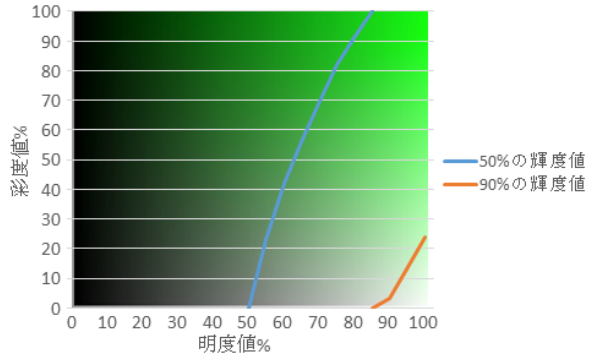

図 4．2　緑色に対する輝度値の変化

図 4．2 は，緑色に対して彩度値を 0~100%変化させた場合の 50%および 90%の輝度値の変化を求めたものである．この場合も彩度値が 0%の時は確かにそれぞれ 50%，90%であるが，彩度値が 0%から大きくなると明度値も少しずつ値が大きくなり，50%の明度値の曲線は彩度値が 100%の時には 85%程度に大きくなり，90%の明度値の曲線は彩度値が 20%程度で 100%になる．

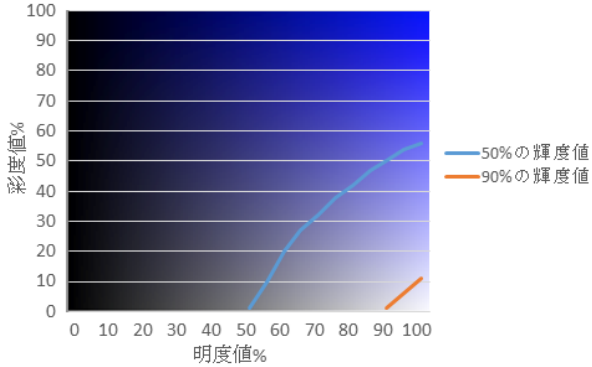

図 4．3　青色に対する輝度値の変化

図 4．3 は，青色に対して彩度値を 0~100%変化させた場合の 50%および 90%の輝度値の変化を求めたものである．この場合も彩度値が 0%の時は確かにそれぞれ 50%，90%であるが，彩度値が 0%から大きくなると明度値の値が大きくなり，彩度値が 50%の際に明度値は 100%となり変化は大きい．また，90%輝度値の場合には彩度値が 10%程度で明度値が 100%となる．

（2）黄，シアン，マゼンタに対する輝度値の変化特性

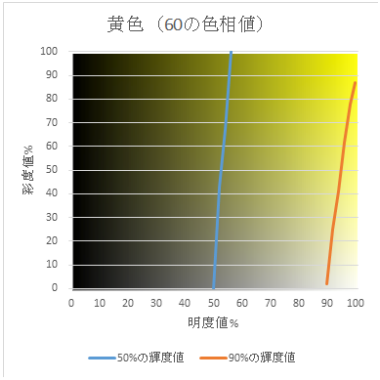

図 4．4　黄色に対する輝度値の変化

図 4．4 は，黄色に対して彩度値を 0~100%変化させた場合の 50%および 90%の輝度値の変化を求めたものである．この場合も彩度値が 0%の時は確かにそれぞれ 50%，90%であるが，彩度値が 0%から大きくなると明度値の値が大きくなり，彩度値が 100%の際に明度値は 55%となり変化は大きい．また，90%輝度値の場合には彩度値が 90%程度で明度値が 100%となる．

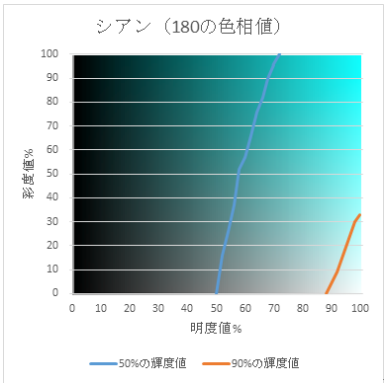

図 4．5　シアンに対する輝度値の変化

図 4．5 は，シアンに対して彩度値を 0~100%変化させた場合の 50%および 90%の輝度値の変化を求めたものである．この場合も彩度値が 0%の時は確かにそれぞれ 50%，90%であるが，彩度値が 0%から大きくなると明度値の値が大きくなり，彩度値が 100%の際に明度値は 70%となり変化は大きい．また，90%輝度値の場合には彩度値が 33%程度で明度値が 100%となる．

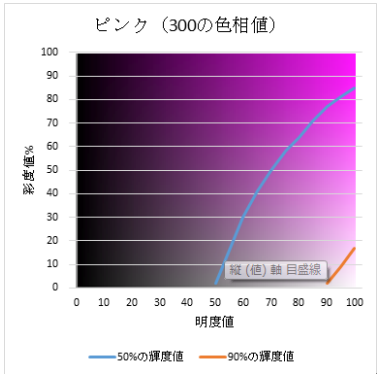

図 4．6　マゼンタに対する輝度値の変化

図 4．6 は，マゼンタ（紫）に対して彩度値を 0~100%変化させた場合の 50%および 90%の輝度値の変化を求めたものである．この場合も彩度値が 0%の時は確かにそれぞれ 50%，90%であるが，彩度値が 0%から大きくなると明度値の値が大きくなり，彩度値が 85%の際に明度値は 100%となり変化は大きい．また，90%輝度値の場合には彩度値が 15%程度で明度値が 100%となる．

（3）色相の変化に対する明度と読み取り特性

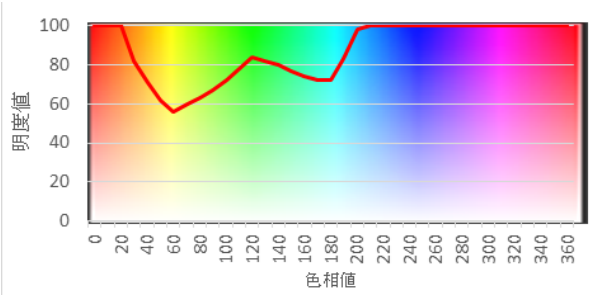

図 4．7　色相の変化に対する明度値と読み取り特性

彩度値が 100%の時の色相の変化に対する明度値とその読み取り特性を示したのが図 4 である．同図で赤は色相値は 0 および 360 であり，輝度値 50%の赤色の曲線以下では“暗”と読み取れる．色相値 60 の黄色は約 60 以下の明度値の場合，色相値 120 の緑色では約 80 以下となるが，色相値が 200 以上であればほぼ“暗”と読み取れる．

（4）　色相の変化に対する彩度と読み取り特性

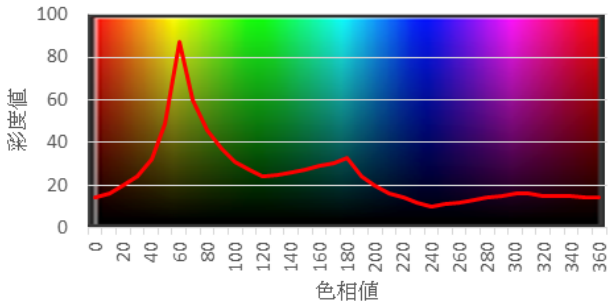

図 4．8　色相の変化に対する彩度値と読み取り特性

図 4．8 は，明度値が 100%の時の色相の変化に対する彩度値とその読み取り特性を示したものである．同図で赤色の曲線は輝度値 90%を示したものであり，これ以下のエリアでは“明”と判定される．色相値 60 の黄色は彩度値が約 80 以下では“明”と読み取れるが，他の色相では彩度値が 10 以下であれば“明”となる．

（５）　色相の変化に対する総合読み取り特性

図 4．9 にその結果を示す．この図では，彩度 100%，明度 100%の普通の輝度値の関係，彩度 50%の場合，明度 50%の場合も示す．輝度値の値が 230（約 90%）以上では“明”と判定され，128（約 50%）以下では“暗”と判定され，この間の 128～230 の間では条件により“明”または“暗”と判定された．色相の値が赤と緑の中間である黄色が 60 で輝度が最も高く，“明“と識別される割合が高かった．一方，最も暗いのは，青と緑を混ぜたシアンであり色相値 240 近辺が”暗“と識別される割合が高かった．彩度を 50%に低下させると輝度が増加し”明“と識別され，明度を 50%に低下させると”暗“と識別される割合が高くなる．この図で輝度値が 80 から 160 の間は，読み取りに時間がかかったり，条件委より”明“または”暗“と識別され読み取りエラーが発生した．

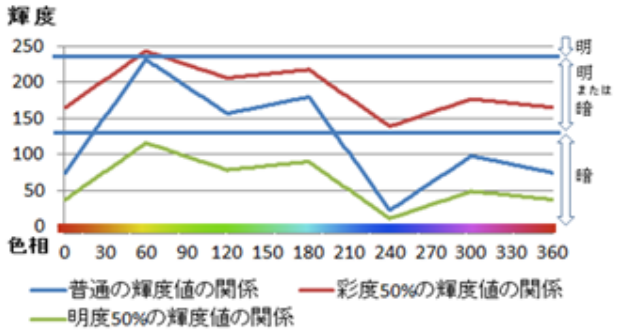

図 4．9　色相変化に対する読み取り特性

４．２．３　カラー二次元コードエディタのイメージ重畳機能

カラー二次元コードエディタの作成にあたり，以下の条件を考慮して構成する．

（１）機能パターン部のカラー化機能

二次元コードエディタで試作したカラー化ドット絵付二次元コードエディタのプロトタイプでは，データコード語と誤り訂正コード語の間の埋め草コード部（冗長部分）のドットに絵を描き，その各ドットのカラーを変化させる方法であり，位置検出パターンやタイミングパターンなどの機能パターン部分のカラーを変更できなかった．これを改善するため，機能パターンについてもカラーを変更できるようにした．

（２）イメージ重畳機能

データコード語と誤り訂正コード語の間の埋め草コード部を中心にイメージを重畳する．このため，

１）イメージファイル読み込み機能

２）イメージの大きさ調整機能

３）イメージ位置調整機能

４）イメージ透明度調整機能

の機能を導入した．

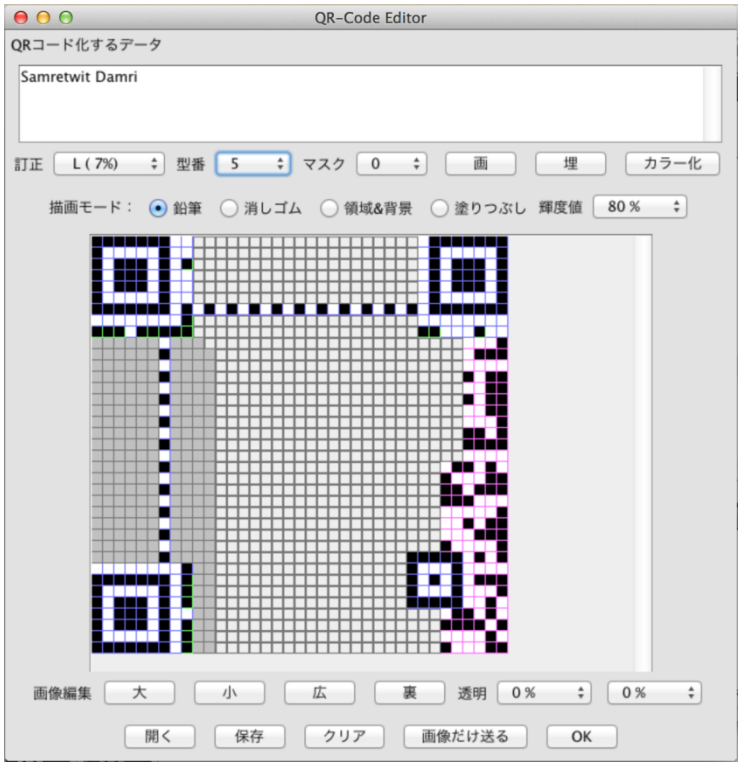

図 4．10　カラー二次元コードエディタのインタフェース

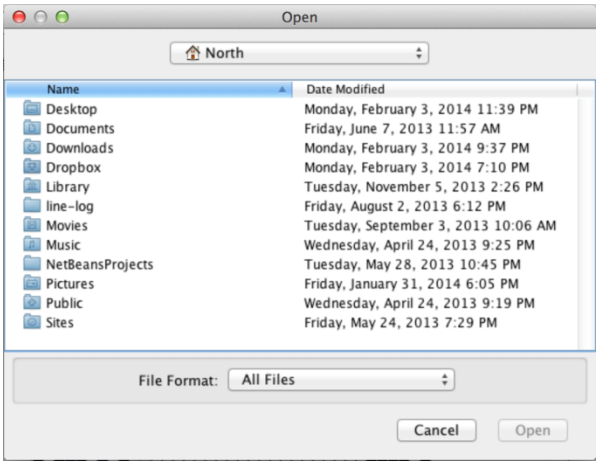

図 4．11　画像の選択画面

以上の機能を導入し，ドット単位で位置や大きさを調整できるようにした．ただし，静止画などのイメージとしては誤り訂正機能を利用すれば，その訂正能力の範囲内であればどの位置に重畳しても復号出来るので，位置および大きさは QR コードのシンボル内の任意の位置および大きさに重畳可能としている．また，重畳した静止画の透明度も調整できるようにしており，静止画の 0～100%の範囲内で調整できるようにしている．図 4．10 に，今回試作したカラー二次元コードエディタのインタフェースを示す．「画」のボタンを押すことにより，図 4．11 の選択画面が現れてユーザーのパソコン内に保存された画像を選ぶことが出来る．

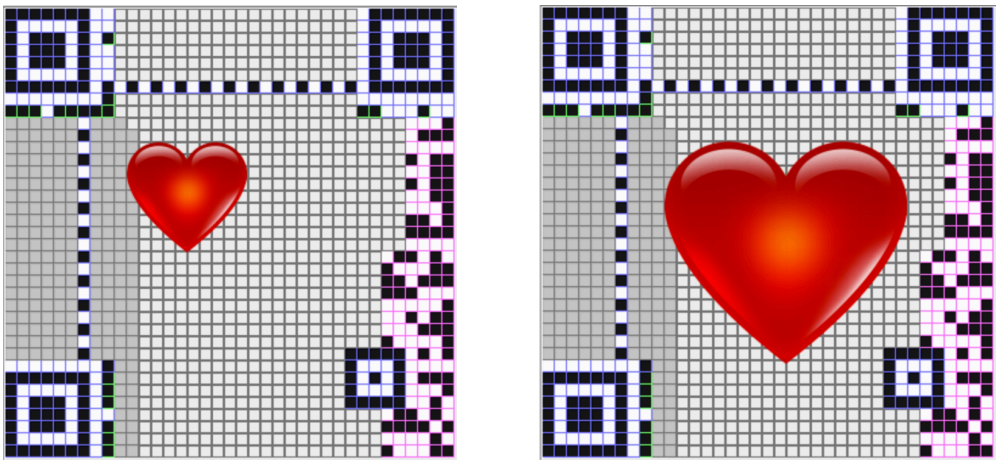

図 4．12　大きさ変更操作

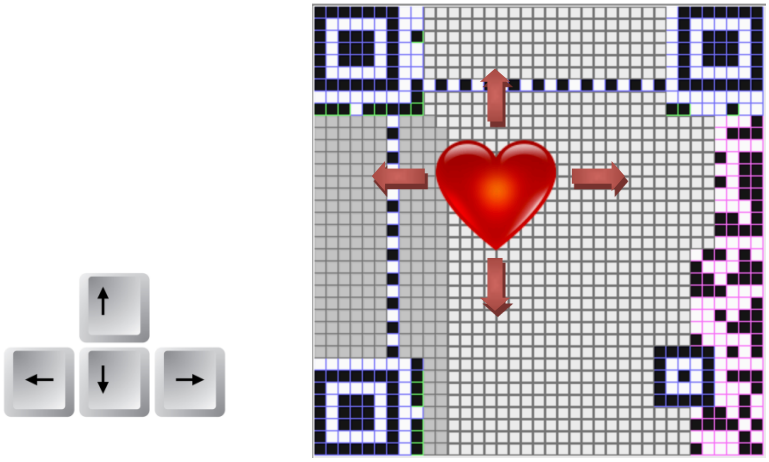

図 4．13　画像移動操作

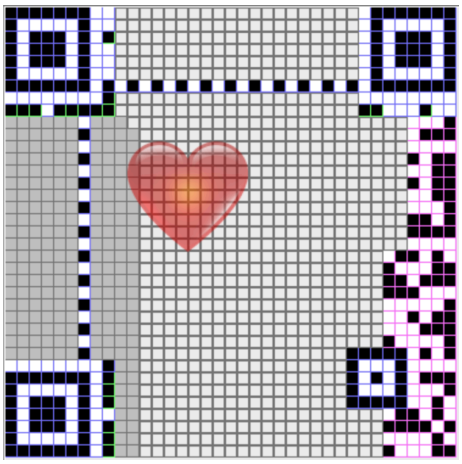

図 4．14　透明度の変更の操作

図 4．12 に同図の下側「大」や「小」などのボタンを押すことにより大きさを変更できる．図 4．13 に，位置はキーボードの 4 種類の矢印(→等)を用いることによりドット毎の調整を可能にしている．これにより，読み取りが不可能な静止画を重畳させる場合であっても，図 4．14 に示すように透明度を変更させることにより読み取りが可能になる．

5. まとめ

カラードット絵付二次元コードエディタ（カラーQREditor）のプロトタイプを引き続き開発し，二次元コードのデータが入っていない冗長エリアを利用してイメージを重畳し，色相，彩度，明度を変化させて，ドットや画像のアルファーチャンネルを変更させて QR コードの読み取り実験を行った．その結果，カラーバーの RGB 値または HSB 値を変化させ, 自由に読み取り可能なカラードット絵を描く事が可能であることも確認した．

試作したカラードット絵二次元コードエディタ（カラーQREditor）で作成した二次元コードを作成するとともにカラーイメージも重畳させることが出来, イメージを重畳したカラー二次元コードの読み取り特性を測定した．

1. カラー調整を RGB スライダーまたは HSB（H：色相，S：彩度，B：明度）スライダーで自由に設定できるようにし，輝度値の値によっては “明”か“暗”を判断できない色もあるので，確実に正しく判定できる色として 50%以下の輝度値は“暗”，90%以上の輝度値は“明”と設定する．
2. 画像のアルファーチャンネルを変更し十分な透明度を設定すれば二次元コードの上に重畳しても正しく読み取ることができることが分かった．

本システムの実験においては 3 つのデコーダだけしか用いず他のデコーダで試していないので十分ではなかったが，プロトタイプはある程度カラー絵ドットの描く事ができるようになった．簡単で使いやすく，ドットのカラーをある程度を選択でき，その選んだ色の輝度値が見せてユーザーは“明”また“暗”を識別できる．

6. 今後の課題

残された課題として，以下のものが挙げられる．

- ユーザが利用しやすいように評価システムのインタフェースを見直す．
- カラーの明暗の識別の計算を詳細化し，iPhone 以外の他の携帯の機種でも実験する．
- ユーザーがさらに楽しめるコンテンツを追加する．
  - 保存オブジョン
  - 連続加入画像

日本語やタイ語などに拡張する．

参考文献

[1] QRCode http://www．qrcode．com/aboutqr．html

[2] 二次元コードシンボル QR コード 基本仕様(JISX0510) 日本規格協会 2004 年

[3] サムレットウィット ダムリ： “イメージ付きカラー化二次元コードの構成と読み取り特性の研究”
福岡工業大学大学院 情報通信工学専攻 修士論文（2013 年 3 月）

[4] i-nigma http://www．i-nigma．com/i-nigmahp．html

[5] QRReader for iPhone http://itunes．apple．com/us/app/qr-reader-for-iphone/id368494609?mt=8

[6] Best Barcode Reader https://itunes．apple．com/us/app/best-barcode-scanner-scan/id454087075?mt=8

[7] 萩原 学：“デザイン二次元コード”, 電子情報通信学会誌 Vol.94, No.4, PP.341-343 （2011 年 4 月）.

[8] 越智祐樹：“QR コードへの画像の挿入”， 山元研究室 福岡工業大学 情報通信工学科 卒業論文 (2008 年 3 月)

発表論文

[1] Damri Samretwit，槇　俊孝，若原俊彦 : ”カラー化ドット絵付二次元コードエディタの構成”，2013 年電子情報通信学会総合大会　D-9-30　（2013 年 3 月）

[2] Damri Samretwit，若原俊彦 : ”カラー二次元コードエディタの構成と読み取り特性”，信学技報，vol．113 ，no．43 ，LOIS2013-2，pp．85-89（2013 年 5 月）

[3] Toshihiko Wakahara，Damri Samretwit，Toshitaka Maki，Noriyasu Yamamoto:"Design and Characteristics of Colored Two-Dimensional Code with Images"，ITCS-Session4: Image Processing and Analysis pp．323 -332 (July 2013)

[4] Damri Samretwit，槇　俊孝，若原俊彦 : ”カラー二次元コードの読み取り特性”，O-0002，pp．507-508　FIT2013（2013 年 9 月）

[5] Damri Samretwit，Toshihiko Wakahara，Toshitaka Maki：”Two-Dimensional Color Code Editor and its Characteristics” 2013 年電子情報通信学会ソサイエティ大会 BS-7-40，S-110（2013 年 9 月）

[6] Toshihiko Wakahara，Damri Samretwit，Toshitaka Maki：”Reading Characteristics of 2-Dimensional Color Code Editor”，SISA2013 2013 International Workshop on Smart Info-Media Systems in Asia RS3-7，pp．246-250（2013 年 9 月）

[7] サムレットウィット ダムリ, 若原俊彦, 槇　俊孝:”カラー二次元コードシミュレータの読み取り特性”, 信学技報，vol. 113，no.　，LOIS2013-（2014 年 3 月）

[8] サムレットウィット ダムリ，若原俊彦，槇　俊孝 : ”カラー二次元コードエディタの構成”，2014 年電子情報通信学会総合大会 D-9-27（2014 年 3 月）

付録　プログラム

- QRCode.java に追加したコード

```
//QR-Code Maker
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.image.*;
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;
import java.io.IOException;
import java.util.Hashtable;

import java.awt.event.KeyEvent;

class QRCode extends JFrame implements ActionListener{

		////////////////////////////////////////////////////////////////////////////
		//////////////////////////////////////////フィールド//////////////////////////////////////
		//DataBase からデータ取得
			public static final int NO_DATA = DataBase.NO_DATA;	//定数(データ無)
			public static final int BLACK_DATA = DataBase.BLACK_DATA;	//定数(黒データ)
			public static final int WHITE_DATA = DataBase.WHITE_DATA;	//定数(白データ)
			public static final int K_DATA = DataBase.K_DATA;	//定数(形式、型番予定)
			public static final int DATA_1 = DataBase.DATA_1;	//定数(コードデータ 1)
			public static final int DATA_0 = DataBase.DATA_0;	//定数(コードデータ 0)
			public static final int DATA_VER = DataBase.DATA_VER;	//定数(型番情報)
			public static final int DATA_K = DataBase.DATA_K;	//定数(形式情報)
			public static final Color TITLE_GRAY = DataBase.TITLE_GRAY;	//定数(タイトル画面背景色)
			public static final Color TITLE_GRAY2 = DataBase.TITLE_GRAY2;	//定数(タイトル画面背景色 2)
			public static final String BR = DataBase.BR;	//定数(改行コード)
			public static final String GRAP_SEPARATOR=DataBase.GRAP_SEPARATOR;	//定数(グラフィック部の分離符を格納)
			public static final String BR_IO=DataBase.BR_IO;	//定数(ファイル入出力時の改行コードの置き換え文字)
			int[][][] versionMaxLen=DataBase.getVersionMaxLen();	//QR コード仕様 DB
			char[] eisuutaiou=DataBase.eisuutaiou;
		//英数字モードの文字対応表 DB
			int[] dataA=DataBase.dataA;
				//GF(2^8)の指数表現の対応 DB
		//ファイルの入出力
			JFileChooser chooser = new JFileChooser();	//ファイルの入出力
		//GUI オブジェクト宣言
			JLabel StatusBar;
					//ステータスバーもどき
			JButton b;
						//変換ボタン
			JButton inB;
					//開くボタン
			JButton editorB;
					//エディタを開くボタン
			JButton clearB;
		  public static JTextArea inputD;
				//入力エリア
			static JTextArea info;
				//情報表示エリア
			JScrollPane inputDS;
				//入力エリアのスクロールバー
			JScrollPane infoS;
					//情報表示エリアのスクロールバー
			static JComboBox verList;
				//型番リスト
			static JComboBox maskList;
				//マスクリスト
			static JComboBox lvList;
				//訂正レベルリスト
			static JComboBox sizeList;
		  QRShow[] qrs=new QRShow[simbolNumMax];
		//QR コード表示オブジェクト
			QREditor qre;
					//エディタオブジェクト
			ImageIcon icon = new ImageIcon("icon.png");	//アイコンの設定
		//メニューバー
			MenuItem NewFile	= new MenuItem( "新規作成" );	//[ファイル]->[新規作成]
			MenuItem Open		= new MenuItem( "開く" );
		//[ファイル]->[開く]
			MenuItem Exit		= new MenuItem( "プログラムの終了" );	//[ファイル]->[プログラムの終了]
			MenuItem Editor		= new MenuItem( "エディタの起動" );	//[　編集]->[エディタの起動]
			MenuItem Do		= new MenuItem( "変換" );	//[　編集]->[変換]
			MenuItem ClearInput	= new MenuItem( "入力エリアのクリア" );	//[表示]->[入力エリアのクリア]
			MenuItem ClearInfo	= new MenuItem( "情報エリアのクリア" );	//[表示]->[情報エリアのクリア]
```

```
		MenuItem Version		= new MenuItem( "バージョン情報" );		//[ヘルプ]->[バージョン情報]
	//その他
		int WindowSizeX=450;
				//ウィンドウサイズ(X)
		int WindowSizeY=770;
				//ウィンドウサイズ(Y)
		float version=(float)0.7;
			//プログラムのバージョンを格納
		static int simbolNumMax=999;
			//シンボル表示オブジェクトの最高数
		boolean coop=false;
					//グラフィックが埋め込まれているかどうか
		int count=0;
					//一時的な変数
		String Title="QR-Code Maker";
			//プログラムのタイトル
		String[] dataBit;
					//データビット列を格納
		String Inputdata;
					//入力されたデータを格納
		String[] InputdataDiv=new String[2];
	//入力データをデータ部とグラフィック部に分けて格納
		String SelectedLv="L(7%)";
			//選択された訂正レベルをテキストで格納
		String SelectedMode;
				//選択されたエンコードモードをテキストで格納
		int verMax=40;
						//最大型番
		static int ver=0;
				//選択された型番を格納 0～40
		static int lv=0;
				//選択された訂正レベルを格納 0～3
		static int encodeMode;
				//エンコードモードを格納 0～3
		int mozisusizisiLen;
				//文字数指示子の長さを格納
		int souCodeGosu=0;
					//総コード語数を格納
		int errorTeiseiSu=0;
				//エラー訂正数を格納
		String[][] errorF;
					//誤り訂正符号を格納する配列
		int errorTS;
					//誤り訂正
		int[] g;
						//誤り訂正
		int[] f;
						//誤り訂正
		String[] f2;
				//誤り訂正
		String[] dtmp;
				//誤り訂正
		String[][] dataB;
				//インタリーブ配置の各データビット列を格納
		int RSSum=1;
				//RS ブロック数の合計
		int dBkazu;
						//RS ブロック数が複数の時にコード数の計算に使用
		int simbolNum=0;
				//シンボルの生成番号
		public static int[][] simbol;
		//シンボルを格納
		public static int simSize=0;
		//シンボルのサイズを格納
		static int mask=0;
					//適用するマスクの種類を格納
		int[][] currentSimbol;
				//マスクの評価をする場合の各シンボルを格納
		int score;
						//シンボルの評価の失点を格納
		String modesikibetusi="0000";
			//モード識別子を格納
		String mozisusizisi=null;
			//文字数指示子を格納
		static BufferedImage readImage=null;
		static int imageX=0;
		static int imageY=0;

   static int initImageSize=10;
   static int imageSize=0;

//////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////コンストラクタ//////////////////////////////////////
QRCode0{
//メニュー
		MenuBar MyMenu = new MenuBar();
		Menu FileMenu = new Menu( "ファイル" );
			NewFile.addActionListener(this);
		Open.addActionListener(this);
		Exit.addActionListener(this);
			FileMenu.add( NewFile );
```

```
                        FileMenu.add( Open );
                        FileMenu.addSeparator();
                        FileMenu.add( Exit );
                Menu EditMenu = new Menu( "編集" );
                        Editor.addActionListener(this);
                        Do.addActionListener(this);
                        EditMenu.add(Editor);
                        EditMenu.addSeparator();
                        EditMenu.add(Do);
                Menu ShowMenu = new Menu( "表示" );
                        ClearInput.addActionListener(this);
                        ClearInfo.addActionListener(this);
                        ShowMenu.add(ClearInput);
                        ShowMenu.add(ClearInfo);
                Menu HelpMenu = new Menu( "ヘルプ" );
                        Version.addActionListener(this);
                        HelpMenu.add(Version);
                MyMenu.add( FileMenu );
                MyMenu.add( EditMenu );
                MyMenu.add( ShowMenu );
                MyMenu.add( HelpMenu );
                setMenuBar(MyMenu);

        //ステータスバー
                StatusBar=new JLabel("Version:"+version);
        //ボタン
                b=new JButton("変換");
                b.addActionListener(this);
        //ボタン(開く)
                inB=new JButton("ファイルを開く");
                inB.addActionListener(this);
        //ボタン(開く)
                editorB=new JButton("エディタを開く");
                editorB.addActionListener(this);
        //ボタン(開く)
                clearB=new JButton("クリア");
                clearB.addActionListener(this);
//テキストエリア
                //入力エリア
                        inputD=new JTextArea("", 4,38);
                        inputD.setLineWrap(true);
                        inputDS=new JScrollPane(inputD);
                        inputDS.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
                        inputDS.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
                        inputDS.setPreferredSize(new Dimension(WindowSizeX-25, 140));
                //情報エリア
                        info=new JTextArea(Title+BR+BR, 4, 30);
                        infoS=new JScrollPane(info);
                        infoS.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
                        infoS.setPreferredSize(new Dimension(WindowSizeX-25, 200));
        //コンボボックス
                //訂正レベル
                        lvList = new JComboBox();
                        lvList.addItem(" L ( 7%)");
                        lvList.addItem(" M (15%)");
                        lvList.addItem(" Q (25%)");
                        lvList.addItem(" H (30%)");
                        lvList.setLightWeightPopupEnabled(false);   //重量コンポーネント化してプルダウンが表示されない不具合解消
                        lvList.addActionListener(this);
                //型番
                        verList = new JComboBox();
                        verList.addItem("Auto");
                        for(int i=1;i<=verMax;i++){
                                verList.addItem(Integer.toString(i));
                        }
                        verList.setLightWeightPopupEnabled(false); //重量コンポーネント化してプルダウンが表示されない不具合解消
                        verList.addActionListener(this);
                //マスク
                        maskList = new JComboBox();
                        maskList.addItem("Auto");
                        for(int i=0;i<8;i++){
                                maskList.addItem(Integer.toString(i));
                        }
                        maskList.setLightWeightPopupEnabled(false);                 //重量コンポーネント化してプルダウンが表示されない不具合解消
                        maskList.addActionListener(this);
          //サイズ
                sizeList = new JComboBox();
                        sizeList.addItem("1");
                        sizeList.addItem("2");
                        sizeList.addItem("3");
                        sizeList.addItem("4");
                sizeList.addItem("5");
                sizeList.addItem("6");
                        sizeList.setLightWeightPopupEnabled(false);//重量コンポーネント化してプルダウンが表示されない不具合解消
                        sizeList.addActionListener(this);
        //レイアウト
                getContentPane().setLayout(new BoxLayout(getContentPane(),BoxLayout.Y_AXIS));
                getContentPane().setBackground(TITLE_GRAY);
                Panel p1,p2,p3,p4,p5,p6,p7,p8,p9,p10,p11;
                getContentPane().add(p1=new Panel());
                getContentPane().add(p2=new Panel());
                getContentPane().add(p3=new Panel());
                getContentPane().add(p4=new Panel());
                getContentPane().add(p5=new Panel());
```

```
                getContentPane().add(p7=new Panel());
                getContentPane().add(p9=new Panel());
                getContentPane().add(p10=new Panel());
                getContentPane().add(p11=new Panel());
                p1.setLayout(new FlowLayout());
                p1.add(new JLabel(new ImageIcon("title.gif")));
                p2.setLayout(new FlowLayout());
                p2.setBackground(TITLE_GRAY2);
                p2.add(inB);
                p2.add(editorB);
                p4.setLayout(new FlowLayout(FlowLayout.LEFT));
                p4.add(new JLabel("QR コード化するデータ"));
                p5.add(inputDS);
                p7.add(new JLabel("訂正"));
                p7.add(lvList);
                p7.add(new JLabel("　型番"));
                p7.add(verList);
                p7.add(new JLabel("　マスク"));
                p7.add(maskList);
        p7.add(new JLabel("　サイズ"));
                p7.add(sizeList);
                p7.add(new JLabel("　　"));
                p7.add(b);
        p7.add(clearB);
                p9.setLayout(new FlowLayout(FlowLayout.LEFT));
                p9.add(new JLabel("情報エリア"));
                p10.add(infoS);
                //p11.setBackground();
                p11.setLayout(new FlowLayout(FlowLayout.LEFT));
                p11.add(StatusBar,BorderLayout.SOUTH);
                setIconImage(icon.getImage());

                //終了処理
                        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                //表示
                        init();
                        setTitle(Title);
                        setSize(WindowSizeX, WindowSizeY);
                        setVisible(true);
        }

        ////////////////////////////////////////////////////////////////////////////////
        ////////////////////////////////////////イベントメソッド////////////////////////////////////
        public void actionPerformed(ActionEvent e) {
                Object obj = e.getSource();
                //ボタン(変換)のイベント
                        if(obj==b || obj==Do){
                                QRSyori();
                                if(simbolNum<simbolNumMax){
                                        int qrsWidth=simSize*QRShow.blockWidth+(QRShow.quiet*2);
                                        qrs[simbolNum]=new QRShow(simbolNum,simbol,simSize,readImage,imageX,imageY);
                                        qrs[simbolNum].setTitle(""+ver+SelectedLv.charAt(0)+"型　: "+inputD.getText());
                                        qrs[simbolNum].setLocation(getX()+(getWidth()/2)-(qrsWidth/2),
getY()+(getHeight()/2)-(qrsWidth/2));
                                        simbolNum++;
                                        ver=verList.getSelectedIndex();
                                        //readImage=null;
                                }else{
                                        System.out.println("エラー：　これ以上シンボルを生成できません。");
                                        info.setText(info.getText()+"エラー：　これ以上シンボルを生成できません。"+BR);
                                }
                        }
                //ボタン(エディタを開く)のイベント
                        if(obj==editorB || obj==Editor){
                                qre=new QREditor();
                                qre.setLocation(getX()+getWidth(), getY());
                        }
        /*//ボタン(クリア)のイベント
                if(obj==clearB || obj==Clear){
                                inputD.setEditable(true);
                        inputD.setText("");
                        readImage = null;
                        }*/
                //ボタン(開く)のイベント
                        if(obj==inB || obj==Open){
                                int returnVal = chooser.showOpenDialog(this);
                                try {
                                        if (returnVal == JFileChooser.APPROVE_OPTION) {          // OK ボタンが押された時の処理
                                                File file = chooser.getSelectedFile();
                                                BufferedReader in   = new BufferedReader(new FileReader(file));
                                                boolean ioError=false;
                                                int simSizeTmp=0;
                                                int[][] simbolTmp=new int[simSizeTmp][simSizeTmp];
                                                String str1=null;
                                                String str2=null;
                                                //型番、モード、訂正レベル、マスク取得
                                                        String[] ioData;
                                                        ioData=in.readLine().split(",");
                                                        if(ioData.length!=4){
                                                                ioError=true;
                                                        }
                                                simError:
                                                if(!ioError){
                                                        //シンボル取得
```

```
                                                                                simSizeTmp = 21+(4*(Integer.parseInt(ioData[0])-1));
                                                                                simbolTmp=new int[simSizeTmp][simSizeTmp];
                                                                                String[] ioData2;
                                                                                for(int i=0;i<simSizeTmp;i++){
                                                                                        ioData2=in.readLine().split(",");
                                                                                        if(ioData2.length!=simSizeTmp){
                                                                                                ioError=true;
                                                                                                break simError;
                                                                                        }
                                                                                }
                                                                        //入力データ、出力データ取得
                                                                                str1=in.readLine();
                                                                                str2=in.readLine();
                                                                                if(str2.indexOf(str1+GRAP_SEPARATOR)==-1){
                                                                                        ioError=true;
                                                                                }
                                                                }
                                                                //エラー処理
                                                                        if(ioError){
                                                                                DataBase.ioError();
                                                                                JOptionPane.showMessageDialog(this, "ファイルの読
み込みに失敗しました。","エラー", JOptionPane.ERROR_MESSAGE);
                                                                        }else{
                                                                                ver=Integer.parseInt(ioData[0]);
                                                                                encodeMode=Integer.parseInt(ioData[1]);
                                                                                lv=Integer.parseInt(ioData[2]);
                                                                                mask=Integer.parseInt(ioData[3]);
                                                                                verList.setSelectedIndex(ver);
                                                                                lvList.setSelectedIndex(lv);
                                                                                maskList.setSelectedIndex(mask+1);
                                                                                str2=str2.replaceAll(BR_IO,BR);
                                                                                inputD.setText(str2);
                                                                        }
                                                                in.close();
                                                                        }
                                                                } catch(Exception ex){}
                                                        }
                                                //コンボボックス(型番)のイベント
                                                        if(obj==verList){
                                                                ver=verList.getSelectedIndex();
                                                        }
                                                //コンボボックス(マスク)のイベント
                                                        if(obj==maskList){
                                                                mask=maskList.getSelectedIndex();
                                                        }
                                                //コンボボックス(訂正レベル)のイベント
                                                        if(obj==lvList){
                                                                lv=lvList.getSelectedIndex();
                                                                switch(lv){
                                                                        case 0:
                                                                                SelectedLv="L(7%)";
                                                                                break;
                                                                        case 1:
                                                                                SelectedLv="M(15%)";
                                                                                break;
                                                                        case 2:
                                                                                SelectedLv="Q(25%)";
                                                                                break;
                                                                        case 3:
                                                                                SelectedLv="H(30%)";
                                                                }
                                                        }
                                        if(obj==sizeList){
                                                                QRShow.blockWidth=sizeList.getSelectedIndex()+4;
                                                QRShow.quiet=QRShow.blockWidth*5;
                                                        }
                                                //新規作成のイベント
                                                        if(obj==NewFile){
                                                                init();
                                                        }
                                                //プログラムの終了のイベント
                                                        if(obj==Exit){
                                                                System.exit(0);
                                                        }
                                                //情報エリアのクリアイベント
                                                        if(obj==clearB || obj==ClearInput){
                                                                inputD.setEditable(true);
                                                        inputD.setText("");
                                                        readImage = null;
                                                        }
                                                //情報エリアのクリアイベント
                                                        if(obj==ClearInfo){
                                                                info.setText("");
                                                        }
                                                //バージョン情報のイベント
                                                        if(obj==Version){
                                                                JOptionPane.showMessageDialog(this, "バージョン:"+version,"バージョン情報", JOptionPane.PLAIN_MESSAGE);
                                                        }
        }

        ////////////////////////////////////////////////////////////////////////////////
        ////////////////////////////////////////////メインメソッド//////////////////////////////////////
        public static void main(String[] args) {
                        QRCode qr=new QRCode();
        }
```

```
////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////QR コード計算処理メソッド////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
public void QRSyori(){
		Inputdata=inputD.getText();
		//////////////////////////////////////////////////////
		//////////////グラフィックが埋め込まれているか/////////////
		int GrapIndex=Inputdata.lastIndexOf(GRAP_SEPARATOR);
			if(GrapIndex==-1){
				//グラフィックデータがない時の処理
				coop=false;
				InputdataDiv[0]=Inputdata;
			}else{
				//グラフィックデータがある時の処理
				coop=true;
				encodeMode=2;
				InputdataDiv[0]=Inputdata.substring(0,GrapIndex);
				InputdataDiv[1]=Inputdata.substring(GrapIndex+GRAP_SEPARATOR.length());
		}

		//////////////////////////////////////////////////////
		////////////////////////モード決定//////////////////////////
			encodeMode=DataBase.getEncodeMode(InputdataDiv[0]);
			switch(encodeMode){
				case 0:
					SelectedMode="数字";
					break;
				case 1:
					SelectedMode="英数字";
					break;
				case 2:
					SelectedMode="8bit バイト";
					break;
				case 3:
					SelectedMode="漢字";
					break;
				default:
					//エラー
					SelectedMode="不明";
					System.out.println("エラー：　エンコードモードの値が異常です。　："+encodeMode);
					info.setText(info.getText()+"エラー：　エンコードモードの値が異常です。　："+encodeMode+BR);
			}
			//エディタで自動でモードを決定できるようになれば↓削除
				/*if(coop){
					encodeMode=2;
				}
				*/
		//////////////////////////////////////////////////////
		///////////////決定されたモードの型番別容量を取得///////////
		for(int i=1;i<=40;i++){
			mozisusizisiLen=DataBase.mozisusizisiLenGet(i,encodeMode);
				switch(encodeMode){
				case 0:
					versionMaxLen[i][lv][encodeMode]=(int)((versionMaxLen[i][lv][4]*8) -(4+mozisusizisiLen))/10*3;
					if(((versionMaxLen[i][lv][4]*8) -(4+mozisusizisiLen))%10>6){
						versionMaxLen[i][lv][encodeMode]+=2;
					}else if(((versionMaxLen[i][lv][4]*8) -(4+mozisusizisiLen))%10>3){
						versionMaxLen[i][lv][encodeMode]+=1;
					}
					break;
				case 1:
					versionMaxLen[i][lv][encodeMode]=(int)((versionMaxLen[i][lv][4]*8)
-(4+mozisusizisiLen))/11*2+(int)((((versionMaxLen[i][lv][4]*8) -(4+mozisusizisiLen))%11+4)/10.0);
					break;
				case 2:
					versionMaxLen[i][lv][encodeMode]=(int)((versionMaxLen[i][lv][4]*8) -(4+mozisusizisiLen))/8;
					break;
				case 3:

				}
		}

		//////////////////////////////////////////////////////
		////////////////////バージョン決定が自動の時/////////////////
		if(ver==0){
			for(int i=1;i<41;i++){
				if(Inputdata.length()<=versionMaxLen[i][lv][encodeMode]){
					ver=i;
					break;
				}
			}
		}
		//デバッグ用
			System.out.println("入力文字数="+Inputdata.length()+"、型番"+ver+"の許容文字数="+versionMaxLen[ver][lv][encodeMode]);

		//////////////////////////////////////////////////////
		////////////////////////文字数指示子//////////////////////////
		mozisusizisiLen=DataBase.mozisusizisiLenGet(ver,encodeMode);
		mozisusizisi = DataBase.get2(InputdataDiv[0].length(), mozisusizisiLen);

		//////////////////////////////////////////////////////
		////////////////////////モード指示子を取得//////////////////
		modesikibetusi=DataBase.getMode(encodeMode);
```

```
////////////////////////////////////////////////////////
//////////////////////データの符号化////////////////////
InputdataDiv[0]=DataBase.DtoB(InputdataDiv[0],encodeMode);

////////////////////////////////////////////////////////
//////////////////////終端パターン挿入////////////////////
String tmpS=null;
int terminatorLen=0;
tmpS = modesikibetusi+mozisusizisi+InputdataDiv[0];
//デバッグ用
		System.out.println("データの量="+tmpS.length()+"　限界量="+((versionMaxLen[ver][lv][4]*8)));
String terminator="";
terminatorLen=(versionMaxLen[ver][lv][4]*8)-tmpS.length();
if(terminatorLen>3){
		terminator = "0000";
}else{
		for(int i=0;i<terminatorLen;i++){
			terminator+="0";
			//デバッグ用
				System.out.println("終端パターン挿入数="+terminatorLen);
		}
}
tmpS +=terminator;
////////////////////////////////////////////////////////
///////////////8bit ずつに区切り、埋め草 bit 挿入/////////////////
//データビットを格納する配列の生成
		dataBit=new String[versionMaxLen[ver][lv][4]+versionMaxLen[ver][lv][5]];
int nowPlace=0;
//グラフィックがあればデータを代入
		if(coop){
			tmpS+=InputdataDiv[1];
		}
//8bit 単位で dataBit 配列にデータを格納
		for (int i=0; tmpS.length()>0; i++) {
			if(tmpS.length()<8){
				count=tmpS.length();
			}else{
				count=8;
			}
			dataBit[nowPlace] = tmpS.substring(0, count);
			tmpS = tmpS.substring(count);
			nowPlace++;
		}
if(!coop){
		//最後の bit 列が 8bit になるよう埋め草 bit 挿入(グラフィックない場合)
			for (int j=0; dataBit[nowPlace-1].length()<8; j++) {
				dataBit[nowPlace-1] += "0";
			}
		//埋め草 bit 列をデータコード数を満たすまで挿入(グラフィックない場合)
			count = 0;
			for (; ; nowPlace++) {
				if (nowPlace<versionMaxLen[ver][lv][4]) {
					if (count%2==1) {
						dataBit[nowPlace] = "00010001";
					} else {
						//dataBit[nowPlace] = "11101100";
						dataBit[nowPlace] = "11101101";
					}
					count++;
				} else {
					break;
				}
			}
}
//デバッグ用
		System.out.println("[データビット列の表示]");
		for (int j=0; j<versionMaxLen[ver][lv][4]; j++) {
			System.out.println("dataBit["+j+"] ="+dataBit[j]+"=>"+DataBase.get10(dataBit[j]));
		}

//データを 10 進数に変換
		for (int i=0; i<versionMaxLen[ver][lv][4]; i++) {
			dataBit[i] = Integer.toString(DataBase.get10(dataBit[i]));
		}

////////////////////////////////////////////////////////
//////////////////////誤り訂正////////////////////////////
//総コード語数を計算
		souCodeGosu = versionMaxLen[ver][lv][4]+versionMaxLen[ver][lv][5];
//RS ブロック数の合計を計算
		RSSum=versionMaxLen[ver][lv][6]+versionMaxLen[ver][lv][7];
//エラー訂正数を計算
		errorTeiseiSu = versionMaxLen[ver][lv][5]/2;
		if (ver == 1 && lv == 0) {
			//1L の時は 2
			errorTeiseiSu = 2;
		} else if (ver == 1 && lv == 1) {
			//1M の時は 4
			errorTeiseiSu = 4;
		} else if (ver == 2 && lv == 0) {
			//2L の時は 4
			errorTeiseiSu = 4;
		} else if (ver == 1 && lv == 3) {
```

```
                                        //1H の時は 8
                                        errorTeiseiSu = 8;
                                }
                        //各ブロックの誤り訂正コード数を取得
                                errorTS=versionMaxLen[ver][lv][5];
                                errorTS=errorTS/RSSum;
                        //生成多項式 g を取得
                                G(errorTS);
                        //RS ブロック数 1 の時の誤り訂正
                                if (RSSum == 1) {
                                        errorF=new String[1][errorTS];
                                        errorCode(dataBit,errorTS,errorF[0],versionMaxLen[ver][lv][4]);
                                        //データを配置する順にソート
                                                for(int i=0;i<errorF[0].length;i++){
                                                        dataBit[versionMaxLen[ver][lv][4]+i]=errorF[0][i];
                                                }
                                        //デバッグ用
                                                for(int i=0;i<dataBit.length;i++){
                                                        //System.out.println("dataBit["+i+"]= "+dataBit[i]);
                                                }
                                }
                        //RS ブロック数が 2 以上の時の誤り訂正
                                if(RSSum>1){
                                        //デバッグ用
                                                System.out.println("RS ブロック数 2 以上");
                                        //データブロックの配列を生成
                                                dataB=new String[RSSum][];
                                        // 1 つめの RS ブロック数
                                                count=0;
                                                for(int i=0;i<versionMaxLen[ver][lv][6];i++){
                                                        dBkazu=(int)(versionMaxLen[ver][lv][4]/RSSum);
                                                        dataB[i]=new String[dBkazu];
                                                        for(int j=0;j<dBkazu;j++){
                                                                dataB[i][j]=dataBit[count];
                                                                count++;
                                                        //デバッグ用

        System.out.println("dataB["+i+"]["+j+"]="+dataB[i][j]+"   →   "+DataBase.get2(Integer.parseInt(dataB[i][j]),10));
                                                        }
                                                }
                                        //2 つめの RS ブロック数
                                                for(int i=0;i<versionMaxLen[ver][lv][7];i++){
                                                        dBkazu=(int)(versionMaxLen[ver][lv][4]/RSSum)+1;
                                                        dataB[i+versionMaxLen[ver][lv][6]]=new String[dBkazu];
                                                        for(int j=0;j<dBkazu;j++){
                                                                dataB[i+versionMaxLen[ver][lv][6]][j]=dataBit[count];
                                                                count++;
                                                                //デバッグ用

        System.out.println("dataB["+(i+versionMaxLen[ver][lv][6])+"]["+j+"]="+dataB[i+versionMaxLen[ver][lv][6]][j]+"                                                →
"+DataBase.get2(Integer.parseInt(dataB[i+versionMaxLen[ver][lv][6]][j]),10));
                                                        }
                                                }

                                        //それぞれのデータブロックの誤り訂正符号を計算
                                                //誤り訂正符号を格納する配列の生成
                                                        errorF=new String[RSSum][errorTS];
                                                for(int i=0;i<RSSum;i++){
                                                        errorCode(dataB[i],errorTS,errorF[i],dataB[i].length);
                                                }
                                        //データを配置する順にソート
                                                dataBit=new String[versionMaxLen[ver][lv][4]+errorTS*RSSum];
                                        //データコード部分をソート
                                                count=0;
                                                for(int i=0;i<Math.floor(versionMaxLen[ver][lv][4]/RSSum);i++){
                                                        for(int j=0;j<dataB.length;j++){
                                                                dataBit[count]=dataB[j][i];
                                                                count++;
                                                        }
                                                }
                                                for(int i=versionMaxLen[ver][lv][6];i<RSSum;i++){
                                                        dataBit[count]=dataB[i][((int)(versionMaxLen[ver][lv][4]/RSSum))];
                                                        count++;
                                                }
                                        //誤り訂正符号部分をソート
                                                for(int i=0;i<errorTS;i++){
                                                        for(int j=0;j<RSSum;j++){
                                                                dataBit[count]=errorF[j][i];
                                                                count++;
                                                        }
                                                }
                                }

                                //デバッグ用
                                        for(int i=0;i<dataBit.length;i++){
                                                //System.out.println("最終 dataBit["+i+"]= "+dataBit[i]);
                                        }
                        ////////////////////////////////////////////////////////
                        ////////////////////データの 2 進化////////////////////////
                                for (int i=0; i<dataBit.length; i++) {
                                        dataBit[i]=DataBase.get2(Integer.parseInt(dataBit[i]),8);
                                }
        ////////////////////////////////////////////////////////////////////////////////
        ////////////////////////////////////////QR コード表示処理////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////////////////////

		//////////////////////////////////////////////////////
		////////////////シンボルの生成、初期設定//////////////////////
		//シンボルを生成
			simSize = 21+(4*(ver-1));
			simbol = new int[simSize][simSize];
		//シンボルに固定データの埋め込み
			DataBase.Indata(ver,simSize,simbol);

		//////////////////////////////////////////////////////
		/////////////////シンボルにデータの配置/////////////////////
		//初期設定
			int direct=1;
			int LeftOrRight=1;
			int nowX=simSize-1;
			int nowY=simSize-1;
			int haitiBitLength;
			int[] haitiBit=new int[dataBit.length*8];
			count=0;
			for(int i=0;i<dataBit.length;i++){
				for(int j=0;j<dataBit[i].length();j++){
					haitiBit[count]=Character.digit(dataBit[i].charAt(j),10);
					count++;
				}
			}
		  //配置
			haitiBitLength=haitiBit.length;
			int compNum=0;
			for(int i=0;compNum<haitiBit.length;i++){
				count=0;
				//デバッグ用
					if(nowX<-1){
						System.out.println("エラー：　データがシンボルに収まりきりません　　残りビット
数："+(haitiBit.length-i-1));
						info.setText(info.getText()+"エラー：　データがシンボルに収まりきりません　　残り
ビット数："+(haitiBit.length-i-1)+BR);
					}
				if(nowX>=0 && nowX<=simSize-1 && nowY>=0 && nowY<=simSize-1){
					if(simbol[nowY][nowX]==0){
						//データが空ならビットを配置
							if(haitiBit[compNum]==1){
								simbol[nowY][nowX]=DATA_1;
							}else if(haitiBit[compNum]==0){
								simbol[nowY][nowX]=DATA_0;
							}
							count=1;
							compNum++;
					}
				}

				if(count==0){
					//移動
						if(LeftOrRight==1){
							//右側
							//デバッグ用
								//System.out.println("右側
"+nowX+",　"+nowY);
							nowX--;
							LeftOrRight=-LeftOrRight;
						}else if(nowY>=0 && nowY<=simSize-1){
							//デバッグ用
								//System.out.println("　上　下　移　動　可　能
"+nowX+",　"+nowY);
							nowY-=direct;
							nowX++;
							LeftOrRight=-LeftOrRight;
						}else{
							//デバッグ用
								//System.out.println("　　上　　下　　限　　界
"+nowX+",　"+nowY);
							nowX--;
							if(nowX==6){
								nowX--;
							}
							LeftOrRight=-LeftOrRight;
							nowY+=direct;
							direct=-direct;
						}

				}
			}

		//////////////////////////////////////////////////////
		////////////////////////マスク処理//////////////////////////
		int Maxscore=99999;
		currentSimbol=new int[simSize][];
		//各マスクの評価
			if(mask==0){
				for(int i=0;i<8;i++){
					//シンボルのコピー
						for(int j=0;j<simSize;j++){
							currentSimbol[j]=(int[])simbol[j].clone();
						}
```

```
                                                //マスクの適用
                                                        DataBase.mask(currentSimbol,i,simSize,0);
                                                //形式情報の埋め込み
                                                        DataBase.setFormatInformation(currentSimbol,lv,i);
                                                //シンボルの評価
                                                        System.out.print("Mask:"+i+"    ");
                                                        score=CheckSimbol(currentSimbol);
                                                //マスクの決定
                                                        if(Maxscore>score){
                                                                mask=i;
                                                                Maxscore=score;
                                                        }
                                        }
                                }else{
                                        mask-=1;
                                        score=CheckSimbol(simbol);
                                }
                        //シンボルの決定
                                        System.out.println("採用されたマスク="+mask);
                                //マスクの適用
                                        if(coop){
                                                DataBase.mask(simbol,mask,simSize,0);
                                        }else{
                                                DataBase.mask(simbol,mask,simSize,0);
                                        }
                                //形式情報の埋め込み
                                        DataBase.setFormatInformation(simbol,lv,mask);
                        //デバッグ用
                                        //DataBase.sTrace(simbol,simSize,"データ配置");

                        //終了
                                        info.setText(info.getText()+"型番="+ver+"　訂正レベル="+SelectedLv+"　マスク="+mask+"　失点="+score+"　エンコードモード
="+SelectedMode+BR);
                                        info.setText(info.getText()+"QR 化処理終了しました。"+BR+"========================================"+BR);
                                        mask=maskList.getSelectedIndex();
        }


        ////////////////////////////////////////////////////////////////////////////
        ////////////////////////////////////////////////////////////////////////////
        ////////////////////////////////////メソッド////////////////////////////////////
        ////////////////////////////////////////////////////////////////////////////
        ////////////////////////////////////////////////////////////////////////////

        ////////////////////////////////////////////////////
        ///////////GF(256)の α を整数に変換するメソッド///////////////
        public int ahenkan(int a){
                if(a<1 || a>255){
                        return 0;
                }
                return dataA[a-1];
        }

        ////////////////////////////////////////////////////
        ///////////GF(256)の整数を α に変換するメソッド///////////////
        public int agyakuhenkan(int a){
                for(int iii=0;iii<255;iii++){
                        if(a==dataA[iii]){
                                return iii+1;
                        }else if(a==255){
                                return 1;
                        }
                }
                System.out.println("エラー：　agyakuhenkan()にて"+a);
                info.setText(info.getText()+"エラー：　agyakuhenkan()にて変換できない数字が入力されました。( 1～255) a="+a+BR);
                return 0;
        }
        ////////////////////////////////////////////////////
        ////////////////べき乗を計算するメソッド//////////////////
        public int beki(int a,int b){
                int c=1;
                for(int iii=0;iii<b;iii++){
                        c*=a;
                }
                return c;
        }

        ////////////////////////////////////////////////////
        ////////////////生成多項式を求めるメソッド/////////////////
        public int[] G(int a){
                //生成多項式 g(x)を求める
                        switch(a){
                                case 7:
                                        int[] g1 = {0, 87, 229, 146, 149,238,102,21};
                                        g=(int[])g1.clone();
                                        break;
                                case 10:
                                        int[] g2 = {0,251,67,46,61,118,70,64,94,32,45};
                                        g=(int[])g2.clone();
                                        break;
                                case 13:
                                        int[] g3 = {0,74,152,176,100,86,100,106,104,130,218,206,140,78};
                                        g=(int[])g3.clone();
                                        break;
```

```
                case 15:
                        int[] g4 = {0,8,183,61,91,202,37,51,58,58,237,140,124,5,99,105};
                        g=(int[])g4.clone();
                        break;
                case 16:
                        int[] g5 = {0,120,104,107,109,102,161,76,3,91,191,147,169,182,194,225,120};
                        g=(int[])g5.clone();
                        break;
                case 17:
                        int[] g6 = {0, 43, 139, 206, 78, 43, 239, 123, 206, 214, 147, 24, 99, 150, 39, 243, 163, 136};
                        g=(int[])g6.clone();
                        break;
                case 18:
                        int[] g7 = {0,215,234,158,94,184,97,118,170,79,187,152,148,252,179,5,98,96,153};
                        g=(int[])g7.clone();
                        break;
                case 20:
                        int[] g8 = {0,17,60,79,50,61,163,26,187,202,180,221,225,83,239,156,164,212,212,188,190};
                        g=(int[])g8.clone();
                        break;
                case 22:
                        int[] g9 = {0,210,171,247,242,93,230,14,109,221,53,200,74,8,172,98,80,219,134,160,105,165,231};
                        g=(int[])g9.clone();
                        break;
                case 24:
                        int[] g10 =
{0,229,121,135,48,211,117,251,126,159,180,169,152,192,226,228,218,111,0,117,232,87,96,227,21};
                        g=(int[])g10.clone();
                        break;
                case 26:
                        int[] g11 =
{0,173,125,158,2,103,182,118,17,145,201,111,28,165,53,161,21,245,142,13,102,48,227,153,145,218,70};
                        g=(int[])g11.clone();
                        break;
                case 28:
                        int[] g12 =
{0,168,223,200,104,224,234,108,180,110,190,195,147,205,27,232,201,21,43,245,87,42,195,212,119,242,37,9,123};
                        g=(int[])g12.clone();
                        break;
                case 30:
                        int[] g13 =
{0,41,173,145,152,216,31,179,182,50,48,110,86,239,96,222,125,42,173,226,193,224,130,156,37,251,216,238,40,192,180};
                        g=(int[])g13.clone();
                        break;
                case 32:
                        int[] g14 =
{0,10,6,106,190,249,167,4,67,209,138,138,32,242,123,89,27,120,185,80,156,38,60,171,60,28,222,80,52,254,185,220,241};
                        g=(int[])g14.clone();
                        break;
                case 34:
                        int[] g15 =
{0,111,77,146,94,26,21,108,19,105,94,113,193,86,140,163,125,58,158,229,239,218,103,56,70,114,61,183,129,167,13,98,62,129,51};
                        g=(int[])g15.clone();
                        break;
                case 36:
                        int[] g16 =
{0,200,183,98,16,172,31,246,234,60,152,115,24,167,152,113,248,238,107,18,63,218,37,87,210,105,177,120,74,121,196,117,251,113,233,30,120};
                        g=(int[])g16.clone();
                        break;
                case 40:
                        int[] g17 =
{0,59,116,79,161,252,98,128,205,128,161,247,57,163,56,235,106,53,26,187,174,226,104,170,7,175,35,181,114,88,41,47,163,125,134,72,20,232,53,35,15};
                        g=(int[])g17.clone();
                        break;
                case 42:
                        int[] g18 =
{0,42,250,103,221,230,25,18,137,231,0,3,58,242,221,191,110,84,230,8,188,106,96,147,15,131,139,34,101,223,39,101,213,199,237,254,201,123,171,162,194,117,50,96};
                        g=(int[])g18.clone();
                        break;
                case 44:
                        int[] g19 =
{0,190,7,61,121,71,246,69,55,168,188,89,243,191,25,72,123,9,145,14,247,1,238,44,78,143,62,224,126,118,114,68,163,52,194,217,147,204,169,37,130,113,102,73,181};
                        g=(int[])g19.clone();
                        break;
                case 46:
                        int[] g20 =
{0,112,94,88,112,253,224,202,115,187,99,89,5,54,113,129,44,58,16,135,216,169,211,36,1,4,96,60,241,73,104,234,8,249,245,119,174,52,25,157,224,43,202,223,19,82,15};
                        g=(int[])g20.clone();
                        break;
                case 48:
                        int[] g21 =
{0,228,25,196,130,211,146,60,24,251,90,39,102,240,61,178,63,46,123,115,18,221,111,135,160,182,205,107,206,95,150,120,184,91,21,247,156,140,238,191,11,94,227,84,50,163,39,
34,108};
                        g=(int[])g21.clone();
                        break;
                case 50:
                        int[] g22 =
{0,232,125,157,161,164,9,118,46,209,99,203,193,35,3,209,111,195,242,203,225,46,13,32,160,126,209,130,160,242,215,242,75,77,42,189,32,113,65,124,69,228,114,235,175,124,17
0,215,232,133,205};
                        g=(int[])g22.clone();
                        break;
                case 52:
                        int[] g23 =
{0,116,50,86,186,50,220,251,89,192,46,86,127,124,19,184,233,151,215,22,14,59,145,37,242,203,134,254,89,190,94,59,65,124,113,100,233,235,121,22,76,86,97,39,242,200,220,101
,33,239,254,116,51};
                        g=(int[])g23.clone();
```

```
                                                break;
                                        /*
                                        case 54:
                                                int[] g24 = {};
                                                g=(int[])g24.clone();
                                                break;
                                        case 56:
                                                int[] g25 = {};
                                                g=(int[])g25.clone();
                                                break;
                                        case 58:
                                                int[] g26 = {};
                                                g=(int[])g26.clone();
                                                break;
                                        case 60:
                                                int[] g27 = {};
                                                g=(int[])g27.clone();
                                                break;
                                        */
                                        //
                                        //
                                        //他の部分の定義も行う
                                        //
                                        //
                                        default:
                                                System.out.println("エラー：　誤り訂正コード語数が異常で、生成多項式 g(x)が作れません　: "+a);
                                                info.setText(info.getText()+"エラー：　誤り訂正コード語数が異常で、生成多項式 g(x)が作れません　:
"+a+BR);
                                }
                        return g;
        }

        ////////////////////////////////////////////////////////
        ////////////////誤り訂正符号を計算するメソッド////////////////
        public void     errorCode (String a[],int b,String c[],int d){

                //各配列を生成、初期化
                        f = new int[d+errorTS];
                        f2 =new String[d+errorTS];
                //各配列に初期値 0 を入れる
                        for (int i=0; i<d+errorTS; i++) {
                                f2[i]="0";
                                f[i]=0;
                        }
                //f2 配列に dataBit をコピー
                        for (int i=0; i<d; i++) {
                                f2[i] = a[i];
                        }
                        //デバッグ用
                        /*
                                for(int i=0;i<f2.length;i++){
                                        System.out.println("f2["+i+"]="+f2[i]);
                                }
                        */
                //誤り訂正符号の計算
                        count=0;
                        for (int j=0; j<d; j++) {
                                count = Integer.parseInt(f2[j]);
                                for (int i=j; i<j+1+errorTS; i++) {
                                        f[i] = g[i-j]+ahenkan(count);
                                        if (f[i]>255) {
                                                f[i] -= 255;
                                        }
                                        f[i] = agyakuhenkan(f[i]);
                                        f2[i] = DataBase.exor(DataBase.get2(f[i], 1), DataBase.get2(Integer.parseInt(f2[i]), 1));
                                        f2[i] = Integer.toString(DataBase.get10(f2[i]));
                                }
                        }
                //誤り訂正符号を配列に格納
                        for (int i=0; i<b; i++) {
                                c[i] = f2[d+i];
                        }
                //デバッグ用
                        /*
                        for(int i=0;i<f2.length;i++){
                                System.out.println("f2["+i+"]="+f2[i]);
                        }
                        System.out.println("==========================================");
                        for(int i=0;i<f.length;i++){
                                System.out.println("f["+i+"]="+f[i]);
                        }
                        System.out.println("[誤り訂正符号を表示]");
                        for (int i=0; i<b; i++) {
                                System.out.println("c["+i+"] ="+c[i]);
                        }
                        */
        }

        ////////////////////////////////////////////////////////
        ////////////////////シンボルの評価をするメソッド////////////////////
        public int CheckSimbol(int a[][]){
        int score=0;
        int countX=0;
        int countY=0;
        boolean colorX=false;
```

```
		boolean colorY=false;
		//シンボルを黒と白にした配列を用意
			boolean[][] b=new boolean[simSize][simSize];
			b=simbolToBoolean(a);
			//デバッグ用
				//DataBase.sTrace(b,simSize,"デバッグ用");
			//同色の行・列の隣接モジュール
				//行
				for(int i=0;i<simSize;i++){
					colorX=b[0][i];
					countX=1;
					for(int j=1;j<simSize;j++){
						if(b[j][i]==colorX && j<simSize-1){
							countX++;
						}else{
							if(b[j][i]==colorX){
								countX++;
							}
							if(countX>=5){
								//System.out.println("同色の行の隣接モジュールs["+i+"]["+(j-countX)+"]-s["+i+"]["+(j-1)+"]：-"+(countX-2));
								score+=countX-2;
							}
							countX=1;
							colorX=b[j][i];
						}
					}
				}
				//列
				for(int i=0;i<simSize;i++){
					colorY=b[i][0];
					countY=1;
					for(int j=1;j<simSize;j++){
						if(b[i][j]==colorY && j<simSize-1){
							countY++;
						}else{
							if(b[i][j]==colorY){
								countY++;
							}
							if(countY>=5){
								//System.out.println("同色の列の隣接モジュールs["+(j-countY)+"]["+i+"]-s["+(j-1)+"]["+i+"]：-"+(countY-2));
								score+=countY-2;
							}
							countY=1;
							colorY=b[i][j];
						}
					}
				}
			//同色のモジュールブロック数
				for(int i=0;i<simSize-1;i++){
					for(int j=0;j<simSize-1;j++){
						if(b[i][j]==b[i+1][j] && b[i][j]==b[i][j+1] && b[i][j]==b[i+1][j+1]){
							//System.out.println("同色のモジュールブロックs["+j+"]["+i+"]-s["+(j+1)+"]["+(i+1)+"]：-3");
							score+=3;
						}
					}
				}
			//全体に占める暗部分の割合
				count=0;
				for(int i=0;i<simSize;i++){
					for(int j=0;j<simSize;j++){
						if(b[i][j]){
							count++;
						}
					}
				}
				//デバッグ用
					//System.out.println("全体="+(simSize*simSize)+"　黒="+count);
				float hi,hi2;
				hi=(float)count/(simSize*simSize)*100;
				hi2=Math.abs(hi-50);
				//System.out.println("全体に占める暗部分の割合="+(int)hi+"%：-"+((int)(hi2/5)*10+10));
				score+=(int)(hi2/5)*10+10;
			//行・列における 1:1:3:1:1 比率のパターン
			int hi3=0;
			int count=0;
			int dankai=0;
			int hit;
			int Save=0;
			int nowX=0,nowY=0;
			//列
			for(int i=0;i<simSize;i++){
				hi3=0;
				count=0;
				dankai=0;
				hit=0;
				for(int j=0;j<simSize;j++){
					if(dankai==0){
						if(b[j][i]){
							hi3++;
						}else{
							if(hi3>0){
```

```
                                        nowX=j-hi3;
                                        nowY=i;
                                        dankai++;
                                }
                        }
                }
                if(dankai==1){
                        //System.out.println("段階 1   比率="+hi3+"   s["+nowY+"]["+nowX+"]");
                        if(!b[j][i]){
                                count++;
                        }else{
                                if(count==hi3){
                                        dankai++;
                                        Save=j;
                                }else{
                                        dankai=0;
                                        hi3=1;
                                }
                                count=0;
                        }
                }
                if(dankai==2){
                        //System.out.println("段階 2   比率="+hi3+"   s["+nowY+"]["+nowX+"]");
                        if(b[j][i]){
                                count++;
                        }else{
                                if(count==(hi3*3)){
                                        dankai++;
                                }else{
                                        dankai=0;
                                        j=Save;
                                        hi3=1;
                                }
                                count=0;
                        }
                }
                if(dankai==3){
                        //System.out.println("段階 3   比率="+hi3+"   s["+nowY+"]["+nowX+"]");
                        if(!b[j][i]){
                                count++;
                        }else{
                                if(count==hi3){
                                        dankai++;
                                }else{
                                        dankai=0;
                                        j=Save;
                                        hi3=1;
                                }
                                count=0;
                        }
                }
                if(dankai==4){
                        //System.out.println("段階 4   比率="+hi3+"   s["+nowY+"]["+nowX+"]");
                        if(b[j][i]){
                                count++;
                                if(j==simSize-1){
                                        hit=1;
                                        dankai++;
                                }
                        }else{
                                if(count==hi3){
                                        dankai++;
                                }else{
                                        dankai=0;
                                        hi3=0;

                                }
                                count=0;
                        }
                }
                if(dankai==5){
                        //System.out.println("段階 5   比率="+hi3+"   s["+nowY+"]["+nowX+"]");
                        loop1:
                        for(int ii=j;ii<j+4;ii++){
                                if(hit==1){
                                        break loop1;
                                }
                                if(ii<simSize){
                                        if(b[ii][i]){
                                                break loop1;
                                        }
                                        if(ii==j+3){
                                                hit=1;
                                        }
                                }else{
                                        hit=1;
                                        break loop1;
                                }
                        }
                        loop2:
                        for(int ii=nowX-1;ii>nowX-5;ii--){
                                if(hit==1){
                                        break loop2;
                                }
                                if(ii>-1){
```

```
                                                if(b[ii][i]){
                                                        dankai=0;
                                                        break loop2;
                                                }
                                                if(ii==nowX-4){
                                                        hit=1;
                                                }
                                        }else{
                                                hit=1;
                                                break loop2;
                                        }
                                }
                                if(hit==1){
                                        //System.out.println("1:1:3:1:1 の比率　s["+nowY+"]["+nowX+"] : -30");
                                        score+=30;
                                        hit=0;
                                        dankai=0;
                                        hi3=0;
                                }
                        }
                }
        }
        //行
        for(int i=0;i<simSize;i++){
                hi3=0;
                count=0;
                dankai=0;
                hit=0;
                for(int j=0;j<simSize;j++){
                        if(dankai==0){
                                if(b[i][j]){
                                        hi3++;
                                }else{
                                        if(hi3>0){
                                                nowX=i;
                                                nowY=j-hi3;
                                                dankai++;
                                        }
                                }
                        }
                        if(dankai==1){
                                //System.out.println("段階 1　比率="+hi3+"　s["+nowY+"]["+nowX+"]");
                                if(!b[i][j]){
                                        count++;
                                }else{
                                        if(count==hi3){
                                                dankai++;
                                                Save=j;
                                        }else{
                                                dankai=0;
                                                hi3=1;
                                        }
                                        count=0;
                                }
                        }
                        if(dankai==2){
                                //System.out.println("段階 2　比率="+hi3+"　s["+nowY+"]["+nowX+"]");
                                if(b[i][j]){
                                        count++;
                                }else{
                                        if(count==(hi3*3)){
                                                dankai++;
                                        }else{
                                                dankai=0;
                                                j=Save;
                                                hi3=1;
                                        }
                                        count=0;
                                }
                        }
                        if(dankai==3){
                                //System.out.println("段階 3　比率="+hi3+"　s["+nowY+"]["+nowX+"]");
                                if(!b[i][j]){
                                        count++;
                                }else{
                                        if(count==hi3){
                                                dankai++;
                                        }else{
                                                dankai=0;
                                                j=Save;
                                                hi3=1;
                                        }
                                        count=0;
                                }
                        }
                        if(dankai==4){
                                //System.out.println("段階 4　比率="+hi3+"　s["+nowY+"]["+nowX+"]");
                                if(b[i][j]){
                                        count++;
                                        if(j==simSize-1){
                                                hit=1;
                                                dankai++;
                                        }
                                }else{
                                        if(count==hi3){
```

```
                                                dankai++;
                                        }else{
                                                dankai=0;
                                                hi3=0;

                                        }
                                        count=0;
                                }
                        }
                        if(dankai==5){
                                //System.out.println("段階 5   比率="+hi3+"   s["+nowY+"]["+nowX+"]");
                                loop3:
                                for(int ii=i;ii<i+4;ii++){
                                        if(hit==1){
                                                break loop3;
                                        }
                                        if(ii<simSize){
                                                if(b[ii][j]){
                                                        break loop3;
                                                }
                                                if(ii==i+3){
                                                        hit=1;
                                                }
                                        }else{
                                                hit=1;
                                                break loop3;
                                        }
                                }
                                loop4:
                                for(int ii=nowY-1;ii>nowY-5;ii--){
                                        if(hit==1){
                                                break loop4;
                                        }
                                        if(ii>-1){
                                                if(b[ii][j]){
                                                        dankai=0;
                                                        break loop4;
                                                }
                                                if(ii==nowY-4){
                                                        hit=1;
                                                }
                                        }else{
                                                hit=1;
                                                break loop4;
                                        }
                                }
                                if(hit==1){
                                        //System.out.println("1:1:3:1:1 の比率   s["+nowY+"]["+nowX+"] : -30");
                                        score+=30;
                                        hit=0;
                                        dankai=0;
                                        hi3=0;
                                }
                        }
                }
        }



System.out.println("失点="+score);
return score;
}

////////////////////////////////////////////////////////////////
//////////シンボルを int から boolean に変換するメソッド//////////
public boolean[][] simbolToBoolean(int[][] a){
        boolean[][] b=new boolean[simSize][simSize];
        for(int i=0;i<simSize;i++){
                for(int j=0;j<simSize;j++){
                        switch(a[i][j]){
                                case NO_DATA:
                                        //データ未定義
                                        b[i][j]=false;
                                        break;
                                case BLACK_DATA:
                                        //黒データ
                                        b[i][j]=true;
                                        break;
                                case WHITE_DATA:
                                        //白データ
                                        b[i][j]=false;
                                        break;
                                case K_DATA:
                                        //形式、型番情報が入る予定
                                        b[i][j]=false;
                                        break;
                                case DATA_1:
                                        //データ 1
                                        b[i][j]=true;
                                        break;
                                case DATA_0:
                                        //データ 0
                                        b[i][j]=false;
                                        break;
```

```
                                                                        case DATA_VER:
                                                                                //型番情報
                                                                                b[i][j]=true;
                                                                                break;
                                                                        case DATA_K:
                                                                                //形式情報
                                                                                b[i][j]=true;
                                                                                break;
                                                                        default:
                                                                                //エラー
                                                                                System.out.println("エラー：　シンボルのデータがおかしいです。　："+a[i][j]);
                                                                                info.setText(info.getText()+"エラー：　シンボルのデータがおかしいです。　：
"+a[i][j]+BR);
                                                        }
                                                }
                                        }
                                        return b;
                        }

                        ////////////////////////////////////////////////////////
                        ////////////////////画面の初期化をするメソッド////////////////////
                        static void init(){
                                //変数初期化
                                        ver=0;
                                        lv=0;
                                        encodeMode=0;
                                        mask=0;
                                //入力フィールド初期化
                                        inputD.setText("");
                                //各コンボボックス初期化
                                        verList.setSelectedIndex(ver);
                                        lvList.setSelectedIndex(lv);
                                        maskList.setSelectedIndex(mask);
                        }
}
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
//////////////////////////////QR コードの表示クラス//////////////////////////////
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////

class QRShow extends JFrame{
                        ImageIcon icon = new ImageIcon("icon.png");                                                                     //アイコンの設定
                        public static GPanel[] Grap=new GPanel[QRCode.simbolNumMax];            //グラフィックパネルの型
                        public static int blockWidth= 4;
                        //ブロックの大きさを格納
                        public static int quiet=blockWidth*5;                                                                                   //クワイエットゾーンの
大きさを格納
                        public int windowSizeX;
                                //ウインドウの大きさを格納
                        public int windowSizeY;
                                //ウインドウの大きさを格納
                        BufferedImage readImage=null;
                        int imageX=0;
                        int imageY=0;
                        QRShow(int simbolNum,int[][] simbol,int simSize,BufferedImage readImage,int imageX,int imageY){
                                //グラフィックパネルインスタンス生成
                                        Grap[simbolNum]=new GPanel(simSize,simbol,readImage,imageX,imageY);
                                        //サイズ調整
                                                windowSizeX=simSize*blockWidth+(quiet*2);
                                                windowSizeY=windowSizeX;
                                                Grap[simbolNum].setPreferredSize(new Dimension(windowSizeX, windowSizeY));
                                //表示
                                        setSize(windowSizeX+5,windowSizeY+25);
                                        System.out.println("simSize="+simSize);
                                        setIconImage(icon.getImage());
                                        getContentPane().add(Grap[simbolNum],BorderLayout.CENTER);
                                        setVisible(true);
                        }
}
class GPanel extends JPanel{
                        int[][] simbol;
                        int simSize=0;
                        BufferedImage readImage=null;
                        int imageX=0;
                        int imageY=0;

                public static Color NO_DATA_COLOR = Color.BLACK;
                public static Color BLACK_DATA_COLOR = Color.BLACK;
                public static Color DATA_COLOR = Color.BLACK;
                public static Color DATA_VER_COLOR = Color.BLACK;
                public static Color DATA_K_COLOR = Color.BLACK;
                public static Color BACKGROUND_COLOR = Color.WHITE;

                public static Color[][] simbolC;
                public static float[][] simbolT;

                        GPanel(int a,int[][] b,BufferedImage c,int d,int e){
                                setBackground(Color.white);
                          Image readImageIm;
                                simSize=a;
                                simbol=new int[simSize][];
                                for(int i=0;i<simSize;i++){
                                        simbol[i]=(int[])b[i].clone();
```

```
                        }
                try{
                    if(!QREditor.fitMode){
                        readImageIm=c.getScaledInstance((int)((QRCode.initImageSize+QRCode.imageSize)*((float)QRShow.blockWidth)), -1,
Image.SCALE_AREA_AVERAGING);
                    }
                    else{
                        readImageIm=c.getScaledInstance((int)(simSize*((float)QRShow.blockWidth)), (int)(simSize*((float)QRShow.blockWidth)),
Image.SCALE_AREA_AVERAGING);
                    }
                    readImage=QREditor.toBufferedImage(readImageIm);
                }
                catch(Exception ex){readImage=c;}
                        imageX=d;
                        imageY=e;
              }
              public void paintComponent(Graphics g) {
              super.paintComponent(g);
            if(!QREditor.swapMode)
            {
                        //QR コード描画
                drawDot(g);

                        //読み込み画像表示
                        drawImage(g);
            }
            else{
                //読み込み画像表示
                        drawImage(g);
                //QR コード描画
                drawDot(g);
            }
              }
        private void drawDot(Graphics g)
        {
            Graphics2D g2d = (Graphics2D)g;
            g2d.setColor(Color.BLACK);
            for(int i=0;i<simSize;i++){
                for(int j=0;j<simSize;j++){
                    if(QRCode.inputD.isEditable()==true){
                        g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER,QREditor.colorTran));
                        if(simbol[i][j]==QRCode.BLACK_DATA || simbol[i][j]==QRCode.DATA_1 || simbol[i][j]==QRCode.DATA_VER || simbol[i][j]==QRCode.DATA_K){
                            g2d.setColor(Color.BLACK);
                            g2d.fillRect(j*QRShow.blockWidth+QRShow.quiet,i*QRShow.blockWidth+QRShow.quiet,QRShow.blockWidth,QRShow.blockWidth);
                        }
                        else //if(simbol[i][j]==QRCode.DATA_0||simbol[i][j]==QRCode.NO_DATA)
                        {
                            g2d.setColor(BACKGROUND_COLOR);
                            g2d.fillRect(j*QRShow.blockWidth+QRShow.quiet,i*QRShow.blockWidth+QRShow.quiet,QRShow.blockWidth,QRShow.blockWidth);
                        }
                    }
                    else{
                        g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER,QREditor.colorTran));
                        switch(simbol[i][j]){
                                case QRCode.BLACK_DATA:
                                    g2d.setColor(BLACK_DATA_COLOR);
                                    break;
                                case QRCode.DATA_1:
                                    if(simbolC[i][j]==NO_DATA_COLOR&&simbolT[i][j]==0.0f){
                                        g2d.setColor(NO_DATA_COLOR);
                                    }
                                    else {
                                        g2d.setColor(simbolC[i][j]);
                                        g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER,simbolT[i][j]));
                                    }
                                    break;
                                case QRCode.DATA_VER:
                                    g2d.setColor(DATA_VER_COLOR);
                                    break;
                                case QRCode.DATA_K:
                                    g2d.setColor(DATA_K_COLOR);
                                    break;
                                case QRCode.DATA_0:
                                    if(simbolC[i][j]==BACKGROUND_COLOR){
                                        g2d.setColor(BACKGROUND_COLOR);
                                        g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER,simbolT[i][j]));
                                    }
                                    else{
                                        g2d.setColor(simbolC[i][j]);
                                        g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER,simbolT[i][j]));
                                    }
                                    break;
                                default:
                                    g2d.setColor(BACKGROUND_COLOR);
                                    g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER,simbolT[i][j]));
                                    break;
                        }
                        g2d.fillRect(j*QRShow.blockWidth+QRShow.quiet,i*QRShow.blockWidth+QRShow.quiet,QRShow.blockWidth,QRShow.blockWidth);
                    }
                }
            }
        }
        private void drawImage(Graphics g)
        {
            if(readImage!=null){
```

```
                    Graphics2D g2 = (Graphics2D)g;
                    g2.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER,QREditor.imgTran));
                    g2.drawImage(readImage, (imageX*QRShow.blockWidth)+QRShow.quiet, (imageY*QRShow.blockWidth)+QRShow.quiet, this);
                }
            }
            private void drawFilter(Graphics g)
            {


            }
}
```

- QREditor.java に追加したコード

```
//QR-Code Editor
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.JFileChooser;
import java.awt.image.*;
import java.awt.image.BufferedImage;
import java.io.File;
import javax.imageio.ImageIO;
import java.io.IOException;
import java.util.Hashtable;

class QREditor extends JFrame implements ActionListener,KeyListener{

                ////////////////////////////////////////////////////////////////////////////////
                ////////////////////////////////////////////フィールド////////////////////////////////////////
                //DataBase からデータ取得
                        public static final int NO_DATA = DataBase.NO_DATA;                                     //定数(データ無)
                        public static final int BLACK_DATA = DataBase.BLACK_DATA;                       //定数(黒データ)
                        public static final int WHITE_DATA = DataBase.WHITE_DATA;                       //定数(白データ)
                        public static final int K_DATA = DataBase.K_DATA;                                       //定数(形式、型番予定)
                        public static final int DATA_1 = DataBase.DATA_1;                                       //定数(コードデータ 1)
                        public static final int DATA_0 = DataBase.DATA_0;                                       //定数(コードデータ 0)
                        public static final int DATA_VER = DataBase.DATA_VER;                           //定数(型番情報)
                        public static final int DATA_K = DataBase.DATA_K;                                       //定数(形式情報)
                        public static final int DATA_E1 = DataBase.DATA_E1;                                     //定数(エディタでの仮コ
ードデータ 1)
                        public static final int DATA_E0 = DataBase.DATA_E0;                                     //定数(エディタでの仮コ
ードデータ 0)
                        public static final Color TITLE_GRAY = DataBase.TITLE_GRAY;                     //定数(タイトル画面背景色)
                        public static final Color TITLE_GRAY2 = DataBase.TITLE_GRAY2;           //定数(タイトル画面背景色 2)
                        public static final String BR = DataBase.BR;                                            //定数(改行コード)
                        public static final String GRAP_SEPARATOR=DataBase.GRAP_SEPARATOR;      //定数(グラフィック部の分離符を格納)
                        public static final String BR_IO=DataBase.BR_IO;                                        //定数(ファイル入出力時
の改行コードの置き換え文字)
                        int[][][] versionMaxLen=DataBase.getVersionMaxLen();                            //QR コード仕様 DB
                //GUI オブジェクト宣言
                        JComboBox verList;
                                                                //型番リスト
                        JComboBox lvList;
                                                                //誤り訂正リスト
                        JComboBox maskList;
                                                                //マスクリスト
                        //
                  JComboBox imgTranList;
                  JComboBox regList;
                  JComboBox colorTranList;
                  //
                  JButton openB;
                                                                //ファイルオープンボタン
                        JButton saveB;
                                                                        //ファイルセーブボタン
                        JButton resetB;
                                                                //リセットボタン
                        JButton colorB;
                                                                //かラー化ボタン
                        JButton okB;
                                                                //OK ボタン
                        JButton ok_imgB;
                                                                //画像だけ送るボタン
                        JButton imageInB;
                                                                //画像 In ボタン
                        JButton addB;
                                                                //画像埋め込みボタン
                        //
                  JButton sizeUp;
                  JButton sizeDown;
                  JButton fit;
                  JButton swap;
                  //
                  JTextArea dataInputArea;
                        //データを入力するエリア
                        JScrollPane dataInputAreaS;
                                        //↑のスクロール部
                        static JRadioButton RadioB_pencil;
```

```
		//描画モードのラジオボタン(鉛筆)
		static JRadioButton RadioB_eraser;
		//描画モードのラジオボタン(消しゴム)
		static JRadioButton RadioB_both;
		//描画モードのラジオボタン(鉛筆&消しゴム)
		static JRadioButton RadioB_fill;
		//描画モードのラジオボタン(塗りつぶし)
		ButtonGroup RadioBgr = new ButtonGroup();
	//描画モードのラジオボタンのグループ
		public static EGPanel EGrap;
		//グラフィックオブジェクト
		JScrollPane EGrapS;
							//↑のスクロール部
		BufferedImage readImageBuf = null;
		static BufferedImage readImage;
	  ColorValueSliderControl tc;
		ImageIcon icon = new ImageIcon("icon.png");                                                                    //アイコンの設定
	//ファイルの入出力
		JFileChooser chooser = new JFileChooser();                                                                     //ファイルの入出力
	//メニューバー
		MenuItem NewFile	= new MenuItem( "新規作成" );                                                              //[ファイル]->[新規作成]
		MenuItem Open			= new MenuItem( "開く" );
	//[ファイル]->[開く]
		MenuItem Save			= new MenuItem( "保存" );
	//[ファイル]->[保存]
		MenuItem bOpen			= new MenuItem( "画像ファイル読み込み" );		//[ファイル]->[画像ファイル読み込み]
		MenuItem Exit			= new MenuItem( "エディタを閉じる" );		//[ファイル]->[エディタを閉じる]
		MenuItem Do			= new MenuItem( "データを QR-Code Maker に送る" );//[編集]->[データを QR-Code Maker に送る]
		MenuItem ClearInput	= new MenuItem( "入力エリアのクリア" );		//[表示]->[入力エリアのクリア]
		MenuItem Version	= new MenuItem( "バージョン情報" );		//[ヘルプ]->[バージョン情報]

	//その他
		int WindowSizeX=0;
							//ウインドウ幅 X
		int WindowSizeY=0;
							//ウインドウ幅 Y
		int encodeMode=0;
							//エンコードモード
		int ver=1;
								//型番を格納
		int lv=0;
								//訂正レベル
		int mask=0;
								//マスク
		public static float imgTran = 1.0f;
	  public static float colorTran = 1.0f;
	  int RSSum=1;
							//RS ブロック数
		int mode_mozisuLen;
							//モード指示子と文字数指示子の長さ
		int verMax=40;
								//処理できる最高の型番
		int moveSpeed=EGPanel.blockSize;
								//画像ファイル移動スピード
		String notGrapCode=null;
			//グラフィック以外の部分のビット列
		String Inputdata="";
					//入力されたデータ
		String dataB;
							//データビット列
		String Title="QR-Code Editor";
			//プログラムのタイトル
		String outstr=null;
							//出力される文字列
		static int paintMode=0;
					//描画モード(1:鉛筆 2:消しゴム 3:両方 4:塗りつぶし)
		//
	  static boolean fitMode = false;
	  static boolean swapMode = false;
	  static boolean edit = false;
	  //
	  static int simSize=21;
			//シンボルのサイズ
		static int[][] simbol;
					//シンボルを格納
		public static Color[][] simbolC;
	  public static float[][] simbolT;
	  int [] dataBit;

	////////////////////////////////////////////////////////////////////////////////
	//////////////////////////////////////////コンストラクタ//////////////////////////////////////
	QREditor(){
		//初期化
			ver=1;
			lv=0;
			mask=0;
			encodeMode=0;
```

```
                        simSize=21+4*(ver-1);
                        simbol=new int[simSize][simSize];
        simbolC=new Color[simSize][simSize];
        simbolT=new float[simSize][simSize];
//メニュー
                        MenuBar MyMenu = new MenuBar();
                        Menu FileMenu = new Menu( "ファイル" );
                                NewFile.addActionListener(this);
                                Open.addActionListener(this);
                                Save.addActionListener(this);
                                bOpen.addActionListener(this);
                                Exit.addActionListener(this);
                                FileMenu.add( NewFile );
                                FileMenu.add( Open );
                                FileMenu.add( Save );
                                FileMenu.add( bOpen );
                                FileMenu.addSeparator();
                                FileMenu.add( Exit );
                        Menu EditMenu = new Menu( "編集" );
                                Do.addActionListener(this);
                                EditMenu.add(Do);
                        Menu ShowMenu = new Menu( "表示" );
                                ClearInput.addActionListener(this);
                                ShowMenu.add(ClearInput);
                        Menu HelpMenu = new Menu( "ヘルプ" );
                                Version.addActionListener(this);
                                HelpMenu.add(Version);
                        MyMenu.add( FileMenu );
                        MyMenu.add( EditMenu );
                        MyMenu.add( ShowMenu );
                        MyMenu.add( HelpMenu );
                        setMenuBar(MyMenu);
            //ウインドウサイズ調節
                        windowResize(simSize);
            //グラフィックエリア
                        EGrap=new EGPanel();
                        EGrap.addMouseListener(new MyMouseListener());
                        EGrap.addMouseMotionListener(new MyMouseMotionListener());
                        EGrap.addKeyListener(this);
            //ボタン
                        //ファイルオープンボタン
                                openB=new JButton("開く");
                                openB.addActionListener(this);
                                openB.addKeyListener(this);
                        //ファイルセーブボタン
                                saveB=new JButton("保存");
                                saveB.addActionListener(this);
                                saveB.addKeyListener(this);
                        //リセットボタン
                                resetB=new JButton("クリア");
                                resetB.addActionListener(this);
                                resetB.addKeyListener(this);
                        //カラー化ボタン
                                colorB=new JButton("カラー化");
                                colorB.addActionListener(this);
                                colorB.addKeyListener(this);
                        //OK ボタン
                                okB=new JButton("OK");
                                okB.addActionListener(this);
                                okB.addKeyListener(this);
                        //画像だけ送るボタン
                                ok_imgB=new JButton("画像だけ送る");
                                ok_imgB.addActionListener(this);
                                ok_imgB.addKeyListener(this);
                        //画像読み込みボタン
                                imageInB=new JButton("画");
                                imageInB.addActionListener(this);
                                imageInB.addKeyListener(this);
                        //画像埋め込みボタン
                                addB=new JButton("埋");
                                addB.addActionListener(this);
                                addB.addKeyListener(this);
        //画像を大きくするボタン
                sizeUp = new JButton("大");
                sizeUp.addActionListener(this);
                                sizeUp.addKeyListener(this);
        //画像を小さくするボタン
                sizeDown = new JButton("小");
                sizeDown.addActionListener(this);
                                sizeDown.addKeyListener(this);
        //
                fit = new JButton("広");
                fit.addActionListener(this);
                                fit.addKeyListener(this);
        //
                swap = new JButton("裏");
                swap.addActionListener(this);
                                swap.addKeyListener(this);
            //データ入力エリア
                        dataInputArea=new JTextArea("", 4,38);
                        dataInputArea.setLineWrap(true);
                        dataInputAreaS=new JScrollPane(dataInputArea);
                        dataInputAreaS.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
```

```
                dataInputAreaS.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
                dataInputAreaS.setPreferredSize(new Dimension(WindowSizeX-25, 70));
                dataInputAreaS.addKeyListener(this);
        //グラフィックオブジェクトのスクロール設定
                EGrapS=new JScrollPane(EGrap);
                EGrapS.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
                EGrapS.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
                EGrapS.setPreferredSize(new Dimension(500, 420));
                EGrapS.addKeyListener(this);
        //コンボボックス
                //訂正レベル
                        lvList = new JComboBox();
                        lvList.addItem(" L ( 7%)");
                        lvList.addItem(" M (15%)");
                        lvList.addItem(" Q (25%)");
                        lvList.addItem(" H (30%)");
                        lvList.setLightWeightPopupEnabled(false);  //重量コンポーネント化してプルダウンが表示されない不具合解消
                        lvList.addActionListener(this);
                        lvList.addKeyListener(this);
                //型番
                        verList = new JComboBox();
                        for(int i=1;i<=verMax;i++){
                                verList.addItem(Integer.toString(i)+" ");
                        }
                        verList.setLightWeightPopupEnabled(false); //重量コンポーネント化してプルダウンが表示されない不具合解消
                        verList.addActionListener(this);
                        verList.addKeyListener(this);
                //マスク
                        maskList = new JComboBox();
                        for(int i=0;i<8;i++){
                                maskList.addItem(Integer.toString(i)+"   ");
                        }
                        maskList.setLightWeightPopupEnabled(false);                 //重量コンポーネント化してプルダウンが表示されない
不具合解消
                        maskList.addActionListener(this);
                        maskList.addKeyListener(this);
        //イメージ透明
                imgTranList = new JComboBox();
                for(int i=100;i>=0;i--){
                                        imgTranList.addItem(Integer.toString(i)+" %");
                                }
                imgTranList.setSelectedIndex(100);
                imgTranList.setLightWeightPopupEnabled(false);      //重量コンポーネント化してプルダウンが表示されない不具合解消
                                imgTranList.addActionListener(this);
                                imgTranList.addKeyListener(this);
        //明暗認識
                regList = new JComboBox();
                for(int i=0;i<=100;i++){
                                        regList.addItem(Integer.toString(i)+" %");
                                }
                regList.setSelectedIndex(80);
                regList.setLightWeightPopupEnabled(false);                  //重量コンポーネント化してプルダウンが表示されない不具合解消
                                regList.addActionListener(this);
                                regList.addKeyListener(this);
        //ドット透明
                colorTranList = new JComboBox();
                for(int i=100;i>=0;i--){
                                        colorTranList.addItem(Integer.toString(i)+" %");
                                }
                colorTranList.setSelectedIndex(100);
                colorTranList.setLightWeightPopupEnabled(false);      //重量コンポーネント化してプルダウンが表示されない不具合解消
                                colorTranList.addActionListener(this);
                                colorTranList.addKeyListener(this);
        //ラジオボタン
                        paintMode=0;
                        RadioB_pencil=new JRadioButton("鉛筆",true);
                        RadioB_eraser=new JRadioButton("消しゴム");
                        RadioB_both=new JRadioButton("領域&背景");
                        RadioB_fill=new JRadioButton("塗りつぶし");
                        RadioBgr=new ButtonGroup();
                        RadioBgr.add(RadioB_pencil);
                        RadioBgr.add(RadioB_eraser);
                        RadioBgr.add(RadioB_both);
                        RadioBgr.add(RadioB_fill);
                        RadioB_pencil.addActionListener(this);
                        RadioB_pencil.addKeyListener(this);
                        RadioB_eraser.addActionListener(this);
                        RadioB_eraser.addKeyListener(this);
                        RadioB_both.addActionListener(this);
                        RadioB_both.addKeyListener(this);
                        RadioB_fill.addActionListener(this);
                        RadioB_fill.addKeyListener(this);
        //レイアウト
                        setFocusable(true);
                        addKeyListener(this);
                        getContentPane().setBackground(TITLE_GRAY);
                        Panel p1,p2,p3,p4,p5,p6,p7;
                        getContentPane().setLayout(new BoxLayout(getContentPane(),BoxLayout.Y_AXIS));
                        getContentPane().add(p1=new Panel());
                        getContentPane().add(p2=new Panel());
                        getContentPane().add(p3=new Panel());
                        getContentPane().add(p4=new Panel());
                        getContentPane().add(p5=new Panel());
getContentPane().add(p7=new Panel());
```

```
                getContentPane().add(p6=new Panel());
                p1.setLayout(new FlowLayout(FlowLayout.LEFT));
                p1.addKeyListener(this);
                p1.add(new JLabel("QR コード化するデータ"));
                p2.setLayout(new FlowLayout());
                p2.addKeyListener(this);
                p2.add(dataInputAreaS);
                p3.addKeyListener(this);
                p3.add(new JLabel("訂正"));
                p3.add(lvList);
                p3.add(new JLabel("型番"));
                p3.add(verList);
                p3.add(new JLabel("マスク"));
                p3.add(maskList);
                p3.add(imageInB);
                p3.add(addB);
                p3.add(colorB);
                p4.addKeyListener(this);
                p4.add(new JLabel("描画モード："));
                p4.add(RadioB_pencil);
                p4.add(RadioB_eraser);
                p4.add(RadioB_both);
                p4.add(RadioB_fill);
        p4.add(new JLabel("輝度値"));
        p4.add(regList);
                EGrap.setPreferredSize(new Dimension(simSize*EGPanel.blockSize, simSize*EGPanel.blockSize));
                p5.addKeyListener(this);
                p5.add(EGrapS);
                p6.addKeyListener(this);
                p6.setLayout(new FlowLayout());
                p6.add(openB);
                p6.add(saveB);
                p6.add(resetB);
                p6.add(ok_imgB);
                p6.add(okB);
        //
        p7.addKeyListener(this);
        p7.add(new JLabel("画像編集"));
        p7.add(sizeUp);
        p7.add(sizeDown);
        p7.add(fit);
        p7.add(swap);
        p7.add(new JLabel("透明"));
        p7.add(imgTranList);
        p7.add(colorTranList);
        //
                setIconImage(icon.getImage());
            //表示
                init();
                setTitle(Title);
                setVisible(true);
    }

    ////////////////////////////////////////////////////////////////////////////////
    ////////////////////////////////////////イベントメソッド////////////////////////////////////////
    public void actionPerformed(ActionEvent e) {
            Object obj = e.getSource();
            //ボタン(開く)のイベント
                if(obj==openB){
                        int returnVal = chooser.showOpenDialog(this);
                        try {
                                if (returnVal == JFileChooser.APPROVE_OPTION) {
                                        File file = chooser.getSelectedFile();
                                        BufferedReader in    = new BufferedReader(new FileReader(file));
                                        boolean ioError=false;
                                        int simSizeTmp=0;
                                        int[][] simbolTmp=new int[simSizeTmp][simSizeTmp];
                                        String str1=null;
                                        String str2=null;
                                        //型番、モード、訂正レベル、マスク取得
                                                String[] ioData;
                                                ioData=in.readLine().split(",");
                                                if(ioData.length!=4){
                                                        ioError=true;
                                                }
                                        simError:
                                        if(!ioError){
                                                //シンボル取得
                                                        simSizeTmp = 21+(4*(Integer.parseInt(ioData[0])-1));
                                                        simbolTmp=new int[simSizeTmp][simSizeTmp];
                                                        String[] ioData2;
                                                        for(int i=0;i<simSizeTmp;i++){
                                                                ioData2=in.readLine().split(",");
                                                                if(ioData2.length==simSizeTmp){
                                                                        for(int
j=0;j<simSizeTmp;j++){

        simbolTmp[i][j]=Integer.parseInt(ioData2[j]);

                                                                        }
                                                                }else{
                                                                        ioError=true;
                                                                        break simError;
                                                                }
                                                        }
                                                //入力データ、出力データ取得
```

```
                                                                        str1=in.readLine();
                                                                        str2=in.readLine();
                                                                        if(str2.indexOf(str1+GRAP_SEPARATOR)==-1){
                                                                                ioError=true;
                                                                        }
                                                                }
                                                                //エラー処理
                                                                        if(ioError){
                                                                        DataBase.ioError();
                                                                        JOptionPane.showMessageDialog(this, "ファイルの読
み込みに失敗しました。","エラー", JOptionPane.ERROR_MESSAGE);
                                                                        }else{
                                                                        setTitle(Title+" "+file.getAbsolutePath());
                                                                        ver=Integer.parseInt(ioData[0]);
                                                                        encodeMode=Integer.parseInt(ioData[1]);
                                                                        lv=Integer.parseInt(ioData[2]);
                                                                        mask=Integer.parseInt(ioData[3]);
                                                                        verList.setSelectedIndex(ver-1);
                                                                        lvList.setSelectedIndex(lv);
                                                                        maskList.setSelectedIndex(mask);
                                                                        simSize = 21+(4*(ver-1));
                                                                        simbol=new int[simSize][];
                                                                        for(int i=0;i<simSize;i++){
                                                                                simbol[i]=(int[])simbolTmp[i].clone();
                                                                        }
                                                                        dataInputArea.setText(str1);
                                                                        str2=str2.replaceAll(BR_IO,BR);
                                                                        outstr=str2;
                                                                        update();
                                                                        }
                                                                in.close();
                                                                }
                                                        } catch(Exception ex){}
                                                }
                                        //ボタン(保存)のイベント
                                                if(obj==saveB){
                                                        data_out();
                                                        int returnVal = chooser.showSaveDialog(this);
                                                        try{
                                                                if (returnVal == JFileChooser.APPROVE_OPTION) {
                                                                BufferedWriter out = new BufferedWriter(new
FileWriter(chooser.getSelectedFile()));
                                                                //型番、モード、訂正レベル、マスク書き出し
                                                                        String[] ioData=new String[1];
                                                                        ioData[0]=ver+","+encodeMode+","+lv+","+mask;
                                                                        out.write(ioData[0]);
                                                                        out.newLine();
                                                                //シンボル書き出し
                                                                        for(int i=0;i<simSize;i++){
                                                                                ioData[0]="";
                                                                                for(int j=0;j<simSize;j++){
                                                                                        ioData[0]+=simbol[i][j]+",";
                                                                                }
                                                                                out.write(ioData[0]);
                                                                                out.newLine();
                                                                        }
                                                                //入力データ書き出し
                                                                        out.write(Inputdata);
                                                                        out.newLine();
                                                                //出力データ書き出し
                                                                        String str=outstr;
                                                                        str=str.replaceAll(BR,BR_IO);
                                                                        out.write(str);
                                                                out.close();
                                                                }
                                                        }
                                                        catch(Exception ex){
                                                                ex.printStackTrace();
                                                        }
                                                }
                                        //ボタン(カラー化)のイベント
                                                if(obj==colorB){
                                                        tc = new ColorValueSliderControl();
                                        tc.setLocation(0, 0);
                                                }
                                        //ボタン(リセット)のイベント
                                                if(obj==resetB){
                                                        init();

                                                }
                                        //ボタン(OK)のイベント
                                                if(obj==okB){
                                                        data_out();
                                                        QRCode.lv=lv;
                                                        QRCode.encodeMode=encodeMode;
                                                        QRCode.ver=ver;
                                                        QRCode.inputD.setText(outstr);
                                                        QRCode.verList.setSelectedIndex(ver);
                                                        QRCode.lvList.setSelectedIndex(lv);
                                                        QRCode.maskList.setSelectedIndex(mask+1);

                                        GPanel.NO_DATA_COLOR = EGPanel.NO_DATA_COLOR;
                                        GPanel.BLACK_DATA_COLOR = EGPanel.BLACK_DATA_COLOR;
                                        GPanel.DATA_COLOR = EGPanel.DATA_COLOR;
                                        GPanel.DATA_VER_COLOR = EGPanel.DATA_VER_COLOR;
```

```
GPanel.DATA_K_COLOR = EGPanel.DATA_K_COLOR;
GPanel.BACKGROUND_COLOR = EGPanel.BACKGROUND_COLOR;

GPanel.simbolC = QREditor.simbolC;
GPanel.simbolT = QREditor.simbolT;

QRCode.inputD.setEditable(false);

                    if(readImage!=null && EGPanel.imgVisible){
                        QRCode.readImage=readImageBuf;
                        //QRCode.imageX=(int)(EGPanel.imageX/(float)(EGPanel.blockSize/QRShow.blockWidth));
                        //QRCode.imageY=(int)(EGPanel.imageY/(float)(EGPanel.blockSize/QRShow.blockWidth));
            QRCode.imageX=(int)(EGPanel.imageX/(float)(EGPanel.blockSize));
                        QRCode.imageY=(int)(EGPanel.imageY/(float)(EGPanel.blockSize));
                    }
                }
            //ボタン(画像だけ送る)のイベント
                if(obj==ok_imgB){
                    QRCode.ver=ver;
                    QRCode.verList.setSelectedIndex(ver);
                    if(readImage!=null && EGPanel.imgVisible){
                        QRCode.readImage=readImageBuf;
                        //QRCode.imageX=(int)(EGPanel.imageX/(float)(EGPanel.blockSize/QRShow.blockWidth));
                        //QRCode.imageY=(int)(EGPanel.imageY/(float)(EGPanel.blockSize/QRShow.blockWidth));
            QRCode.imageX=(int)(EGPanel.imageX/(float)(EGPanel.blockSize));
                        QRCode.imageY=(int)(EGPanel.imageY/(float)(EGPanel.blockSize));
                    }
                }
            //ボタン(画像 In)のイベント
                if(obj==imageInB){
                    Image readImageIm;
                    int returnVal = chooser.showOpenDialog(this);
                    try {
                        if (returnVal == JFileChooser.APPROVE_OPTION) {
                            File file = chooser.getSelectedFile();
                            try {
                                readImageBuf = ImageIO.read(file);
                    QRCode.imageSize = 0;
                                //画像サイズの調整

        //readImageIm=readImageBuf.getScaledInstance((int)(readImageBuf.getWidth()*((float)EGPanel.blockSize/QRShow.blockWidth)),  -1,
Image.SCALE_AREA_AVERAGING);

readImageIm=readImageBuf.getScaledInstance((int)(QRCode.initImageSize*((float)EGPanel.blockSize))-1, -1, Image.SCALE_AREA_AVERAGING);
                                //Image を BufferedImage に変換
                                    readImage=toBufferedImage(readImageIm);
                            } catch (Exception e2) {
                                e2.printStackTrace();
                                readImageBuf = null;
                            }
                        }
                    } catch(Exception ex){}



                    update();
                }
            //ボタン(画像埋め込み)のイベント
                if(obj==addB && readImage!=null){
                    int sp_x=0;
                    int sp_y=0;
                    int ep_x=0;
                    int ep_y=0;
                    int x=0;
                    int y=0;
                    int rgb=0;
                    //減色処理
                        readImage=imgToGray(readImage);
                    //メイン処理
                        sp_x=EGPanel.imageX/EGPanel.blockSize;
                        sp_y=EGPanel.imageY/EGPanel.blockSize;
                        ep_x=(EGPanel.imageX+readImage.getWidth())/EGPanel.blockSize;
                        ep_y=(EGPanel.imageY+readImage.getHeight())/EGPanel.blockSize;
                        for(int i=sp_x;i<=ep_x;i++){
                            for(int j=sp_y;j<=ep_y;j++){

        //System.out.println("x="+(i*EGPanel.blockSize+(EGPanel.blockSize/2)-(sp_x*EGPanel.blockSize)));

        x=i*EGPanel.blockSize+(EGPanel.blockSize/2)-(sp_x*EGPanel.blockSize);

        y=j*EGPanel.blockSize+(EGPanel.blockSize/2)-(sp_y*EGPanel.blockSize);
                                if(x>=0  &&  x<=readImage.getWidth()  &&  y>=0  &&
y<=readImage.getHeight()){

        System.out.println("getRGB="+readImage.getRGB(x,y));
                                    rgb=readImage.getRGB(x,y);
                                }else{
                                    //デバッグ用
                                    System.out.println("座標が画像のピクセル外  x="+x+"
y="+y);
                                }
                                if(rgb!=-1){
                                    EGPanel.bitTurn(i,j);
                                }
                            }
```

```
                                                                }
                                                update();
                                        }
                        //コンボボックス(型番)のイベント
                                        if(obj==verList){
                                                ver=verList.getSelectedIndex()+1;
                                                init();
                                        }
                        //コンボボックス(訂正レベル)のイベント
                                        if(obj==lvList){
                                                lv=lvList.getSelectedIndex();
                                                init();
                                        }
                        //コンボボックス(マスク)のイベント
                                        if(obj==maskList){
                                                mask=maskList.getSelectedIndex();
                                                init();
                                        }
        //イメージ透明
                        if(obj==imgTranList){
                                                imgTran = (float)imgTranList.getSelectedIndex()/100;
                                                update();
                                        }
        //明暗認識
                        if(obj==regList){
                                                EGPanel.regList = regList.getSelectedIndex();
                                //JOptionPane.showMessageDialog(null,regList.getSelectedIndex());
                                JOptionPane.showMessageDialog(null, "明暗輝度値は"+regList.getSelectedIndex()
                                        + "に設定されました。¥n"
                                        +"輝度値を設定することは一回のみ¥n"
                                        + "変更したい場合は『クリア』ボタンで全てをリセットしてくさだい。");
                                regList.setEnabled(false);
                                edit = true;
                                                update();
                                        }
        //ドット透明
                        if(obj==colorTranList){
                                                colorTran = (float)colorTranList.getSelectedIndex()/100;
                                                update();
                                        }
                        //ラジオボタン(鉛筆)のイベント
                                        if(obj==RadioB_pencil){
                                                paintMode=0;
                                        }
                        //ラジオボタン(消しゴム)のイベント
                                        if(obj==RadioB_eraser){
                                                paintMode=1;
                                        }
                        //ラジオボタン(鉛筆&消しゴム)のイベント
                                        if(obj==RadioB_both){
                                                paintMode=2;
                                        }
                        //ラジオボタン(塗りつぶし)のイベント
                                        if(obj==RadioB_fill){
                                                paintMode=3;
                                        }

                        if(obj==sizeUp){
                                if(EGPanel.imgVisible && readImage!=null){
                                        QRCode.imageSize += 1;
                                }
                                update();
                                        }

                        if(obj==sizeDown){
                                if(EGPanel.imgVisible && readImage!=null){
                                                QRCode.imageSize -= 1;
                                }
                                update();
                                        }
                        if(obj==fit){
                                if(EGPanel.imgVisible && readImage!=null){
                                        if(fitMode){
                                                fitMode = false;
                                        }
                                        else{
                                                fitMode = true;
                                                EGPanel.imageX = 0;
                                                EGPanel.imageY = 0;
                                        }
                                }
                                update();
                                        }
                        if(obj==swap){
                                if(EGPanel.imgVisible && readImage!=null){
                                        if(swapMode){
                                                swapMode = false;
                                        }
                                        else{
                                                swapMode = true;
                                        }
                                }
                                update();
                                        }
}
```

```
//////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////メソッド//////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////


//////////////////////////////////////////////////////
////////////////シンボルを初期化するメソッド////////////////
public void init (){
		simSize = 21+(4*(ver-1));
		simbol = new int[simSize][simSize];
		simbolC = new Color[simSize][simSize];
  simbolT = new float[simSize][simSize];
  setSimbolInitData();
		readImage=null;
		EGPanel.imageX=0;
		EGPanel.imageY=0;
		//画面更新
  regList.setEnabled(true);
  edit = false;
		update();
}

//////////////////////////////////////////////////////
////////////////画面を更新するメソッド////////////////////
public void update (){
		windowResize(simSize);
  try{
	Image readImageIm;
	if(!fitMode)
	{
		readImageIm=readImageBuf.getScaledInstance((int)(((QRCode.initImageSize+QRCode.imageSize)*((float)EGPanel.blockSize))-1,	-1,
Image.SCALE_AREA_AVERAGING);
	}
	else
	{
		readImageIm=readImageBuf.getScaledInstance((int)(simSize*((float)EGPanel.blockSize)),	(int)(simSize*((float)EGPanel.blockSize)),
Image.SCALE_AREA_AVERAGING);
	}
	readImage=toBufferedImage(readImageIm);
  }
  catch(Exception ex){}
		EGrap.repaint();
		EGrap.setPreferredSize(new Dimension(simSize*EGPanel.blockSize, simSize*EGPanel.blockSize));
		EGrap.revalidate();
}

//////////////////////////////////////////////////////
///////////クリックドラッグ時に実行されるメソッド//////////
public static void Clicked(int x,int y) {
		EGrap.Clicked(x,y,paintMode);
}

//////////////////////////////////////////////////////
////////ドラッグが終わったら座標をリセットするメソッド///////
public static void Released() {
		EGrap.Released();
}

//////////////////////////////////////////////////////
///////////ウインドウサイズを適正にするメソッド//////////////
public void windowResize(int simSize){
		WindowSizeX=650;
		WindowSizeY=670;
		setSize(WindowSizeX,WindowSizeY);
}

//////////////////////////////////////////////////////////////
////////////////////////出力メソッド////////////////////////////
public void data_out(){
		RSSum=versionMaxLen[ver][lv][6]+versionMaxLen[ver][lv][7];
		dataB="";
		String[] dataB8;
		//シンボルのコピーsimbol_copy を取得
			int[][] simbol_copy=new int[simSize][];
			for(int i=0;i<simSize;i++){
				simbol_copy[i]=(int[])simbol[i].clone();
			}
		//データ部分にマスクをかける
			DataBase.mask(simbol_copy,mask,simSize,2);
		//グラフィック部分のビット列を取得
			//シンボルの全ビット列抽出
				dataB=getBit(simSize,simbol_copy,0);
			//RSB が複数の時はインタリーブ取得
				if(RSSum>1){
					String[][] RSB=new String[RSSum][];
					int dBkazu=(int)(versionMaxLen[ver][lv][4]/RSSum);
					for(int i=0;i<versionMaxLen[ver][lv][6];i++){
						RSB[i]=new String[dBkazu];
					}
					for(int i=versionMaxLen[ver][lv][6];i<RSSum;i++){
						RSB[i]=new String[dBkazu+1];
					}
```

```
                                        //データビット列を 8 ビットずつに区切る
                                                dataB8=new String[dataB.length()/8];
                                                for(int i=0;i<dataB8.length;i++){
                                                        dataB8[i]=dataB.substring((i*8),(i*8)+8);
                                                }
                                        //RSB 配列に順番にデータビット列を格納していく
                                                int nowRSB=0;
                                                for(int i=0;i<dataB8.length;i++){
                                                        if(i==RSSum*dBkazu){
                                                                nowRSB=versionMaxLen[ver][lv][6];
                                                        }
                                                        RSB[nowRSB][(int)i/RSSum]=dataB8[i];
                                                        nowRSB++;
                                                        nowRSB%=RSSum;
                                                }
                                        //RSB 配列を 1 つにまとめる
                                                dataB="";
                                                for(int i=0;i<RSSum;i++){
                                                        for(int j=0;j<RSB[i].length;j++){
                                                                dataB+=RSB[i][j];
                                                        }
                                                }
                                }
                        //全ビット列からグラフィック部分のビット列のみを取得
                                dataB=dataB.substring(mode_mozisuLen);
                        //データの出力
                                outstr=Inputdata+GRAP_SEPARATOR+dataB;
        }

        ////////////////////////////////////////////////////////////////
        //////////////シンボルに初期データビットを設定するメソッド//////////
        public void setSimbolInitData(){
                        //入力データ取得
                                Inputdata=dataInputArea.getText();
                                //入力データが容量を超えていたら入力データ調整
                                        //作成中
                                encodeMode=DataBase.getEncodeMode(Inputdata);
                                String InputdataBit=DataBase.DtoB(Inputdata,encodeMode);
                        //グラフィックビット前の全ビット数を計算
                                mode_mozisuLen=4+DataBase.mozisusizisiLenGet(ver,encodeMode);
                                mode_mozisuLen+=InputdataBit.length();
                                //終端パターンの数を計算
                                        String terminator="";
                                        int terminatorLen=((versionMaxLen[ver][lv][4]*8)-mode_mozisuLen);
                                        terminatorLen=terminatorLen>4?4:terminatorLen;
                                        //デバッグ用
                                                System.out.println("終端パターン数="+terminatorLen);
                                        for(int i=0;i<terminatorLen;i++){
                                                terminator+="0";
                                                mode_mozisuLen++;
                                        }
                        //グラフィックビット前の全ビットデータを取得
                                notGrapCode=DataBase.getMode(encodeMode);
                                                                                        //データモード指示子
                                notGrapCode+=DataBase.get2(Inputdata.length(),DataBase.mozisusizisiLenGet(ver,encodeMode));                    //デ ー タ
文字数指示子
                                notGrapCode+=InputdataBit;
                                                                                                                                //データ
                                notGrapCode+=terminator;
                                                                                                                                //終端パ
ターン
                                //グラフィック部分前の全ビットデータを、シンボルに格納する形に変換
                                        dataBit=new int[versionMaxLen[ver][lv][4]*8];
                                        for(int i=0;i<notGrapCode.length();i++){
                                                if(notGrapCode.charAt(i)=='0'){
                                                        dataBit[i]=DATA_E0;
                                                }else{
                                                        dataBit[i]=DATA_E1;
                                                }
                                        }
                                        for(int i=mode_mozisuLen;i<versionMaxLen[ver][lv][4]*8;i++){
                                                dataBit[i]=DATA_0;
                                        }
                        //インタリーブ配置
                                RSSum=versionMaxLen[ver][lv][6]+versionMaxLen[ver][lv][7];
                                if(RSSum>1){
                                        String[] dataB8;
                                        String[][] RSB=new String[RSSum][];
                                        int dBkazu=(int)(versionMaxLen[ver][lv][4]/RSSum);
                                        dataB8=new String[dataBit.length/8];
                                        for(int i=0;i<dataB8.length;i++){
                                                dataB8[i]="";
                                                for(int j=0;j<8;j++){
                                                        dataB8[i]+=Integer.toString(dataBit[(i*8)+j]);
                                                }
                                        }
                                        for(int i=0;i<versionMaxLen[ver][lv][6];i++){
                                                RSB[i]=new String[dBkazu];
                                                }
                                        for(int i=versionMaxLen[ver][lv][6];i<RSSum;i++){
                                                RSB[i]=new String[dBkazu+1];
                                        }
                                        // 1 つめの RS ブロック数
                                                int tmp=0;
```

```
                                        for(int i=0;i<versionMaxLen[ver][lv][6];i++){
                                                System.out.println("1 つめの RS ブロック");
                                                dBkazu=(int)(versionMaxLen[ver][lv][4]/RSSum);
                                                RSB[i]=new String[dBkazu];
                                                for(int j=0;j<dBkazu;j++){
                                                        RSB[i][j]=dataB8[tmp++];
                                                }
                                        }
                                //2 つめの RS ブロック数
                                        for(int i=0;i<versionMaxLen[ver][lv][7];i++){
                                                System.out.println("2 つめの RS ブロック");
                                                dBkazu=(int)(versionMaxLen[ver][lv][4]/RSSum)+1;
                                                RSB[i+versionMaxLen[ver][lv][6]]=new String[dBkazu];
                                                for(int j=0;j<dBkazu;j++){
                                                        RSB[i+versionMaxLen[ver][lv][6]][j]=dataB8[tmp++];
                                                }
                                        }
                                //データを配置する順にソート
                                        dataBit=new int[versionMaxLen[ver][lv][4]*8];
                                        tmp=0;
                                        for(int i=0;i<Math.floor(versionMaxLen[ver][lv][4]/RSSum);i++){
                                                for(int j=0;j<RSB.length;j++){
                                                        for(int k=0;k<8;k++){

        dataBit[tmp++]=Integer.parseInt(String.valueOf(RSB[j][i].charAt(k)));
                                                        }
                                                }
                                        }
                                        for(int i=versionMaxLen[ver][lv][6];i<RSSum;i++){
                                                for(int k=0;k<8;k++){

        dataBit[tmp++]=Integer.parseInt(String.valueOf(RSB[i][((int)(versionMaxLen[ver][lv][4]/RSSum))].charAt(k)));
                                                }
                                        }
                                }
                        //シンボルに固定データ埋め込み
                                DataBase.Indata(ver,simSize,simbol);
                        //ビット列をシンボルに埋め込む
                                setBit(simSize,dataBit,simbol);
                        //E1,E0 のみマスクを適用
                                DataBase.mask(simbol,mask,simSize,1);
                        //形式情報を埋め込む
                                DataBase.setFormatInformation(simbol,lv,mask);
        }

        ////////////////////////////////////////////////////////
        ////////シンボルにデータビット列を挿入するメソッド////////////
        public void setBit(int simSize,int[] haitiBit,int[][] simbol){
                //初期設定
                        int direct=1;
                        int LeftOrRight=1;
                        int nowX=simSize-1;
                        int nowY=simSize-1;
                        int haitiBitLength;
                        int[] tmp=new int[2];
                //配置
                        haitiBitLength=haitiBit.length;
                        tmp[1]=0;
                        for(int i=0;tmp[1]<haitiBit.length;i++){
                                tmp[0]=0;
                                //デバッグ用
                                        if(nowX<-1){
                                                System.out.println("エラー：　データがシンボルに収まりきりません　　残りビット数："+(haitiBit.length-i-1));
                                                QRCode.info.setText(QRCode.info.getText()+"エラー：　データがシンボルに収まりきりません　　残りビット数："+(haitiBit.length-i-1)+BR);
                                        }
                                if(nowX>=0 && nowX<=simSize-1 && nowY>=0 && nowY<=simSize-1){
                                //データをそれぞれ配置
                                        if(simbol[nowY][nowX]==0){
                                                if(haitiBit[tmp[1]]==DATA_E1){
                                                        simbol[nowY][nowX]=DATA_E1;
                                                }else if(haitiBit[tmp[1]]==DATA_E0){
                                                        simbol[nowY][nowX]=DATA_E0;
                                                }else if(haitiBit[tmp[1]]==DATA_0){
                                                        simbol[nowY][nowX]=DATA_0;
                                          simbolC[nowY][nowX]=Color.WHITE;
                                          simbolT[nowY][nowX]=0.0f;
                                                }
                                                tmp[0]=1;
                                                tmp[1]++;
                                        }
                                }

                                if(tmp[0]==0){
                                //移動
                                        if(LeftOrRight==1){
                                                //右側
                                                nowX--;
                                                LeftOrRight=-LeftOrRight;
                                        }else if(nowY>=0 && nowY<=simSize-1){
                                                //上下移動可能
                                                nowY-=direct;
                                                nowX++;
```

```
                                                                        LeftOrRight=-LeftOrRight;
                                                                }else{
                                                                        //上下移動不可能
                                                                        nowX--;
                                                                        if(nowX==6){
                                                                                nowX--;
                                                                        }
                                                                        LeftOrRight=-LeftOrRight;
                                                                        nowY+=direct;
                                                                        direct=-direct;
                                                                }

                                                        }
                                                }
                                        }

                                        ////////////////////////////////////////////////////
                                        ////////シンボルからデータビット列を取り出すメソッド//////////
                                        public String getBit(int simSize,int[][] simbol,int mode){
                                                //
                                                //mode=1：グラフィック部分のビット抽出
                                                //mode=2：グラフィック以外の部分のビット抽出
                                                //mode=0：両方抽出
                                                //
                                                //初期設定
                                                        int direct=1;
                                                        int LeftOrRight=1;
                                                        int nowX=simSize-1;
                                                        int nowY=simSize-1;
                                                        int[] tmp=new int[2];
                                                        String dataB="";
                                                //配置
                                                        tmp[1]=0;
                                                        for(int i=0;tmp[1]<versionMaxLen[ver][lv][4]*8;i++){
                                                                tmp[0]=0;
                                                                //デバッグ用
                                                                        if(nowX<-1){
                                                                                //エラー
                                                                        }
                                                                if(nowX>=0 && nowX<=simSize-1 && nowY>=0 && nowY<=simSize-1){
                                                                        //データを取り出し
                                                                                if(simbol[nowY][nowX]==DATA_E1){
                                                                                        if(mode==2 || mode==0){
                                                                                                dataB+="1";
                                                                                        }
                                                                                        tmp[1]++;
                                                                                }else if(simbol[nowY][nowX]==DATA_E0){
                                                                                        if(mode==2 || mode==0){
                                                                                                dataB+="0";
                                                                                        }
                                                                                        tmp[1]++;
                                                                                }
                                                                                if(simbol[nowY][nowX]==DATA_1){
                                                                                        if(mode==1 || mode==0){
                                                                                                dataB+="1";
                                                                                        }
                                                                                        tmp[1]++;
                                                                                }else if(simbol[nowY][nowX]==DATA_0){
                                                                                        if(mode==1 || mode==0){
                                                                                                dataB+="0";
                                                                                        }
                                                                                        tmp[1]++;
                                                                                }
                                                                }
                                                                if(tmp[0]==0){
                                                                        //移動
                                                                                if(LeftOrRight==1){
                                                                                        //右側
                                                                                        nowX--;
                                                                                        LeftOrRight=-LeftOrRight;
                                                                                }else if(nowY>=0 && nowY<=simSize-1){
                                                                                        //上下移動可能
                                                                                        nowY-=direct;
                                                                                        nowX++;
                                                                                        LeftOrRight=-LeftOrRight;
                                                                                }else{
                                                                                        //上下移動不可能
                                                                                        nowX--;
                                                                                        if(nowX==6){
                                                                                                nowX--;
                                                                                        }
                                                                                        LeftOrRight=-LeftOrRight;
                                                                                        nowY+=direct;
                                                                                        direct=-direct;
                                                                                }
                                                                                if(nowX<-1){
                                                                                        System.out.println("エラー：getBit()にてシンボルを取得しきれません
でした。");

                                                                                        QRCode.info.setText(QRCode.info.getText()+"エラー:getBit()にてシン
ボルを取得しきれませんでした。"+BR);

                                                                                        break;
                                                                                }
                                                                }
                                                        }
```

```
                return dataB;
        }

        ////////////////////////////////////////////////////////
        ///////////Image を BufferedImage に変換するメソッド//////////
        static public BufferedImage toBufferedImage(Image img){
                BufferedImage bimg=null;
                try{
                        //java.awt.MediaTracker でロードを待機
                                MediaTracker tracker = new MediaTracker(new Component(){});
                                tracker.addImage(img, 0);
                                tracker.waitForAll();
                //Image を BufferedImage に変換
                                PixelGrabber pixelGrabber = new PixelGrabber(img, 0, 0, -1, -1, false);
                                pixelGrabber.grabPixels();
                                ColorModel cm = pixelGrabber.getColorModel();
                                final int w = pixelGrabber.getWidth();
                                final int h = pixelGrabber.getHeight();
                                WritableRaster raster = cm.createCompatibleWritableRaster(w, h);
                                bimg=new BufferedImage(cm,raster,cm.isAlphaPremultiplied(),new Hashtable());
                                bimg.getRaster().setDataElements(0, 0, w, h, pixelGrabber.getPixels());
                }catch(InterruptedException e){}
                return bimg;
        }

        ////////////////////////////////////////////////////////
        ////////画像をグレースケールに変換するメソッド//////////////
        public BufferedImage imgToGray(BufferedImage bimg){
                for (int iy = 0; iy < bimg.getHeight(); iy++) {
                        for (int ix = 0; ix < bimg.getWidth(); ix++) {
                                int col = bimg.getRGB(ix, iy);
                                bimg.setRGB(ix, iy, toGray(col));
                        }
                }
                return bimg;
        }

        ////////////////////////////////////////////////////////
        ////////画素をグレースケールに変換するメソッド//////////////
        public int toGray(int col){
                Color c = new Color(col);
                int max = Math.max(c.getRed(), Math.max(c.getGreen(), c.getBlue()));
                int min = Math.min(c.getRed(), Math.min(c.getGreen(), c.getBlue()));
                int a = c.getAlpha();
                int v = (max+min)/2;
                c = new Color(v, v, v, a);
                return c.getRGB();
        }

        ////////////////////////////////////////////////////////
        ////////////キーボード入力イベントメソッド/////////////////
          public void keyPressed(KeyEvent e) {
        //画像の移動操作
                System.out.println("押されたキーコード:"+e.getKeyCode());
                if(e.getKeyCode()==37){//←
                        EGPanel.imageX-=moveSpeed;
                }else if(e.getKeyCode()==39){//→
                        EGPanel.imageX+=moveSpeed;
                }else if(e.getKeyCode()==38){// ↑
                        EGPanel.imageY-=moveSpeed;
                }else if(e.getKeyCode()==40){// ↓
                        EGPanel.imageY+=moveSpeed;
                }else if(e.getKeyCode()==67){//C　画像の表示切替
                        if(EGPanel.imgVisible){
                                EGPanel.imgVisible=false;
                        }else{
                                EGPanel.imgVisible=true;
                        }
                }
                if(moveSpeed<2*EGPanel.blockSize){
                        moveSpeed+=EGPanel.blockSize;
                }
                update();
          }
          public void keyReleased(KeyEvent e) {
                moveSpeed=EGPanel.blockSize;
          }
          public void keyTyped(KeyEvent e) {
          }
}

class MyMouseListener extends MouseAdapter{
          public void mousePressed(MouseEvent e) {
        if(QREditor.edit)
                QREditor.Clicked(e.getX(),e.getY());
        else
            JOptionPane.showMessageDialog(null,"輝度値を設定してください");
          }
}

class MyMouseMotionListener implements MouseMotionListener{
          public void mouseDragged( MouseEvent e ) {
        if(QREditor.edit)
                QREditor.Clicked(e.getX(),e.getY());
        else
```

```
                JOptionPane.showMessageDialog(null,"輝度値を設定してください");
            }
            public void mouseMoved( MouseEvent e ) {
        if(QREditor.edit)
            QREditor.Released();
            }

}

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
//////////////////////////////////グラフィックパネル//////////////////////////////////
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
class EGPanel extends JPanel{
        //DataBase からデータ取得
                public static final int NO_DATA = DataBase.NO_DATA;                                     //定数(データ無)
                public static final int BLACK_DATA = DataBase.BLACK_DATA;                       //定数(黒データ)
                public static final int WHITE_DATA = DataBase.WHITE_DATA;                       //定数(白データ)
                public static final int K_DATA = DataBase.K_DATA;                                       //定数(形式、型番予定)
                public static final int DATA_1 = DataBase.DATA_1;                                       //定数(コードデータ 1)
                public static final int DATA_0 = DataBase.DATA_0;                                       //定数(コードデータ 0)
                public static final int DATA_VER = DataBase.DATA_VER;                           //定数(型番情報)
                public static final int DATA_K = DataBase.DATA_K;                                       //定数(形式情報)
                public static final int DATA_E1 = DataBase.DATA_E1;                                     //定数(エディタでの仮コ
ードデータ 1)
                public static final int DATA_E0 = DataBase.DATA_E0;                                     //定数(エディタでの仮コ
ードデータ 0)
                public static final String BR = DataBase.BR;                                            //定数(改行コード)
        //ドットのカラー
            public static Color rgbValue = Color.BLACK;
            public static double briV = 0;
            public static Color NO_DATA_COLOR = Color.BLACK;
            public static Color BLACK_DATA_COLOR = Color.BLACK;
            public static Color K_DATA_COLOR = Color.GRAY;
            public static Color DATA_COLOR = Color.BLACK;
            public static Color DATA_VER_COLOR = Color.BLACK;
            public static Color DATA_K_COLOR = Color.BLACK;
            public static Color DATA_E_COLOR = Color.BLACK;
            public static Color BACKGROUND_COLOR = Color.WHITE;
    //その他
                static int quiet=1;
                static int blockWidth=9;
                static int blockSize=blockWidth+quiet;
                static int imageX=0;
                static int imageY=0;
                static boolean imgVisible=true;
                        //画像ファイルの表示切替
                int nowX=-1;
                int nowY=-1;
            static int regList = 80;
            //static int regWhite = 80;
        //データビット列を取得
            public static float tranValue = 1.0f;
        EGPanel(){

        }

    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        if(QREditor.swapMode){
            drawImage(g);
            drawDot(g);
        }
        else{
            drawDot(g);
                //読み込み画像表示
            drawImage(g);
        }
          }
    private void drawDot(Graphics g)
    {
        //QR コード外枠描画
        Graphics2D g2d = (Graphics2D)g;
        g2d.setColor(Color.black);
        //g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER,0.5f));
        for(int i=0;i<QREditor.simSize;i++){
            for(int j=0;j<QREditor.simSize;j++){
                g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER,0.5f));
                switch(QREditor.simbol[j][i]){
                    case NO_DATA:
                        //データ未定義
                        QREditor.simbolC[j][i] = NO_DATA_COLOR;
                        //QREditor.simbolT[j][i] = tranValue;
                        g2d.setColor(NO_DATA_COLOR);
                        g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER,0.5f));
                        g2d.drawRect(i*blockSize,j*blockSize,blockWidth,blockWidth);
                        g2d.setColor(Color.LIGHT_GRAY);
                        g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER,QREditor.colorTran));
                        g2d.fillRect(i*blockSize,j*blockSize,blockWidth,blockWidth);
                        break;
                    case BLACK_DATA:
                        //黒データ
                        g2d.setColor(Color.BLUE);
```

```
                        g2d.drawRect(i*blockSize,j*blockSize,blockWidth,blockWidth);
                        g2d.setColor(BLACK_DATA_COLOR);
                        g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER,QREditor.colorTran));
                        g2d.fillRect(i*blockSize,j*blockSize,blockWidth,blockWidth);
                        break;
                case WHITE_DATA:
                        //白データ
                        g2d.setColor(Color.BLUE);
                        g2d.drawRect(i*blockSize,j*blockSize,blockWidth,blockWidth);
                        g2d.setColor(BACKGROUND_COLOR);
                        g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER,0.3f));
                        g2d.fillRect(i*blockSize,j*blockSize,blockWidth,blockWidth);
                        break;
                case K_DATA:
                        //形式、型番情報が入る予定地
                        g2d.setColor(K_DATA_COLOR);
                        g2d.drawRect(i*blockSize,j*blockSize,blockWidth,blockWidth);
                        break;
                case DATA_1:
                        //データ 1
                        g2d.setColor(Color.BLACK);
                        g2d.drawRect(i*blockSize,j*blockSize,blockWidth,blockWidth);
                        g2d.setColor(QREditor.simbolC[j][i]);
                        g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER,QREditor.simbolT[j][i]));
                        g2d.fillRect(i*blockSize,j*blockSize,blockWidth,blockWidth);
                        break;
                case DATA_0:
                        //データ 0
                        g2d.setColor(Color.BLACK);
                        g2d.drawRect(i*blockSize,j*blockSize,blockWidth,blockWidth);
                        g2d.setColor(QREditor.simbolC[j][i]);
                        g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER,QREditor.simbolT[j][i]));
                        g2d.fillRect(i*blockSize,j*blockSize,blockWidth,blockWidth);
                        //System.out.println(QREditor.simbolC[j][i].getRed());
                        break;
                case DATA_VER:
                        //型番情報
                        g2d.setColor(Color.GREEN);
                        g2d.drawRect(i*blockSize,j*blockSize,blockWidth,blockWidth);
                        g2d.setColor(DATA_VER_COLOR);
                        g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER,QREditor.colorTran));
                        g2d.fillRect(i*blockSize,j*blockSize,blockWidth,blockWidth);
                        break;
                case DATA_K:
                        //型番情報
                        g2d.setColor(Color.GRAY);
                        g2d.drawRect(i*blockSize,j*blockSize,blockWidth,blockWidth);
                        g2d.setColor(DATA_K_COLOR);
                        g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER,QREditor.colorTran));
                        g2d.fillRect(i*blockSize,j*blockSize,blockWidth,blockWidth);
                        break;
                case DATA_E1:
                        //エディタ固定データ 1
                        QREditor.simbolC[j][i] = DATA_E_COLOR;
                        //QREditor.simbolT[j][i] = tranValue;
                        g2d.setColor(Color.MAGENTA);
                        g2d.drawRect(i*blockSize,j*blockSize,blockWidth,blockWidth);
                        g2d.setColor(DATA_E_COLOR);
                        g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER,QREditor.colorTran));
                        g2d.fillRect(i*blockSize,j*blockSize,blockWidth,blockWidth);
                        break;
                case DATA_E0:
                        //エディタ固定データ 0
                        QREditor.simbolC[j][i] = BACKGROUND_COLOR;
                        //QREditor.simbolT[j][i] = tranValue;
                        g2d.setColor(Color.MAGENTA);
                        g2d.drawRect(i*blockSize,j*blockSize,blockWidth,blockWidth);
                        g2d.setColor(BACKGROUND_COLOR);
                        g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER,0.3f));
                        g2d.fillRect(i*blockSize,j*blockSize,blockWidth,blockWidth);
                        break;
                default:
                        //エラー
                        System.out.println("エラー：　シンボルのデータがおかしいです。　："+QREditor.simbol[i][j]);
                        QRCode.info.setText(QRCode.info.getText()+"エラー：　シンボルのデータがおかしいです。　："+QREditor.simbol[i][j]+BR);
            }
        }
    }
}
private void drawImage(Graphics g)
{
    if(imgVisible && QREditor.readImage!=null){
        Graphics2D g2d = (Graphics2D)g;
        g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER,QREditor.imgTran));
        g2d.drawImage(QREditor.readImage, imageX, imageY, this);
    }
}
private void drawFilter(Graphics g)
{
    //未定義
}
    public void Clicked(int x,int y,int paintMode){
            if(x>=0 && y>=0 && x<=QREditor.simSize*(blockWidth+quiet) && y<=QREditor.simSize*(blockWidth+quiet)){
                    x=(int)x/(blockWidth+quiet);
                    y=(int)y/(blockWidth+quiet);
```

```
                    if(nowX!=x || nowY!=y){
                        nowX=x;
                        nowY=y;
                        if((QREditor.simbol[y][x]==DATA_0||QREditor.simbol[y][x]==DATA_1) && paintMode==1){
                            //消しゴム&両方
                            QREditor.simbol[y][x]=DATA_0;
                    QREditor.simbolC[y][x] = BACKGROUND_COLOR;
                    QREditor.simbolT[y][x] = 0.0f;
                        }else if((QREditor.simbol[y][x]==DATA_0||QREditor.simbol[y][x]==DATA_1) && paintMode==0){
                            //鉛筆&両方
                if(briV<regList)
                {
                    QREditor.simbol[y][x]=DATA_1;
                    QREditor.simbolC[y][x] = rgbValue;
                    QREditor.simbolT[y][x] = tranValue;
                }else if(briV>regList)
                {
                    QREditor.simbol[y][x]=DATA_0;
                    QREditor.simbolC[y][x] = rgbValue;
                    QREditor.simbolT[y][x] = tranValue;
                }
                        }else if(QREditor.simbol[y][x]==DATA_0 && paintMode==3){
                            //塗りつぶし
                        }else if(QREditor.simbol[y][x]==NO_DATA&&briV<regList && paintMode==2){
                            NO_DATA_COLOR = rgbValue;
                        }
                else if(QREditor.simbol[y][x]==BLACK_DATA&&briV<regList && paintMode==2){
                            BLACK_DATA_COLOR = rgbValue;
                        }
                else if(QREditor.simbol[y][x]==K_DATA && paintMode==2){
                            //未定義
                        }
                else if(QREditor.simbol[y][x]==DATA_VER&&briV<regList && paintMode==2){
                            DATA_VER_COLOR = rgbValue;
                        }
                else if(QREditor.simbol[y][x]==DATA_K){
                            DATA_K_COLOR = rgbValue;
                        }
                else if(QREditor.simbol[y][x]==DATA_E1&&briV<regList && paintMode==2){
                            DATA_E_COLOR = rgbValue;
                        }
                /*else if((QREditor.simbol[y][x]==WHITE_DATA||QREditor.simbol[y][x]==DATA_E0)&&briV>regList && paintMode==2){
                    BACKGROUND_COLOR = rgbValue;
                            }*/
                        //未定義
                        repaint();
                    }
                }else{
                    System.out.println("枠外です");
                }
            }
            public void Released(){
               nowX=-1;
                nowY=-1;
            }
            static void bitTurn(int x,int y){
                if(x>=0 && x<QREditor.simSize && y>=0 && y<QREditor.simSize){
                    if(QREditor.simbol[y][x]==DATA_0){
                        QREditor.simbol[y][x]=DATA_1;
                    }else if(QREditor.simbol[y][x]==DATA_1){
                        QREditor.simbol[y][x]=DATA_0;
                    }
                }
            }
}
```

- ColorValueSliderControl.java に追加したコード

```
import java.awt.BorderLayout;
import java.awt.Canvas;
import java.awt.Color;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import javax.swing.JButton;

import java.awt.Dimension;

import javax.swing.JTextField;

import javax.swing.*;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JSlider;
import javax.swing.border.TitledBorder;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

public class ColorValueSliderControl extends JFrame {
    public ColorValueSliderControl() {
        getContentPane().add(new TColor());
        //setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(900, 600);
        setVisible(true);
```

```
    }
}

class TColor extends JPanel implements ActionListener {

    public static final double RV = 0.299;
    public static final double GV = 0.587;
    public static final double BV = 0.114;

    TColor.DrawingCanvas canvas = new TColor.DrawingCanvas();
    TColor.DrawingCanvas currentC = new TColor.DrawingCanvas();
    JLabel rgbValue = new JLabel("000000");
    JLabel bValue = new JLabel("0");
    JButton okB;

    JSlider sliderR, sliderG, sliderB, sliderH, sliderS, sliderA,
          sliderAlpha;

    JTextField RVTB;
    JTextField GVTB;
    JTextField BVTB;

    JTextField HVTB;
    JTextField SVTB;
    JTextField AVTB;

    public TColor() {
       sliderR = getSlider(0, 255, 0, 50, 5);
       sliderG = getSlider(0, 255, 0, 50, 5);
       sliderB = getSlider(0, 255, 0, 50, 5);
       sliderH = getSlider(0, 360, 0, 120, 60);
       sliderS = getSlider(0, 255, 0, 50, 5);
       sliderA = getSlider(0, 255, 0, 50, 5);
       sliderAlpha = getSlider(0, 100, 100, 20, 10);

       RVTB = new JTextField("0");
       GVTB = new JTextField("0");
       BVTB = new JTextField("0");

       HVTB = new JTextField("0");
       SVTB = new JTextField("0");
       AVTB = new JTextField("0");
       JPanel panel = new JPanel();
       //OK ボタン
       okB=new JButton("OK");
       okB.addActionListener(this);

       panel.setLayout(new GridLayout(6, 2, 15, 0));

       panel.add(new JLabel("R-G-B Sliders (0 - 255)"));
       panel.add(new JLabel("RGB Value"));
       panel.add(new JLabel("H-S-A Sliders (ex-1)"));
       panel.add(new JLabel("HSA Value"));
       panel.add(sliderR);
       panel.add(RVTB);
       RVTB.addKeyListener(new KeyAdapter() {
                public void keyReleased(KeyEvent e) {
                     int key=e.getKeyCode();
                     if(e.getSource()==RVTB) {
                          if(key==KeyEvent.VK_ENTER) {
                               try{
                                    int r = Integer.parseInt(RVTB.getText());
                                    if(0<=r||r<=255){
                                    sliderR.setValue(r);
                                    }else if(r>255){
                                         RVTB.setText("255");
                                    }
                               }
                               catch(Exception ex)
                               {
                                    sliderR.setValue(0);
                                    RVTB.setText("0");
                               }          //System.out.print("");
                          }
                     }
                }

                public void keyTyped(KeyEvent e) {

                }

                public void keyPressed(KeyEvent e) {

                }
           });
       panel.add(sliderH);
       panel.add(HVTB);
       HVTB.addKeyListener(new KeyAdapter() {
                public void keyReleased(KeyEvent e) {
                      int key=e.getKeyCode();
                      if(e.getSource()==HVTB) {
                           if(key==KeyEvent.VK_ENTER) {
                               try{
                                    int h = Integer.parseInt(HVTB.getText());
                                    if(0<=h||h<=360){
                                    sliderH.setValue(h);
                                    }else if(h>360){
                                         HVTB.setText("360");
                                    }
                               }
```

```
                        catch(Exception ex)
                        {
                            sliderH.setValue(0);
                            HVTB.setText("0");
                        }           //System.out.print("");
                    }
                }
            }

            public void keyTyped(KeyEvent e) {

            }

            public void keyPressed(KeyEvent e) {

            }
        });
panel.add(sliderG);
panel.add(GVTB);
GVTB.addKeyListener(new KeyAdapter() {
            public void keyReleased(KeyEvent e) {
                int key=e.getKeyCode();
                if(e.getSource()==GVTB) {
                    if(key==KeyEvent.VK_ENTER) {
                        try{
                            int g = Integer.parseInt(GVTB.getText());
                            if(0<=g||g<=255){
                            sliderG.setValue(g);
                            }else if(g>255){
                                GVTB.setText("255");
                            }
                        }
                        catch(Exception ex)
                        {
                            sliderG.setValue(0);
                            GVTB.setText("0");
                        }           //System.out.print("");
                    }
                }
            }

            public void keyTyped(KeyEvent e) {

            }

            public void keyPressed(KeyEvent e) {

            }
        });
panel.add(sliderS);
panel.add(SVTB);
SVTB.addKeyListener(new KeyAdapter() {
            public void keyReleased(KeyEvent e) {
                int key=e.getKeyCode();
                if(e.getSource()==SVTB) {
                    if(key==KeyEvent.VK_ENTER) {
                        try{
                            int s = Integer.parseInt(SVTB.getText());
                            if(0<=s||s<=100){
                            sliderS.setValue(s);
                            }else if(s>100){
                                SVTB.setText("100");
                            }
                        }
                        catch(Exception ex)
                        {
                            sliderS.setValue(0);
                            SVTB.setText("0");
                        }           //System.out.print("");
                    }
                }
            }

            public void keyTyped(KeyEvent e) {

            }

            public void keyPressed(KeyEvent e) {

            }
        });
panel.add(sliderB);
panel.add(BVTB);
BVTB.addKeyListener(new KeyAdapter() {
            public void keyReleased(KeyEvent e) {
                int key=e.getKeyCode();
                if(e.getSource()==BVTB) {
                    if(key==KeyEvent.VK_ENTER) {
                        try{
                            int b = Integer.parseInt(BVTB.getText());
                            if(0<=b||b<=255){
                            sliderB.setValue(b);
                            }else if(b>255){
                                BVTB.setText("255");
                            }
                        }
                        catch(Exception ex)
                        {
                            sliderB.setValue(0);
                            BVTB.setText("0");
```

```
                        }           //System.out.print("");
                    }
                }
            }

            public void keyTyped(KeyEvent e) {

            }

            public void keyPressed(KeyEvent e) {

            }
        });
    panel.add(sliderA);
    panel.add(AVTB);
    AVTB.addKeyListener(new KeyAdapter() {
            public void keyReleased(KeyEvent e) {
                int key=e.getKeyCode();
                if(e.getSource()==AVTB) {
                    if(key==KeyEvent.VK_ENTER) {
                        try{
                            int a = Integer.parseInt(AVTB.getText());
                            if(0<=a||a<=100){
                            sliderA.setValue(a);
                            }else if(a>100){
                                AVTB.setText("100");
                            }
                        }
                        catch(Exception ex)
                        {
                            sliderA.setValue(0);
                            AVTB.setText("0");
                        }           //System.out.print("");
                    }
                }
            }

            public void keyTyped(KeyEvent e) {

            }

            public void keyPressed(KeyEvent e) {

            }
        });

    panel.add(new JLabel("Alpha Adjustment (0 - 100): ", JLabel.RIGHT));
    panel.add(sliderAlpha);

    panel.add(new JLabel("RGB 値: ", JLabel.RIGHT));

    rgbValue.setBackground(Color.white);
    rgbValue.setForeground(Color.black);
    rgbValue.setOpaque(true);
    panel.add(rgbValue);

    panel.add(new JLabel("輝度値  (MAX:100%): ", JLabel.RIGHT));

    bValue.setBackground(Color.white);
    bValue.setForeground(Color.black);
    bValue.setOpaque(true);
    panel.add(bValue);

    add(panel, BorderLayout.SOUTH);
    add(currentC);
    add(new JLabel("現在の色"));
    add(canvas);
    add(new JLabel("選択色"));
    add(okB);
}
////////////////////////////////////////////イベントメソッド//////////////////////////////////////
            public void actionPerformed(ActionEvent e) {
            Object obj = e.getSource();
        //ボタン(OK)のイベント
                                        if(obj==okB){
                                    double briT;
                                    currentC.setBackground(canvas.color);
                                    currentC.repaint();
                                    EGPanel.rgbValue = canvas.color;
                                    briT = Double.parseDouble(bValue.getText());
                                    EGPanel.briV = briT;
                                    EGPanel.tranValue = sliderAlpha.getValue()/100.0f;
                                    /*if(briT>50&&briT<90){
                                        if(briT>80){
                                            JOptionPane.showMessageDialog(null, "あなたが選択した色は“明”と設定されます"
                                                    + "があるソフトで読み取れない事もありますのでご注意ください。¥n"
                                                    + "安全の 90%以上の輝度値の“明”がおすすめです。");
                                        }else if(briT<60){
                                            JOptionPane.showMessageDialog(null, "あなたが選択した色は“暗”と設定されます"
                                                    + "があるソフトで読み取れない事もありますのでご注意ください。¥n"
                                                    + "安全の 50%以下の輝度値の“暗”がおすすめです。");
                                        }else{
                                            JOptionPane.showMessageDialog(null, "あなたが選択した色は“明”また“暗”と認識"
                                                    + "できないようです。¥n"
                                                    + "安全の輝度値は 90%以上の“明”また 50%以下の“暗”を選んでください。");
                                        }
                                    }*/
                                        }
```

```
        }

public JSlider getSlider(int min, int max, int init, int mjrTkSp, int mnrTkSp) {
    JSlider slider = new JSlider(JSlider.HORIZONTAL, min, max, init);
    slider.setPaintTicks(true);
    slider.setMajorTickSpacing(mjrTkSp);
    slider.setMinorTickSpacing(mnrTkSp);
    slider.setPaintLabels(true);
    slider.addChangeListener(new TColor.SliderListener());
    return slider;
}

class DrawingCanvas extends Canvas {
    Color color;
    int redValue, greenValue, blueValue;
    int alphaValue = 255;
    float[] hsbValues = new float[3];
    int mode = 0;

    public DrawingCanvas() {
        setSize(100, 100);
        color = new Color(0, 0, 0);
        setBackgroundColor();
    }

    public void setBackgroundColor() {
        color = new Color(redValue, greenValue, blueValue, alphaValue);
        setBackground(color);
    }
}

class SliderListener implements ChangeListener {
    public void stateChanged(ChangeEvent e) {
        JSlider slider = (JSlider) e.getSource();
        if (slider == sliderAlpha) {

            //canvas.alphaValue = slider.getValue();
            //canvas.setBackgroundColor();
        } else if (slider == sliderR) {
            canvas.redValue = slider.getValue();
            RVTB.setText(Integer.toString(canvas.redValue));
            if(canvas.mode != 2){
                displayRGBColor();
            }
        } else if (slider == sliderG) {
            canvas.greenValue = slider.getValue();
            GVTB.setText(Integer.toString(canvas.greenValue));
            if(canvas.mode != 2){
                displayRGBColor();
            }
        } else if (slider == sliderB) {
            canvas.blueValue = slider.getValue();
            BVTB.setText(Integer.toString(canvas.blueValue));
            if(canvas.mode != 2){
                displayRGBColor();
            }
        } else if (slider == sliderH) {
            canvas.hsbValues[0] = (float) (slider.getValue() / 360.0);
            HVTB.setText(Integer.toString((int) (canvas.hsbValues[0] * 360)));
            if(canvas.mode != 1){
                displayHSBColor();
            }
        } else if (slider == sliderS) {
            canvas.hsbValues[1] = (float) (slider.getValue() / 255.0);
            SVTB.setText(Integer.toString((int) (canvas.hsbValues[1] * 255)));
            if(canvas.mode != 1){
                displayHSBColor();
            }
        } else if (slider == sliderA) {
            canvas.hsbValues[2] = (float) (slider.getValue() / 255.0);
            AVTB.setText(Integer.toString((int) (canvas.hsbValues[2] * 255)));
            if(canvas.mode != 1){
                displayHSBColor();
            }
        }
        double briV = (((double)canvas.redValue*RV)+((double)canvas.greenValue*GV) + ((double)canvas.blueValue*BV))/2.55;
        rgbValue.setText(Integer.toString(canvas.color.getRGB() & 0xffffff, 16));
        bValue.setText(Float.toString((float)briV));
        canvas.repaint();
    }

    public void displayRGBColor() {
        canvas.mode = 1;
        canvas.setBackgroundColor();
        Color.RGBtoHSB(canvas.redValue, canvas.greenValue, canvas.blueValue,canvas.hsbValues);
        sliderH.setValue((int) (canvas.hsbValues[0] * 360));
        sliderS.setValue((int) (canvas.hsbValues[1] * 255));
        sliderA.setValue((int) (canvas.hsbValues[2] * 255));

        canvas.mode = 0;
    }

    public void displayHSBColor() {
        canvas.mode = 2;
        canvas.color = Color.getHSBColor(canvas.hsbValues[0],
                canvas.hsbValues[1], canvas.hsbValues[2]);
        canvas.redValue = canvas.color.getRed();
        canvas.greenValue = canvas.color.getGreen();
        canvas.blueValue = canvas.color.getBlue();
```

```
                    sliderR.setValue(canvas.redValue);
                    sliderG.setValue(canvas.greenValue);
                    sliderB.setValue(canvas.blueValue);

                    canvas.color = new Color(canvas.redValue, canvas.greenValue,
                            canvas.blueValue, canvas.alphaValue);
                    canvas.setBackground(canvas.color);
                    canvas.mode = 0;
                }
            }
}
```