高知大学教育学部の情報数学のテキスト

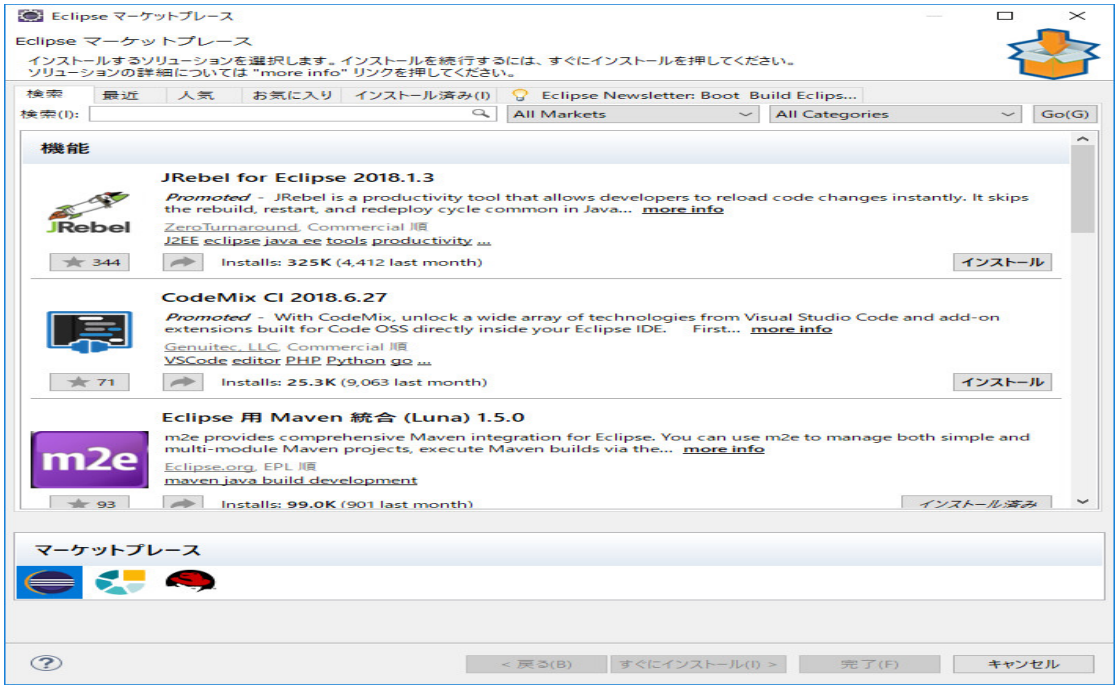
文責　：　高知大学名誉教授　中村　治

数独のヒントを表示してくれるプログラム (Java/FX 版)

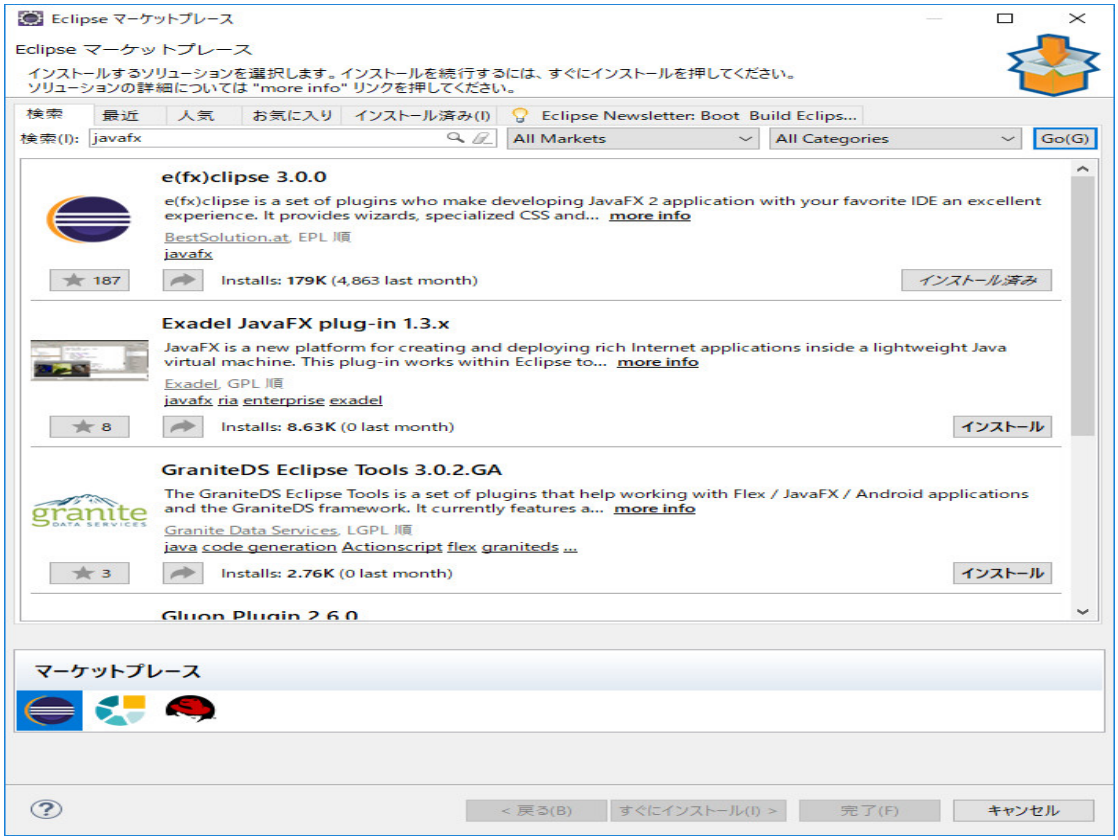
数独の解法のプログラムが載っている Java の本として、棚床弘樹著「鉛筆パズルゲームプログラミング　ナンバープレス、お絵かきパズル、ナンバークロスワードのアルゴリズム」ソフトバンククリエイティブ　２００７年がありましたが、今は絶版で、改訂版は出ていません。古い本を取り出し、添付の CD のプログラムをコンパイルしてみましたが、現在の Java version 10.0.1 では実行できません。Java は互換性がないみたいです。使われているアルゴリズム・アイデアだけ本から読み取れば良いですが、プログラムを実際に実行できなければ、初心者にはつらい作業になります。

VC++ や Python や Ruby で「数独のヒントを表示してくれるプログラム」を作ってきたので、Java でも同じようなものを作ってみます。Java でプログラミングするには、初心者は Eclipse を使うのが良いと思います。Java が出たばかりの時から色々な本を読んできましたが、最近、勉強しなおすために、立木秀樹、有賀妙子著「すべての人のための Java プログラミング　第 3 版　Java for Everyone」共立出版を読んでみました。この本の第 2 版を使って 10 年ぐらい前に授業をしたことがありますが、プログラミングを初めて学ぶ学生さんが独学で学ぶには少しきついかとも思いますが、数学を勉強している学生さん達には、扱っている例が面白くて、Java の概要を理解するのには、良い本だと思います。「情報数学」の講義を随分長くやってきましたが、この教科書を使ったときだけ、もっとプログラミングを勉強したいから休講になっている「情報数学特講」を開講してほしいと数名の学生さんが言ってき、ゲームや Midi のプログラミングの集中講義をしました。このような本で Java の概要を理解したら、足らないことはインターネットで検索すれば良いです。

JavaFX でグラフィックスを扱うには、Shape クラスを使う方法と Canvas クラスを使う方法と二種類ありますが、ここでは Canvas クラスを使う方法でプログラミングします。まず、Eclipse の「ヘルプ」メニューの「Eclipse マーケットプレース（M）...」をクリックする。



「javafx」で検索します。



最初の「e(fx)clipse 3.0.0」をインストールします。Eclipse を再起動すれば、JavaFX が使えるよ

うになります。
　Java のプログラミングは、クラスを作ることから始めます。Eclispe の「新規」の「クラス」で、「Sudoku」として、ひな型を作り、絵を描くプログラムのひな型は次のように修正します。

```
import javafx.application.Application;
public class Sudoku extends Application {

  public static void main(String... args) {
  }
}
```

Sudoku は Application を継承し、Application を import します。クラス Sudoku の中身は

```
public class Sudoku extends Application {
  Canvas canvas;
  int board_size = 500;
  @Override
  public void start(Stage pstage) {
    Pane root = new Pane();
    canvas = new Canvas(board_size, board_size);
    root.getChildren().add(canvas);
    drawCanvas();
    Scene scene = new Scene(root);
    pstage.setTitle("数独");
    pstage.setScene(scene);
    pstage.show();
  }

  void drawCanvas() {
  }

  public static void main(String... args) {
    launch(args);
  }
}
```

のようにします。Canvas を保持する変数 canvas と canvas のサイズを保持する `board_size` を宣言し、関数 start(Stage pstage) をオーバーロードします。start(Stage pstage) の基本形は上のようにします。単にキャンバスに絵を描くにはこのようにプログラミングすれば良いです。Python や Ruby と比べると複雑ですが、いつでもこのようにプログラミングすれば良いです。

```
    canvas = new Canvas(board_size, board_size);
```

で、キャンバスのサイズをセットしています。
　関数 drawCanvas() に描きたい絵を描写するプログラムを記述します。ここでは

```
  void drawCanvas() {
    int k = 9;
    int s = board_size / (k+2);
    int w = (board_size-s*9)/2;
    int h = (board_size-s*9)/2;
    GraphicsContext gc = canvas.getGraphicsContext2D();
    gc.setStroke(Color.BLUE);
    for (int i=0; i<=9; i++) {
      if (i % 3 == 0) {
        gc.setLineWidth(2);
        gc.strokeLine(w, h+i*s, w+9*s, h+i*s);
      } else {
        gc.setLineWidth(1);
        gc.strokeLine(w, h+i*s, w+9*s, h+i*s);
      }
    }
    for (int i=0; i<=9; i++) {
      if (i % 3 == 0) {
        gc.setLineWidth(2);
        gc.strokeLine(w+i*s, h, w+i*s, h+9*s);
      } else {
        gc.setLineWidth(1);
        gc.strokeLine(w+i*s, h, w+i*s, h+9*s);
      }
    }
  }
```

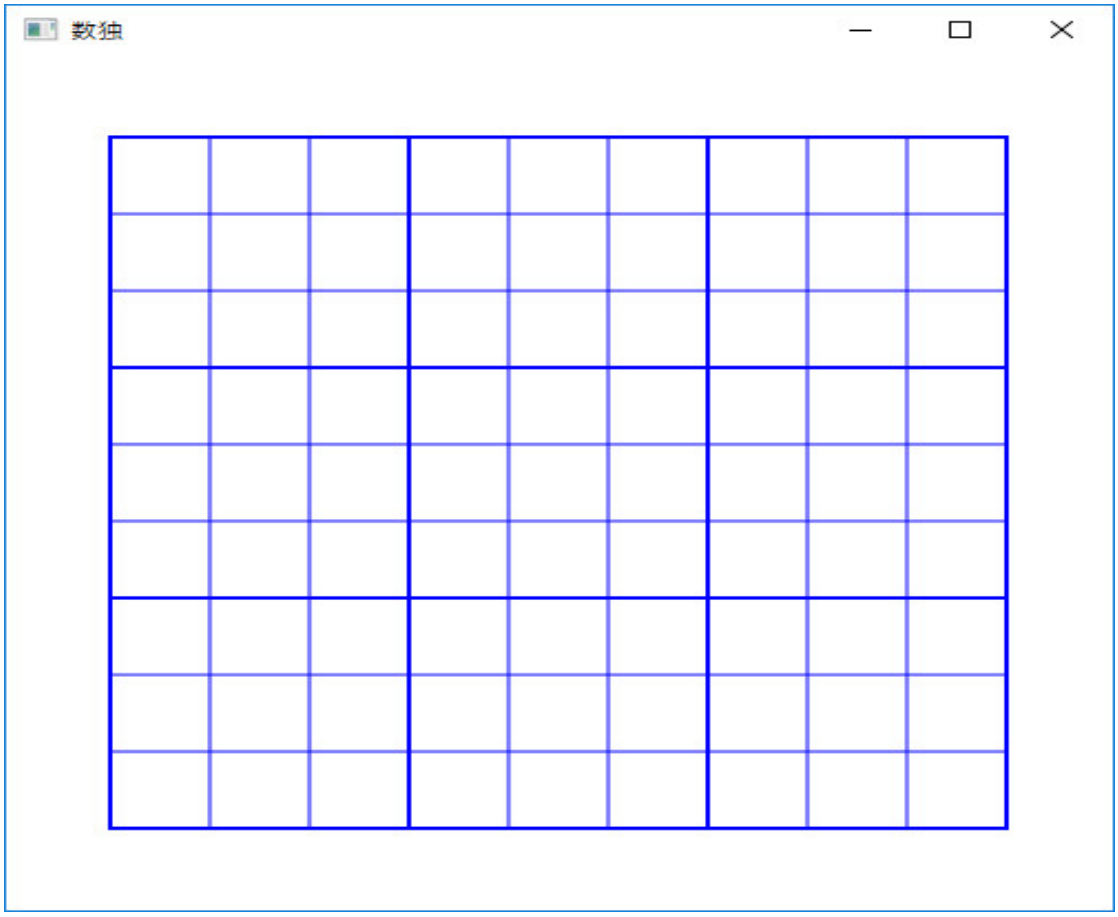
とします。セルのサイズを計算し、直線を引いて、数独の盤を描いています。

従って、プログラムの全体は

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.stage.Stage;

public class Sudoku extends Application {
  Canvas canvas;
  int board_size = 500;
  @Override
  public void start(Stage pstage) {
    Pane root = new Pane();
```

```
    canvas = new Canvas(board_size, board_size);
    root.getChildren().add(canvas);
    drawCanvas();
    Scene scene = new Scene(root);
    pstage.setTitle("数独");
    pstage.setScene(scene);
    pstage.show();
  }

  void drawCanvas() {
    int k = 9;
    int s = board_size / (k+2);
    int w = (board_size-s*9)/2;
    int h = (board_size-s*9)/2;
    GraphicsContext gc = canvas.getGraphicsContext2D();
    gc.setStroke(Color.BLUE);
    for (int i=0; i<=9; i++) {
      if (i % 3 == 0) {
        gc.setLineWidth(2);
        gc.strokeLine(w, h+i*s, w+9*s, h+i*s);
      } else {
        gc.setLineWidth(1);
        gc.strokeLine(w, h+i*s, w+9*s, h+i*s);
      }
    }
    for (int i=0; i<=9; i++) {
      if (i % 3 == 0) {
        gc.setLineWidth(2);
        gc.strokeLine(w+i*s, h, w+i*s, h+9*s);
      } else {
        gc.setLineWidth(1);
        gc.strokeLine(w+i*s, h, w+i*s, h+9*s);
      }
    }
  }

  public static void main(String... args) {
    launch(args);
  }
}
```

です。実行すると



です。

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
```

は

```
import javafx.application.Application;
```

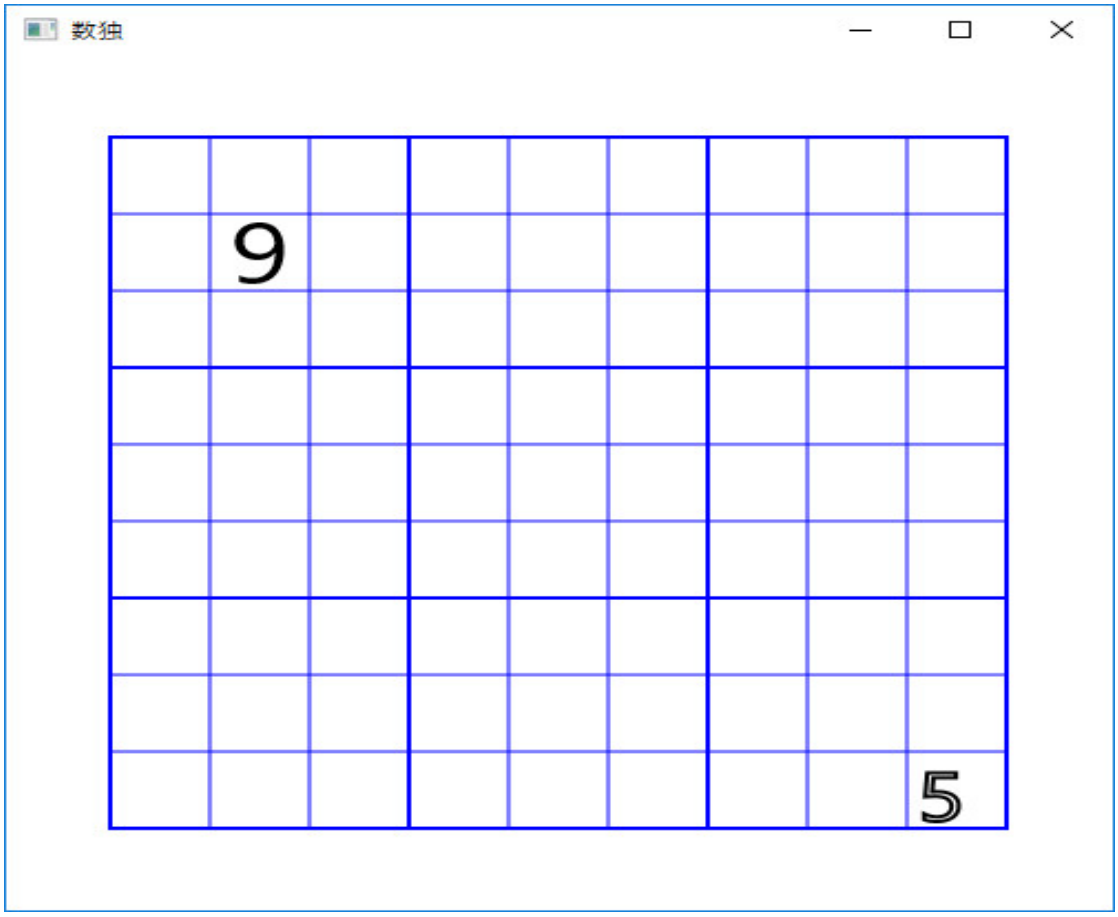
だけ書けば、後は必要な import を Eclipse が見つけてくれます。

次に、セルに数字を表示しましょう。関数 drawCanvas() の最後に

```
    int i = 1, j = 1;
    int n = 9;
    gc.setFont(new Font("courier", 50));
    gc.setStroke(Color.BLACK);
    gc.setTextAlign(TextAlignment.CENTER);
```

```
    gc.fillText(String.valueOf(n), w+(i+0.5)*s, h+(j+0.9)*s);
    i = 8;
    j = 8;
    n = 5;
    gc.setFont(new Font("ＭＳ　Ｐゴシック", 40));
    gc.setTextAlign(TextAlignment.LEFT);
    gc.strokeText(String.valueOf(n), w+(i+0.1)*s, h+(j+0.9)*s);
```

を追加します。実行すると



です。

```
    gc.strokeText(String.valueOf(n), w+(i+0.1)*s, h+(j+0.9)*s);
```

と gc.strokeText() を使うと、中抜きの文字になります。

```
    gc.setTextAlign(TextAlignment.LEFT);
```

とするとデフォルトで、座標が文字列の最初の文字の左下隅の座標になります。

```
    gc.setTextAlign(TextAlignment.CENTER);
```

とすると x 座標が文字列の中央の座標で、y 座標は文字列の下端の座標になります。

```
    gc.setFont(new Font("courier", 50));
    gc.setStroke(Color.BLACK);
```

で、フォントと文字色を支持できます。

数字の表示方法が分かりました。数独の問題の数字の配置は変数 ban にセットしておいて、使います。ban は

```
int[][] ban = {
{0,1,0,0,0,0,9,0,3},
{0,9,0,0,0,7,0,0,0},
{0,0,0,0,0,0,0,7,1},
{6,0,9,0,0,0,0,0,0},
{0,0,7,6,0,0,0,0,0},
{2,0,0,8,0,5,0,0,0},
{0,0,4,0,0,8,0,0,0},
{0,3,0,0,2,0,0,0,0},
{0,0,8,5,3,0,0,9,4}};
```

のように、2 次元配列で数独の問題の数字の配置を表すことにします。関数 drawCanvas() の最後を

```
    gc.setFont(new Font("courier", 50));
    gc.setStroke(Color.BLACK);
    gc.setTextAlign(TextAlignment.CENTER);
    for (int j=0; j<9; j++) {
      for (int i=0; i<9; i++) {
        if (ban[j][i] >0) {
          gc.fillText(String.valueOf(ban[j][i]), w+(i+0.5)*s, h+(j+0.9)*s);
        }
      }
    }
```

と書き直します。全体のプログラムは

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.text.TextAlignment;
import javafx.stage.Stage;

public class Sudoku extends Application {
  Canvas canvas;
  int board_size = 500;
```

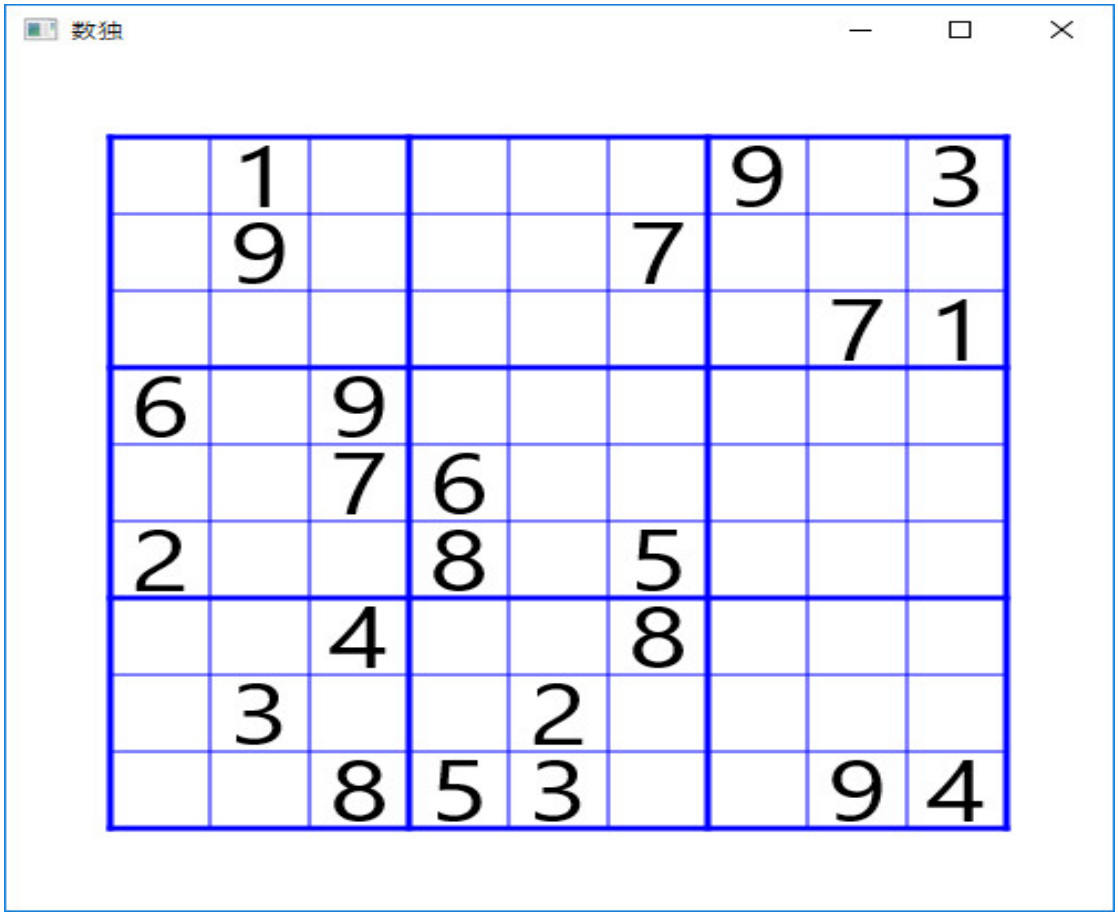
```
int[][] ban = {
  {0,1,0,0,0,0,9,0,3},
  {0,9,0,0,0,7,0,0,0},
  {0,0,0,0,0,0,0,7,1},
  {6,0,9,0,0,0,0,0,0},
  {0,0,7,6,0,0,0,0,0},
  {2,0,0,8,0,5,0,0,0},
  {0,0,4,0,0,8,0,0,0},
  {0,3,0,0,2,0,0,0,0},
  {0,0,8,5,3,0,0,9,4}};

@Override
public void start(Stage pstage) {
  Pane root = new Pane();
  canvas = new Canvas(board_size, board_size);
  root.getChildren().add(canvas);
  drawCanvas();
  Scene scene = new Scene(root);
  pstage.setTitle("数独");
  pstage.setScene(scene);
  pstage.show();
}

void drawCanvas() {
  int k = 9;
  int s = board_size / (k+2);
  int w = (board_size-s*9)/2;
  int h = (board_size-s*9)/2;

  GraphicsContext gc = canvas.getGraphicsContext2D();
  gc.setStroke(Color.BLUE);
  for (int i=0; i<=9; i++) {
    if (i % 3 == 0) {
      gc.setLineWidth(3);
      gc.strokeLine(w, h+i*s, w+9*s, h+i*s);
    } else {
      gc.setLineWidth(1);
      gc.strokeLine(w, h+i*s, w+9*s, h+i*s);
    }
  }
  for (int i=0; i<=9; i++) {
    if (i % 3 == 0) {
      gc.setLineWidth(3);
```

```
        gc.strokeLine(w+i*s, h, w+i*s, h+9*s);
      } else {
        gc.setLineWidth(1);
        gc.strokeLine(w+i*s, h, w+i*s, h+9*s);
      }
    }
    gc.setFont(new Font("courier", 50));
    gc.setStroke(Color.BLACK);
    gc.setTextAlign(TextAlignment.CENTER);
    for (int j=0; j<9; j++) {
      for (int i=0; i<9; i++) {
        if (ban[j][i] >0) {
          gc.fillText(String.valueOf(ban[j][i]), w+(i+0.5)*s, h+(j+0.9)*s);
        }
      }
    }
  }

  public static void main(String... args) {
    launch(args);
  }
}
```

です。但し、

```
      if (i % 3 == 0) {
        gc.setLineWidth(3);
        gc.strokeLine(w, h+i*s, w+9*s, h+i*s);
      } else {
        gc.setLineWidth(1);
        gc.strokeLine(w, h+i*s, w+9*s, h+i*s);
      }
```

とブロックの境界の線幅を大きくしました。
実行すると



です。色々な問題を表示できるようにします。そのためにはメニューを作り、問題を選べるようにします。まず、メニューを作ります。関数 start(Stage pstage) を次のように作り直します。

```
public void start(Stage pstage) {
  MenuBar bar = new MenuBar();
  Menu m1 = new Menu("Sample");
  MenuItem sample1 = new MenuItem("Sample1");
  MenuItem sample2 = new MenuItem("Sample2");
  m1.getItems().addAll(sample1, sample2);
  bar.getMenus().add(m1);
  VBox root = new VBox();
  canvas = new Canvas(board_size, board_size);
  root.getChildren().addAll(bar, canvas);
  drawCanvas();
  Scene scene = new Scene(root);
  pstage.setTitle("数独");
  pstage.setScene(scene);
  pstage.show();
}
```

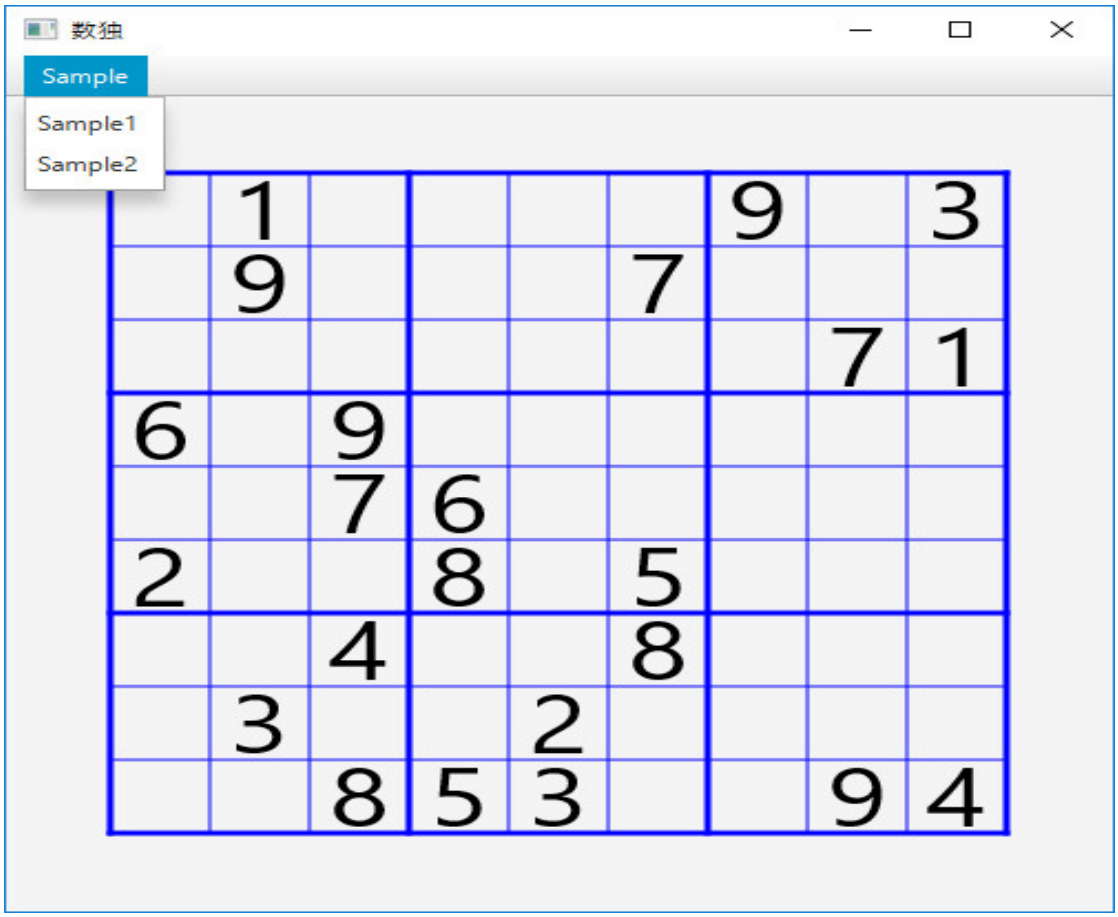
ここで、

```
    Pane root = new Pane();
```

を

```
    VBox root = new VBox();
```

に変えています。実行すると



です。次に、メニューをクリックすれば、問題を表示するようにします。関数 start(Stage pstage) の最後に

```
    sample1.setOnAction((event)->{
      ban = ban1;
      drawCanvas();
    });
    sample2.setOnAction((event)->{
      ban = ban2;
      drawCanvas();
    });
```

を追加し、クラス Sudoku の最初を

```
  int[][] ban = {
    {0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0}
  };
  int[][] ban1 = {
    {0,1,0,0,0,0,9,0,3},
    {0,9,0,0,0,7,0,0,0},
    {0,0,0,0,0,0,0,7,1},
    {6,0,9,0,0,0,0,0,0},
    {0,0,7,6,0,0,0,0,0},
    {2,0,0,8,0,5,0,0,0},
    {0,0,4,0,0,8,0,0,0},
    {0,3,0,0,2,0,0,0,0},
    {0,0,8,5,3,0,0,9,4}
  };
  int[][] ban2 = {
    {0,0,0,0,0,6,2,0,1},
    {0,0,0,0,4,0,3,7,0},
    {0,0,3,0,9,2,0,0,6},
    {4,0,5,0,0,0,0,6,0},
    {0,0,0,0,7,0,0,0,0},
    {0,2,0,0,0,0,4,0,5},
    {9,0,0,4,5,0,1,0,0},
    {0,5,4,0,2,0,0,0,0},
    {7,0,1,9,0,0,0,0,0}
  };
```

と修正します。関数 drawCanvas() の 2 行目に

```
    gc.clearRect(0,  0,  board_size,  board_size);
```

を追加します。

プログラムの全体は

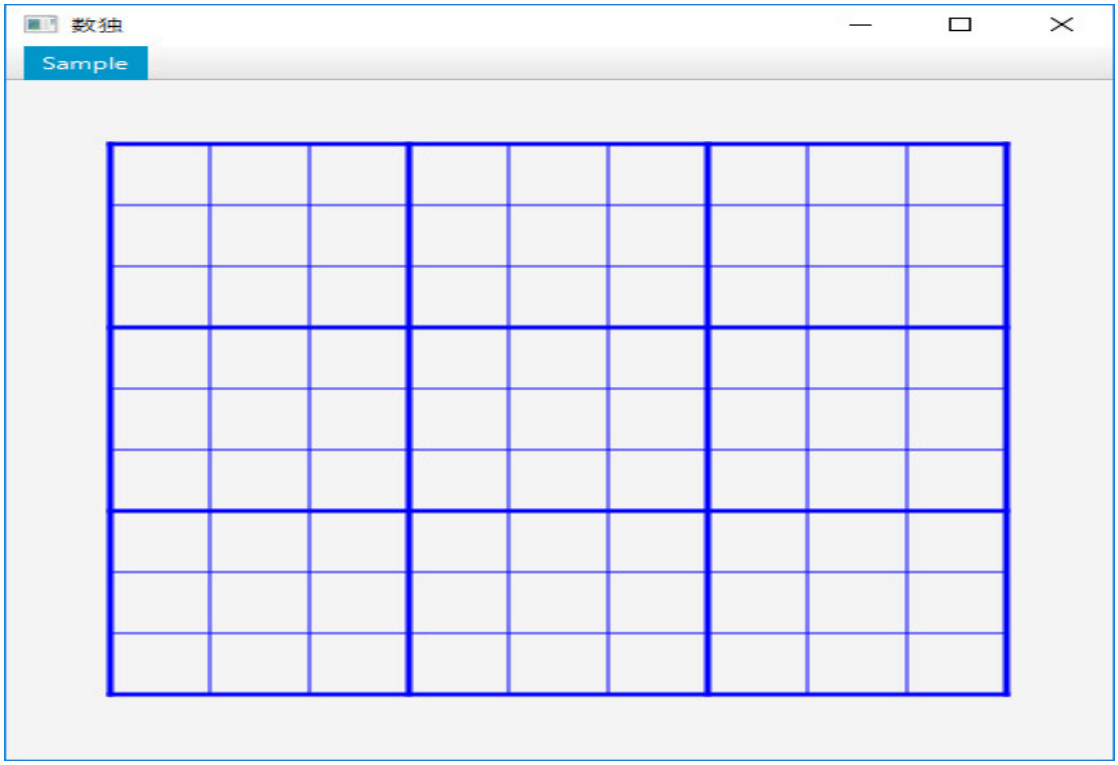
```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
```

```
import javafx.scene.control.Menu;
import javafx.scene.control.MenuBar;
import javafx.scene.control.MenuItem;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.text.TextAlignment;
import javafx.stage.Stage;

public class Sudoku extends Application {
  Canvas canvas;
  int board_size = 500;
  int[][] ban = {
    {0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0}
  };
  int[][] ban1 = {
    {0,1,0,0,0,0,9,0,3},
    {0,9,0,0,0,7,0,0,0},
    {0,0,0,0,0,0,0,7,1},
    {6,0,9,0,0,0,0,0,0},
    {0,0,7,6,0,0,0,0,0},
    {2,0,0,8,0,5,0,0,0},
    {0,0,4,0,0,8,0,0,0},
    {0,3,0,0,2,0,0,0,0},
    {0,0,8,5,3,0,0,9,4}
  };
  int[][] ban2 = {
    {0,0,0,0,0,6,2,0,1},
    {0,0,0,0,4,0,3,7,0},
    {0,0,3,0,9,2,0,0,6},
    {4,0,5,0,0,0,0,6,0},
    {0,0,0,0,7,0,0,0,0},
    {0,2,0,0,0,0,4,0,5},
    {9,0,0,4,5,0,1,0,0},
    {0,5,4,0,2,0,0,0,0},
```

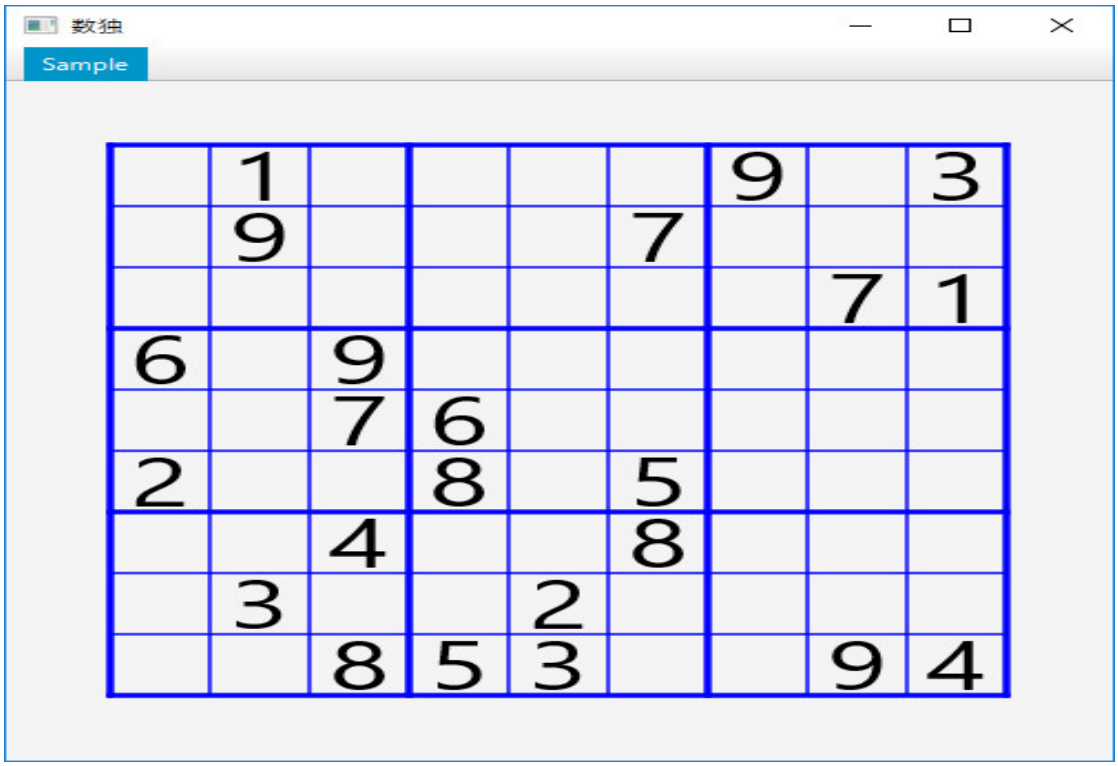
```
  {7,0,1,9,0,0,0,0,0}
};

@Override
public void start(Stage pstage) {
  MenuBar bar = new MenuBar();
  Menu m1 = new Menu("Sample");
  MenuItem sample1 = new MenuItem("Sample1");
  MenuItem sample2 = new MenuItem("Sample2");
  m1.getItems().addAll(sample1, sample2);
  bar.getMenus().add(m1);
  VBox root = new VBox();
  canvas = new Canvas(board_size, board_size);
  root.getChildren().addAll(bar, canvas);
  drawCanvas();
  Scene scene = new Scene(root);
  pstage.setTitle("数独");
  pstage.setScene(scene);
  pstage.show();

  sample1.setOnAction((event)->{
    ban = ban1;
    drawCanvas();
  });
  sample2.setOnAction((event)->{
    ban = ban2;
    drawCanvas();
  });
}
void drawCanvas() {
  int k = 9;
  int s = board_size / (k+2);
  int w = (board_size-s*9)/2;
  int h = (board_size-s*9)/2;

  GraphicsContext gc = canvas.getGraphicsContext2D();
  gc.clearRect(0,  0,  board_size,  board_size);
  gc.setStroke(Color.BLUE);
  for (int i=0; i<=9; i++) {
    if (i % 3 == 0) {
      gc.setLineWidth(3);
      gc.strokeLine(w, h+i*s, w+9*s, h+i*s);
    } else {
```

```
        gc.setLineWidth(1);
        gc.strokeLine(w, h+i*s, w+9*s, h+i*s);
      }
    }
    for (int i=0; i<=9; i++) {
      if (i % 3 == 0) {
        gc.setLineWidth(3);
        gc.strokeLine(w+i*s, h, w+i*s, h+9*s);
      } else {
        gc.setLineWidth(1);
        gc.strokeLine(w+i*s, h, w+i*s, h+9*s);
      }
    }
    gc.setFont(new Font("courier", 50));
    gc.setStroke(Color.BLACK);
    gc.setTextAlign(TextAlignment.CENTER);
    for (int j=0; j<9; j++) {
      for (int i=0; i<9; i++) {
        if (ban[j][i] >0) {
          gc.fillText(String.valueOf(ban[j][i]), w+(i+0.5)*s, h+(j+0.9)*s);
        }
      }
    }
  }

  public static void main(String... args) {
    launch(args);
  }
}
```

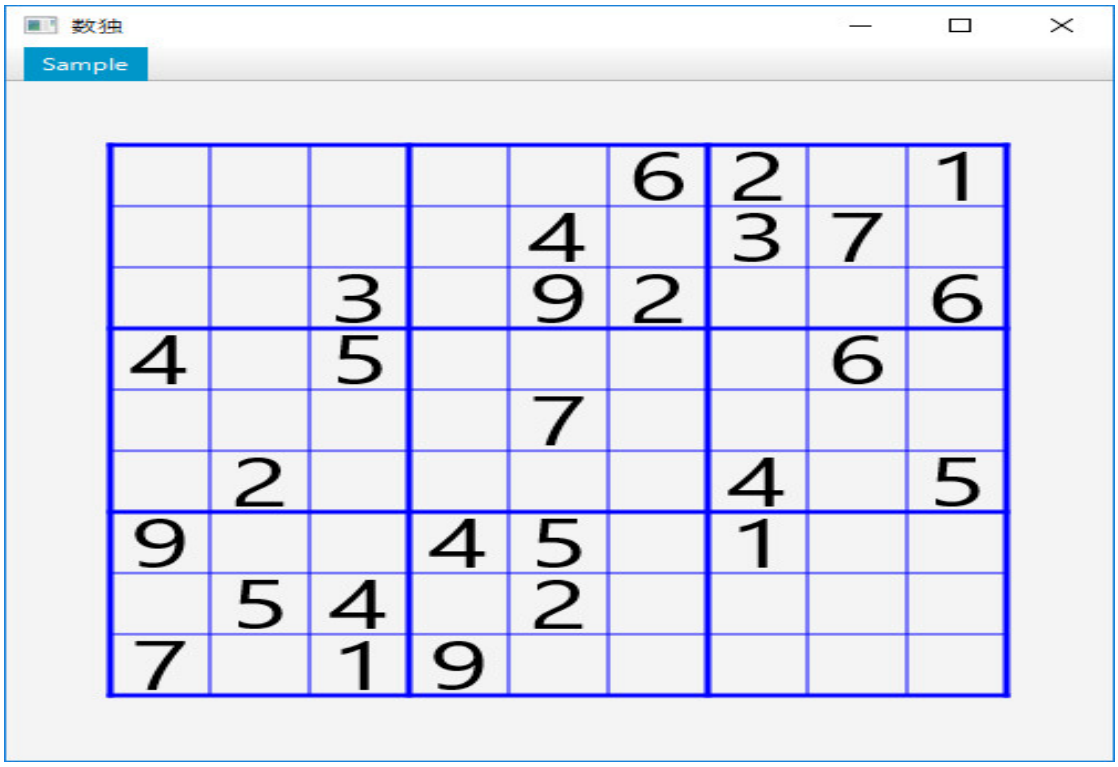
です。実行すると



です。メニューの Sample1 をクリックすると



で、メニューの Sample2 をクリックすると



となります。これも私のプログラムで作った問題で、かなりの難問です。このようにして、問題の数を増やしていけばいいです。

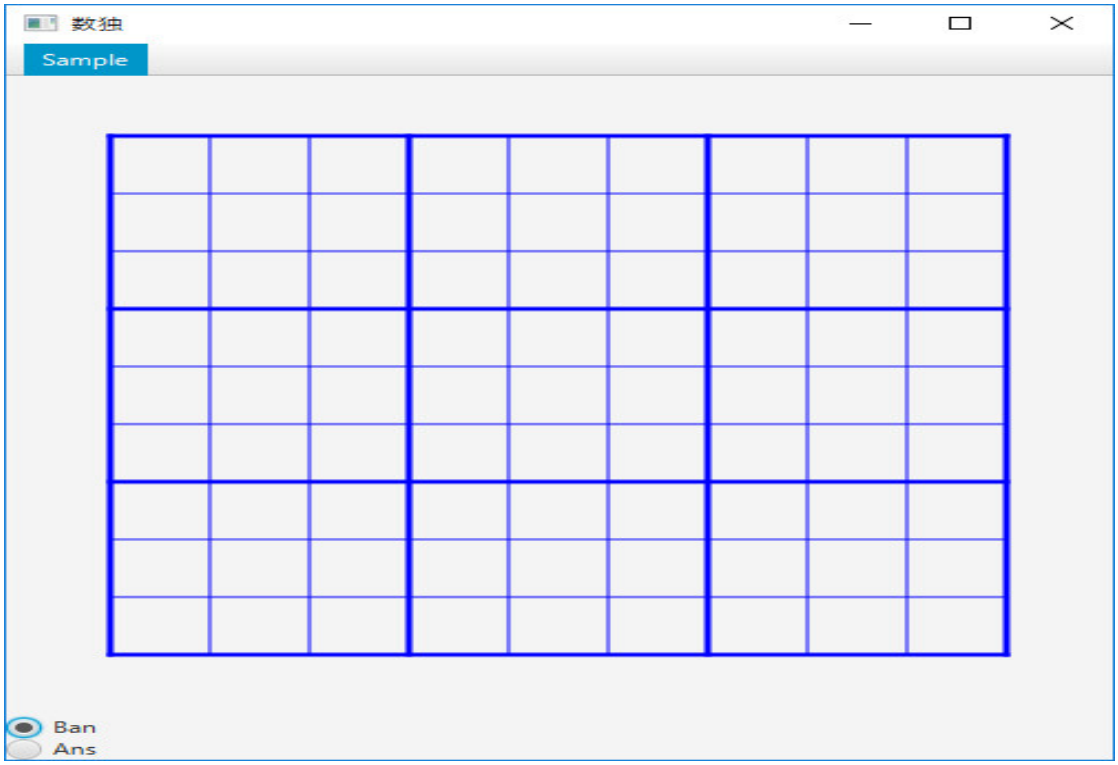
次に、マウスで解や問題を入力できるようにしましょう。まず、解の数字を入力するか問題の数字を入力するか指示できるように、ラジオボタンを配置します。関数 start(Stage pstage) に

```
RadioButton rb1 = new RadioButton("Ban");
RadioButton rb2 = new RadioButton("Ans");
ToggleGroup group = new ToggleGroup();
rb1.setToggleGroup(group);;
rb2.setToggleGroup(group);;
rb1.setSelected(true);;
```

を追加し、

```
root.getChildren().addAll(bar, canvas, rb1, rb2);
```

と修正します。実行すると



となります。左下にラジオボタンが表示されています。次に、マウスで盤をクリックすると数字を表示するようにします。関数 start(Stage pstage) に

```
    canvas.setOnMouseClicked((event)->{
      buttonPressed(event.getX(), event.getY());
    });
```

を追加し、クラス Sudoku にメソッド buttonPressed(double x, double y) を

```
  void buttonPressed(double x, double y) {
    int k = 9;
    int s = board_size / (k+2);
    int w = (board_size-s*9)/2;
    int h = (board_size-s*9)/2;

    int i = (int) Math.floor((x - w) / s);
    int j = (int) Math.floor((y - h) / s);
    int n = 7;
    if (i >= 0 && i < 9 && j >= 0 && j < 9) {
      gc.setFont(new Font("courier", 50));
      gc.setStroke(Color.BLACK);
      gc.setTextAlign(TextAlignment.CENTER);
      gc.fillText(String.valueOf(n), w+(i+0.5)*s, h+(j+0.9)*s);
    }
  }
```

のように定義します。さらに、gc が drawCanvas() 以外からアクセスできるように、クラス Sudoku の最初に

```
  GraphicsContext gc;
```

の宣言を置き、 drawCanvas() では

```
    gc = canvas.getGraphicsContext2D();
```

と修正します。プログラムの全体は

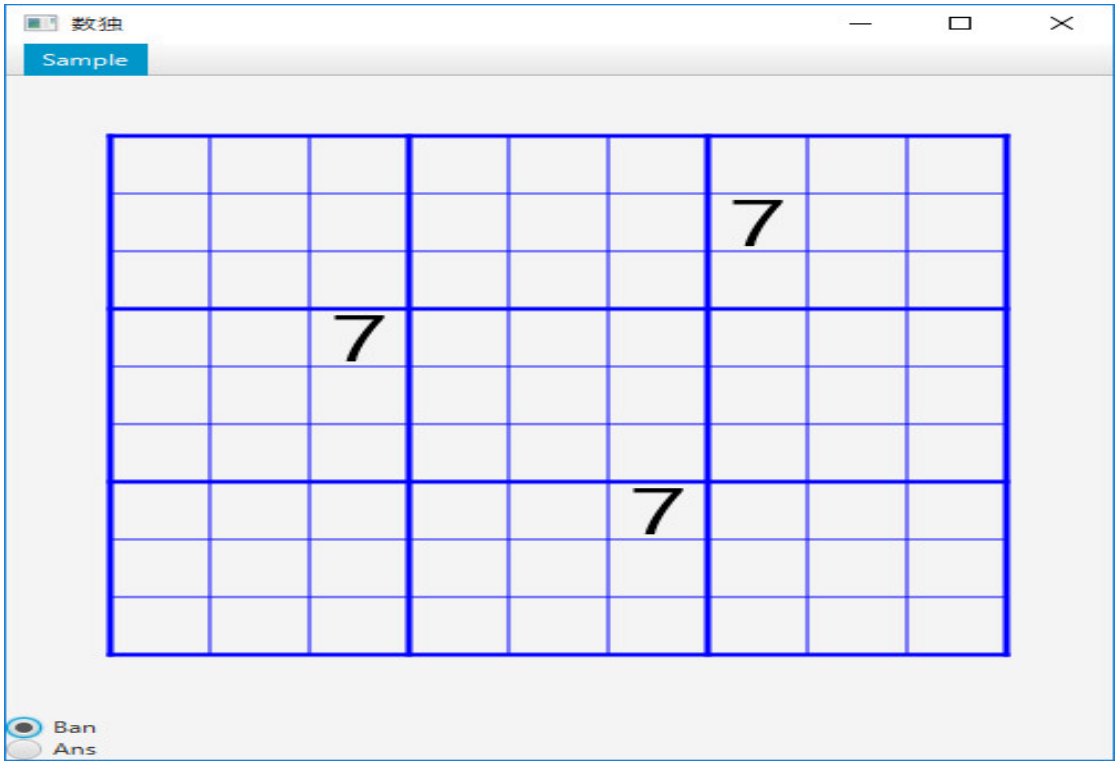
```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.control.Menu;
import javafx.scene.control.MenuBar;
import javafx.scene.control.MenuItem;
import javafx.scene.control.RadioButton;
import javafx.scene.control.ToggleGroup;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.text.TextAlignment;
import javafx.stage.Stage;

public class Sudoku extends Application {
  Canvas canvas;
  GraphicsContext gc;
  int board_size = 500;
  int[][] ban = {
    {0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0}
  };
  int[][] ban1 = {
    {0,1,0,0,0,0,9,0,3},
    {0,9,0,0,0,7,0,0,0},
    {0,0,0,0,0,0,0,7,1},
    {6,0,9,0,0,0,0,0,0},
```

```
  {0,0,7,6,0,0,0,0,0},
  {2,0,0,8,0,5,0,0,0},
  {0,0,4,0,0,8,0,0,0},
  {0,3,0,0,2,0,0,0,0},
  {0,0,8,5,3,0,0,9,4}
};
 int[][] ban2 = {
  {0,0,0,0,0,6,2,0,1},
  {0,0,0,0,4,0,3,7,0},
  {0,0,3,0,9,2,0,0,6},
  {4,0,5,0,0,0,0,6,0},
  {0,0,0,0,7,0,0,0,0},
  {0,2,0,0,0,0,4,0,5},
  {9,0,0,4,5,0,1,0,0},
  {0,5,4,0,2,0,0,0,0},
  {7,0,1,9,0,0,0,0,0}
};

@Override
 public void start(Stage pstage) {
   MenuBar bar = new MenuBar();
   Menu m1 = new Menu("Sample");
   MenuItem sample1 = new MenuItem("Sample1");
   MenuItem sample2 = new MenuItem("Sample2");
   m1.getItems().addAll(sample1, sample2);
   bar.getMenus().add(m1);
   RadioButton rb1 = new RadioButton("Ban");
   RadioButton rb2 = new RadioButton("Ans");
   ToggleGroup group = new ToggleGroup();
   rb1.setToggleGroup(group);;
   rb2.setToggleGroup(group);;
   rb1.setSelected(true);;
   VBox root = new VBox();
   canvas = new Canvas(board_size, board_size);
   root.getChildren().addAll(bar, canvas, rb1, rb2);
   drawCanvas();
   Scene scene = new Scene(root);
   pstage.setTitle("数独");
   pstage.setScene(scene);
   pstage.show();

   sample1.setOnAction((event)->{
     ban = ban1;
```

```
    drawCanvas();
  });
  sample2.setOnAction((event)->{
    ban = ban2;
    drawCanvas();     }
  });
  canvas.setOnMouseClicked((event)->{
    buttonPressed(event.getX(), event.getY());
  });
}
void buttonPressed(double x, double y) {
  int k = 9;
  int s = board_size / (k+2);
  int w = (board_size-s*9)/2;
  int h = (board_size-s*9)/2;

  int i = (int) Math.floor((x - w) / s);
  int j = (int) Math.floor((y - h) / s);
  int n = 7;
  if (i >= 0 && i < 9 && j >= 0 && j < 9) {
    gc.setFont(new Font("courier", 50));
    gc.setStroke(Color.BLACK);
    gc.setTextAlign(TextAlignment.CENTER);
    gc.fillText(String.valueOf(n), w+(i+0.5)*s, h+(j+0.9)*s);
  }
}
void drawCanvas() {
  int k = 9;
  int s = board_size / (k+2);
  int w = (board_size-s*9)/2;
  int h = (board_size-s*9)/2;

  gc = canvas.getGraphicsContext2D();
  gc.clearRect(0,  0,  board_size,  board_size);
  gc.setStroke(Color.BLUE);
  for (int i=0; i<=9; i++) {
    if (i % 3 == 0) {
      gc.setLineWidth(3);
      gc.strokeLine(w, h+i*s, w+9*s, h+i*s);
    } else {
      gc.setLineWidth(1);
      gc.strokeLine(w, h+i*s, w+9*s, h+i*s);
    }
```

```
    }
    for (int i=0; i<=9; i++) {
      if (i % 3 == 0) {
        gc.setLineWidth(3);
        gc.strokeLine(w+i*s, h, w+i*s, h+9*s);
      } else {
        gc.setLineWidth(1);
        gc.strokeLine(w+i*s, h, w+i*s, h+9*s);
      }
    }
    gc.setFont(new Font("courier", 50));
    gc.setStroke(Color.BLACK);
    gc.setTextAlign(TextAlignment.CENTER);
    for (int j=0; j<9; j++) {
      for (int i=0; i<9; i++) {
        if (ban[j][i] >0) {
          gc.fillText(String.valueOf(ban[j][i]), w+(i+0.5)*s, h+(j+0.9)*s);
        }
      }
    }
  }

  public static void main(String... args) {
    launch(args);
  }
}
```

となりました。
実行すると

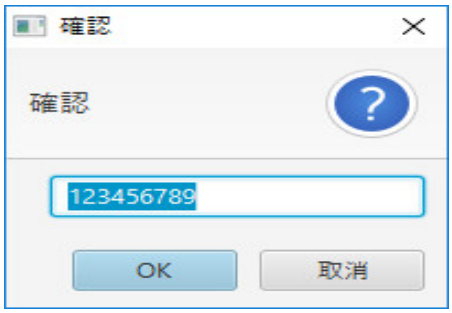


のように、クリックしたセルに数字７を表示してくれます。つぎに、表示すべき数字をダイアログボックスで指示できるようにします。

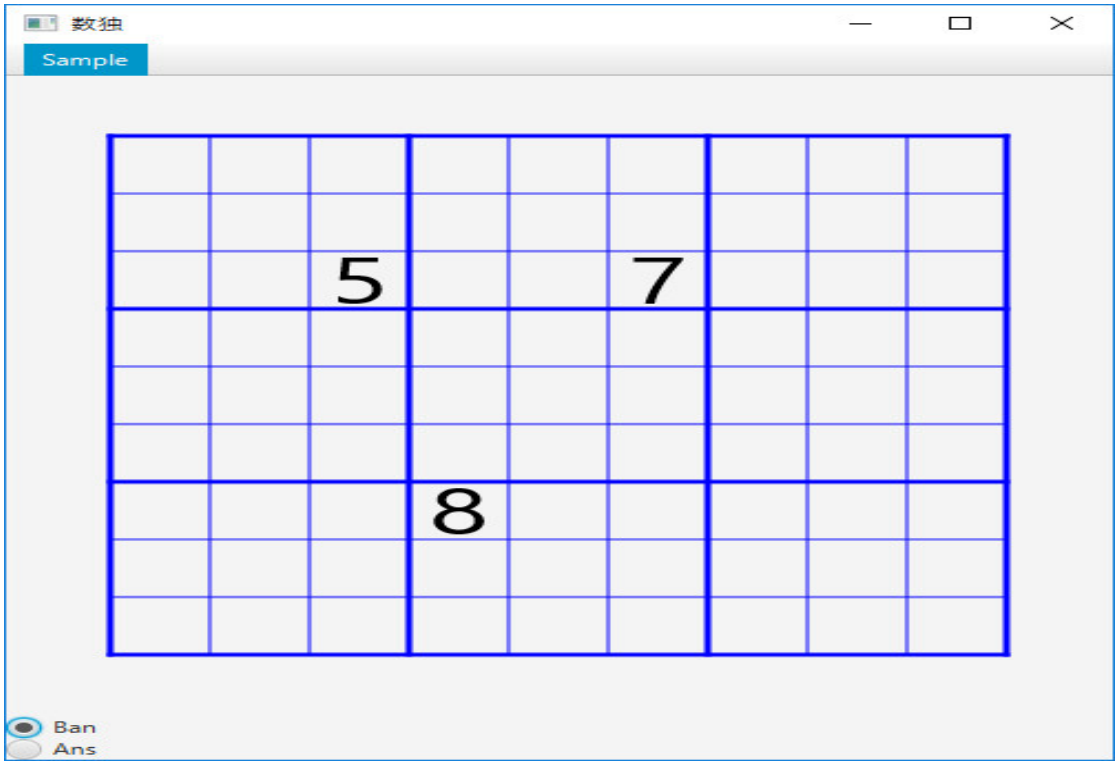
関数 buttonPressed(double x, double y) を

```
void buttonPressed(double x, double y) {
  int k = 9;
  int s = board_size / (k+2);
  int w = (board_size-s*9)/2;
  int h = (board_size-s*9)/2;

  int i = (int) Math.floor((x - w) / s);
  int j = (int) Math.floor((y - h) / s);
  if (i >= 0 && i < 9 && j >= 0 && j < 9) {
    String str = "123456789";
    TextInputDialog dlg = new TextInputDialog(str);
    str = dlg.showAndWait().orElse("");
    if (str != "") {
      int n = Integer.parseInt(str);
      ban[j][i] = n;
      drawCanvas();
    }
  }
}
```

と修正します。実行すると



のように、セルをクリックするとダイアログが表示されます。「確認」とか変な表示ですが気になる人は表示を変えるようにするにはどのようにすればいいか、インターネットで調べて修正してください。数字を入力し、「OK」のボタンをクリックすれば、



のように、数字を入力できます。ラジオボタンで、解と問題の入力を区別できるようにします。そのために解を保持する配列 ans[][] を作ります。更に、ラジオボタンをクラス Sudoku 全体でアクセスできるように修正するために、クラス Sudoku の先頭で

```
int[][] ans = new int[9][9];;
RadioButton rb1;
RadioButton rb2;
```

の宣言をします。そして、関数 start(Stage pstage) で

```
    rb1 = new RadioButton("Ban");
    rb2 = new RadioButton("Ans");
```

と変更し、sample1.setOnAction() と sample1.setOnAction() を

```
    sample1.setOnAction((event)->{
      ban = ban1;
      for (int j=0; j<9; j++) {
        for (int i=0; i<9; i++) {
          ans[j][i] = ban[j][i];
        }
      }
      drawCanvas();
    });
    sample2.setOnAction((event)->{
      ban = ban2;
      for (int j=0; j<9; j++) {
        for (int i=0; i<9; i++) {
          ans[j][i] = ban[j][i];
        }
      }
      drawCanvas();
    });
```

と修正します。関数 buttonPressed(double x, double y) を

```
  void buttonPressed(double x, double y) {
    int k = 9;
    int s = board_size / (k+2);
    int w = (board_size-s*9)/2;
    int h = (board_size-s*9)/2;

    int i = (int) Math.floor((x - w) / s);
    int j = (int) Math.floor((y - h) / s);
    if (i >= 0 && i < 9 && j >= 0 && j < 9) {
      String str = "123456789";
      TextInputDialog dlg = new TextInputDialog(str);
      str = dlg.showAndWait().orElse("");
      if (str != "") {
        int n = Integer.parseInt(str);
        if (rb1.isSelected()) {
          ban[j][i] = n;
          ans[j][i] = n;
        } else {
          ans[j][i] = n;
```

```
      }
      drawCanvas();
    }
  }
}
```

と修正します。最後に、関数 drawCanvas() を

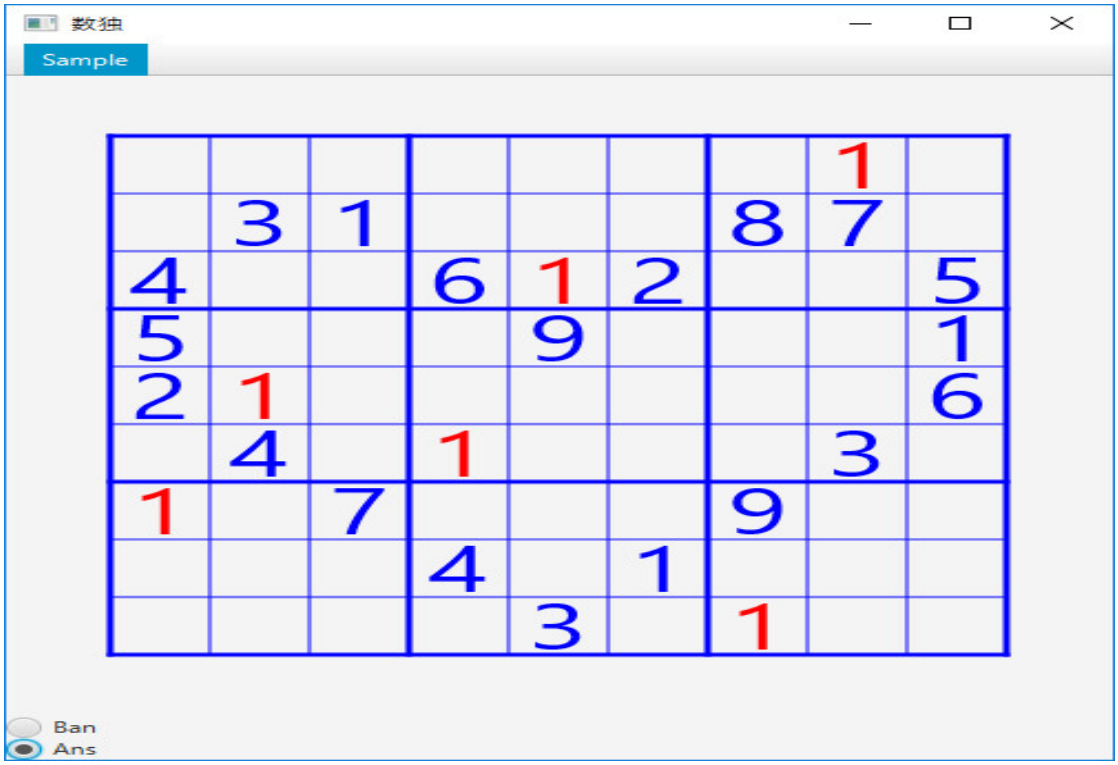
```
void drawCanvas() {
  int k = 9;
  int s = board_size / (k+2);
  int w = (board_size-s*9)/2;
  int h = (board_size-s*9)/2;

  gc = canvas.getGraphicsContext2D();
  gc.clearRect(0,  0,  board_size,  board_size);
  gc.setStroke(Color.BLUE);
  for (int i=0; i<=9; i++) {
    if (i % 3 == 0) {
      gc.setLineWidth(3);
      gc.strokeLine(w, h+i*s, w+9*s, h+i*s);
    } else {
      gc.setLineWidth(1);
      gc.strokeLine(w, h+i*s, w+9*s, h+i*s);
    }
  }
  for (int i=0; i<=9; i++) {
    if (i % 3 == 0) {
      gc.setLineWidth(3);
      gc.strokeLine(w+i*s, h, w+i*s, h+9*s);
    } else {
      gc.setLineWidth(1);
      gc.strokeLine(w+i*s, h, w+i*s, h+9*s);
    }
  }
  gc.setFont(new Font("courier", 50));
  gc.setTextAlign(TextAlignment.CENTER);
  for (int j=0; j<9; j++) {
    for (int i=0; i<9; i++) {
      if (ban[j][i] >0) {
        gc.setFill(Color.BLUE);
        gc.fillText(String.valueOf(ban[j][i]), w+(i+0.5)*s, h+(j+0.9)*s);
      } else if (ans[j][i] > 0) {
        gc.setFill(Color.RED);
```

```
            gc.fillText(String.valueOf(ans[j][i]), w+(i+0.5)*s, h+(j+0.9)*s);
        }
      }
    }
  }
```

と修正します。ここで fillText() で描く文字の色は

```
            gc.setFill(Color.BLUE);
```

で、指示する必要があることに気付く。実行すると



のように、ban のデータと ans のデータを入力できます。これは私の VC++ のプログラムが作った（見つけた）問題で、超難問です。このような局面をファイルに保存したり、復元したりできるようにしましょう。

まず、メニューを追加します。

```
MenuBar bar = new MenuBar();
Menu m2 = new Menu("File");
MenuItem savemenu = new MenuItem("Save");
MenuItem openmenu = new MenuItem("Open");
m2.getItems().addAll(savemenu, openmenu);
Menu m1 = new Menu("Sample");
MenuItem sample1 = new MenuItem("Sample1");
MenuItem sample2 = new MenuItem("Sample2");
```

```
    m1.getItems().addAll(sample1, sample2);
    bar.getMenus().addAll(m2, m1);
```

と修正します。メニューをクリックしたときの処理を定義します。

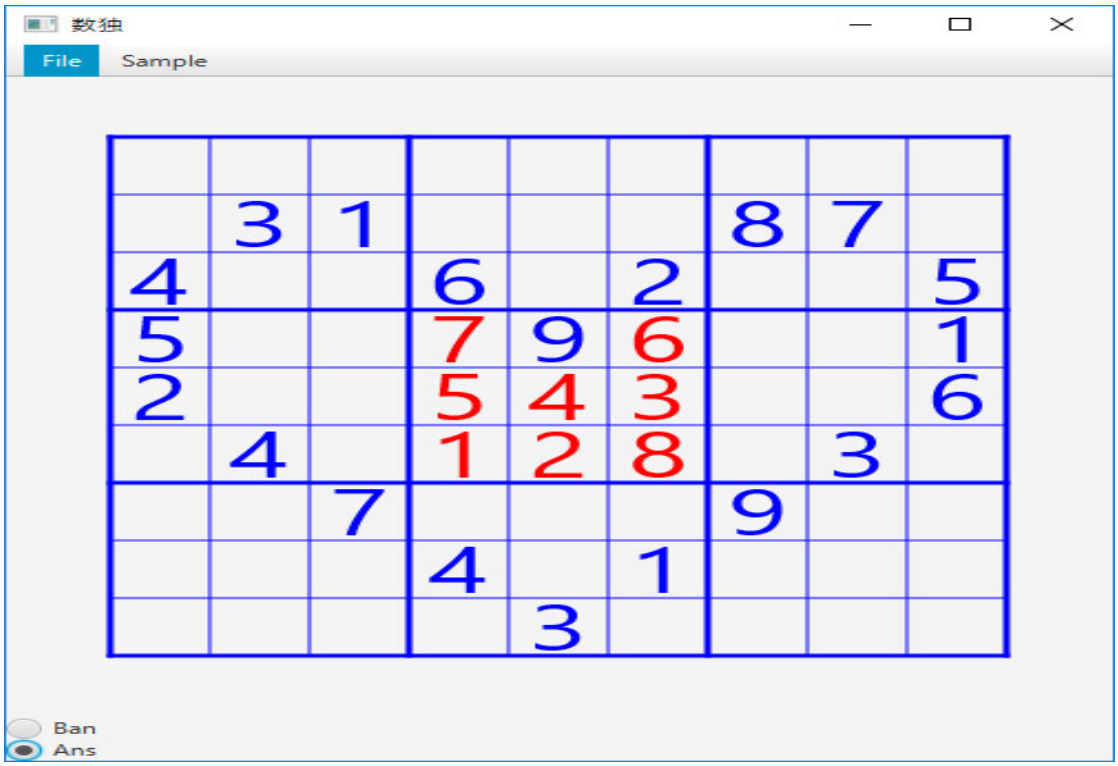
```
    savemenu.setOnAction((event)->{
    try {
      FileChooser fc = new FileChooser();
      File sf = fc.showSaveDialog(pstage);
      if (sf != null) {
        FileWriter fileout = new FileWriter(sf);
          for (int j=0; j<9; j++) {
            String str = "";
            for (int i=0; i<9; i++) {
              str += String.valueOf(ban[j][i])+ " ";
            }
            fileout.write(str + "\n");
          }
          for (int j=0; j<9; j++) {
            String str = "";
            for (int i=0; i<9; i++) {
              str += String.valueOf(ans[j][i])+ " ";
            }
            fileout.write(str + "\n");
          }
          fileout.close();
        }
      } catch(Exception e) {};
    });
```

と

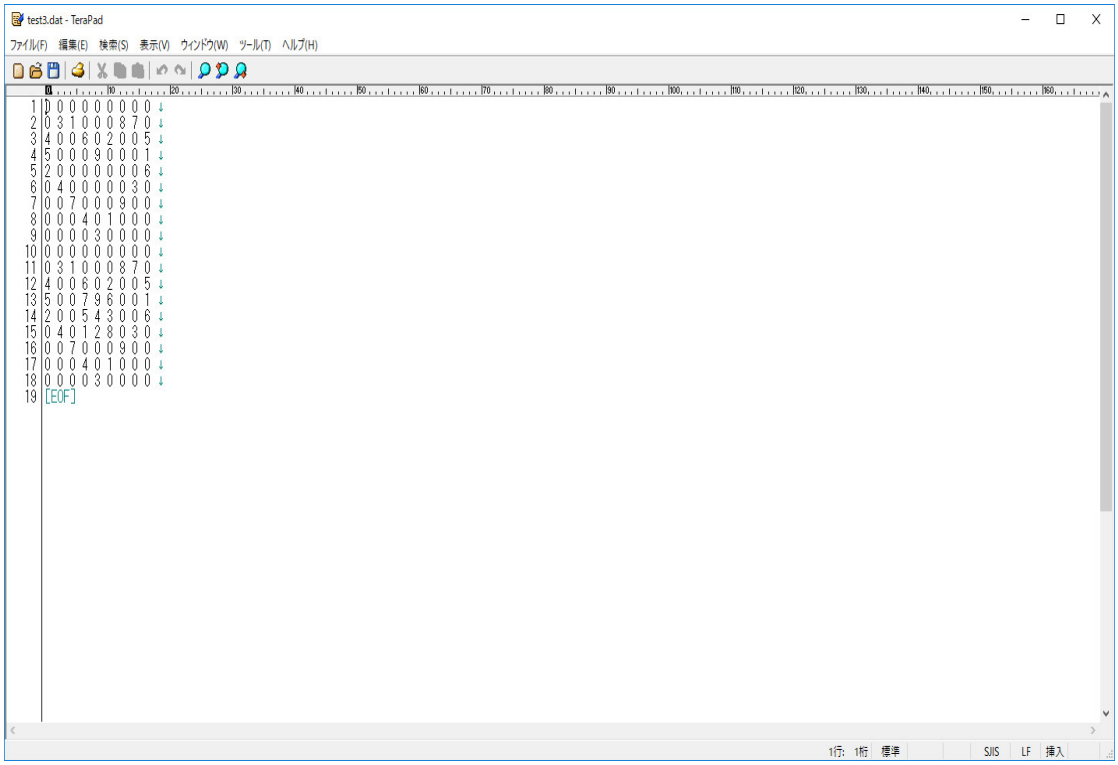
```
    openmenu.setOnAction((event)->{
      try {
        FileChooser fc = new FileChooser();
        File sf = fc.showOpenDialog(pstage);
        if (sf != null) {
          FileReader filein = new FileReader(sf);
          BufferedReader buf = new BufferedReader(filein);
          String data;
          int row = 0;
          while ((data = buf.readLine())!= null) {
            String[] sa = data.split(" ");
            if (row < 9) {
              for (int i=0; i<9; i++) {
```

```
            ban[row][i] = Integer.parseInt(sa[i]);
          }
        } else {
          for (int i=0; i<9; i++) {
            ans[row-9][i] = Integer.parseInt(sa[i]);
          }
        }
        row++;
      }
      filein.close();
    }
  } catch(Exception e) {};
  drawCanvas();
});
```

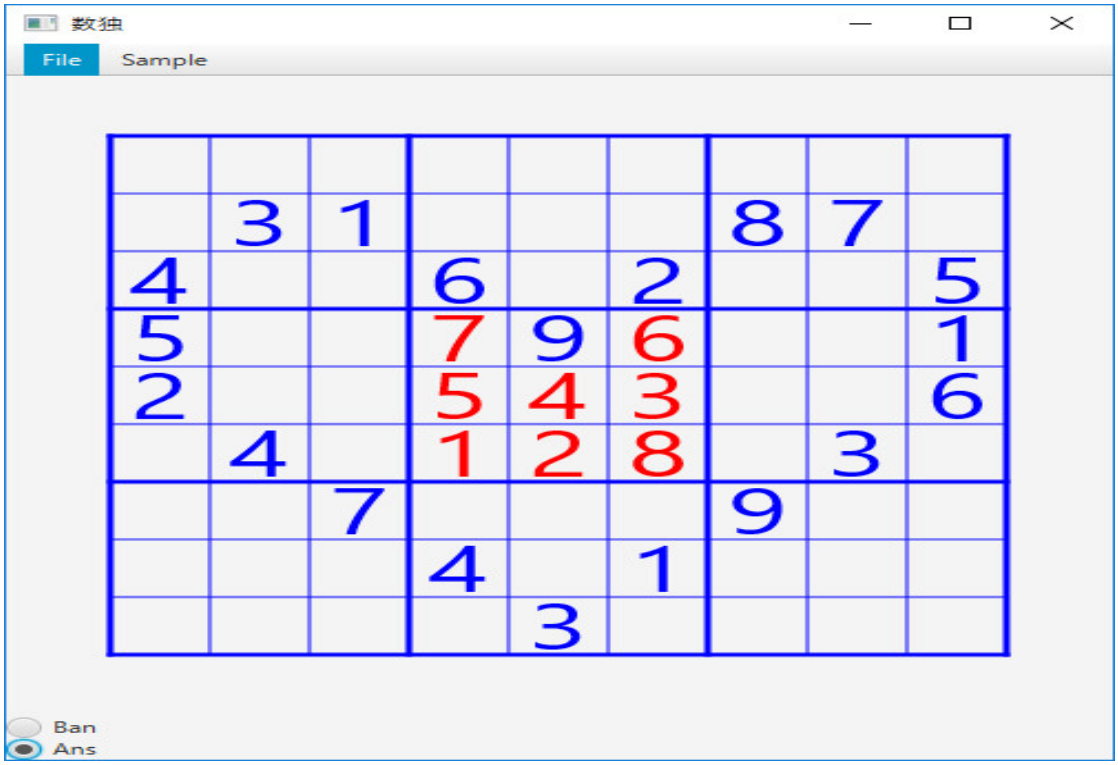
です。実行して、



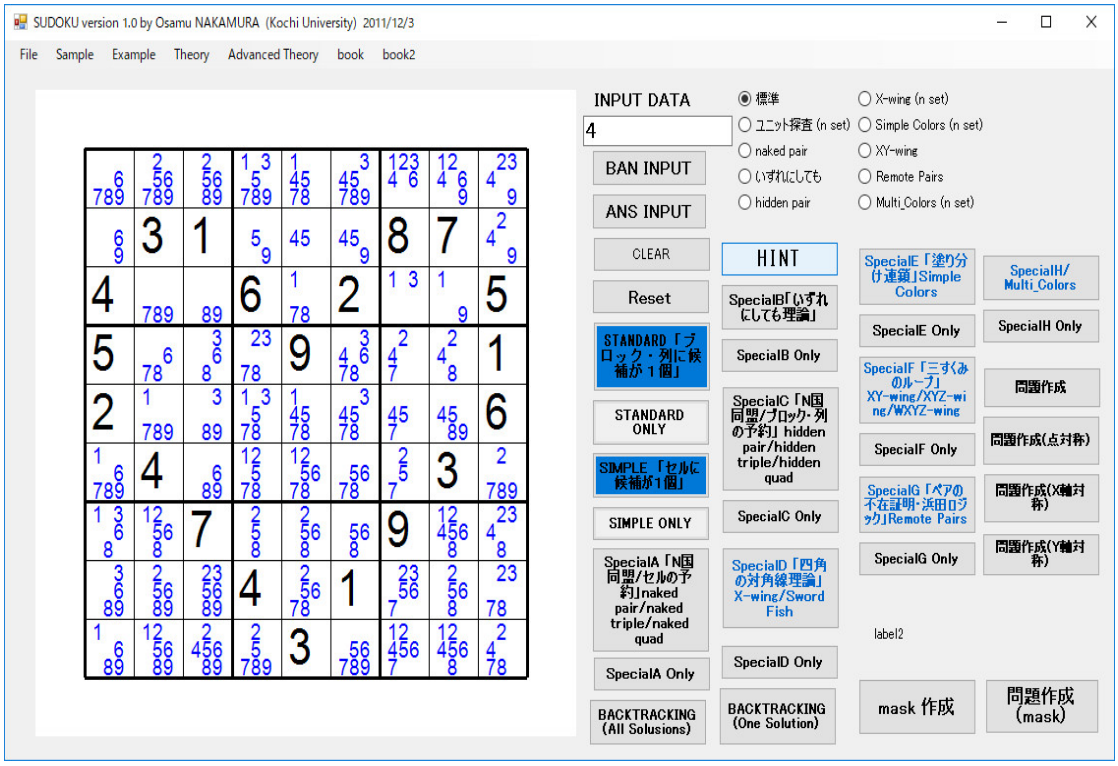
の局面を保存すると



のように ban[][] と ans[][] の数字を単に並べたファイルを作るだけです。このファイルを「Open」すると



ともとに戻ります。
　次に



のように、ヒントを表示するようにしましょう。ヒントの小さい数字は、1 から 9 までの数字のうち、縦の列、横の行、ブロックに現れる数字を除いた、そのセルに入りうる可能性のある数字を表しています。Java には、Python や Ruby のように二項演算を備えた set のデータ構造が備わっていないので、boolean[][][] hint[j][i][n] = true if n is candidate, = false if n is not candidate という配列を使うことにします。まず、クラス Sudoku の最初に

```
boolean[][][] hint = new boolean[9][9][10];
```

の宣言を置きます。最後が 101 であることに注意してください。n を 1 から 9 まで動かします。

```
    sample1.setOnAction((event)->{
      ban = ban1;
      for (int j=0; j<9; j++) {
        for (int i=0; i<9; i++) {
          ans[j][i] = ban[j][i];
        }
      }
      setHint();
      drawCanvas();
    });
    sample2.setOnAction((event)->{
```

```
      ban = ban2;
      for (int j=0; j<9; j++) {
        for (int i=0; i<9; i++) {
          ans[j][i] = ban[j][i];
        }
      }
      setHint();
      drawCanvas();
    });
```

と

```
      setHint();
```

を追加します。

```
    openmenu.setOnAction((event)->{
    try {
      FileChooser fc = new FileChooser();
      File sf = fc.showOpenDialog(pstage);
      if (sf != null) {
        FileReader filein = new FileReader(sf);
        BufferedReader buf = new BufferedReader(filein);
        String data;
        int row = 0;
        while ((data = buf.readLine())!= null) {
          String[] sa = data.split(" ");
          if (row < 9) {
            for (int i=0; i<9; i++) {
              ban[row][i] = Integer.parseInt(sa[i]);
            }
          } else {
            for (int i=0; i<9; i++) {
              ans[row-9][i] = Integer.parseInt(sa[i]);
            }
          }
          row++;
        }
        filein.close();
      }
    } catch(Exception e) {};
    setHint();
    drawCanvas();
  });
```

にも

```
    setHint();
```

を追加します。関数 setHint() は

```
  void setHint() {
    int[][][] box = new int[9][9][2];
    for (int j=0; j<9; j++) {
      for (int i=0; i<9; i++) {
        int m = j / 3 * 3 + i / 3;
        int n = j % 3 * 3 + i % 3;
        box[m][n][0] = j;
        box[m][n][1] = i;
      }
    }
    for (int j=0; j<9; j++) {
      for (int i=0; i<9; i++) {
        for (int n=1; n<=9; n++) {
          boolean flag = true;
          for (int k=0; k<9; k++) {
            if (ans[j][k]==n) {
              flag = false;
              break;
            }
          }
          for (int k=0; k<9; k++) {
            if (ans[k][i]==n) {
              flag = false;
              break;
            }
          }
          int m = j / 3 * 3 + i / 3;
          for (int k=0; k<9; k++) {
            if (ans[box[m][k][0]][box[m][k][1]]==n) {
              flag = false;
              break;
            }
          }
          hint[j][i][n] = flag;
        }
      }
    }
  }
```

と定義します。

```
  void buttonPressed(double x, double y) {
    int k = 9;
    int s = board_size / (k+2);
    int w = (board_size-s*9)/2;
    int h = (board_size-s*9)/2;

    int i = (int) Math.floor((x - w) / s);
    int j = (int) Math.floor((y - h) / s);
    if (i >= 0 && i < 9 && j >= 0 && j < 9) {
      String str = "123456789";
      TextInputDialog dlg = new TextInputDialog(str);
      str = dlg.showAndWait().orElse("");
      if (str != "") {
        int n = Integer.parseInt(str);
        if (rb1.isSelected()) {
          ban[j][i] = n;
          ans[j][i] = n;
        } else {
          ans[j][i] = n;
        }
        setHint();
        drawCanvas();
      }
    }
  }
```

と

```
      setHint();
```

を追加します。最後に drawCanvas() の最後に

```
    gc.setFont(new Font("courier", 12));
    gc.setTextAlign(TextAlignment.CENTER);
    gc.setFill(Color.BLACK);
    for (int j=0; j<9; j++) {
      for (int i=0; i<9; i++) {
        if (ans[j][i] == 0) {
          for (int n=1; n<=9; n++) {
            if (hint[j][i][n]) {
              int r = (int) ((j+((n-1)/3*2+1.5)/6.0)*s);
              int l = (int) ((i+((n-1)%3*2+1)/6.0)*s);
              gc.fillText(String.valueOf(n), w+l, h+r);
            }
          }
```

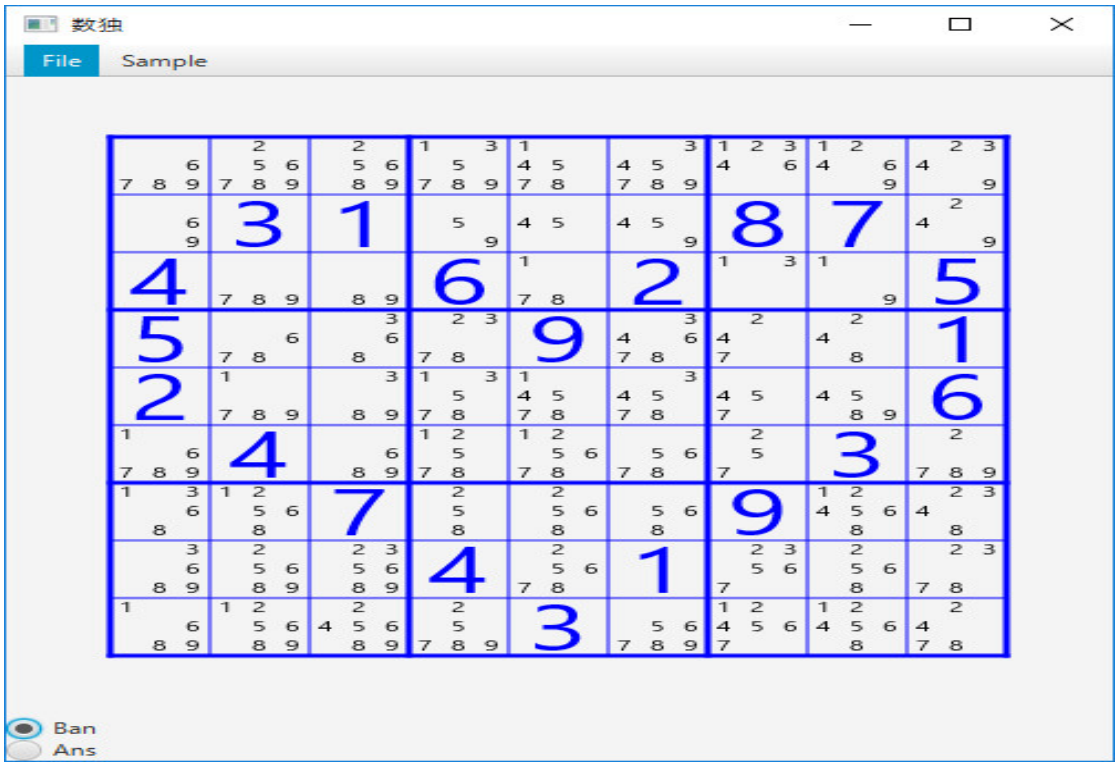
```
      }
    }
  }
```

を追加して、

```
  void drawCanvas() {
    int k = 9;
    int s = board_size / (k+2);
    int w = (board_size-s*9)/2;
    int h = (board_size-s*9)/2;

    gc = canvas.getGraphicsContext2D();
    gc.clearRect(0,  0,  board_size,  board_size);
    gc.setStroke(Color.BLUE);
    for (int i=0; i<=9; i++) {
      if (i % 3 == 0) {
        gc.setLineWidth(3);
        gc.strokeLine(w, h+i*s, w+9*s, h+i*s);
      } else {
        gc.setLineWidth(1);
        gc.strokeLine(w, h+i*s, w+9*s, h+i*s);
      }
    }
    for (int i=0; i<=9; i++) {
      if (i % 3 == 0) {
        gc.setLineWidth(3);
        gc.strokeLine(w+i*s, h, w+i*s, h+9*s);
      } else {
        gc.setLineWidth(1);
        gc.strokeLine(w+i*s, h, w+i*s, h+9*s);
      }
    }
    gc.setFont(new Font("courier", 50));
    gc.setTextAlign(TextAlignment.CENTER);
    for (int j=0; j<9; j++) {
      for (int i=0; i<9; i++) {
        if (ban[j][i] >0) {
          gc.setFill(Color.BLUE);
          gc.fillText(String.valueOf(ban[j][i]), w+(i+0.5)*s, h+(j+0.9)*s);
        } else if (ans[j][i] > 0) {
          gc.setFill(Color.RED);
          gc.fillText(String.valueOf(ans[j][i]), w+(i+0.5)*s, h+(j+0.9)*s);
        }
```

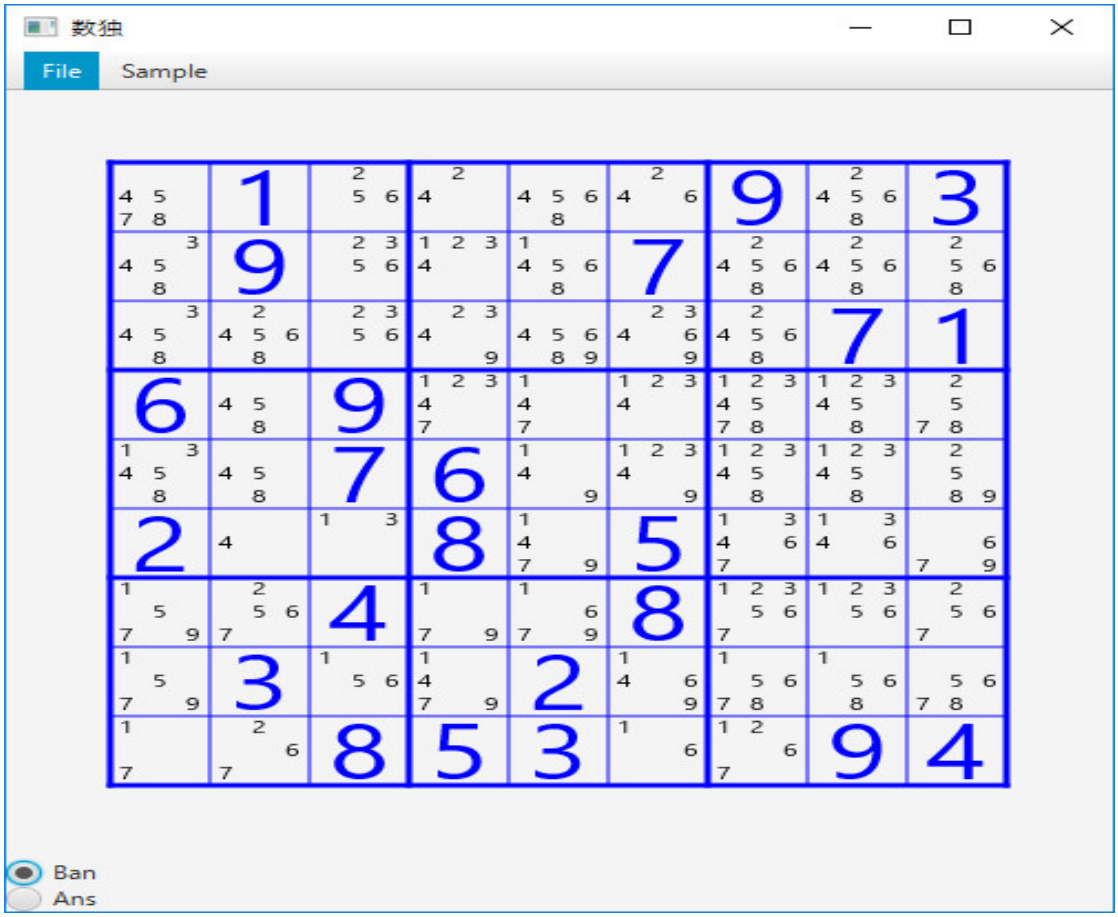
```
      }
    }
    gc.setFont(new Font("courier", 12));
    gc.setTextAlign(TextAlignment.CENTER);
    gc.setFill(Color.BLACK);
    for (int j=0; j<9; j++) {
      for (int i=0; i<9; i++) {
        if (ans[j][i] == 0) {
          for (int n=1; n<=9; n++) {
            if (hint[j][i][n]) {
              int r = (int) ((j+((n-1)/3*2+1.5)/6.0)*s);
              int l = (int) ((i+((n-1)%3*2+1)/6.0)*s);
              gc.fillText(String.valueOf(n), w+l, h+r);
            }
          }
        }
      }
    }
  }
}
```

と修正します。実行し、保存していた問題を読み込むと

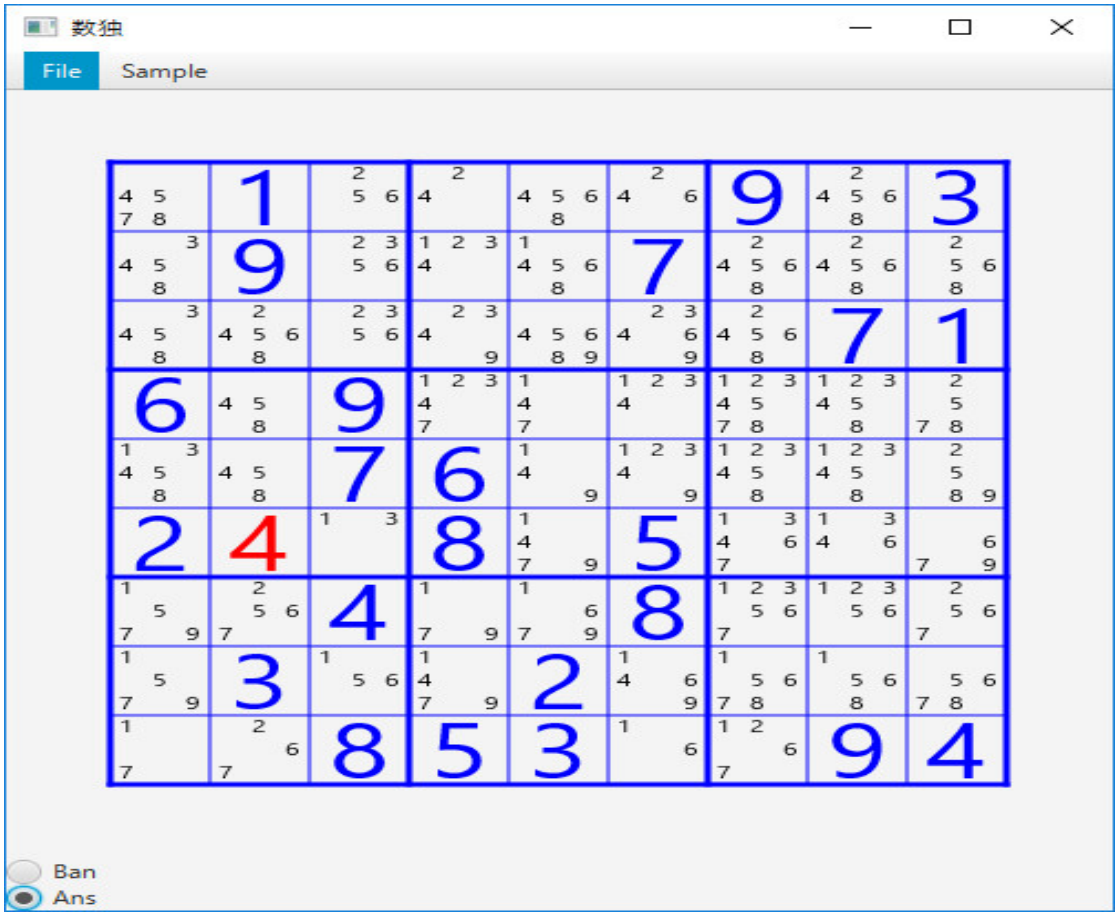


のように、ヒントを表示します。

次に、コンピュータに問題を解かせてみましょう。ヒントが手掛かりになります。Sample1 のヒントは

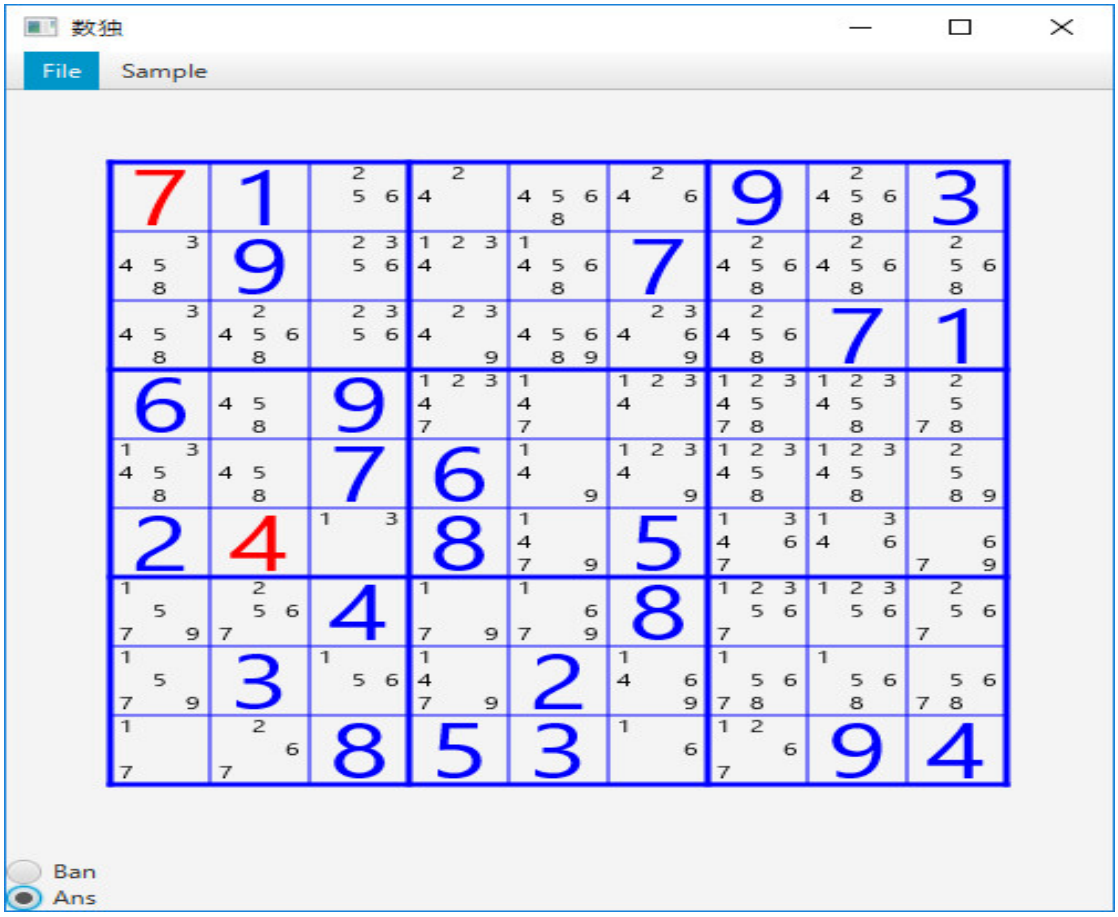


です。ヒントの小さい数字は、1 から 9 までの数字のうち、縦の列、横の行、ブロックに現れる数字を除いた、そのセルに入りうる可能性のある数字を表しています。「セルに候補が 1 個」の法則 (One-choice: A cell that contains only one candidate value) で、この数字が 1 個ならその数字に確定します。(5,1) のセルには 4 しか候補者がないです。従って、このセルは 4 に確定します。



となります。もう「セルに候補が 1 個」の法則 (One-choice: A cell that contains only one candidate value) を適応できるセルはないです。

しかし、左上のブロックの左上隅 のセルに注目すると、このブロックで、7 が候補として挙げられているセルは左上隅 のセルだけです。このことは、数独の解説書では、普通は 7 が置けないところに線を引いてみることをします。「ブロック・列・行に候補が 1 個」の法則 (One-place: A region(row, column, or block) that has only one cell available for a given number) で、このセルは 7 です。



となります。入門レヴェルの「数独」の問題はこの二つの法則を繰り返し適応することにより解けます。この二つの法則を使ってわかる解をコンピュータに見つけさせるようにしてみましょう。

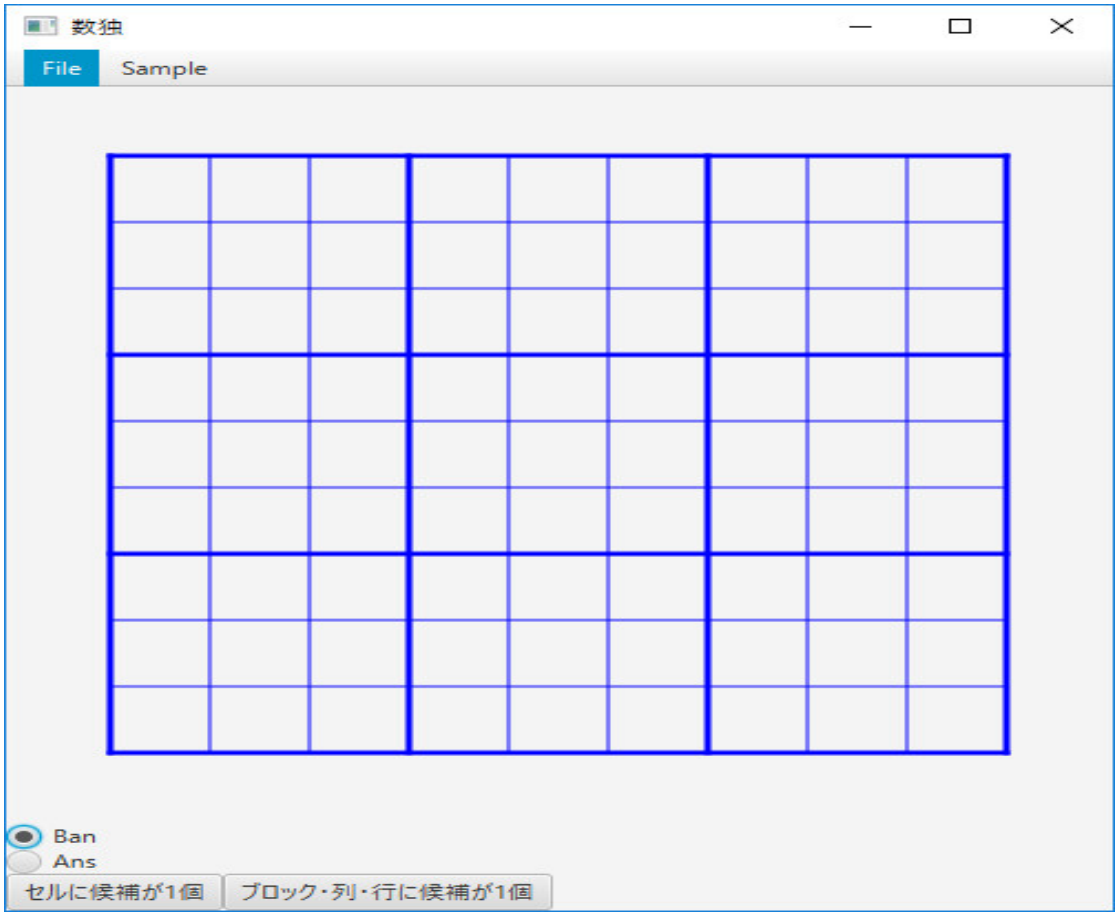
まず、ボタンを配置します。start(Stage pstage) に

```
Button b1 = new Button("セルに候補が 1 個");
Button b2 = new Button("ブロック・列・行に候補が 1 個");
HBox hb = new HBox();
hb.getChildren().addAll(b1, b2);
```

を追加し、

```
root.getChildren().addAll(bar, canvas, rb1, rb2, hb);
```

と修正します。実行すると



と一番下にボタンが表示されています。これらのボタンをクリックすると法則を実行するようにプログラミングします。

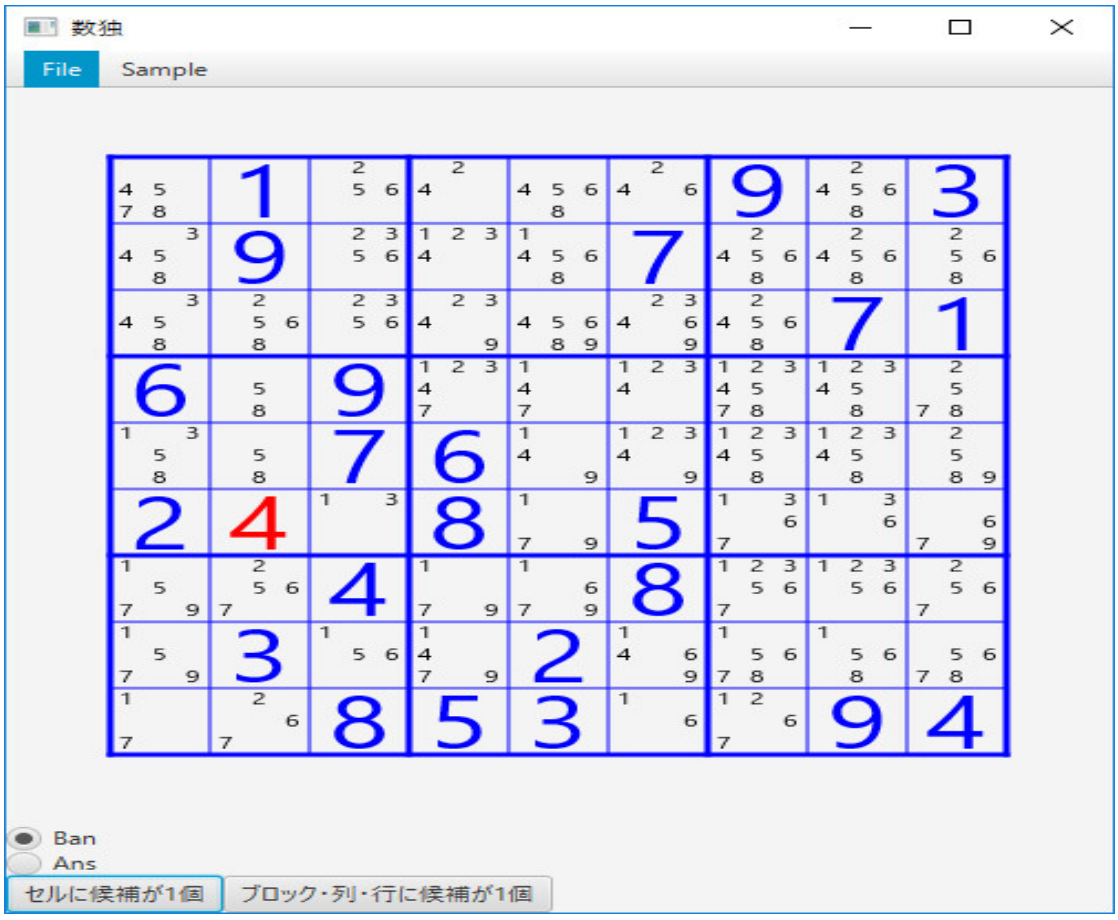
```
b1.setOnAction((event) -> {
  setUniqueSell();
});
```

を start(Stage pstage) に追加し、関数 setUniqueSell() を

```
boolean setUniqueSell() {
  boolean FLAG = false;
  for (int j=0; j<9; j++) {
    for (int i=0; i<9; i++) {
      if (ans[j][i] > 0) continue;
      int count = 0;
      int index = -1;
      for (int n=1; n<=9; n++) {
        if (hint[j][i][n]) {
          count++;
          index = n;
        }
```

```
        }
        if (count == 1) {
          ans[j][i] = index;
          FLAG = true;
        }
      }
    }
    setHint();
    drawCanvas();
    return FLAG;
  }
```

と定義します。この段階では、boolean setUniqueSell() でなく、void setUniqueSell() で良いですが、後のためにこのようにしています。実行し、Sample1 でボタン「セルに候補が 1 個」をクリックすると



となります。次に、ボタン「ブロック・列・行に候補が 1 個」の処理を定義します。

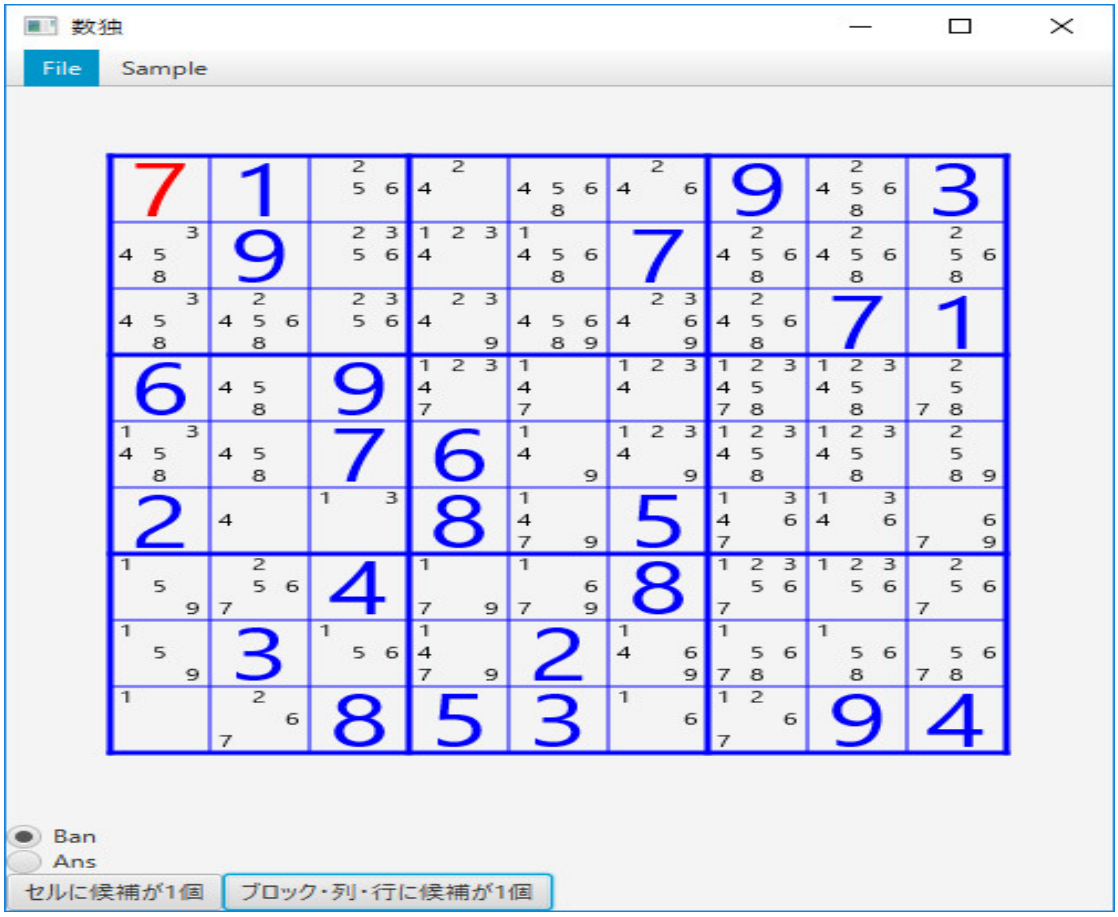
```
    b2.setOnAction((event) -> {
      setSingleCand();
    });
```

を start(Stage pstage) に追加し、関数 setSingleCand() を

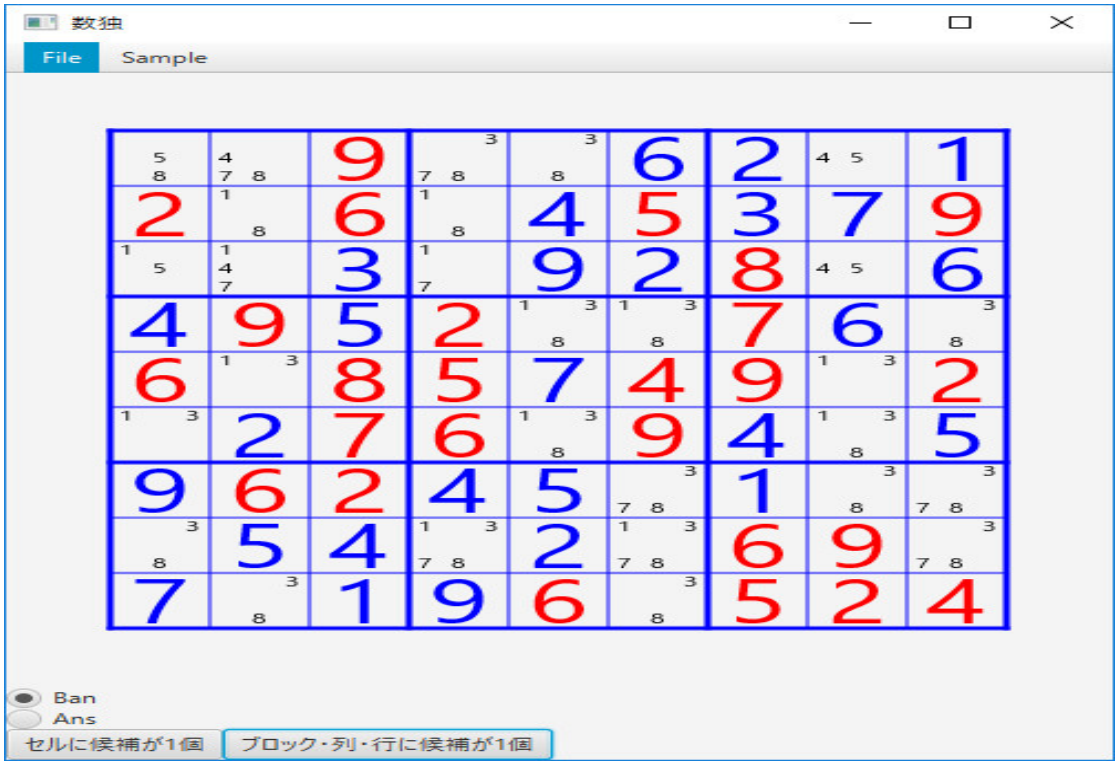
```
boolean setSingleCand() {
  boolean F = false;
  int[][][] box = new int[9][9][2];
  for (int j=0; j<9; j++) {
    for (int i=0; i<9; i++) {
      int m = j / 3 * 3 + i / 3;
      int n = j % 3 * 3 + i % 3;
      box[m][n][0] = j;
      box[m][n][1] = i;
    }
  }
  for (int j=0; j<9; j++) {
    for (int i=0; i<9; i++) {
      if (ans[j][i] > 0) continue;
      for (int n=1; n<=9; n++) {
        if (!hint[j][i][n]) continue;
        boolean flag = true;
        for (int k=0; k<9; k++) {
          if (k == i || ans[j][k]>0) continue;
          if (hint[j][k][n]) {
            flag = false;
            break;
          }
        }
        if (flag) {
          ans[j][i] = n;
          F = true;
          setHint();
          drawCanvas();
          return F;
        }
        flag = true;
        for (int k=0; k<9; k++) {
          if (k == j || ans[k][i]>0) continue;
          if (hint[k][i][n]) {
            flag = false;
            break;
          }
        }
        if (flag) {
          ans[j][i] = n;
```

```
          F = true;
          setHint();
          drawCanvas();
          return F;
        }
        int m = j / 3 * 3 + i / 3;
        int p = j % 3 * 3 + i % 3;
        flag = true;
        for (int k=0; k<9; k++) {
          if (k == p | ans[box[m][k][0]][box[m][k][1]]>0)
            continue;
          if (hint[box[m][k][0]][box[m][k][1]][n]) {
            flag = false;
            break;
          }
        }
        if (flag) {
          ans[j][i] = n;
          F = true;
          setHint();
          drawCanvas();
          return F;
        }
      }
    }
  }
  setHint();
  drawCanvas();
  return F;
}
```

と定義します。実行し、Sample1 でボタン「ブロック・列・行に候補が 1 個」をクリックすると



となります。Sample2 で何回かボタン「ブロック・列・行に候補が 1 個」をクリックすると



となります。

この二つの法則を繰り返して解を求めるボタンを作ってみましょう。

```
Button b3 = new Button("両方");
```

を start(Stage pstage) に追加し、

```
hb.getChildren().addAll(b1, b2, b3);
```

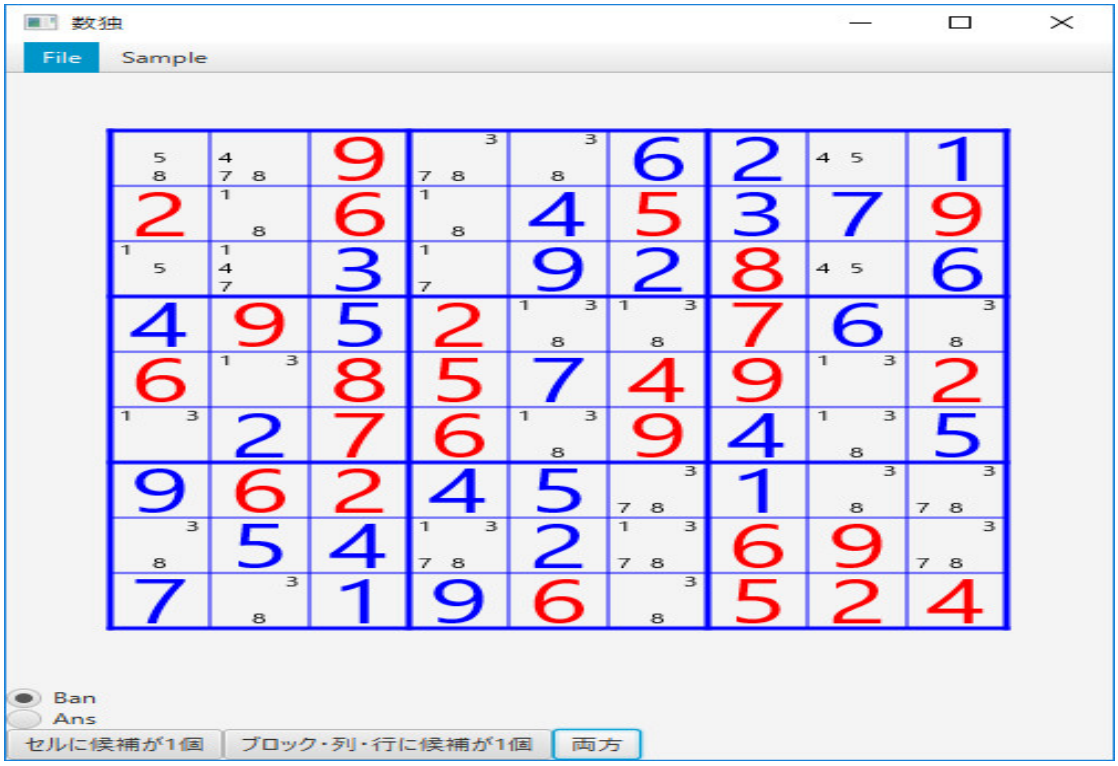
と修正します。関数 regularPlay() を

```
void regularPlay() {
  boolean F= false;
  do {
    do {
      F = setUniqueSell();
    } while (F);
    F = setSingleCand();
  } while (F);
}
```

と定義します。さらに、ボタン「両方」をクリックしたときの処理を

```
b3.setOnAction((event) -> {
  regularPlay();
  drawCanvas();
});
```

と定義します。
　実行し、Sample2 でボタン「両方」をクリックすると



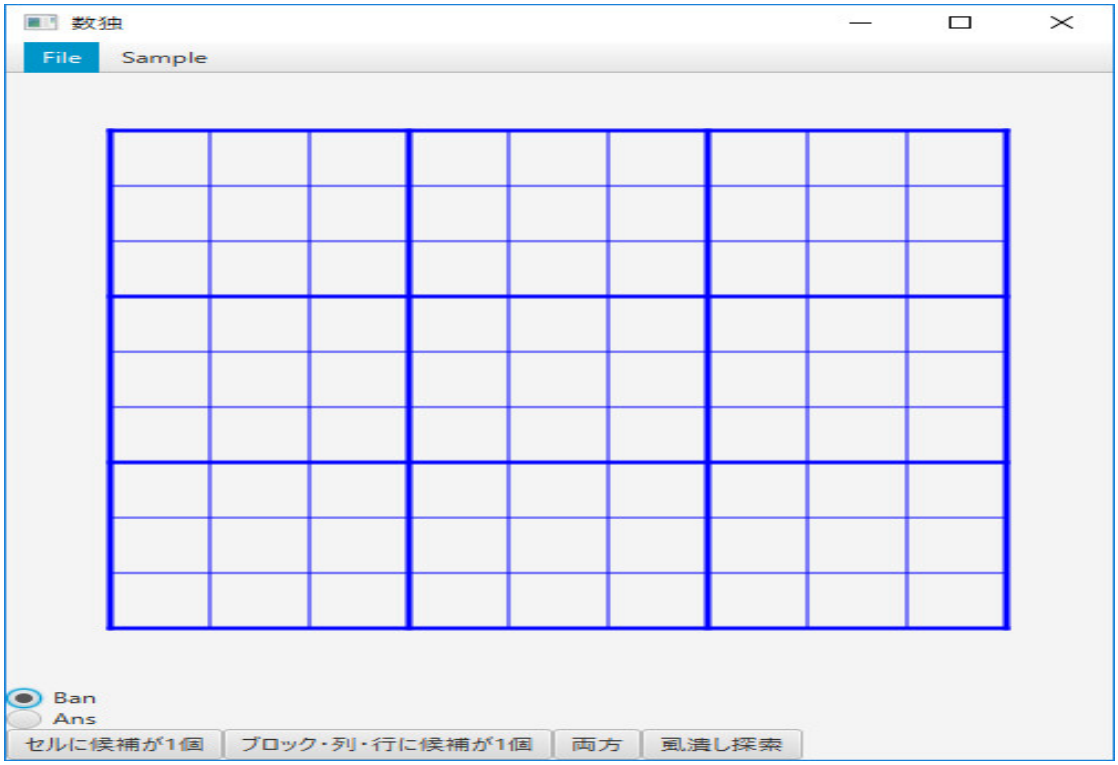
となります。
　最後に、「虱潰し探索」（バックトラッキング）で、解をコンピュータに探索させましょう。まず、ボタンを追加します。

```
Button b4 = new Button("虱潰し探索");
```

を start(Stage pstage) に追加し、

```
hb.getChildren().addAll(b1, b2, b3, b4);
```

と修正します。実行すると



となります。

関数 completeP() を

```
boolean completeP() {
  for (int j=0; j<9; j++) {
    for (int i=0; i<9; i++) {
      if (ans[j][i] == 0) {
        return false;
      }
    }
  }
  for (int j=0; j<9; j++) {
    for (int n=1; n<=9; n++) {
      boolean flag = false;
      for (int i=0; i<9; i++) {
        if (ans[j][i] == n) {
          flag = true;
          break;
        }
      }
      if (!flag)
        return false;
    }
  }
```

```
    for (int i=0; i<9; i++) {
      for (int n=1; n<=9; n++) {
        boolean flag = false;
        for (int j=0; j<9; j++) {
          if (ans[j][i] == n) {
            flag = true;
            break;
          }
        }
        if (!flag)
          return false;
      }
    }
    int[][][] box = new int[9][9][2];
    for (int j=0; j<9; j++) {
      for (int i=0; i<9; i++) {
        int m = j / 3 * 3 + i / 3;
        int n = j % 3 * 3 + i % 3;
        box[m][n][0] = j;
        box[m][n][1] = i;
      }
    }
    for (int k=0; k<9; k++) {
      for (int n=1; n<=9; n++) {
        boolean flag = false;
        for (int i=0; i<9; i++) {
          if (ans[box[k][i][0]][box[k][i][1]] == n) {
            flag = true;
            break;
          }
        }
        if (!flag)
          return false;
      }
    }
    return true;
  }
```

と定義します。関数 losingP() を

```
  boolean losingP() {
    setHint();
    for (int j=0; j<9; j++) {
      for (int i=0; i<9; i++) {
```

```
    boolean flag = false;
    for (int n=1; n<=9; n++) {
      if (hint[j][i][n]) {
        flag = true;
        break;
      }
    }
    if (!flag) return true;
  }
}
for (int j=0; j<9; j++) {
  int[] P = {0,0,0,0,0,0,0,0,0,0};
  for (int i=0; i<9; i++) {
    P[ans[j][i]] = P[ans[j][i]]+1;
  }
  for (int n=1; n<=9; n++) {
    if (P[n] > 1)
      return true;
  }
}
for (int i=0; i<9; i++) {
  int[] P = {0,0,0,0,0,0,0,0,0,0};
  for (int j=0; j<9; j++) {
    P[ans[j][i]] = P[ans[j][i]]+1;
  }
  for (int n=1; n<=9; n++) {
    if (P[n] > 1)
      return true;
  }
}
int[][][] box = new int[9][9][2];
for (int j=0; j<9; j++) {
  for (int i=0; i<9; i++) {
    int m = j / 3 * 3 + i / 3;
    int n = j % 3 * 3 + i % 3;
    box[m][n][0] = j;
    box[m][n][1] = i;
  }
}
for (int k=0; k<9; k++) {
  int[] P = {0,0,0,0,0,0,0,0,0,0};
  for (int i=0; i<9; i++) {
    P[ans[box[k][i][0]][box[k][i][1]]] =
```

```
          P[ans[box[k][i][0]][box[k][i][1]]]+1;
      }
      for (int n=1; n<=9; n++) {
        if (P[n] > 1)
          return true;
      }
    }
    return false;
  }
```

と定義します。これらの関数を使って、まず、ボタン「虱潰し探索」をクリックしたときの処理を定義します。

```
    b4.setOnAction((event) -> {
      solver();
      drawCanvas();
    });
```

と定義し、関数 solver() を

```
  boolean solver() {
    if (losingP()) {
      return false;
    }
    if (completeP())
      return true;
      int jj = -1, ii = -1;
      boolean flag = false;
out: for (int j=0; j<9; j++) {
        for (int i=0; i<9; i++) {
          if (ans[j][i] == 0) {
            jj = j;
            ii = i;
            flag = true;
            break out;
          }
        }
      }
      if (flag) {
        setHint();
        boolean[] S = new boolean[10];
        for (int n=1; n<=9; n++)
          S[n] = hint[jj][ii][n];
        for (int n=1; n<=9; n++) {
          if (S[n]) {
```

```
            ans[jj][ii] = n;
            if (solver())
              return true;
            ans[jj][ii] = 0;
          }
        }
      }
    return false;
  }
```

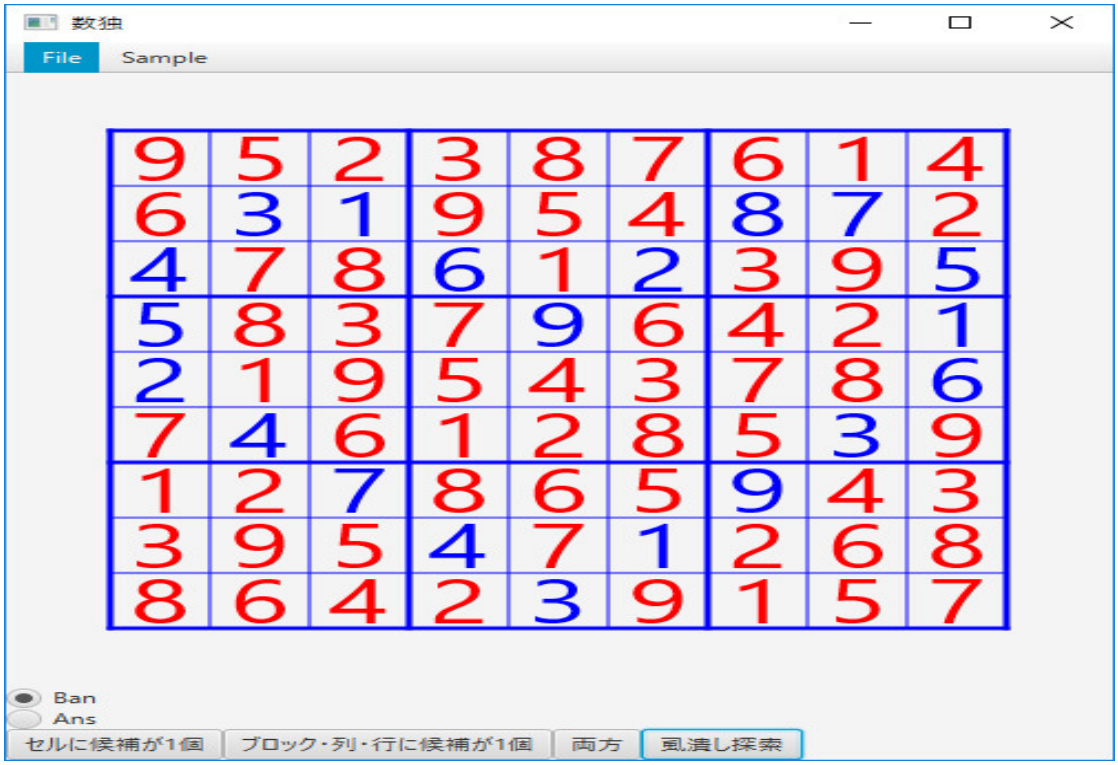
と定義します。実行し、問題を表示し、ボタン「虱潰し探索」をクリックすると

となります。これでどんな問題でも解いてくれますが、もう少し高速にしてみましょう。関数 solver() を

```
  boolean solver() {
    regularPlay();
    if (losingP()) {
      return false;
    }
    if (completeP())
      return true;
    int jj = -1, ii = -1;
    boolean flag = false;
```

```
out: for (int j=0; j<9; j++) {
      for (int i=0; i<9; i++) {
        if (ans[j][i] == 0) {
          jj = j;
          ii = i;
          flag = true;
          break out;
        }
      }
    }
    if (flag) {
      int[][] tempAns = new int[9][9];
      for (int j=0; j<9; j++) {
        for (int i=0; i<9; i++) {
          tempAns[j][i] = ans[j][i];
        }
      }
      setHint();
      boolean[] S = new boolean[10];
      for (int n=1; n<=9; n++)
        S[n] = hint[jj][ii][n];
      for (int n=1; n<=9; n++) {
        if (S[n]) {
          ans[jj][ii] = n;
          if (solver())
            return true;
          for (int j=0; j<9; j++) {
            for (int i=0; i<9; i++) {
              ans[j][i] = tempAns[j][i];
            }
          }
          ans[jj][ii] = 0;
        }
      }
    }
    return false;
  }
```

と修正します。実行し、問題を表示し、ボタン「虱潰し探索」をクリックすると



と同じ結果を表示しますが、はるかに高速になりました。

VC++ や Python では局面を印刷できるようにしましたが、Java でのやり方はまだ分かりません。

何とか VC++で作ったプログラムと同等のものを作りましたが、やはりまとまった書籍の情報がある VC++ の方が楽です。JavaFX の情報は立木秀樹、有賀妙子著「すべての人のための Java プログラミング第 3 版」にも概説が載っていますが、この情報だけでは不十分で、インターネットが頼りで、気長に情報を探せば、それを使って、 Java でもこの様にプログラミングすることは可能です。後は、これを雛形に他のパズルのプログラムも作れます。興味があれば、それぞれの命令の意味はインターネットで調べて下さい。親切な人たちが必要な情報を書き込んでくれています。

Java は初心者にとっては Python や Ruby より習得が難しいと思いますが、andoroid のアプリを作るためのプログラミング言語でもあり（最近は Kotlin という言語が出て来たみたいですが）勉強しておいて損はないです。Python も Ruby も Java も、1 日 10 時間 × 30 日間、合計 300 時間ぐらい勉強すれば、このようなプログラムが作れるようになります。多分。「数独」の解法と問題作成を解説した本には、英語の本ですが、Giulio Zambon 著「Sudoku Programming with C」Apress があります。ダウンロードしたプログラムは、VC++ では、何か所か修正しなければいけませんが、修正は簡単です。配列 box[][][] を使うことはこの本から借用しました。問題の作り方も書いていますが、既に知っていた常識的なことしか書いてなくて、棚床弘樹さんの本同様、期待したものではなかったです。

「ひとりにしてくれ」の解法解析の文書も株式会社ニコリの許可が下りましたのでアップしました。こちらも参照してください。