作ってみる

いまさらですが、私の考え

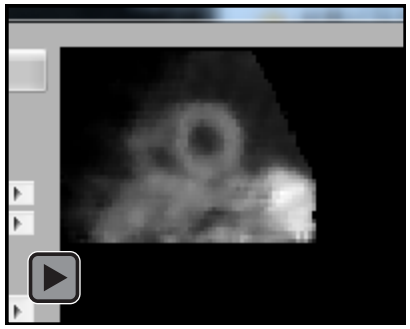
- 数回のC++講義程度では、それを使えても理解できない
- C#もC++もわからなさはそんなに変わらない
- わからないながらもいじるならC#の方が格好良い
- C#もC++と同じように、まずは手を動かすことが大事

つくってあそぼう

最近遊びでつくってご好評頂いたプログラムを紹介

つくってあそんだプログラム

くるくるまわる。



ダウンロードすれば動く。

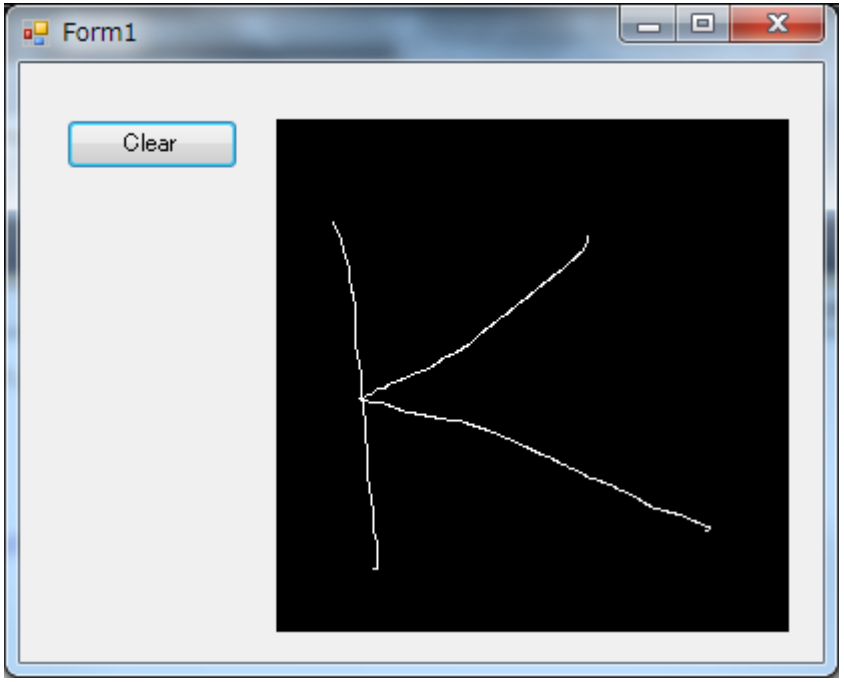
つくってあそんだプログラム

プログラムの中身は

①最初の画像表示
②ドラッグしている間、プログラムが実行されるようにして
③回転処理、処理後表示

回転の計算が面倒。
ここでは肝の②を使ったプログラムを作ってみる。

つくってあそぼう



自由描画。

ペイントでいう”鉛筆”

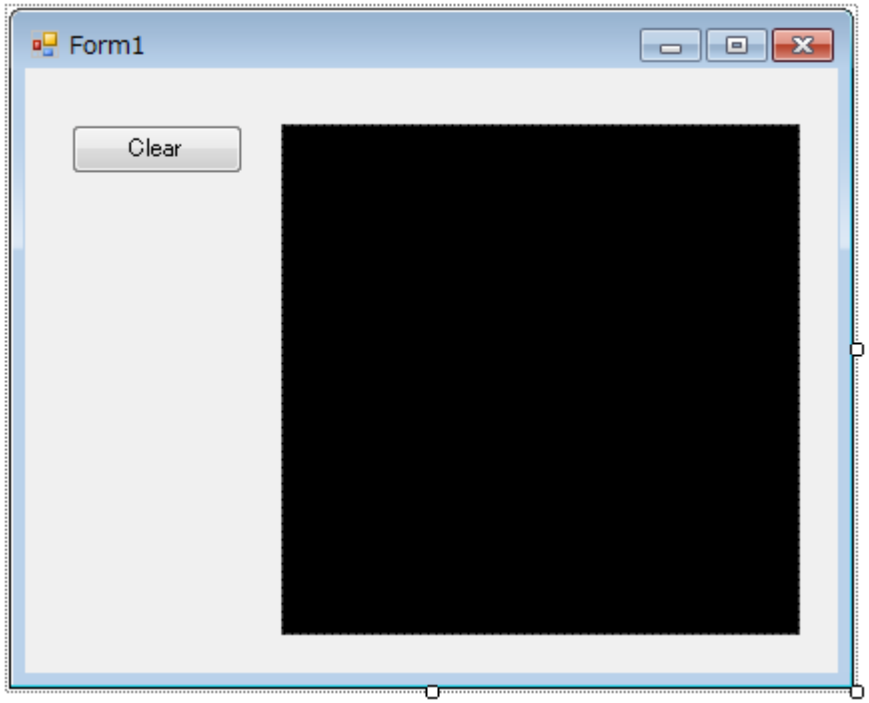
とか”ブラシ”とか。

これをつくってみる。

つくってあそぼう

必要なのは、画面をクリアするためのボタンと

表示のための picturebox



picturebox のsizeは

256, 256

にでもしておく。

プロパティでサイズ指定可能

つくってあそぼう

```
namespace tsukutteasobou
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }


        private void button1_Click(object sender, EventArgs e)
        {

        }

        private void pictureBox1_MouseDown(object sender, MouseEventArgs e)
        {

        }

        private void pictureBox1_MouseMove(object sender, MouseEventArgs e)
        {

        }

        private void pictureBox1_MouseUp(object sender, MouseEventArgs e)
        {

        }
    }
}
```

用意するイベントハンドラ。

マウスダウン（クリック）

されてからアップされるまで

マウスの動きを読む

といった流れを狙う。

つくってあそぼう

描画の方法は

①Graphics を使う

②配列に情報を入れて表示させる

の二通りが考えられるだろう。

まずは①について紹介する。

つくってあそぼう　①Graphics

Graphicsは簡単に線を引いたり、円を描いたりできて便利。

ただ、大量に処理を行おうとすると描画の遅さが浮き出る。

今回は問題ない。

```
Graphics g = pictureBox1.CreateGraphics();
```

表示させたい場所

つくってあそぼう　①Graphics

```
namespace tsukutteasobou
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        int flg = 0;

        private void button1_Click(object sender, EventArgs e)
        {
            Graphics g = pictureBox1.CreateGraphics();
            g.Clear(Color.FromArgb(0, 0, 0));
            g.Dispose();
        }

        private void pictureBox1_MouseDown(object sender, MouseEventArgs e)
        {
            Graphics g = pictureBox1.CreateGraphics();
            g.FillRectangle(Brushes.White, e.X, e.Y, 1, 1);
            g.Dispose();
            flg++;
        }

        private void pictureBox1_MouseMove(object sender, MouseEventArgs e)
        {
            if (flg == 1)
            {
                Graphics g = pictureBox1.CreateGraphics();
                g.FillRectangle(Brushes.White, e.X, e.Y, 1, 1);
                g.Dispose();
            }
        }

        private void pictureBox1_MouseUp(object sender, MouseEventArgs e)
        {
            flg = 0;
        }
    }
}
```

これで描ける。

flg でクリックされている状態かどうかをみている。

Rectangleは長方形。

Fillで塗りつぶし。

Disposeは描画終わりに書く癖をつけていた方がよいだろう。

Clearで全面塗りつぶし。

つくってあそぼう　①Graphics

いじってみればわかるが、点々になる。

これはgraphicsのせいではなく、

MouseMoveのせい（たぶん）。

これを改善したコードを次に示す。

```
namespace tsukutteasobou
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        int flg = 0;
        int x0 = 0, y0 = 0;

        private void button1_Click(object sender, EventArgs e)
        {
            Graphics g = pictureBox1.CreateGraphics();
            g.Clear(Color.FromArgb(0, 0, 0));
            g.Dispose();
        }

        private void pictureBox1_MouseDown(object sender, MouseEventArgs e)
        {
            Graphics g = pictureBox1.CreateGraphics();
            g.FillRectangle(Brushes.White, e.X, e.Y, 1, 1);
            x0 = e.X; y0 = e.Y;
            g.Dispose();
            flg = 1;
        }

        private void pictureBox1_MouseMove(object sender, MouseEventArgs e)
        {
            if (flg == 1)
            {
                Graphics g = pictureBox1.CreateGraphics();
                g.DrawLine(Pens.White, x0, y0, e.X, e.Y);
                x0 = e.X; y0 = e.Y;
                g.Dispose();
            }
        }

        private void pictureBox1_MouseUp(object sender, MouseEventArgs e)
        {
            flg = 0;
        }
    }
}
```

これで問題なく動作する。

DrawLineで一つ前に打った点と今の点をつなげる。

MouseDownでFillRectangleを残しているのは、最初のドットを打つため。

消した場合はクリックしただけだと何も表示されなくなる。

つくってあそぼう　①Graphics

動作は確認したので、紹介したコードで動く。

今後ROIツール等作成する際には、MouseDownの扱いに慣れる必要がある。

flgをたてて、望みのコードのみ実行させるという手法は今後でてくると思うので、おさえておくとよい。

つくってあそぼう　②配列

次に配列を使った描画を紹介する。

結構面倒。

細かいところは説明しない。

今の目標は

・マウスの扱い

・配列やgraphicsの扱いとその表示のさせ方

を理解することなので、そこをおさえる。

```
namespace tsukutteasobou
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        int flg = 0;
        int[,] image = new int[256, 256];
        int x0 = 0, y0 = 0;

        private void button1_Click(object sender, EventArgs e)
        {
            image = new int[256, 256];
            Bitmap bmp = new Bitmap(256, 256);
            for (int j = 0; j < 256; j++){for (int i = 0; i < 256; i++){
                Color col = Color.White;
                if (image[i, j] == 1) bmp.SetPixel(i, j, col);
            }}
            pictureBox1.Image = bmp;
            pictureBox1.Refresh();
        }

        private void pictureBox1_MouseDown(object sender, MouseEventArgs e)
        {
            image[e.X, e.Y] = 1;
            Bitmap bmp = new Bitmap(256, 256);
            for (int j = 0; j < 256; j++){for (int i = 0; i < 256; i++){
                Color col = Color.White;
                if (image[i, j] == 1) bmp.SetPixel(i, j, col);
            }}
            pictureBox1.Image = bmp;
            pictureBox1.Refresh();
            flg = 1;
        }

        private void pictureBox1_MouseMove(object sender, MouseEventArgs e)
        {
            if (flg == 1)
            {
                image[e.X, e.Y] = 1;
                Bitmap bmp = new Bitmap(256, 256);
                for (int j = 0; j < 256; j++){for (int i = 0; i < 256; i++){
                   Color col = Color.White;
                   if (image[i, j] == 1) bmp.SetPixel(i, j, col);
                }}
                pictureBox1.Image = bmp;
                pictureBox1.Refresh();
            }
        }

        private void pictureBox1_MouseUp(object sender, MouseEventArgs e)
        {
            flg = 0;
        }
    }
}
```

点々バージョン。

描画は例のBitmapPlusは

使わずに行っている。

Bitmap bmp = …

Color col = …

という使い方に慣れていこう。

画像表示の流れについても

「そういうものだ」と

思ってしまうべき。

つくってあそぼう　②配列

ところで

画像表示のための同じコードが繰り返し出てくる。

これは冗長であるから、なにか対策をとりたい。

そんなときにclassを使う。

つくってあそぼう　②配列

```
    private void button1_Click(object sender, EventArgs e)
    {
        image = new int[256, 256];
        D.Disp(image, pictureBox1);
    }

    private void pictureBox1_MouseDown(object sender, MouseEventArgs e)
    {
        image[e.X, e.Y] = 1;
        D.Disp(image, pictureBox1);
        flg = 1;
    }

    private void pictureBox1_MouseMove(object sender, MouseEventArgs e)
    {
        if (flg == 1)
        {
            image[e.X, e.Y] = 1;
            D.Disp(image, pictureBox1);
        }
    }

    private void pictureBox1_MouseUp(object sender, MouseEventArgs e)
    {
        flg = 0;
    }
}

class D
{
    public static void Disp(int[,] img, PictureBox picbox)
    {
        Bitmap bmp = new Bitmap(256, 256);
        for (int j = 0; j < 256; j++)
        {
            for (int i = 0; i < 256; i++)
            {
                Color col = Color.White;
                if (img[i, j] == 1) bmp.SetPixel(i, j, col);
            }
        }
        picbox.Image = bmp;
        picbox.Refresh();
    }
}
```

すっきり。

最初からこれをつかいこなされたら私が焦るので、

あまり無理してこのようなコーディングにしなくてもよいと思います。

なんとなくプログラムが打てるようになってきて、繰り返しがうざったいと感じたら挑戦してみよう。

次が面倒な直線補間。

直線補間のコードはこのようになります。

```
    static public void Line(int x1, int y1, int x2, int y2, PictureBox picbox, ref int[,] line)
    {
        int xm1 = 0, xm2 = 0, ym1 = 0, ym2 = 0;
        int yyy = 0;
        if (x1 < x2) { xm1 = x1; xm2 = x2; yyy = y1; }
        if (x1 > x2) { xm2 = x1; xm1 = x2; yyy = y2; }
        if (y1 < y2) { ym1 = y1; ym2 = y2; }
        if (y1 > y2) { ym2 = y1; ym1 = y2; }
        if (x1 == x2)
        {
            for (int k = ym1; k <= ym2; k++)
            {
                int a = x1;
                int b = k;
                line[a, b] = 1;
            } goto END;
        }
        if (y1 == y2)
        {
            for (int ii = xm1; ii <= xm2; ii++)
            {
                int a = ii;
                int b = y1;
                line[a, b] = 1;
            } goto END;
        }
        double xym = (double)(y1 - y2) / (x1 - x2);//y軸は正負反対だから符号も反対
        for (int ii = xm1; ii <= xm2; ii++)
        {
            for (int k = ym1; k <= ym2; k++)
            {
                if ((ii - xm1) == (int)(((double)k - (double)yyy) / xym))
                {
                    int a = ii;
                    int b = k;
                    line[a, b] = 1;
                }
            }
        }
        for (int k = ym1; k <= ym2; k++)
        {
            for (int ii = xm1; ii <= xm2; ii++)
            {
                if (k == yyy + (int)(xym * (double)(ii - xm1)))
                {
                    int a = ii;
                    int b = k;
                    line[a, b] = 1;
                }
            }
        }
    END: ;
    }
}
```

もっときれいなコーディングがあるでしょうが、これでも動きます。

直線補間を含むコード。

クラスは折りたたんでいます。

```
namespace tsukutteasobou
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        int flg = 0;
        int[,] image = new int[256, 256];
        int x0 = 0, y0 = 0;

        private void button1_Click(object sender, EventArgs e)
        {
            image = new int[256, 256];
            D.Disp(image, pictureBox1);
        }

        private void pictureBox1_MouseDown(object sender, MouseEventArgs e)
        {
            image[e.X, e.Y] = 1;
            D.Disp(image, pictureBox1);
            x0 = e.X; y0 = e.Y;
            flg = 1;
        }

        private void pictureBox1_MouseMove(object sender, MouseEventArgs e)
        {
            if (flg == 1)
            {
                image[e.X, e.Y] = 1;
                D.Line(x0, y0, e.X, e.Y, pictureBox1, ref image);
                D.Disp(image, pictureBox1);
                x0 = e.X; y0 = e.Y;
            }
        }

        private void pictureBox1_MouseUp(object sender, MouseEventArgs e)
        {
            flg = 0;
        }
    }

    class D
    {
        public static void Disp(int[,] img, PictureBox picbox)...
        static public void Line(int x1, int y1, int x2, int y2, PictureBox picbox, ref int[,] line)...
    }
}
```

つくってあそぼう

①と②で異なる点は

マウスの軌跡が保存されているかいないか

という点。

保存されていれば、それをいじれる。

手囲いROIツールなどの作成が可能である。

つくってあそぼう

Graphicsは簡単なので使い勝手が良い。

状況に応じて使い分けよう。

最後のコードはアップしておきます。

おわり