```
//-----------------------------------------------------------------------------------------------------------
//                      GENES  Version 1.00     突然変異と自然淘汰による仮想生命進化シミュレーション
//
//                      2019 (C) Radiun.net  Takashi Kato
//
//                      Ver 1.00    2019/03/03  初版リリース
//-----------------------------------------------------------------------------------------------------------

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace VirtualLife
{
    public partial class Form1 : Form
    {
        //------------------------------------------------ グローバル変数クラスの定義 ---------------------------------
        public static class St
        {
```

```
    public static Random crnd = new System.Random();

    public static byte[,] gene = new byte[4096, 20];       // Gene of 4096 lives
    public static ushort[] eg = new ushort[4096];          // Energy of Life
    public static ushort[] cells = new ushort[4096];       // Count of Cells
    public static ushort[] cx = new ushort[4096];          // Locate X
    public static ushort[] cy = new ushort[4096];          // Locate X

    public static ushort[] fx = new ushort[4096];          // Locate X of Food
    public static ushort[] fy = new ushort[4096];          // Locate X of Food

    public static ulong[] age = new ulong[4096];           // age of Life      <-- 寿命、年齢 対応
}
//---------------------------------------------- 初期値の定義 ----------------------------------------------
 public Form1()
{
    InitializeComponent();

    //   進行速度を決めるタイマー
    timer1.Interval = 10;                          // Timer Interval 10ms

    //   最初の10匹の遺伝情報
    for (int n = 0; n < 10; n++)
    {
        St.eg[n] = 0x2000;                         // 0:Dead >1:Alive(Energy)
```

```
        St.cells[n] = 1;                        // 1 cell

        St.cx[n] = 500;                         // Locate X
        St.cy[n] = 350;                         // Locate Y

        St.gene[n, 0] = 0xD0;                   // Gene of Cell 1

        St.age[n] = (ulong)numericUpDown3.Value;        // Reset age of life  <-- 寿命、年齢 対応
    }
}

private void button1_Click(object sender, EventArgs e)
{
    //  START Button
    timer1.Enabled = true;
}

private void button2_Click(object sender, EventArgs e)
{
    //  STOP Button
    timer1.Enabled = false;
}

//---------------------------------------- メインルーチン（タイマーイベント） ----------------------------------------
private void timer1_Tick(object sender, EventArgs e)
```

```
{
    //   グラフィックス・オブジェクトの作成
    Graphics g1;
    Bitmap DrawArea;

    int pw1 = pictureBox1.Size.Width;                    // Get Picture Box size
    int ph1 = pictureBox1.Size.Height;

    int pws1 = pw1 / 10;                                 // Size of Div of Hrizon
    int pwh1 = pw1 / 2;                                  // Center of Hrizon
    int pwb1 = pwh1 - 5 * pws1;                          // Ref Line  of Hrizon

    int phs1 = ph1 / 10;                                 // Size of Div of Vertical
    int phh1 = ph1 / 2;                                  // Center of Vertical
    int phb1 = phh1 - 5 * phs1;                          // Ref Line  of Vertical

    int count = 0;
    int fcount = 0;

    //   描画色の指定
    Pen bpen = new Pen(Color.Blue, 1);
    Pen rpen = new Pen(Color.Red, 1);
    Pen gpen = new Pen(Color.DarkGreen, 1);
    Pen apen = new Pen(Color.Aqua, 1);
```

```
//   フォームから給餌最大値を得る
int foodmax = (int)numericUpDown1.Value;

//   フォームから突然変異確率を得る
int mutation = (int)numericUpDown2.Value;

//   フォームから捕食者設定を得る
bool predator = checkBox1.Checked;

//   フォームから捕食者雑食設定を得る
bool omnivores = checkBox2.Checked;

//------------------------------------------------ 画面を初期化 ------------------------------------------------
pictureBox1.Image = null;                                    // Picture Box Clear

DrawArea = new Bitmap(pw1, ph1);                             // Picture Box Defined
pictureBox1.Image = DrawArea;

g1 = Graphics.FromImage(DrawArea);                           // Make Graphics Object

for (int y = phb1; y <= ph1; y = y + phs1)                   // Draw Horizon Lines
{
    g1.DrawLine(gpen, 0, y, pw1, y);
}
```

```
for (int w = pwb1; w <= pw1; w = w + pws1)                    // Draw Vertical Lines
{
    g1.DrawLine(gpen, w, 0, w, ph1);
}

//------------------------------------------------ 生命の活動 ------------------------------------------------
for (int n = 0; n < 4096; n++)
{
    int size = 3;
    byte itv = 0;
    int cells = St.cells[n];

    //  エネルギーがあれば生きていると判断
    if (St.eg[n] > 0)
    {
        //   個体数のカウント
        count++;

        //   細胞の移動を決める確率を遺伝子から読出
        itv = (byte)(St.gene[n, 0] & 0x03);

        //   移動方向を決める乱数の発生
        int rdir = St.crnd.Next(4 + itv);

        //   乱数が4以上の場合は0番細胞の遺伝情報から移動方向を得る
```

```
if (rdir >= 4) rdir = (St.gene[n, 0] & 0x0C) / 4;

//  元の位置の検出
int x = St.cx[n];
int y = St.cy[n];

//  新しい位置
if (rdir == 0) x--;
if (rdir == 1) x++;
if (rdir == 2) y--;
if (rdir == 3) y++;

//  境界処理
if (x <= 0) x = pw1 + x;
if (x >= pw1) x = x - pw1;
if (y <= 0) y = ph1 + y;
if (y >= ph1) y = y - ph1;

//  新しい位置を書き込む
St.cx[n] = (ushort)x;
St.cy[n] = (ushort)y;

//------------------------------------- 個体の移動 ------------------------------------------------
for (int m = 0; m < cells; m++)
{
```

```
    //   結合方向の遺伝子を読む
    byte dna = St.gene[n, m];
    byte dir = (byte)((dna & 0xC0) / 0x40);

    //   新しい位置
    if (dir == 0) x = x - size;
    if (dir == 1) x = x + size;
    if (dir == 2) y = y - size;
    if (dir == 3) y = y + size;

    //   遺伝情報を反映した色を付ける
    int R = (byte)((dna & 0xC0) / 0x40) * 64 + 63;      // RED
    int G = (byte)((dna & 0x0C) / 0x04) * 64 + 63;      // GREEN
    int B = (byte)((dna & 0x03) / 0x01) * 64 + 63;      // BLUE
    Pen rgb = new Pen(Color.FromArgb(R, G, B), 1);

    //   細胞の描画
    g1.DrawEllipse(rgb, x, y, size, size);
}

//------------------------------------ 捕食によるエネルギー獲得 ------------------------------------
for (int f = 0; f < 4096; f++)
{
    //   個体の原点座標
    int tx = St.cx[n];
```

```
int ty = St.cy[n];
int cl = St.cells[n];

//  フードの座標
int fdx = St.fx[f];
int fdy = St.fy[f];

//  他個体の座標
int adx = St.cx[f];
int ady = St.cy[f];
int acl = St.cells[f];

for (int m = 0; m < cells; m++)
{
   //  結合方向の遺伝子を読む
   byte dna = St.gene[n, m];
   byte dir = (byte)((dna & 0xC0) / 0x40);

   //  新しい位置
   if (dir == 0) tx = tx - size;
   if (dir == 1) tx = tx + size;
   if (dir == 2) ty = ty - size;
   if (dir == 3) ty = ty + size;

   //  単細胞または雑食設定時に細胞の大きさの範囲内にフードがあるか判定
```

```
if ((omnivores == true) | (cells == 1))
{
    int dist = (int)(Math.Sqrt((tx - fdx) * (tx - fdx) + (ty - fdy) * (ty - fdy)));
    if (dist <= size)
    {
        //  エネルギー充填（x=0付近の誤作動防止）
        if (tx > 10) St.eg[n] = (ushort)(St.eg[n] + 500);

        //  フードを消費、消滅
        St.fx[f] = 0;
        St.fx[f] = 0;
    }
}

//  細胞の大きさの範囲内に自分よりも弱い個体があるか判定
if (predator == true)
{
    int dist = (int)(Math.Sqrt((tx - adx) * (tx - adx) + (ty - ady) * (ty - ady)));
    if ((dist <= size) & (cl > acl))
    {
        //  エネルギー搾取（x=0付近の誤作動防止）
        if (tx > 10) St.eg[n] = (ushort)(St.eg[n] + St.eg[f]);

        //  被捕食個体消滅
        St.eg[f] = 0;
```

```
                St.cx[f] = 0;
                St.cy[f] = 0;
            }
        }
    }
}

//------------------------------------- 細胞数に応じたエネルギーの消費 -------------------------------------
int eg = St.eg[n];
eg = eg - cells;
if (eg < 0) eg = 0;
St.eg[n] = (ushort)eg;

//------------------------------------- エネルギーが閾値を越えたら増殖 -------------------------------------
//   細胞分裂の閾値を得る
byte cdv = (byte)((St.gene[n, 0] & 0x30) / 0x10 + 1);

if (St.eg[n] > 0x1000 * cdv * cells)
{
    //   ランダムに番号を指定
    int nn = St.crnd.Next(4096);

    while(St.eg[nn] > 0 && count < 4096) nn = St.crnd.Next(4096);    // <-- 生きているものを上書きしない

    //   遺伝情報をコピー
```

```
St.eg[n] = (ushort)(0x500 * cdv * cells);
St.eg[nn] = (ushort)(0x500 * cdv * cells);
St.cx[nn] = St.cx[n];
St.cy[nn] = St.cy[n];
St.cells[nn] = St.cells[n];

St.age[n] = (ulong)numericUpDown3.Value;            // 分裂の元を年齢リセット
St.age[nn] = (ulong)numericUpDown3.Value;           // 分裂先の年齢リセット

for (int m = 0; m < cells; m++)
{
    St.gene[nn, m] = St.gene[n, m];
}

//   突然変異の発生確立
int rnd1 = St.crnd.Next(mutation);
if (rnd1 == 0)
{
    //   乱数によって突然変異する細胞を選ぶ
    int rnd2 = St.crnd.Next(cells + 1);
    //   乱数によって突然変異の内容を決定
    int rnd3 = St.crnd.Next(256);
    //   突然変異の内容を遺伝子に書込
    St.gene[nn, rnd2] = (byte)rnd3;
    //   セル数の変更
```

```
                    if (St.cells[nn] <= rnd2) St.cells[nn]++;
                    //   ゼロの遺伝子の場合、切断して短くする
                    if (rnd3 == 0) St.cells[nn] = (ushort)(rnd2 + 1);
                }
            }
            //------------------------------------ 年齢カウントダウン ------------------------------------
            if (checkBox3.Checked == true)
            {
                if (St.age[n] > 0) St.age[n]--;
                if (St.age[n] == 0) St.eg[n] = 0;     // 寿命対応
            }
        }

        //   フードの描画
        if (St.fx[n] > 0 & St.fy[n] > 0)
        {
            g1.DrawEllipse(bpen, St.fx[n], St.fy[n], 1, 1);
            fcount++;
        }
    }

//      フードを増やせるなら
    if (fcount < foodmax)
    {
        //   乱数によるフードの散布
```

```
            int l = St.crnd.Next(foodmax);
        //  空いている管理番号に増やす
            while (St.fx[l] > 0 & St.fy[l] > 0)
                l = St.crnd.Next(foodmax);
            ushort fx = (ushort)St.crnd.Next(pw1);
            ushort fy = (ushort)St.crnd.Next(ph1);
            St.fx[l] = fx;
            St.fy[l] = fy;
        }
        //   個体数の表示
        textBox1.Text = count.ToString();

        //   フード数の表示
        textBox2.Text = fcount.ToString();
    }

    //-------------------------------------------------------------------------------------------------------------
    //                                        マウスでクリックした個体の遺伝子を見る
    //-------------------------------------------------------------------------------------------------------------
    private void pictureBox1_MouseDown(object sender, MouseEventArgs e)
    {
        int tn = 0;

        //   マウスクリックした座標
        int mx = e.X;
```

```
int my = e.Y;

//　近くの個体を補足
for (int n = 0; n < 4096; n++)
{
    //　ターゲット個体の座標
    int tx = St.cx[n];
    int ty = St.cy[n];

    //　距離を計算
    int dist = (int)(Math.Sqrt((tx - mx) * (tx - mx) + (ty - my) * (ty - my)));

    //　ターゲット判定
    if (dist < 10) tn = n;
}

//　ターゲット遺伝子の番号を表示
textBox3.Text = tn.ToString();

//　エネルギー残量の表示
textBox4.Text = St.eg[tn].ToString();

//　遺伝子内容の表示
int dna = St.gene[tn, 0];
textBox5.Text = dna.ToString("X2");
```

```
dna = St.gene[tn, 1];
textBox6.Text = dna.ToString("X2");
dna = St.gene[tn, 2];
textBox7.Text = dna.ToString("X2");
dna = St.gene[tn, 3];
textBox8.Text = dna.ToString("X2");
dna = St.gene[tn, 4];
textBox9.Text = dna.ToString("X2");
dna = St.gene[tn, 5];
textBox10.Text = dna.ToString("X2");
dna = St.gene[tn, 6];
textBox11.Text = dna.ToString("X2");
dna = St.gene[tn, 7];
textBox12.Text = dna.ToString("X2");
dna = St.gene[tn, 8];
textBox13.Text = dna.ToString("X2");
dna = St.gene[tn, 9];
textBox14.Text = dna.ToString("X2");

//　セル数の表示
int cells = St.cells[tn];
textBox15.Text = cells.ToString();

//　グラフィックス・オブジェクトの作成
Graphics g2;
```

```
Bitmap DrawArea;

int pw2 = pictureBox2.Size.Width;                                   // Get Picture Box size
int ph2 = pictureBox2.Size.Height;

//------------------------------------------------- 画面を初期化 ----------------------------------------------
pictureBox2.Image = null;                                           // Picture Box Clear

DrawArea = new Bitmap(pw2, ph2);                                    // Picture Box Defined
pictureBox2.Image = DrawArea;

g2 = Graphics.FromImage(DrawArea);                                  // Make Graphics Object

int size = 5;
int x = pw2 / 2;
int y = ph2 / 2;

//   個体サンプルの描画
for (int m = 0; m < cells; m++)
{
   //   結合方向の遺伝子を読む
   dna = St.gene[tn, m];
   byte dir = (byte)((dna & 0xC0) / 0x40);

   //   新しい位置
```

```
                if (dir == 0) x = x - size;
                if (dir == 1) x = x + size;
                if (dir == 2) y = y - size;
                if (dir == 3) y = y + size;

                //  遺伝情報を反映した色を付ける
                int R = (byte)((dna & 0xC0) / 0x40) * 64 + 63;      // RED
                int G = (byte)((dna & 0x0C) / 0x04) * 64 + 63;      // GREEN
                int B = (byte)((dna & 0x03) / 0x01) * 64 + 63;      // BLUE
                Pen rgb = new Pen(Color.FromArgb(R, G, B), 1);

                //  細胞の描画
                g2.DrawEllipse(rgb, x, y, size, size);
            }
        }

        private void Form1_Load(object sender, EventArgs e)
        {

        }
    }
}
```