formStart.cs

```
//=============================================================================
//
// File Name    :  formStart.cs
// Purpose      :  To give a broad overview of the program and provides options
//              :  for creating a homogeneous or heterogeneous simulation
//
// Author       :  Micheal Wang
//              Albuquerque Academy
// Created on   :  December 2007
// Copyright    :  All Rights Reserved.
//
//=============================================================================

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

namespace NanoSimulator
{
   public partial class form_start : Form
   {
      /**
       * constructor
       */
      public form_start()
      {
         InitializeComponent();
      }

      //---------------------------------------------------------------------
      /**
       *
       */
      private void form_start_Load(object sender, EventArgs e)
      {

      }

      //---------------------------------------------------------------------
      /**
       * Calls either the heterogeneous or homogeneous input forms depending
       * on which of the choices is selected
       */
      private void bt_start_Click(object sender, EventArgs e)
      {
         if (rbt_homo.Checked.Equals(true))
         {
            formHomogeneous fhomo = new formHomogeneous();
              fhomo.Show();
         }
         else
         {
            formHeterogeneous fhetero = new formHeterogeneous();
              fhetero.Show();
         }
```

```
        }

        //---------------------------------------------------------------------
        /**
         * Exits program
         */
        private void bt_quit_Click(object sender, EventArgs e)
        {
            this.Dispose();
        }

        //---------------------------------------------------------------------
        /**
         * This sets the homogeneous option to true and the heterogeneous option
         * to false
         */
        private void rbt_homo_CheckedChanged(object sender, EventArgs e)
        {
            rbt_homo.Checked.Equals(true);
            rbt_hetero.Checked.Equals(false);
        }

        //---------------------------------------------------------------------
        /**
         * This sets the heterogeneous option to true and the homogeneous option
         * to false
         */
        private void rbt_hetero_CheckedChanged(object sender, EventArgs e)
        {
            rbt_homo.Checked.Equals(false);
            rbt_hetero.Checked.Equals(true);
        }
        //---------------------------------------------------------------------
    }
}
```

formHomogeneous.cs

```
//=============================================================================
//
// File Name    :  formHomogeneous.cs
// Purpose      :  To create an interface for changing parameter values
//
// Author       :  Micheal Wang
//                 Albuquerque Academy
// Created on   :  December 2007
// Copyright    :  All Rights Reserved.
//
//=============================================================================

using System;
using System.IO;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace NanoSimulator
{
    public partial class formHomogeneous : Form
    {
```

```
Parameters pHomo;

/**
 * constructor
 */
public formHomogeneous()
{
   pHomo = new Parameters();
   InitializeComponent();
}

//---------------------------------------------------------------------
/**
 *
 */
private void formHomogeneous_Load(object sender, EventArgs e)
{

}

//---------------------------------------------------------------------
/**
 * This exits the homogeneous input screen
 */
private void closeToolStripMenuItem_Click(object sender, EventArgs e)
{
   this.Dispose();
}

//---------------------------------------------------------------------
/**
 * This exits the homogeneous input screen
 */
private void bt_close_Click(object sender, EventArgs e)
{
   this.Dispose();
}

//---------------------------------------------------------------------
/**
 * This resets the parameters to their default values
 */
private void bt_reset_Click(object sender, EventArgs e)
{
   tb_C1.Text = "0";
   tb_C2.Text = "0";
   tb_Q1.Text = "1.0";
   tb_Q2.Text = "1.0";
   tb_Q3.Text = "1.0";
   tb_S.Text = "1.0";
   tb_H.Text = "1.0";
   tb_bond012.Text = "2.9";
   tb_bond112.Text = "0.0";
   tb_bond013.Text = "2.9";
   tb_bond113.Text = "0.0";
   tb_bond023.Text = "2.9";
   tb_bond123.Text = "0.0";
   tb_initTemp.Text = "400";
   tb_noise.Text = "0";
   tb_delta.Text = "0.3";
   tb_deltaE.Text = "55.1";
}
```

```
//-----------------------------------------------------------------------
/**
 * This stores all of the parameter values and calls/opens the
 * simulation window
 */
private void bt_startSim_Click(object sender, EventArgs e)
{
   pHomo.homoC1 = Double.Parse(tb_C1.Text);
   pHomo.homoC2 = Double.Parse(tb_C2.Text);

   pHomo.iTemperature = Double.Parse(tb_initTemp.Text);
   pHomo.sqLength = Int16.Parse(tb_simSqLength.Text);
   pHomo.delta = Double.Parse(tb_delta.Text);

   pHomo.cMatSize = 2 * pHomo.sqLength * pHomo.sqLength;
   pHomo.kSize = pHomo.sqLength * pHomo.sqLength;

   pHomo.Q1 = Double.Parse(tb_Q1.Text);
   pHomo.Q2 = Double.Parse(tb_Q2.Text);
   pHomo.Q3 = Double.Parse(tb_Q3.Text);

   pHomo.S = Double.Parse(tb_S.Text);
   pHomo.H = Double.Parse(tb_H.Text);

   pHomo.o_0_12 = Double.Parse(tb_bond012.Text);
   pHomo.o_1_12 = Double.Parse(tb_bond112.Text);
   pHomo.o_0_13 = Double.Parse(tb_bond013.Text);
   pHomo.o_1_13 = Double.Parse(tb_bond113.Text);
   pHomo.o_0_23 = Double.Parse(tb_bond023.Text);
   pHomo.o_1_23 = Double.Parse(tb_bond123.Text);
   pHomo.a_12 = Double.Parse(tb_alpha12.Text);
   pHomo.a_13 = Double.Parse(tb_alpha13.Text);
   pHomo.a_23 = Double.Parse(tb_alpha23.Text);
   pHomo.beta = Double.Parse(tb_beta.Text);

   pHomo.activationEnergy = Double.Parse(tb_deltaE.Text);

   pHomo.percentNoise = Double.Parse(tb_noise.Text) / 100;

   formSimWindow fsm = new formSimWindow(pHomo);
     fsm.homogeneousArray();
     fsm.Show();
}

//-----------------------------------------------------------------------
/**
 * This opens the browse dialog box where the user can choose a
 * parameter file
 */
private void bt_parameterBrowse_Click(object sender, EventArgs e)
{
   openParameterFile();
}

//-----------------------------------------------------------------------
/**
 * This opens the browse dialog box where the user can choose a
 * parameter file
 */
private void openToolStripMenuItem_Click(object sender, EventArgs e)
{
```

```
    openParameterFile();
}

//---------------------------------------------------------------------
/**
 * This reloads the parameters from the selected file
 */
private void bt_reload_Click(object sender, EventArgs e)
{
    readInParam();
}

//---------------------------------------------------------------------
/**
 * This opens the save dialog box where the user can save the parameters
 */
private void saveToolStripMenuItem_Click(object sender, EventArgs e)
{
    SaveFileDialog sfd = new SaveFileDialog();
    sfd.Filter = "Text File|*.txt";
    sfd.Title = "Save Image as Text File";
    sfd.FileName = "";

    if (sfd.ShowDialog() == DialogResult.OK)
    {
        System.IO.StreamWriter writer = new System.IO.StreamWriter(sfd.FileName);

        writer.WriteLine(tb_C1.Text);
        writer.WriteLine(tb_C2.Text);
        writer.WriteLine(tb_Q1.Text);
        writer.WriteLine(tb_Q2.Text);
        writer.WriteLine(tb_Q3.Text);
        writer.WriteLine(tb_S.Text);
        writer.WriteLine(tb_H.Text);
        writer.WriteLine(tb_bond012.Text);
        writer.WriteLine(tb_bond112.Text);
        writer.WriteLine(tb_bond013.Text);
        writer.WriteLine(tb_bond113.Text);
        writer.WriteLine(tb_bond023.Text);
        writer.WriteLine(tb_bond123.Text);
        writer.WriteLine(tb_initTemp.Text);
        writer.WriteLine(tb_noise.Text);
        writer.WriteLine(tb_delta.Text);
        writer.WriteLine(tb_alpha12.Text);
        writer.WriteLine(tb_alpha13.Text);
        writer.WriteLine(tb_alpha23.Text);
        writer.WriteLine(tb_beta.Text);
        writer.WriteLine(tb_deltaE.Text);

        writer.Close();
    }
}

//---------------------------------------------------------------------
/**
 * This reduces code since the same code for opening a file is used
 * more than once
 */
private void openParameterFile()
{
    OpenFileDialog ofd = new OpenFileDialog();
    ofd.Filter = "Text File|*.txt";
```

```
            ofd.Title = "Open Text File";
            ofd.FileName = "";

            if (ofd.ShowDialog() == DialogResult.OK)
            {
                tb_parameterFile.Text = ofd.FileName;

                readInParam();
            }
        }

        //---------------------------------------------------------------------
        /**
         * This reads in the parameters from the selected file
         */
        private void readInParam()
        {
            if (tb_parameterFile.Text.Equals(""))
            {
            }
            else
            {
                StreamReader paramRd = new StreamReader(tb_parameterFile.Text);

                tb_C1.Text = paramRd.ReadLine();
                tb_C2.Text = paramRd.ReadLine();
                tb_Q1.Text = paramRd.ReadLine();
                tb_Q2.Text = paramRd.ReadLine();
                tb_Q3.Text = paramRd.ReadLine();
                tb_S.Text = paramRd.ReadLine();
                tb_H.Text = paramRd.ReadLine();
                tb_bond012.Text = paramRd.ReadLine();
                tb_bond112.Text = paramRd.ReadLine();
                tb_bond013.Text = paramRd.ReadLine();
                tb_bond113.Text = paramRd.ReadLine();
                tb_bond023.Text = paramRd.ReadLine();
                tb_bond123.Text = paramRd.ReadLine();
                tb_initTemp.Text = paramRd.ReadLine();
                tb_noise.Text = paramRd.ReadLine();
                tb_delta.Text = paramRd.ReadLine();
                tb_alpha12.Text = paramRd.ReadLine();
                tb_alpha13.Text = paramRd.ReadLine();
                tb_alpha23.Text = paramRd.ReadLine();
                tb_beta.Text = paramRd.ReadLine();
                tb_deltaE.Text = paramRd.ReadLine();

                paramRd.Close();
            }
        }
        //---------------------------------------------------------------------
    }
}
```

formHeterogeneous.cs

```
//=============================================================================
//
// File Name    :  formHeterogeneous.cs
// Purpose      :  To create an interface for changing parameter values and
//              :  selcting a prescibed pattern
//
// Author       :  Micheal Wang
//              Albuquerque Academy
```

```
// Created on   :  December 2007
// Copyright    :  All Rights Reserved.
//
//=============================================================================

using System;
using System.IO;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace NanoSimulator
{
   public partial class formHeterogeneous : Form
   {
      Parameters pHetero;
      OpenFileDialog ofd = new OpenFileDialog();

      /**
       * constructor
       */
      public formHeterogeneous()
      {
         pHetero = new Parameters();
         InitializeComponent();
      }

      //---------------------------------------------------------------
      /**
       *
       */
      private void formHeterogeneous_Load(object sender, EventArgs e)
      {

      }

      //---------------------------------------------------------------
      /**
       * This closes the heterogeneous input screen
       */
      private void closeToolStripMenuItem_Click(object sender, EventArgs e)
      {
         this.Dispose();
      }

      //---------------------------------------------------------------
      /**
       * This also closes the heterogeneous input screen
       */
      private void bt_close_Click(object sender, EventArgs e)
      {
         this.Dispose();
      }

      //---------------------------------------------------------------
      /**
       * This resets the parameters to their default values
       */
      private void bt_reset_Click(object sender, EventArgs e)
```

```
        {
            tb_Q1.Text = "1.0";
            tb_Q2.Text = "1.0";
            tb_Q3.Text = "1.0";
            tb_S.Text = "1.0";
            tb_H.Text = "1.0";
            tb_bond012.Text = "2.9";
            tb_bond112.Text = "0.0";
            tb_bond013.Text = "2.9";
            tb_bond113.Text = "0.0";
            tb_bond023.Text = "2.9";
            tb_bond123.Text = "0.0";
            tb_initTemp.Text = "400";
            tb_noise.Text = "0";
            tb_delta.Text = "0.3";
            tb_deltaE.Text = "55.1";
        }

        //-----------------------------------------------------------------
        /**
         * This stores the parameter inputs and checks to make sure that:
         * (1) there are two pattern files
         * (2) the size of the simulation squares are a power of two
         * (3) the size of the pattern file correspondes to the size input
         */
        private void bt_startSim_Click(object sender, EventArgs e)
        {
            pHetero.Q1 = Double.Parse(tb_Q1.Text);
            pHetero.Q2 = Double.Parse(tb_Q2.Text);
            pHetero.Q3 = Double.Parse(tb_Q3.Text);

            pHetero.iTemperature = Double.Parse(tb_initTemp.Text);
            pHetero.sqLength = Int16.Parse(tb_simSqLength.Text);
            pHetero.delta = Double.Parse(tb_delta.Text);

            pHetero.cMatSize = 2 * pHetero.sqLength * pHetero.sqLength;
            pHetero.kSize = pHetero.sqLength * pHetero.sqLength;

            pHetero.S = Double.Parse(tb_S.Text);
            pHetero.H = Double.Parse(tb_H.Text);

            pHetero.o_0_12 = Double.Parse(tb_bond012.Text);
            pHetero.o_1_12 = Double.Parse(tb_bond112.Text);
            pHetero.o_0_13 = Double.Parse(tb_bond013.Text);
            pHetero.o_1_13 = Double.Parse(tb_bond113.Text);
            pHetero.o_0_23 = Double.Parse(tb_bond023.Text);
            pHetero.o_1_23 = Double.Parse(tb_bond123.Text);
            pHetero.a_12 = Double.Parse(tb_alpha12.Text);
            pHetero.a_13 = Double.Parse(tb_alpha13.Text);
            pHetero.a_23 = Double.Parse(tb_alpha23.Text);
            pHetero.beta = Double.Parse(tb_beta.Text);

            pHetero.activationEnergy = Double.Parse(tb_deltaE.Text);

            pHetero.percentNoise = Double.Parse(tb_noise.Text) / 100;

            //there must be a file in the each field to read
            if (tb_patternFileC1.Text.Equals("") || tb_patternFileC2.Text.Equals(""))
            {
                MessageBox.Show("One or both pattern file fields are empty", "Warning", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
            }
```

```
        //the FFT algorithm can only transform data with dimension lenghths that are powers of 2
        else if (pHetero.sqLength / 64 != 1 && pHetero.sqLength / 128 != 1 && pHetero.sqLength / 256 != 1 &&
pHetero.sqLength / 512 != 1)
        {
            MessageBox.Show("Size of simulation square must be a power of 2 and in the given range", "Warning",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
        }
        //the image size has to match the simulation square size; otherwise, BitmapConverter will read past the end of the pattern
file
        else if (getImgSize(tb_patternFileC1.Text) < Math.Pow(pHetero.sqLength, 2) || getImgSize(tb_patternFileC2.Text) <
Math.Pow(pHetero.sqLength, 2))
        {
            MessageBox.Show("The size of one or both of the image files is smaller than the input for the simulation square size",
"Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);
        }
        else
        {
            BitmapConverter bmpC = new BitmapConverter(pHetero);
            bmpC.convert(tb_patternFileC1.Text, tb_patternFileC2.Text);
        }
    }

    //--------------------------------------------------------------------
    /**
     * This returns the size of the image file
     */
    private int getImgSize(String imgFilename)
    {
        int picSize = 0;
        StreamReader rd = new StreamReader(imgFilename);
        for (; ; )
        {
            rd.Read();
            picSize++;

            if (rd.EndOfStream == true)
                break;
        }

        rd.Close();
        return (picSize - 54);
    }

    //--------------------------------------------------------------------
    /**
     * This opens the browse dialog box where the user can choose a
     * pattern file
     */
    private void bt_patternBrowse_Click_1(object sender, EventArgs e)
    {
        openPatternFile("C1");
    }

    //--------------------------------------------------------------------
    /**
     * This opens the browse dialog box where the user can choose a
     * pattern file
     */
    private void patternFileToolStripMenuItem_Click(object sender, EventArgs e)
    {
        openPatternFile("C1");
    }
```

```
//---------------------------------------------------------------------
/**
 * This opens the browse dialog box where the user can choose a
 * pattern file
 */
private void bt_patternBrowseC2_Click(object sender, EventArgs e)
{
   openPatternFile("C2");
}

//---------------------------------------------------------------------
/**
 * This opens the browse dialog box where the user can choose a
 * pattern file
 */
private void patternFileC2ToolStripMenuItem_Click(object sender, EventArgs e)
{
   openPatternFile("C2");
}

//---------------------------------------------------------------------
/**
 * This opens the browse dialog box where the user can choose a
 * parameter file
 */
private void bt_parameterBrowse_Click(object sender, EventArgs e)
{
   openParameterFile();
}

//---------------------------------------------------------------------
/**
 * This opens the browse dialog box where the user can choose a
 * parameter file
 */
private void parameterFileToolStripMenuItem_Click(object sender, EventArgs e)
{
   openParameterFile();
}

//---------------------------------------------------------------------
/**
 * This reloads the parameters from the selected file
 */
private void bt_reload_Click(object sender, EventArgs e)
{
   readInParam();
}

//---------------------------------------------------------------------
/**
 * Method for saving parameters
 */
private void saveToolStripMenuItem_Click(object sender, EventArgs e)
{
   SaveFileDialog sfd = new SaveFileDialog();
   sfd.Filter = "Text File|*.txt";
   sfd.Title = "Save Paramters";
   sfd.FileName = "";

   if (sfd.ShowDialog() == DialogResult.OK)
```

```
    {
        System.IO.StreamWriter writer = new System.IO.StreamWriter(sfd.FileName);

        writer.WriteLine(tb_Q1.Text);
        writer.WriteLine(tb_Q2.Text);
        writer.WriteLine(tb_Q3.Text);
        writer.WriteLine(tb_S.Text);
        writer.WriteLine(tb_H.Text);
        writer.WriteLine(tb_bond012.Text);
        writer.WriteLine(tb_bond112.Text);
        writer.WriteLine(tb_bond013.Text);
        writer.WriteLine(tb_bond113.Text);
        writer.WriteLine(tb_bond023.Text);
        writer.WriteLine(tb_bond123.Text);
        writer.WriteLine(tb_initTemp.Text);
        writer.WriteLine(tb_noise.Text);
        writer.WriteLine(tb_delta.Text);
        writer.WriteLine(tb_alpha12.Text);
        writer.WriteLine(tb_alpha13.Text);
        writer.WriteLine(tb_alpha23.Text);
        writer.WriteLine(tb_beta.Text);
        writer.WriteLine(tb_deltaE.Text);

        writer.Close();
    }
}

//-----------------------------------------------------------------
/**
 * This is the method for opening a pattern file for C1 or C2
 */
private void openPatternFile(String C1orC2)
{
    ofd.Filter = "Bitmap Image|*.bmp";
    ofd.Title = "Open an Image File";
    ofd.FileName = "";
    ofd.ShowDialog();

    if (C1orC2.Equals("C1"))
    {
        tb_patternFileC1.Text = ofd.FileName;
    }
    else
    {
        tb_patternFileC2.Text = ofd.FileName;
    }
}

//-----------------------------------------------------------------
/**
 * This is the method for opening a parameter file
 */
private void openParameterFile()
{
    ofd.Filter = "Text File|*.txt";
    ofd.Title = "Open Parameter File";
    ofd.FileName = "";
    ofd.ShowDialog();

    tb_parameterFile.Text = ofd.FileName;

    readInParam();
```

```
        }

        //---------------------------------------------------------------
        /**
         * This reads in the parameters from the selected file
         */
        private void readInParam()
        {
            if (tb_parameterFile.Text.Equals(""))
            {
            }
            else
            {
                StreamReader paramRd = new StreamReader(tb_parameterFile.Text);

                tb_Q1.Text = paramRd.ReadLine();
                tb_Q2.Text = paramRd.ReadLine();
                tb_Q3.Text = paramRd.ReadLine();
                tb_S.Text = paramRd.ReadLine();
                tb_H.Text = paramRd.ReadLine();
                tb_bond012.Text = paramRd.ReadLine();
                tb_bond112.Text = paramRd.ReadLine();
                tb_bond013.Text = paramRd.ReadLine();
                tb_bond113.Text = paramRd.ReadLine();
                tb_bond023.Text = paramRd.ReadLine();
                tb_bond123.Text = paramRd.ReadLine();
                tb_initTemp.Text = paramRd.ReadLine();
                tb_noise.Text = paramRd.ReadLine();
                tb_delta.Text = paramRd.ReadLine();
                tb_alpha12.Text = paramRd.ReadLine();
                tb_alpha13.Text = paramRd.ReadLine();
                tb_alpha23.Text = paramRd.ReadLine();
                tb_beta.Text = paramRd.ReadLine();
                tb_deltaE.Text = paramRd.ReadLine();

                paramRd.Close();
            }
        }
        //---------------------------------------------------------------
    }
}
```

BitmapConverter.cs

```
//=============================================================================
//
// File Name    :  BitmapConverter.cs
// Purpose      :  To interpret the colors from a pattern file as concentration
//              :  values
//
// Author       :  Micheal Wang
//              Albuquerque Academy
// Created on   :  December 2007
// Copyright    :  All Rights Reserved.
//
//=============================================================================

using System;
using System.IO;
using System.Collections.Generic;
using System.Text;

namespace NanoSimulator
```

```
{
   class BitmapConverter
   {
      Parameters pBmpConv;

      /**
       * constructor
       */
      public BitmapConverter(Parameters p)
      {
         pBmpConv = p;
      }

      //-------------------------------------------------------------------
      /**
       * This calls the converting method and then calls the simulation
       * window
       */
      public void convert(String filenameC1, String filenameC2)
      {
         pBmpConv.C1Mat = colorToConcentration(filenameC1);
         pBmpConv.C2Mat = colorToConcentration(filenameC2);

         formSimWindow fsm = new formSimWindow(pBmpConv);
           fsm.heterogeneousArray();
           fsm.Show();
      }

      //-------------------------------------------------------------------
      /**
       * This returns the concentrations corresponding to the colors in
       * the bmp file
       */
      private double[] colorToConcentration(String filename)
      {
         double[] Mat = new double[pBmpConv.cMatSize];
         byte hex1, hex2, hex3;
         int pixelColor;

         FileStream file = new FileStream(filename, FileMode.Open, FileAccess.Read);
         BinaryReader reader = new BinaryReader(file);

         // read past the 54 byte header
         for (int i = 0; i < 54; i++)
         {
            reader.ReadByte();
         }

         for (int i = 0; i < pBmpConv.cMatSize; i += 2)
         {
            hex1 = reader.ReadByte();
            hex2 = reader.ReadByte();
            hex3 = reader.ReadByte();
            pixelColor = (hex1 << 16) + (hex2 << 8) + hex3;
            switch (pixelColor)
            {
               case 0x000000://black
                  Mat[i] = 0.99;
                  break;
               case 0x061310:
                  Mat[i] = 0.95;
                  break;
```

```
        case 0x0E2520:
            Mat[i] = 0.9;
            break;
        case 0x143830:
            Mat[i] = 0.85;
            break;
        case 0x1D4940:
            Mat[i] = 0.8;
            break;
        case 0x245B4F:
            Mat[i] = 0.75;
            break;
        case 0x2B6F60:
            Mat[i] = 0.7;
            break;
        case 0x32816F:
            Mat[i] = 0.65;
            break;
        case 0x37957D:
            Mat[i] = 0.6;
            break;
        case 0x3EA88D:
            Mat[i] = 0.55;
            break;
        case 0x48B798:
            Mat[i] = 0.5;
            break;
        case 0x59BFA8:
            Mat[i] = 0.45;
            break;
        case 0x6CC6B4:
            Mat[i] = 0.4;
            break;
        case 0x7ECDBD:
            Mat[i] = 0.35;
            break;
        case 0x93D2C5:
            Mat[i] = 0.3;
            break;
        case 0xA4DBCE:
            Mat[i] = 0.25;
            break;
        case 0xB6E2D9:
            Mat[i] = 0.2;
            break;
        case 0xC8EAE3:
            Mat[i] = 0.15;
            break;
        case 0xDAF1ED:
            Mat[i] = 0.1;
            break;
        case 0xEDF8F5:
            Mat[i] = 0.05;
            break;
        default:
            Mat[i] = 0.05;
            break;
    }
    Mat[i + 1] = 0;
}

reader.Close();
```

```
            file.Close();

            return Mat;
        }
    }
}
```

formSimWindow.cs

```
//=============================================================================
//
// File Name    :  fromSimWindow.cs
// Purpose      :  To display the new concentration as well as time elapsed, etc.
//
// Author       :  Micheal Wang
//              Albuquerque Academy
// Created on   :  December 2007
// Copyright    :  All Rights Reserved.
//
//=============================================================================

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

namespace NanoSimulator
{
    public partial class formSimWindow : Form
    {
        Parameters pSimWin;
        FFTandCalc calc;
        Random r = new Random();
        bool warningAlreadyDisp = false;
        bool loadNextIsClicked = false;
        double tempVar;
        double timeStep;
        double timeElapsed;
        int numTimeStepsPerDisp;

        /**
         * constructor
         */
        public formSimWindow(Parameters p)
        {
            pSimWin = p;
            InitializeComponent();
        }

        //--------------------------------------------------------------------
        /**
         *
         */
        private void formSimWindow_Load(object sender, EventArgs e)
        {

        }

        //--------------------------------------------------------------------
        /**
```

```
        * A random noise is added to each concentration based on the "percent
        * noise" input.  If the sum of the concentrations is greater or
        * equal to 1, they are scaled by their sum.
        */
       public void heterogeneousArray()
       {
          for (int i = 0; i < pSimWin.cMatSize; i += 2)
          {
             pSimWin.C1Mat[i] = pSimWin.C1Mat[i] + pSimWin.percentNoise * (r.NextDouble() - 0.5);
             pSimWin.C2Mat[i] = pSimWin.C2Mat[i] + pSimWin.percentNoise * (r.NextDouble() - 0.5);

             tempVar = pSimWin.C1Mat[i] + pSimWin.C2Mat[i];
             if (tempVar > 1)
             {
                if (warningAlreadyDisp == false)
                {
                   MessageBox.Show("Note: C1 + C2 must be < 1.  Otherwise, C2[i, j] will scaled i.e. will be divided by (C1Mat[i]
+ C2Mat[i])",
                      "Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);
                   warningAlreadyDisp = true;
                }
                pSimWin.C1Mat[i] = pSimWin.C1Mat[i] / tempVar;
                pSimWin.C2Mat[i] = pSimWin.C2Mat[i] / tempVar;
             }
          }

          tb_timeStep.Text = "0.001";
          tb_temp.Text = pSimWin.iTemperature + "";
          calc_k();
       }

       //---------------------------------------------------------------
       /**
        * The homogeneous arrays are assigned the input values.  The addition
        * of noise and scaling are similar to those in heterogeneousArray().
        */
       public void homogeneousArray()
       {
          pSimWin.C1Mat = new double[pSimWin.cMatSize];
          pSimWin.C2Mat = new double[pSimWin.cMatSize];

          for (int i = 0; i < pSimWin.cMatSize; i += 2)
          {
             pSimWin.C1Mat[i] = pSimWin.homoC1 + pSimWin.percentNoise * (r.NextDouble() - 0.5);
             pSimWin.C2Mat[i] = pSimWin.homoC2 + pSimWin.percentNoise * (r.NextDouble() - 0.5);

             pSimWin.C1Mat[i + 1] = 0.0;
             pSimWin.C2Mat[i + 1] = 0.0;

             tempVar = pSimWin.C1Mat[i] + pSimWin.C2Mat[i];
             if (tempVar > 1)
             {
                if (warningAlreadyDisp == false)
                {
                   MessageBox.Show("Note: C1 + C2 must be < 1.  Otherwise, C2[i, j] will scaled i.e. will be divided by (C1Mat[i]
+ C2Mat[i])",
                      "Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);
                   warningAlreadyDisp = true;
                }

                pSimWin.C1Mat[i] = pSimWin.C1Mat[i] / tempVar;
                pSimWin.C2Mat[i] = pSimWin.C2Mat[i] / tempVar;
```

```
        }
    }

    tb_timeStep.Text = "0.0001";
    tb_temp.Text = pSimWin.iTemperature + "";
    calc_k();
}

//---------------------------------------------------------------------
/**
 * Since the frequencies in Fourier space remain unchanged throughout
 * the entire simulation, they are calculated ahead of time (see
 * Final Report for details on calculating k).
 */
private void calc_k()
{
    double freq1 = 0;
    double freq2 = 0;
    double fourCellSize = 2 * Math.PI / (pSimWin.sqLength * pSimWin.delta);
    int j = 0;

    pSimWin.k = new double[pSimWin.kSize];

    for (int i = 0; i < pSimWin.kSize; i++)
    {
        pSimWin.k[i] = Math.Sqrt(freq1 * freq1 + freq2 * freq2);

        if ((i + 1) % pSimWin.sqLength <= pSimWin.sqLength / 2)
            freq1 += fourCellSize;
        else
            freq1 -= fourCellSize;

        if ((i + 1) % pSimWin.sqLength == 0)
        {
            if ((j + 1) <= pSimWin.sqLength / 2)
                freq2 += fourCellSize;
            else
                freq2 -= fourCellSize;

            freq1 = 0;
            j++;
        }
    }
}

//---------------------------------------------------------------------
/**
 * This closes simulation window
 */
private void bt_close_Click(object sender, EventArgs e)
{
    this.Dispose();
}

//---------------------------------------------------------------------
/**
 * This calls the calculations class when the button is clicked and
 * the displaying method when the calculations a finished
 */
private void bt_next_Click(object sender, EventArgs e)
{
    timeStep = Double.Parse(tb_timeStep.Text);
```

```
    numTimeStepsPerDisp = Int16.Parse(tb_dispEvery.Text);
    timeElapsed += numTimeStepsPerDisp * timeStep;

    //This prevents calling FFTandCalc again and erasing some of the calculated parameters
    if (loadNextIsClicked == false)
    {
        calc = new FFTandCalc(pSimWin);
        loadNextIsClicked = true;
    }

    bt_next.Hide();
    bt_close.Hide();
    tb_dispEvery.Hide();
    tb_timeStep.Hide();
    tb_temp.Hide();

    calc.calc_TempConstants(Double.Parse(tb_temp.Text));
    calc.calcNext(numTimeStepsPerDisp, timeStep);  //contains a loop

    bt_next.Show();
    bt_close.Show();
    tb_dispEvery.Show();
    tb_timeStep.Show();
    tb_temp.Show();

    Graphics g = p_graphics.CreateGraphics();
    Rectangle r = new Rectangle(0, 0, 1165, 559);
    PaintEventArgs e1 = new PaintEventArgs(g, r);
    p_graphics_Paint(sender, e1);
}

//--------------------------------------------------------------------
/**
 * This displays the calcultated concentrations.  In addition, the
 * outline rectangles, component labels, color scheme, time, and
 * tmperature are displayed.
 */
private void p_graphics_Paint(object sender, PaintEventArgs e)
{
    int xyShift = (512 - pSimWin.sqLength) / 2;  //recenters simulation squares when their size is changed
    int locationX, locationY;
    int xy_to_i;
    SolidBrush colorBrush;
    SolidBrush stringBrush = new SolidBrush(Color.Black);
    Pen pen = new Pen(Color.Black, 1);
    Font font;

    //outline rectangles
    int simRectSize = pSimWin.sqLength + 1;
    locationX = 15 + xyShift;
    locationY = 25 + xyShift;

    e.Graphics.DrawRectangle(pen, locationX, locationY, simRectSize, simRectSize);
    e.Graphics.DrawRectangle(pen, locationX + 620, locationY, simRectSize, simRectSize);

    //color scheme
    font = new Font(FontFamily.GenericMonospace, 9);
    for (int i = 0; i < 100; i += 5)
    {
        colorBrush = new SolidBrush(getColor(i / 100.0));
        e.Graphics.FillRectangle(colorBrush, 597, 447 - 4 * (i - 2), 30, 10);
        e.Graphics.DrawString(i / 100.0 + "-" + (i + 5) / 100.0, font, stringBrush, 533, 445 - 4 * (i - 2));
```

```
    }

    //simulation square labels
    font = new Font(FontFamily.GenericMonospace, 12);
    e.Graphics.DrawString("Species 1", font, stringBrush, locationX, locationY - 20);
    e.Graphics.DrawString("Species 2", font, stringBrush, locationX + 620, locationY - 20);

    //total scaled time elapsed and current temperature
    time.Text = timeElapsed + "";
    temperature.Text = tb_temp.Text + " K";

    //display new concentrations
    for (int y = 0; y < pSimWin.sqLength; y++)
    {
        for (int x = 0; x < pSimWin.sqLength; x++)
        {
            xy_to_i = 2 * (pSimWin.sqLength * y + x);
            locationX = xyShift + x + 16;
            locationY = 25 + xyShift + pSimWin.sqLength - y;

            colorBrush = new SolidBrush(getColor(pSimWin.C1Mat[xy_to_i]));
            e.Graphics.FillRectangle(colorBrush, locationX, locationY, 1, 1);

            colorBrush = new SolidBrush(getColor(pSimWin.C2Mat[xy_to_i]));
            e.Graphics.FillRectangle(colorBrush, locationX + 620, locationY, 1, 1);
        }
    }
}

//--------------------------------------------------------------------
/**
 * Returns a color corresponding to a range of concentrations.
 */
private Color getColor(double CValue)
{
    if (CValue < 1 && CValue > 0.95)
        return Color.Black;
    else if (CValue <= 0.95 && CValue > 0.9)
        return Color.FromArgb(16, 19, 6);
    else if (CValue <= 0.9 && CValue > 0.85)
        return Color.FromArgb(32, 37, 14);
    else if (CValue <= 0.85 && CValue > 0.8)
        return Color.FromArgb(48, 56, 20);
    else if (CValue <= 0.8 && CValue > 0.75)
        return Color.FromArgb(64, 73, 29);
    else if (CValue <= 0.75 && CValue > 0.7)
        return Color.FromArgb(79, 91, 36);
    else if (CValue <= 0.7 && CValue > 0.65)
        return Color.FromArgb(96, 111, 43);
    else if (CValue <= 0.65 && CValue > 0.6)
        return Color.FromArgb(111, 129, 50);
    else if (CValue <= 0.6 && CValue > 0.55)
        return Color.FromArgb(125, 149, 55);
    else if (CValue <= 0.55 && CValue > 0.5)
        return Color.FromArgb(141, 168, 62);
    else if (CValue <= 0.5 && CValue > 0.45)
        return Color.FromArgb(152, 183, 72);
    else if (CValue <= 0.45 && CValue > 0.4)
        return Color.FromArgb(168, 191, 89);
    else if (CValue <= 0.4 && CValue > 0.35)
        return Color.FromArgb(180, 198, 108);
    else if (CValue <= 0.35 && CValue > 0.3)
```

```
            return Color.FromArgb(189, 205, 126);
         else if (CValue <= 0.3 && CValue > 0.25)
            return Color.FromArgb(197, 210, 147);
         else if (CValue <= 0.25 && CValue > 0.2)
            return Color.FromArgb(206, 219, 164);
         else if (CValue <= 0.2 && CValue > 0.15)
            return Color.FromArgb(217, 226, 182);
         else if (CValue <= 0.15 && CValue > 0.1)
            return Color.FromArgb(227, 234, 200);
         else if (CValue <= 0.1 && CValue > 0.05)
            return Color.FromArgb(237, 241, 218);
         else if (CValue <= 0.05 && CValue > 1E-6)
            return Color.FromArgb(245, 248, 237);
         else
            return Color.White;
      }
      //-----------------------------------------------------------------
   }
}
```

FFTandCalc

```
//=============================================================================
//
// File Name    :  FFTandCalc.cs
// Purpose      :  To perform all of the calculations
//
// Author       :  Micheal Wang
//                 Albuquerque Academy
// Created on   :  December 2007
// Copyright    :  All Rights Reserved.
//
//=============================================================================

using System;
using System.Collections.Generic;
using System.Text;

namespace NanoSimulator
{
   class FFTandCalc
   {
      Parameters pfc;

      /**
       * constructor
       */
      public FFTandCalc(Parameters p)
      {
         pfc = p;
      }

      double tempExpTerm, tempTermA12, tempTermA13, tempTermA23, tempTermBeta, tempRatio;

      //-----------------------------------------------------------------
      /**
       * This calculates the temperature constants used in calculations
       */
      public void calc_TempConstants(double newTemp1)
      {
         double R = 8.314472;    //ideal gas constant J/(mol K)
         double iTemp = pfc.iTemperature;
```

```
   tempExpTerm = Math.Pow(Math.E, 1000 * pfc.activationEnergy / R * (1 / iTemp - 1 / newTemp1));
   tempTermA12 = 1 + pfc.a_12 * (iTemp - newTemp1);
   tempTermA13 = 1 + pfc.a_13 * (iTemp - newTemp1);
   tempTermA23 = 1 + pfc.a_23 * (iTemp - newTemp1);
   tempTermBeta = 1 + pfc.beta * (iTemp - newTemp1);
   tempRatio = newTemp1 / iTemp;
}

//---------------------------------------------------------------------
/**
 * This calculates the next timestep.
 *
 * Because of the natural log in the P1 and P2 functions, C1 and C2
 * cannot equal zero and their sum cannot exceed one.  These
 * restrictions are checked for.
 *
 * The next timestep is calculated.
 */
public void calcNext(int numLoops, double timeStep)
{
   int[] ithDimLength = { pfc.sqLength, pfc.sqLength };
   double[] P1 = new double[pfc.cMatSize];
   double[] P2 = new double[pfc.cMatSize];
   double C1, C2, C11, C22, tempVar, tempVar1;

   for (int numTimeSteps = 0; numTimeSteps < numLoops; numTimeSteps++)
   {
      for (int i = 0; i < pfc.cMatSize; i += 2)
      {
         C11 = (C1 = pfc.C1Mat[i]);
         C22 = (C2 = pfc.C2Mat[i]);

         tempVar = 1 - C1 - C2;

         if (C1 <= 0)
         {
            C11 = 1E-20;
            C1 = 0;
            pfc.C1Mat[i] = 0;
         }

         if (C2 <= 0)
         {
            C22 = 1E-20;
            C2 = 0;
            pfc.C2Mat[i] = 0;
         }

         tempVar1 = C1 + C2;
         if (tempVar1 >= 1)
         {
            C1 = (pfc.C1Mat[i] = pfc.C1Mat[i] / tempVar1);
            C2 = (pfc.C2Mat[i] = pfc.C2Mat[i] / tempVar1);
            tempVar = 1E-20;
         }

         // calculate P1 and P2, "Simulation on nanoscale self-assembly of ternary-epilayers" page 25
         P1[i] = Math.Log(C11 / tempVar, Math.E) * tempRatio;
         P1[i] += C2 * (pfc.o_0_12 + pfc.o_1_12 * ((2 * C1) - C2)) * tempTermA12;
         P1[i] -= C2 * (pfc.o_0_23 + pfc.o_1_23 * ((2 * C1) + (3 * C2) - 2)) * tempTermA23;
         P1[i] += (pfc.o_0_13 * (1 - (2 * C1) - C2) + pfc.o_1_13 *
            ((6 * C1) + (2 * C2) - (6 * C1 * C1) - (C2 * C2) - (6 * C1 * C2) - 1)) * tempTermA13;
```

```
            P2[i] = Math.Log(C22 / tempVar, Math.E) * tempRatio;
            P2[i] += C1 * (pfc.o_0_12 + pfc.o_1_12 * (C1 - (2 * C2))) * tempTermA12;
            P2[i] -= C1 * (pfc.o_0_13 + pfc.o_1_13 * ((3 * C1) + (2 * C2) - 2)) * tempTermA13;
            P2[i] += (pfc.o_0_23 * (1 - C1 - (2 * C2)) + pfc.o_1_23 *
                ((2 * C1) + (6 * C2) - (C1 * C1) - (6 * C2 * C2) - (6 * C1 * C2) - 1)) * tempTermA23;

            P1[i + 1] = 0;
            P2[i + 1] = 0;
        }

        nDimFFT(P1, P2, ithDimLength, 1);//transform P1 and P2 from real to Fourier space
        nDimFFT(pfc.C1Mat, pfc.C2Mat, ithDimLength, 1);//transform C1 and C2 from real to Fourier space

        // update concentrations, "Simulation on nanoscale self-assembly of ternary-epilayers" page 25
        double a, b, c, d, determinant;
        double termC1P1r, termC1P1i, termC2P2r, termC2P2i, term1, term2;
        int ii;
        for (int i = 0; i < pfc.cMatSize; i += 2)
        {
            ii = i >> 1;    // i/2
            term1 = pfc.k[ii] * pfc.k[ii] * timeStep * tempExpTerm;
            term2 = 2 * term1 * pfc.k[ii];

            a = 1 + term2 * (pfc.k[ii] - pfc.Q1 / tempTermBeta);
            b = -pfc.Q2 / tempTermBeta * term2;
            c = pfc.S * b;
            d = 1 + term2 * pfc.S * (pfc.H * pfc.k[ii] - pfc.Q3 / tempTermBeta);
            determinant = 1 / (a * d - b * c);      //used to find inverse of the matrix

            termC1P1r = pfc.C1Mat[i] - term1 * P1[i];  //real part
            termC1P1i = pfc.C1Mat[i + 1] - term1 * P1[i + 1];  //imaginary part
            termC2P2r = pfc.C2Mat[i] - term1 * pfc.S * P2[i];  //real part
            termC2P2i = pfc.C2Mat[i + 1] - term1 * pfc.S * P2[i + 1];  //imaginary part

            pfc.C1Mat[i] = determinant * (d * termC1P1r - b * termC2P2r);
            pfc.C1Mat[i + 1] = determinant * (d * termC1P1i - b * termC2P2i);
            pfc.C2Mat[i] = determinant * (-c * termC1P1r + a * termC2P2r);
            pfc.C2Mat[i + 1] = determinant * (-c * termC1P1i + a * termC2P2i);
        }

        nDimFFT(pfc.C1Mat, pfc.C2Mat, ithDimLength, -1);  //transform C1 and C2 from Fourier to real space
        for (int i = 0; i < pfc.cMatSize; i += 2)
        {
            //The FFT algorithm returns the real data times the product of the lengths of each dimension
            pfc.C1Mat[i] = pfc.C1Mat[i] / pfc.kSize;
            pfc.C2Mat[i] = pfc.C2Mat[i] / pfc.kSize;

            pfc.C1Mat[i + 1] = 0;   //reduce numerical error.  Imaginary part supposed to be zero anyways.
            pfc.C2Mat[i + 1] = 0;   //reduce numerical error

            tempVar1 = pfc.C1Mat[i] + pfc.C2Mat[i];
            if (tempVar1 > 1)
            {
                pfc.C1Mat[i] = pfc.C1Mat[i] / tempVar1;
                pfc.C2Mat[i] = pfc.C2Mat[i] / tempVar1;
            }
        }
    }
}

//-----------------------------------------------------------------
```

```
/**
 * FFT algorithm.  Slightly altered to transform two sets of data at
 * once.
 *
 * Adopted from "Numerical Recipes in C++", pages 528-529.
 */
private void nDimFFT(double[] data1, double[] data2, int[] nn, int isign)
{
   int idim, i1, i2, i3, i2rev, i3rev, ip1, ip2, ip3, ifp1, ifp2;
   int ibit, k1, k2, n, nprev, nrem, ntot;
   double theta, tempi, tempr, wi, wpi, wr, wpr, wtemp, tempSwap;

   int ndim = nn.Length;
   ntot = data1.Length >> 1;
   nprev = 1;
   //bit reverse section
   for (idim = ndim - 1; idim >= 0; idim--)
   {
      n = nn[idim];
      nrem = ntot / (n * nprev);
      ip1 = nprev << 1;
      ip2 = ip1 * n;
      ip3 = ip2 * nrem;
      i2rev = 0;
      for (i2 = 0; i2 < ip2; i2 += ip1)
      {
         if (i2 < i2rev)
         {
            for (i1 = i2; i1 < i2 + ip1 - 1; i1 += 2)
            {
               for (i3 = i1; i3 < ip3; i3 += ip2)
               {
                  i3rev = i2rev + i3 - i2;

                  //swap data1[i3] and data1[i3rev]
                  tempSwap = data1[i3];
                  data1[i3] = data1[i3rev];
                  data1[i3rev] = tempSwap;

                  //swap data1[i3 + 1] and data1[i3rev + 1]
                  tempSwap = data1[i3 + 1];
                  data1[i3 + 1] = data1[i3rev + 1];
                  data1[i3rev + 1] = tempSwap;

                  //swap data2[i3] and data2[i3rev]
                  tempSwap = data2[i3];
                  data2[i3] = data2[i3rev];
                  data2[i3rev] = tempSwap;

                  //swap data2[i3 + 1] and data2[i3rev + 1]
                  tempSwap = data2[i3 + 1];
                  data2[i3 + 1] = data2[i3rev + 1];
                  data2[i3rev + 1] = tempSwap;
               }
            }
         }
         ibit = ip2 >> 1;
         while ((ibit >= ip1) && (i2rev + 1 > ibit))
         {
            i2rev -= ibit;
            ibit >>= 1;
         }
```

```
                i2rev += ibit;
            }
            ifp1 = ip1;
            //FFT calculations section
            while (ifp1 < ip2)
            {
                ifp2 = ifp1 << 1;
                theta = isign * 6.28318530717959 / (ifp2 / ip1);
                wtemp = Math.Sin(0.5 * theta);
                wpr = -2.0 * wtemp * wtemp;
                wpi = Math.Sin(theta);
                wr = 1.0;
                wi = 0.0;
                //Another name for the following method if the "Butterfly method"
                for (i3 = 0; i3 < ifp1; i3 += ip1)
                {
                    for (i1 = i3; i1 < i3 + ip1 - 1; i1 += 2)
                    {
                        for (i2 = i1; i2 < ip3; i2 += ifp2)
                        {
                            k1 = i2;
                            k2 = k1 + ifp1;

                            //Danielson-Lanczos Formula for data 1
                            tempr = wr * data1[k2] - wi * data1[k2 + 1];
                            tempi = wr * data1[k2 + 1] + wi * data1[k2];
                            data1[k2] = data1[k1] - tempr;
                            data1[k2 + 1] = data1[k1 + 1] - tempi;
                            data1[k1] += tempr;
                            data1[k1 + 1] += tempi;

                            //Danielson-Lanczos Formula for data 2
                            tempr = wr * data2[k2] - wi * data2[k2 + 1];
                            tempi = wr * data2[k2 + 1] + wi * data2[k2];
                            data2[k2] = data2[k1] - tempr;
                            data2[k2 + 1] = data2[k1 + 1] - tempi;
                            data2[k1] += tempr;
                            data2[k1 + 1] += tempi;
                        }
                    }
                    wr = (wtemp = wr) * wpr - wi * wpi + wr;
                    wi = wi * wpr + wtemp * wpi + wi;
                }
                ifp1 = ifp2;
            }
            nprev *= n;
        }
    }
  }
}
```

Parameters.cs

```
//=============================================================================
//
// File Name    :  Parameters.cs
// Purpose      :  To store all of the parameter that get tranferred around a lot
//
// Author       :  Micheal Wang
//              Albuquerque Academy
// Created on   :  December 2007
// Copyright    :  All Rights Reserved.
//
```

```
//============================================================================

using System;
using System.Collections.Generic;
using System.Text;

namespace NanoSimulator
{
   public struct Parameters
   {
      public double[] C1Mat;         //concentration 1 array
      public double[] C2Mat;         //concentration 2 array
      public double[] k;             //magnitude of frequency vector

      public double homoC1;          //homogeneous concentration 1 value
      public double homoC2;          //homogeneous concentration 2 value

      public int sqLength;           //length of a side of the simulation sq
      public double delta;           //nanometers per pixel
      public double iTemperature;    //initial temperature

      public int cMatSize;           //size of the concentrations arrays
      public int kSize;              //size magnitude of frequency vector

      public double Q1;              //constant
      public double Q2;              //constant
      public double Q3;              //constant

      public double S;               //ratio of moltilities (M2/M1)
      public double H;               //ratio of energies per species (h2/h1)

      //o is omega
      public double o_0_12;          //bonding strength 0 from species 1 to 2
      public double o_1_12;          //bonding strength 1 from species 1 to 2
      public double o_0_23;          //bonding strength 0 from species 2 to 3
      public double o_1_23;          //bonding strength 1 from species 2 to 3
      public double o_0_13;          //bonding strength 0 from species 1 to 3
      public double o_1_13;          //bonding strength 1 from species 1 to 3
      //a is alpha
      public double a_12;            //temperature constant
      public double a_13;            //temperature constant
      public double a_23;            //temperature constant
      public double beta;            //temperature constant

      public double activationEnergy; //Energy required for a reaction to start

      public double percentNoise;    //percent of disturbance on the surface
   }
}
```