Zellomat3D

Style Guide

Projekt:	3D Cellular Automata Simulator – Diplomarbeit – SS/2005	
Auftraggeber:	Hochschule Rapperswil HSR	
Betreuer:	Eduard Glatz – Prof. Dipl. Ing. ETH	eglatz@hsr.ch
Mitarbeiter:	Michael Florin	loop@loop.li
	Andreas Weinmann	a.weinmann@gmx.ch
Ablage:	StyleGuide - 16032005.doc	

Inhaltsverzeichnis

1. Einführung

Zweck

Dieses Dokument beschreibt den Code Style Guide der Projektarbeit. Dieser Code Style Guide bestimmt die grundlegenden Programmierregeln und das Aussehen des Source Codes der zu entwickelnden Applikation.

Gültigkeitsbereich

Dieses Dokument gilt für die Diplomarbeit "Zellomat3D", welche im SS/2005 an der Hochschule Rapperswil HSR durchgeführt wurde.

2. Style Guide

Grundlagen

Die Code Style Guide basiert weitgehend auf den im Software Engineering Unterricht der HSR gelernten Regeln. Die wichtigsten Punkte sind nachfolgend nochmals beschrieben.

Visual Studio

Im Allgemeinen werden die gebräuchlichen und voreingestellten Formatierungsfunktionen von Visual Studio .NET verwendet.

Sprache

Die Variabeln, Methoden und Klassen werden in englischer Sprache benannt. Sie sollen möglichst eindeutig und verständlich sein.

Namen / Abkürzungen

Bei den Benennungen der Variablen ist vor allem wichtig, dass ein ausdrucksvoller Name gewählt wird. Abkürzungen sollten nicht verwendet werden. Eine Ausnahme bilden etwa max oder min.

Sichtbarkeit

Die Instanz- oder Klassenvariabeln sollten per default „private“ deklariert werden. Am besten verwendet man „Properities“.

Klassennamen / Methodennamen / Namespaces

Klassennamen, Methodennamen und Namespaces beginnen immer mit einem grossen Buchstaben und fahren mit Kleinbuchstaben fort. Bei zusammengesetzten Wörtern, wird jedes neue Wort mit einem Grossbuchstaben begonnen, z.B. „SkyGuide“.

Namen

Namen von Attributen und Objekten beginnen immer mit einem Kleinbuchstaben. Bei zusammengesetzten Wörtern wird jedes neue Wort mit einem Grossbuchstaben begonnen.

Kommentare

Grundsätzlich sind die Kommentare von C# zu verwenden, aus welchen später die API-Referenz gebildet werden kann. Sie sind ebenfalls in englischer Sprache zu schreiben. Es sollen nicht nur die Methoden kommentiert werden, sondern auch einzelne Codezeilen, sofern dies zum Verständnis beiträgt.

Anonyme Konstanten

Im Programmcode dürfen keine anonymen Konstanten vorkommen.

Bsp. Schlecht:
for (i = 1; i < 27; i++) // Bedeutung von 27 nicht bekannt

Bsp. Gut:
const int MAX_OPEN_FILES = 27;
for (i = 1; i < MAX_OPEN_FILES; i++)

Ausnahmen sind: Konstanten 0 oder 1 zur Initialisierung von Attributen Konstante 2 zur Halbierung oder Ähnliches.
Bsp. mitte = (obereGrenze + untereGrenze) / 2;

Auf jeden Fall muss eine klar feststellbare Bedeutung der Konstante vorhanden sein. (Für jeden, nicht nur für den Programmierer!)

Klammern

Die Klammern werden auf einer neuen separaten Zeile geöffnet und auch auf einer neuen separaten Zeile wieder geschlossen.

Beispiel:

```
if(IsTrue())
{
        Console.Writeline(„true“);
}
else
{
        Console.Writeline(„false“);
}
```

Abstände / Leerzeichen

Das wichtigste Ziel dieses Grundsatzes ist es, übersicht in den Code zu bringen. (Also genügend Abstände machen, damit der Code lesbarer wird.)

- Zwischen Hauptblöcken (using, Konstanten-Deklarationen, Klassen, Methoden) sollten 2-3 Leerzeilen eingefügt werden.
- Zwischen Code-Segmenten innerhalb einer Methode sollte 1 Leerzeile eingefügt werden.
- Ca 15% des Sourcecodes sind Leerzeilen.

3. Beispiel

Schlecht: **Beispiel**

```
for (int i = 1; i < num; ++i) {
	meetsCriteria[i] = true;
}

for (int i = 2; i < num / 2; ++i) {
	int j = i + i;
	while (j <= num) {
		meetsCriteria[j] = false;
		j += i;
	}
}

for (int i = 0; i < num; ++i) {
	if (meetsCriteria[i]) {
		Console.WriteLine(i + " meets criteria");
	}
}
```

Besser:

```
for (int primeCandidate = 1; primeCandidate < num; ++primeCandidate)
{
	isPrime[primeCandidate] = true;
}

for (int factor = 2; factor < num / 2; ++factor)
{
	int factorableNumber = factor + factor;
	while (factorableNumber <= num)
	{
		isPrime[factorableNumber] = false;
		factorableNumber += factor;
	}
}

for (int primeCandidate = 0; primeCandidate < num; ++primeCandidate)
{
	if (isPrime[primeCandidate])
	{
		Console.WriteLine(primeCandidate + " is prime.");
	}
}
```

4. Code Beautifier

Polystyle

Um ein einheitliches Erscheinungsbild des Codes auch bei verschiedenen Programmierern zu erreichen, verwenden wir einen Code-Beautifier namens Polystyle. Dieses Programm formatiert den Code nach frei vordefinierbaren Regeln, die wir unseren Bedürfnissen angepasst haben Diese Regeln werden in Form eines Beispielcodes angegeben. Dieser Beispielcode dient als Formatvorlage.

Beispielcode von Polystyle für C#

```
sealed public internal class Example : Sclass
{
    public ~Example (int a, out int b, int [,,] c) : base(0)
    {
        // if statements with empty braces
        if(x)
        {
        }
        else if(x)
        {
        }
        else
        {
        }

        // if statements with exactly one braced statement
        if(x)
        {
            a();
        }
        else if(x)
        {
            b();
        }
        else
        {
            c();
        }

        // special 'if' cases
        if(x)
        {
            a();
        }
        else
        {
            c();
        }

        if(x)
        {
            a();

            a();
        }
        else
        {
            c();

            c();
        }
```

```
// if statements with a single implicit substatement
if(x)
    a();

else if(x)
    b();

else
    c();

// if statements with multiple statements
if(!z)
{
    j=++i & (i());

    z=j[0][0][0];
}
else if(a is Class)
{
    a();

    return method(new String[] { a, b, c }, ref b, 'c');
}
else
{
    a();
}

// while statements with a single implicit substatement
while(x && y)
    new a();

// while statements with a single implicit 'if' substatement
while(x)
    if(x)
        a();

// while with multiple statements
while(k==&j)
{
    label:

    h[0]=@a ? * **b : ((int [])c).d->e;

    break;
}

// for statements with a single braced statement
for(int [,] i = { 1, 2, 3 }; i>6; i++)
{
    break label;
}

foreach(int i in S)
{
    i>>1;
}

// do statements with a single braced substatement
do
{
    continue label;
} while(false);

// local blocks
{
    a();
}
```

```
        // various blocked code
        using (int ***x)
        {
            unsafe
            {
                a();
            }
        }

        fixed (int a)
            a();

        /* Switch blocks:
         *
         * You can have case substatements be aligned by giving an
example like:
         *
         * case 1: a();
         *
         *         b();
         *
         *         c();
         *
         *         etc...
         */
        switch(c)
        {
            case 1:
            case 2:
            case 3:
                a();

                b();

                c();

            case 4:
                goto default; // case with exactly one substatement

            default:
                break;
        }
    }

    public void method ()
    {
        // try-catch-finally with empty bodies
        try
        {
        }
        catch (Throwable e)
        {
        }
        finally
        {
        }

        // try-catch-finally with exactly one statement
        try
        {
            a();
        }
        catch (Throwable t)
        {
            a();
        }
        finally
        {
            a();
        }
```

```
        // try-catch-finally with multiple statements
        try
        {
            for(;;)
            {
            } // for statement with empty control statements

            a();
        }
        catch (Throwable e)
        {
            do
            {
            } while(x);

            throw e;
        }
        finally
        {
            int i = i*3, j, k;

            return i+"0";
        }
    }
}

private class Class2
{
    // array with multiple elements
    protected boolean bools1 [][][] =
    {
        false,
        false,
        false
    };

    // array with a multiple element
    boolean bools2 [] = {false, false, false};

    // array with a single element
    boolean bools2 [] = {false};

    // array with no elements
    boolean bools3 [] = { };

    // multidimensional array
    private const int [][][] array = new int[1][][][]
    {
        { 1, 2, 3 },
        { 1, 2, 3 },
        { 1, 2, 3 },
    };

    abstract Object method1 ();

    public event int Event
    {
    [a]
        add
        {
        }
    }
    }

namespace N
{
public struct S
{
static int operator + (int i, int y);
};
}
```

```
delegate int myDelegate ();

[a]
[b]
[field : attrib(i, x=0)]
public enum Enum : int
    {
    a = 0,

    [x]
    b = 1,
    c = 2,
    };

protected interface Intf : Super, Super2
{
    public void emptyMethod ()
    {
    }

    [a]
    int prop
    {
        get;
        set;
    }

    // property with exactly one accessor
    int this [int i]
    {
       get{};
    }
}
```