Waterwolrd Analysis

Ing. Thomas Herzog

<S1310307011@students.fh-hagenberg.at>

Revision History

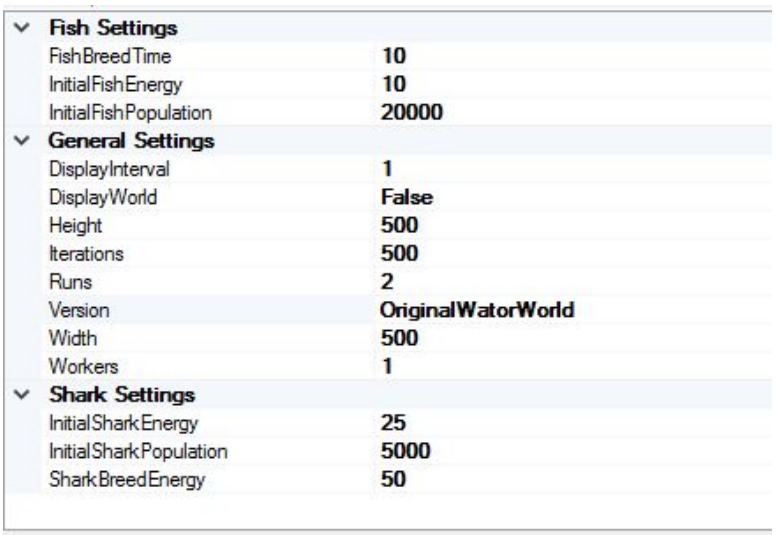
Revision 1.0	March 31 2016	ITH

Folgendes Dokument beschäfftigst sich mit der Analyse und verbessurng des C# Programms *Waterworld*.

Im ersten Kapitel Laufzeitanalyse wird die Laufzeit mit mehreren Durchläufen mit einer bestimmten Konfiguration betrachtet.

1. Laufzeit-Analyse

Folgendes Bild zeigt die festegestzte Konfiguration für die Testdurchläufe.



Fish Settings	
FishBreedTime	**10**
InitialFishEnergy	**10**
InitialFishPopulation	**20000**
General Settings	
DisplayInterval	**1**
DisplayWorld	**False**
Height	**500**
Iterations	**500**
Runs	**2**
Version	**OriginalWatorWorld**
Width	**500**
Workers	**1**
Shark Settings	
InitialSharkEnergy	**25**
InitialSharkPopulation	**5000**
SharkBreedEnergy	**50**

Mit dieser Konfiguration wurden **10** Durchläufe vorgenommen, deren ERgebnisse folgend tabelarisch aufgelistet sind.

```
Runs:                       10
Iterations:                 100
Runtime in Milliseconds:    53626.3151
Avg. Milliseconds / Run:    5362.63151
Std. Deviation:             63.4535428582706
------------------------
Runtimes in Milliseconds:
Run 01:                     5413.8321
Run 02:                     5216.142
Run 03:                     5315.6675
Run 04:                     5309.4686
Run 05:                     5398.8009
Run 06:                     5407.1325
Run 07:                     5407.93
Run 08:                     5339.8962
Run 09:                     5385.8673
Run 10:                     5431.578
```

Figure 1. Ergebnisse der Testdurchläufe

2. *Heap*-Analyse

Dieser Abschnitt beschäftigt sich mit der *Heap*-Analyse, die aufzeigen wird, wie der Heap sich zur Programmlaufzeit verhält und welche Objekte am *Heap* in welcher Verteilung vorzufinden sind.

2.1. Originalversion

Folgender Teil zeigt die Analyseergebnisse der Originalversion.

Functions Allocating Most Memory

Name	Bytes %
VSS.Wator.Original.OriginalWatorWorld.GetNeighbors(valuetype VSS.Wator.Original.MatrixItemType,valuetype System.Drawing.Point)	90.83
VSS.Wator.Original.MatrixItem.ExecuteFishSpawn()	3.56
VSS.Wator.Original.OriginalWatorWorld..ctor(class VSS.Wator.Settings)	3.17
System.Windows.Forms.Form.ShowDialog(class System.Windows.Forms.IWin32Window)	0.84
VSS.Wator.Original.OriginalWatorWorld.GenerateRandomMatrix(int32,int32)	0.66

Types With Most Memory Allocated

Name	Bytes %
System.Drawing.Point[]	90.83
VSS.Wator.Original.MatrixItem	4.84
VSS.Wator.Original.MatrixItem[]	1.32
System.Byte[]	0.73
System.Int32[]	0.67

Types With Most Instances

Name	Instances %
System.Drawing.Point[]	94.68
VSS.Wator.Original.MatrixItem	3.61
System.Version	0.43
System.Drawing.KnownColor	0.21
System.String	0.09

Eklatant fällt hier auf, dass es sher viele **Point** Instanzen am Heap gibt, die sehr kurzlebig sind und daher den *Grabage Collector* stark belasten.

2.2. Erste Optimierungen

Folgender Teil zeigt die Analyseergebnisse nach dem ersten Optimierungsversuch.

3. Programmanalyse

hello

Umstrukturierung Teil 1||Std. Deviation|Run 1|Run 2 hello

4. Umstrukturierung Teil 2

hello

5. Umstrukturierung Teil 3

hello