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*.

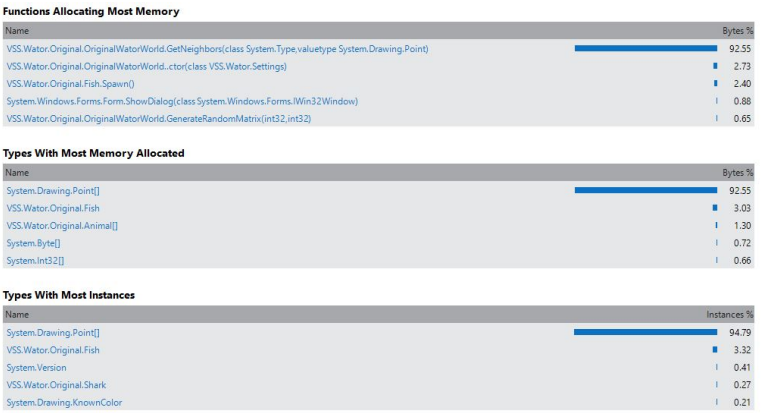
Im ersten Kapitel Laufzeitanalyse wird die Laufzeit mit mehreren Durchläufen mit einer bestimmten Konfiguration betrachtet.

1. *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.

1.1. Originalversion

Folgender Teil zeigt die Analyseergebnisse der Originalversion.



Functions Allocating Most Memory

Name	Bytes %
VSS.Wator.Original.OriginalWatorWorld.GetNeighbors(class System.Type,valuetype System.Drawing.Point)	92.55
VSS.Wator.Original.OriginalWatorWorld..ctor(class VSS.Wator.Settings)	2.73
VSS.Wator.Original.Fish.Spawn()	2.40
System.Windows.Forms.Form.ShowDialog(class System.Windows.Forms.IWin32Window)	0.88
VSS.Wator.Original.OriginalWatorWorld.GenerateRandomMatrix(int32,int32)	0.65

Types With Most Memory Allocated

Name	Bytes %
System.Drawing.Point[]	92.55
VSS.Wator.Original.Fish	3.03
VSS.Wator.Original.Animal[]	1.30
System.Byte[]	0.72
System.Int32[]	0.66

Types With Most Instances

Name	Instances %
System.Drawing.Point[]	94.79
VSS.Wator.Original.Fish	3.32
System.Version	0.41
VSS.Wator.Original.Shark	0.27
System.Drawing.KnownColor	0.21

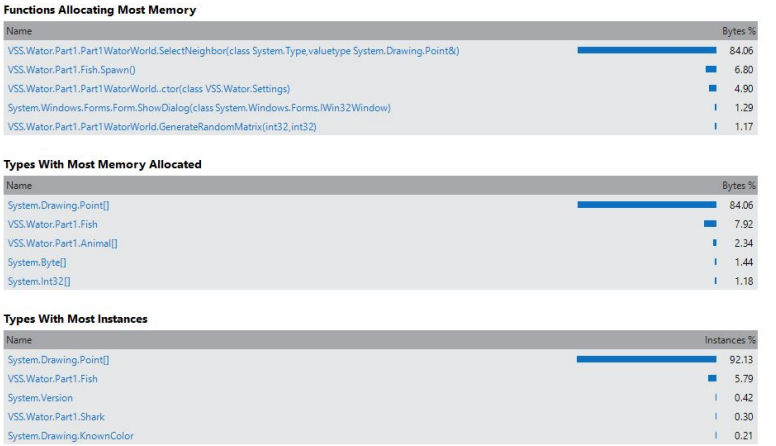
Figure 1. Ergebnisse der *Heap*-Analyse der Originalversion

Eklatant fällt hier auf, dass es sher viele `Point` Instanzen am Heap gibt, die sehr kurzlebig sind und daher den *Grabage Collector* stark belasten. Daher sollte hier angesetzt werden, um zu versuchen diese `Point` Instanzen zu vermeiden.

Des Weiteren sieht man in Abbildung Figure 1, “Ergebnisse der *Heap*-Analyse der Originalversion”, dass die Methode `GetNeighbors(Type, Point)` die meisten `Point` Instanzen produziert, daher sollte man hier zuerst ansetzen. In der Methode `GetNeighbors(Type, Point)` werden alle NAchbarn gesucht und die `Animal` Instanz entscheidet durch Zufall welcher Nachbar genommen wird.

1.2. Ergebnisse von: *Point Instanzen vermeiden*

Folgende Teil zeigt die Analyseergebnisse der *Heap*-Analyse der ersten Verbesserungen.



Functions Allocating Most Memory

Name	Bytes %
VSS.Wator.Part1.Part1WatorWorld.SelectNeighbor(class System.Type,valuetype System.Drawing.Point&)	84.06
VSS.Wator.Part1.Fish.Spawn()	6.80
VSS.Wator.Part1.Part1WatorWorld..ctor(class VSS.Wator.Settings)	4.90
System.Windows.Forms.Form.ShowDialog(class System.Windows.Forms.IWin32Window)	1.29
VSS.Wator.Part1.Part1WatorWorld.GenerateRandomMatrix(int32,int32)	1.17

Types With Most Memory Allocated

Name	Bytes %
System.Drawing.Point[]	84.06
VSS.Wator.Part1.Fish	7.92
VSS.Wator.Part1.Animal[]	2.34
System.Byte[]	1.44
System.Int32[]	1.18

Types With Most Instances

Name	Instances %
System.Drawing.Point[]	92.13
VSS.Wator.Part1.Fish	5.79
System.Version	0.42
VSS.Wator.Part1.Shark	0.30
System.Drawing.KnownColor	0.21

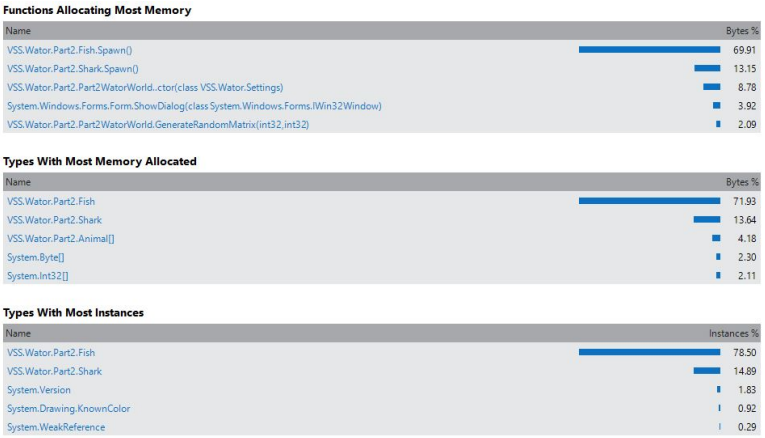
Figure 2. Ergebnisse von: Point Instanzen vermeiden

Diese Verbesserung hat erreicht das anstatt **~ 94 %** nur mehr **~ 84 %** der Objekte am *Heap* *Point*-Instanzen sind. Dadurch ist der *Garbage Collector* weniger stark ausgelastet, was einen positiven Effekt auf das Laufzeitverhalten hat.

Trotzdem sei angemerkt, dass sich herausgestellt hat, dass der Ansatz des *Random*-Zugriff auf die *Direction* beim Ermitteln der Nachbarn die Laufzeit um **~ 500ms** verschlechtert, daher wurde der Randomzugriff entfernt. Dadurch wird aber immer in die gleiche Richtung gegangen bzw nach Nachbanr gesucht, was zur Folge hat, dass dies auf der grafischen Asugabe sichtbar wird.

1.3. Ergebnisse von: *SelectNeighbor entfernen*

Folgender Teil zeigt die Analyseergebnisse der *Heap*-Analyse der Verbesserung *SelectNeighbor entfernen*.



Functions Allocating Most Memory

Name	Bytes %
VSS.Wator.Part2.Fish.Spawn()	69.91
VSS.Wator.Part2.Shark.Spawn()	13.15
VSS.Wator.Part2.Part2WatorWorld..ctor(class VSS.Wator.Settings)	8.78
System.Windows.Forms.Form.ShowDialog(class System.Windows.Forms.IWin32Window)	3.92
VSS.Wator.Part2.Part2WatorWorld.GenerateRandomMatrix(int32,int32)	2.09

Types With Most Memory Allocated

Name	Bytes %
VSS.Wator.Part2.Fish	71.93
VSS.Wator.Part2.Shark	13.64
VSS.Wator.Part2.Animal[]	4.18
System.Byte[]	2.30
System.Int32[]	2.11

Types With Most Instances

Name	Instances %
VSS.Wator.Part2.Fish	78.50
VSS.Wator.Part2.Shark	14.89
System.Version	1.83
System.Drawing.KnownColor	0.92
System.WeakReference	0.29

Figure 3. Ergebnisse von: SelectNeighbor entfernen

Durch das Entfernen der Methode `SelectNeighbor(Type, Point)` sind die *Point*-Objekte nicht mehr die dominanten Objekte am *Heap* sondern *Fish*-Objekte, die aber nicht zu vermeiden sind, da es nur soviele Fische am *Heap* gibt als wie in der Matrix. Und die *Fish*-Objekte sind langläbigere Objelkte als die *Point*-Objekte.

2. 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.

2.1. Ergebnisse von: *Originalversion*

Folgende Teil zeigt die Analyseergebnisse der *Runtime*-Analyse der Originalversion.

```
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 4. Ergebnisse vom: Originalversion

2.2. Ergebnisse von: *Point Instanzen vermeiden*

Folgende Teil zeigt die Analyseergebnisse der *Runtime*-Analyse dieser angewandten Verbesserung.

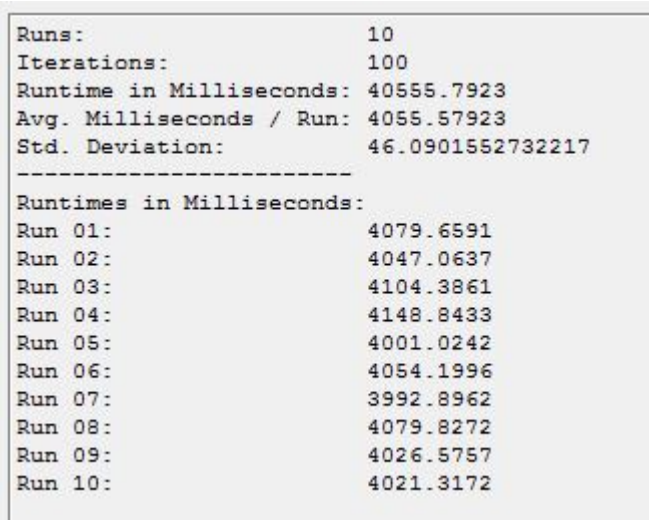
```
Runs:                       10
Iterations:                 100
Runtime in Milliseconds:    46086.6748
Avg. Milliseconds / Run:    4608.66748
Std. Deviation:             42.2188100154815
------------------------
Runtimes in Milliseconds:
Run 01:                     4648.7922
Run 02:                     4563.9884
Run 03:                     4611.6269
Run 04:                     4623.2412
Run 05:                     4660.7016
Run 06:                     4566.9648
Run 07:                     4557.5137
Run 08:                     4672.7191
Run 09:                     4623.5439
Run 10:                     4557.583
```

Figure 5. Ergebnisse von: Point Instanzen vermeiden

Mit dieser ersten Verbesserung wurde die Laufzeit der Anwendung um **~ 0.7 sec** verbessert.

2.3. Ergebnisse von: *SelectNeighbor entfernen*

Folgende Teil zeigt die Analyseergebnisse der *Runtime*-Analyse dieser angewandten Verbesserung.



```
Runs:                       10
Iterations:                 100
Runtime in Milliseconds:    40555.7923
Avg. Milliseconds / Run:    4055.57923
Std. Deviation:             46.0901552732217
------------------------
Runtimes in Milliseconds:
Run 01:                     4079.6591
Run 02:                     4047.0637
Run 03:                     4104.3861
Run 04:                     4148.8433
Run 05:                     4001.0242
Run 06:                     4054.1996
Run 07:                     3992.8962
Run 08:                     4079.8272
Run 09:                     4026.5757
Run 10:                     4021.3172
```

Figure 6. Ergebnisse von: SelectNeighbor entfernen

Mit dieser zweiten Verbesserung wurde die Laufzeit der Anwendung um **~ 1.4 sec** verbessert.

2.4. Ergebnisse von: *GenerateMatrix, RandomizeMatrix modifizieren*

Folgende Teil zeigt die Analyseergebnisse der *Runtime*-Analyse dieser angewandten Verbesserung.

```
Runs:                       10
Iterations:                 100
Runtime in Milliseconds:    35784.5365
Avg. Milliseconds / Run:    3578.45365
Std. Deviation:             18.5103239067533
-------------------------
Runtimes in Milliseconds:
Run 01:                     3606.3749
Run 02:                     3595.0335
Run 03:                     3593.541
Run 04:                     3587.0629
Run 05:                     3558.9391
Run 06:                     3581.4392
Run 07:                     3557.9559
Run 08:                     3580.595
Run 09:                     3543.0791
Run 10:                     3580.5159
```

Figure 7. Ergebnisse von: GenerateMatrix, RandomizeMatrix modifizieren

Mit dieser letzten Verbesserung wurde die Laufzeit der Anwendung um **~ 1.8 sec** verbessert.

3. Quelltextverbesserungen

Folgender Teil beschäftigt sich mit den Optimierungen, die angewendet wurden um das Laufzeitverhalten zu verbessern.

3.1. Point Instanzen vermeiden

Die Methode `GetNeighbors(Type, Point)` wurde dahingehend verändert, dass nicht mehr alle Nachbarn besucht und zurückgeliefert werden, sondern dass per Zufall solange die möglichen Nachbarn besucht werden bis das erwartete Resultat eintritt. DasDer erste gefundene Nachbar, der die Anforderungen erfüllt, wird zurückgeliefert.

Siehe hierzu die folgenden Methoden in *Part1WatorWorld*:

- *public Point GetNeighbors(Type type, Point position)*
- *public Point SelectNeighbor(Type type, Point position)*

In der Methode `SelectNeighbor` wurde lediglich folgende Änderung vorgenommen `Point[] neighbors = new Point[] { GetNeighbors(type, ref position) };`.

Durch die Änderungen in `GetNEighbors(Type, Point)` wird den `Animal` Instanzen verwehrt zu entschieden in welchge Richtung sie gehen wollen. Dieser Ansatz wurde aber gewählt, da er gut für die Verbesserung des Laufzeitverhaltens ist. Siehe dazu folgende Analyseergebnisse Section 1.2, “Ergebnisse von: *Point Instanzen vermeiden*”

3.2. SelectNeighbor entfernen

Nachdem die Methode `GetNeighbors(Type, Point)` so geändert wurde, dass hbier bereits eine einzige Position eines Nachbarn zurückgelifert wird, kann auf die Methode *SelectNeighbor* verzichtet werden, da die implementierte Logik keine Anwendung mehr findet. Im Zuge dessen wird die Methode `GetNeighbors` unbenannt in `GetNeighbor`, da diese Methode nurmehr ein Resultat und kein Array mehr zurückliefert.

3.3. GenerateRandomMatrix, RandomizeMatrix modifizieren

Anstatt ein zweidimensionales Array zu verwenden in dem die Indixes abgebildet sind, wird eine Liste von *Point*-Objekten beim erstamligen Erstellen der Matric erstellt und bei jedem Aufruf der Methode `ExecuteStep` zufällig neu geordnet. Es werden zwar alle Positionen auf der Matrix über *Point*-Objekte abgebildet, aber diese *Point*-Objekte bleiben über die Laufzeit erhalten und werden daher vom *Garbage Collector* nicht beachtet, da immer eine Referenz

auf diese Objekte besteht. Das zufällige Besuchen der Positionen bleibt gewährleistet. Des Weiteren werden alle *Animal*-Objekte, die als in der MEthode `ExecuteStep` als *Moved* markiert wurden in einer Liste gespeichert und nachdem alle Positionen besucht wurden *commited*. Dadurch wird ein weiteres Iterieren über alle Positionen vermieden.

Siehe dazu die folgenden Methoden in der Klasse *Part3WatorWorld*:

- `ExecuteStep`
- `ShuffelPoints` (Ersetzt RandomizeMatrix)