Gobales Gedächtnis

AntMe

Sebastian Loers

lb-sys.info

Projekt: AntMe - Globales Gedächtnis (Version 1.0)

Dokument Version: 1.0
Erstellt am: 15.04.2007
Geändert am: 30.04.2007

Inhalt

1 EINLEITUNG

In diesem kleinen Tutorial möchte ich euch zeigen, wie ihr mit relativ einfachen Mitteln euren Ameisen ein globales Gedächtnis verleiht.

2 GLOBALES GEDÄCHTNIS ERSTELLEN

Zuerst müssen wir uns überlegen, wie bekommen wir es hin einen Speicher für zum Beispiel Koordinaten zu erstellen, auf den alle Ameisen Zugriff haben.

Manche Leute kommen jetzt sicher auf Gedanken wie eine Datenbank oder das Speichern im Dateisystem, doch das ist viel zu umständlich für unsere zierlichen Ameisen, also müssen wir uns etwas Einfacheres überlegen.

Das Zauberwort heisst in unserem Fall „static“. Wir möchten also unserer globalen Daten, hier in diesem Tutorial speziell die Koordinaten, in einer statischen Collection abspeichern.

Kommen wir also schon zu unserem ersten Schritt. Wir erstellen uns eine neue Klasse mit dem Namen „KoordinatenSpeicher“. Diese Klasse machen wir statisch und erstellen uns dort drin 3 Collections.

[C#] KoordinatenSpeicher

```
public static class KoordinatenSpeicher<K,V> {
  private static Dictionary<K, V> zuckerKoordinaten = new Dictionary<K, V>();
  private static Dictionary<K, V> obstKoordinaten = new Dictionary<K, V>();
  private static Collection<int> geloeschteObjekte = new Collection<int>();

  public static Dictionary<K, V> ZuckerKoordinaten {
    get {
      return zuckerKoordinaten;
    }
  }

  public static Dictionary<K, V> ObstKoordinaten {
    get {
      return obstKoordinaten;
    }
  }

  public static Collection<int> GeloeschteObjekte {
    get {
      return geloeschteObjekte;
    }
  }
}
```

[VB] KoordinatenSpeicher

```
Imports System.Collections.ObjectModel
Public Class KoordinatenSpeicher(Of K, V)
    Private Shared zuckerKoordinaten As New Dictionary(Of K, V)
    Private Shared obstKoordinaten As New Dictionary(Of K, V)
    Private Shared geloeschteObjekte As New Collection(Of Int32)
    Public Shared ReadOnly Property KoordinatenZucker() As Dictionary(Of K, V)
        Get
            Return zuckerKoordinaten
        End Get
    End Property
    Public Shared ReadOnly Property KoordinatenObst() As Dictionary(Of K, V)
        Get
            Return obstKoordinaten
        End Get
    End Property
    Public Shared ReadOnly Property ObjekteGeloescht() As Collection(Of Int32)
        Get
            Return geloeschteObjekte
        End Get
    End Property
End Class
```

Jetzt Fragt ihr euch bestimmt was hat der Typ da gemacht. Ich erkläre es euch.

Also wir ihr seht haben wir jetzt 3 Collections erstellt. Die erste Collection „zuckerKoordinaten“ ist dafür bestimmt alle Zucker Koordinaten abzuspeichern. Die 2te Collection „obstKoordinaten“ ist für die Obst Koordinaten zuständig. Dann seht ihr hier noch, dass ich hier generische Typen benutzt habe und zwar K und V. K steht in diesem Falle für den Typen des Keys und V für den Typen des Values. Als Key nehmen wir im weiteren Tutorial immer die ID des Obstes oder Zuckers und also Value immer das Obst oder den Zucker selbst, aber dazu später mehr.

Als drittes haben wir noch die „geloeschteObjekte“, hier werden alle IDs der Obst oder Zuckerobjekte gespeichert, die schon gelöscht wurden, auch dazu später mehr.

Das war dann schon erst einmal alles was wir für unseren Speicher benötigen um ihn zu benutzen.

3 GLOBALES GEDÄCHTNIS BENUTZEN

Hier zeige ich euch, wie ihr richtig mit dem Speicher umgeht.

3.1 Die Initialisiere() Methode

In dieser Methode müssen wir unseren Speicher initialisieren. In unserem Fall heißt das, diesen zu leeren um sicher zu gehen, dass sich keine alten Koordinaten mehr darin befinden. Außerdem initialisieren wir die Typen mit int und IKoordinate.

[C#] Initialisiere()

```
public static void Initialisiere() {
  KoordinatenSpeicher<int, IKoordinate>.ZuckerKoordinaten.Clear();
  KoordinatenSpeicher<int, IKoordinate>.ObstKoordinaten.Clear();
  KoordinatenSpeicher<int, IKoordinate>.GeloeschteObjekte.Clear();
}
```

[VB] Initialisiere

```
Public Shared Sub Initialisiere()
    KoordinatenSpeicher(Of Int32, IKoordinate).KoordinatenObst.Clear()
    KoordinatenSpeicher(Of Int32, IKoordinate).KoordinatenZucker.Clear()
    KoordinatenSpeicher(Of Int32, IKoordinate).ObjekteGeloescht.Clear()
End Sub
```

3.2 Koordinaten speichern

Jetzt sind wir bereit um Koordinaten in unseren Speicher hinzuzufügen. Damit dies auch richtig funktioniert und wir die Daten nicht einfach nur unserer Collection hinzufügen erstellen wir uns eine Methode SpeicherKoordinate(). In dieser Überprüfen wir ob diese Koordinate nicht bereits im Speicher existiert und ob sie nicht schon einmal wieder gelöscht wurde. Dieser Methode übergeben wir die ID des Objektes und das Objekt selbst als IKoordinate, weil Obst und Zucker jeweils davon ableiten.

[C#] SpeicherKoordinate(int id, IKoordinate koordinate)

```
    private static void SpeicherKoordinate(int id, IKoordinate koordinate) {
      if(koordinate is Zucker) {
        if(!KoordinatenSpeicher<int, IKoordinate>.ZuckerKoordinaten.ContainsKey(id) &&
           !KoordinatenSpeicher<int, IKoordinate>.GeloeschteObjekte.Contains(id)) {
          KoordinatenSpeicher<int, IKoordinate>.ZuckerKoordinaten.Add(id, koordinate);
        }
      } else {
        if(!KoordinatenSpeicher<int, IKoordinate>.ObstKoordinaten.ContainsKey(id) &&
           !KoordinatenSpeicher<int, IKoordinate>.GeloeschteObjekte.Contains(id)) {
          KoordinatenSpeicher<int, IKoordinate>.ObstKoordinaten.Add(id, koordinate);
        }
      }
    }
```

[VB] SpeicherKoordinate()

```
    Public Shared Sub SpeicherKoordinate(ByVal id As Int32, ByRef koordinate As IKoordinate)
        If TypeOf koordinate Is Zucker Then
            If Not KoordinatenSpeicher(Of Int32, IKoordinate).KoordinatenZucker.ContainsKey(id)
AndAlso Not KoordinatenSpeicher(Of Int32, IKoordinate).ObjekteGeloescht.Contains(id) Then
                KoordinatenSpeicher(Of Int32, IKoordinate).KoordinatenZucker.Add(id, koordinate)
            End If
        Else
            If Not KoordinatenSpeicher(Of Int32, IKoordinate).KoordinatenObst.ContainsKey(id)
AndAlso Not KoordinatenSpeicher(Of Int32, IKoordinate).ObjekteGeloescht.Contains(id) Then
                KoordinatenSpeicher(Of Int32, IKoordinate).KoordinatenObst.Add(id, koordinate)
            End If
        End If
    End Sub
```

Sobald eine Koordinate weder existiert noch schon einmal gelöscht wurde wird diese im jeweiligen Speicher gespeichert, je nach dem ob es sich um Obst oder Zucker handelt.

3.3 Koordinaten Löschen

Um eine Koordinate zu löschen erstellen wir uns wiederum eine Methode LoescheKoordinate(). In dieser löschen wir die Koordinate aus dem Speicher und fügen Sie dem GelöschteObjekte Speicher hinzu um überprüfen zu können ob diese Koordinate schon gelöscht wurde, wichtig bei Obst.

[C#] LoescheKoordinate(int id)

```
    private static void LoescheKoordinate(int id) {
      KoordinatenSpeicher<int, IKoordinate>.ObstKoordinaten.Remove(id);
      KoordinatenSpeicher<int, IKoordinate>.ZuckerKoordinaten.Remove(id);
      KoordinatenSpeicher<int, IKoordinate>.GeloeschteObjekte.Add(id);
    }
```

[VB] LoescheKoordinate()

```
    Public Shared Sub LoescheKoordinate(ByVal id As Int32)
        KoordinatenSpeicher(Of Int32, IKoordinate).KoordinatenObst.Remove(id)
        KoordinatenSpeicher(Of Int32, IKoordinate).KoordinatenZucker.Remove(id)
        KoordinatenSpeicher(Of Int32, IKoordinate).ObjekteGeloescht.Add(id)
    End Sub
```

3.4 Ziel für die Ameise holen

Jetzt möchten wir in der Lage sein für die Ameise die nächste Koordinate zu beschaffen. Hierfür bauen wir uns jeweils für Obst und für Zucker eine Methode GibZucker() und GibObst(). Hier müssen wir jeweils prüfen, ob sich überhaupt schon Koordinaten im Speicher befinden. Wenn dem so ist holen wir uns die erst beste Koordinate aus dem Speicher, ansonsten geben wir null zurück.

[C#] GibZucker() und GibObst()

```
    private static IKoordinate GibZucker() {
      if(KoordinatenSpeicher<int, IKoordinate>.ZuckerKoordinaten.Count > 0) {
        foreach(KeyValuePair<int, IKoordinate> koordinate in KoordinatenSpeicher<int,
           IKoordinate>.ZuckerKoordinaten) {
          return koordinate.Value;
        }
      }
      return null;
    }

    private static IKoordinate GibObst() {
      if(KoordinatenSpeicher<int, IKoordinate>.ObstKoordinaten.Count > 0) {
        foreach(KeyValuePair<int, IKoordinate> koordinate in KoordinatenSpeicher<int,
           IKoordinate>.ObstKoordinaten) {
          return koordinate.Value;
        }
      }
      return null;
    }
```

[VB] GibZucker()

```
    Public Shared Function GibZucker() As IKoordinate
        If KoordinatenSpeicher(Of Int32, IKoordinate).KoordinatenZucker.Count > 0 Then
            For Each pair As KeyValuePair(Of Int32, IKoordinate) In KoordinatenSpeicher(Of
Int32, IKoordinate).KoordinatenZucker
                Return pair.Value
            Next
        End If
        Return Nothing
    End Function
```

3.5 Koordinaten in den Speicher schreiben

Die sinnvollste Position um Koordinaten zu speichern, sind die Sieht() Methoden jeweils für Obst oder für Zucker.

Dort rufen wir jeweils unsere Speicher Methode auf.

[C#] Speichern der Koordinaten

```
public override void Sieht(Zucker zucker) {
  SpeicherKoordinate(zucker.Id,zucker);
}
public override void Sieht(Obst obst) {
  SpeicherKoordinate(obst.Id, obst);
}
```

[VB] Speichern der Koordinate

```
Public Overrides Sub Sieht(ByRef zucker As Zucker)
    SpeicherKoordinate(zucker.id, zucker)
End Sub
```

Damit hätten wir jetzt unseren Speicher gefüllt.

3.6 Koordinaten auslesen

Unsere Stelle um Koordinaten auszulesen ist die Tick() Methode. Diese eignet sich am besten um der Ameise ein Ziel zu geben.

Als erstes müssen wir hier prüfen, ob die Ameise bereits ein Ziel hat. Wenn dem nicht so ist müssen wir ihr eins zuweisen.

In meinem Fall habe ich 2 Arten von Ameisen, Sammler und ObstSammler, dazu muss ich wohl nix mehr sagen. Je nach dem welcher Typ grad aktuell ist, hole ich mir ein Ziel. Bekomme ich etwas anderes wie null von meiner Gib Methode zurück, Gehe ich zu diesem Ziel, ansonsten lasse ich die Ameise weiter geradeaus gehen.

Meine Kriterien um eine Koordinate wieder zu löschen sind folgende.

Bei Obst überprüfe ich ob dieses Obst noch Träger braucht, ist dem nicht so wird die Koordinate des Obstes gelöscht.

Beim Zucker frage ich ob der Zucker kleiner oder gleich der maximalen Last der Ameise ist, diese löscht dann die Koordinate und holt sich den letzten Rest Zucker, so gehen andere Ameisen nicht mehr zu diesem nicht vorhandenen Zuckerhaufen.

[C#] Ziel zuweisen

```
if(Ziel == null) {
  IKoordinate koord = (Typ == "Sammler") ? GibZucker() : GibObst();
  if(koord != null) {
    if(koord is Obst) {
      Obst obst = koord as Obst;
      if(BrauchtNochTräger(obst)) {
        GeheZuZiel(obst);
      } else {
        LoescheKoordinate(obst.Id);
        GeheZuBau();
      }
    } else {
      if((koord as Zucker).Menge >= 10) {
        GeheZuZiel(koord);
      } else {
        LoescheKoordinate ((koord as Zucker).Id);
        GeheZuBau();
      }
    }
  } else {
    GeheGeradeaus(Reichweite / 2 +1);
  }
}
```

[VB] Ziel zuweisen

```
 If Ziel Is Nothing Then
      If Me.Typ = "Sammler" Then
          Dim koord As Zucker
          koord = GibZucker()
          If koord Is Nothing Then
              GeheGeradeaus(Reichweite / 2 + 1)
              Return
          Else
              If koord.Menge >= 10 Then
                  GeheZuZiel(koord)
              Else
                  LoescheKoordinate(koord.Id)
                  GeheZuBau()
              End If
          End If
      Else
          Dim koord As Obst
          koord = GibObst()
          If koord Is Nothing Then
              GeheGeradeaus(Reichweite / 2 + 1)
              Return
          Else
              If BrauchtNochTräger(koord) Then
                  GeheZuZiel(koord)
              Else
                  LoescheKoordinate(koord.Id)
                  GeheZuBau()
              End If
          End If
      End If
```

4 ABSCHLUSS

Ich hoffe ich konnte euch verständlich erklären wie ihr einen einfachen Globalen Speicher einbaut und wie ihr ihn benutzt. Das einzige was ihr noch machen müsst sind eure ZielErreicht Methoden zu implementieren, ich denke das kann man aus dem originalen Tutorial ganz gut entnehmen.

Wenn ihr noch Fragen habt, dann schreibt doch einfach ins Forum.

Vielen Dank

5 GLOSSAR

Begriff	Erklärung
Static	Bedeutet dass diese Variable beim ersten Gebrauch initialisiert wird und das auch nur an dieser Stelle, dass heisst das diese Variable global für alle sichtbar ist
Collection	Generell gesehen eine Liste von Daten. Weiterhin findet man in einer Collection ein paar hilfreiche Methoden zum Sortieren etc.
Generisch	Generisch heisst, das der Typ von Daten den eine Collection oder ähnliches halten kann variieren kann. So umgeht man das lässtige Boxing, was natürlich wesentlich mehr Performanz bietet, daher sind Generische Listen immer schneller als das nicht generische equivalent, Da nach dem Kompilieren schon feststeht welcher Typ sich in der Liste befindet.