Deel 2: vechten met references en de garbage collector

Eerst even een herhaling van de basis: bij het programmeren moeten we kunnen bijhouden waar we onze data opslaan. Dit doen we door variabelen mooie namen te geven waarmee we ze altijd kunnen terugvinden. Maar voor de computer werkt dit heel anders. Deze houdt een verwijzing bij naar waar het stukje data opgeslagen staat in het geheugen. Dit heet een memory address, en wordt vaak opgeslagen in een pointer of een reference.

Elk programma op de computer krijgt een stukje geheugen toegewezen als die daar om vraagt. De programmeur kan zeggen “ik heb 10 megabyte aan geheugen nodig” en dan gaat de computer kijken waar hij 10 megabyte heeft, en geeft jou een adres terug van een plekje in het geheugen dat jij mag hebben. Maar je krijgt daarmee ook de verantwoordelijkheid om dit geheugen weer vrij te maken. Een programmeertaal als C# is ontworpen zodat jij hier niet over na hoeft te denken. Het enige wat jij als programmeur moet doen is zeggen wat je gaat opslaan, en de rest bepaalt het systeem: hoe groot jouw data is, het geheugen dat moet worden opgevraagd, en zelfs het opruimen daarvan. Dit scheelt de programmeur veel werk, maar net als bij de meeste dingen die de programmeur werk schelen gaat dit ook ten koste van performance.

Reference types en value types

Het verschil tussen een reference en een value type is een van de belangrijkste dingen die je moet weten over werken met geheugen in C#. Een value type bewaart de volledige inhoud van de instance in de variabele. Een reference type bewaart slechts een memory address naar de plek waar de inhoud van de variabele in het geheugen staat. Een value type is dus vergelijkbaar met een pointer in C/C++.

Om te weten of iets een reference of een value type is, moet je vooral kijken naar of het een class of een struct is. We kunnen bijvoorbeeld een data type maken voor een vertex met een class of een struct:

```
public class VertexRefType
{
    float x, y, z;
}

public struct VertexValueType
{
    float x, y, z;
}
```

Als je een lokale variabele opslaat, komt deze terecht op de zogenaamde ***stack***. Je kunt dit zien als het lokale geheugen van een functie of een statement. De stack staat altijd op een vaste plek, en alles wat op de stack staat, staat ook naast elkaar in het fysieke geheugen. Als je een variabele van een value type aanmaakt, wordt alle inhoud ervan op de stack geplaatst. Maar als je een reference type aanmaakt, wordt de inhoud op een willekeurige andere plek in het geheugen opgeslagen, op de zogenaamde ***heap***, en wordt slechts de pointer naar die plek in het geheugen opgeslagen.

```
VertexRefType myVert = new VertexRefType(2, 3, 4);


------------ stack -------------        ------------- heap ------------
 ## | ## | ## | 0x003b | ## | ##         ## | 2 | 3 | 4 | ## | ## | ##
---------------- ^ --------------        ----- ^ ----------------------
                 myVert                       0x003b
```

```
VertexValueType myVert = new VertexValueType(2, 3, 4);

------------ stack -----------
 ## | ## | ## | 2 | 3 | 4 | ##
---------------- ^ ------------
                 myVert
```

Een ander belangrijk verschil tussen reference en value types is hoe de data gekopieerd wordt. Bij een reference type maak je alleen een kopie van het memory adres. Bij een value type kopieer je de volledige inhoud. Dit maakt uit als je de variabelen daarna gaat aanpassen. Dit is wat er gebeurt bij een reference type:

```
VertexRefType myVert = new VertexRefType(2, 3, 4);

var copyVert = myVert;

--------------------------------        ------------------------------
 ## | ## | 0x003b | 0x003b | ##        ## | 2 | 3 | 4 | ## | ## | ##
---------- ^ ----------- ^ -----        ---- ^ ----------------------
      myVert      copyVert                   0x003b
```

```
// Pas een waarde van copyVert aan
copyVert.y = 5;

// De waarde veranderd voor zowel myVert als copyVert
--------------------------------        ------------------------------
 ## | ## | 0x003b | 0x003b | ##        ## | 2 | 5 | 4 | ## | ## | ##
---------- ^ ----------- ^ -----        ---- ^ ----------------------
      myVert      copyVert                   0x003b
```

En hier dezelfde code met een value type:

```
VertexRefType myVert = new VertexRefType(2, 3, 4);

var copyVert = myVert;
```

```
-------------------------------------
 ## | ## | 2 | 3 | 4 | 2 | 3 | 4 | ##
---------- ^ --------- ^ -------------
      myVert           copyVert
```

```
// Pas een waarde van copyVert aan
copyVert.y = 5;

// De waarde veranderd alleen voor copyVert
-------------------------------------
 ## | ## | 2 | 3 | 4 | 2 | 5 | 4 | ##
---------- ^ --------- ^ -------------
      myVert           copyVert
```

De volgende onderdelen gaan over waarom het verschil tussen value en reference types uit kan maken voor performance.

De garbage collector

De garbage collector ruimt jouw ongebruikte variabelen op. In principe is dit een super geoptimaliseerd proces. Het systeem is slim genoeg om te weten welke variabelen in gebruik zijn zonder dat hiervoor veel

Snelheid [waar hoort dit bij??]

De basis van deze hele masterclass is dat geheugen best langzaam is. Vroeger werden computers gemaakt als set, en werd het geheugen afgestemd op de processor. Langzamerhand werden deze steeds meer losgekoppeld, en werden ze apart van elkaar ontwikkeld. Processoren werden veel en veel sneller, terwijl geheugen lang niet even snel ontwikkelde