De Struct

Door Team 1D, 11-02-2020

Naar aanleiding van de vraag waarom wij gebruikmaken van een struct in onze Console Applicatie hebben wij een kortdurend, doch diepgaand, onderzoek uitgevoerd.

Wat is nou eigenlijk zo'n struct?

Een struct is een *value type* die normaliter gebruikt wordt om smalle groepen van gerelateerde variabelen in te kapselen, bijvoorbeeld de coördinaten van een rechthoek of de kenmerken van een item in een inventaris. De variabele van een *value type* bevat de instantie van het type. Dit betekent dat een *operation* op zo'n variabele alleen de instantie van die *value type* beïnvloedt. Instanties van *value types* worden gekopieerd wanneer ze als waarde doorgegeven worden, wanneer de instantie van een *value type* wordt veranderd, beïnvloedt dit niet de kopieën van deze instantie. *Value types* worden opgeslagen op de stack. Dit is het gedeelte van het geheugen van de computer waar de gebruiker niet bij kan. Allocatie en deallocatie wordt hier automatisch geregeld. Het nadeel van de stack is, is dat deze in verhouding tot de heap kleiner is en sneller "overstroomt" (stack overflow).
Structs kunnen onder andere constructors, constanten, fields, methoden, properties, indexers, operators, events en nested types bevatten. Let wel, wanneer meerdere van deze members gebruikt worden is het wellicht verstandiger een *class* te gebruiken.

Wat is dan een class?

Een class is een *reference type*. Dit houdt in dat wanneer een class aangemaakt wordt, de variabele waarnaar het object is gericht alleen een referentie naar dat geheugen bevat. Classes worden gebruikt om complex gedrag te modelleren of om data te bevatten die intentioneel weer wordt veranderd na het creëren van een class object. Veranderingen aan een instantie van een *reference type* beïnvloedt alle referenties die naar die instantie wijzen. *Reference types* worden opgeslagen op de heap van de computer. Het risico van de heap is dat er memory fragmentatie kan ontstaan. Dit ontstaat als volgt: op de heap kun je handmatig geheugen toewijzen en weer vrijmaken. Omdat dit willekeurig gebeurt bestaat er de kans dat er een soort gatenkaas van geheugenblokken ontstaat. Ook is de access time trager dan die van de stack. Natuurlijk kent de heap ook voordelen, zo is de kans op overstroming (overflow)

veel kleiner en kun je veel grotere “brokken data” opslaan, zoals vaak met classes gebeurd. Ook kun je later de grootte van je benodigde geheugenblok wijzigen.

Wat is dan precies het verschil tussen een struct en een class?

Struct	Class
Value Type	Reference Type
Gebruikt de stack	Gebruikt de heap
Bevat de instantie van het type.	Bevat de referentie naar de instantie van het type.
Wordt gebruikt voor kleine en onveranderlijke data.	Wordt gebruikt voor complexe en mogelijk veranderlijke data.
Structs kunnen niet overerven van andere classes of structs.	Classes kunnen overerven van andere classes of structs.
Struct members kunnen niet protected zijn, omdat ze niet overerven.	Class members kunnen protected zijn, omdat ze wel overerven.

Wanneer gebruik je dan een struct?

Je gebruikt een struct als:

- Het een enkele waarde representeert, bijvoorbeeld primitieve types (int, double, etc.)
- De grootte van de instantie kleiner is dan 16 bytes (int = 4 bytes)
- Het onveranderlijk is.
- Wanneer er geen frequente *boxing** plaatsvindt

**Boxing* = Het proces van het converteren van een *value type* naar een *reference type*, waarbij logischerwijs *unboxing* het proces van het converteren van een *reference type* naar een *value type* is. *Boxing* is impliciet, dit betekent dat je de compiler niet hoeft te “vertellen” dat je een integer naar een object “boxt”. Zie onderstaand voorbeeld:

```
Int32 x = 10;
object o = x ;  // Impliciet boxing

Int32 y = 10;
object obj = (object) y; // Expliciet Boxing
x = o; // Impliciet Unboxing
```

Waarom dan een struct en geen class binnen Tic Tac Toe?

Tic Tac Toe bestaat normaliter uit een speelboord met 9 cellen. In deze 9 cellen moeten 2 spelers om beurten een X of een O zetten, dit gaat door totdat één van de spelers 3 gelijke tekens heeft horizontaal, verticaal of diagonaal. Een andere mogelijkheid is dat het spel eindigt in een gelijkspel, dit gebeurt wanneer alle cellen gevuld zijn.

Binnen de Console Applicatie wordt een struct gebruikt om de coördinaten van een cel vast te leggen. Dit zijn dus 9 cellen met een X- en Y-waarde van 0, 1 of 2. Het betreft hier integers, dus de grootte van de instantie blijft dus onder de 16 bytes. De eerste twee voorwaarden kunnen dus al afgestreept worden.

- ✓ Het een enkele waarde representeert, bijvoorbeeld primitieve types (int, double, etc.)
- ✓ De grootte van de instantie kleiner is dan 16 bytes (int = 4 bytes)

Omdat het hier een vast speelbord van 9 cellen betreft met een vaste X- en Y-waarde, mag ook de derde voorwaarde afgestreept worden.

- ✓ Het onveranderlijk is.

De laatste voorwaarde, *boxing*, is een ingewikkeld onderwerp, toch kan ook deze vrij makkelijk afgestreept worden, omdat de cel niet wordt omgezet naar een *reference type*, maar deze op basis van zijn coördinaten wordt geschreven in de console.

- ✓ Wanneer er geen frequente boxing* plaatsvindt.

Conclusie

Kijkend naar bovenstaande beschrijving van de struct en de toepassing ervan op Tic Tac Toe, kan geconcludeerd worden dat de keus voor een struct een valide keus is. Natuurlijk was het ook mogelijk geweest om in plaats van een struct een class te gebruiken. In dit geval was een struct afdoende om het beoogde doel te ondervangen, omdat het “goedkoper” is voor het geheugen en de waarden niet per ongeluk gewijzigd kunnen worden door het verkeerd gebruiken van referenties in het geval van een class.

Hopende u hiermee voldoende geïnformeerd te hebben,

Wensen wij u veel succes met het spelen van Tic Tac Toe.