Programmeren Bulletin

hcc! programmeren

Interessegroep

26ste jaargang april 2019 Nummer 1

Inhoud

Redactioneel

Welkom in het nieuwe jaar. Het jaar dat we weer nieuwe programmeermogelijkheden zullen ontdekken.

Een deel van mijn website is klaar en u kunt al wat informatie lezen. Er komen nog veel meer tips, voorbeelden en tutorials op mijn website, dus ik houd u op de hoogte.
www.tronicasoftware.nl

Beginners, maar ook hobbyisten die verder zijn, kunnen problemen tegenkomen die opgelost moeten worden en dat hoeft geen ingewikkelde code te zijn.

De condities, de lussen, het nesten van code en zelfs de wiskundige berekeningen zijn nodig in de code. In deze artikelen laat ik zien hoe we dat kunnen programmeren.

De tutorial Arkanoid was een leuk voorbeeld over hoe we een game kunnen maken, maar het gaat er meer om hoe we de geheimen op het gebied van de grafische wereld kunnen ontdekken. Er komen dus nog meer leuke artikelen over het mooie programma Unity.

Marco Kurvers

Expressies en condities

Het meeste wat we in code gebruiken zijn expressies en condities. Daarnaast gebruiken we ook toekenningen. Een waarde toekennen aan een andere variabele doen we niet met condities. Het kan wel, maar er wordt dan niet een resultaat van een berekening terug gegeven, maar een waarde die alleen een 'waar' of 'onwaar' aangeeft.

Een expressie kan een som zijn, maar ook bestaan uit tekenreeksen. Wanneer we tekenreeksen gaan gebruiken, dan wordt de hele expressie een tekenreeks. Dat noemen we concatenatie.

Eigenlijk hoeft een expressie niet uit een meervoud van waarden te bestaan. Eén waarde is al een expressie. We kunnen met een expressie het volgende doen:

- **Een formule berekenen.**
- **Een object methode uitvoeren (de methode moet altijd een functie zijn).**
- **Een array toekennen (de variabele waar het array aan toegekend wordt, moet ook een array zijn. Een array element aan een heel array toekennen of andersom is niet toegestaan).**
- **Voorwaardelijke condities (de variabele waar het resultaat van de voorwaarde aan toegekend wordt, kan alleen een boolean 'true' of 'false', 'waar' of 'onwaar' zijn).**
- **Numerieke waarden en alfanumerieke waarden bij elkaar. Niet alle programmeertalen ondersteunen dat. Later meer daarover.**

Soms proberen we een som te berekenen terwijl in dezelfde expressie ook een conditie staat. Dat mag, maar de variabele kan dan geen boolean zijn. Controleer of u in uw code ook condities in expressies mag gebruiken.

De praktijk

Laten we eens kijken hoe de bovenstaande theorie eigenlijk werkt. In elke programmeertaal kan het anders zijn.

```
A = 0                 'is een toekenning dat A de waarde 0 krijgt
B = 20 + 40           'is een expressie met een som berekening waarvan B
                      'het resultaat krijgt van de som
C = B + 2             'is ook een som, maar samen met het resultaat van B

20 + 10 * 4           'een expressie waarvan eerst 10 * 4 wordt berekend
                      'en dan 20 + het resultaat ervan berekend wordt.
(20 + 10) * 4         'soms willen we dat echter niet en willen we juist
                      'eerst optellen; haakjes zijn dan noodzakelijk
(A + B) < C           'is een expressie die als een voorwaarde gecontroleerd
                      'wordt; eerst A + B, dan controleren of het
                      'resultaat kleiner is dan C
B = (A + B) < C       'werkt op dezelfde manier; B is een boolean die alleen
                      'een true of een false kan hebben
B = (A < C) + 2       'nu werkt het niet meer op dezelfde manier, want er is
                      'een conditie waarvan een 'waar' of 'onwaar' in een
                      'expressie opgeteld wordt met 2
```

```
                    'niet alle programmeertalen ondersteunen dat
S = "Hallo allemaal"    'een tekenreeks (in oudere Basic dialecten wordt
                        'een $ gebruikt: S$)
S = ""                  'de tekenreeks is leeg
S = 4                   'als S nog steeds een string is, dan kan dit in
                        'de meeste programmeertalen niet
S = "" + 4              'sommige programmeertalen, zoals C#, hebben een
                        'oplossing om toch numerieke waarden aan een
                        'string variabele toe te kennen
                        'Tip, ook Excel heeft die mogelijkheid door in
                        'cel A1 '4 te typen (inclusief de apostrof) en
                        'in de andere cel A1 op te geven plus een getal:
                        '=A1+2 zal een 6 geven
```

De laatste code 'S = "" + 4' wordt zelfs door VBA niet ondersteund. Hoewel C# het wel accepteert, is het toch beter om waarden van verschillende types te converteren naar het type dat hetzelfde is als het resultaat type:

```
S = "" + CStr(4)        'wordt ondersteund door de meeste Basic dialecten
S = "" + (4).ToString() 'wordt ondersteund door de .NET programmeertalen
S = "" & "Ha!"          'wordt ondersteund door Visual Basic .NET
```

Zo kunnen we een atoi("20") in C als Basic code schrijven: CInt("20").

Condities als voorwaarden
Bovenaan in de lijst kunt u zien dat

B = (A + B) < C

niet op dezelfde manier werkt als

B = (A < C) + 2

Er kunnen programmeertalen zijn die hierop een foutmelding geven, omdat je een resultaat van een voorwaarde niet kunt optellen. Typen we de tweede regel echter in VBA, dan wordt het toch berekend. Maar hoe?

De reden is dat VBA geen boolean waarden true en false kent. Een conditie in VBA geeft numerieke waarden terug. Een -1 als het waar is en een 0 als het niet waar is. Typen we onderstaande regel in VBA

B = (10 > 5) + 9

dan krijgen we na deze print regel

print B

het getal 8 te zien.
Er wordt dus eigenlijk berekend: (-1) + 9

Het is dus duidelijk dat we goed moeten kijken hoe we een conditie in een expressie schrijven.

Expressies en condities in lussen

De bovenstaande voorbeelden gebruiken we niet alleen op basis van toekennen, en controleren of iets waar of onwaar is. We kunnen ook de code meermalen herhalen op basis van een voorwaarde.

De FOR lus

Bijna alle programmeertalen hebben deze lus. Bij sommige talen is de FOR lus de enige lus die geen conditie gebruikt. Het is een herhalingslus met een startwaarde en een eindwaarde. Deze waarden kunnen ook expressies zijn.

Het hangt af van welke programmeertaal de FOR lus gebruikt wordt. Basic heeft een NEXT statement, zodat de structuur FOR … NEXT is. C en Java gebruiken FOR { … } en Pascal gebruikt FOR BEGIN … END.

De feitelijke syntax van deze lus is:

FOR <v=beginwaarde> TO <eindwaarde>	**Basic**
FOR <v:=beginwaarde> [DOWN]TO <eindwaarde>	**Pascal**
FOR <v=beginwaarde> TO <eindwaarde> STEP <stapwaarde>	**Basic**
FOR (<v=beginwaarde>; <voorwaarde>; <stapwaarde met v>	**C/C++/C#/Java**

```
FOR v=1 TO 10
FOR v:=1 TO 10            FOR v:=10 DOWNTO 1
FOR v=0 TO 100 STEP 5     FOR v=100 TO 0 STEP -5
for (v=0; v<=100; v+=5)   for (v=100; v>=0; v-=5)
```

Kijk hoe de FOR lus werkt in uw programmeertaal.

Onregelmatig herhalen

In sommige Basic dialecten, waaronder VB6 en VBA, kan men de FOR lus onregelmatig laten uitvoeren, door een conditie als start- en/of eindwaarde te gebruiken. De volgende code laat het zien.

```
FOR I = -5 TO (A > 10)
     PRINT I
NEXT
```

Drukt het volgende af als A groter is dan 10:

```
-5
-4
-3
-2
-1
```

Was A niet groter geweest dan 10, dan had er onder de -1 nog een 0 gestaan.

Merk op dat u een gegeven conditie in een Basic FOR lus niet kunt vergelijken met een conditie als eindwaarde in een C for lus.

Andere lussen

Het enige statement dat we in Pascal missen, is het STEP statement. We kunnen geen Pascal FOR lus in stappen laten uitvoeren. Ook voor omhoog naar omlaag moeten we DOWNTO gebruiken.

Pascal kent een andere lus om toch een FOR lus met stappen te kunnen nabootsen, maar we zullen zien dat de lus geen eindwaarde kent, maar alleen eindigt op een bepaalde voorwaarde.

```
var
     I: Integer;
begin
     I := 0;
     repeat
          Writeln('Waarde is: ', IntToStr(I);
          Inc(I, 5);
     until I > 100;
end.
```

Maar onderstaande While lus kan in alle bovengenoemde talen werken. Let wel op de syntax van uw programmeertaal.

```
var
     I: Integer;
begin
     I := 100;
     while I >= 0 do
     begin
          Writeln('Waarde is: ', IntToStr(I);
          Dec(I, 5);
     end
end.
```

Merk op dat alleen een repeat ... until lus geen begin ... end heeft.

In Basic en in C kunnen we de while lus ook omkeren.

```
Basic               C/C++/C#/Java(Script)
DO                  do
     ...            {
WHILE <conditie>         ...
                    } while (<conditie>);
```

Het omkeren van de while lus was al toegestaan in de tijd van Commodore 128 BASIC 7.0.

De FOR EACH lus

Sommige programmeertalen ondersteunen een object gestructureerde lus die door een lijst van elementen bladert tot het laatste element is bereikt. Dit kan een array zijn of een lijst van objecten.

De syntax is in Basic zo:
FOR EACH <element> IN <elementen> ... NEXT

En in C en anderen:
foreach (<element> in <elementen>) { ... }

Object Pascal en Delphi ondersteunen ook de lus, maar dan wel zonder each:
FOR <element> IN collectie DO

Sommige programmeertalen die de for each lus kennen, ondersteunen het OF statement in plaats van het IN statement.

Er bestaat ook een syntax die hetzelfde doet, maar sterk afwijkt van de schrijfwijze van de for each lus. De programmeertaal heet LINQ.

```
myArray.ToList().ForEach(x => Console.WriteLine(x));
```

Het is een query programmeertaal die delegate en lambda expressies ondersteund.

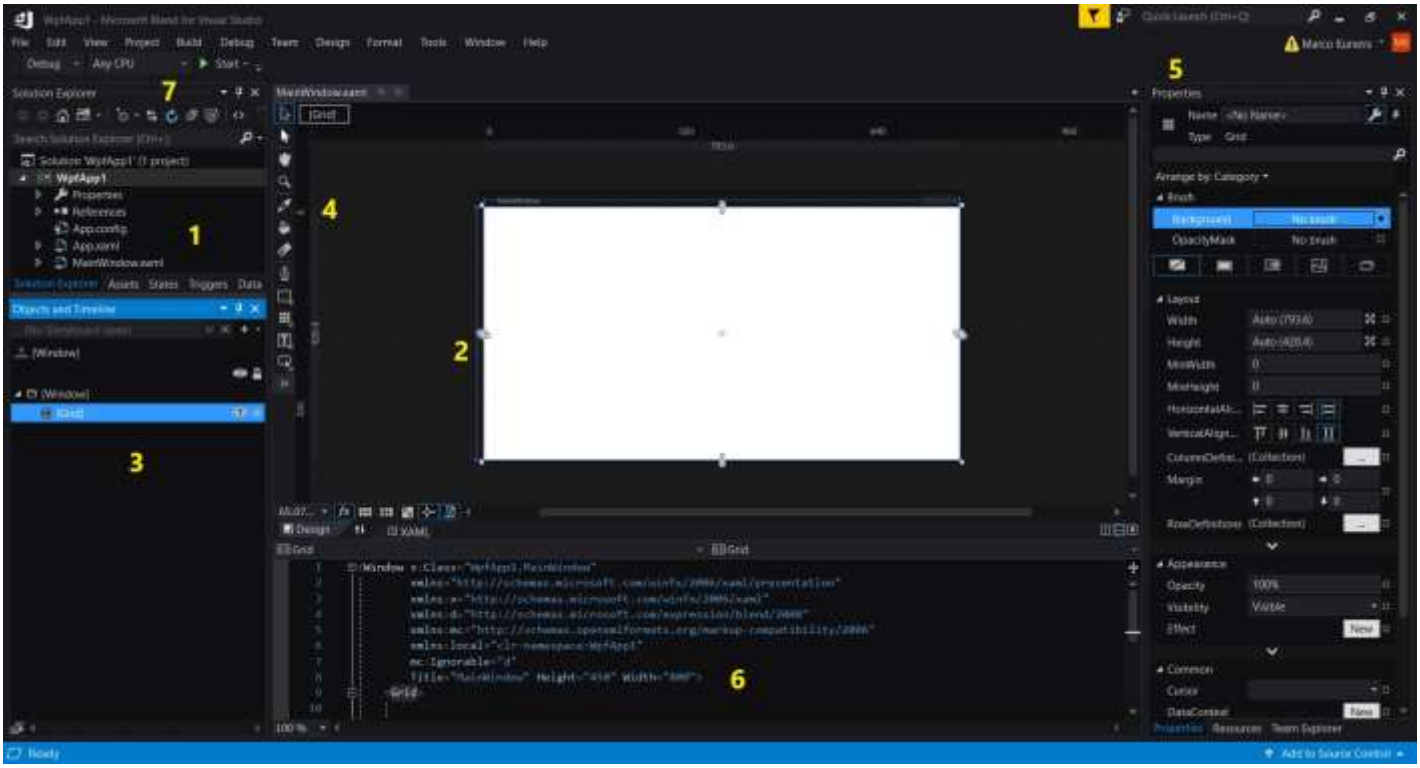
De volgende keer gaan we eens kijken hoe we expressies en condities in grotere programma's kunnen gebruiken en hoe we deze als parameters door kunnen geven in functies.

Blend in Visual Studio

Na de website editors Microsoft Frontpage en Microsoft Expression Web 4, kwam Microsoft met een hele nieuwe versie. Niet te vergelijken met de twee voorgangers, want deze heeft een hele nieuwe IDE, die eruit ziet als een normale Visual Studio IDE met een solution projectlijst, een form1 en een toolbox.

De nieuwe versie heet Blend. Verwar het niet met Blender. Heeft u Visual Studio met het .NET Framework, kijk dan eens of u ook Blend erbij heeft. Ik gebruik Visual Studio 2017.

Hieronder ziet u het IDE venster van Blend. De nummers geven aan wat het betekent.



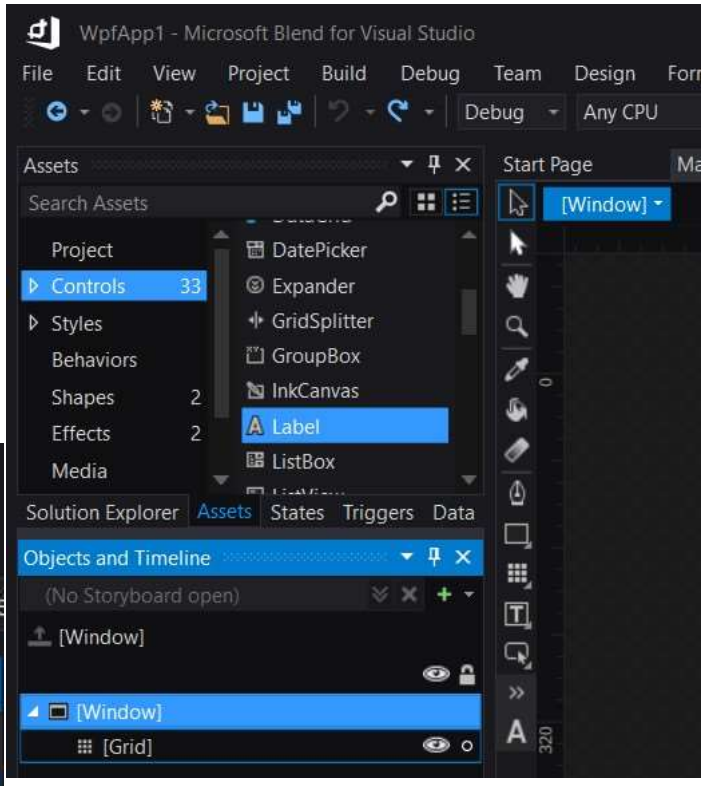
1. **Solution Explorer**
   Hier staan alle onderdelen van een project. Een solution kan meerdere projecten hebben.
2. **De MainWindow**
   Dit is het venster waarop we werken. Hier worden de componenten op geplaatst. Het witte gedeelte is niet het venster zelf, maar de Grid component. We hebben de Grid component nodig om de componenten op het venster te rangschikken.
3. **Objects and Timeline**
   Hier worden in een boomstructuur de objecten weergegeven die op de Grid component zijn geplaatst. U ziet dat het Grid als onderdeel bij de Window hoort.
4. **Toolbox**
   Hier kunt u alle componenten vinden die u gebruiken kunt voor uw project.
5. **Properties**
   Elk object heeft properties, dus ook het Grid.
6. **XAML code venster**
   Dit venster lijkt op een HTML code venster. De structuur is hetzelfde en de componenten worden ook als web componenten opgenomen. Er zijn twee mogelijkheden om de code te bewerken, via de Design mode of via de XAML mode. Wanneer u XAML

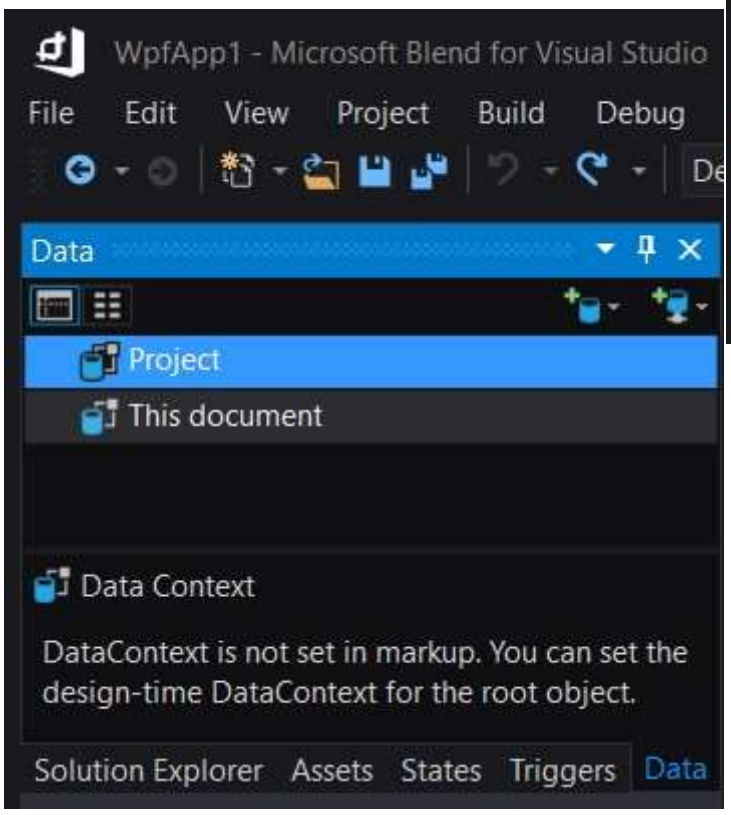
kiest, verdwijnen de menu-popups Design en Format (zie het menu helemaal bovenaan) en kunt u de code bewerken. Bent u minder bekent in XAML, dan is het beter om in de Design mode te blijven en gewoon de toolbox te gebruiken. Alles wat u met de muis doet, wordt automatisch in het XAML code venster bijgewerkt.

7. **Debug**
   Hoewel de naam Debug er niet staat, noem ik het toch zo. Met deze toolbox kunt u namelijk door uw WPF project bladeren en elke pagina debuggen. Bent u ergens op een pagina en wilt u direct terug naar de MainWindow, dan kunt u klikken op de Home knop (huisje). De eerste twee knoppen dienen om door elke pagina te bladeren. U kunt ook naar andere projecten en mappen die in de lijst onderaan staan en is het mogelijk om uw pagina's te vernieuwen met de Refresh knop.

Bij nummer 1 ziet u dat de Solution Explorer als tabblad open staat. U kunt ook andere tabbladen kiezen zoals de Assets, States, Triggers en Data.

Assets tabblad
In de Assets zijn meer mogelijkheden, zoals Styles, Behaviors, Shapes, Effects en Media. In de tweede lijst staan de componenten die u via Controls kunt gebruiken. De meeste componenten kunt u ook vinden in de toolbox. Het maakt dus niet uit welke u gebruikt. Heeft u de Solution Explorer even niet nodig, dan is het handig om gebruik te maken van de Assets om daaruit de componenten te gebruiken.





Data tabblad
Hier kunt u uw eigen gegevensstructuur maken. Rechtsboven op het Data tabblad ziet u twee knoppen:

Create sample data
Create data source

Zoals u de melding ziet, is er nog geen DataContext. Die moet altijd worden ingesteld voor het root object.

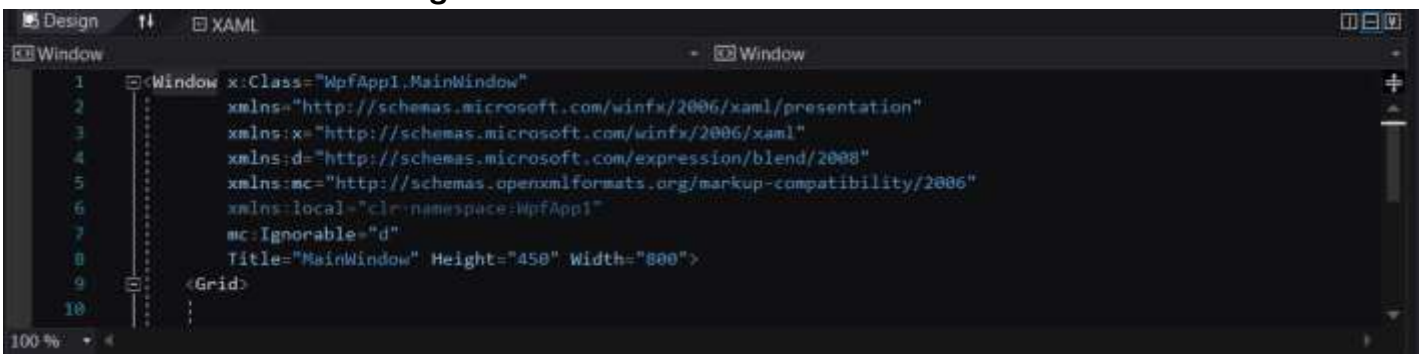
Het Project menu
In het Project menu kunt u onderdelen toevoegen, zoals een venster en een pagina. U kunt ook een eigen control maken.

Blender ondersteunt niet alleen XAML, maar ook script code. Door te klikken op Add New Item, kunt u een C# script, of welke andere Visual Studio programmeertaal dan ook, toevoegen aan uw WPF project.

Design en Format menu's
Deze twee menu's hebben te maken met het ontwikkelvenster. In het Design menu kunt u bepalen wat u aan wilt hebben staan, zoals de toolbox. In het Format menu kunt u de controls uitlijnen en rangschikken.

Deze twee menu's zijn niet aanwezig wanneer onderaan het XAML tabblad actief is.

Het XAML code venster
De afbeelding laat de XAML code zien. Voor het openen van een venster zijn een aantal schema's nodig die ingevoegd moeten worden, voordat het Grid mag verschijnen. Elk venster heeft deze schemaregels. De schema's zijn echter online en moeten met de instructie xmlns worden ingesteld.
Merk op dat het project met een namespace lokaal en als alleen-lezen wordt ingesteld, dat te zien is aan de donkere regel nummer 6.

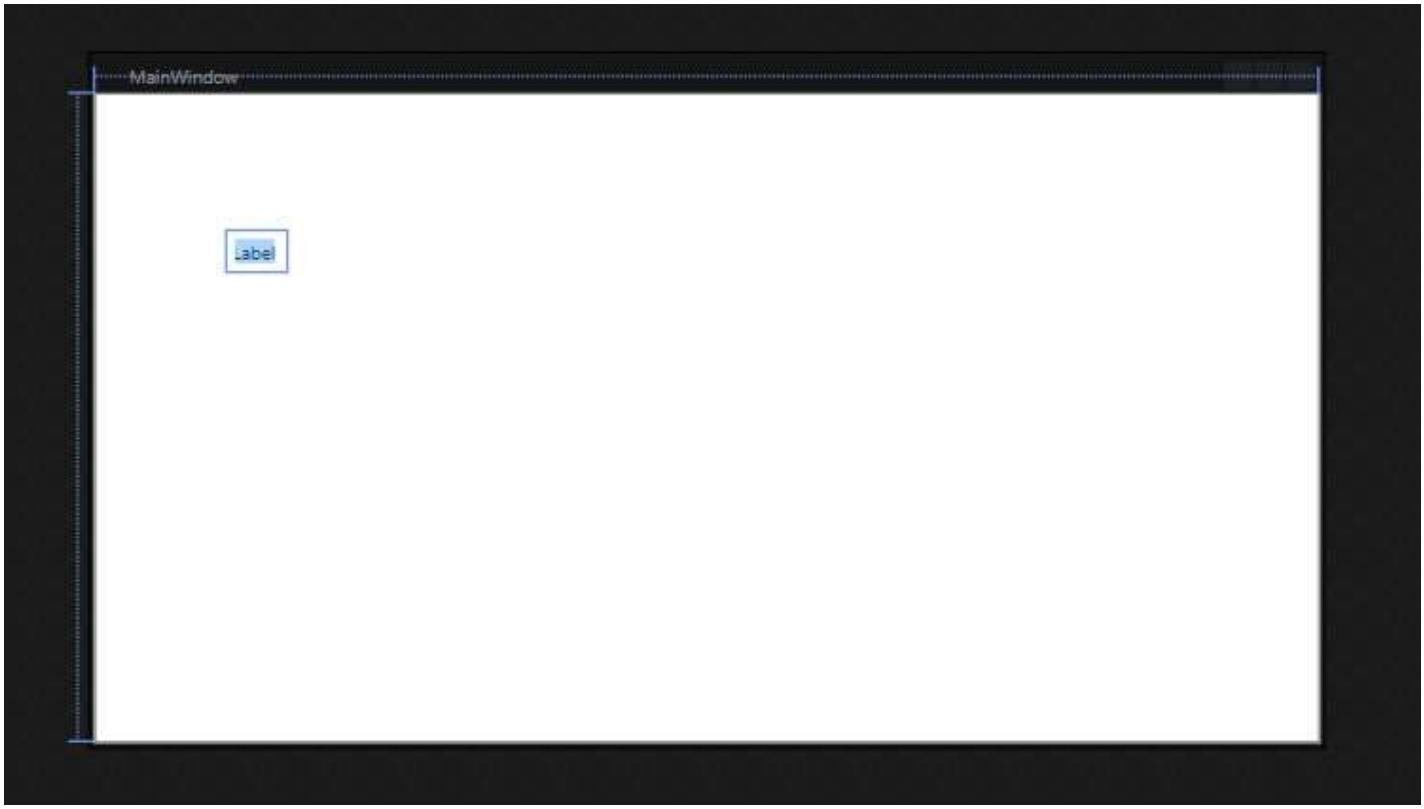


De code is vrij toegankelijk en mag gewijzigd worden, maar de drag en drop modus is nog altijd gemakkelijker. Alles wat u op het venster doet, wordt automatisch in de code bijgewerkt.

Wilt u echt in de code werken, klik dan op het tabblad XAML. Er verandert niets op de IDE, behalve dat twee menu's verdwijnen, Design en Format, die alleen met het draggen op het venster te maken hebben.

Laten we eens een Label control op het venster plaatsen en kijken wat er gebeurt.
Klik op het Assets tabblad, zie vorige pagina, en kies Controls.

Sleep de control naar het venster en laat het los.

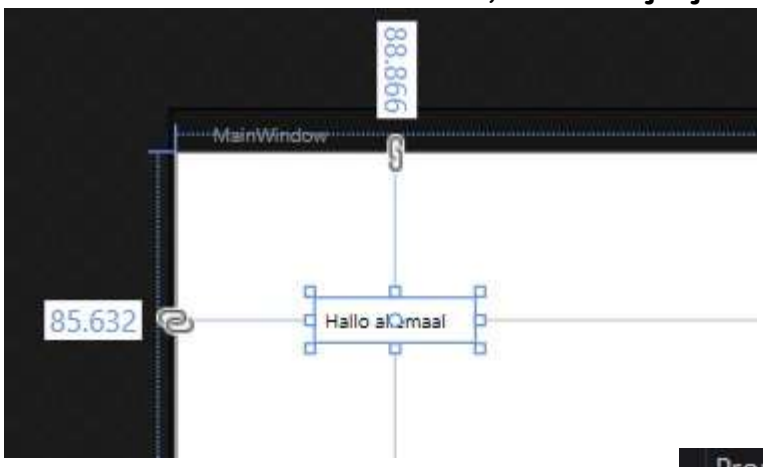


Standaard zal altijd de tekst Label verschijnen. U kunt het wijzigen door erop te klikken en uw tekst in te voeren. De code is ook veranderd.

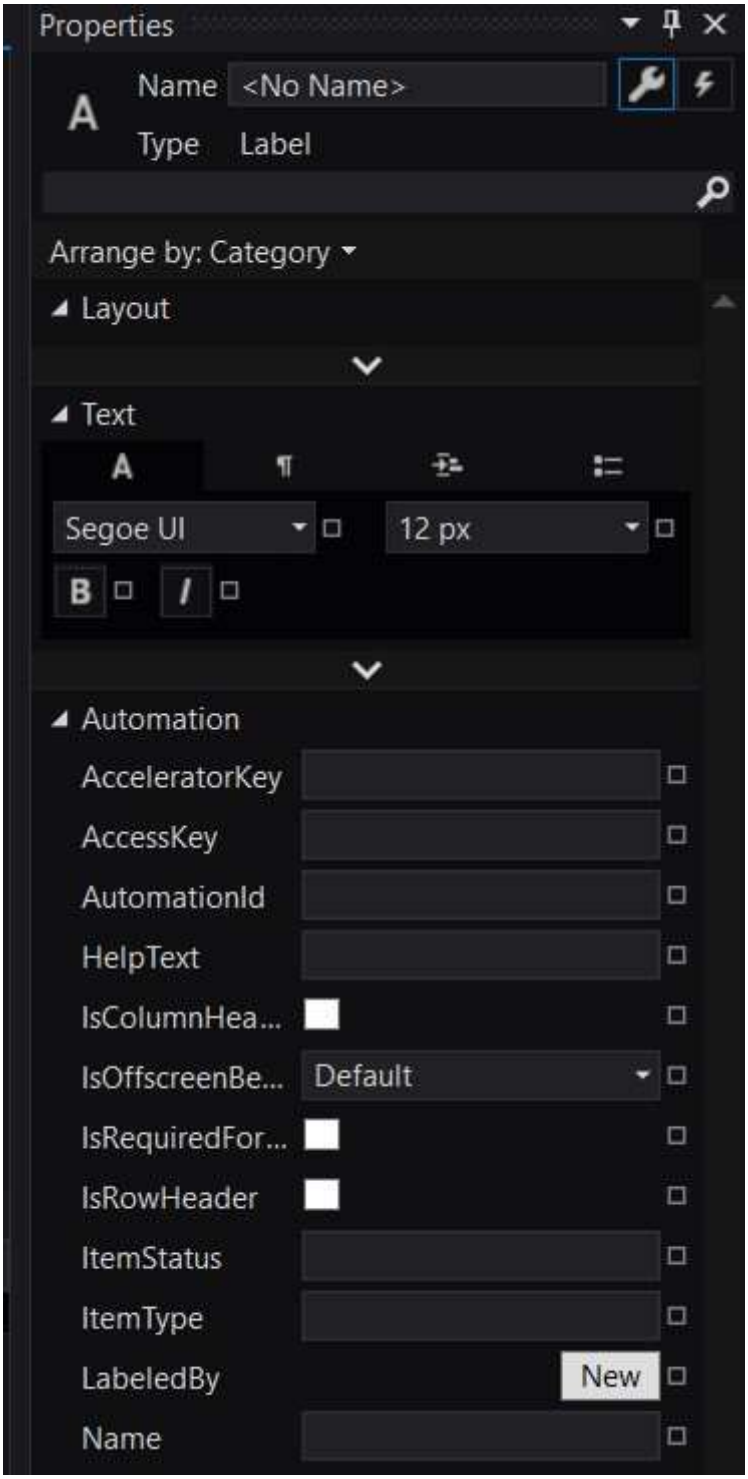
De instellingen die ook bij Properties aanwezig zijn, kunt u in de code wijzigen.
De tekstinhoud is de Content, dan volgt de horizontale uitlijning en dan de positie (Margin).
In plaats van op het venster te werken, kunt u de label aan uw eigen ideeën aanpassen.

Wijzig de tekst Label in Hallo allemaal.

U ziet een deel van het venster, waarschijnlijk met uw label op een andere positie.



Omdat de Label in de code actief is, wordt de control op het venster automatisch geselecteerd. De afstandspositie (Margin) wordt aangegeven met een horizontale lijn en een verticale lijn. De afstand wordt altijd gecentreerd bepaald.



De label heeft veel eigenschappen. U kunt hier alles wijzigen wat u wilt. De afbeelding hier laat niet alles zien, want u kunt ook een achtergrondkleur voor de control kiezen. Alles wat u in de Properties doet, wordt in de code opgenomen en andersom. Het maakt dus niet uit hoe u met Blend werkt.

Heeft u meer interesse in Blend en denkt u er genoeg van te hebben om nog met de oude HTML en CSS techniek te werken? Download Visual Studio 2017, want Blend die ik in dit artikel liet zien is een component van Visual Studio. Er is ook een Microsoft Blend 3 die u kunt downloaden, maar die is niet gratis en de trial is 60 dagen te gebruiken.

Zie de Appendix voor details.

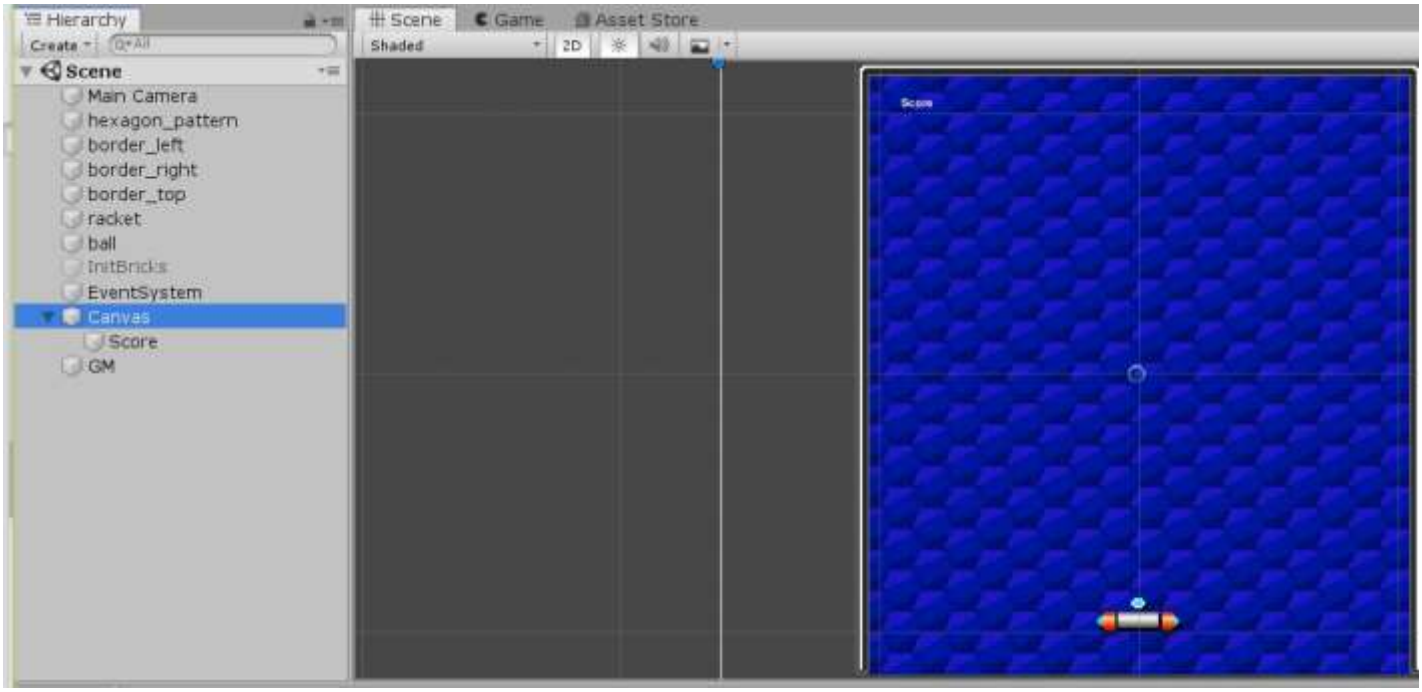
Unity 2D – De canvas en de prefabs

De tutorial Arkanoid was een leuke demo over hoe een 2D game ontwikkeld kan worden. Unity is geschikt voor zulke games en bovendien zeer gebruiksvriendelijk.

Er zijn tal van mogelijkheden in Unity. Ik ben zelf ook nog steeds meer mogelijkheden aan het uitproberen en het lukt mij al aardig. Wat ik allemaal gevonden heb, wil ik graag delen via de Bulletins, maar het komt ook op mijn website. Daarom zullen er meer leuke artikelen komen over Unity.

Hoewel Arkanoid een tutorial was op het gebied van de 2D omgeving, kan Unity ook werken op een 3D omgeving.
De twee onderdelen die ik nog niet uitgelegd heb, maar wel al in Arkanoid heb gemaakt, zijn de canvas en de prefabs. Die werken ook in 3D.

De canvas
De canvas is een venster waar we controls op kunnen plaatsen, zoals tekst, knoppen, lijsten, enzovoort. De reden dat het op een apart venster moet, komt omdat het de controls van Windows zijn en niet van grafische afkomst zijn. We kunnen deze daarom niet rechtstreeks op de scene plaatsen.



Bent u in het project Arkanoid of heeft u een nieuw project gekozen, ga dan naar het menu GameObject. Wijs UI aan en klik op Text. In de Hierarchy worden drie onderdelen toegevoegd:

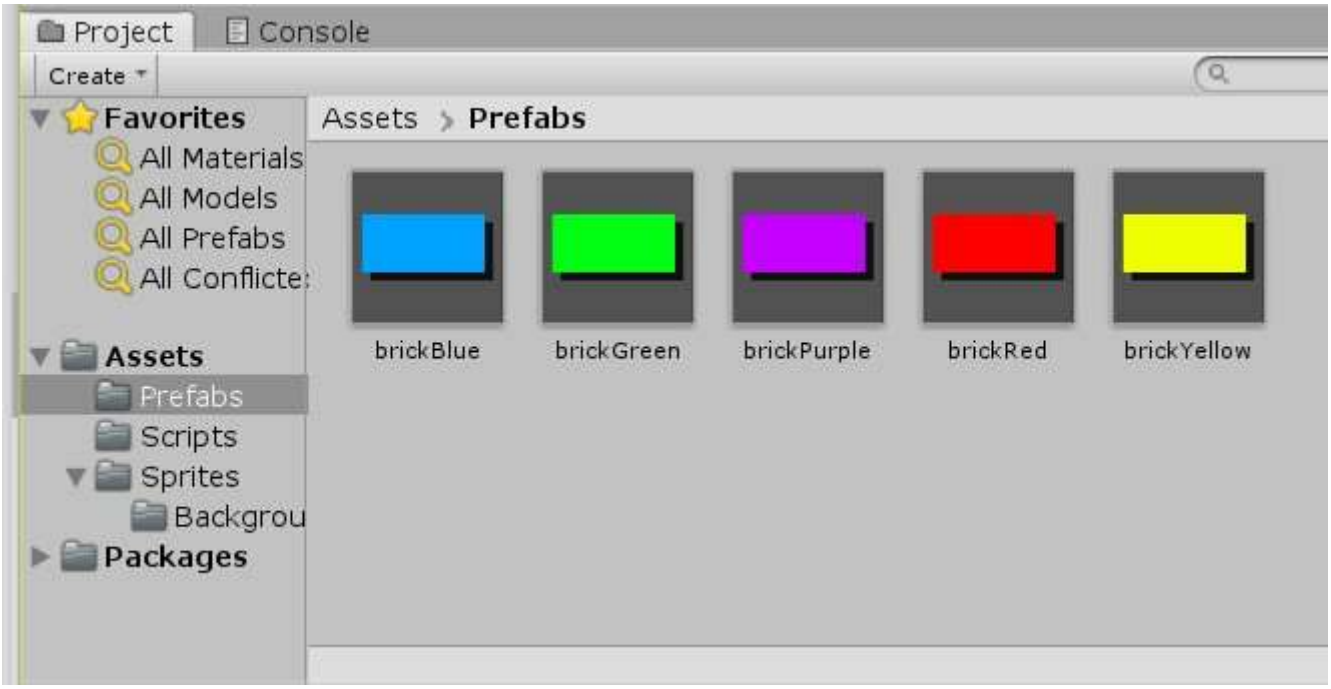
- **Canvas: het venster om tekst en andere controls op te plaatsen**
- **EventSystem: voor acties te ondernemen op de canvas**
- **Text: een van de controls die u ook via UI kunt kiezen**

Op de afbeelding ziet u dat ik de naam van de textcontrol gewijzigd heb in Score. Elke control wordt uitgelijnd onder de Canvas.

Hoewel de Canvas nodig is voor gebruik van de Windows controls, is het ook toegestaan om grafische Game Objecten uit te lijnen op de Canvas, zodat zij ook onderdeel er van worden.
Ook al is het toegestaan, het is beter om andere onderdelen buiten de Canvas toe te voegen. Bijvoorbeeld de prefabs kunnen een andere grootte krijgen wanneer ze binnen de Canvas worden toegevoegd.

Prefabs
Al vanaf het eerste deel van de tutorial heb ik gesproken over prefabs. Wat zijn eigenlijk prefabs?

Een prefab is een asset die opgebouwd is met componenten, vanuit een Game Object. Dat wil zeggen: een Game Object dat alles heeft wat nodig is. Zodra het Game Object naar de Assets wordt gesleept, wordt het een prefab. Zie afbeelding.

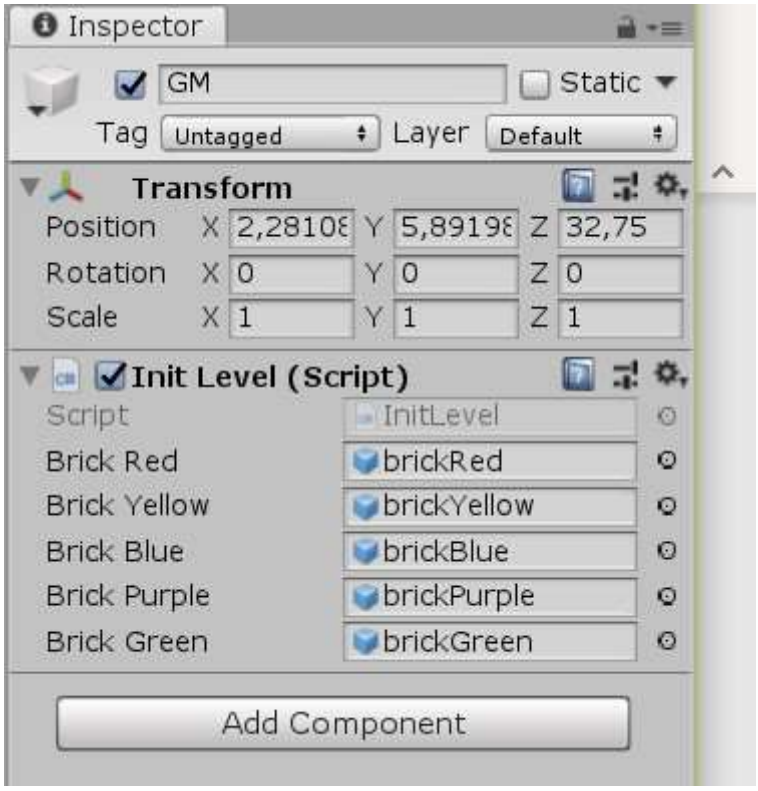


Elke prefab die we terug slepen naar de Hierarchy wordt weer een Game Object, met alle componenten erin. Prefabs zijn daarom zeer handig om tijd te besparen. Bij elk nieuw Game Object zouden we telkens weer componenten en scripts toe moeten voegen, wat extra veel werk kost. Door prefabs te maken, bespaart dat heel veel tijd.

Dit is ook de reden waarom ik geen bricks op de scene had staan, want de prefabs heb ik toegevoegd aan het GM Game Object. De GameManager die de level opbouwt.

Tegenwoordig weet ik nu dat deze manier helemaal niet hoeft. Alleen de prefabs op de goede plaats zetten is voldoende, en de level via code op laten bouwen is niet nodig. Maar het werkt wel.

Hieronder de afbeelding van de GameManager (GM).



De enige component die aan de Game-Manager is toegevoegd is de InitLevel script. Deze klasse bouwt de level op door elke instantie van de prefab te definiëren en op een nieuwe positie te plaatsen. Hoewel ik zei dat deze manier van levels maken niet hoeft, laat ik toch de code zien. Soms is het in het maken van games toch nodig om Game Objecten te instantiëren.

Hieronder ziet u de listing.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class InitLevel : MonoBehaviour {
   public GameObject brickRed;
   public GameObject brickYellow;
   public GameObject brickBlue;
   public GameObject brickPurple;
   public GameObject brickGreen;

   // Use this for initialization
   void Start () {
      for (int y = 74; y >= 42; y -= 8)
      {
        for (int x = -96; x <= 96; x += 16)
        {
          GameObject instBrick;
          switch (y)
          {
            case 74:
              instBrick = Instantiate(brickRed, transform.position, transform.rotation)
                as GameObject;
              break;
            case 66:
              instBrick = Instantiate(brickYellow, transform.position, transform.rotation)
                as GameObject;
              break;
            case 58:
              instBrick = Instantiate(brickBlue, transform.position, transform.rotation)
                as GameObject;
              break;
            case 50:
              instBrick = Instantiate(brickPurple, transform.position, transform.rotation)
                as GameObject;
              break;
            default:
              instBrick = Instantiate(brickGreen, transform.position, transform.rotation)
                as GameObject;
              break;
```

```
            }
            instBrick.transform.position = new Vector3(x, y, 0);

        }
    }
}
}
```

De functie Instantiate

Game Objecten kunnen doorgegeven worden aan de script, ook de prefabs. Er worden 5 Game Objecten gedeclareerd die de doorgegeven prefabs krijgen. In de binnenste for lus wordt een lokaal Game Object gedeclareerd die voor elke instantie moet zorgen. De cases bepalen welk Game Object geïnstantieerd mag worden.

De functie Instantiate heeft meerdere overloads, waardoor er verschillende parameters doorgegeven kunnen worden. Zo is het niet altijd nodig om 'as GameObject' aan het eind van de regel op te geven. Omdat niet de identity ervan wordt opgegeven, moet Unity weten wat voor een identity type gebruikt wordt. In plaats daarvan is onderstaande regel ook toegestaan:

```
instBrick = Instantiate(gameObject, transform.position,
Quaternion.identity);
```

Waarvan gameObject elk ander Game Object kan zijn.
Merk op dat ook de rotation er niet bij is. De identity zorgt er allemaal voor dat het object bekend is voor Unity. We hoeven dan ook alleen maar de position door te geven.

De volgende keer wil ik u meer details geven over Unity. Er komt dan vooral een artikel over hoe we met Unity kunnen beginnen.

Structures en lijsten in C#

Nieuwe gebruikerstypes maken geeft een uitbreiding in de programmeertaal. In Basic werkt het met een Type ... End Type en in Pascal met een Type <n> = Record ... End.

Het werken met types en records wordt beschouwd als nieuwe structuren. In de nieuwe Basic code, de .NET versies, wordt nu Structure ... End Structure ondersteund in plaats van Type ... End Type.

In de C programmeertalen worden ook structures gebruikt, met het sleutelwoord struct. We kunnen in C# onderstaande structure maken:

```
public struct nawStruct
{
     public string naam;
     public string adres;
     public string woonPlaats;
}
```

Om nawStruct te kunnen gebruiken, hebben we een variabele van het struct type nodig.

```
public nawStruct adresEtiket;
```

Willen we meerdere etiketten gebruiken, dan kunnen we een array maken.

```
public nawStruct[] adresEtiket;
```

In de declaratie mogen we niet direct het aantal elementen opgeven, maar als we dat toch willen doen dan zijn er twee mogelijkheden.

1. **Het aantal opgeven met het sleutelwoord new achter de declaratie**
2. **Het aantal met het sleutelwoord new pas opgeven in een functie of functie-event**

De eerste mogelijkheid is: `public nawStruct[] adresEtiket = new nawStruct[20];`
De tweede mogelijkheid is:

```
function functienaam
{
     adresEtiket = new nawStruct[20];
     ...
}
```

Structures in klassen
Wanneer we structures maken in een klasse en we willen de structure in object instanties kunnen gebruiken, dan moeten we ze public declareren. Dit is altijd zo als we types buiten de klasse ook nodig hebben.

Er geldt een belangrijk aspect met structures: wordt een structure public gemaakt, dan mag de structure variabele public zijn, maar is het structure type public, dan mag de structure variabele ook private zijn. Niet andersom. De reden is dat alleen de variabele public

maken niet voldoende is, omdat het structure type zelf buiten de klasse anders niet toegankelijk is.

Een structure variabele kan ook static zijn. Statische structures zijn in alle klassen toegankelijk zonder eerst een object instantie van de klasse, waar de structure in zit, te hoeven declareren. Overigens mag het structure type niet static zijn. Dat hoeft ook niet, want een struct type heeft public altijd toegang vanuit een andere klasse.

```
public class NAWClass
{
     public struct NAWStruct
     {
          string naam;
          string adres;
          string woonPlaats;
     }
     public static NAWStruct[] adresEtiket = new NAWStruct[20];

     public static void VulEtiketten()
     {
          // plaats hier code om de array te vullen
     }

     public static NAWStruct HaalEtiket(int etiketNr)
     {
          return adresEtiket[etiketNr];
     }
}

public class MainClass
{
     NAWClass.NAWStruct etiket;
     int nr = (int)eNr.text;  // haal invoer (Visual C/C#)

     NAWClass.VulEtiketten();
     etiket = NAWClass.HaalEtiket(nr);
     // dit werkt ook: etiket = NAWClass.adresEtiket[nr];
     // doe wat met het etiket
}
```

Statische variabelen en functies zijn niet altijd nodig. Bovenstaande code kan ook zonder het sleutelwoord static. De volgende code is dezelfde NAWStruct, maar nu met een instantie.

```
public class NAWClass
{
     public struct NAWStruct
     {
          string naam;
          string adres;
          string woonPlaats;
     }
     public NAWStruct[] adresEtiket = new NAWStruct[20];

     public void VulEtiketten()
     {
          // plaats hier code om de array te vullen
     }

     public NAWStruct HaalEtiket(int etiketNr)
     {
          return adresEtiket[etiketNr];
     }
}

public class MainClass
{
     NAWClass etiketten;
     NAWClass.NAWStruct etiket;
     int nr = (int)eNr.text;  // haal invoer (Visual C/C#)

     etiketten.VulEtiketten();
     etiket = etiketten.HaalEtiket(nr);
     // dit werkt ook: etiket = etiketten.adresEtiket[nr];
     // doe wat met het etiket
}
```

Het gebruik met of zonder statische variabelen maakt verschil met de hoeveelheid gebruikt geheugen. Statische variabelen maakt de variabelen als het ware globaal en kost daardoor geheugen. Lokale variabelen niet, want die staan op de stack en worden vernietigd als de functie verlaten wordt of een object van een klasse wordt vernietigd.

Structures in lijsten

Structure arrays zijn best handig, maar arrays hebben altijd een vast aantal elementen. In plaats van arrays kunnen we ook lijsten gebruiken. Bovenstaande structure kunnen we als een lijst declareren:

```
public List<NAWStruct> adresLijst = new List<NAWStruct>();
```

Nu hoeven we niet het aantal elementen op te geven. Telkens wanneer we een nieuw adresetiket willen maken, hoeven we alleen een structure, of genoemd als een record, aan de lijst toe te voegen.

```
public NAWStruct persoon = new NAWStruct;

persoon.naam = "Marco Kurvers"
persoon.adres = "Voorbeeldstraat 1"
persoon.woonPlaats = "Voorbeelddorp"

adresLijst.Add(persoon);
```

Let op! Als u probeert toegang te krijgen tot een structure veld rechtstreeks in een lijst, dan verschijnt er een compiler fout.

```
Cannot modify the return value of 'List<NAWStruct>.this[string]' because
it is not a variable
```

Vandaar dat u eerst een aparte structure moet declareren, daar de gegevens aan moet toekennen, voordat u alles aan de lijst kunt toevoegen.

Met de methode Remove() kan een structure uit een lijst worden verwijderd.

```
adresLijst.Remove(persoon);
```

Daarbij moet wel gelden dat de structure in de lijst bestaat. Het kan ook worden gecontroleerd met een if() statement, want de methode is een functie die een boolean teruggeeft.

```
if (adresLijst.Remove(persoon)) { }
```

Er kan dan een melding worden gegeven of het verwijderen wel of niet gelukt is.

Wilt u niet speciaal dat er een melding moet verschijnen, dan is er ook een andere methode waarmee u records kunt verwijderen uit de lijst met een gegeven index.

```
adresLijst.RemoveAt(index);
```

Maar waar halen we de index vandaan? Door een sleutelveld te gebruiken, kunnen we gebruik maken van deze methode. Denk er echter aan dat als u een record uit een lijst verwijdert, u ook het index nummer kwijt bent. Om ervoor te zorgen dat er geen 'gaten' in de lijst ontstaan, kunt u met een lus door de lijst bladeren en de index nummers vernieuwen.

```
int n = 0;

foreach (NAWStruct adres in adresLijst)
{
     adres.index++;
}
```

Is er een adres dat u verwijderen wilt, dan kunt u het huidig adres verwijderen door de index te gebruiken.

```
adresLijst.RemoveAt(huidigAdres.index);
```

Appendix – Websites en links voor hulp

Blend voor Visual Studio
https://docs.microsoft.com/en-us/visualstudio/designers/creating-a-ui-by-using-blend-for-visual-studio?view=vs-2019

Free Visual Studio
https://visualstudio.microsoft.com/downloads/

Free Pascal Documentatie en download
https://www.freepascal.org/docs.var
https://www.freepascal.org/download.html

Website PowerBASIC (versies niet gratis)
https://www.powerbasic.com/

Delphi from Embarcadero RAD Studio

https://www.embarcadero.com/products/delphi/starter

Python downloads
https://www.python.org/downloads/windows/

Go programmeertaal
https://golang.org
https://pythonprogramming.net/go/introduction-go-language-programming-tutorial/

Website over de LINQ Query programmeertaal
https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/getting-started-with-linq

Python documentatie
https://docs.python.org/3/index.html

Voor download Python 3.7 is er een directe link:
https://www.python.org/ftp/python/python.exe

Of kijk hier op de website:
https://www.pytorials.com/python-3-7-download-and-install-for-windows/

Unity website
https://unity.com/

Unity download
https://unity3d.com/get-unity/download