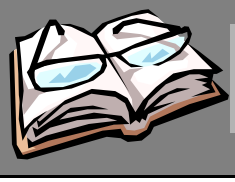
# Programmeren Bulletin

**24ste jaargang juni 2017** **Nummer 2**

hcc! programmeren

**Interessegroep**



# Inhoud



# Redactioneel

Door omstandigheden kan ik even niet verder gaan over Liberty BASIC API Reference, mijn excuses daarvoor.

We gebruiken structuren om het programmeren eenvoudiger te maken. Types gebruiken we om velden in een groep te gebruiken in plaats van losse variabelen. Enumeraties zijn zeer handig om namen te gebruiken op de plaats waar we getallen hebben staan. Losse constanten zijn dan niet speciaal nodig. Maar ook al bestaan deze computertermen, elke programmeertaal noemt ze weer anders.

Het klokproject in Unity was een voorbeeld hoe u 3D objecten d.m.v. scripts kunt uitvoeren op een scene. Unity biedt nog veel meer mogelijkheden. Een scene kan ook een level zijn van een spel.

In Visual Studio 6 zijn variabelen nog gewoon zoals we een variabele gebruiken: een waarde toekennen en ermee werken. In sommige programmeertalen werkt het nog steeds zo, maar in Visual Studio .NET niet meer. Variabelen zijn objecten geworden, en dat komt doordat alles wat we declareren uit het .NET Framework komt. We erven alles van het System.Object.

Macro's zijn gemakkelijk te maken in Excel. Uw bewerking even opnemen, klaar! Toch is Visual Basic voor Applications een betere methode om uw map uit te breiden dan alleen maar een macro maken. Bovendien neemt een macro ook handelingen op die helemaal niet nodig zijn. Dit komt allemaal aan bod in dit onderwerp.

Met Python gaan we kijken hoe we met tekst, condities en expressies kunnen werken. Condities hoeven niet speciaal in een if statement te staan, we kunnen het ook gebruiken in expressies. Dat is ook in andere programmeertalen het geval, maar het gaat om het geraamte van Python; de opbouw van de code.

**Marco Kurvers**

# Gedefinieerde structuren, records en enumeraties.

Elke programmeertaal gaat anders om met structuren, records en enumeraties. Sommige Basic versies en dialecten hebben Type definities, terwijl de .NET versies Structure definities hebben. Wat is het verschil?

Bekijken we een Type definitie in Visual Basic 6 en in Pascal, dan zien we dat in Pascal een type een record is, of ook genoemd: een recordtype.

**Basic**

```
Type PersoonType
     Voornaam As String * 20
     Achternaam As String * 20
End Type
```

**Pascal**

```
type TPersoon = record
     Voornaam: string[20];
     Achternaam: string[20];
end;
```

Tip! We kunnen in Pascal meerdere recordtypes definiëren zonder telkens met het sleutelwoord type te hoeven beginnen. In Basic is dat niet mogelijk.
Een andere tip is: U mag in Basic ook de typenaam met een T laten beginnen. Echter, alle voorgedefinieerde types en klassen in Pascal beginnen ook met een T.

Nu is de vraag wat nu het verschil is tussen een structuur, een type en een record? Ze lijken op elkaar. Vooral nu, omdat Microsoft het nog lastiger heeft gemaakt als we Visual Basic .NET bekijken.

```
Structure PersoonStruct
     Public Voornaam As String * 20
     Public Achternaam As String * 20
End Structure
```

Het bovenstaande normale Basic sleutelwoord Type wordt niet meer ondersteund. Basic code vanuit versie 6 naar .NET zal niet goed geconverteerd worden, ook al is er een converteerwizard aanwezig. In het onderwerp ‘Variabelen als objecten’ is er meer over de verandering van versie 6 naar .NET.

Een structure gebruiken in Liberty BASIC is geen type. Na de definitie kunnen we alleen deze structure gebruiken en kunnen we geen variabelen voor de structure declareren.

```
struct PersoonStruct, Voornaam$, Achternaam$

PersoonStruct.Voornaam$.struct = ...
```

U ziet dat in Liberty BASIC elk veld moet eindigen met struct. In andere Basic dialecten en versies hoeft dat niet.

Bekijken we meerdere Basic dialecten en versies, dan zien we verschillende structuren die dezelfde werking hebben. In Pascal en Delphi is dat niet het geval. Al vanaf de oude DOS Pascal versies (denk maar aan Turbo Pascal) zien we structuren die nu nog bestaan. Een voorbeeld zijn de enumeraties.

In Basic maken we een enumeratie als volgt:

```
Public Enum MenuKaart
     Visgerecht = 0
     Groentesoep
     Kersenvla
     Hutspot
End Enum
```

Omdat er zoveel Basic dialecten bestaan, beperk ik me hiermee tot één voorbeeld dat zeker zal werken in Visual Basic 6 en in VBA.
In plaats van vier constanten te declareren kunnen we nu dankzij de enumeratie alle vaste waarden vervangen door de enumeratievelden. Het menu Visgerecht begint met waarde 0 die echter optioneel is. Indien u voor het eerste veld geen waarde geeft, zal het automatisch beginnen met 0. U mag natuurlijk ook beginnen met een andere waarde en u mag ook voor elk veld een waarde geven. De andere menugerechten zullen vanaf de eerste veldwaarde met 1 worden verhoogd. Zo zal Groentesoep de waarde 1 krijgen.

In Pascal maken we enumeraties niet met het woord enum, maar gewoon met het woord type.

```
type
     MaandEnum = (januari, februari, maart, april,
                  mei, juni, juli, augustus, september,
                  oktober, november, december);

var Maand: MaandEnum;

begin
     Maand := juni;
end.
```

Zoals u ziet is het in Pascal erg handig om een variabele te moeten declareren voor een enumeratie. Op die manier kunnen we meerdere variabelen gebruiken voor dezelfde enumeratie.

Helaas werkt het in Basic niet, hoewel een enumeratie declaratie niet door de compiler als een fout wordt gezien. We moeten de enumeratie typenaam of alleen maar een enumeratie veldnaam gebruiken. Hieronder een module voorbeeld dat ik geschreven heb in een Excel VBA project.

```
Public Enum PersoonEnum
     Voornaam = 0
     Achternaam
End Enum

Public Persoon As PersoonEnum

Public Sub EnumTest()
     Blad1.Range("A1").Value = Achternaam
End Sub
```

Toetsen we in de directe modus in: Module1.EnumTest, dan zullen we in cel A1 de waarde 1 krijgen. Helaas werkt het niet om achter Value ‘Persoon.Achternaam’ op te geven. De compiler vindt het niet goed, terwijl de Persoon declaratie prima is. Is het een foutje in de programmeertaal zelf?!

# Unity 2D – Sprites en Game Objecten.

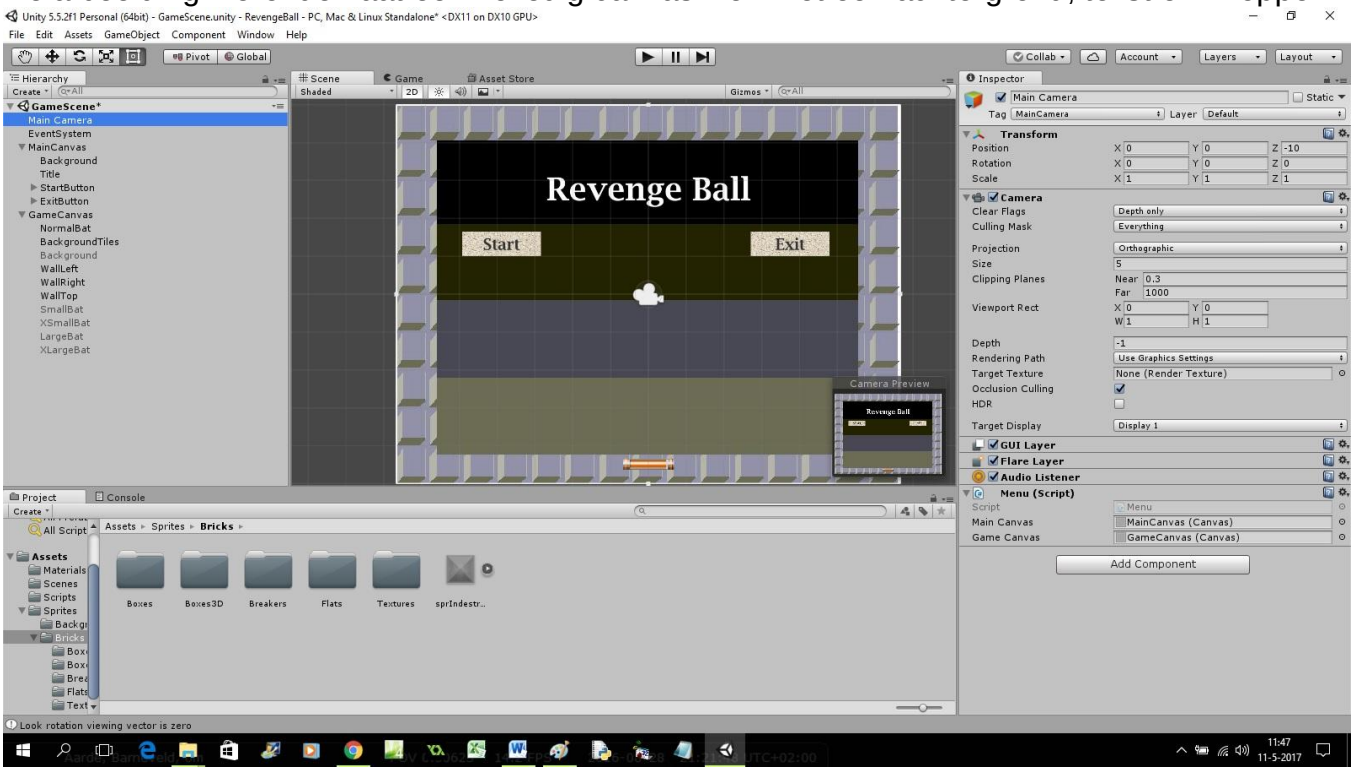
Hebben we het over sprites, dan denken we aan afbeeldingen in verschillende soorten. Het klokproject, het onderwerp uit de vorige Bulletin, kunnen we daar niet mee vergelijken. Een cube is geen sprite, maar een 3D object. Dat wil niet zeggen dat een sprite niet als een object afgebeeld kan worden. In vergelijking met Game Maker zou een sprite niets kunnen doen als we het direct op de scene zouden plaatsen. Unity maakt er echter wel een GameObject van die in de hiërarchie komt te staan. We kunnen dan componenten toevoegen, zoals we dat ook doen met 3D objecten. Toch is er een reden om niet altijd direct een sprite naar de hiërarchie te slepen. Er is een verschil tussen sprites, zoals bricks, ballen en paddles, en achtergronden en kaders. Verder in dit onderwerp laat ik zien waarom er verschillen zijn.

Voordat we met stappen gaan proberen een 2D project te maken, wil ik eerst met afbeeldingen laten zien wat het verband is tussen gebruikersprites, zoals de paddle, en een achtergrond, die we niet gebruiken maar alleen instellen.

Wat is nodig in een 2D Game project en waar moeten we op letten?

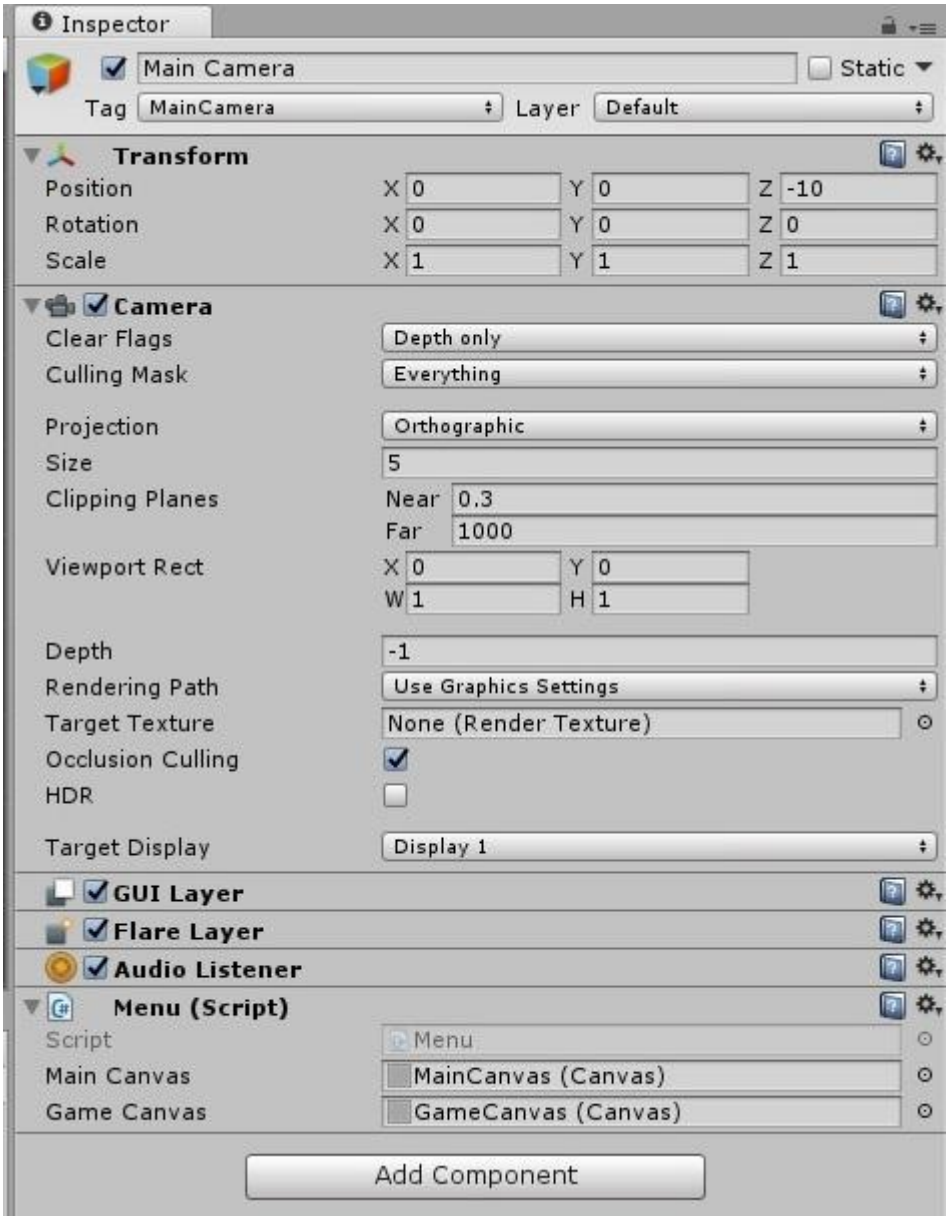
- Er is minimaal één scene nodig. We mogen met meerdere scenes werken, maar het is optioneel;
- We hoeven niet speciaal een canvas (zie meer daarover in de volgende Bulletin) in de hiërarchie te hebben, maar het is wel aanbevolen;
- In een project kunnen we meerdere canvassen gebruiken. Handig voor het weergeven van verschillende spelvensters, zoals onderstaande afbeeldingen laten zien;
- We kunnen maar één camera per scene gebruiken. Bij meerdere scenes kan het toch fout gaan als we via een script naar de volgende scene willen gaan;
- Sprites die correct uitgelijnd op een canvas geplaatst moeten worden, moeten als images worden geplaatst. Gewone sprites, zoals een paddle, kunnen direct als game objecten op de scene worden geplaatst. Deze sprites zijn geen images;
- Elk game object heeft componenten. Het is als beginner uitzoeken welke componenten voor welke game objecten geschikt zijn. Ook scripts zijn componenten die voor besturing van de game objecten zorgen. Een voorbeeld is het script in het klokproject.

De afbeelding hieronder laat een volledig canvas zien met een achtergrond, tekst en knoppen.



Dankzij de MainCamera kunnen we het canvas zien. Het canvas bevat de hele achtergrond, inclusief de titel en de knoppen. Onderaan ziet u ook een paddle, een sprite die eigenlijk niet op dit canvas hoort. De sprite mag pas gezien worden in het volgende canvas. Dat toch de paddle op het MainCanvas te zien is, komt door de editor. Tijdens het bewerken van het project kunnen sprites op de canvassen over elkaar heen liggen, maar bij het starten van het project zal de paddle echter niet te zien zijn, zie de laatste uitvoer afbeeldingen.

**Tip! Met gebruik van meerdere scenes is het probleem van de sprites, die op meerdere canvassen over elkaar heen kunnen liggen, niet van pas. Dat komt doordat elke scene zijn eigen camera heeft en ook zijn eigen canvassen kan hebben. Echter is het mij nog niet gelukt om met meerdere scenes te werken, daarom vind ik het wel even voldoende om canvassen als aparte vensters te gebruiken.**




Voordat we op de scene sprite objecten kunnen plaatsen, moet de camera goed ingesteld worden. Voor een 2D platte scherm scene hebben we een diepte instelling nodig. Wijzig de Z positie dus met -10 en stel de Clear Flags in op Depth only.
Ook de projectie van de camera moet gewijzigd worden. We kunnen voor 2D platvorm spellen geen Perspective gebruiken. Stel deze daarom in op Orthographic. Verder omlaag is er ook een Depth instelling, die moet op -1 staan.

Helemaal onderaan ziet u het component Menu (Script). Dit is het script dat ik geschreven heb om de knoppen van het menu te kunnen besturen en van canvas te kunnen wisselen. Meer over het script component ziet u in de volgende Bulletin.

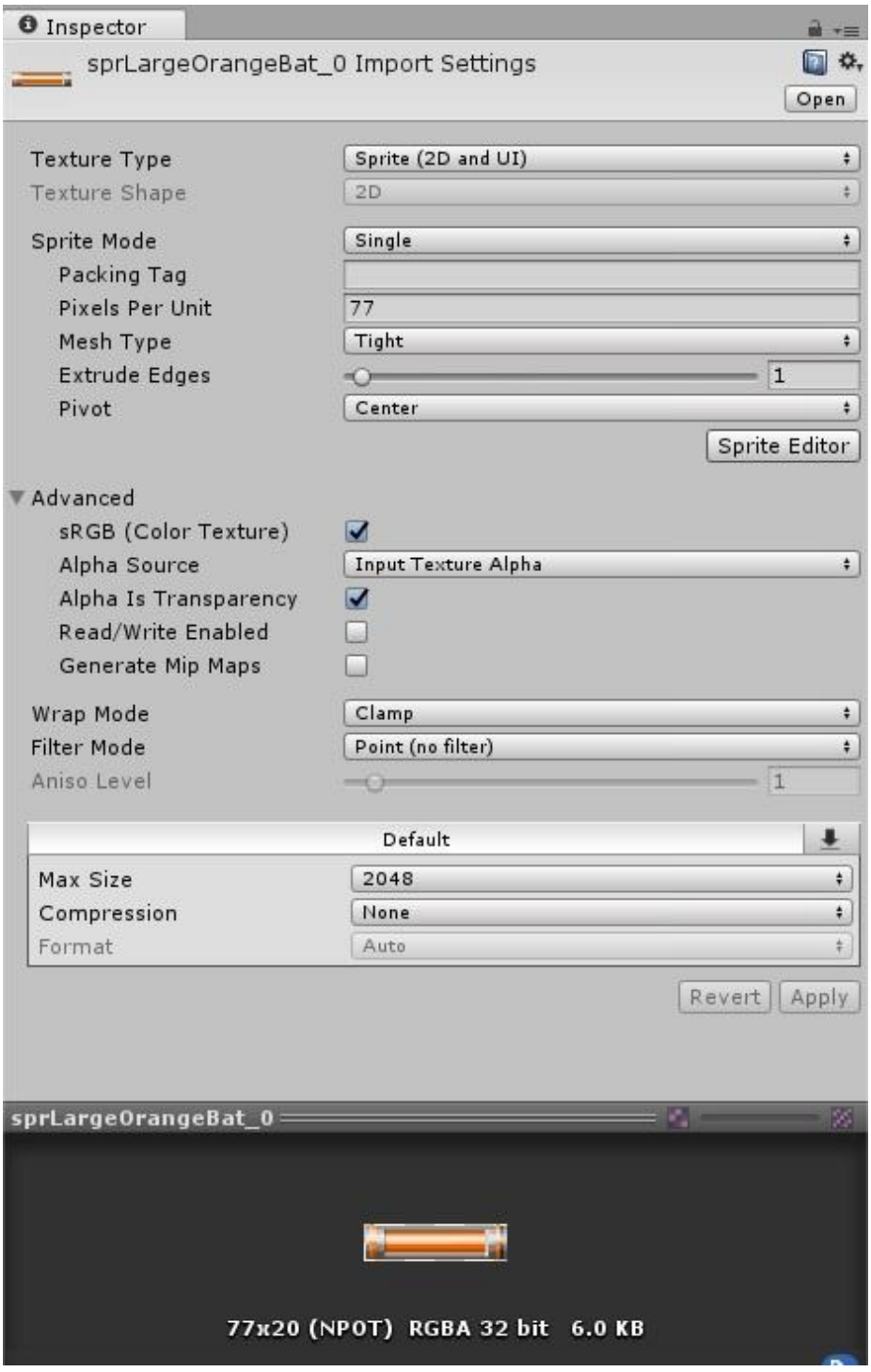
Mijn project maakt gebruik van een 1024×768 schermgrootte om er voor te zorgen dat de achtergronden precies op het canvas passen.

Het gebruik van een canvas is niet speciaal nodig, hoewel het wel wordt aanbevolen. Het is zelfs verplicht als u tekst of knoppen wilt gebruiken:

- Kiest u om tekst weer te geven, dan wordt het canvas direct aangemaakt als er nog geen canvas is.
- Kiest u om knoppen te gebruiken, dan zal het canvas ook direct aangemaakt worden als het canvas er nog niet is.
- Wilt u achtergronden correct weergeven? Een canvas hoeft dan niet per se, maar het werkt wel gemakkelijker om daarmee de achtergronden uitgelijnd op het scherm te plaatsen.

**Sprites en images**

Nu we een juist ingestelde camera hebben, kunnen we sprites in de assets toevoegen. Hebt u een sprite en staat deze in de assets, klik er dan op en zie de Inspector die er ongeveer zo uitziet als onderstaande afbeelding.




De sprite die u ziet is de paddle die op de scene te zien is. De sprite bevat alleen gegevens die ingesteld zijn zodra ze toegevoegd worden in de assetlijst. De sprites die dus in de assetlijst staan zijn geen game objecten. Dat zijn ze pas als ze gesleept zijn naar de hiërarchie, waar ze een andere Inspector krijgen.

Voordat ze als game objecten gebruikt kunnen worden, moeten ze eerst de juiste instellingen krijgen. Het kan anders niet goed gaan met het renderen van de objecten.

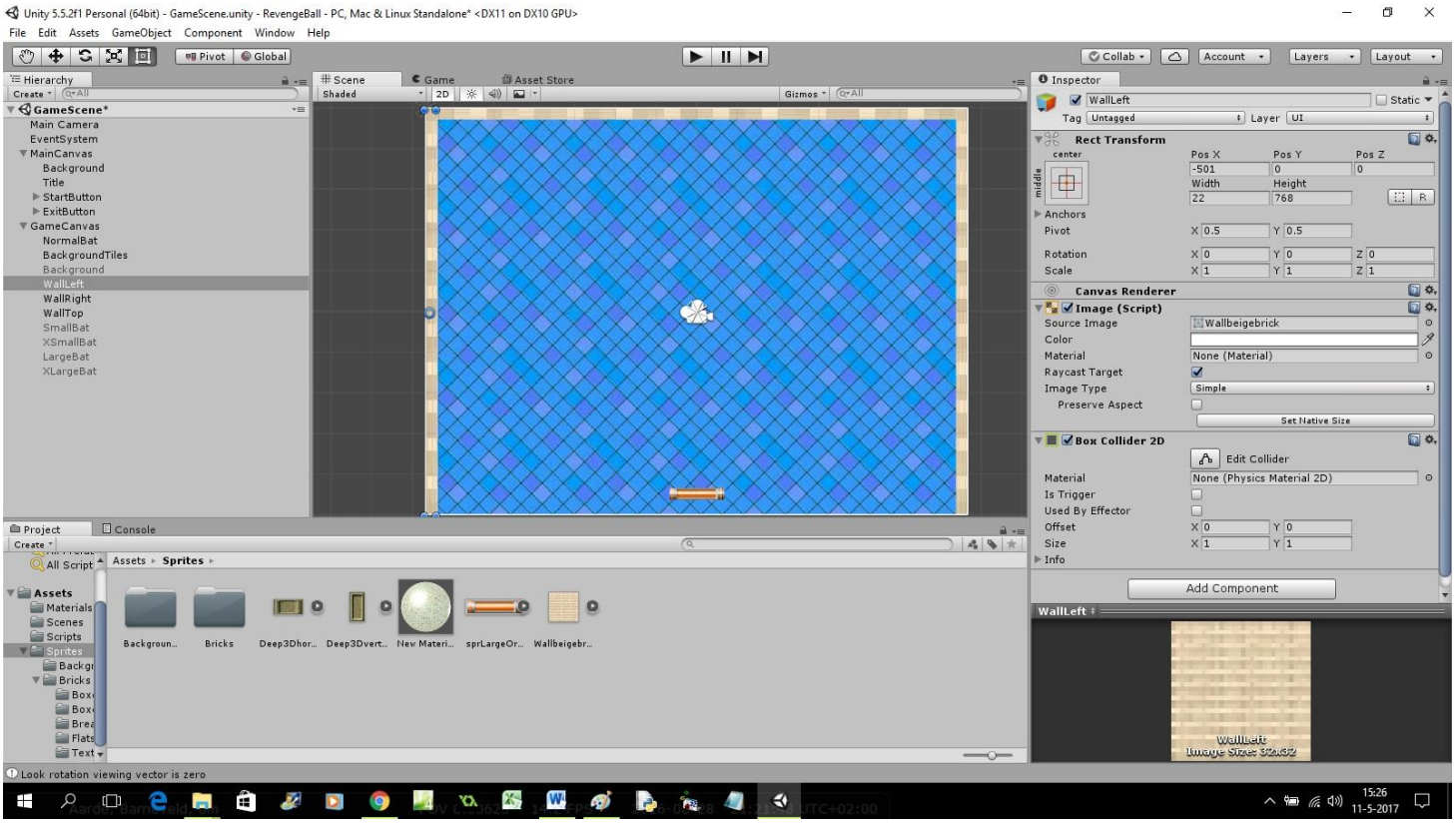
De eerste instelling waar goed op gelet moet worden is de Pixels Per Unit. Dit zorgt er voor dat de juiste dichtheid van de sprite gerendeerd wordt. Het getal 77 is precies de juiste breedte van de sprite, maar u kunt ook de instelling op 1 zetten. In de volgende Bulletin vertel ik daar meer over.

U ziet ook een instelling voor de transparantie. Indien de sprite geen transparante kleur heeft (solide is), kan het vinkje worden uitgezet. De paddle heeft echter wel een transparantie waardoor het vinkje aan moet blijven.

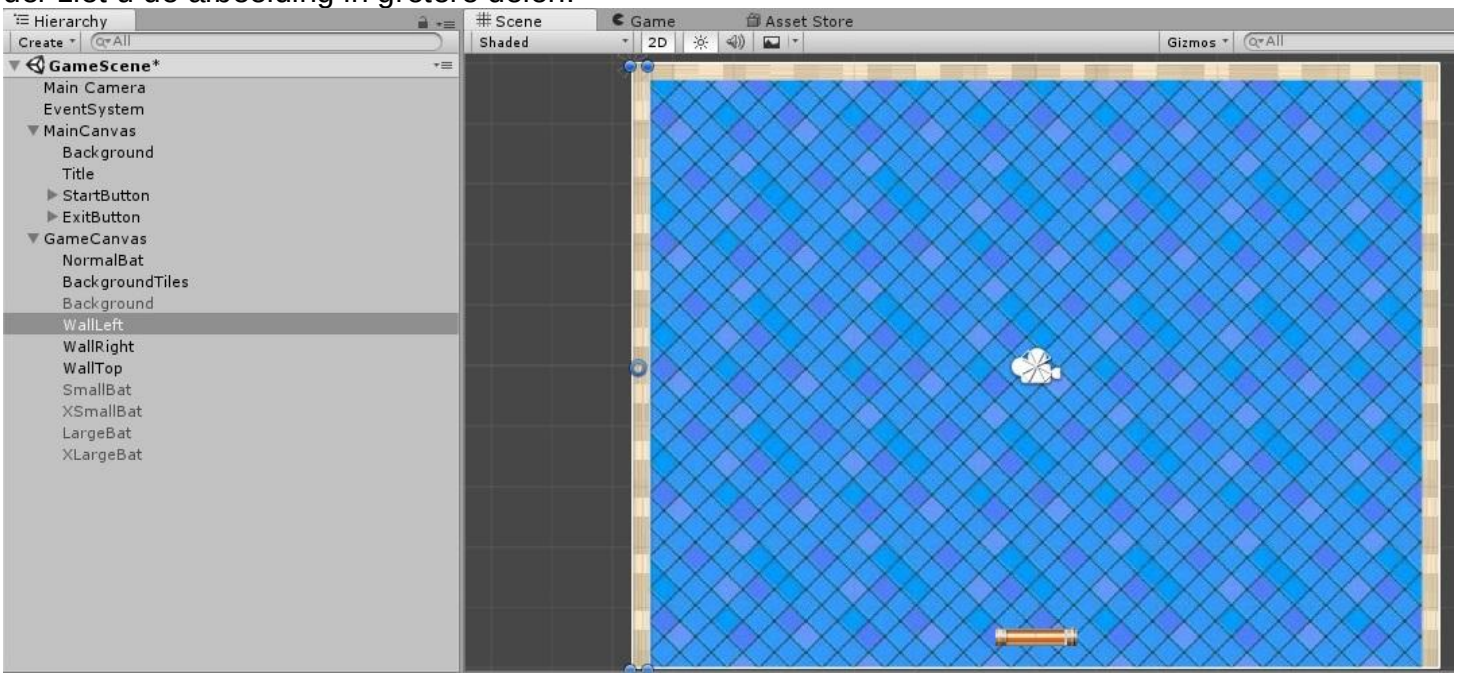
Sprites bestaan altijd uit getekende punten, daarom hoeven ze geen filter te hebben. De Filter Mode moet dan ook ingesteld worden op Point (no filter). De reden van deze

instelling is dat deze Inspectorlijst ook verschijnt voor texture instellingen, zoals voor 3D objecten. Zie bovenaan in de Inspector: Texture Type

Een sprite als game object is prima. We kunnen componenten aan het object toevoegen en met scripts het besturen. Er zijn ook sprites die maar enkele componenten nodig hebben of helemaal geen. Een voorbeeld is te zien met het GameCanvas met het WallLeft object aangeklikt.

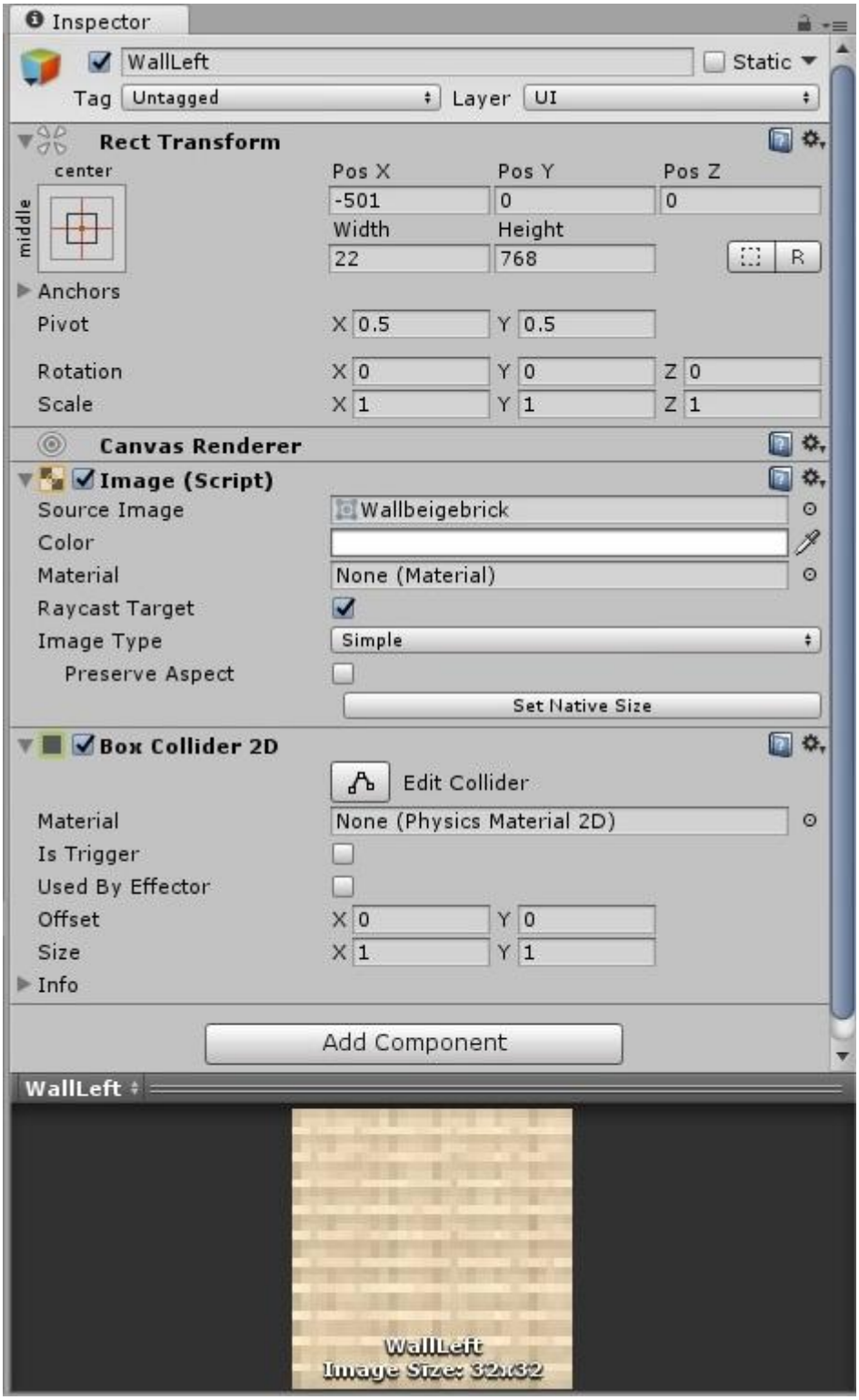



Het kader, gemaakt met WallLeft, WallRight en WallTop, zijn alleen te zien op het GameCanvas met de opgegeven achtergrond. In de Inspector kunt u al verschil zien ten opzichte van een aangeklikte sprite. Hoewel het lijkt alsof deze drie delen sprites zijn, is dat echter niet het geval. Wat u ziet zijn *images*. Images zijn speciaal geschikt om texturen op een scene of canvas netjes uit te lijnen. Hieronder ziet u de afbeelding in grotere delen.



Standaard zijn images witte plaatjes in verschillende vormen. Elke soort sprite, die in de assetlijst staat, mag als textuur aan een image worden toegevoegd, zoals u hieronder de Inspector ziet van WallLeft.

Het Rect Transform is een component dat sprites niet hebben. Hiermee kunt u het image precies uitlijnen zoals u het wilt hebben. Rect Transform zorgt er ook voor dat u met de muis op de scene kunt slepen naar de juiste positie. Het zal nog gemakkelijker gaan wanneer dit wordt gedaan op een canvas, omdat de lijnen aan de grepen langs de kant op de rand van het canvas vallen, zodat er minder kans is dat het image er buiten valt. Met sprites kan dit niet correct worden uitgelijnd wegens het ontbreken van Rect Transform. Zoals ik hierboven al heb gezegd, is WallLeft geen sprite. De andere twee, WallRight en WallTop zult u daarom met WallLeft niet in de assetlijst zien staan.

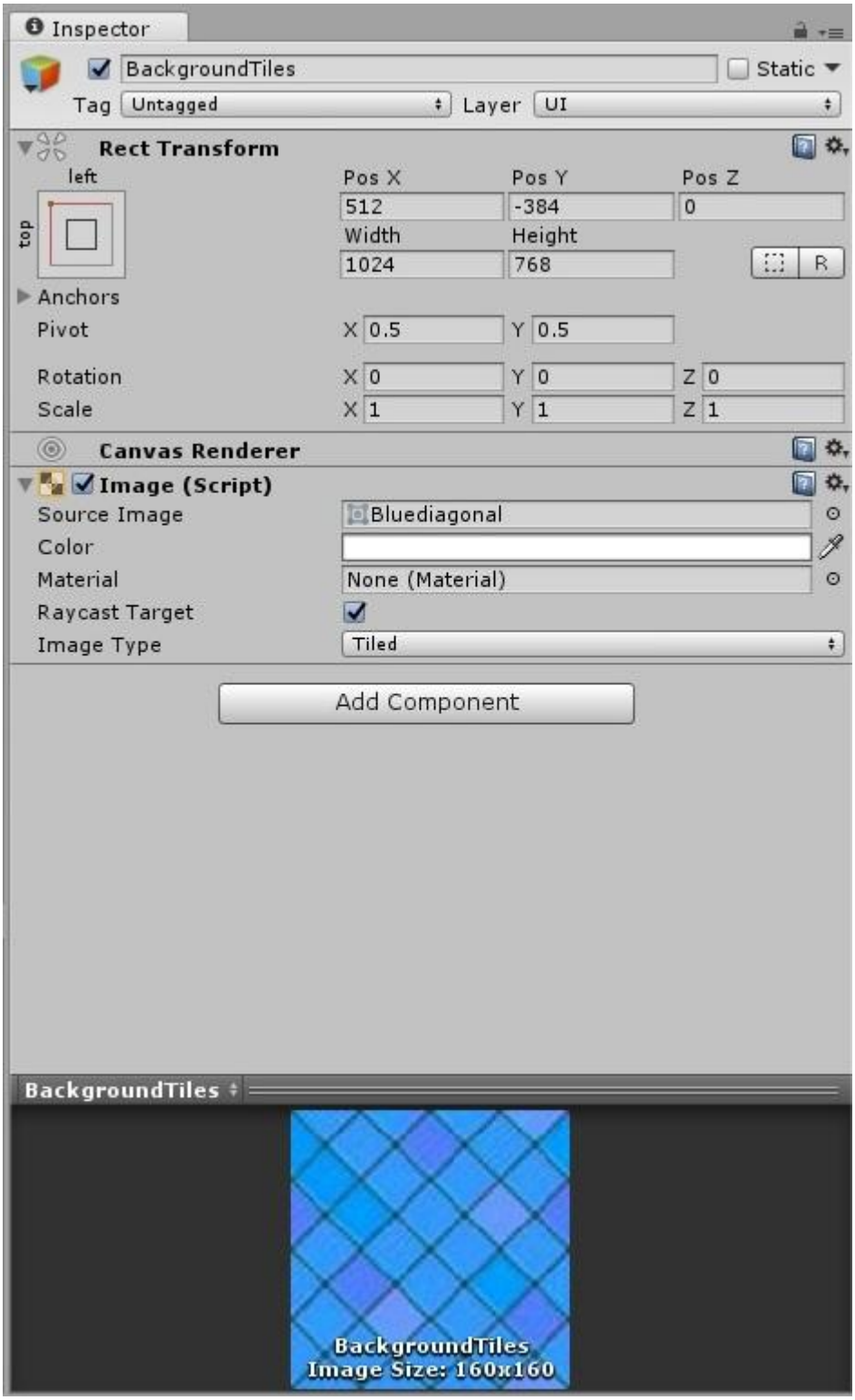



Het image zelf heeft een script met instellingen voor het Source Image, de Color, het Material en vooral wat voor Image Type het moet zijn. Op de afbeelding ziet u Wallbeigebrick. De sprite gebruik ik eigenlijk voor een brick in het spel, maar als textuur voor het kader vond ik ook wel geinig.

Gebruikt u geen sprite textuur, dan kunt u eventueel een kleur kiezen bij Color. Ook een Material kan worden gekozen als u in de assetlijst materialen hebt. Materialen komen voorlopig nog niet aan de orde.

**Sprites voor achtergronden**

Sprites die u voor achtergronden wilt gebruiken, kunnen niet als sprite game objecten op het canvas worden geplaatst. De sprite moet uitgelijnd worden over het hele canvas met een image. De blauwe diagonale achtergrond staat in het game object BackgroundTiles. Hieronder ziet u de afbeelding van het game object met de Inspector.




Eerder zei ik dat achtergronden niet altijd de grootte van het hele canvas hebben. De sprite Bluediagonal ook niet. Toch moet deze op het hele canvas worden geplaatst. Met alleen de sprite was het geen oplossing geweest, want dan had ik de sprite moeten schalen op de grootte van het canvas, waardoor de sprite er niet meer glad uit zal zien.
Dankzij het image kan de sprite als een textuur *getiled*, of ook wel gezegd *betegeld*, worden. Door het image op de juiste positie van het canvas te plaatsen met de grootte ervan, kunnen er met allerlei soorten sprites tiled achtergronden gemaakt worden.

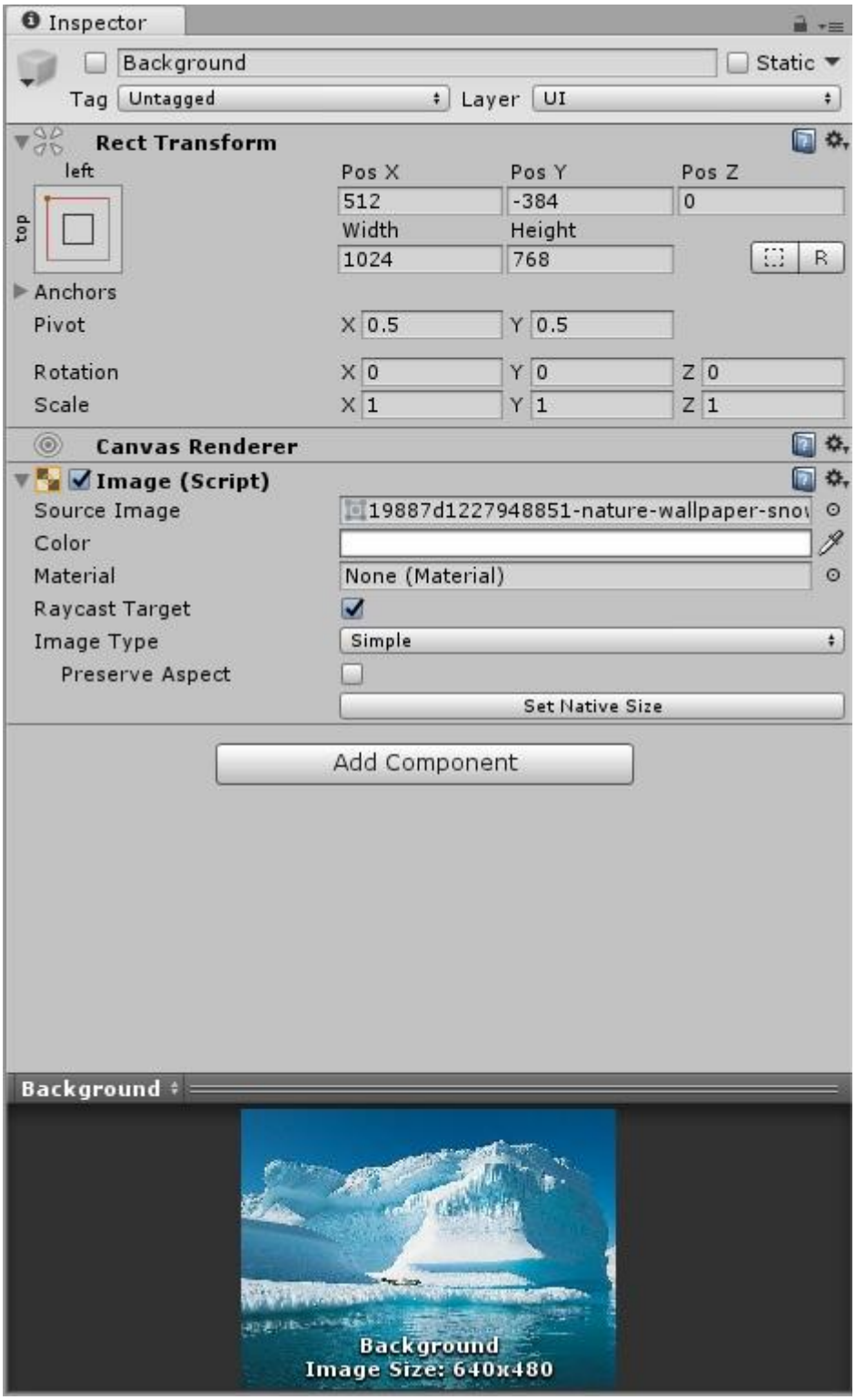
Onderaan ziet u hoe groot de sprite is. Ook al zou een 160×160 sprite niet goed betegeld worden, doordat het een sprite is met diagonale lijnen is dat niet erg en is het zeer geschikt om als 2D achtergrond te gebruiken.

Als voorbeeld ziet u hier een achtergrond met een sprite die niet betegeld wordt. Beide achtergronden worden linksboven uitgelijnd, zodat het goed in het canvas komt. Bovendien weten we dan ook wat de positie is van linksboven. Daardoor is het wat gemakkelijker de posities van de bricks te bepalen.

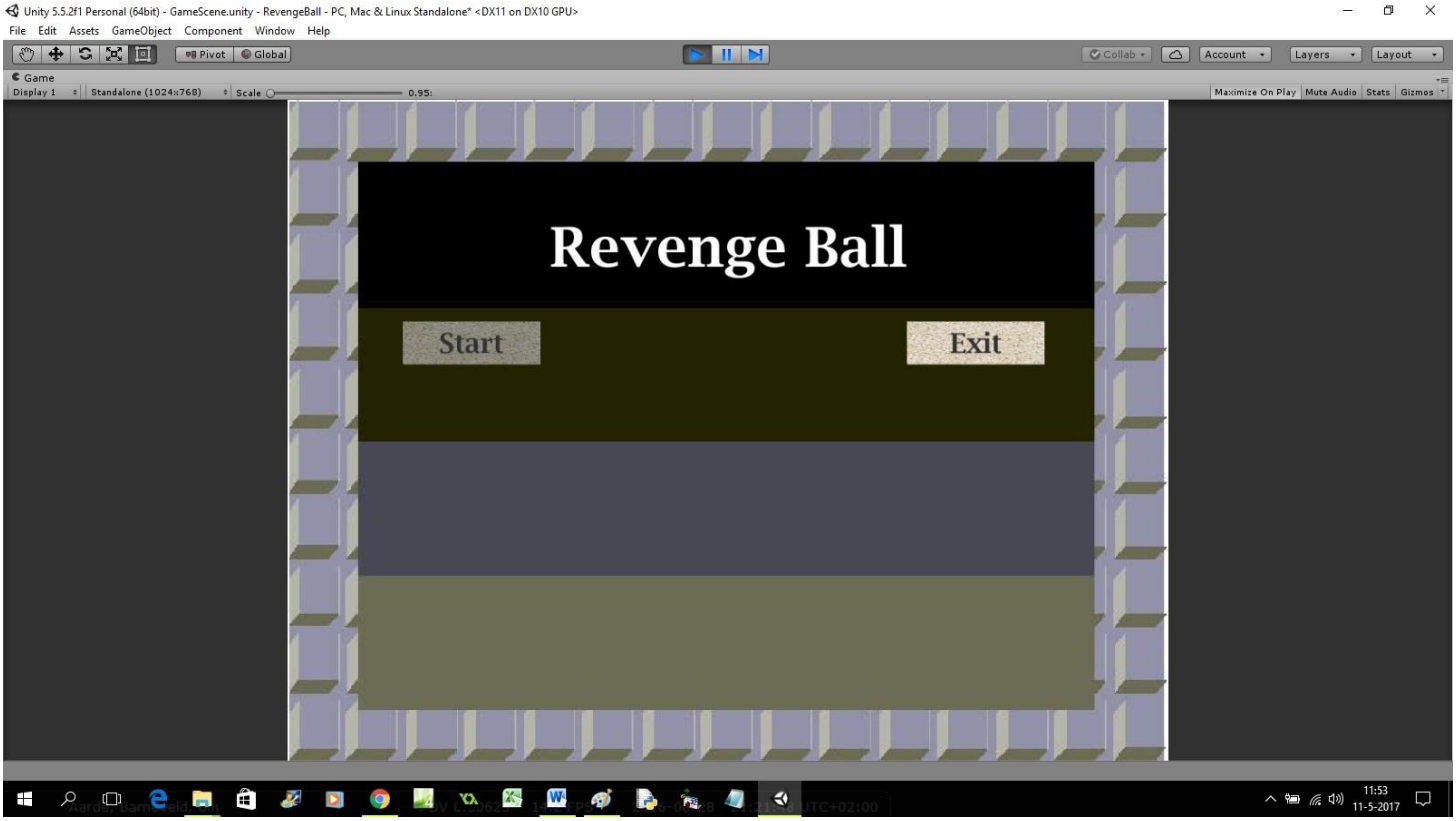
De sprite heeft niet de grootte van het canvas. De grootte is meer dan de helft van het canvas, waardoor het volledig schalen van het canvas nog kan. Bij kleinere sprites zal de weergave juist verslechteren en is het beter ze te tilen of ze niet te gebruiken.

Vergeet niet bij deze grote sprites het tilen uit te zetten, door het Image Type op Simple te zetten.

Hieronder ziet u de canvassen nogmaals in uitvoer. Merk nu op dat de paddle in het MainCanvas er niet is.

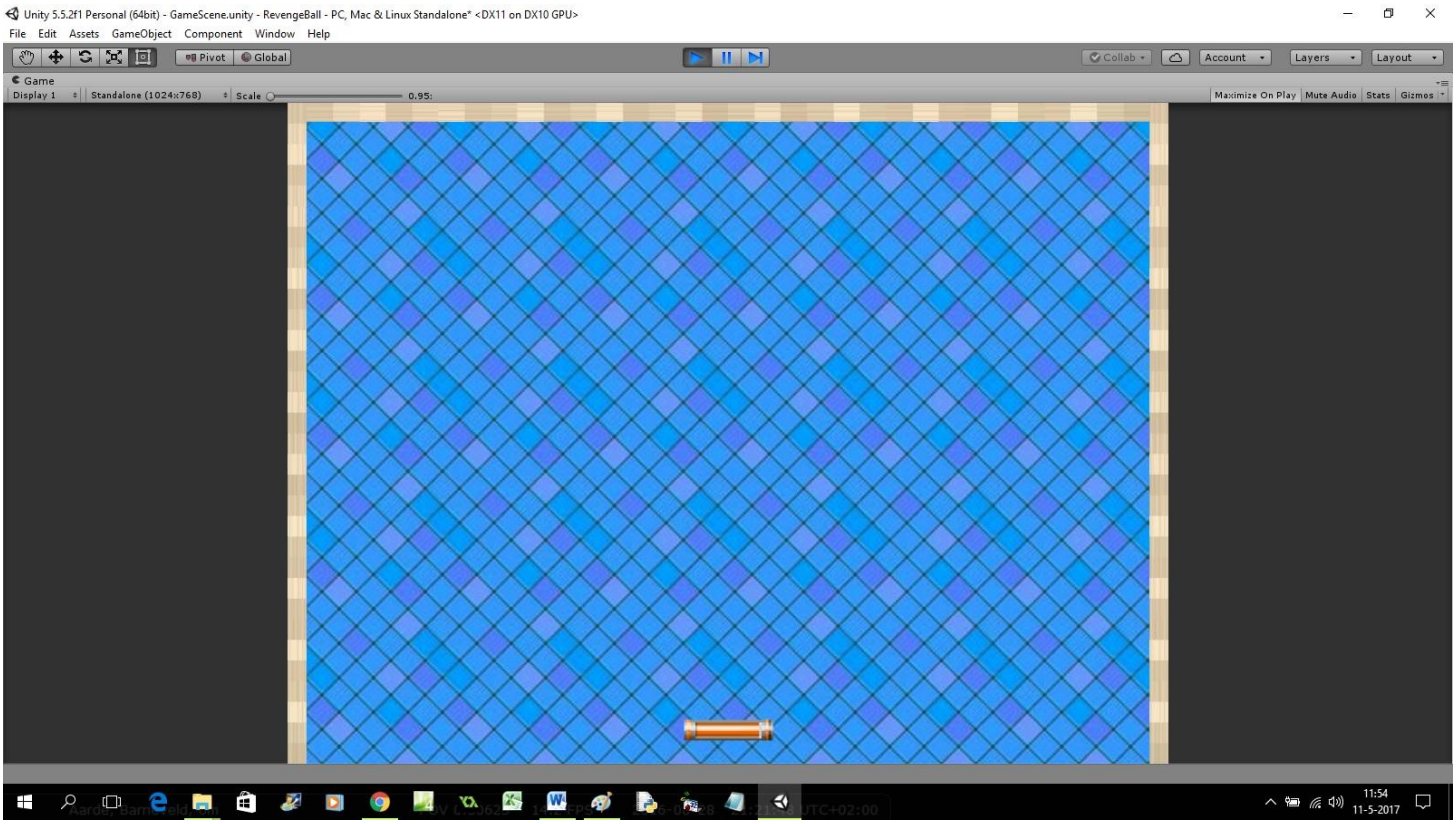


## Uitvoer MainCanvas




Merk op dat de knop Start donker van kleur is. Hiermee wil ik laten zien dat de knop van kleur verandert zodra de muis eroverheen gaat. In Bulletin nummer 1 volgend jaar kom ik terug op dit onderwerp en laat ik de Inspector zien van de knoppen en leg ik het wisselen met deze twee canvassen uit. Hieronder zal dus het GameCanvas verschijnen zodra er op de knop Start geklikt wordt.

## Uitvoer GameCanvas



# Visual Studio – Variabelen als objecten.

Variabelen die we declareren van het type Integer of een ander standaard type, werken enkel om waarden vast te houden. We hebben functies nodig om nog wat meer met de variabelen te kunnen doen. Zo willen we wel eens de lengte van de inhoud van een stringvariabele weten of willen we een getal, dat in een stringvariabele is opgeslagen, converteren naar een numerieke waarde, zodat we het getal kunnen gebruiken met een numerieke variabele. In de .NET Framework Library bestaan deze functies als methoden van het System.Object. Variabelen die we declareren kunnen van die methoden gebruik maken. Laten we eerst kijken hoe dat ging in de oude Visual Studio versies in de Basic code.

Een declaratie als:

```
Dim I As Integer, S As String
```

zegt van een variabele die alleen numerieke waarden kan aannemen en een variabele die alleen alfanumerieke waarden kan aannemen. Alfanumerieke waarden wil dus zeggen: alles wat in een string opgeslagen kan worden, dus niet alleen tekst maar ook getallen en symbolen.

Pas na de declaratie kunnen we waarden toekennen. Het is niet toegestaan in een declaratie direct waarden toe te kennen, wat nu in de nieuwe Studio versies wel kan.

Een veelvoorkomend probleem, waar zelfs de converteerwizard (later daar meer over) verward van kan raken, is het oude Variant type. Dit type zorgt ervoor dat elke soort waarde aan de variabele toegekend kan worden, of het nou getallen zijn of strings.

Visual Basic 6 gebruikers zijn een declaratie als hieronder gewend om het in één regel te schrijven, dat echter niet op dezelfde manier werkt.

```
'aparte regels
Dim I As Integer
Dim J As Integer
Dim K As Integer

'een regel
Dim I, J, K As Integer   'werkt niet zoals de aparte regels werken
```

In de aparte regels worden de drie variabelen gedeclareerd van het type Integer. Dat gebeurt echter niet in de één regel. Alleen variabele K wordt gedeclareerd van het type Integer, maar de variabelen I en J niet. Visual Basic 6 beschouwt het als Variant variabelen, omdat er geen types staan. De oplossing is gebruik te maken van de aparte regels, maar het kan alsnog in één regel, zoals hieronder:

```
Dim I As Integer, J As Integer, K As Integer
```

Natuurlijk vind ik het beter om toch elke declaratie apart te schrijven.

**Hulpmiddelen voor gebruik van de variabelen.**
Visual Basic kent functies om met variabelen te kunnen werken. Om bijvoorbeeld de lengte van een string te weten, kunnen we onderstaande regel gebruiken:

```
L = Len(S)
```

Het converteren van een string naar een Integer kan op twee manieren:

```
I = Val(S)
I = CInt(S)
```

Beide hebben echter een nadeel. De Val() functie geeft een 0 terug als de waarde van S alfanumeriek is, maar geeft ook een 0 als het wel numeriek is en een 0 waarde bevat. De functie CInt() veroorzaakt een foutmelding als de waarde van S geen numerieke stringwaarde heeft, bijvoorbeeld een letter.

Nu we het .NET Framework kunnen gebruiken om hulp te vragen voor de variabelen; wat is de waarde in numerieke weergave, wat is de lengte, is het een gehele waarde, enzovoort, dan nog zullen de problemen die in berekeningen kunnen ontstaan niet honderd procent door de System.Object methoden verholpen kunnen worden. Gelukkig hoeven we dankzij de methoden niet meer de globale oude Basic functies te gebruiken.

**De converteerwizard**
Wie moeite heeft om oude programma's in de .NET versie om te zetten, kan gebruik maken van de converteerwizard. De wizard controleert het hele project en vergelijkt of het een en ander verouderd is en vernieuwd moet worden. De wizard doet dit correct, op één ding na niet: het vernieuwen van de declaratieregels en expressies blijven zoals ze eerder waren gemaakt en ook het toekennen en creëren van waarden en dimensioneren van de objectinstanties verlopen niet soepel. Oude versies kennen geen klassenstructuren en daarom ook geen klasseninstantie constructors. De constructor is daarom ook de reden dat nu initialisatiewaarden direct toegekend worden zodra een object gemaakt wordt. Omdat een Variant type nu obsoleet is, kan de wizard niet weten dat nu een één regel declaratie, zoals hierboven, nu wel correct zal werken. Maar de programmeur had het waarschijnlijk niet zo gewild en had deze variabele liever voor allerlei waardensoorten willen gebruiken dat nu niet zal gebeuren. Conclusie: de programmeur zal toch een deel ervan handmatig op moeten lossen.

**Hoe is het verband met variabelen en objecten en hoe werkt het nu?**
Declareren we een variabele in de nieuwe versie, dan zal er een groene streep eronder verschijnen. Nu een variabele ook een object is, verwacht Visual Basic .NET direct een initialisatiewaarde voor de constructor. Het is niet verplicht om direct een waarde er aan toe te kennen. Het mag ook later. Let wel op dat u eerst een waarde er aan toekent voordat u ermee een controle uit gaat voeren.

Onderstaande code laat een voorbeeld zien hoe we numerieke waarden in een expressie kunnen weergeven en kunnen toekennen aan een tekstvenster.

```
Dim Getal As Integer = 50
Tekst1.Text = "Het getal heeft de waarde " & Getal.ToString()
```

In Visual Studio .NET is er voor het samenvoegen van alfanumerieke waarden een andere operator bij gekomen: de ampersand. De plus operator mag nu alleen nog maar worden gebruikt voor numerieke waarden. Eerder hadden we één van de hulpfuncties van Visual Basic nodig, maar dankzij de library kunnen we nu de methode ToString() erven van System.Object zodat de numerieke waarde correct als alfanumerieke waarde weergegeven wordt.

Kunnen we nu zeggen: variabelen bestaan niet meer? Gelukkig niet, want we kunnen niet in een variabele kijken. Het is daarom niet te vergelijken met een klasse die door een programmeur zelf geschreven wordt. Denk maar aan een gebruikerstype. Die maken we, maar een variabele maken we niet. Die declareren we alleen. Het verband is nu dat we moeten leren om te gaan met het Framework, omdat alles wat we maken daarvandaan komt en niet meer de wielen opnieuw uitgevonden hoeven te worden. Stel dat we 15 variabelen zouden gebruiken en van het bovenstaande de tweede regel gebruiken met 15 tekstvensters. Gebruiken we dan ook 15 ToString() functies? Het lijkt van wel, maar dat is niet zo. Vroeger werd dan 15 keer dezelfde Str$() functie gebruikt, maar nu maar 1 keer ToString(), ook al zou je 20 variabelen hebben. Er bestaat maar één ToString() functiemethode die bij elke aanroep door een objectinstantie uit het geheugen geërfd en aangeroepen wordt. De objectinstanties zelf hebben deze methoden niet. Denk maar aan de takken in een boom die aan de stam groeien. Geen één blad heeft zijn eigen voedsel. Alles komt uit één stam en dat werkt nu precies zo met het .NET Framework library.

**Tip!** Zoals ik hierboven het verteld heb, lijkt het erop dat er maar één ToString() methode bestaat, omdat die geërfd wordt. Maar er zijn nog steeds verschillende types, niet alleen een integer, en omdat System.Object niet weet om welk type het gaat, heeft elk type de geërfde ToString() methode van System.Object, maar zal de ToString() *overriden* zijn, dat wil zeggen *overschreven* worden, zodat het vanuit het juiste type naar een string geconverteerd kan worden. Hebben we dus drie Integer variabelen en drie Double variabelen, dan zullen er maar twee overriden ToString() methoden zijn, omdat de Integer zijn eigen overriden ToString() heeft en de Double ook.

**Expressies als objecten**

Willen we een berekening uitvoeren in een tekstvenster? Als het maar nodig is voor tijdelijk, dan heeft het geen zin om een variabele te declareren voor de expressie. Ook expressies werken als instanties, maar dat kan alleen als een expressie tussen haakjes staat. .NET ziet de expressie alleen als een objectinstantie als het tussen haakjes staat. Dat komt doordat de haakjes de hoogste prioriteit hebben in expressies.

In plaats van: `Tekst1.Text = Str$(20 + 40 * 8)`

Kunnen we nu typen: `Tekst1.Text = (20 + 40 * 8).ToString()`

Het omzetten van het antwoord van de expressie naar een integerwaarde is wat lastiger. Er is geen ToInt() functie aanwezig, zodat we een converteer mogelijkheid moeten gebruiken. Onderstaande declaratie definieert variabele Getal met als waarde het resultaat van de expressie in Tekst1.

```
Dim Getal As Integer = CType(Tekst1.Text, Integer)
```

Het sleutelwoord mag voor alle types worden gebruikt, dus ook in plaats van ToString(). Houd er wel rekening mee dat CType() geen berekeningen in strings kan omzetten naar numerieke waarden. Bovendien kent Visual Basic niet de functie Eval() zoals andere Basic dialecten het wel kennen om complete expressies, die in strings staan, uit te laten rekenen. Een handige evaluatiefunctie waar CType() niet tegenop kan.

Toch heeft .NET een Conversion object met alle hulpfuncties die globaal in de oudere versies werden gebruikt. Helaas is het een module en geen type. Het staat dus los van de variabelenobjecten.

Een ander hulpmiddel is wel een converteerklasse. De methoden zijn onderdeel van de basisklasse en worden dus geërfd. De klasse heet Convert en u kunt hem gebruiken zoals hieronder:

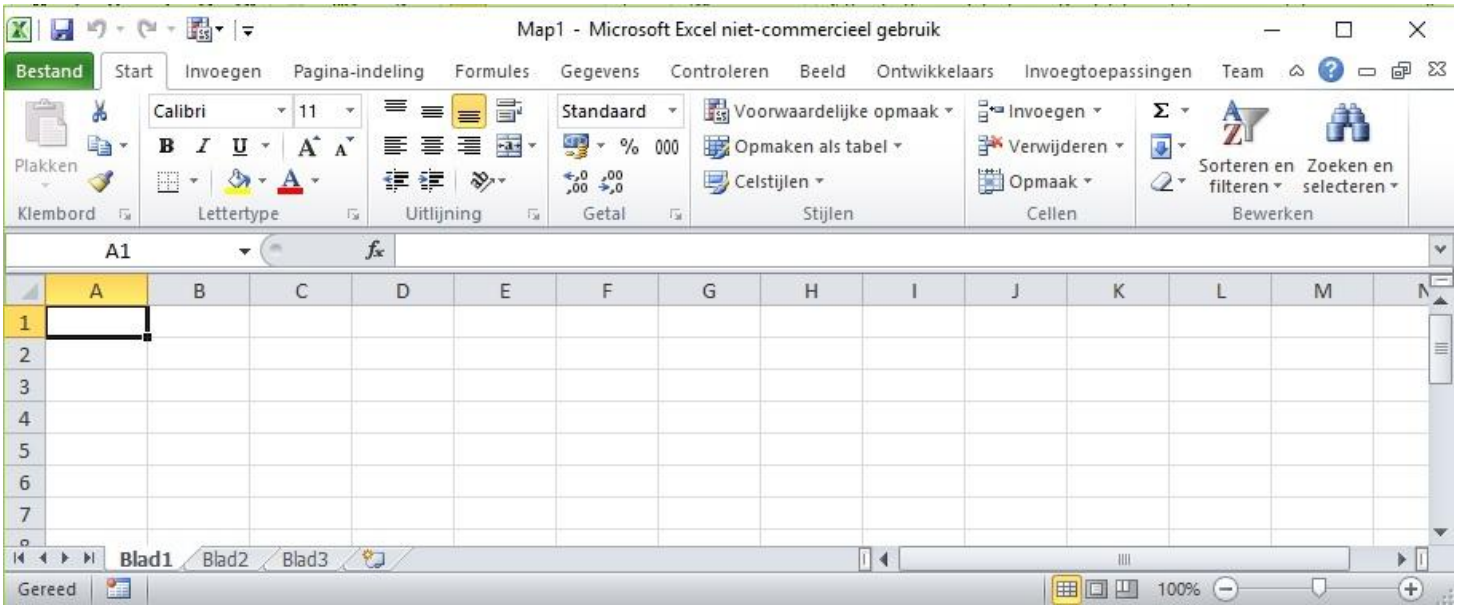
```
Getal = Convert.ToInt32(Tekst1.Text)
```

Omdat Convert direct de waarde omzet naar een 32 bit Integer, mag u er zelfs een expressie van maken. Door achter de regel bijvoorbeeld + 20 te typen, zal meteen ook die waarde erbij opgeteld worden:

```
Getal = Convert.ToInt32(Tekst1.Text) + 20
```

# Leren programmeren met Excel VBA.

Excel is een handig flexibel programma om alles ermee te kunnen doen wat met administratie te maken heeft. Lijsten, draaitabellen en grafieken zijn goede hulpmiddelen voor het onderhouden van gegevens.




Excel is niet alleen een administratief programma. Het kan uitgebreid worden naar uw wensen. Het lijkt daarom op een IDE met een project codevenster. Elke cel is een object waar een waarde met opmaak ingegeven kan worden. De cellen kunnen met expressies en structuren beheerd worden. Elke waarde die u in een cel invoert wordt met het celtype of getal notatie bepaald. We kunnen Excel dus een soort compiler noemen om de werkbladen goed te laten controleren.

Cellen worden gestructureerd in werkbladen. Op het eerste gezicht zou je denken aan tabellen, en ze zijn ook als tabellen te gebruiken. Een Excel werkblad kan ook hiërarchisch werken en de werkbladen kunnen zelfs samenwerken. Het is zelfs mogelijk meerdere werkmappen samen te gebruiken.

Excel kent, wat rekenkundig betreft, veel mogelijkheden. Links naast de formulebalk (ook wel genoemd invoerbalk) is er een categorielijst waar u vele numerieke en alfanumerieke functies kunt vinden.
De cellen kunnen samenwerken. Het is daardoor mogelijk jaarverslagen te schrijven met de maandtotalen die in de andere werkbladen staan. Excel heeft daardoor vele voordelen, maar helaas ook nadelen, die gelukkig in VBA op te lossen zijn. Hieronder een aantal voordelen en nadelen.

- Cellen zijn onder te brengen in een hiërarchische structuur. De lengte en breedte van een cel hoeft daardoor niets te maken te hebben met een lengte en breedte van een andere cel.
- Cellen kunnen verwijzen naar andere cellen. Rekenkundige lijsten zullen daardoor automatisch aangepast worden, mocht een waarde waar een cel naar verwijst gewijzigd worden.
- Functies wijzigen het celbereik automatisch, mocht er een kolom of rij binnen het bereik ingevoegd of verwijderd worden. Niet alle rekenkundige programma's hebben die mogelijkheid.

- Cellen kunnen niet met zichzelf rekenen. Een som als =A1+20 in cel A1 veroorzaakt een kringverwijzingsfout.
- Geselecteerde cellen kunnen niet genest worden. Het is daardoor niet mogelijk een bereik binnen een bereik uit te voeren, wat normaal gesproken in programmeertalen wel kan. Ook in VBA is dit te verhelpen.
- U kunt geen toekenningen uitvoeren in formules, alleen expressies. Dat komt doordat de formules altijd eenregelig zijn. Ook dit veroorzaakt een kringverwijzingsfout. Zie hieronder de ALS

functie die geen juiste onwaar clausule heeft:
=ALS(A1=10;A1+20;A2=A1)
Toch hoeft het niet speciaal een kringverwijzingsfout te zijn. Indien cel A2 niet de actieve cel is, wordt cel A2 vergeleken met cel A1 en niet toegekend.

U ziet dus, we kunnen niet altijd in Excel flexibiel te werk gaan. Als we meer willen doen dan op de voorgrond kan, dan is het gebruik van macro's of VBA projecten een goede keuze.

**Macro's gebruiken**
Hebben we op een werkblad vaker eenzelfde formule nodig, dan kan het handig zijn deze op te nemen in een macro. Een macro is een subroutine die code bevat die gedaan is tijdens de opname.

Macro's worden vaak gebruikt om werk te besparen, maar ze zijn helaas niet altijd helpvol. Macro's hebben nadelen:

- Er worden nauwkeurige bewerkingen opgeslagen, zelfs het klikken op een cel. Een cel selectief maken kan handig zijn, maar voor alleen maar een waarde in een cel te typen heeft het geen nut.
- Alles wordt opgenomen, maar niet de laatste regel die daarna ongedaan is gemaakt. Het is niet te zeggen of dit nou handig is of niet.
- Een macro opnemen en stoppen is te vergelijken met een OV kaart. Check hem in en check hem uit. Vergeet u het laatste, dan krijgt u een macrosubroutine die reuzegroot wordt.
- Macro's zijn altijd globaal. Ze worden geplaatst in een module. De macro's hebben ook geen argumenten, omdat alles globaal in de subroutine opgenomen wordt.
- Macro's veroorzaken overtollige rommel. Hoe langer een macro opgenomen wordt, hoe meer rommel erin komt. Rommel is bijvoorbeeld onzinnige selecties en indirecte berekeningen. Macro's structureren zichzelf ook niet, waardoor er een grote kans is dat er dubbele code ontstaat.

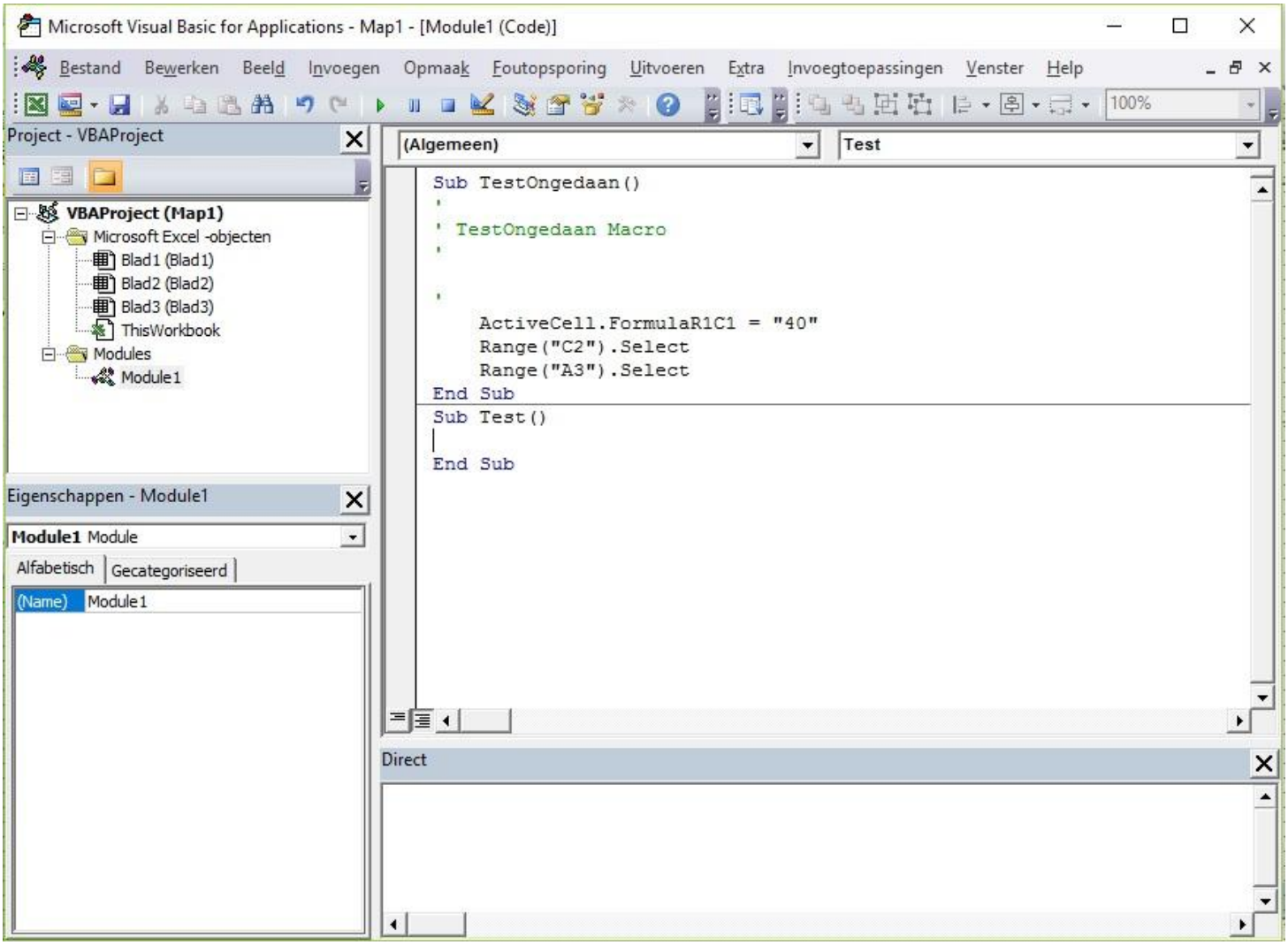
Het is wel mogelijk om een macro in de juiste map op te nemen. Die kunt u dan kiezen. Helaas blijven macro's globaal en als u ze niet in de modules wilt hebben, kunt u ze altijd nog later verplaatsen naar de juiste werkbladen.

**VBA leren, de achtergrond van Excel**
De komende Bulletins ga ik veel uitleg geven over VBA. VBA is niet zomaar een Basic dialect. Het is Basic met een Excel library. De COM library heeft veel mogelijkheden die ook op de voorgrond werken, maar de code maakt het programmeren van de werkbladen veel flexibeler. Bovendien kunt u complete structurele projecten maken.

VBA heeft geen mogelijkheid om een Excel VBA project in een EXE formaat uit te kunnen laten voeren. U kunt een project alleen als een Excel document compileren en in Excel uitvoeren. Vandaar ook de naam Visual Basic for Applications.

Indien u in de Excel tabbladen geen Ontwikkelaars ziet staan, ga dan naar tabblad Bestand en klik op Opties. Klik in de lijst aan de linkerkant op Lint aanpassen en vink het vakje Ontwikkelaars aan.

Hieronder ziet u een afbeelding van een VBA project.

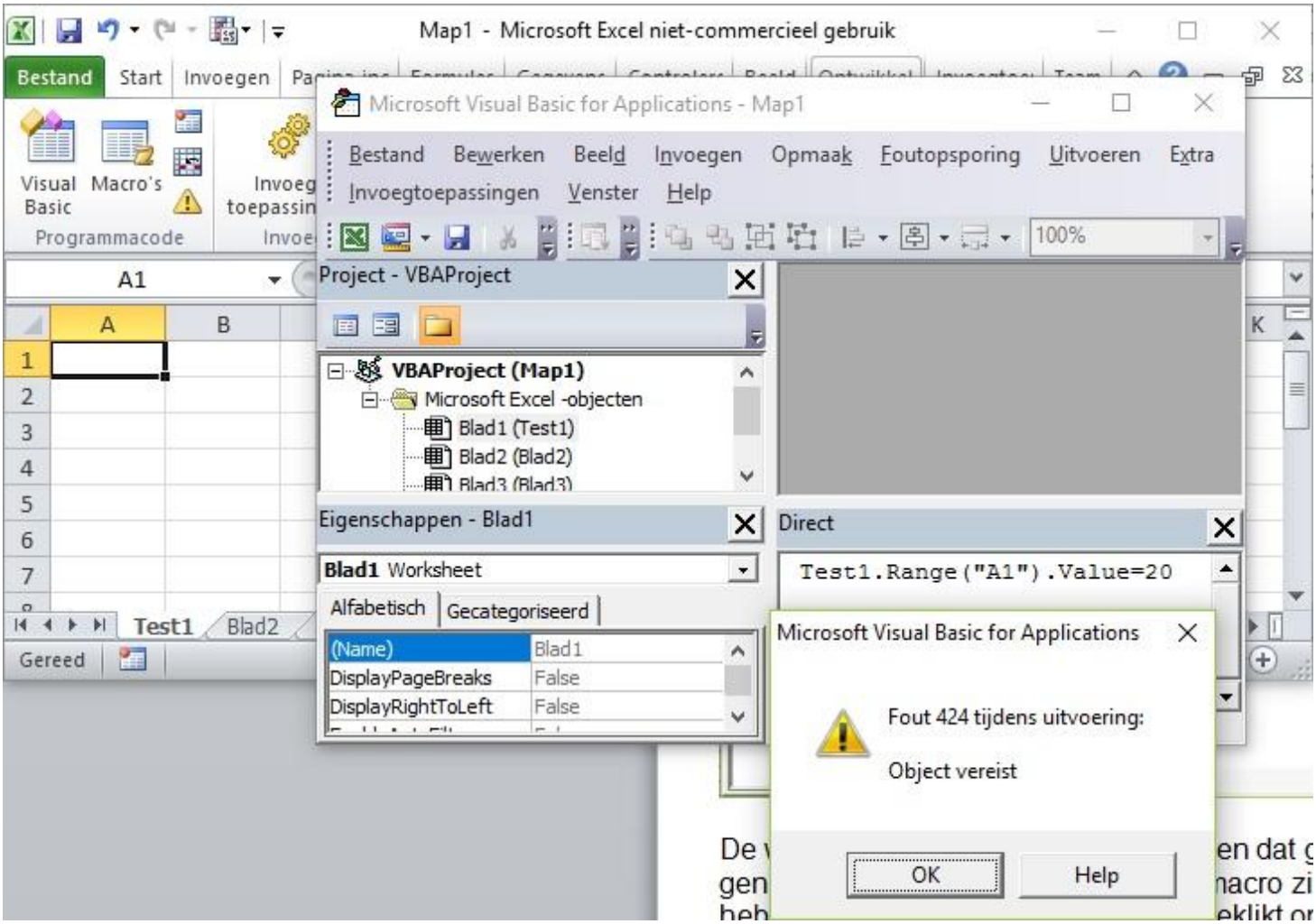



De werkbladen werken lokaal per map en dat geldt ook voor de modules. In Module 1 ziet u twee opgenomen macro's. Wat u in de eerste macro ziet is echter niet zo gegaan op de voorgrond. In cel A1 heb ik het getal 40 ingetoetst, daarna geklikt op cel C2 en daar nog een getal ingetoetst. Die heb ik ongedaan gemaakt en voordat ik de opname stop heb gezet, klikte ik nog op cel A3. Wat er tussen de twee Range methoden is gebeurd, is niet meer te achterhalen. De conclusie is dus dat macro's alles opnemen, maar weer de laatste opdracht verwijderen omdat het ongedaan maken meteen uitgevoerd wordt.

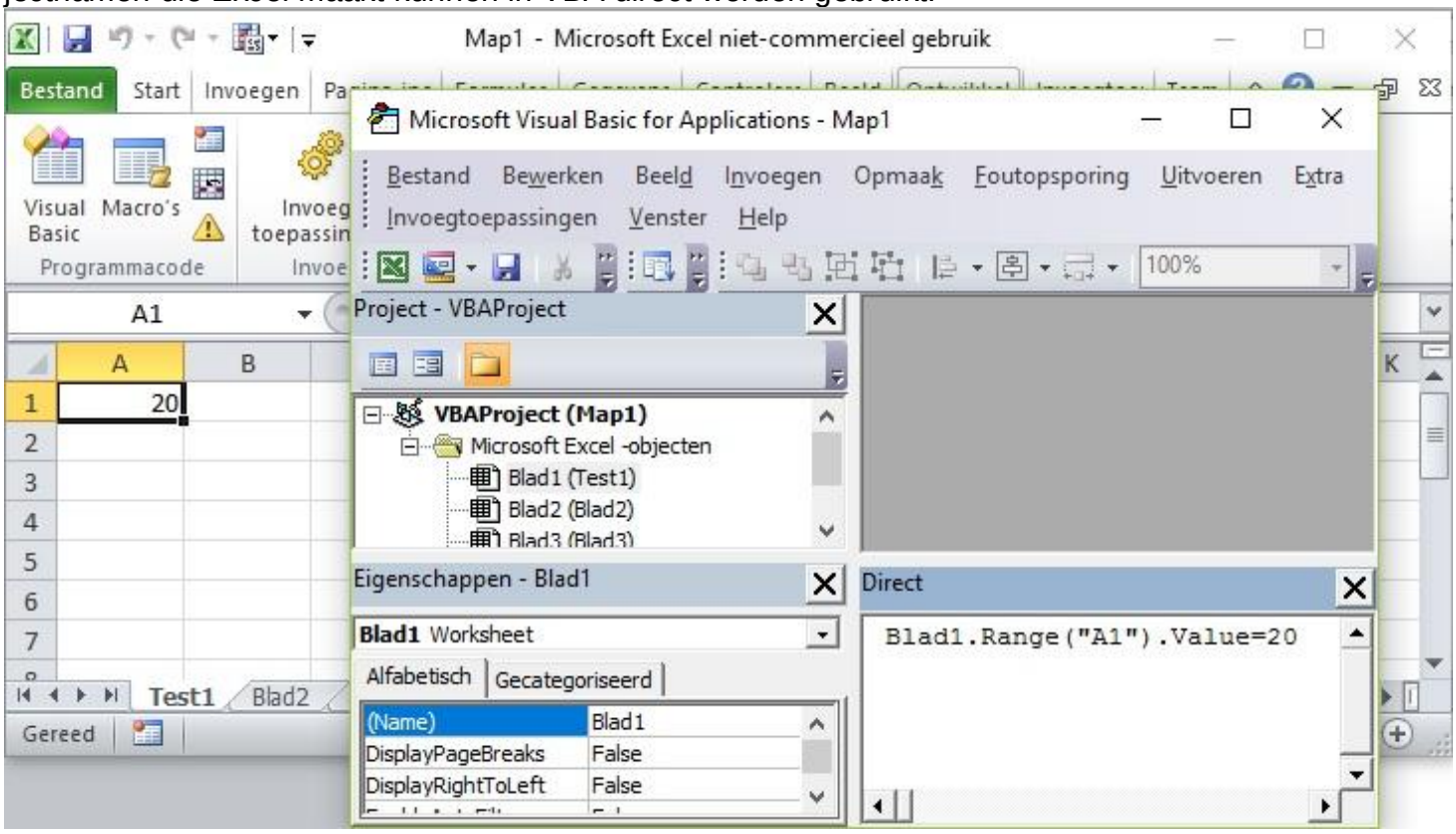
Een werkmap heeft minimaal één ThisWorkbook. Hierin kan de werkmap worden beheerd, bijvoorbeeld initialisaties en handelingen voor, tijdens en na het uitvoeren van de werkmap.

In ThisWorkbook kunt u ook bepalen welk werkblad als eerst geopend moet worden.

Bent u op de voorgrond, wijzig dan onderaan de naam Blad1 in Test1. Ga terug naar het VBA project. In de projectlijst ziet u nog steeds Blad1 staan met tussen haakjes de naam Test1. U zou verwachten dat u uw eigen bladnamen als variabelen in VBA mag gebruiken, maar helaas accepteert Excel dat echter niet, zoals u onderstaande afbeelding ziet.




Excel vereist een objectnaam, maar niet de omschrijving van het object. Alleen de standaard blad objectnamen die Excel maakt kunnen in VBA direct worden gebruikt.



Toch is er een mogelijkheid gebruik te maken van de omschrijving Test1. Door in plaats van het object van het werkblad te gebruiken, mogen we ook met de Sheets objectcollectie alle werkbladen beheren. Elk werkblad heeft een itemnummer dat altijd met een1 begint. In plaats van een itemnummer op te geven, is het ook toegestaan om de omschrijving van het werkblad op te geven. Zo hoeft u zich niet te vergissen of het itemnummer wel het juiste werkblad is.

U mag wel de naam in de eigenschappenlijst wijzigen in Test1 en dan accepteert Excel wel uw eigen naam Test1 als objectnaam. Let wel op dat u zichzelf niet verward met namen noemen. Noem bijvoorbeeld een objectnaam geen Peer terwijl in de omschrijving Appel staat. U zult in ieder geval geen foutmelding krijgen zoals in de afbeelding, als u de naam in de eigenschappenlijst Test1 noemt. Onthoud alleen goed dat de omschrijving niet als objectvariabele gebruikt kan worden. Een andere reden is ook dat in een omschrijving symbolen kunnen staan, zoals spaties, die in een objectnaam niet zijn toegestaan.

# Python – Werken met tekst, condities en expressies.

Ook al heeft Python een ander soort structuur, de manier zoals we in de code met tekst, condities en expressies moeten werken, blijft hetzelfde. Er kan natuurlijk een verschil in zitten, maar dat zit wel in elke programmeertaal.

**Expressies en de karakteristieken in Python**
Oudere versies van Python kenden geen haakjes bij de print statements, maar vanaf versie 3 is het wel nodig. Een voorbeeld is hieronder als we een resultaat van een expressie willen weergeven.

```
>>> x=20
>>> y=30
>>> print("x+y=",x+y)
x+y= 50
```

Zoals u aan het resultaat ziet, is er een spatie tussen het = teken en het getal. Men zou verwachten dat Python dit doet om ruimte voor het minteken te reserveren, zoals andere programmeertalen, vooral BASIC, wel doen, maar dat is niet het geval, zoals u in het volgende voorbeeld zult zien.

In plaats van drie korte regels te typen, mogen we ook meerdere codes in één regel doen. We moeten dan de code wel scheiden door puntkomma's.

```
>>> x=30;y=90;print("Aftreksom: x-y=",x-y)
Aftreksom: x-y= -60
```

Het makkelijkste van Python is dat we niet de variabelen hoeven te declareren. Python weet gelijk het type van de variabele zodra er een waarde aan toegekend wordt. Zorg er wel voor dat u dit als eerste doet voordat u een variabele gaat controleren, anders kent Python de variabele niet zoals u hieronder ziet:

```
>>> if a==0:
     a=1


Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    if a==0:
NameError: name 'a' is not defined
```

Waarden kunnen we aan dezelfde variabele verhogen of verlagen:

```
>>> i=42
>>> i=i+1;print(i)
43
>>> i+=1;print(i)
44
>>> i-=4;print(i)
40
```

Zoals we weten zien we ook de aanhalingstekens als we een string direct weergeven:

```
>>> "veertig"
'veertig'
```

```
>>> i
'veertig'
>>> print(i)
veertig
```

Alleen als we de print() functie gebruiken, zullen de aanhalingstekens niet verschijnen.

Variabelen kunnen in Python van type veranderen. Zo kan variabele i plots een string toegekend krijgen:

```
i = 42              # gegevenstype is een integer
i = 42 + 0.11       # gegevenstype is een float
i = "veertig"       # en nu is het een string
```

Zoals u ziet hoeven variabelen niet de types hebben naar waar ze verwijzen. Dit is anders dan in programmeertalen als C/C++ en Visual Basic, waar wel de variabelen hun vaste types hebben. Vandaar dat variabelen in Python elke soorten waarden kunnen aannemen. De eerste waarde zal de waarde zijn waarmee de variabele geïnitialiseerd wordt, zodat Python de variabele kent.

In Python 2 waren er twee types voor gehele getallen, een int en een long. Nu in Python 3 bestaat er alleen maar een int. Onderstaande regels worden niet geaccepteerd.

```
>>> 1L
  File "<stdin>", line 1
    1L
     ^
SyntaxError: invalid syntax
>>> x = 43
>>> long(x)
Traceback (most recent call last):
  File "<h;stdin>", line 1, in <module>
NameError: name 'long' is not defined
```

Handig in Python zijn de complexe getallen. Het is even kijken hoe het precies werkt. In Python kunnen we de letter j gebruiken om complexe getallen uit te voeren.

```
>>> x = 3 + 4j
>>> y = 2 - 3j
>>> z = x + y
>>> print(z)
(5+1j)
```

Hoe werkt dit?
Het letterteken j vertelt Python om niet de expressie te berekenen. Door een tweede expressie te maken met een complex getal, zal er een berekening worden gemaakt als deze twee met elkaar worden opgeteld en toegekend worden aan variabele z, die echter complex wordt. Het getal 5 komt uit 3 + 2 en het getal 1 uit 4 – 3. Het is handig dat we dit in Python kunnen gebruiken.
Hieronder een groter voorbeeld:

```
>>> x = 4 + 9 - 3j
>>> y = 4 + x + 2j
>>> c = -3 + 4 * 2j
>>> z = x + y - c
>>> print(z)
(33-12j)
```

Zoals u ziet kunt u elke expressie erbij gebruiken, zoals u gewend bent een berekening uit te voeren.

We hoeven niet speciaal pas de berekening te printen waar geen j teken meer staat, zoals hieronder kunt u zien dat we ook de resultaten uit x en y kunnen zien:

```
>>> print(y)
(17-1j)
>>> print(x)
(13-3j)
```

In Python 3 zijn de resultaten van delingen altijd met het type float.

```
>>> 10 / 3
3.3333333333333335
```

Om in Python erachter te komen van welk type het resultaat is, kunnen we gebruik maken van het sleutelwoord type, zoals hieronder.

```
>>> x = 11
>>> y = 3
>>> z = x / y
>>> type(z)
<class 'float'>
>>> print(z)
3.6666666666666665
```

Het sleutelwoord type geeft een class aan met tussen aanhalingstekens van welk type de variabele is. Om een integer type als resultaat te krijgen, moeten we twee slashes gebruiken. We kunnen dan ook nog met het type achterhalen of we een int hebben of niet.

```
>>> z = 10 // 3
>>> type(z)
<class 'int'>
>>> print(z)
3
```

**Strings**
Werken met tekst is in Python ook iets anders. Een vak apart zou je zeggen. Python kent stringmogelijkheden die we ook wel in de oude printf() functie kennen in C: de tekstformattering.

Ook het transformeren kunnen we in Python doen. Willen we een hexgetal weergeven, dan zal dat als een string worden geprint, zoals hieronder. Met een int() kunnen we die weer omkeren.

```
>>> hex(65)
'0x41'
>>> int(0x41)
65
```

Voor hexadecimale of octale getallen moeten we die niet tussen aanhalingstekens typen, maar omdat int() ook stringgetallen kan omzetten, moet dat juist wel.

```
>>> int('65')
65
```

```
>>> int(0xff)
255
```

We kunnen tekst op twee manieren gebruiken: tussen enkele aanhalingstekens en dubbele aanhalingstekens.

```
>>> s = 'Dit staat tussen enkele aanhalingstekens.'
>>> s = "Dit staat tussen dubbele aanhalingstekens."
```

We kunnen ze beide in één string gebruiken. Degene die binnen de string staat wordt ook afgedrukt.

```
>>> print(s)
Dit is de 'vierde' keer!
>>> s = 'Dit is de "zoveelste" keer!'
>>> print(s)
Dit is de "zoveelste" keer!
```

We kunnen de lengte van de string bepalen met de functie len().

```
>>> l = len(s)
>>> print(l)
27
```

Lijkt wat Basic, niet?
Maar Python zal echter een foutmelding geven als we proberen met int() te achterhalen of een string omgezet kan worden in een getal of niet.

```
>>> a = '25L'
>>> int(a)
Traceback (most recent call last):
  File "<pyshell#80>", line 1, in <module>
    int(a)
ValueError: invalid literal for int() with base 10: '25l'
```

In andere programmeertalen kunnen we het wel doen met int() of met de functie val(). De val() functie in Basic geeft 25 als achter de waarde andere tekens dan cijfers staan.

Voor strings kunnen we met de slash een extra aanhalingsteken gebruiken om weer te geven.

```
>>> s = 'It doesn\'t matter'
>>> print(s)
It doesn't matter
```

Maar dankzij de dubbele aanhalingstekens is dat niet nodig.

```
>>> s = "It doesn't matter"
>>> print(s)
It doesn't matter
```

We kunnen ze ook beide weergeven, maar dan zijn de slashes wel nodig.

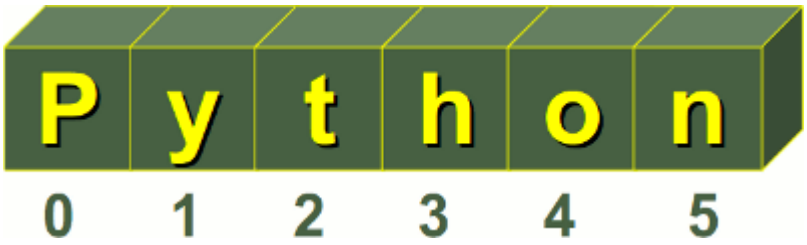
```
>>> txt = "He said: \"It doesn't matter, if you enclose a string in single
or double quotes!\""
>>> print(txt)
```

```
He said: "It doesn't matter, if you enclose a string in single or double quotes!"
```

In Python kunnen we elk teken uit een string halen door een index op te geven:

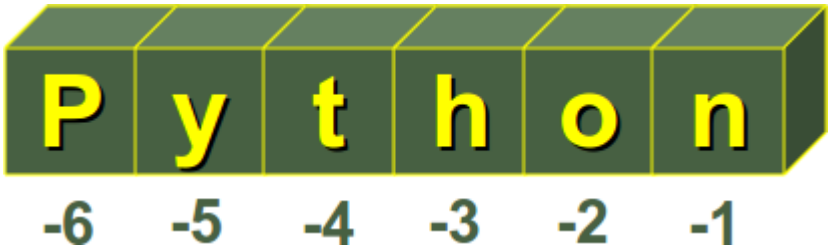
```
>>> s[0]
'I'
>>> s[-1]
'r'
```

Een negatieve index haalt het meest rechtse teken uit een string. Dat is altijd een -1, want -0 is gewoon 0.




Hier ziet u de positieve en negatieve indexwaarden van de tekst Python. U ziet dus dat -6 de indexwaarde is van de eerste letter of dat de indexwaarde 0 is.

Om de letter t uit de positieve index te kunnen vinden via de negatieve index, moeten we de lengte van de string bij de indexwaarde optellen.




```
>>> t[-4]
't'
>>> t[-4+6]
't'
```

Weten we de lengte niet, dan mag ook de functie len() gebruikt worden.

```
>>> t[-4+len(t)]
't'
```

Het maakt niet uit met welke index u begint. Als u altijd de lengte van de string erbij optelt, komt u bij hetzelfde teken terug. We kunnen dus ook eerst een expressie uitvoeren en het resultaat als index gebruiken.

```
>>> i = -2 + len(t)
>>> t[i]
'o'
>>> t[-2]
'o'
```

We kunnen met strings nog meer in Python doen.

- Concatenatie, of ook wel genoemd samenvoegen, kan worden gedaan met de plus operator.
- Herhalen. Dit is handig als we een deelstring meerdere keren achter elkaar willen weergeven. Voorbeeld: `"*-*" * 3` geeft `'*_**_**_*'`
- Slicing, of ook wel genoemd bereik, kan worden gebruikt om een deel van een string te kunnen weergeven. Voorbeeld uit afbeelding Python: `t[2:4]` geeft `'th'`. Onthoud dat slicing alleen

met indexwaarden werkt, de tweede waarde is dus niet de lengtewaarde. We kunnen het dus niet vergelijken met de mid() functie uit andere programmeertalen.

In Python kunnen we niet de inhoud van een string wijzigen met gebruik van de index.

```
>>> s[0] = 'V'
Traceback (most recent call last):
  File "<pyshell#102>", line 1, in <module>
    s[0] = 'V'
TypeError: 'str' object does not support item assignment
```

We kunnen strings met elkaar vergelijken met de == operator.

```
>>> a = "Met"
>>> b = "Met"
>>> a == b
True
```

Python kent een formattering die in strings gebruikt kunnen worden.

Formatteringssymbolen in strings

| | |
|---|---|
| \newline | Genegeerd |
| \\ | Backslash (\) |
| \' | Aanhalingsteken (') |
| \" | Aanhalingstekens (") |
| \a | ASCII Bell (BEL) |
| \b | ASCII Backspace (BS) |
| \f | ASCII Formfeed (FF) |
| \n | ASCII Newline (LF) |
| \N{name} | Teken naam in de Unicode-database (alleen Unicode) |
| \r | ASCII Returnline (CR) |
| \t | ASCII Horizontal Tab (TAB) |
| \uxxxx | Teken met 16-bit hex waarde xxxx (alleen Unicode) |
| \Uxxxxxxxx | Teken met 32-bit hex waarde xxxxxxxx (alleen Unicode) |
| \v | ASCII Vertical Tab (VT) |
| \ooo | Teken met octale waarde ooo |
| \xhh | Teken met hex waarde hh |

De newline is een van de handigste formattering om in strings te gebruiken. Hiermee kunnen we meer regels tegelijk in één string weergeven:

```
>>> print("Hallo!\nTot ziens!")
Hallo!
Tot ziens!
```

Met karakter \t kunt u een horizontale tab in een string gebruiken:

```
>>> print("Dit maakt\truimte.")
Dit maakt       ruimte.
```

Strings, condities en expressies kunnen we ook samen gebruiken. Tot nu toe is alles direct ingevoerd en uitgevoerd in de Shell van Python. Python heeft echter ook een IDLE venster. Hiermee kunnen we programma's mee schrijven die pas uitgevoerd worden zodra we de module starten. Deze worden dan via de Shell gerestart.

Ga naar het menu File en kies New File om een nieuw bestand te maken. Het IDLE venster wordt dan geopend. Merk op dat u de drie symbolen niet meer ziet. Ook zal elke regel niet meer direct uitgevoerd worden, maar denk er wel aan dat u gebruik moet maken van de suits (zie vorige bulletin) wanneer codeblokken nodig zijn, zoals na een if, een while of een for statement.

Typ onderstaand programma over in het IDLE venster.

```
s = input("Wat vindt u lekker? ")
if s == "appel":
    print("Lekker gezond!")
elif s == "patat":
    print("Ook lekker, maar eet niet te vaak.")
elif s == "tosti":
    print("Mmmmm, dat kan wel als lunch!")
else:
    print("Ja, die keuze ", s, " is ook lekker.")
```

Sla het programma op en start het door te klikken op Run Module in het menu Run. U gaat meteen terug naar het Shell venster waar Restart verschijnt en uw programma in uitgevoerd wordt.

```
>>> ================================ RESTART
================================
>>>
Wat vindt u lekker? appel
Lekker gezond!
>>> ================================ RESTART
================================
>>>
Wat vindt u lekker? patat
Ook lekker, maar eet niet te vaak.
>>> ================================ RESTART
================================
>>>
Wat vindt u lekker? brood
Ja, die keuze  brood  is ook lekker.
```

We kunnen dus veel doen in het IDLE venster, daarom kom ik er later wat meer op terug.

In C kennen we wel de korte manier om een boolean waarde als resultaat te krijgen met een controle-regel. In Basic kennen we het als een iif() functie.

**C voorbeeld**

```
max = (a > b) ? a : b;
```

**Basic voorbeeld**

```
max = iif(a > b, a, b)
```

Dit is hetzelfde als:

```
if (a > b)
   max=a;
else
   max=b;
```

C programmeurs zouden moeten wennen aan de structuur van Python die het heel anders kent:

```
max = a if a > b else b
```

Het is in Python veel beter te lezen en daardoor sneller te begrijpen wat er staat. Handig is ook dat een if en een else expressies zijn, dus het is zelfs met berekeningen samen te gebruiken:

```
max = a if a > b else b * 2.45 – 4
```

De expressie b * 2.45 – 4 zal alleen uitgevoerd worden als de conditie onwaar is, maar u kunt het ook uit laten voeren als de conditie waar is zoals onderstaande regel laat zien:

```
max = a + 20 if a > b else b
```

Of gebruik het in beide gevallen. Hoe dan ook, u kunt expressies als waar en onwaar uit laten voeren of als conditie gebruiken:

```
max = a + 20 if a > b * 3 else b / 2
```

Met behulp van if statements binnen programma's, kunnen ze gemakkelijk tot complexe beslissingsstructuren werken, dat wil zeggen, de statements kunnen worden gezien als de takken van een boom.

Wilt u meer weten over hoe expressies en functies in Python gemaakt kunnen worden en werken? In de volgende Bulletin komt er een onderwerp over expressies en functies.

**! Bij voorbeeldprogramma's: Geen drietekens links betekent: maak gebruik van het IDLE venster!**

In het volgende programma zullen drie floatwaarden ingevoerd worden. De grootste waarde hiervan zal worden weergegeven.

```
x = float(input("1st Number: "))
y = float(input("2nd Number: "))
z = float(input("3rd Number: "))

if x > y and x > z:
    maximum = x
elif y > x and y > z:
    maximum = y
else:
    maximum = z

print("The maximal value is: " + str(maximum))
```

Dit programma heb ik in de Shell twee keer uitgevoerd:

```
>>> ================================ RESTART
================================
>>>
1st Number: 4
```

```
2nd Number: 3.6
3rd Number: 3.4
The maximal value is: 4.0
>>> ================================ RESTART
================================
>>>
1st Number: 8.2
2nd Number: 4
3rd Number: 9.3
The maximal value is: 9.3
```

Er zijn nog meer mogelijkheden om het soortgelijks weer te geven:

```
x = float(input("1st Number: "))
y = float(input("2nd Number: "))
z = float(input("3rd Number: "))

if x > y:
    if x > z:
        maximum = x
    else:
        maximum = z
else:
    if y > z:
        maximum = y
    else:
        maximum = z

print("The maximal value is: " + str(maximum))
```

Als we willen lezen in een willekeurig aantal elementen, dat niet opgeslagen moet worden in een lijst, kunnen we het maximale berekenen op de volgende manier:

```
number_of_values = int(input("How many values? "))

maximum = float(input("Value: "))
for i in range(number_of_values - 1):
    value = float(input("Value: "))
    if value > maximum:
        maximum = value

print("The maximal value is: " + str(maximum))
```

In de volgende Bulletin komen ook andere lusstructuren aan bod, zoals de while lus, en gaan we eens kijken wat we nog meer kunnen in het IDLE venster.