Programmeren Bulletin

24ste jaargang najaar 2017 Nummer 3

hcc! programmeren

Interessegroep

Inhoud

Redactioneel

Wie in Pascal wil programmeren kan nog steeds de console Pascal versie gebruiken. Dat is handig om Pascal te leren, voor de overstap naar Object Pascal.

In dit onderwerp laat ik een tutorial zien van een Unity 2D project. Hier kunt u de uitleg volgen en de afbeeldingen zien over hoe we een leuke Arkanoid kunnen maken.
Wilt u de tutorial volgen met de juiste sprites, ga dan naar het internetadres:
https://noobtuts.com/unity/2d-arkanoid-game

Waarom is de programmeertaal C cryptisch? Waarschijnlijk om het gebruik van de operatoren. Met symbolen worden operatoren gebruikt. Het gebruik ervan kan zelfs uniek zijn; een syntaxis die we in andere programmeertalen niet kunnen vinden.

Door gezondheidsproblemen en werkoverplaatsing is deze Bulletin helaas een paar maanden vertraagd en daardoor later uitgekomen. De volgende Bulletin is nummer 1 van 2018.

Marco Kurvers

Programmeren in Pascal – Documentatie.

Nu we wat kennis hebben gemaakt met andere programmeertalen, is Pascal eens aan de beurt. De Free Pascal versie is een geschikte versie om Pascal te leren voor een eventuele overstap naar Object Pascal. De uitgebreide Pascal versies zijn bijvoorbeeld Lazarus en Delphi.

Commentaar
In Pascal zijn verschillende commentaar mogelijkheden, zoals:

```
(* My beautiful function returns an interesting result *)
Function Beautiful : Integer;
```

Het gebruik van (* en *) als commentaarregels werden al in de oudere Pascal versies gebruikt. Nu worden ze meer vervangen door de accolades { en }.

```
{ My beautiful function returns an interesting result }
Function Beautiful : Integer;
```

Het commentaar kan ook in meerdere regels worden gegeven:

```
{
   My beautiful function returns an interesting result,
   but only if the argument A is less than B.
}
Function Beautiful (A,B : Integer): Integer;
```

Eenregelig commentaar kan ook met dubbele slashes worden gedaan:
Dit commentaar loopt tot het einde van een commentaarregel en werd geïntroduceerd door Borland in de Delphi Pascal compiler, vandaar dat de dubbele slashes nog steeds ondersteund worden.

```
// My beautiful function returns an interesting result
Function Beautiful : Integer;
```

Free Pascal ondersteunt het gebruik van genest commentaar. De volgende regels worden geaccepteerd:

```
{ Comment 1 (* comment 2 *) }
(* Comment 1 { comment 2 } *)
{ comment 1 // Comment 2 }
(* comment 1 // Comment 2 *)
// comment 1 (* comment 2 *)
// comment 1 { comment 2 }
```

De volgende twee commentaarregels moeten een regel zijn en zijn daarom fout:

```
// Valid comment { No longer valid comment !!
    }

// Valid comment (* No longer valid comment !!
    *)
```

Identifiers
Net als de Pascal sleutelwoorden zowel met hoofdletters als met kleine letters hetzelfde zijn, is dat ook zo bij identifiers. Voor degenen die Pascal niet goed kennen: identifiers worden ook wel variabelen genoemd.

Dit betekent dat de naam:

```
myprocedure;
```

dezelfde is als

```
MyProcedure;
```

Tip! Vanaf versie 2.5.1 is het mogelijk gereserveerde woorden als identifiers te gebruiken. De naam moet dan wel beginnen met een ampersand &.

```
program testdo;

procedure &do;
begin
end;

begin
  &do;
end.
```

De meeste identifiers kunnen een hint directive hebben aan het einde van de definitie. De hint directives zijn:

- deprecated
  Het gebruik van deze identifier is verouderd, gebruik in plaats daarvan een alternatief. Het deprecated sleutelwoord kan worden gevolgd door een tekenreeksconstante met een boodschap. De compiler zal dit bericht weergeven als de identifier wordt aangetroffen.
- experimental
  Het gebruik van deze identifier is experimenteel: dit kan worden gebruikt voor het markeren van nieuwe functies die met voorzichtigheid gebruikt moeten worden.
- platform
  Dit is een platform-afhankelijke identifier: deze kan niet worden gedefinieerd op alle platforms.
- unimplemented
  Dit moet alleen worden gebruikt op functies en procedures. Het moet worden gebruikt om aan te geven dat een bepaalde functie nog niet is geïmplementeerd.

Hieronder volgen enkele voorbeelden:

```
Const
  AConst = 12 deprecated;

var
  p : integer platform;

Function Something : Integer; experimental;

begin
  Something:=P+AConst;
end;
```

```
begin
  Something;
end.
```

Dit zal het volgende resultaat geven:

```
testhd.pp(11,15) Warning: Symbol "p" is not portable
testhd.pp(11,22) Warning: Symbol "AConst" is deprecated
testhd.pp(15,3) Warning: Symbol "Something" is experimental
```

Hint directives kunnen achter elke soort identifiers staan: units, constants, types, variables, functions, procedures en methoden.

Getallen

Getallen zijn standaard in decimale notatie. Real of decimale getallen zijn geschreven met behulp van engineering of wetenschappelijke notatie (bijvoorbeeld 0.314E1).

1. Normale, decimale formaat (base 10). Dit is het standaard formaat
2. Hexadecimaal formaat (base 16), op dezelfde manier als Turbo Pascal doet. Om een constante waarde op te geven in hexadecimaal formaat gebruikt u het dollar teken ($). Dus $FF is hetzelfde als 255. Onthoud dat hoofdletters niet belangrijk zijn wanneer u het hexadecimaal formaat gebruikt.
3. Vanaf versie 1.0.7 wordt ook het octale formaat (base 8) ondersteund. Om een octaal formaat op te geven gebruikt u het ampersand teken (&). Bijvoorbeeld 15 is in octaal formaat &17.
4. Binaire notatie (base 2). Een binair getal kan worden opgegeven met gebruik van het procent teken (%). Dus 255 wordt in binaire notatie %11111111.

! De octale en binaire notatie worden niet ondersteund in Turbo Pascal en Delphi compatibiliteitsmode.

Labels

Een label is een naam voor een locatie in de code waar naartoe gesprongen kan worden vanuit een andere locatie met een goto statement. Een label is een standaard identifier of een getal.

! De –Sg of –Mtp schakelaars moeten opgegeven worden voordat labels gebruikt mogen worden. Normaal ondersteunt Free Pascal geen label en goto statements. De {$GOTO ON} directive kan ook worden gebruikt om het gebruik van labels en het goto statement toe te staan.

Onderstaande voorbeelden zijn toegestane labels:

```
Label
  123,
  abc;

var
     i: Integer;
begin
     goto 123;
     writeln('niet zichtbaar');
     123:
     writeln('wel zichtbaar');
     i := 0;
     abc:
          Inc(i, 1);
          writeln('Waarde i: ', i);
```

```
    if i < 10 then goto abc;
end.
```

Natuurlijk is het gebruik van het goto statement niet altijd nodig. De label abc waarnaartoe gesprongen wordt vormt een herhalingslus, zolang de waarde van i kleiner is dan 10. Pascal kent een for lus of een repeat ... until lus om hetzelfde te kunnen doen.

Stringkarakters

Een tekenreeks (stringkarakters of korte tekenreeks) is een reeks van nul of meer tekens (byte grootte), tussen aanhalingstekens, en op een enkele regel van de broncode van het programma: geen letterlijke carriage return of regelinvoer tekens kunnen worden weergegeven in de tekenreeks.

Een lege stringkarakterset met alleen maar de aanhalingstekens (‘’) is een lege string.

Een string bestaat standaard uit 7-bit ASCII tekens of Unicode tekens (normaal UTF-8 gecodeerd). In een string kunnen ook tekens opgegeven worden die normaal niet vanuit het toetsenbord ingetoetst kunnen worden, zoals #27 voor het escape teken. De C-constructietekens in de tekenreeks (met behulp van een backslash) worden niet ondersteund in Pascal.

De volgende voorbeelden zijn toegestane strings:

```
'This is a pascal string'
'a'
'A tabulator character: '#9' is easy to embed'
```

Het volgende is geen toegestane string:

```
'the string starts here
 and continues here'
```

De string had getypt moeten worden als:

```
'the string starts here'#13#10'   and continues here'
```

of

```
'the string starts here'#10'   and continues here'
```

Constanten

Net zoals in Turbo Pascal, ondersteunt Free Pascal gewone en getypeerde constanten.

Gewone constanten
Getypeerde constanten
Resource strings

Gewone constanten

Deze declaraties zijn constanten met gebruik van een identifiernaam gevolgd door een = teken met daarachter een expressie, een getal, een string, een booleaanse waarde of een enumeratiewaarde.

De compiler moet de expressie in een constante declaratie in compileer tijd evalueren. Dit betekent dat het merendeel van de functies in de Run-Time library niet kan worden gebruikt in een constante declaratie.
Als een eerder gedeclareerde gewone constante in de code wordt gebruikt, wordt de werkelijke waarde van de constante in plaats van de naam ingevoegd door de compiler. Dat wil zeggen dat de volgende 2 stukken code volledig gelijkwaardig zijn:

```
Const
  One = 1;

begin
  Writeln(One);
end.
```

De volgende code werkt hetzelfde:

```
begin
  Writeln(1);
end.
```

Alleen constanten van de volgende types kunnen worden gebruikt:

- Ordinal types
- Set types
- Pointer types (maar de enige toegestane waarde is Nil).
- Real types
- Char,
- String

De volgende constanten zijn toegestaan:

```
Const
  e = 2.7182818;  { Real type constant. }
  a = 2;          { Ordinal (Integer) type constant. }
  c = '4';        { Character type constant. }
  s = 'This is a constant string'; {String type constant.}
  sc = chr(32)
  ls = SizeOf(Longint);
  P = Nil;
  Ss = [1,2];
```

U kunt geen andere waarde toekennen aan een constante identifier. De compiler zal een foutmelding geven wanneer u het toch doet.

Voor de constanten van de string is het type van de tekenreeks afhankelijk van de compilerschakelopties. Als een bepaald type gewenst is, zal een getypte constante moeten worden gebruikt, zoals uitgelegd in de volgende sectie.

Voor versie 1.9 werd nog geen 64-bit ondersteund. Na die versie wel.

Getypeerde constanten
Soms is het noodzakelijk om het type op te geven van een constante, bijvoorbeeld voor constanten van complexe structuren. In tegenstelling tot gewoneconstanten, kan een waarde worden toegewezen tijdens de uitvoeringstijd. Dit is een oud concept van Turbo Pascal, dat is vervangen met ondersteuning voor geïnitialiseerde variabelen. De ondersteuning voor het toewijzen van waarden aan getypeerde constanten wordt geregeld door de directive {$J}. Het kan worden uitgeschakeld, maar is stan-

daard ingeschakeld (voor Turbo Pascal-compatibiliteit). Geïnitialiseerde variabelen zijn altijd toegestaan.

Het moet worden benadrukt dat getypeerde constanten automatisch geïnitialiseerd zijn bij het starten van het programma. Dit geldt ook voor lokale getypeerde constanten en geïnitialiseerde variabelen. Lokale getypeerde constanten zijn ook geïnitialiseerd bij het starten van het programma. Als hun waarde is veranderd tijdens eerdere aanroepingen van een functie, zullen zij hun gewijzigde waarde behouden, dat wil zeggen, ze worden niet geïnitialiseerd telkens wanneer die functie wordt aangeroepen.

```
procedure TypedConst;
const a: Integer = 10;
begin
     Writeln('Initialisatiewaarde: ', a);
     a := 15;
     Writeln('Nieuwe waarde: ', a);
end;

begin
     TypedConst;
     TypedConst;
end.
```

Als we het programma uitvoeren, verschijnen de waarden als volgt:

```
Initialisatiewaarde: 10
Nieuwe waarde: 15
Initialisatiewaarde: 15
Nieuwe waarde: 15
```

Resource strings
Een speciaal soort constante declaratieblok is het Resourcestring blok. Resourcestring declaraties zijn vergelijkbaar met constante string declaraties: resource strings fungeren als constante strings, maar ze kunnen worden gelokaliseerd door middel van een set speciale routines in de objPas unit.

Het volgende is een voorbeeld van een resource string definitie:

```
Resourcestring
  FileMenu = '&File...';
  EditMenu = '&Edit...';
```

Alle string constanten gedefinieerd in de resourcestring sectie worden opgeslagen in speciale tabellen. De strings in deze tabellen kunnen worden gemanipuleerd tijdens de uitvoertijd met enkele speciale mechanismen in de objPas unit.
Semantisch fungeren de strings als gewone constanten. Het is niet toegestaan om er waarden aan toe te wijzen (behalve via de speciale mechanismen in de objPas unit). Ze kunnen echter worden gebruikt in toewijzingen of expressies als gewone tekenreeksconstanten. De voornaamste toepassing van het resourcestring gedeelte is bedoeld als een eenvoudig middel van internationalisering.

Merk op dat een resource string, die wordt gegeven als een expressie, niet verandert als de delen van de expressie worden gewijzigd:

```
resourcestring
  Part1 = 'First part of a long string.';
  Part2 = 'Second part of a long string.';
```

```
  Sentence = Part1+' '+Part2;
```

Als de lokalisatie routines Part1 en Part2 verwerkt zijn, zal de Sentence constante niet automatisch verwerkt worden. Het heeft een aparte ingang in de resource string tabellen en moet daarom apart worden verwerkt. De bovenstaande constructie zegt gewoon dat de resulterende waarde van Sentence gelijk is aan Part1 + ' ' + Part2.

Ook bij gebruik van resource strings in een array zullen alleen de geïnitialiseerde waarden van de resource strings worden gebruikt in de array. Wanneer de individuele constanten verwerkt zijn, zullen in de array elementen de waarden behouden blijven.

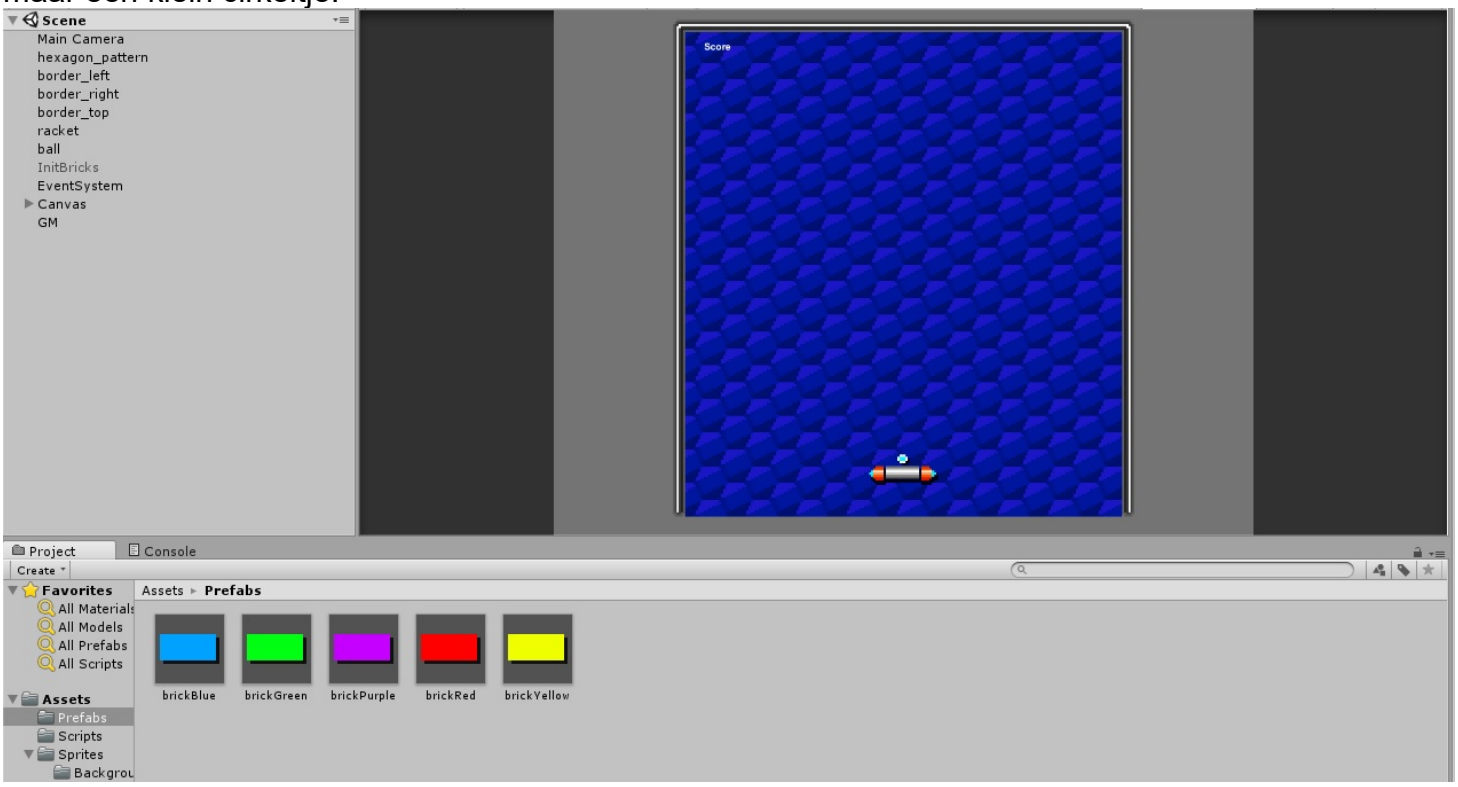
```
resourcestring
  Yes = 'Yes.';
  No = 'No.';

Var
  YesNo : Array[Boolean] of string = (No,Yes);
  B : Boolean;

begin
  Writeln(YesNo[B]);
end.
```

Dit zal 'No.' of 'Yes.' afdrukken afhankelijk van de waarde van B, zelfs als de constanten No en Yes zijn vertaald door sommige gelokaliseerde mechanismen.

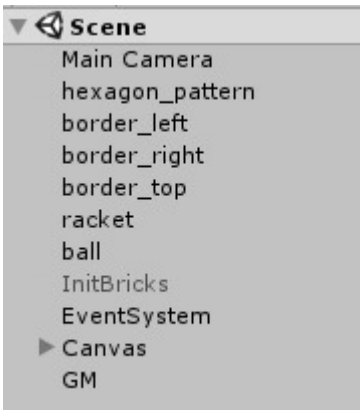
Unity 2D – Tutorial: een leuke Arkanoid (1).

In de vorige Bulletin heeft u kunnen zien hoe de Main Camera met het canvas, samen met de sprites, images en Game Objecten, in de Inspector ingesteld kan worden. U hebt gezien hoe het samenwerkt om een gamevenster goed in elkaar te zetten. Echter was de uitleg beknopt en ging het erom wat u nodig hebt als u in 2D een Unity project wilt maken. Daarom komt er in dit onderwerp een tutorial, zodat u kunt zien hoe u moet beginnen vanaf een leeg venster om een leuk Arkanoid spel te maken.

Arkanoid is een beroemd spel van een oude BreakOut Arkanoid die toen op de Commodore 64 werd gespeeld. Hieronder ziet u het complete project in Unity staan. Wie het spel kent, kan zich misschien wel de achtergrond en de racket herinneren. De bal is nu gevuld. De oude bal was vroeger alleen maar een klein cirkeltje.



Zoals u de borderranden ziet, is het niet helemaal hetzelfde als het oude spel, maar het is leuk om een Arkanoid te maken die er ongeveer zo uitziet.

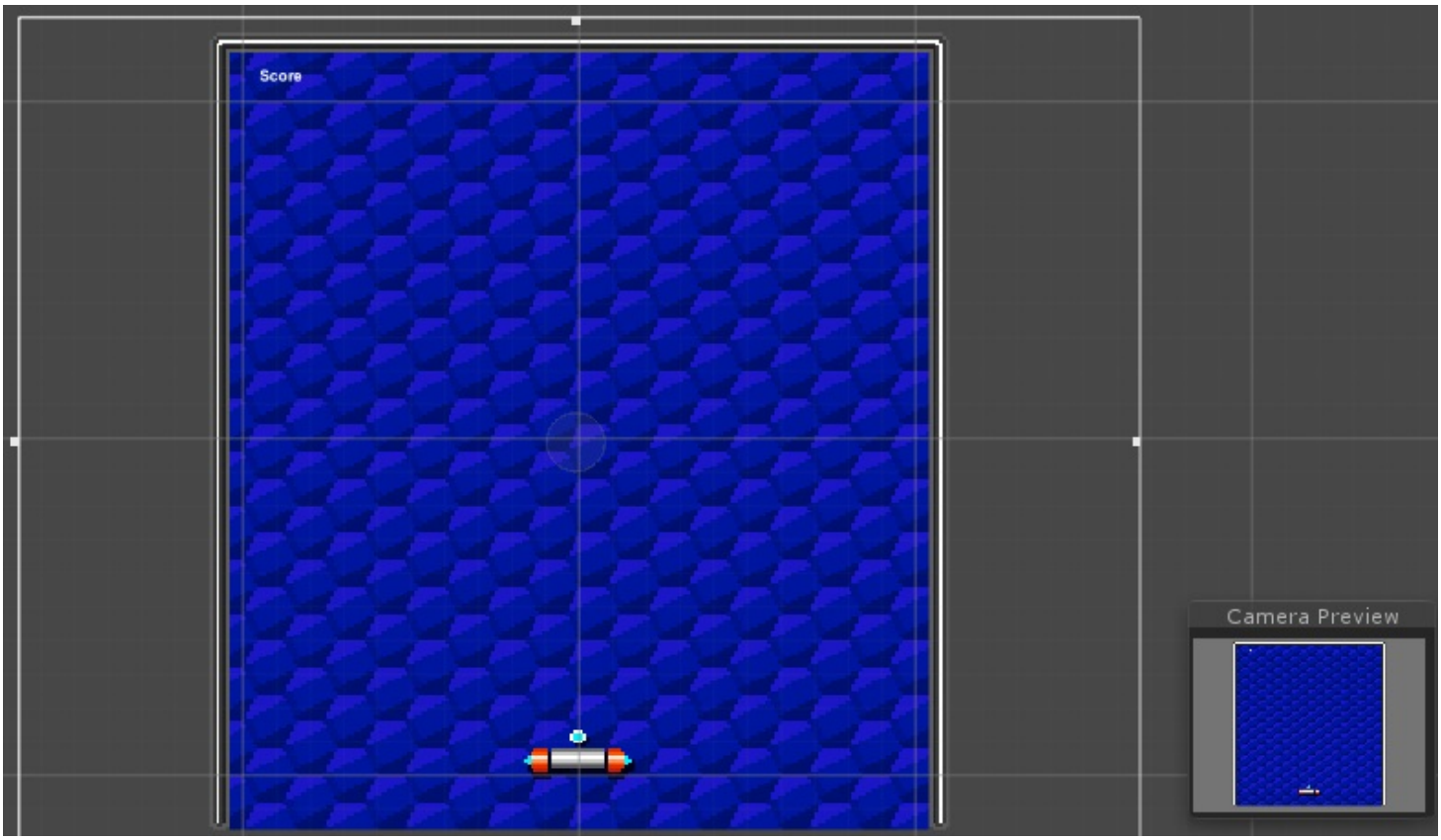


Links ziet u de Main Camera, die nodig is om de Scene tijdens het spelen te kunnen zien. De hexagon_pattern is de blauwe achtergrond. In de vorige Bulletin liet ik u zien dat het handig is om de achtergrond op een canvas te plaatsen voor uitlijning. Een canvas is niet speciaal nodig en wordt in dit project ook niet gebruikt. De achtergrond is geen tiled sprite, maar past goed binnen de ingestelde grootte van de camera. Ook geldt dit voor de borderranden, die precies de lengte hebben om op de achtergrond te kunnen passen. De racket en de bal zijn de Game Objecten die, samen met de bricks, het belangrijkst zijn. Daar moet mee gespeeld kunnen worden, dus hebben de Game Objecten scripts nodig.

Het enige canvas dat in dit project nodig is, is voor de tekst. Zonder het canvas kunnen we geen tekst plaatsen.

De enige vraag is nu: waar zijn de bricks Game Objecten? Waarom zijn ze niet op de Scene aanwezig? Het antwoord komt nu nog even niet. Op de eerste afbeelding ziet u de bricks al, alleen zijn het

prefabs. De tutorial is te groot om alles in één Bulletin uit te leggen, daarom komt het onderwerp prefabs later aan de orde.

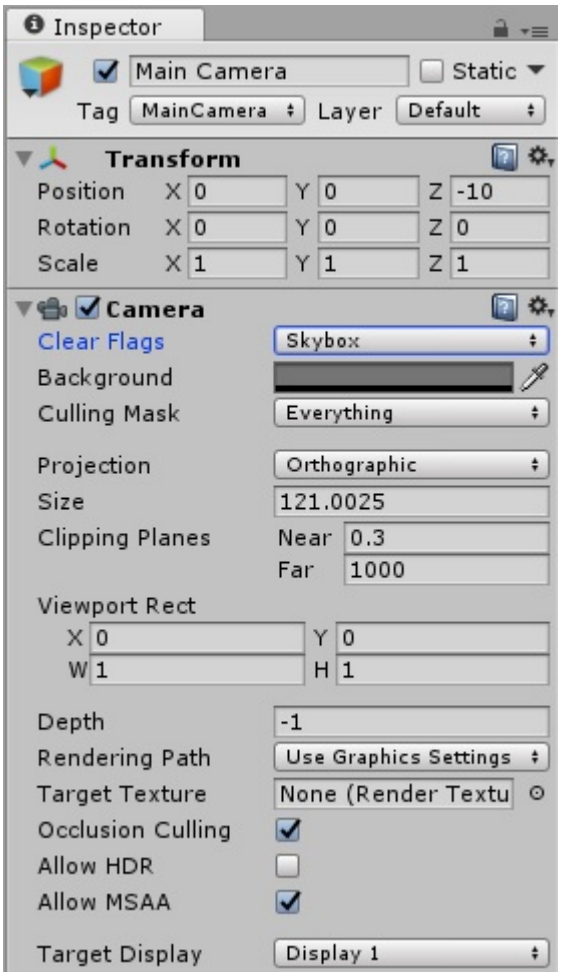


Deze afbeelding kunt u zien in de Scene. De dikke lijnen geven de grootte van de Camera aan. Rechts ziet u dan wat u ziet als u het spel gaat spelen. De achtergrond zal dus niet het hele venster in beslag nemen.

Tip! Wilt u andere achtergronden gebruiken, maar hebben ze een andere grootte, dan kunt u ze gewoon gebruiken. Er zijn verschillende mogelijkheden: 1. De camera op de grootte van uw achtergrond instellen. 2. Zorgen dat uw achtergrond de grootte heeft dat deze binnen de cameraranden past (zoals in dit project het geval is). 3. Gebruik maken van een image om de achtergrond op een canvas uit te lijnen of te tilen (zie vorige Bulletin).

De tutorial

Start Unity met een leeg 2D project en noem deze Arkanoid of een naam dat u leuk vindt. Wilt u ook deze sprites gebruiken? De sprites van de tutorial Arkanoid is gratis op Internet te downloaden. Houd er rekening mee dat de Engelstalige tutorial anders is dan deze, omdat ik wijzigingen heb aangebracht, zoals de prefabs, betere besturing en een goede racket- en balbeweging die ervoor moeten zorgen dat de sprites niet gaan ‘vliegen’. Later kom ik daar op terug, omdat die instellingen ook belangrijk zijn om de spellen soepel uit te laten voeren.



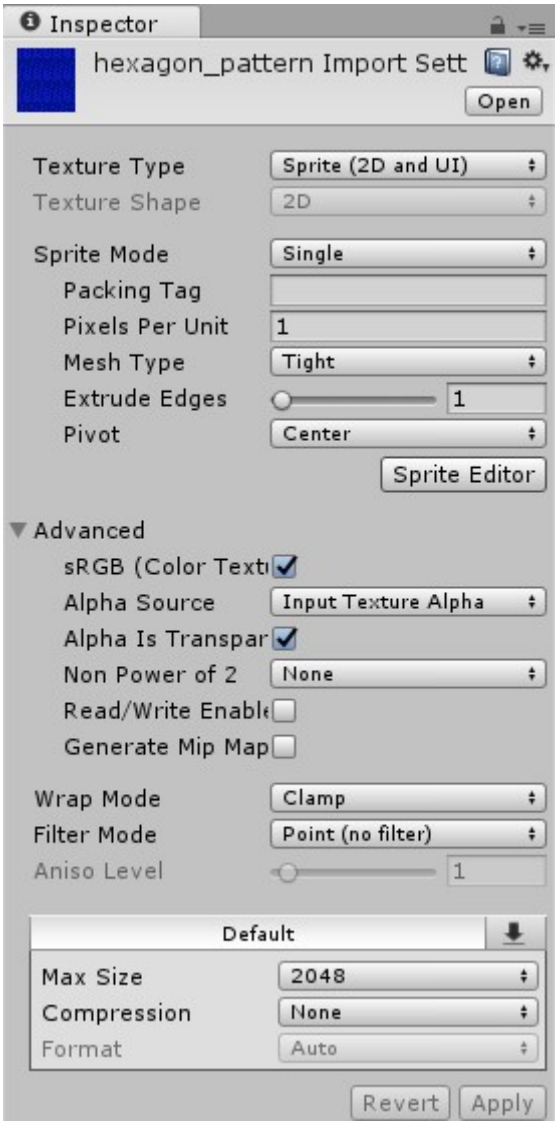
Klik op de Main Camera in de Hierarchy en kijk naar de Inspector. De afbeelding geeft de Inspector aan met de instellingen zoals ze nu moeten zijn. Schakel de **Z** positie van de camera in op **-10** en kies **Skybox** bij **Clear Flags**. De **Projection** moet staan op **Orthographic**. Controleer ook de **Depth** of deze op **-1** staat. Zo niet, typ de waarde dan in.

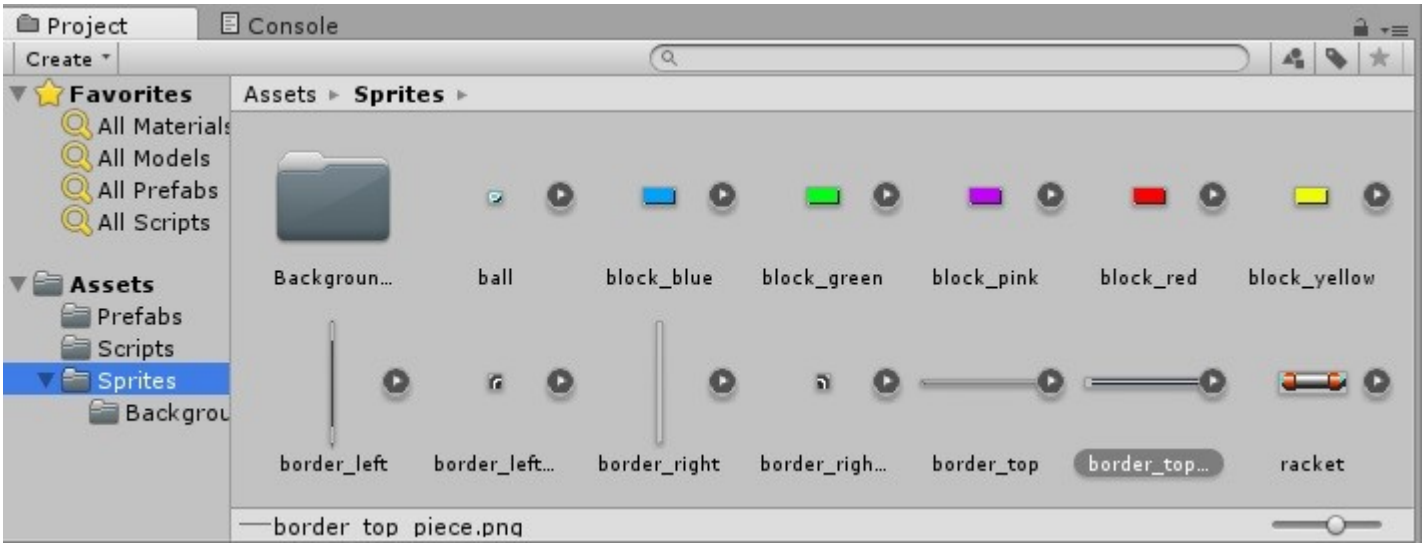
Kijk ook voor de zekerheid de rest van de instellingen na, zodat ze goed overeen komen met de instellingen die u op deze afbeelding ziet.

Als u in de Assets een map heeft aangemaakt met de naam **Backgrounds**, eventueel in een nieuwe **Sprites** map, en een mooi geschikte achtergrond sprite toegevoegd hebt, dan kunt u de Inspector van de achtergrond bekijken door op de sprite te klikken. De afbeelding rechts geeft de Inspector aan van de hexagon_pattern sprite. Stel de **Pixels Per Unit** in op **1**, de **Pivot** op **Center**, de **Wrap Mode** op **Clamp** en de **Filter Mode** op **Point (no filter)**.

Vergeet niet om bij elke Inspector instelling van de assets op de knop **Apply** te klikken. Met deze knop worden de wijzigingen doorgevoerd. Indien u het wel vergeet, zal Unity alsnog met de vraag komen of u de wijzigingen door wilt voeren of niet.

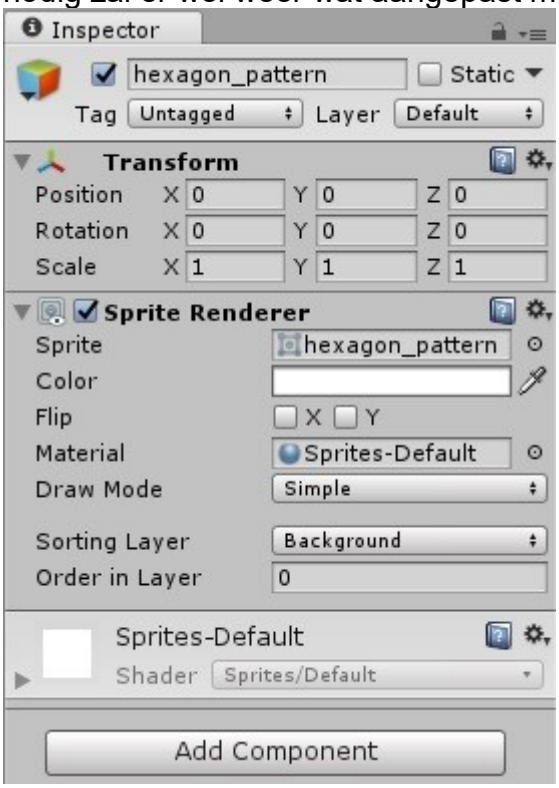
De andere sprites hebben dezelfde instellingen als de hexagon_pattern sprite. Als u alle sprites in de Assets in de Sprites map hebt staan, klik dan op elke sprite en wijzig de instellingen in de Inspector.





Dit zijn de sprites van het tutorial. Zoals gezegd kunt u de sprites downloaden van de Engelstalige pagina. Zie https://noobtuts.com/unity/2d-arkanoid-game
Op de Internetpagina worden ook de blocksprites direct gebruikt als Game Objecten. Die manier gebruik ik echter niet. De prefabs, die de Game Objecten met alle componenten in zich hebben, zijn zeer handig en het extra werk om telkens Game Objecten te dupliceren is dan ook niet nodig.

De achtergrond
Sleep de hexagon_pattern sprite naar de Hierarchy en bekijk onderstaande Inspector afbeelding. Zo nodig zal er wel weer wat aangepast moeten worden.



Bekijk goed of u een instelling anders ziet. De **Draw Mode** moet op **Simple** staan en de **Sorting Layer** op **Background**. Om een Background Layer aan te kunnen maken, klik op het balkje naast **Sorting Layer**. Kies **Add Sorting Layer…** en maak een nieuwe aan met de naam **Background**. De nieuwe layer moet er voor zorgen dat de bal over de achtergrond kan blijven bewegen en niet in aanraking komt met de hexagon_sprite.
Als het goed is moet de sprite precies kunnen passen zoals u op de Scene afbeelding ziet staan. Kunt u de Scene afbeelding niet vinden, blader dan even een stukje terug totdat u de afbeelding tegenkomt.

De Sprite Renderer is een component om er voor te zorgen dat het correct getekend (gerenderd) wordt. Elke sprite Game Object heeft in de inspector deze component.

Om de achtergrond heen komen de borderranden. Deze randen moeten er voor zorgen dat de racket en de bal niet de achtergrond kunnen verlaten. De borderranden moeten dus de botsingen op kunnen vangen, zodat de racket niet verder naar links of naar rechts kan en de bal terugkaatst.

Sleep alle borderrand sprites, border_left, border_right en border_top naar de Hierarchy. De afbeelding rechts geeft de Inspector aan van de sprite border_left.
Wijzig de waarden en instellingen zoals deze afbeelding laat zien. U zult ook zien dat u een component nog niet heeft.

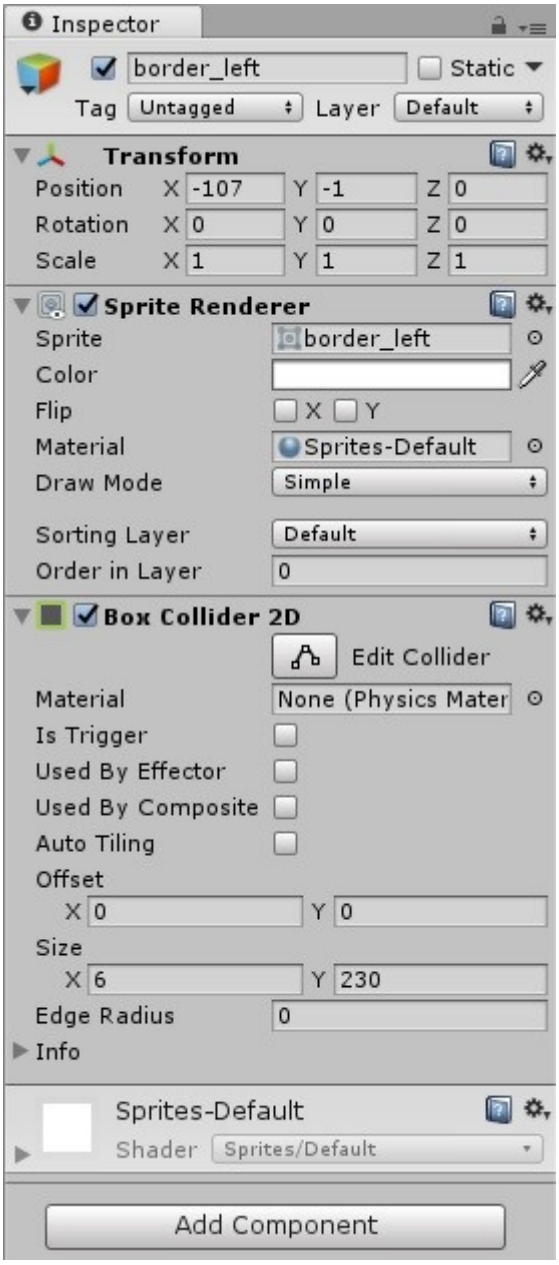
Voor het border_right Game Object heeft u een positieve waarde **107** nodig voor **Position X** en de waarde voor **Position Y** dezelfde als op deze afbeelding staat.

Voor het border_top Game Object moet voor **Position** de waarde voor **X** op **0** staan en de waarde voor **Y** op **117**.

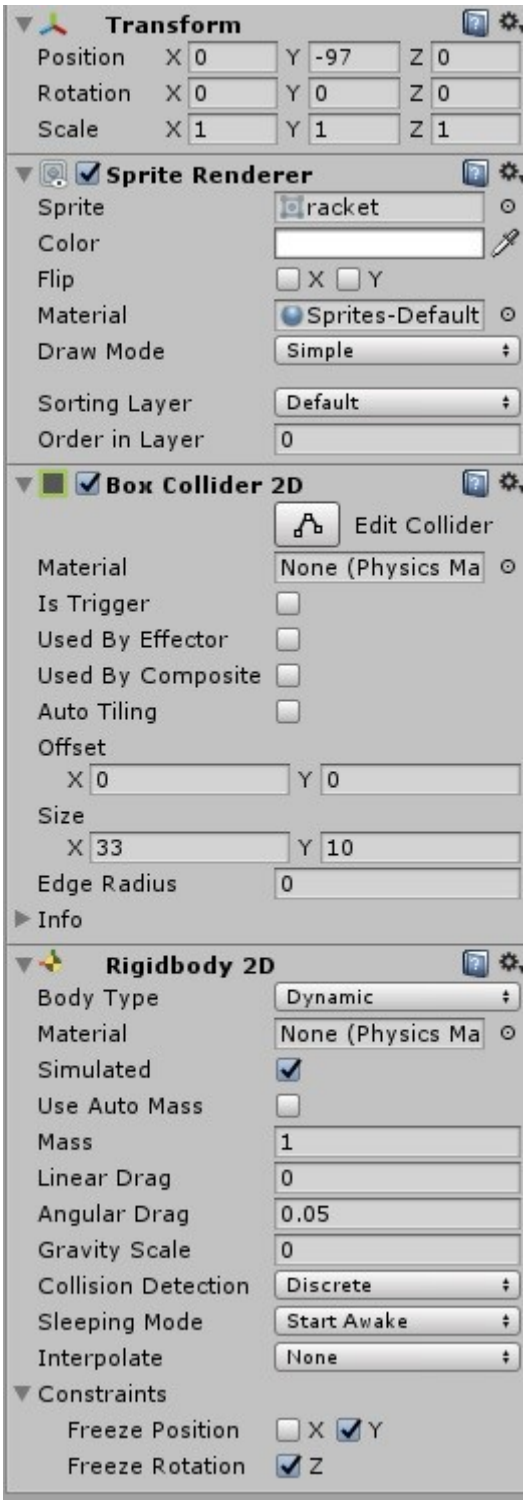
De component Box Collider 2D
Deze component is voor veel Game Objecten belangrijk om te kunnen reageren op botsingen (collisions).

Klik in de Inspector op de **Add Component** knop, zie onderaan op deze afbeelding. Een lijst verschijnt. Klik op het item **Physics 2D** en klik op het sub-item **Box Collider 2D**. Nu zal de component toegevoegd worden in de Inspector. Bekijk met deze afbeelding of er eventueel wat gewijzigd moet worden.

Nadat de component is toegevoegd, kunt u in de scene een smalle rechthoek om het Game Object zien met grepen. Het is een bescherming die er voor zorgt dat andere Game Objecten zullen reageren als ze de Box Collider tegenkomen.



Het racket Game Object



Sleep de racket sprite naar de Hierarchy. De afbeelding hier laat een lange Inspector zien van de racket. Nu hebt u nog een deel ervan. Net als bij de borderranden heeft het racket Game Object ook een Box Collider nodig om met de borderranden te kunnen communiceren, maar dat niet alleen; ook de bal zal reageren op de box collider van de racket.

Eerst moet de Transform component ingesteld worden, zodat de racket onderaan in het midden van de scene komt te staan.

De Box Collider 2D moet toegevoegd worden. Klik onderaan in de Inspector op de knop **Add Component**. Klik op **Physics 2D** en klik op **Box Collider 2D**. Bekijk of de instellingen van de component overeenkomen met de afbeelding links.

De component Rigidbody 2D
Deze component biedt veel mogelijkheden bij 2D games. Van het wegwerpen van ballen tot het laten springen van Mario. Met de component kunnen we er voor zorgen dat het Game Object een bepaalde kracht heeft.

Klik onderaan op de knop **Add Component**. Klik op **Physics 2D** en klik op **Rigidbody 2D**. Standaard zullen er wat instellingen niet overeen komen met de afbeelding hier links. Daarom zal ik ook wat uitleg geven over wat sommige instellingen betekenen.

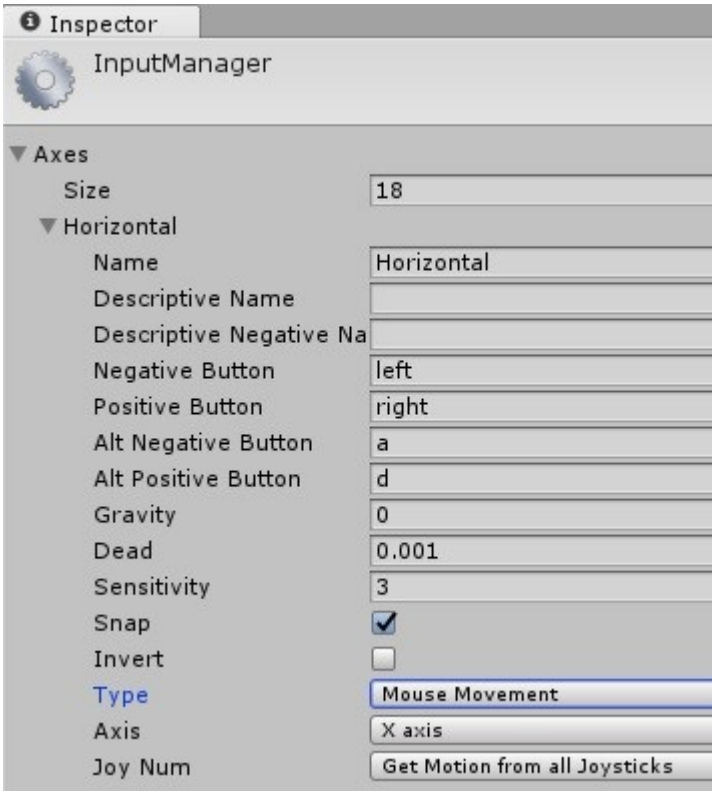
De **Linear Drag** bepaalt de vertraging van een Game Object als gevolg van wrijving met de lucht of water eromheen. De **Angular Drag** geldt voor rotatiebeweging en is gescheiden van de lineaire weerstand (Linear Drag) die de positiebeweging beinvloedt. Een hogere waarde van een hoek belemmering (Angular Drag) veroorzaakt een objectrotatie en een snellere rust na een botsing.
De **Gravity Scale** bepaalt de zwaartekracht van het Game Object. Staat deze niet op 0 maar op een positieve waarde, dan zal het object naar beneden vallen. Met deze instelling hoeft de gravitatie dus niet geprogrammeerd te worden. Het instellen van de waarde die u wilt is voldoende.

De **Collision Detection** bepaald hoe er gereageerd moet worden op botsingen. Er zijn twee instellingen: Discrete en Continuous. Voor de racket hoeft er niet per frame op botsingen gecontroleerd te worden. Daarom laten we deze op **Discrete** staan.
De **Sleeping Mode** bepaalt de ‘luiheid’ van het Game Object. De instellingen zijn: Never Sleep, Start Awake of Start Asleep. De racket moet meteen wakker worden zodra het project wordt gestart. Bij vloeren, zoals bij Mario, is de instelling Start Asleep beter. Er is dan geen kans dat een vloer na een botsing naar beneden valt of wordt weggeduwd. Dit heeft te maken met script gebruik. De methoden **IsKinematic** en **AddForce** hebben ermee te maken. Dit valt buiten de Bulletin, omdat het lastig is deze methoden uit te leggen en te laten zien wat er gebeurt. In BreakOut games, zoals in deze Arkanoid, zijn de methoden niet nodig.
Als laatste ziet u **Constraints**. Klik op het pijltje links van Constraints. Er zijn twee instellingen, **Freeze Position** en **Freeze Rotation**. Hier wordt het bevriezen van bewegingen van het Game Object be-

paald. Eerder heb ik u verteld dat de racket kan ‘vliegen’ als de snelheid in verwarring komt met de weerstand. In het script, dat later komt, zorg ik er voor dat de Y positie van de racket onderaan blijft, maar zonder het aanvinken van de Constraints opties kan de racket alsnog omkiepen. Dit geldt ook voor de bal. De bal moet een bevroren Z rotatie hebben, anders zal de bal ‘dronken’ (of tollend) over het scherm bewegen.

Het racket script
Als laatste van de tutorial deel 1 gaan we het script maken om de racket heen en weer te laten bewegen. In de Engelstalige tutorial worden toetsen gebruikt met een andere snelheid toekenning. Ik laat hier echter een mogelijkheid zien hoe we de muis kunnen gebruiken voor objectbeweging.

Voordat we het script gaan maken, moeten we eerst de muis instellen in de InputManager. De InputManager is bedoeld om allerlei invoermogelijkheden te kunnen kiezen voor spelbesturing. Ga naar het menu **Edit -> Project Settings**. Klik op **Input**. De Inspector verandert in de InputManager, zie hier rechts.



Het deel dat ik hier laat zien is voor de Horizontal instelling. Het **Type** dat blauw staat aangegeven is degene die u moet wijzigen. Klik op het balkje en kies **Mouse Movement**.

Klik in de Hierarchy weer op het racket Game Object om de racket Inspector te zien.
Klik met de rechter muisknop op de Assets. Klik op **Create** en kies **Folder**. Noem de map **Scripts**. Klik met de rechter muisknop op de Scripts map. Klik op **Create** en kies **C# Script**. Wijzig de naam in **Racket**. Ga naar de Inspector en klik bovenaan op de **Open…** knop.

Er bestaan twee soorten Unity Editors: de MonoDevelop editor en de Visual Studio Community editor. Beide hebben dezelfde C# script compiler, de code verschilt dus niet.

Unity maakt alvast een geraamte met onderstaande code:

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

public class Racket : MonoBehaviour {

	// Use this for initialization
	void Start() {

	}

	// Update is called once per frame
	void Update() {

	}
}
```

De void functie Start() hebben we niet nodig. U kunt deze gewoon verwijderen. De void functie Update() zouden we wel kunnen gebruiken, maar deze voldoet niet aan de wens om per frame voor een goede beweging te kunnen zorgen. Unity kent een andere Update() functie: FixedUpdate(). Wat is het verschil?
De standaard Update() functie wordt per frame uitgevoerd, maar de FixedUpdate() functie per stap. In een seconde kan de Update() functie meerdere keren uitgevoerd worden, met andere woorden, de uitvoer kan met FixedUpdate() variëren.

Voor een goed voorbeeld kunt u kijken en naar de uitleg luisteren op de Unity site. Ga naar
https://unity3d.com/learn/tutorials/topics/scripting/update-and-fixedupdate

Wijzig de Update() functie in de FixedUpdate() functie, maar als u experimenteren leuk vindt kunt u ook gewoon de Update() functie gebruiken om straks te kijken of er verschil is of niet. U kunt de functie zo vaak wijzigen als u wilt. Handig om Unity te leren en de functies te kennen.

Typ boven de FixedUpdate() functie onderstaande regel:

```
public float speed = 7;
```

Met een goede speed zal in de FixedUpdate() functie de juiste velocity (beweging) worden berekend samen met de Input waarde van de muis.

Typ in de functie onderstaande code:

```
float h = Input.GetAxisRaw("Horizontal");
GetComponent<Rigidbody2D>().velocity = Vector2.right * h * speed;
```

De GetAxisRaw() functie geeft de waarde terug met de richting van de muis. Gaat de muis naar links, dan geeft de functie een -1 terug. Gaat de muis naar rechts, dan geeft de functie een 1 terug. Een teruggegeven 0 betekent dat de muis geen beweging maakt.
Unity kent nog een functie, de GetAxis() functie. Deze functie geeft de waarden nauwkeuriger terug, namelijk tussen 0 en 1 en tussen 0 en -1. Voor de racket maakt het niet uit welke functie er gebruikt wordt. Met de muisbeweging zal het verschil nihil zijn.

Om de Vector.right waarde maal de input waarde maal de speed waarde toe te kennen aan de velocity, moeten we toegang krijgen tot de component Rigidbody. We kunnen niet direct als een object de Rigidbody component opgeven. We moeten, zogezegd, ernaar vragen. Met GetComponent<>() kunnen we elke component van het Game Object opvragen om toegang te kunnen krijgen naar de waarden van de component eigenschappen en methoden, zoals de velocity.

Klik bovenaan op de diskette knop om de code op te slaan en ga terug naar het Arkanoid project.

Start het project en probeer met de muis de racket heen en weer te bewegen. Vindt u de racket te snel gaan of te langzaam, dan kunt u de Speed wijzigen. U hoeft niet terug te gaan naar de code om de snelheid te wijzigen. In de Inspector van de racket kunt u ook de waarde van Speed wijzigen.

In deel 2 van de tutorial gaan we de bal toevoegen en laten botsen tegen de borderranden en laten we de bal stuiteren op de racket en gelijk de richting van de bal bepalen.

Tot de volgende Bulletin en experimenteer met Unity.

Excel – VBA leren.

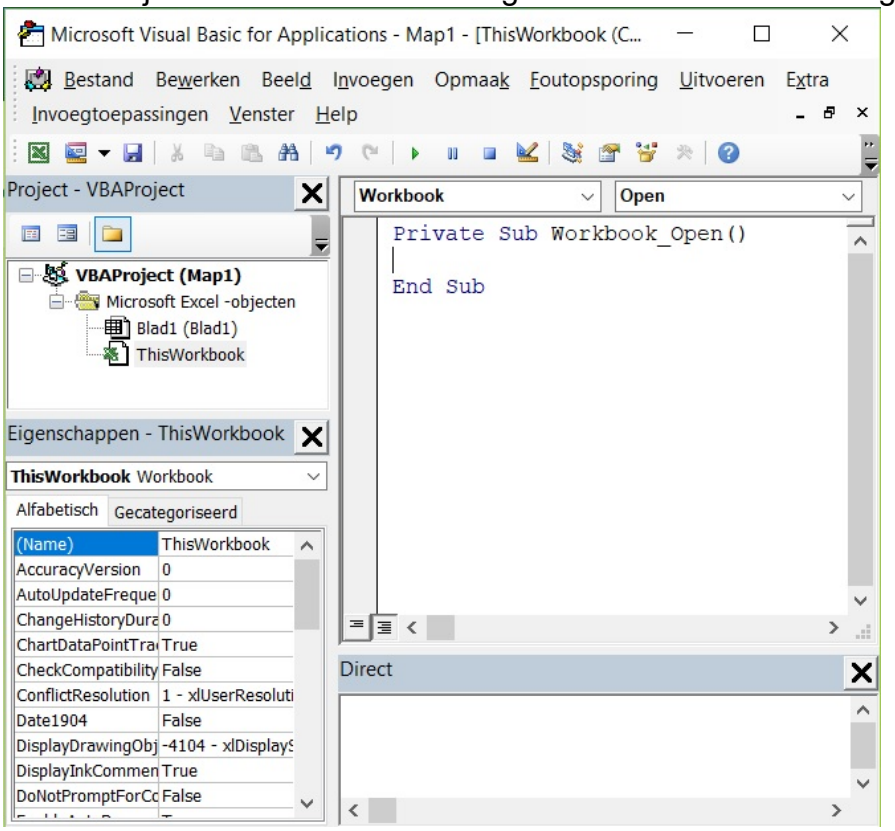
Een Excel werkboek kan in een snel tempo worden gemaakt. Met achtergrond code en misschien zelfs met userforms wordt het echter al een werkboek met denkwerk.

Er zijn drie manieren om Excel te gebruiken:

- alleen de voorgrond;
- alleen de achtergrond;
- beide.

Als u alleen de voorgrond gebruikt, moet u zich aan de eisen van Excel houden. Meestal gebruikt men beide, de voorgrond en de achtergrond. Een werkboek maken met alleen de achtergrond is een geavanceerd werkje. U zorgt voor uw eigen regels en u bouwt daarmee het werkboek.

ThisWorkbook
Hoewel het hoofdobject ThisWorkbook optioneel is voor gebruik, is het toch beter de initialisaties hierin te programmeren. Eventueel kunt u bepalen welk werkblad er als eerste geactiveerd moet worden. Als u dit bepaald via een macro, door bijvoorbeeld tijdens de opname Blad2 te activeren, zult u ontdekken dat de macro niet automatisch werkt als u uw werkboek opent. ThisWorkbook heeft werkboekevents om voor deze acties te kunnen zorgen. Het is dan ook mogelijk om een event te kunnen starten die objecten eerst moet vernietigen voordat het werkboek gesloten wordt.



Dubbelklik op ThisWorkbook. Rechts ziet u het codevenster van ThisWorkbook, nog leeg. Klik op de balk waar Algemeen staat. U ziet Workbook verschijnen. Klik daarop.
Meteen ziet u onderstaand event in het codevenster verschijnen.

```
Private Sub Workbook_Open()

End Sub
```

VBA is geen scripttaal. We kunnen projecten naar eigen hand schrijven. VBA heeft ook een IDE en een compiler. Het enige verschil met Visual Basic is dat we geen EXE bestand van het project kunnen maken.

Nu komen allerlei vragen. Wat gaan we doen? Hoe doen we dat? Wat hebben we nodig? Hebben we ook Excel nodig voor ondersteuning?

Alleen VBA gebruiken is niet altijd zinvol. Excel heeft functies die in VBA niet bestaan, maar VBA heeft wel weer methoden en handigheden die we niet in Excel kunnen gebruiken. Laten we VBA eens goed leren voordat we aan een project gaan beginnen.

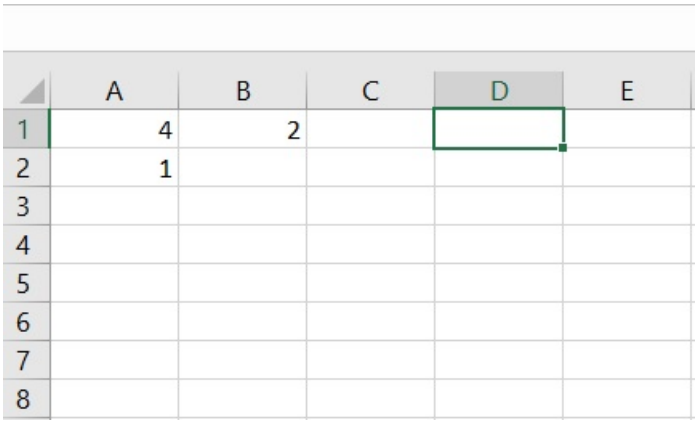
Eén van de bekendste problemen in Excel: cellen naar zichzelf verwijzen werkt niet
Een voorbeeld van een groot probleem in Excel is de manier hoe we de cellen gebruiken. Graag zouden we een waarde in een cel met zichzelf plus 1 op willen tellen, zoals we dat in programmeertalen kunnen doen:

```
A1 = A1 + 1
```

Typen we in cel A1 de expressie =A1+1, dan geeft Excel meteen een foutmelding dat we cellen niet naar zichzelf kunnen laten verwijzen. Willen we dit toch doen, dan zouden we VBA code moeten schrijven met iets soortgelijks. De code moet dan ergens uitgevoerd kunnen worden, bijvoorbeeld zodra op cel A1 geklikt wordt of een gewijzigde selectie wordt gemaakt. We kunnen onderstaande code in Blad1 programmeren, dat op een A1 = A1 + 1 lijkt:

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
    Target.Value = Target.Value + 1
End Sub
```

Onderstaand voorbeeld laat zien dat alleen maar die regel perfect werkt. Excel stribbelt niet tegen en telt elke waarde in een cel op met 1 zodra we met de muis een cel selecteren (of aanklikken).



	A	B	C	D	E
1	4	2			
2	1				
3					
4					
5					
6					
7					
8					

Omdat niet een constante cel gegeven wordt, maar de doorgegeven Target wordt gebruikt, zal er in elke cel waar we komen een nieuwe waarde komen te staan.

Op die manier kunnen we ook spelen met VBA, bijvoorbeeld het telkens verhogen van het alfabet zodra op een cel wordt geklikt.

Wijzig bovenstaande code naar de code hieronder:

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
    If Target.Row <> 3 And Target.Column <> 1 Then
        Target.Value = Target.Value + 1
    Else
        If Target.Value >= Chr$(65) And Target.Value < Chr$(90) Then
            Target.Value = Chr$(Asc(Target.Value) + 1)
        End If
    End If
End Sub
```

Typ in cel A3 de letter A en klik daarna op een andere cel. Zie wat er gebeurt zodra u weer op cel A3 klikt. De letter B zal dan verschijnen.

De code werkt als volgt: eerst wordt gecontroleerd of de celselectie niet cel A3 is, om er voor te zorgen dat we in andere cellen nog steeds op kunnen tellen. Is het wel cel A3, dan wordt er gekeken of er een geldig ASCII teken in de cel staat dat in het alfabet voorkomt. Merk op dat voor het verhogen van het alfabet eerst de waarde van de cel geconverteerd moet worden naar de ASCII code. Vergeet u de Asc() functie, dan zal de Chr$() functie niet werken, omdat de waarde van de cel een teken bevat en geen getal.

Het Range() object
Het SelectionChange() event heeft een Range type argument waarmee we toegang hebben tot de cellen van het blad. Helaas kunnen we met de target alleen toegang krijgen tot de cellen die geselecteerd zijn. Willen we vrij kunnen werken met de cellen, dan moeten we zonder de target toegang krijgen tot de cellen van het blad. Met behulp van het Range() object is dat mogelijk.

Het Range() object kan voor een heel cellenbereik worden gebruikt. Als we dat niet willen, kunnen we ook een eigenschap van het blad gebruiken. Met de eigenschap Cells() hebben we toegang tot één cel, maar er zijn dan geen extra celmogelijkheden, zoals het instellen van de aligning en de opmaak.

Onderstaande regels doen beide hetzelfde:

```
Blad1.Range("A4").Value = 100
Blad1.Cells(4, 1) = 100
```

Merk op dat de parameters van Cells() anders zijn. Eerst de rij en dan de kolom, alsof er 4A staat.

Typen we onderstaande regel:

```
Blad1.Range("A10:A15").Value = 50
```

Dan zien we op het werkblad van regel 10 tot en met regel 15 in kolom A allemaal 50'jes staan.

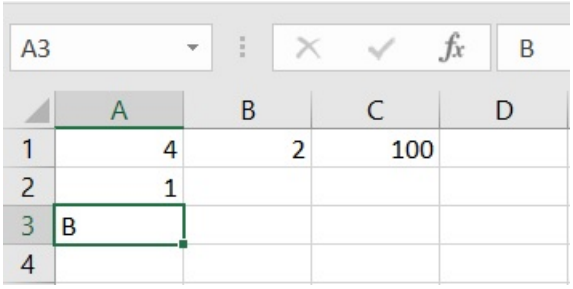
Het Range() object biedt nog meer functionele mogelijkheden. Weer een truc die niet op het blad van Excel mogelijk is. Zouden we een bereik selecteren en ervoor willen zorgen dat we alleen in die selectie de gegevens willen beheren, dan kunnen we niet de gegevens in de selectie nog eens selecteren. Een selectie nesten is in Excel niet mogelijk, althans niet op de voorgrond.

Dankzij het Range() object kunnen we relatief met de cellen werken. Hoe raar het ook is: cel A1 hoeft niet per se cel A1 te zijn, het kan relatief ook cel C10 zijn!

Maak een macro aan door eerst een nieuwe module toe te voegen. Laat de naam gewoon Module1 heten.
Typ onderstaande subroutine in de module:

```
Public Sub RangeTest()
    Blad1.Range("C1:D4").Range("A1").Value = 100
End Sub
```

Ga naar het tabblad Ontwikkelaars en kies Macro's. U ziet in de lijst RangeTest staan. Klik op de knop Uitvoeren. U ziet direct het getal 100 in cel C1 verschijnen, maar niet in cel A1!



U kunt net zoveel Range() objecten nesten als u zelf wilt. Het is bijvoorbeeld mogelijk om één werkblad te verdelen in meerdere werkbladen. Door deze werkbladen te bewaren in eigen Range() instanties, kunt u uw eigen Excel project structureren zodat het werken met grote werkbladen veel gemakkelijker wordt.

Naast de eigenschap Value is er ook een eigenschap Value2 die op dezelfde manier werkt, behalve de eigenschap Text. De eigenschap Text is alleen lezen.

Hoe dan ook, bovenstaande VBA mogelijkheden kunnen ook niet automatisch aangemaakt worden tijdens het opnemen van een macro. Zouden we een selectie maken en op de cel linksboven in de selectie een waarde invoeren, dan zal de macro niet de cel relatief opnemen en geen A1 aangeven.

Waarom C cryptisch is.

Al vanaf jaren ’90 werd programmeertaal C als een cryptische taal gezien. Het was een programmeertaal met pointers en adresverwijzingen. Gewone string types waren er nog niet en we moesten een array van tekens declareren of een pointer aanmaken die als een adresarray werkte (char*). Het laatste teken was altijd een null-teken om ervoor te zorgen dat de pointer de juiste reservering had.

Maar het gebruik van pointers was, en is, niet het enige probleem waardoor programmeertaal C cryptisch en moeilijk wordt gezien. Het gaat vooral om de structuur, de operatoren en de expressies die heel anders kunnen werken dan in andere programmeertalen.

Programmeertaal C was in de jaren ’90 niet mijn favoriete programmeertaal. Ik bleef het liefst werken in Basic. Vanaf twee jaar geleden ben ik echter naar C# gegaan vanwege het gebruik van XNA en Unity.

Er zijn twee operatoren in C die, ook in C# en in C++, op een eigenaardige manier kunnen werken. Visual Basic .NET kent een deel van die operatoren, die in versie 6 nog niet bestonden. Onderstaande operatoren bestonden al in C:

- **++**
- **--**
- **+=**
- **-=**
- ***=**
- **/=**
- **^=**

En misschien nog wel meer.

De eerste twee operatoren zijn de operatoren waar ik het in dit onderwerp vooral over wil hebben. Ze bestaan in C/C#/C++, en in diverse andere programmeertalen. De volgende twee kunnen in Visual Basic op dezelfde manier worden gebruikt als de eerste twee, door de waarde met 1 op te tellen of af te trekken. De rest van de operatoren bestaan ook in Visual Basic.

Wat is er anders aan de eerste twee operatoren?
We kunnen de eerste twee operatoren omdraaien.

Een regel als: **L++;** werkt hetzelfde als **++L;**, dit geldt ook voor de **--** operator.

Hebben we een regel als: **L = L + 1**, dan werkt het ook net zo. Er is echter een verschil: hier kunnen we niet spreken van een enkele operator, maar een expressie resultaat die toegekend wordt aan variabele **L**. Omdat een operator als een expressie mag werken, kunnen we onderstaande C code uitvoeren:

```
J = 5;
I = 5;
J = I++;
// typ hier het statement om de waarden J en I af te drukken
```

Typ waar het commentaar staat het statement dat u nodig heeft om de waarden weer te geven.
Als u de code start, zult u zien dat onderstaande waarden verschijnen:

```
5
```

```
6
```

Ook al wordt er een `I++` uitgevoerd, toch wordt de waarde 5 weergegeven van variabele J.
Wijzig de derde regel zoals hieronder:

```
J = ++I;
```

Nu zullen de waarden anders zijn, namelijk:

```
6
6
```

Wat betekenen de operatoren eigenlijk, en hoe komt het dat er verschil ontstaat?
Dit heeft te maken met de benamingen *prefix en postfix*, ook genoemd: before en after. De eerste voorbeeldregel heeft een postfix, waardoor de optelling later is dan de toekenning aan variabele J. De andere voorbeeldregel is een prefix, waardoor eerst de optelling begint en dan pas het toegekend wordt aan variabele J.

Helaas kan ik niet uitgebreid een onderwerp vertellen over deze programmeertaal. Misschien dat ik later nog eens terugkom met C.

Natuurlijk kom ik wel met C# van deze tijd met de onderwerpen over Unity.