Programmeren Bulletin

hcc! programmeren

Interessegroep

26ste jaargang najaar 2019 Nummer 2

Inhoud

Redactioneel

Programmeren is geheimen onthullen en nieuwe vaardigheden ontwikkelen. Wie dat interesseert kan ver komen. Door veel uit te proberen kan de kortste weg worden ontdekt. Er kan ook een route doodlopen. Het is dan uitzoeken wat er mis gaat. Maar wie de programmeertaal goed kent en de puntjes op de i heeft, kan altijd een weg weer terugvinden.

Marco Kurvers

Functies en formules in Excel

Ook al kunnen we met VBA veel besturing maken voor de werkmap, toch zijn er Excel functies die niet in VBA aanwezig zijn. Deze functies kunnen we wel in VBA uitvoeren door ze in tekstformaat op te geven.

De functies bestaan dus niet als VBA methoden. Een VBA functie als SIN() kan wel op beide manieren werken. De functie Sin() staat namelijk ook in de functielijst in Excel.

Door deze functies vanuit Excel in VBA aan te roepen, hoeven we ze niet te programmeren. Een functie als Som.Als() is zeer handig om waarden uit een lijst te filteren door middel van een opgegeven voorwaarde.

We zouden bijvoorbeeld de Som() functie kunnen namaken in VBA.
Hebben we 10 rijen in kolom A, dan kunnen we in rij 11 de Som() functie plaatsen die alle 10 rijen optelt. In Excel hoeven we alleen maar op te geven:

```
=SOM(A1:A10)
```

Maar in VBA hebben we een lus nodig die herhaalt van rij 1 t/m rij 10, zoals onderstaande code doet.

```
N = 0
For I = 1 To 10
     N = N + Range("A" & I).Value
Next I
Range("A11").Value = N
```

In plaats daarvan zouden we ook de Som() functie kunnen aanroepen:

```
Range("A11").Value = "=SUM(A1:A10)"
```

Houd er rekening mee dat je de functie Engelstalig in een string moet opgeven. Als je het in het Nederlands opgeeft, dan zal Excel een #NAAM? melding in de cel plaatsen, ook al zal de SOM() functie in de formulebalk juist zijn. Klik je op de formulebalk en druk je op Enter, dan zal de som correct worden uitgevoerd.

Het nadeel is dus dat als je een functie in een string wilt gebruiken, eerst moet weten hoe de functie in het Engels wordt genoemd. Je kunt een functie in de categorie kiezen tijdens het opnemen van een macro. Wanneer de macro is opgenomen, kun je in VBA kijken hoe de functie in het Engels wordt genoemd.

Tip: Er is een lijst van alle functies te vinden op:
https://www.excel-function-translation.com/index.php?page=nederlands-english.html

Zo heet de functie Gemiddelde() in VBA: Average(). Maar dan geen echte functie, maar een gegeven string die door Value wordt uitgevoerd.

Formules

Excel heeft ook een functie als Rente() met een formule Getal% / N. Handig, want dan hoef je zelf niet de procenten te berekenen. Maar wat als je wilt weten hoeveel procent een bepaalde waarde is van een totale waarde? Dat is niet hetzelfde als het berekenen van een kortingsprijs. Een kortingsprijs is bijvoorbeeld:

```
Kortingsprijs = TotalePrijs - TotalePrijs / 100 * Procentwaarde
```

Een voorbeeld:

```
Prijs: € 2,39
Korting: 35%
Kortingsprijs: € 1,55

Kortingsprijs = 2.39 – 2.39 / 100 * 35 = 1.5535
```

Om te weten hoeveel procent een bepaalde waarde is van een totale waarde, moet er een andere formule worden gebruikt. Installatieprogramma's doen dat ook. Die laten een balk zien zodat de gebruiker weet hoe ver een installatieprogramma is. Ook in games wordt het gebruikt, voor het bijhouden van de health en hoeveel monsters er nog zijn als een speelveld wordt verlaten.

We typen in cel A1 het getal 400 en in cel A2 het getal 398. In cel B1 typen we onderstaande formule:

```
=A2/(A1/100)   ' of wat ook kan: =A2/A1*100
```

Het resultaat in B1 zal zijn: 99,5. Dus 99,5% van 400 = 398.

Unity – Experimenteren met Game Objecten

Voordat we een asset, zoals een sprite, op een scene kunnen plaatsen, hebben we een object nodig. Een asset heeft namelijk geen inhoud om mee te werken. Game objecten hebben componenten met instellingen en besturing.

Om te ervaren hoe een game object werkt, is het verstandig om eerst te leren er mee om te gaan. Plaats gewoon een paar game objecten in de Hierarchy en wijzig de instellingen om te zien wat er gebeurt. Laten we eens in dit onderwerp een oefening maken.

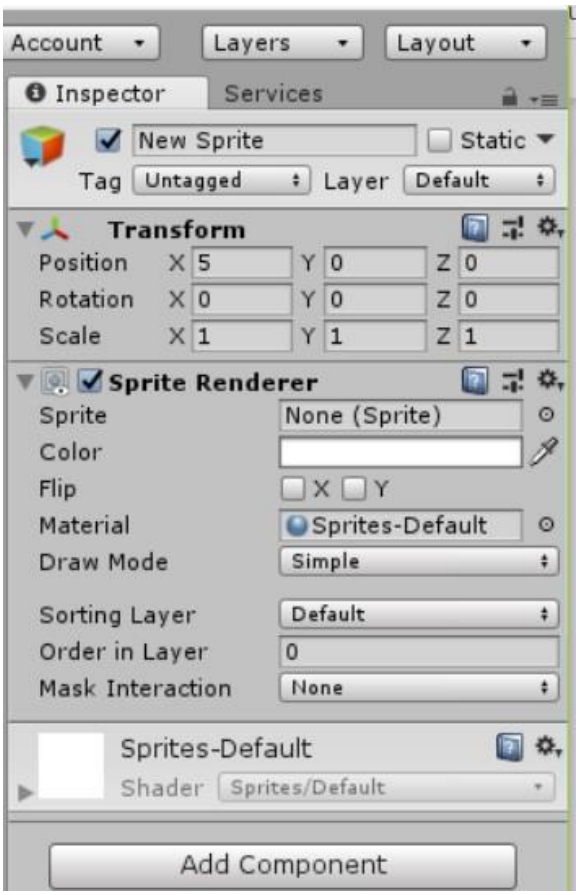
2D of 3D of beide

Voordat we een nieuw project gaan openen, moeten we kiezen in welke dimensie we willen werken. Unity kent 2D en 3D, maar het is mogelijk om ook in beide dimensies te werken. Dat kan alleen in het project zelf worden gekozen. In de volgende Bulletins gaan we daarmee verder.

Een oefening met game objecten

Kies 2D en open het project. Je ziet een 2D project met een platte 2D scene. Het is plat, omdat we voor 2D hebben gekozen en er geen diepte is.

We kunnen nu game objecten toevoegen door naar het GameObject menu te gaan en te kiezen voor 2D Object. Deze game objecten hebben geen inhoud en geen vorm. We kunnen alleen kiezen voor een sprite aan te maken. Als we dat doen, zien we op de scene echter niets. Het game object heeft twee componenten in de Inspector, de Transform en de Sprite Renderer. De Transform bepaalt de plaats, de rotatie en de grootte van het object. De Sprite Renderer zorgt voor het weergeven van de sprite, maar niet het object zelf. Het object wordt namelijk al als een game object weergegeven. Wat Sprite Renderer doet is er voor zorgen dat de sprite met de juiste instellingen, zoals de kleur en de soort sprite (vorm of texture) gerendeerd wordt.



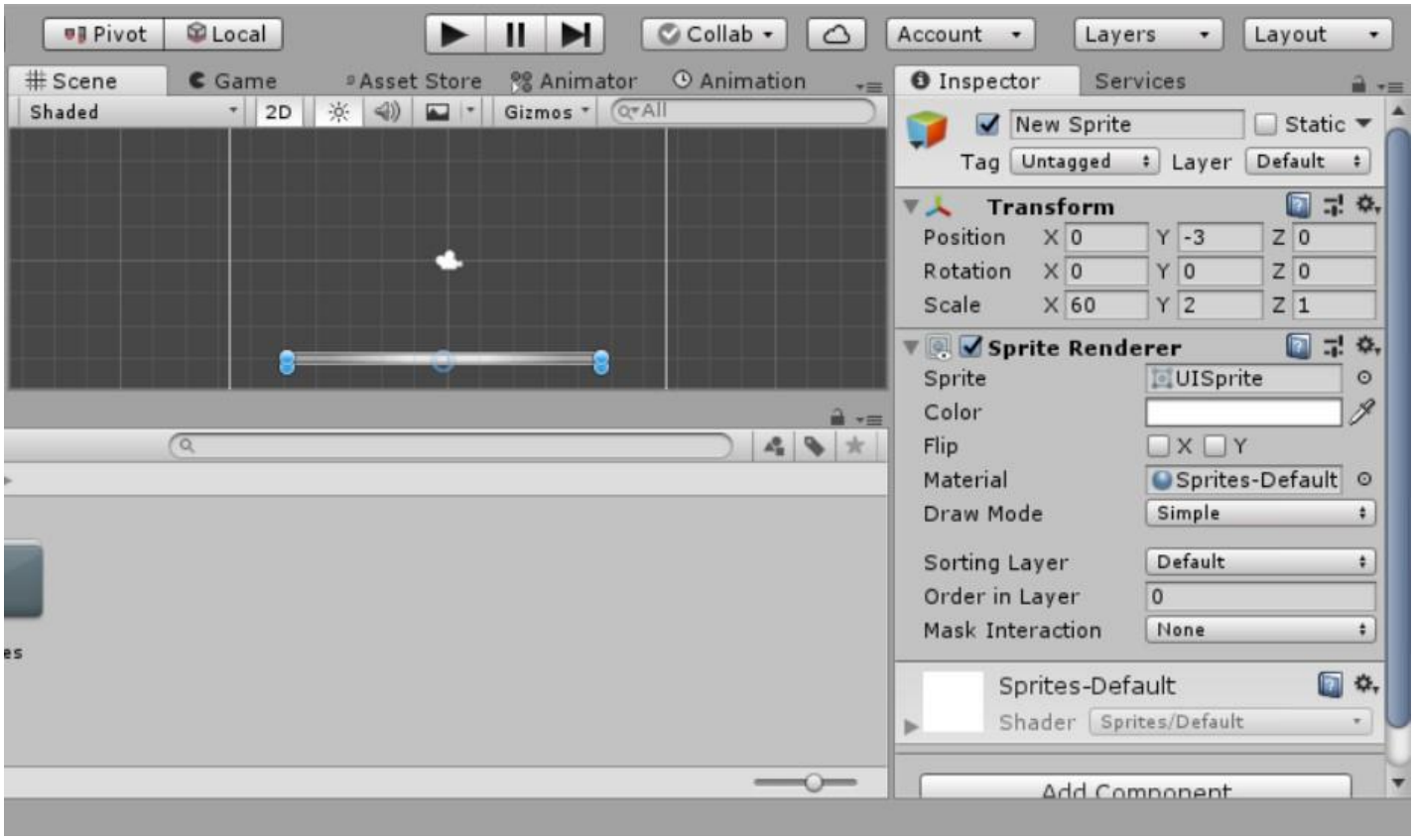
Klik naast None (Sprite) op het wieltje, zie pijltje 1.

←1

Een venster gaat open met een keuze uit voor gedefinieerde spritevormen. Kies de UISprite en sluit het venster. None (Sprite) verandert in UISprite. De sprite zien we nog steeds niet goed. De scale is te klein om de sprite te kunnen zien.

Wijzig Scale X in 60 en Scale Y in 2. Wijzig Position X in 0 en Position Y in -3. Negatieve waarde betekent naar beneden.

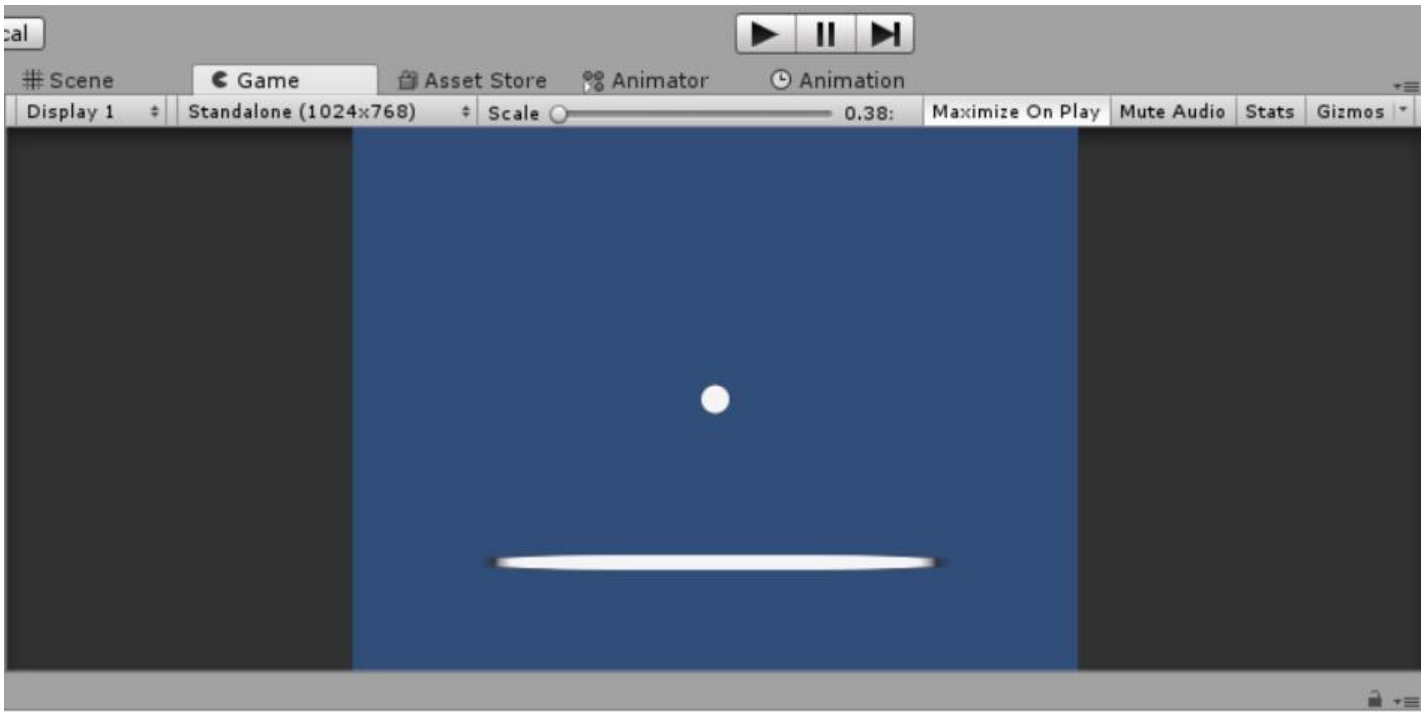
Hieronder zie je het game object, weergegeven met de gewijzigde instellingen.



Laten we nu nog een game object toevoegen. Kies weer een 2D Object in het GameObject menu en klik weer in de Sprite Renderer op het wieltje, zie eerste afbeelding.

Kies nu voor de Knob sprite om een cirkel weer te geven. Wijzig de Position X, Y en Z in waarden 0 en wijzig de Scale X en Y in waarden 3. In de scene is het niet helemaal goed te zien, maar als je naar het tabblad Game gaat, zie je onderstaande afbeelding.

2



Als we op de Play knop klikken, zie de afbeelding hiervoor, de knop bij nummer 2, dan gebeurt er niets. Logisch, want er is geen besturing. Laten we de bal eens naar beneden vallen. Dat kan heel gemakkelijk door een component toe te voegen die de zwaartekracht bevat.

Klik op het tabblad Scene en klik op het tweede game object. Je kunt de game objecten ook een naam geven door met de rechter muisknop te klikken en te kiezen voor Rename. Heb je geklikt op het bal game object, dan gaan we in de Inspector het volgende doen:

Klik onderaan op de knop Add Component en kies de categorie Physics 2D. Zoek in de lijst naar Rigidbody 2D en klik deze.

Het component heeft heel veel mogelijkheden die met vallen, glijden, wrijving en druk te maken hebben. De Rigidbody kan dat niet helemaal alleen. Er moet een Physics Material in de Material eigenschap worden toegevoegd dat voor een bepaald karakter moet zorgen. Eerst laten we de bal naar beneden vallen. We hoeven daar niets voor te doen, want de zwaartekracht staat al ingesteld. Dit kun je zien door te kiezen: Edit -> Project Settings -> Physics 2D. De gravity staat standaard ingesteld op -9.81. De waarde is negatief, want anders zouden de objecten niet naar beneden vallen.

Wijzig Position Y in waarde 4. De bal staat dan wat hoger. Start het project door op Play te klikken, zie nummer 2 bij de afbeelding.
De bal zie je naar beneden vallen, maar valt verder dan waar de vloer ligt. Het lijkt ook alsof de bal er niet meer is, maar de bal blijft gewoon vallen buiten het zicht van de camera. Je kunt dat zien door op het Bal game object te klikken, Maximize On Play uit te klikken (zie vorige afbeelding) en het project nog eens te starten. Je ziet de bal vallen, maar je ziet ook dat de positie in de Inspector blijft veranderen, omdat de bal blijft vallen.

Om de bal niet door de vloer te laten vallen, hebben we colliders nodig. Een collider is een beschermde rechthoek om het game object. Deze zorgt voor botsingeffecten, vooral samen met de Rigidbody, waarmee we glij- en stuitereffecten kunnen maken. Laten we eerst eens zien wat de bal doet als we een collider component in het Vloer game object toevoegen.

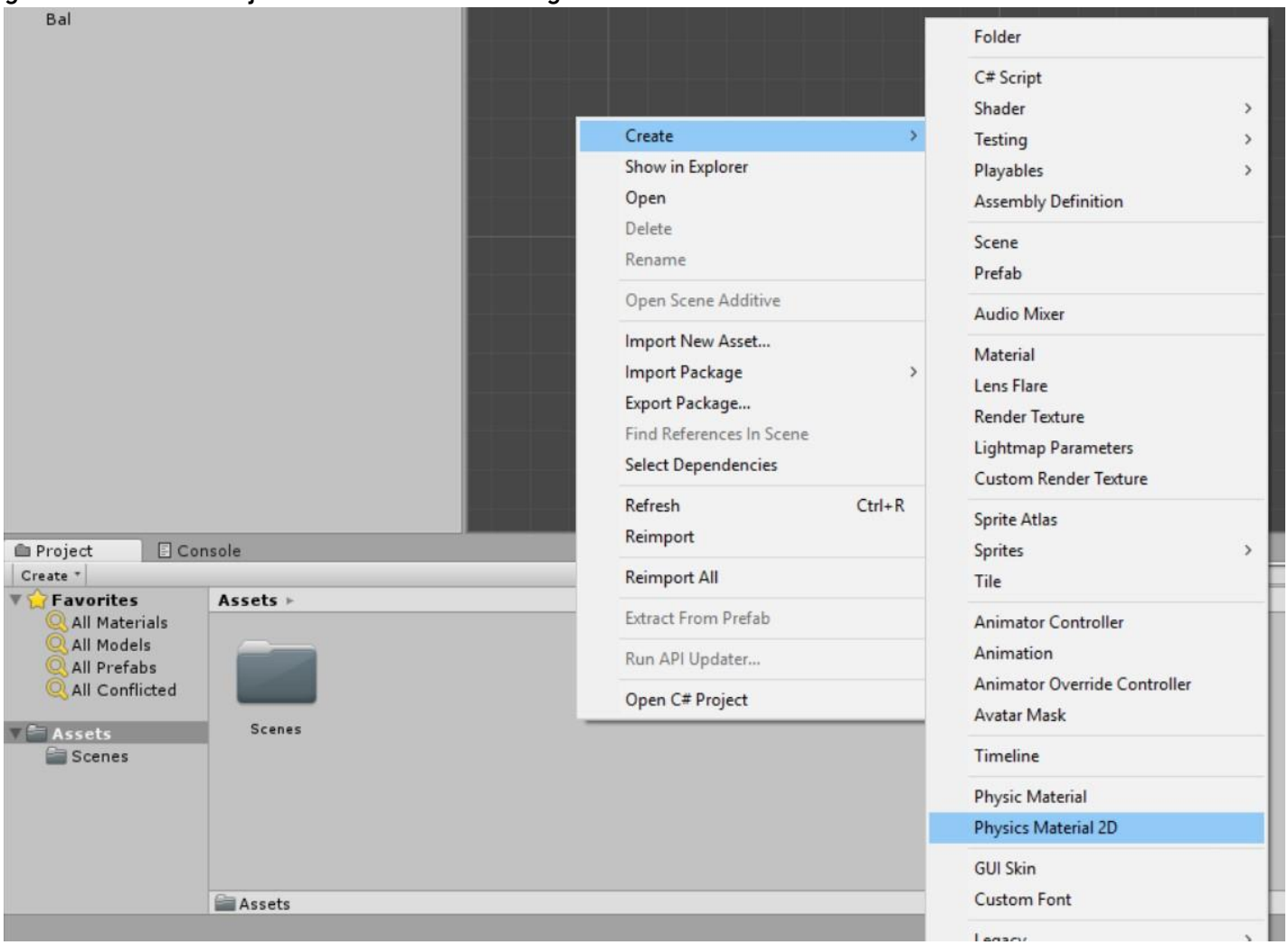
Klik op het Vloer game object en klik op Add Component. Kies weer in de lijst de categorie Physics 2D en zoek naar: Box Collider 2D. Klik op deze.

Klik weer op Play. De bal gaat weer door de vloer alsof er helemaal geen collider is. Die is er wel, maar de bal moet ook een collider hebben. Net als een geest die zonder collider door een muur gaat.

Voeg weer een collider toe, maar nu in het Bal game object. Kies nu niet de Box Collider 2D, maar de Circle Collider 2D. Deze collider zorgt voor een precies botsingeffect, zoals in snooker games, pinbal games en brick breaker games.

Schakel de Maximize On Play knop weer aan in het Game tabblad en start het project door op Play te klikken. De bal valt nu niet meer door de vloer, maar stuiteren doet de bal ook niet. Dat komt doordat er geen botsingeffect is. Voordat we het effect toe kunnen voegen, moeten we het als een

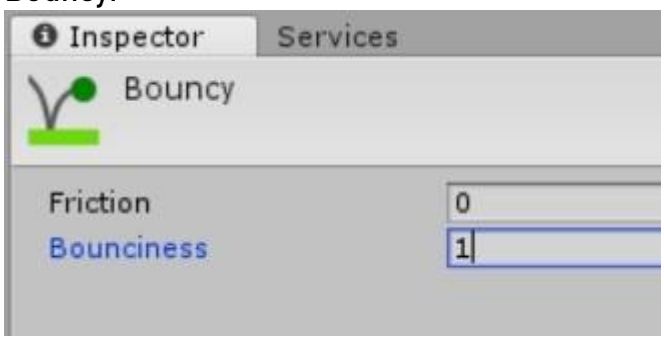
asset creëren. Niet als een sprite of een texture, maar als een materiaal. Onderstaande afbeeldingen laten zien hoe je het materiaal toevoegt.

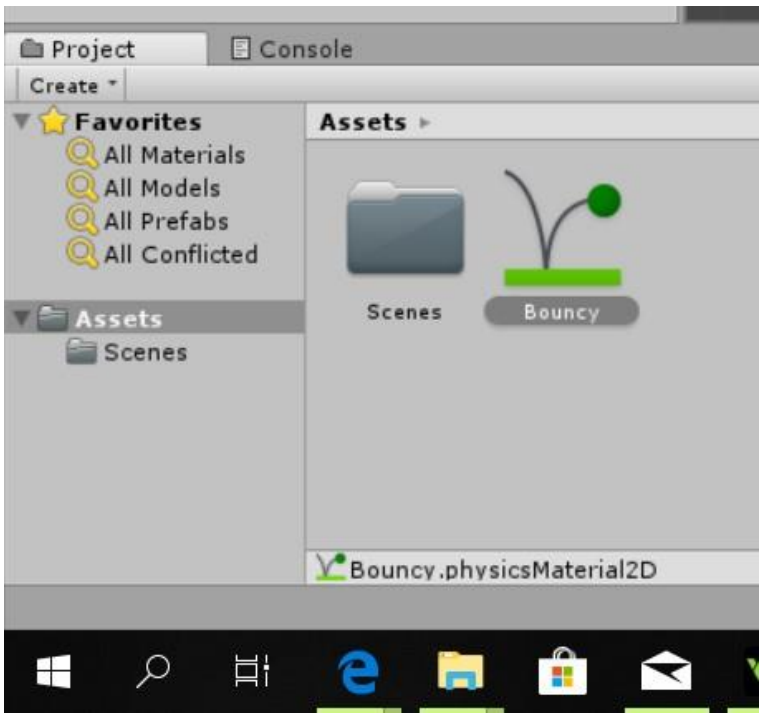


Zoals je ziet klik je met de rechter muisknop op Assets. Klik op Create en kies Physics Material 2D, zie de blauwe selectie.

In de Assets map verschijnt het materiaal. Wijzig de naam in Bouncy, door met de rechter muisknop er op te klikken en Rename te kiezen.

In de Inspector zie je twee instellingen van Bouncy.





Friction

Deze instelling zorgt ervoor hoe zwaar het object is. Hoe hoger de friction, des te meer wrijving er ontstaat tijdens een botsing met een ander object.

Bounciness

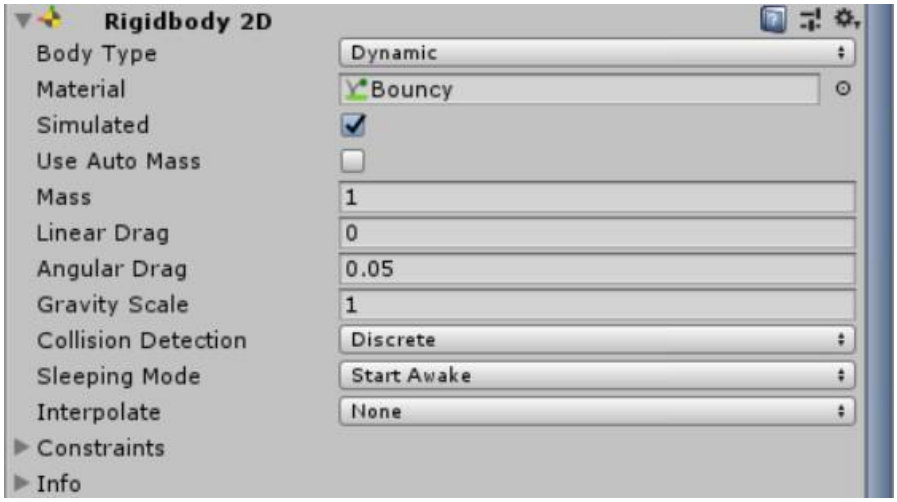
Deze instelling geeft een tegengesteld effect. De naam van de optie zegt het al: de waarde bepaalt de tegengestelde kracht van het object, waardoor het object kan glijden of stuiteren aan de hand van wat de waarde van Friction is. Een voorbeeld: hoe lager de Friction en hoe hoger de Bounciness, hoe krachtiger het object zal stuiteren. Is de Friction wat hoger, dan zal het object nog steeds stuiteren, maar zal ook sneller weer naar beneden vallen.

Echter kan het gebeuren dat er geen verschil ontstaat bij het wijzigen van de friction. Dat heeft te maken met hoe het object in aanraking komt met een ander object. Staat het andere object parallel met het object dat de instelling heeft, dan zal er weinig wrijving of glijding zijn. Staat het andere object niet parallel, dan kan de friction gaan werken.

Voordat we gaan kijken hoe het werkt, moeten we eerst het Bouncy materiaal toevoegen aan het game object dat het nodig heeft, maar welke?

Gooi maar eens een bal tegen de muur. De muur doet niets, maar de bal wel. De bal moet dus het materiaal hebben voor het botsingeffect.
De bal dus, maar naar welk component? Hoewel dit niet zal werken zonder de circle collider, heeft toch die component geen invloed op het effect. Wel de Rigidbody, omdat die component de body (het lichaam) van de bal is.

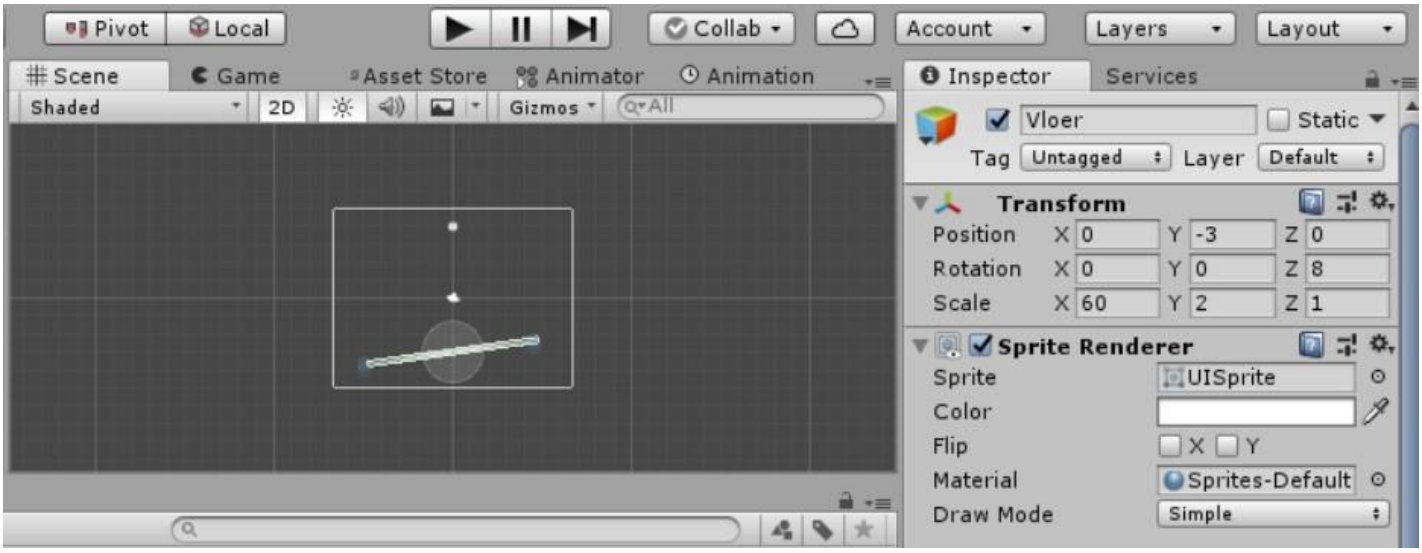
Klik op Bouncy, houd de linker muisknop ingedrukt en sleep deze naar Material in de Rigidbody 2D, zie afbeelding.



De Friction en Bounciness zijn niet de enige opties voor de body. De Mass, Linear Drag en Angular Drag geven extra effecten op het gedrag van het object.

Zijn de Friction en Bounciness ingesteld zoals de voorgaande afbeelding laat zien, start dan het project om te zien wat er gebeurt.

Je ziet de bal stuiteren op de vloer. Laat maar eens even doorgaan en merk op dat de bal steeds hoger stuitert. Dit heeft te maken met de force, de kracht die de bal heeft.
Stop de uitvoer en klik op het Bouncy materiaal in de Assets.



Wijzig de Friction in 10 en start het project opnieuw. Er lijkt geen verschil te zijn, zoals eerder uitgelegd, wanneer beide objecten parallel aan elkaar zijn. Daarom gaan we eens wat experimenteren.

Klik op de Vloer GameObject. Wijzig de rotatie Z van de vloer in 8. In 2D gebruiken we niet de Z-as voor de positie, maar wel om objecten te roteren.

Start het project. Je ziet de bal een mooie botsing maken en met een scherpe zwaai naar beneden vallen. Maar dat met zo'n hoge Friction? Ja, maar dat komt weer doordat de Bounciness op 1 staat.

Wijzig de Friction in 0 en start nog eens. Nu zie je de zwaai minder scherp, waardoor de bal veel verder naar beneden valt, namelijk buiten de camera.

Laten we eens kijken hoe de bal reageert met een andere Bounciness. Wijzig deze in 0.5 en start het project. Nu stuitert de bal twee keer op de vloer naar beneden en niet meer met een flinke zwaai. Wijzig de Friction ook nog eens in 8 en start het. De situatie lijkt hetzelfde, maar het stuiteren op de vloer is nu meer dan twee keer.

Je mag de Bounciness ook hoger dan 1 instellen, maar de verhouding met de val en de botsing van de bal zal dan niet meer gelijk zijn. Probeer het zelf eens. Deze situaties zullen de natuurwet overschrijden, maar dat zal voor Unity niets uitmaken. Tenzij je een object wilt laten springen op een springmat of springkussen, of beter: een flipperkast die dan een extra stoot geeft waardoor het object een grotere snelheid krijgt. Natuurlijk zullen in de programmeerwereld niet de flippers die kracht hebben. Wij weten nu dat we de speler voor de gek houden en we alleen maar een hogere Bounciness hoeven te geven als we zoiets willen doen. Maar het kan ook anders. Wanneer deze effecten elke keer moeten variëren, moeten we zelf code schrijven om via de Rigidbody de stoot te kunnen geven. De methode AddForce() zorgt daarvoor. In de volgende Bulletin gaan we wat experimenteren met de scripts van de game objecten.

Code in- en uitvouwen

Zoals we nu de code schrijven ziet het programma eruit als een structuur met ingesprongen codeblokken. Soms kunnen de codeblokken zeer ver rechts op de regels komen. Bovendien moeten we door de code bladeren als we een paar regels willen wijzigen. We hebben dan even geen belang bij de andere regels code.

In sommige editors is er een mogelijkheid om veel code in te krimpen, of beter gezegd: in te vouwen. Je kunt dan codeblokken verbergen, net zoals je alleen een map ziet in de verkenner zonder de inhoud van de map.

Er zijn twee manieren om code in- en uit te kunnen vouwen.

- Gebruik van de plus of min links van de code, bijvoorbeeld bij functies.
- Gebruik van een directive die zelf een blok aanmaakt met een min-teken, zoals een *region* directive om de code overzichtelijker te kunnen maken. Niet alle editors ondersteunen dat.

Onderstaande C# code laat zien dat de eerste mogelijkheid niet bij alles werkt. Lussen kun je bijvoorbeeld niet invouwen, maar wel als je één of meerdere lussen in een region zet.

```
-     void Test () {
          int r = -4;
          for (int i = 0; i < 50; i += 2)
          {
              r = (r + i) * 2;
          }
-         #region TestLus
          r = -4;
          for (int i = 2; i <= 50; i += 2)
          {
              r = (r + i) * 2;
          }
          #endregion
      }
```

Klikken we op de min voor TestLus, dan zie je dit:

```
-     void Test () {
          int r = -4;
          for (int i = 0; i < 50; i += 2)
          {
              r = (r + i) * 2;
          }
+         TestLus
      }
```

Ook Visual Basic ondersteunt beide mogelijkheden.

Textures, 3D engine en ACS scripts

Er wordt soms gedacht: "Als je programmeert, gebruik je code." Dat hoeft echter niet zo te zijn. Ook iets ontwikkelen in een visuele 3D omgeving betekent dat je programmeert.

Er zijn verschillende soorten 3D programma's waarmee we voor 3D kunnen ontwikkelen. Maar dat betekent niet dat het in elk programma hetzelfde werkt. In het ene programma hoef je bijvoorbeeld minder code te gebruiken dan in het andere programma en in sommige ontwikkelprogramma's helemaal niet. Eén ding is zeker: je bent aan het programmeren.

Unity

Wie de onderwerpen over Unity gelezen heeft, weet dat we visueel gemakkelijk een omgeving kunnen maken met 3D Game Objecten. We hebben echter veel code nodig om in de omgeving leven toe te passen. Beweegbare objecten, wind en water. Maar ook leven inblazen om deuren te kunnen openen, liften op en neer te bewegen, en nog veel meer.

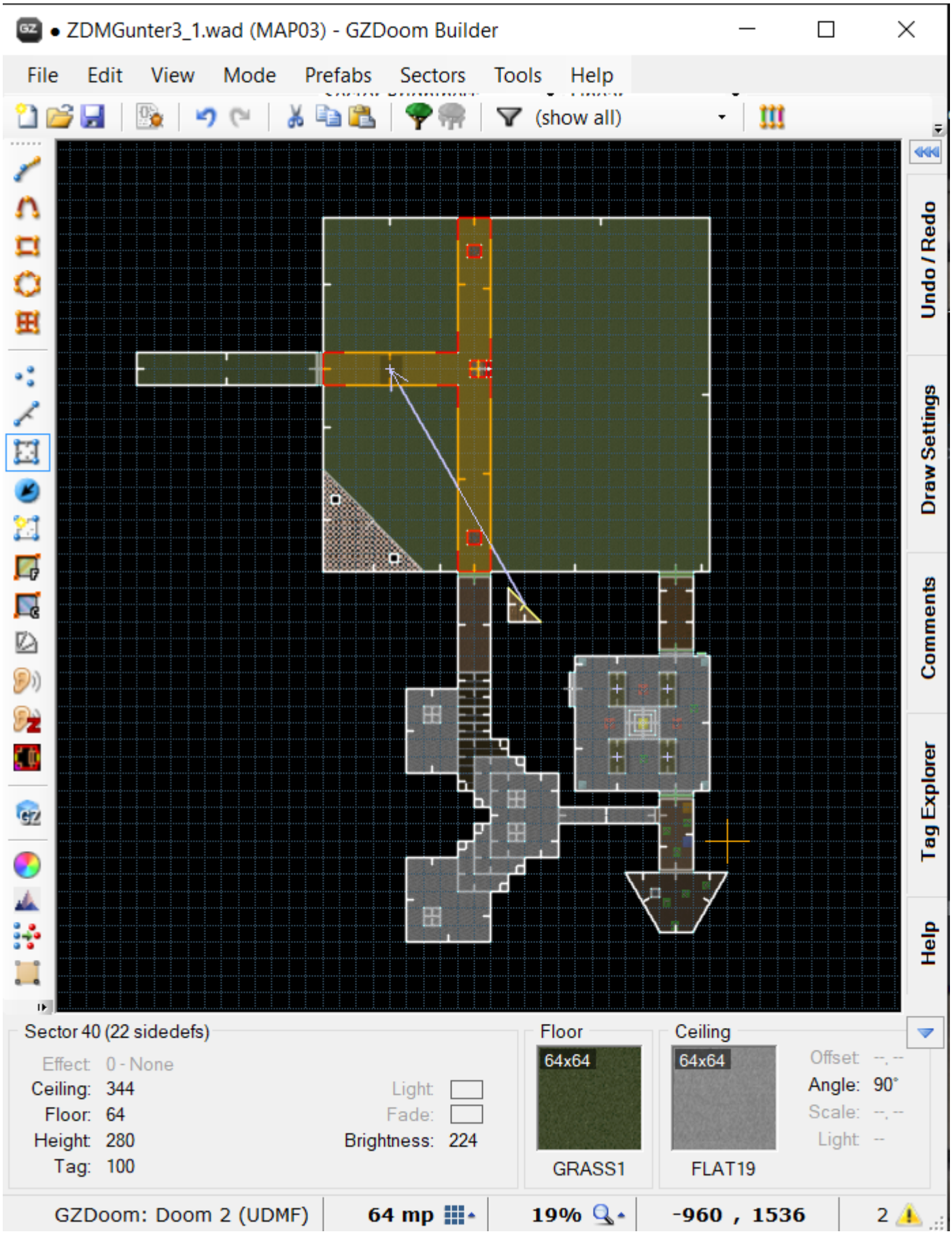
Slade en GZDoom Builder

Deze twee 3D game editors geven een gemakkelijke werkomgeving waarmee je snel een game in elkaar kunt zetten, zonder gebruik te hoeven maken van code. Slade en GZDoom Builder hebben een ingebouwde library die uit DOOM onderdelen bestaat. Oude editors, zoals DeePSea, dat de vorige versie was vóór Slade, ondersteunen alleen maar de standaard DOOM versie, waardoor alleen maar onderdelen van DOOM 1 en DOOM 2, Heretic, Hexen en Strife gebruikt kunnen worden. Maar met de nieuwe versies, Slade en GZDoom Builder, is het enorm uitgebreid en kan men nu ook eigen onderdelen ontwerpen en kan zelfs een eigen besturing worden geprogrammeerd met ACS scripts.

De twee design types: Vanilla en UDMF

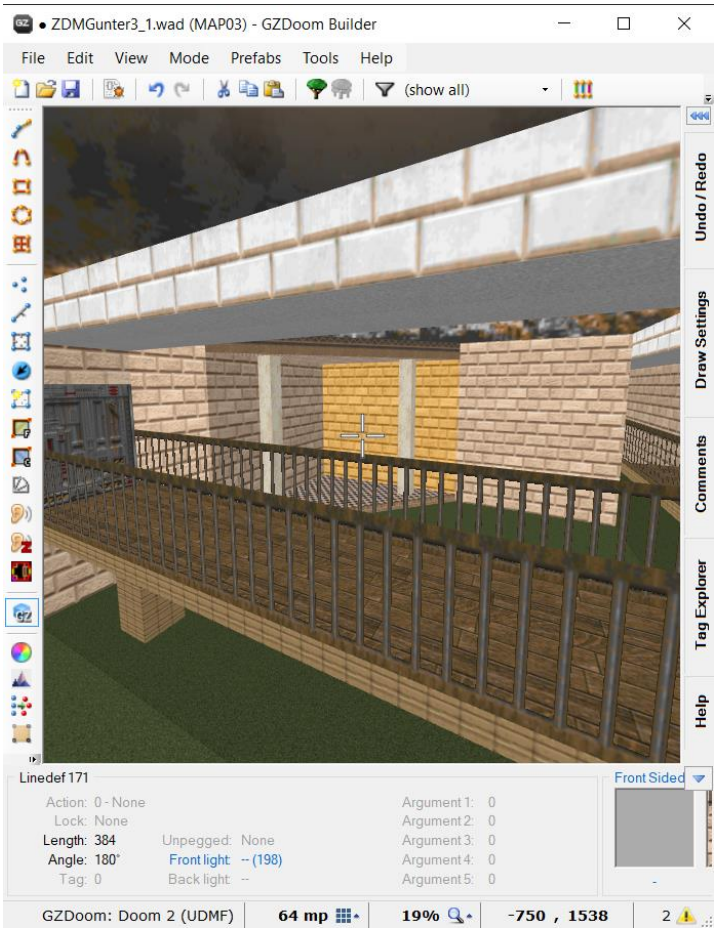
Het DOOM type Vanilla is het standaard type met beperkte acties en geen 3D omgeving, maar een 2½D omgeving, ook al ziet het er wel als een 3D omgeving uit. Omdat er in 2½D geen diepte is (dus geen Z-as), kunnen we alleen maar de breedte en hoogte gebruiken. Bovendien is er maar één hoogte in te stellen vanaf de vloer naar het plafond. Hoe dik de vloer of het plafond is, kan gewoon niet bepaald worden. Alle muren, vloeren en plafonds worden opgebouwd met flats, die geen dikte hebben. Ook schuine vloeren en schuine plafonds ontbreken in Vanilla, waardoor mooie ronde poorten en tunnels maken onmogelijk is.

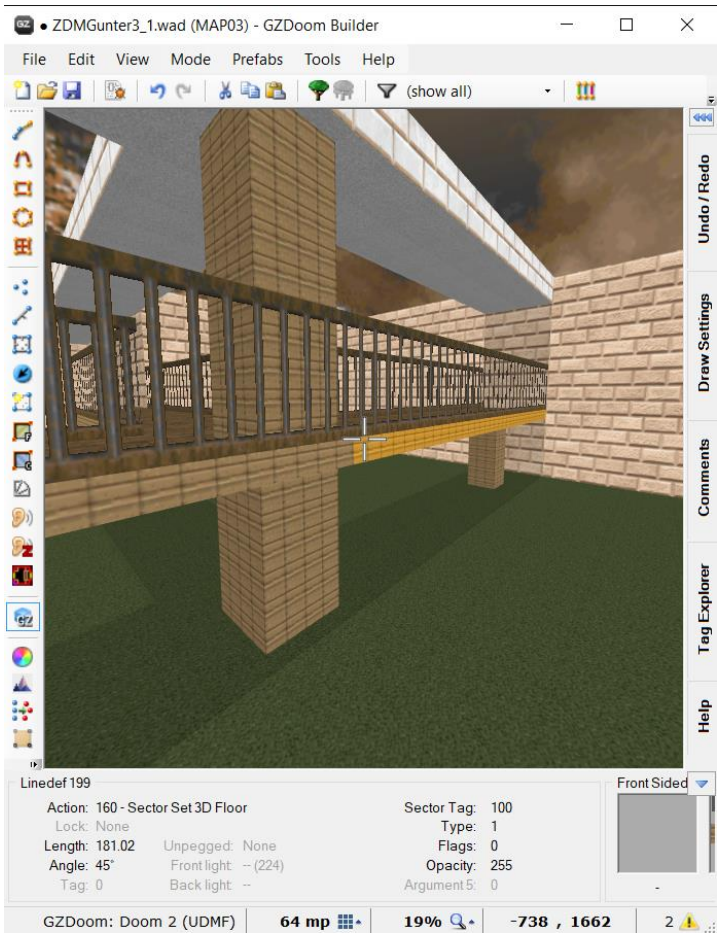
Het type UDMF (Universal Doom Map Format) is eigenlijk geen DOOM type, want hiermee kunnen we wel de mogelijkheden gebruiken die elke 3D editor kent. Toch moeten we in Slade en GZ er rekening mee houden dat we niet direct de mogelijkheid hebben om bijvoorbeeld een 3D vloer te kunnen maken. Onderstaande afbeeldingen laten een game level zien in 2D en binnen de game level in 3D.



Aan deze 2D game level is te zien waar we rekening mee moeten houden. Het heeft te maken met een getekend driehoekje waar een pijl gericht is op een gegeven verlichte sector. Deze sector

moet een 3D vloer als een soort brug worden. Dat kan alleen door de hoogte van de onderkant van de brug en de hoogte van de bovenkant van de brug in de control sector (de driehoek) op te geven. Door middel van een tag code en de actie 3D sector in te stellen, zal de sector los raken van de grasvloer, zie de 3D afbeeldingen hieronder.



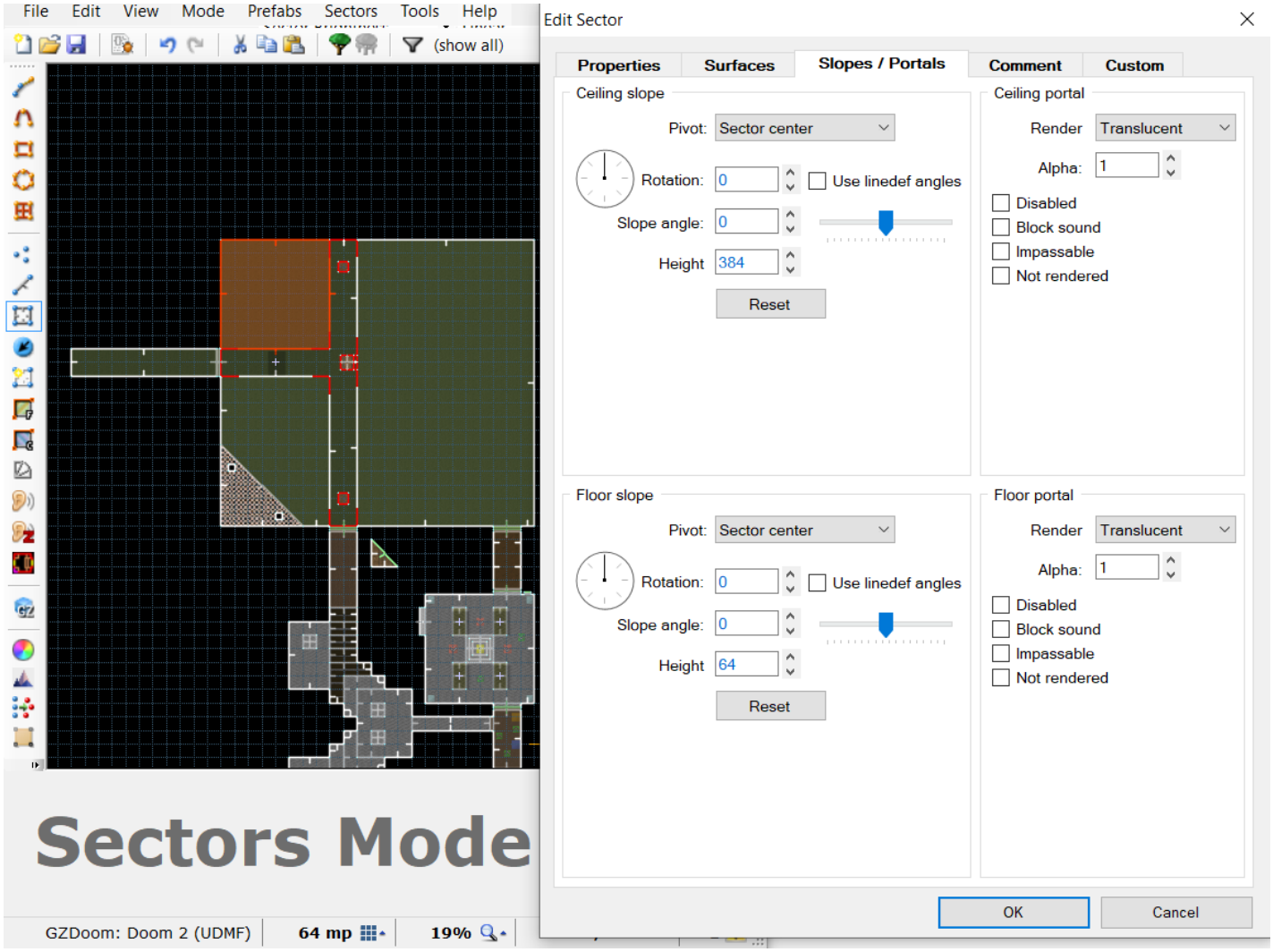


Slopes en portals

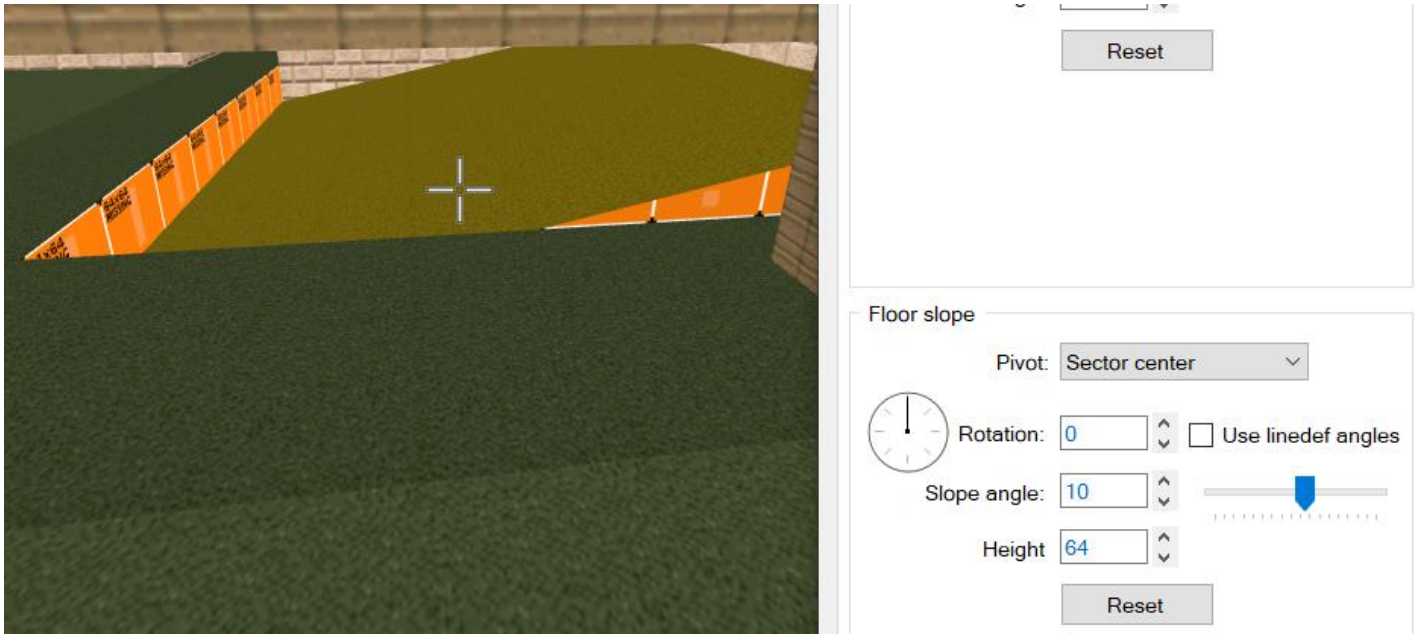
Toch hoeft niet alles via control sectors te worden gedaan. GZDoom heeft voor elke sector die geselecteerd wordt een apart tabblad met mogelijkheden om de vloeren of plafonds van een sector verticaal te kunnen roteren in een bepaalde gradenhoek.

Triangle sectors

Om een juiste sector daarvoor te kiezen, moet de slope goed kunnen werken. Een compleet geselecteerde vierkante sector, zoals op onderstaande afbeelding te zien is, werkt niet altijd naar behoren. Dat heeft te maken met het aantal hoeken dat de sector heeft. Door sectors op te bouwen als driehoeken binnen een grote sector, kan elke hoek van een triangle van een andere X,Y en Z worden voorzien of automatisch worden ingesteld op het Slope tabblad. Ter vergelijking: 3D modellen worden net zo opgebouwd.



Hier kun je zien dat een niet-triangle sector het hele vlak meeneemt met een slope angle van 10 graden. De twee zijden zouden daardoor niet netjes schuin afgevlakt kunnen worden.



Door het vlak in meer driehoekige sectoren te verdelen, zouden beide zijden kunnen kantelen en aan de zijden van de andere sectoren aan kunnen sluiten.

Textures

Hoewel het ontwerpen van 3D levels het meeste werk is, lijkt het echter half werk te zijn. Er is nog een tweede moeilijkheid met level design: het gebruik van textures. Kennis hebben in het gebruiken van de juiste textures is een vak. Niet alle kleuren passen bij elkaar en het is veel experimenteerwerk om een mooie 3D omgeving met de beste textures te ontwikkelen. Zoals aan de brug afbeelding te zien is, past de witte muur boven de brug goed bij de bruine brug, maar niet als er een rode muur boven de brug gebruikt zal worden. Ook moeten textures rust geven aan de omgeving. Textures die teveel tinten kleuren hebben kunnen teveel drukte geven. Een texture die als vloerkleed wordt gebruikt, ga je niet leggen op een buitensector, zoals de grasvloer op de afbeeldingen laat zien.

ACS scripts

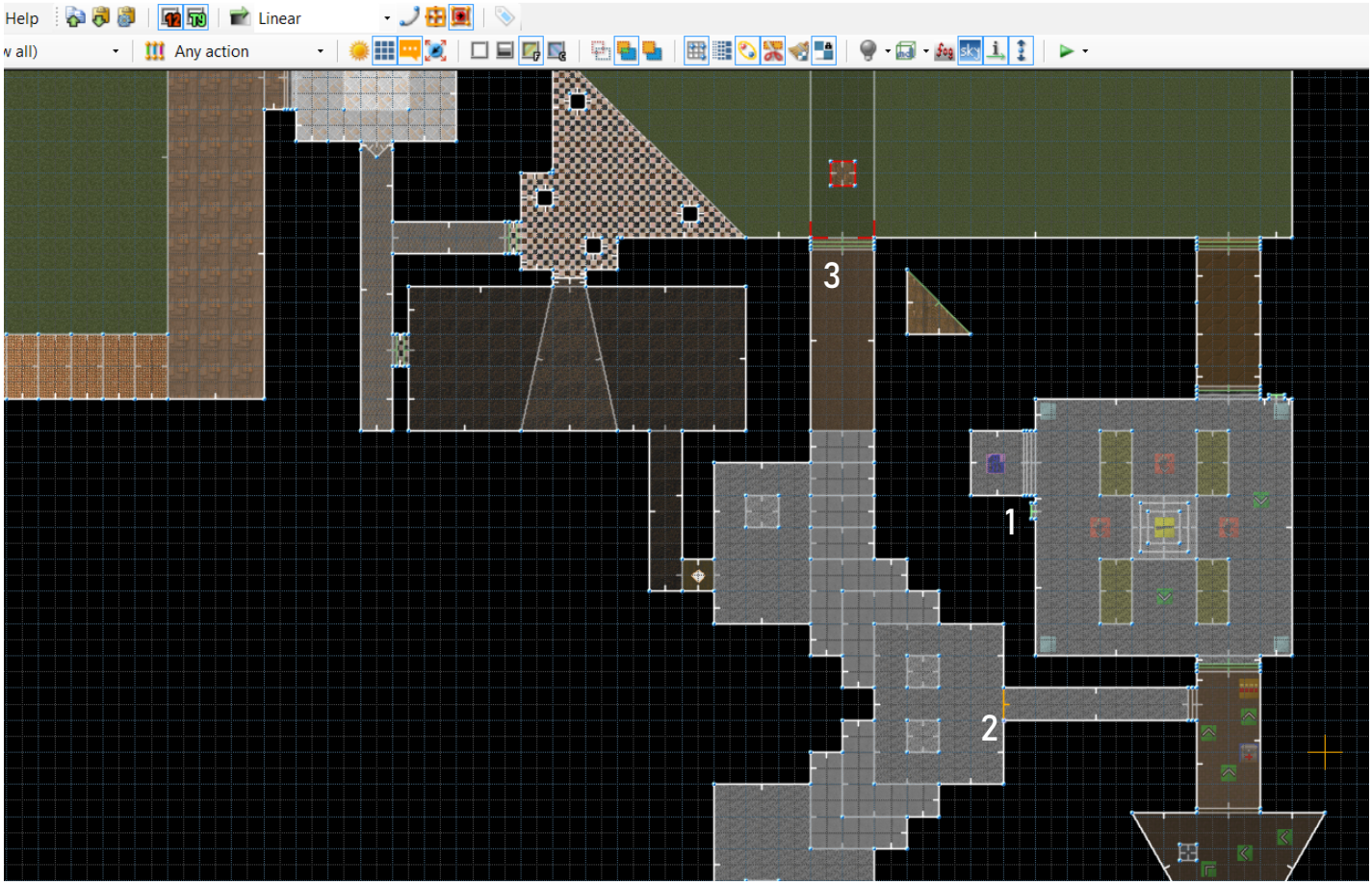
Maken we twee sectoren met een deursector ertussen, dan kunnen we de deur gemakkelijk met alleen de eigenschappen laten werken. Daar is geen code voor nodig en als we het level testen, dan zal alles prima werken. Mooi, een 3D game in elkaar gezet.

Maar we willen allemaal meer. Niet alleen maar een deur die open en dicht gaat, een lift die omhoog en omlaag gaat of een actie door middel van een switch (hendel) uit willen voeren. Deze acties, die in de eigenschappen uit de lijst gekozen kunnen worden, zitten er standaard in. Dat betekent niet dat ook Vanilla de hele lijst met acties heeft die UDMF heeft, en als we toch meer willen dan zouden we toch over moeten stappen van Vanilla naar UDMF. Met ACS scripts is het mogelijk om meer acties in één keer uit te kunnen voeren. Hieronder is een voorbeeld met scripts voor verschillende acties die in Vanilla niet uitgevoerd kunnen worden.

```
#include "zcommon.acs"

bool doorCanOpenSwitch = FALSE;
int textSeconds = 4;

script 1 (int line1)
{
     SetLineBlocking(line1, OFF);
     Line_SetTextureOffset(line1, 0, 0, SIDE_FRONT, TEXFLAG_MIDDLE);
     SetLineTexture(line1, SIDE_FRONT, TEXTURE_MIDDLE, "GRNOPEN");
}

script 2 (int doorTag)
{
     if(doorCanOpenSwitch)
     {
          Door_Open(doorTag, 16);
     }
     else
     {
          Print(s:"That doesn't working now!");
          Delay(35 * textSeconds);
     }
}
```

```
script 3 (int doorTag)
{
	Door_Open(doorTag, 16);
	doorCanOpenSwitch = TRUE;
	print(s:"The Gunter door can open now!");
	Delay(35 * textSeconds);
}
script 4 (void)
{
	print(s:"Open the door from another side!");
	Delay(35 * textSeconds);
}
```

Tijdens het schrijven van dit Bulletin was ik ook weer een stukje verder met het level in GZDoom Builder. Deze afbeelding laat een deel van het level zien waar script nummer 2 en script nummer 3 mee te maken hebben, namelijk het bepalen wanneer de switch de Gunter deur mag openen en welk script daarvoor verantwoordelijk is.



De bool variabele doorCanOpenSwitch heeft eerst de waarde FALSE, zie de listing. Script 2 bepaalt of de switch bij punt 1, zie afbeelding, de deur mag openen waar de blauwe sleutel te vinden is.

In Script 3 wordt de variabele op TRUE gezet, zodra de player over de gele line loopt, zie punt 2, en ook wordt er een extra deur geopend in dezelfde hal, zodat de player uit de gang kan lopen en te-rug kan naar de Gunter deur om nog eens op de switch te drukken. Onderstaande 3D afbeeldingen zijn snapshots tijdens de game.

Deze afbeelding is punt 1.
Wanneer er op de switch wordt gedrukt, staat de variabele doorCanOpenSwitch op FALSE. De code wordt in het else blok uitgevoerd en de tekst verschijnt.

Deze afbeelding is punt 2.
Er wordt hier over een "voor de player onzichtbare" line gelopen. Script 3 wordt uitgevoerd. Eerst wordt de muur vooraan als een deur omhoog geschoven, zodat de player door kan lopen. Dan wordt de variabele doorCanOpenSwitch op TRUE gezet en verschijnt de tekst door de Print() functie dat de Gunter deur nu geopend kan worden. De switch zal dus nu wel werken.

Eigenlijk kunnen we zeggen dat twee scripts samen werken. Zonder de switch kan de deur niet open, maar we kunnen met de switch dat nog niet doen als we nog niet in deze gang hebben gelopen.

Scripts geven daarom veel functionaliteit. Zo zal script 1 er voor zorgen dat er een gat in een muur geschoten of getrapt kan worden. Het is ook mogelijk particles te laten verschijnen als er een object ontploft.

Script 4 zal ook uitgevoerd worden bij een deur, zie punt 3 bij de afbeelding. De deur kan namelijk alleen vanaf de brug geopend worden. Gek genoeg wordt de player voor de gek gehouden, want het script voert geen actie uit. Alleen maar tekst die door de Print() functie verschijnt. De deur heeft geen tag code om open te kunnen, vandaar dat script 4 een void als argument heeft.

De software

De twee editors Slade en GZDoom Builder zijn gratis programma's, zie:
http://slade.mancubus.net/ voor Slade 3.
https://forum.zdoom.org/viewtopic.php?t=32392 voor GZDoom 2.3. Scroll wat naar beneden tot je Download tegenkomt. Daar staat GZDoom.

Onthoud!

Om de levels te kunnen ontwerpen en te kunnen testen, heb je een IWAD bestand nodig. Dit kan DOOM 1 of DOOM 2 zijn, de BFG versies van DOOM 3 of Final DOOM TNT of Final DOOM Plutonia. De editors zelf hebben intern geen IWAD en zonder een IWAD zijn er geen onderdelen om mee te werken.
Zie hier de download voor DOOM 2: https://gamesnostalgia.com/download/doom-ii_/1559

Wil je liever een goede versie die betrouwbaar is, kijk dan op:
https://www.gog.com/game/doom_ii_final_doom?pp=35e995c107a71caeb833bb3b79f9f54781b33fa1

IdSoftware heeft alleen nog de hele nieuwe Doom versies. Kijk op:
https://www.idsoftware.com/en-gb/
Deze nieuwe versies werken niet met de editors.

Toch is het mogelijk om zonder IWAD een level te maken. Slade accepteert om met een nieuw WAD bestand te beginnen met een lege texturelijst. Wanneer alles klaar is met nieuwe textures en map levels, kan je zelf een eigen IWAD maken door deze als IWAD bestandstype op te slaan. Bedenk wel dat je echt met een schone lei begint, dus ook zonder decoraties en monsters. Er zijn websites, zoals www.realm667.com, waar je van alles kunt vinden voor je project. Op de Slade website, zie de link hierboven, en op de Zdoom Wiki staan veel tutorials over hoe je de map levels maakt en hoe je met ACS programmeert.

Wanneer de IWAD in Slade is aangemaakt en de onderdelen erin staan, kan deze ook in GZDoom Builder worden gebruikt.

Het initialiseren van C# objecten

Wie al weet wat een klasse is, weet ook hoe handig het is deze te gebruiken. Klassen hebben nut wanneer we de gegevens, die we daar in opslaan, in meer objecten nodig hebben. Het maken van een goede klassenstructuur beperkt de vele code in het hoofdprogramma.

Het antwoord, waarom ik het speciaal over C# heb, is omdat ik wat ontdekt heb. Iets nieuws voor beginners en gewoon iets leuks voor hobbyisten die C# al aardig kennen. Het heeft te maken met het initialiseren (vullen) tijdens het declareren van objecten.

Hieronder is een normale klasse te zien voor NAW gegevens. Even zonder methoden als voorbeeld, over wat ik gevonden heb.

```
public class NAW
{
     public string naam;
     public string adres;
     public string woonPlaats;
}
```

In een andere klasse, zoals het hoofdprogramma, zou je de object variabelen als volgt in kunnen stellen:

```
private NAW _naw = new NAW();

private void Instellen_NAW()
{
     _naw.naam = "Naam1";
     _naw.adres = "Adres1";
     _naw.woonPlaats = "Woonplaats1";
}
```

Voor meer NAW gegevens zou je een array of een List<> object kunnen gebruiken en op deze manier in een lus in kunnen lezen vanaf een bestand of als het maar om bijvoorbeeld 3 NAW gegevensobjecten gaat dan zou je 3 Instel methoden moeten maken en aan moeten roepen, maar het kan ook anders en dat is de manier die ik pas gevonden heb in C#. In Basic en Pascal werkt deze techniek niet. Onderstaand voorbeeld doet hetzelfde als het vorige voorbeeld, maar nu zonder de void methode Instellen_NAW().

```
private NAW _naw = new NAW
{
     naam = "Naam1",
     adres = "Adres1",
     woonPlaats = "Woonplaats1"
};
```

Denk eraan dat elke regel tussen accolades gescheiden moet worden met een komma en niet met een puntkomma. De laatste regel 'woonPlaats' heeft geen komma nodig. In deze definitie moet de sluitaccolade een puntkomma hebben.

Geavanceerdere definities

Wanneer in een klasse een object gedeclareerd wordt die ook te vullen is, zoals card hieronder, dan wordt de structuur wat gecompliceerder, en ook nog eens als je alles in een lijst wilt hebben:

```
private List<baseClass> _base;

_base = new List<baseClass>
{
     person1 = new NAW
     {
          name = "Name1",
          address = "Address1",
          place = "Place1",
          card = new Card()
          {
               width = 85,    // portret card
               height = 125
          }
     },
     person2 = new NAW
     {
          name = "Name2",
          address = "Address2",
          place = "Place2",
          card = new Card()
          {
               width = 125,   // landscape card
               height = 85
          }
     }
};
```

Nu staat de puntkomma niet achter de sluitaccolade van de NAW klasse.

Natuurlijk is bovenstaande techniek niet altijd zinvol. Als er heel veel personen zijn die verschillende soorten kaarten moeten krijgen, dan wordt de definitie wel heel lang. Het gebruik van een gegevensbestand is dan het beste. Door functiemethoden in de NAW klasse gebruik te laten maken van een List<> en elk record die gelezen wordt toe te voegen, zal het veel beter werken.

Toch kan bovenstaande definitie voor kleine instellingen best worden gebruikt.

Zie voor meer informatie https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/how-to-initialize-objects-by-using-an-object-initializer en https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/object-and-collection-initializers.