Game Maker Tutorial

Platform Spellen Maken

Geschreven door Mark Overmars

Vertaling: Lars
Nederlandse Game Maker Community (www.game-maker.nl)
Laatst veranderd: 26 februari 2007
Gebruikt: Game Maker 7.0, Lite of Pro versie, Advanced Mode
Niveau: Gemiddeld

Platformspellen zijn erg populair, vooral op apparaten als de Game Boy. In een platformspel bekijk je de scène van de zijkant. De speler bestuurt normaal gesproken een karakter die rondloopt in de wereld. Deze wereld bestaat uit platformen. De speler kan op deze platformen lopen, springen of vallen van het ene naar het andere platform, ladders of touwen gebruiken om op verschillende plaatsen te komen, enz. Op de platformen zijn objecten om te verzamelen, vijanden om te vermijden of te doden (vaak door op hen te schieten of bovenop hen te springen), schakelaars om in te drukken voor het openen van doorgangen, enz. Ook heeft de speler normaalgesproken vaardigheid nodig om over gevaarlijke gebieden te springen. In sommige platformspellen zie je het hele level in één keer, maar meestal zie je alleen een deel rondom het karakter. In een dergelijk geval wordt het vinden van de weg een toegevoegde uitdaging.

Het maken van een goed platformspel is niet oppervlakkig, ook niet met Game Maker. Er zijn drie belangrijke aspecten:

- Het maken van een natuurlijke beweging voor het karakter.
- Genoeg variaties maken in monsters, achtergronden, enz.
- Het zorgvuldig ontwerpen van de levels zodat ze leuk zijn om te spelen en langzamerhand steeds moeilijker worden.

In deze tutorial leer je hoe je een eenvoudig platform spel kunt maken in *Game Maker*. We zullen het spel in enkele stappen bouwen. De verschillende stappen zijn beschikbaar als aanpasbare spellen in de map `Examples`. Ze bestaan maar uit één level om enkele bijzondere aspecten te laten zien. Je kunt ze gebruiken als basis voor je eigen platform spellen.

De Basis

We beginnen met het simpelste platformspel. Je kunt deze vinden in het bestand `platform_1.gmk`. In elk platformspel zijn er twee basisobjecten: het karakter die door de speler bestuurd wordt, en een blok die gebruikt wordt voor de vloeren (platformen) waar de speler op kan lopen. Hetzelfde blok wordt vaak gebruikt voor de muren die de speler niet kan passeren. We hebben twee sprites nodig: één voor het karakter en één voor het blok. Voor het karakter gebruiken we een eenvoudige bal. Voor het blok gebruiken we een (niet-transparant) zwart vierkant. We maken twee objecten. Het blok object is een solid object zonder events of actions. Het staat er alleen. Het karakter object is ingewikkelder.

Beweging

Het belangrijkste aspect dat we behandelen in dit eerste gedeelte is het definiëren van de beweging van het karakter. Het probleem is dat het karakter moet lopen op de bovenkant van de vloeren. Het mag de vloer niet kruisen. Als het karakter springt of valt van een platform moet het goed landen op het volgende platform. Er zijn een aantal verschillende manieren waarop het karakter kan lopen, springen en vallen. Verschillende platformspellen gebruiken verschillende manieren. Normaalgesproken gebruiken we slechts drie toetsen voor het besturen van de beweging. Het linkerpijltje zou het karakter naar links moeten bewegen, het rechterpijltje zou hem naar rechts moeten bewegen, en het pijltje omhoog of de spatiebalk laat hem springen.

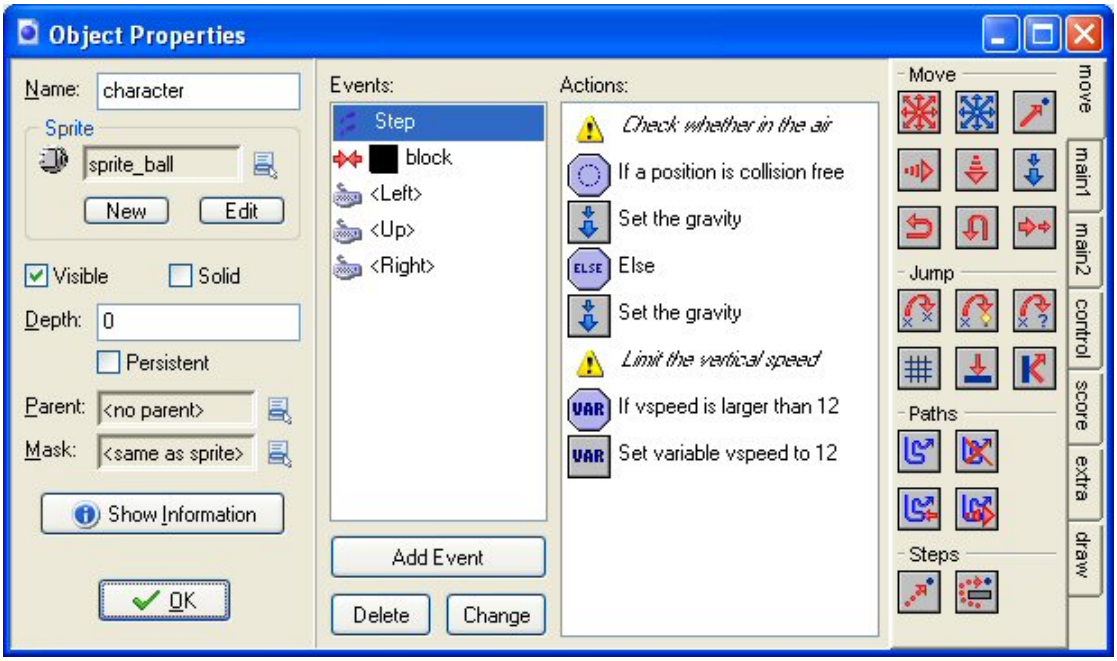
Laten we eerst kijken naar het naar links en naar rechts bewegen. De eerste keuze die je moet maken is of de speler alleen zijn richting kan veranderen wanneer hij op een platform is, of ook in de lucht wanneer hij springt of valt. Ook al is de tweede mogelijkheid onnatuurlijk (het is moeilijker om naar links te gaan terwijl je aan het vallen bent) kiezen we toch voor de tweede mogelijkheid, wat betekent dat het karakter overal naar links en rechts kan bewegen. Dit leidt tot een betere game play en is eigenlijk ook makkelijker om uit te voeren.

De tweede keuze is of de beweging een constante snelheid heeft of dat het versneld wordt wanneer je de toets ingedrukt houdt. Omdat het eenvoudiger is kiezen we voor het eerste. Het toestaan van versnellen geeft normaalgesproken een betere game play: de speler moet bijvoorbeeld van een afstand beginnen met rennen om over een groot gat te springen.

Zoals je weet zijn er meerdere manieren om het karakter te laten bewegen. We kunnen de snelheid van een beweging instellen of we kunnen simpelweg het karakter in een richting bewegen. In platformspellen is het normaalgesproken het makkelijkst om de verticale beweging automatisch te laten gaan (zoals we hieronder zullen zien) maar de horizontale beweging zelf te doen. Dit is eenvoudiger. In het keyboard event van het linkerpijltje controleren we of de relatieve positie (-4,0) vrij is. Als dat zo is laten we het karakter springen naar die positie. We doen hetzelfde met het keyboard event van het rechterpijltje. Zie het bijgevoegde voorbeeld.

Springen

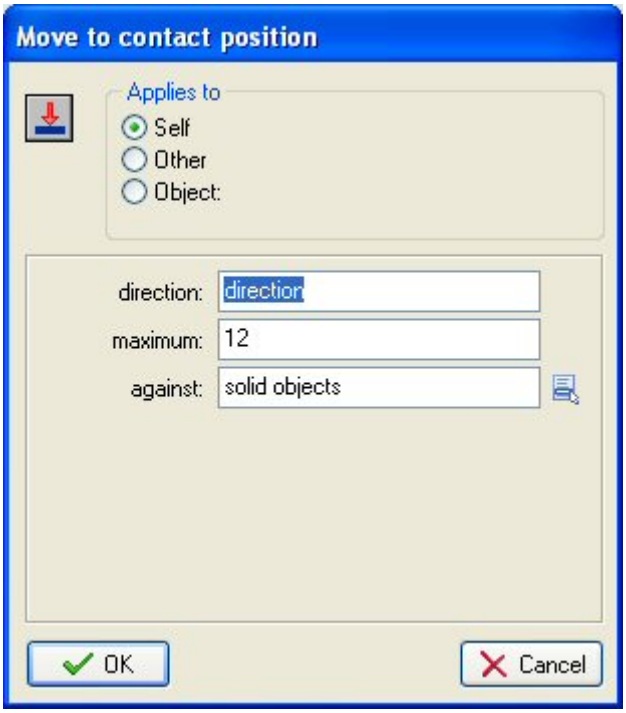
Vervolgens hebben we een verticale beweging nodig. Dit is moeilijker. Om het karakter naar beneden te laten vallen kunnen we zwaartekracht gebruiken. Maar hij zou moeten stoppen met de beweging wanneer we de vloer raken. Daarnaast wil je normaalgesproken een maximale valsnelheid, anders zal het karakter te snel bewegen. (Dit is beide niet erg aangenaam maar het kan ook problemen geven tijdens het spelen. Het karakter zou bijvoorbeeld door de grond kunnen vallen wanneer hij te snel beweegt.) Om dit probleem op te lossen, kijken we in het step event van het karakter of de positie direct onder het karakter botsingvrij is. Als dat zo is, is het karakter in de lucht en zetten we de zwaartekracht op een positieve waarde. Anders zetten we het op 0. We kijken ook naar de variabele `vspeed` welke de verticale snelheid aanduidt. Als deze groter dan 12 is zetten we hem terug op 12. Op deze manier beperken we de verticale snelheid tot 12. Dus het event ziet er ongeveer zo uit:



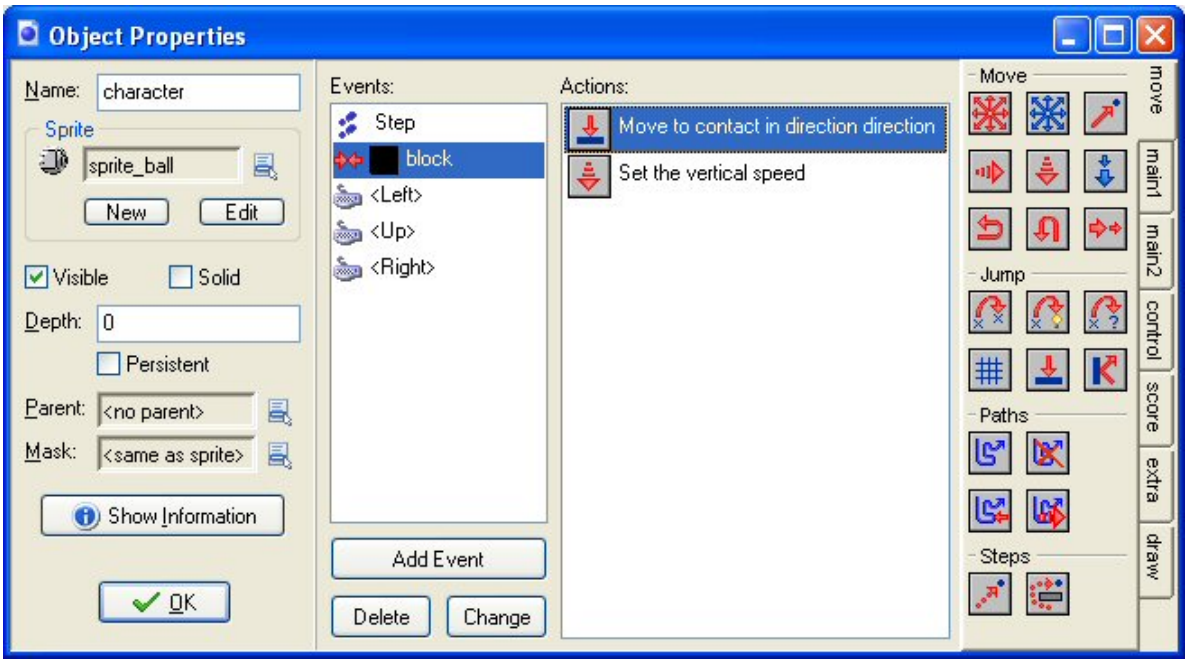
Vervolgens moeten we goed op de vloer landen. Dit is moeilijker dan het lijkt. Het gebeurt wanneer het karakter botst met het blok object. In dit collision event zouden we de verticale beweging op 0 moeten zetten. Maar dit zou het karakter een beetje in de lucht boven de grond kunnen laten hangen. (De reden is dat het karakter terug is geplaatst naar zijn vorige positie voor de botsing.) Om dit te stoppen willen we het karakter naar het exacte punt waar de botsing plaatsvindt laten bewegen. Gelukkig is hier een actie voor in *Game Maker*:

> **Move to contact position**
> Met deze actie kun je de instantie bewegen in een aangegeven richting totdat er een contact positie met een ander object is bereikt. Als er al een botsing is op de huidige positie wordt de instantie niet bewogen. Anders wordt de instantie geplaatst vlak voordat er een botsing plaatsvindt. Je kan de richting aangeven maar ook een maximale afstand om te bewegen. Ook kan je aangeven of er alleen op solid objecten moet worden gelet of op alle objecten.

We gebruiken deze actie. Als richting vullen we de variabele direction in wat de huidige beweegrichting van de instantie is. Als maximale afstand vullen we 12 in (hoewel dit niet echt belangrijk is hier):



Dus het complete collision event met het blok ziet er zo uit:



Je zou kunnen zeggen dat we dit alleen moeten doen wanneer we de vloer onder ons raken. Maar eigenlijk willen we ook naar de contact positie bewegen wanneer we een vloer van de onderkant raken of een muur van de zijkant raken. Er is hier één belangrijk ding wat vaak de oorzaak van problemen is: We gaan ervan uit dat het karakter op zijn vorige positie inderdaad botsingvrij is. Je zou dit verwachten maar dit is niet altijd het geval. Een veel voorkomende misvatting is dat wanneer het karakter een geanimeerde afbeelding heeft, het botsing gebied ook elke step verandert. Dit betekent dat het nieuwe plaatje op de vorige plaats nog steeds een botsing

veroorzaakt. Dus je kunt er beter voor zorgen dat het karakter één botsing gebied heeft (zie ook het volgende deel).

Tenslotte moeten we het karakter laten springen wanneer het pijltje naar boven is ingedrukt. Maar dit moet alleen gebeuren als het karakter op dat moment op de grond is. Dus we controleren eerst of de positie onder het karakter een botsing maakt en, als dat zo is, zetten we de verticale snelheid bijvoorbeeld op -10. Je moet een beetje spelen met de waarde -10 voor de verticale snelheid en de waarde 0.5 voor de zwaartekracht om de beweging te krijgen die je wilt.

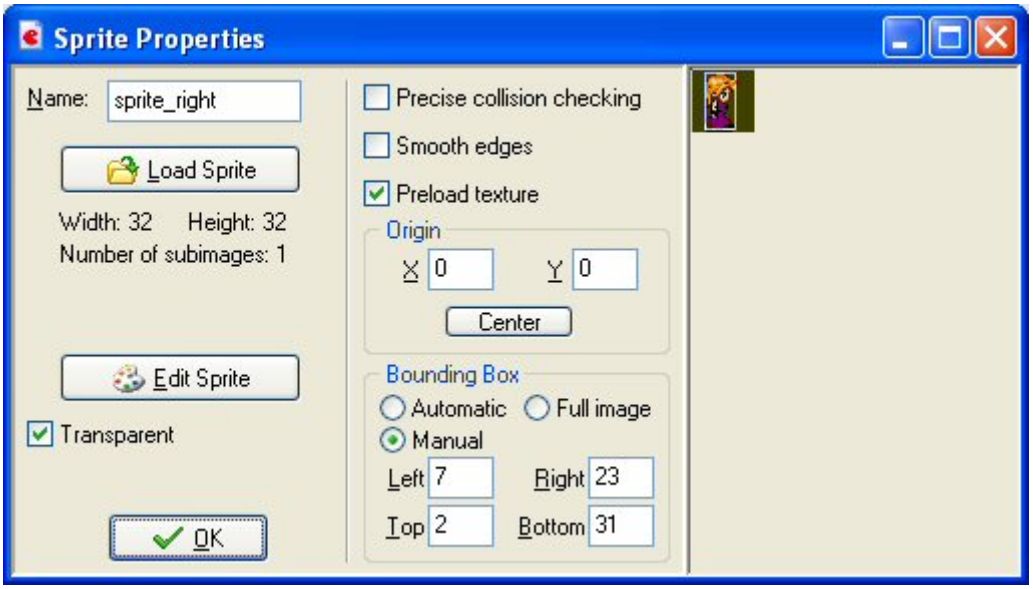
Nu is de basis voor het platformspel klaar. Ontwerp een level met een paar vloeren en muren, gebouwd van instanties van het blok object. Plaats een instantie van het karakter in de room en je bent klaar.

Betere graphics

Het basis platformspel dat we in het vorige gedeelte hebben gemaakt doet het maar het ziet er slecht uit. Er zijn twee dingen die we willen veranderen: hoe de speler eruit ziet, en hoe de achtergrond eruit ziet. Het aangepaste spel kun je vinden in het bestand `platform_2.gmk`.

Afbeeldingen van het karakter

Laten we beginnen met de graphics van het karakter. We zullen twee verschillende (niet-animerende) sprites gebruiken: één voor het karakter dat naar links kijkt en één voor het karakter dat naar rechts kijkt. Nu is het het makkelijkst om in het event voor het linker pijltje een actie te plaatsen die de sprite veranderd in degene die naar links kijkt. Op dezelfde manier, in het event voor het rechterpijltje verander je hem in het karakter dat naar rechts kijkt. Het is erg belangrijk dat je precise collision checking voor de twee sprites uitzet. Hier zijn een aantal redenen voor. Ten eerste, het voorkomt dat de sprite halverwege de rand van het platform vast komt te zitten. Ten tweede, wanneer de sprite van linkskijkend naar rechtskijkend wordt veranderd moeten ze hetzelfde botsing gebied gebruiken, anders kan het karakter vast komen te zitten. Dit is nog belangrijker wanneer je geanimeerde sprites gebruikt. Om dezelfde reden kan je er beter voor zorgen dat de bounding boxes van de sprite hetzelfde zijn. Hiervoor kan je altijd handmatige bounding boxes gebruiken. Dus wanneer je de sprites toevoegt zouden de instellingen ongeveer zo moeten zijn.

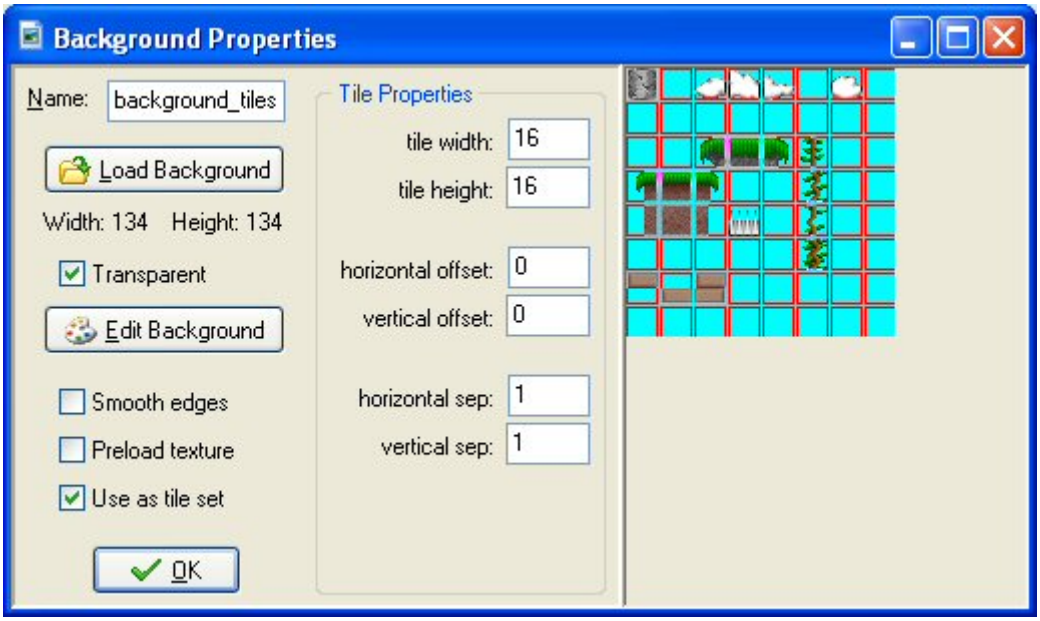


In meer gevorderde spellen zal je waarschijnlijk geanimeerde sprites willen gebruiken. In dit geval heb je ook een sprite nodig voor het karakter wanneer die niet beweegt. Ook wil je misschien sprites toevoegen voor het karakter wanneer hij springt, valt, schiet, enz. In dit geval moet je de sprite op verschillende plaatsen in de events veranderen. In het bijzonder, in de no key event wil je waarschijnlijk de sprite veranderen naar de niet bewegende. Als alternatief, kan je de goede sprite in het draw event tekenen gebaseerd op de situatie. Je kunt bijvoorbeeld kijken of `xprevious<x` om uit te vinden of het karakter naar rechts is bewogen. Zoals ik al eerder aangaf, zorg ervoor dat alle sprites dezelfde bounding box hebben en precise collision checking uit staat.

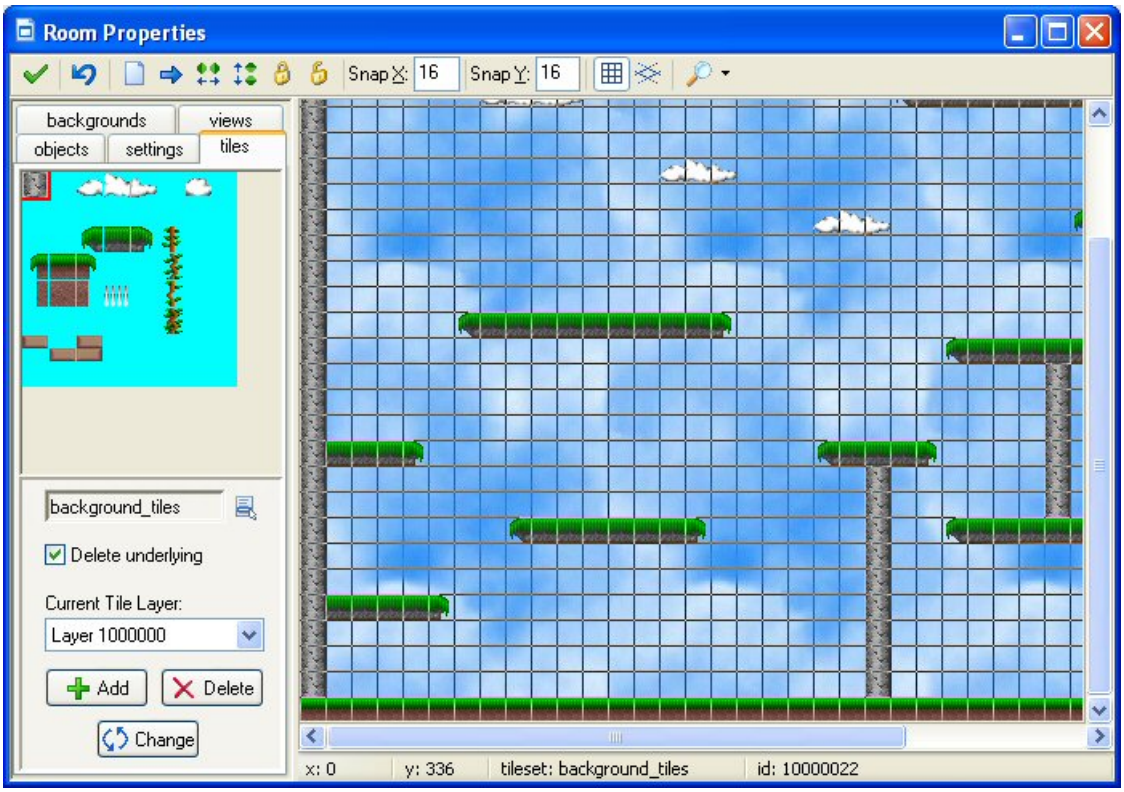
De platformen en muren

Als tweede willen we de achtergrond en de platformen verbeteren. Hiervoor gebruiken we een standaard techniek. In plaats van het gebruiken van objecten voor alle verschillende onderdelen van muren en vloeren, gebruiken we zogenaamde tiles. Tiles zijn stukjes van achtergrondafbeeldingen die op bepaalden plaatsen in de room zijn getekend. Ze hebben geen bijbehorende events en ze maken geen botsingen. Het voordeel is dat ze snel zijn en weinig geheugen gebruiken. Dus je kunt grote rooms maken zonder dat je grote plaatjes nodig hebt.

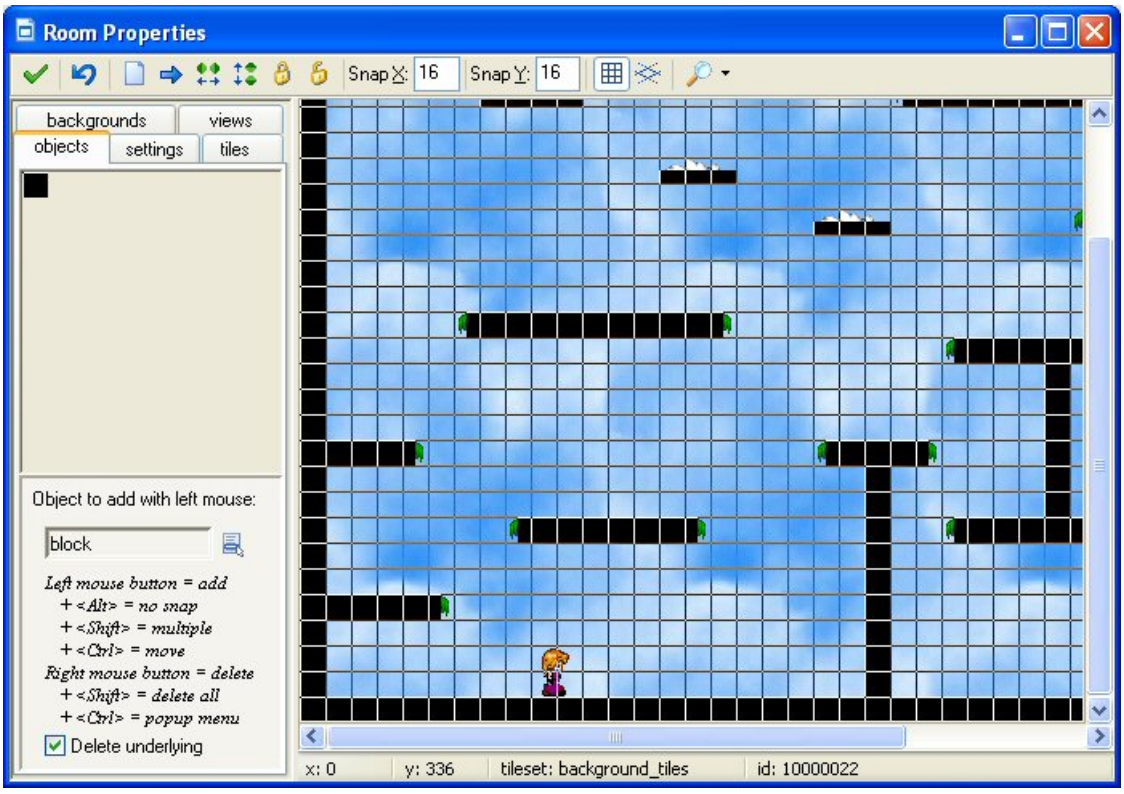
Om tiles toe te voegen aan je rooms heb je eerst een achtergrondafbeelding nodig die de tiles bevat. Tiles in een achtergrondafbeelding hebben bij voorkeur een vaste grootte en hebben een kleine (1-pixel) grens tussen elkaar zodat ze gemakkelijk gescheiden kunnen worden. Voor ons simpele platformspel hebben we onze eigen gemaakt welke is bijgevoegd in de map `Resources`. We voegden het als een transparante achtergrond resource toe genaamd `back_tiles`. Wanneer je het in het spel toevoegt, geef je in de achtergrond eigenschappen aan dat het moet worden gebruikt als een tile set en vul de correcte tile grootte en scheiding in, zoals hier volgt:



Nu kan je, tijdens het maken van de room, klikken op het tabblad **Tiles**. Je kunt de tile set selecteren (dat is de toegewezen achtergrond resource). Nu kan je tiles tekenen door op de toepasselijke tile te klikken en ze vervolgens in de room te plaatsen, zoals je ook voor objecten doet. De rechter muisknop verwijdert de tiles. Gebruik je fantasie om uitdagende rooms te maken. (Merk op dat je tiles op verschillende lagen van diepte kan plaatsen door lagen (layers) toe te voegen. Je kunt bijvoorbeeld een laag maken van tiles die voor de bewegende karakters ligt. We zullen ze hier niet gebruiken maar ze zijn geweldig om een beter 3D effect te geven.)



Er is nog een probleem over. Zoals hierboven is aangegeven zijn tiles alleen mooie graphics. Er zijn geen events of botsingen aan verbonden. Dus het karakter zal er recht door heen vallen. Om dit te vermijden hebben we nog de blok objecten nodig die we hiervoor hadden. We plaatsen de blok objecten op de toegewezen plaatsen bovenop de muren en platformen die je met de tiles op de achtergrond hebt gemaakt. Door nu de blok objecten onzichtbaar te maken zul je niet de zwarte blokken zien maar de mooie tiles. Maar de blok objecten zijn er eigenlijk wel, dus het karakter kan niet door de muren heen en zal op de platformen landen.



Er zal misschien een probleem zijn. De blok objecten van 16x16 zullen te groot zijn om de achtergrond mooi te bedekken. Dus we willen een paar andere blok objecten maken met een grootte van 16x8 en 8x16. Deze maken we opnieuw solid. Om te voorkomen dat we de collision events ook bij dezen moeten aangeven, gebruiken we het parent mechanisme. Dit is een erg krachtig mechanisme die je zeker zou moeten leren om te gebruiken. Als een object A een parent is van object B, gedraagt B zich hetzelfde als A. Het erft al het gedrag van A (tenzij je dit met ander gedrag overschrijft). Ook worden botsingen met B hetzelfde behandeld als botsingen met A. Dus voor de kleinere blokken stellen we de parent in van het grotere blok. Op deze manier worden ze hetzelfde behandeld als het grotere blok.

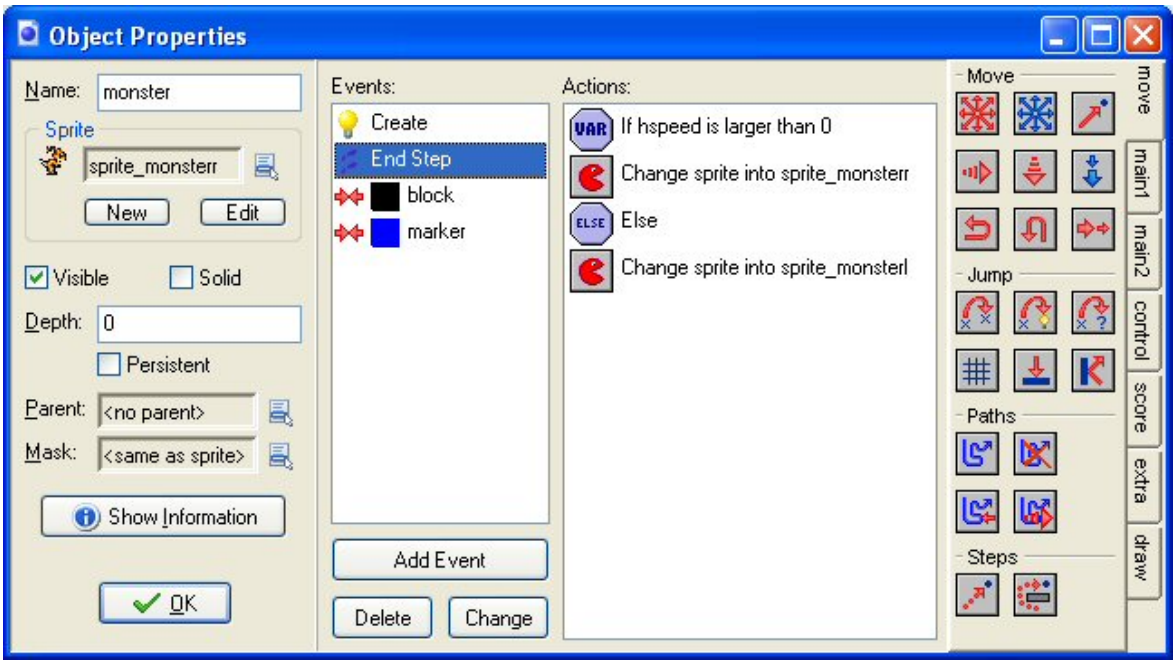
Gevaar en plezier

Gewoon rondspringen van platform naar platform is nogal saai. Je hebt zeker wat meer uitdagingen en doelen nodig. In dit gedeelte behandelen we er hier een paar van. Bekijk het spel `platform_3.gmk` voor het resultaat.

Monsters

Laten we eerst een paar monsters toevoegen. We gaan twee monsters maken, één die op een platform naar links en rechts beweegt en een ander die in de lucht naar links en naar rechts vliegt. De eerste kan geplet worden door bovenop hem te springen; de tweede moet altijd ontweken worden.

Laten we beginnen met het monster dat op de platformen beweegt. Hier hebben we twee sprites voor nodig, één van het monster dat naar links kijkt en een ander die naar rechts kijkt. Opnieuw is het beter om precise collision checking niet te gebruiken om dezelfde redenen als hierboven is aangegeven en kies een relevante bounding box. Nu maken we het monster object aan. In het create event laten we hem naar rechts bewegen met een bepaalde snelheid. Telkens als hij een muur aanraakt wordt zijn horizontale snelheid omgedraaid. Om de juiste sprite van het monster in te stellen gebruiken we het E**nd Step** event. Dit event vindt vlak voordat de instanties worden getekend plaats. Hierin stellen we de goede sprite in gebaseerd op de waarde van de variabele `hspeed` dat de horizontale snelheid aangeeft. Als het kleiner dan 0 is laten we het monster naar links kijken en anders laten we hem naar rechts kijken. (Als je de geregistreerde versie van *Game Maker* hebt kan je ook de actie om de afbeelding te spiegelen gebruiken. In dit geval heb je maar één sprite nodig.)



Om te voorkomen dat monsters van platformen afvallen, gebruiken we een ander object, welke we een marker noemen. Deze marker is een onzichtbaar blauw blok. Telkens wanneer een monster het raakt, draait hij zijn richting van beweging om. Het hebben van onzichtbare markers is een goede algemene truc om instanties bepaalde acties op speciale plaatsen in je room te laten verrichten. Naast het veranderen van de richting kun je markers gebruiken om te schieten, bommen te leggen, enz.

Als het karakter een monster raakt, moet het karakter doodgaan. Maar eigenlijk, zoals in de meeste platformspellen willen we het mogelijk maken voor het karakter om bovenop het monster te springen en hem te pletten. Dus in het collision event van het

karakter met het monster moeten we kijken of we het monster van boven raken om hem te pletten. Om dit uit te vinden verrichten we de volgende test:

```
vspeed > 0 && y < other.y+8
```

Het is waar als `vspeed` groter is dan 0, dus het karakter naar beneden beweegt, en het karakter dicht bij de bovenkant van het monster is, dus hij wordt inderdaad geraakt van boven. In dit geval moet het monster vernietigd worden. (In het voorbeeld veranderen we het monster in een plat dood monster, dat zichzelf na een tijdje vernietigt. Dit geeft een mooier grafisch effect.) In dit eenvoudige platformspel, komt het doodgaan van het karakter overeen met het herstarten van het level, wat door een paar simpele acties kan worden bereikt.

Het vliegende monster is nog makkelijker. We doen het op precies dezelfde manier. Alleen, in het collision event van het karakter met het vliegende monster, hoeven er geen testen te worden verricht omdat je een vliegend object niet kan pletten.

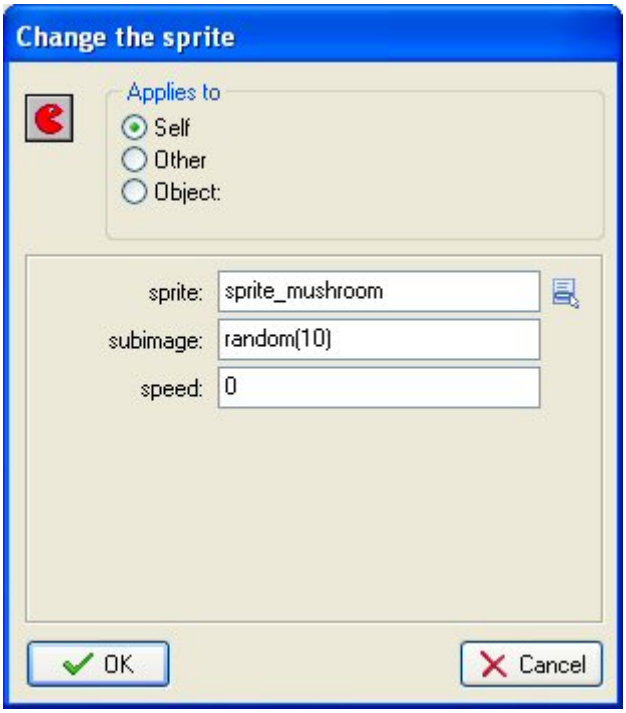
Je wilt misschien wat meer monsters toevoegen, bijvoorbeeld met verschillende snelheden, om dingen moeilijker te maken. Je kunt ook een monster of rots maken die naar beneden valt of naar boven en beneden beweegt. Gebruik gewoon je eigen fantasie.

Kuilen

De meeste platformspellen vereisen zorgvuldige timing van sprongen om te voorkomen dat je in kuilen valt. Normaalgesproken gaat het karakter dood als het in een kuil valt. Hiervoor voegen we een nieuw object toe, genaamd death (dood). Dit object is een rood blok dat ook niet zichtbaar is. Je kunt deze op de bodem van een kuil plaatsen. (In de room met tiles kan je daar een paar scherpe punten zetten.) In het collision event van het karakter met het death object moet er een geluid worden afgespeeld, een tijdje gewacht worden, en de room herstart worden. Je kunt ook kuilen maken die oneindig naar beneden gaan. In dit geval wil je gelijke acties toevoegen in het **Outside Room** event (bij other events) van het karakter, misschien een toegevoegde test of `y > room_height` om er zeker van te zijn dat het karakter naar beneden valt, in plaats van naar boven springt buiten het speelveld.

Punten verzamelen

De meeste platformspellen hebben een mechanisme waarmee de speler punten kan verzamelen. Normaalgesproken moet je bepaalde objecten oppakken of bepaalde dingen vangen. In onze example kan de speler paddestoelen verzamelen. Dus we maken een paddestoel object. Om een beetje variatie te geven, bevat de paddestoel sprite 10 verschillende paddestoel subimages. Het paddestoel object kiest er willekeurig één wanneer die gemaakt wordt, waarvoor we de actie **Change the sprite** gebruiken:



We stellen subimage in op `random(10). random(10)` is een oproep functie. Het geeft een random getal terug lager dan het opgegeven argument (dus in ons geval lager dan 10). We stellen de snelheid in op 0 om het rondcirkelen tussen de subimages te stoppen. In het collision event van het karakter met het paddestoel object spelen we een geluid af, vernietigen we het andere object (dat is de paddestoel) en voegen we 10 aan de score toe.

In sommige platformspellen heeft het verzamelen van dingen een belangrijkere functie dan gewoon je score te verhogen. Je zou bijvoorbeeld een extra leven kunnen krijgen wanneer je genoeg objecten hebt verzameld. Er zouden ook objecten kunnen zijn die je gezondheid herstellen (als je aanneemt dat monsters je niet doden maar alleen verzwakken), je beweging sneller maken, je sprong hoger maken, enz. Dit kan gemakkelijk worden toegevoegd.

Volgend level

Natuurlijk moet er een manier zijn om je level te beëindigen, zodat de speler naar het volgende level kan gaan. Hiervoor maken we een `levelexit` object. Wanneer het karakter daar aankomt ga je naar het volgende level. In de example is dit tamelijk eenvoudig gedaan. We voegen een test actie toe om te kijken of de volgende room bestaat. Als deze test waar is gaan we naar de volgende room. Anders wordt de highscore lijst getoond en wordt het spel herstart.

Je zou ervoor kunnen kiezen om het levelexit object alleen te laten verschijnen wanneer bijvoorbeeld alle paddestoelen zijn verzameld. Hiervoor verplaats je in het create event van het `obj_levelexit` object, het object naar positie –100,-100 (dus buiten het scherm). Nu kijken we in het step event van het object of het aantal paddestoel objecten gelijk is aan 0 (hier is een actie voor) en, als dat zo is, verplaatsen we het object terug naar zijn start positie (ook hier is een actie voor). Allemaal erg eenvoudig.

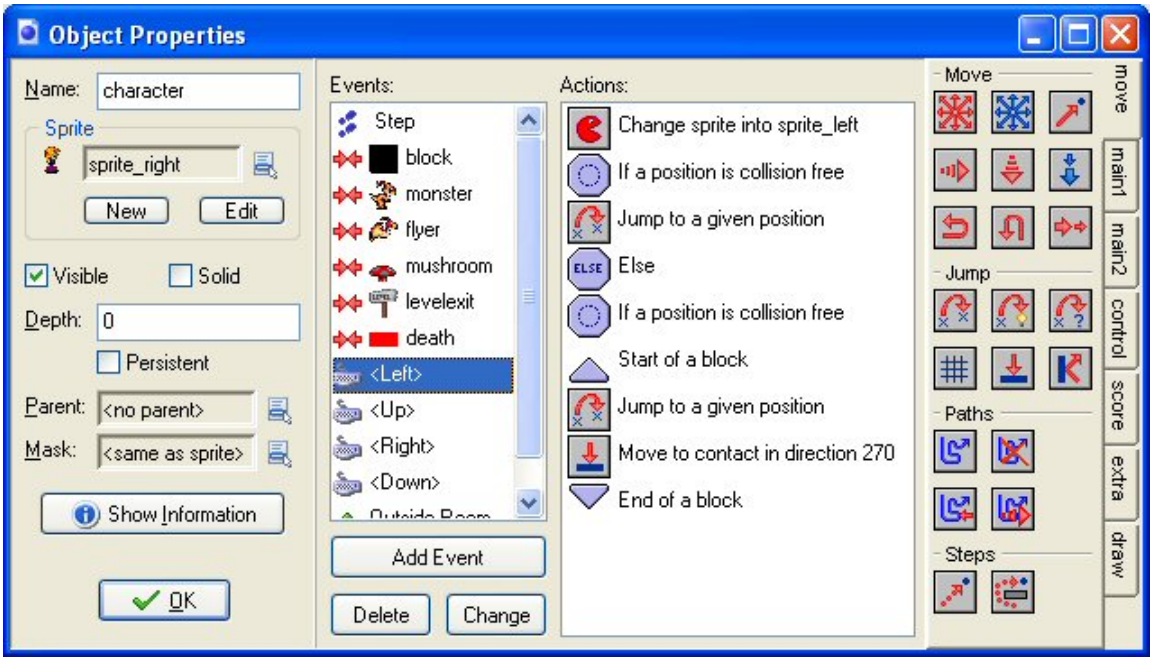
Meer bewegingen

Ons huidige platformspel heeft slechts een paar beperkte bewegingsmogelijkheden. Het karakter kan naar links en rechts bewegen, en hij kan springen. Om dingen interessanter te maken, voegen we enkele mogelijkheden toe. Het resultaat kan gevonden worden in het spel `platform_4.gmk`.

Hellingen

Het is leuk als de speler hellingen op kan lopen (naar beneden gaat automatisch door het vallen). Hiervoor moeten we de code in het event voor het linkerpijltje verplaatsen. We zetten daar het volgende:

In plaats van alleen te testen of de linkerpositie botsingvrij is testen we ook of de positie 8 pixels hoger botsingvrij is. Als dat zo is verplaatsen we het karakter daar naar toe en gebruiken we de actie voor het landen om hem naar de contact positie onder hem te bewegen. Dus het event ziet er zo uit:



Het rechterpijltje wordt op dezelfde manier behandeld.

Ladders

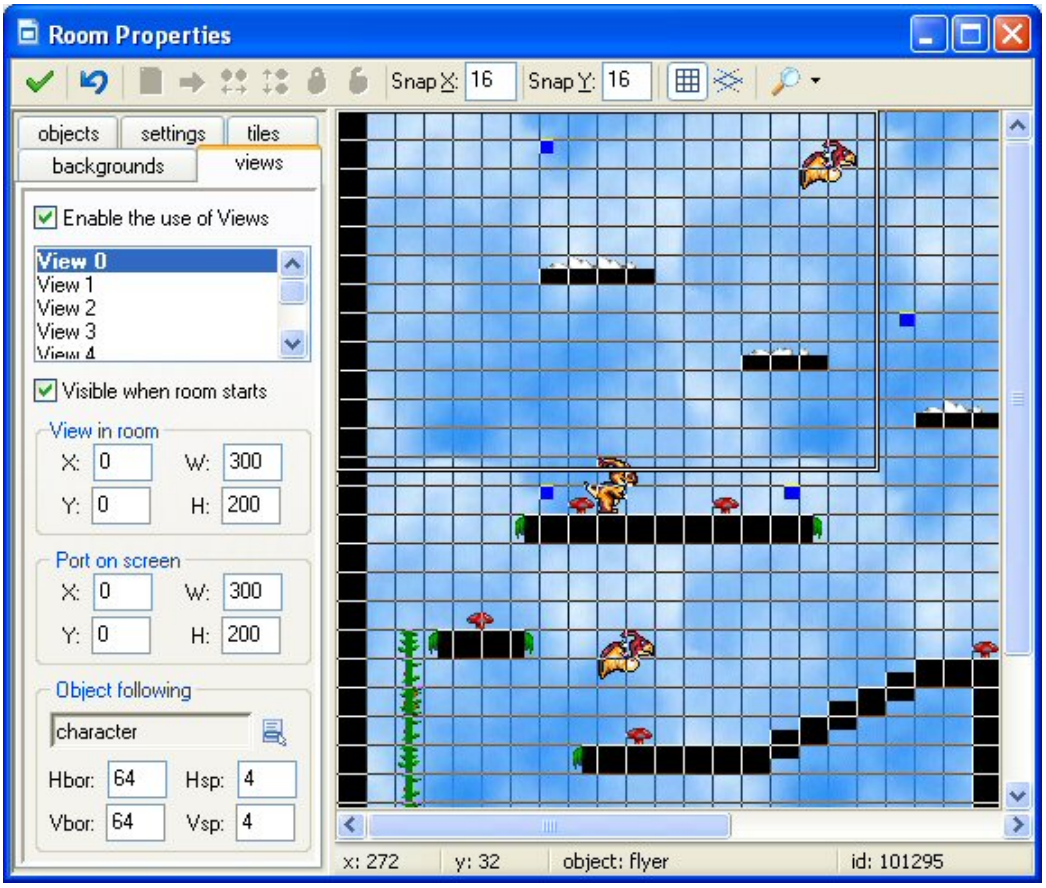
Mensen willen altijd ladders in een platformspel waarmee het karakter van het ene naar het andere platform kan bewegen. Dit vereist een beetje werk. Een ladder wordt vertegenwoordigd door een dun verticaal blok dat onzichtbaar (de echter ladder, klimplant of wat dan ook dat gebruikt wordt voor het klimmen wordt weer door middel van tiles getekend) en niet solid is. Wanneer het karakter niet met een ladder in contact is, moet de beweging zo zijn als hiervoor. Maar wanneer hij in contact is met de ladder moeten dingen anders gaan. Ten eerste moet het karakter niet naar beneden vallen. Dus in het step event moet we hiervoor een verandering maken door het toevoegen van een aantal acties, dat de verticale snelheid en de zwaartekracht op 0 zetten wanneer hij in contact met een ladder is. In dat geval stellen we ook de sprite in op de klimmende sprite.

Het tweede ding dat veranderd moet worden is het event voor het pijltje omhoog. Wanneer het karakter bij een ladder is, moet het pijltje naar boven hem naar boven bewegen, in plaats van te springen. Ook hiervoor moeten we een paar acties toevoegen. We testen of het karakter in contact met een ladder is en, als dat zo is, bewegen we hem een beetje omhoog. We gebruiken gelijke acties voor het pijltje naar beneden.

Het gebruiken van een view

Tot nu toe lieten we altijd de volledige room zien. Voor veel platformspellen is dit niet wat je wilt. In plaats daarvan wil je alleen een deel van de room zien, rondom het karakter dat je bestuurt. Dit maakt het spel uitdagender omdat de speler moet proberen om zijn weg door de platformen te vinden. Ook kan je prijzen verbergen op lastig bereikbare plaatsen in de room.

Gelukkig is dit enorm simpel te bereiken in *Game Maker*. Wanneer je een room ontwerpt, klik je op het tabblad **Views**. Klik op het aankruisvakje **Enable the use of Views** om het gebruik van views te starten. Selecteer de eerste view en kruis het vakje **Visible when room starts** aan om er zeker van te zijn dat deze view gezien kan worden. Geef het een breedte van 300 en een hoogte van 200 (of iets anders dat je wilt). (Als we de view het karakter laten volgen is het niet nodig om de linker en boven positie van de view in de room aan te geven. Omdat we maar één view gebruiken, hoeven we ook niet de x en y positie van de port op het scherm aan te geven.) Beneden kunnen we aangeven welk object er gevolgd moet worden. Hier kiezen we het karakter. De view zal nu automatisch bewegen om het karakter in het centrum te houden. We willen niet dat het karakter te dicht bij de rand komt. Hiervoor zetten we de **Hbor** en **Vbor** waardes op 64. Nu zal er altijd een 64 pixel gebied zichtbaar zijn rondom het karakter. Ten slotte, om een vloeiende view beweging te krijgen stellen we de maximale view snelheid in op 4. (Dit geeft ook een erg mooi effect aan het begin omdat het karakter langzaam in de view komt.) Dus de instellingen zullen er zo uitzien:



Het hebben van een view is mooi maar het maakt het scherm waarin dingen gebeuren nogal klein. Om dit te voorkomen, geven we in de **Global Game Settings** een verbeterde schaal van 200 procent aan. Je kunt spelen met deze waardes om het effect te krijgen dat je wilt.

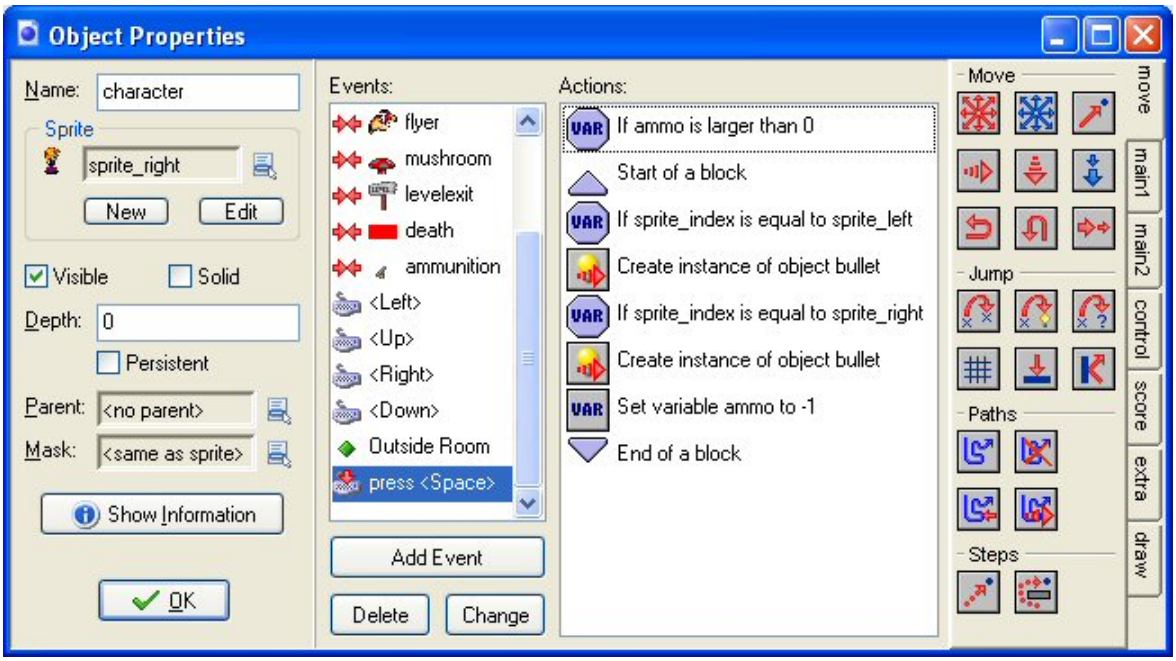
Een aantal verdere tips

Monsters schieten

De volgende stap is het voor de speler mogelijk maken om monsters te schieten. Om de dingen een beetje interessanter te maken, moet te speler eerst wat munitie vinden om te kunnen schieten. Hiervoor maken we een variabele genaamd `ammo` aan die aangeeft hoeveel munitie de speler heeft. In het **Create** event van het karakter stellen we deze in op 0 door gebruik te maken van de actie om de variabele in te stellen. Het munitie object heeft een simpele sprite en doet niets. Het wacht alleen tot het wordt opgepakt door de speler. Wanneer het karakter botst met het munitie object voegen we 10 toe aan de variabele `ammo` (stel het relatief in op 10) en vernietigen we de munitie instantie.

Vervolgens hebben we een kogel object nodig. Wanneer de speler op de spatiebalk drukt moet er een instantie van dit object gemaakt worden, ervan uitgaande dat er munitie is, en de waarde van de variabele `ammo` neemt af met 1. Maar er is één belangrijke kwestie. We willen dat de kogel schiet in de richting waar het karakter naar toe kijkt. Hiervoor kijken we naar de waarde van de variabele `sprite_index`. Deze variabele bevat de index van de sprite voor het karakter object. Hierop

gebaseerd maken we een kogel met de juiste bewegingsrichting. Wanneer we klimmen wordt er geen kogel gemaakt. (Schieten tijdens het klimmen is niet mogelijk.) Dus het spatiebalk event ziet er als volgt uit:

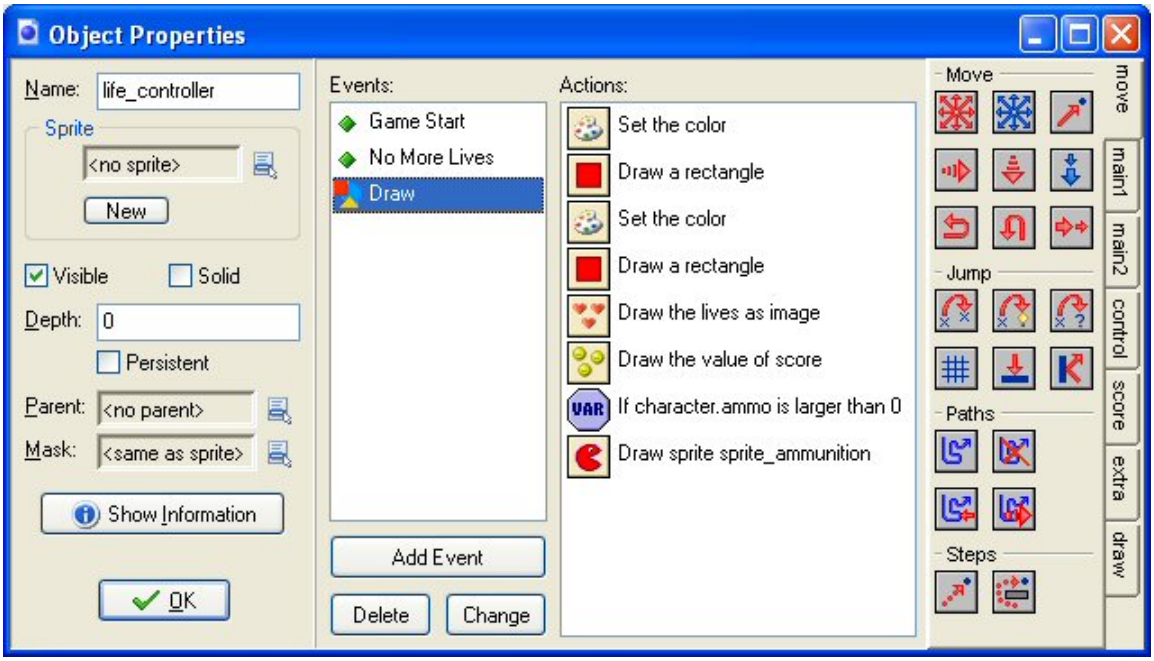


Nu blijft het vernietigen van de kogel over wanneer het een muur raakt of wanneer het buiten de room gaat en het doden van het monster wanneer de kogel hem raakt. Dit is allemaal eenvoudig. Zie het bestand `platform_5.gmk` voor de nieuwe versie van het spel.

Een score paneel

De speler heeft nu een score en munitie. We gaan hem ook een aantal levens geven. Het raken van een monster of het vallen in een kuil zal een leven kosten. Er is een gemakkelijk mechanisme voor levens in *Game Maker*. We gaan een speciaal `life_controller` object maken. Het heeft geen sprite nodig. In het **Create** event zet het het aantal levens op 3. Telkens als de speler doodgaat, verminderen we het aantal levens. In het **No More lives** event van de controller laten we de highscore tabel zien en herstart het spel.

Maar het zou ook mooi zijn als we het aantal levens, de score, de munitie, enz. kunnen zien. Hiervoor gaan we een klein paneel maken met deze informatie. We tekenen dit in het **Draw** event van het controller object. Er is hier wel een probleem. Waar zullen we het tekenen? We kunnen het niet tekenen op een bepaalde plaats in de room omdat de view verandert en we willen het paneel altijd in de view hebben. Gelukkig kunnen we de positie van de view opvragen. Dit is aangegeven door de 2 variabelen `view_xview` en `view_yview` die respectievelijk de linker positie en de boven positie van de view aangeven. Dus we kunnen het paneel met de informatie relatief aan deze positie tekenen. Hier zie je hoe het draw event van het controller object er uitziet:



Merk op dat we ook een plaatje tekenen wanneer de speler kan schieten. In het spel, dat kan worden gevonden in het bestand `platform_5.gmk`, resulteert dit in het volgende plaatje:

Wat nu?

De onderdelen hierboven hebben enkele dingen van de basis voor het maken van platformspellen uitgelegd. Nu is het jouw beurt. Je zal deze technieken en een aantal andere ideeën van jezelf moeten gebruiken om een echt mooi platformspel te maken. Onthoud dat het belangrijkste deel van platformspellen wordt gevormd door de levels. Begin met het maken van de levels één voor één. Speel ze totdat je er blij mee bent. Voer nu en dan wat nieuwe gameplay aspecten in. Hier zijn een paar toegevoegde ideeën die je kunt gebruiken:

- verschillende monsters, bijv. stuiterende ballen en monsters die schieten
- sleutels die je moet vinden om deuren te openen
- mijnen die je ergens kan plaatsen en die afgaan wanneer er een monster (of jijzelf) op staat
- water om in te zwemmen (dit veranderd de bewegingen totaal; geen zwaartekracht meer, of een langzaam oplopende zwaartekracht totdat je het oppervlak bereikt, beperkte tijd voordat je buiten adem raakt, luchtbellen om te pakken, enz.)
- muren en vloeren die je kunt vernietigen, bijv. door er tegen te schieten of door er met kracht op te springen
- trampolines die je hoger laten springen
- platformen die verschijnen en verdwijnen
- eenrichtingswegen
- bewegende platformen (dit is niet gemakkelijk!)
- …

Succes.

Verder lezen

Om verder te lezen over het maken van games met *Game Maker* kun je ons boek kopen:

> Jacob Habgood en Mark Overmars, *Leer jezelf MAKKELIJK... Games ontwerpen met Game Maker*, 2007, ISBN 978-90-5940-284-3.

Dit boek geeft stap-voor-stap een introductie van de vele aspecten van *Game Maker* en in dit proces maak je negen prachtige spellen die ook nog eens leuk zijn om te spelen.