Basic Bulletin

21ste jaargang maart 2014 Nummer 1

HCC BASIC interessegroep

Inhoud

Contacten

Functie	Naam	Telefoonnr.	E-mail
Voorzitter	Jan van der Linden	071-3413679	voorz@basic-gg.hcc.nl
Secretaris	Gordon Rahman Tobias Asserstraat 6 2037 JA Haarlem	023-5334881	secr@basic-gg.hcc.nl
Penningmeester	Piet Boere	0348-473115	penm@basic-gg.hcc.nl
Bestuurslid	Titus Krijgsman	075-6145458	t.krijgsman8@upcmail.nl
Redacteur	M.A. Kurvers Schaapsveld 46 3773 ZJ Barneveld	0342-424452	m.a.kurvers@hccnet.nl
Ledenadministratie	Fred Luchsinger	0318-571187	f.luchsinger@kader.hcc.nl
Webmaster	Jan van der Linden	071-3413679	j.vd.linden@kader.hcc.nl

http://www.basic.hcc.nl

Redactioneel

BBC BASIC is na de jaren ’90 een stuk verder gegaan. Ook al konden we al met routines werken, en dat was zeker in die tijd uniek, er zijn mogelijkheden bijgekomen. Nooit had ik dit verwacht. Zeker niet als ik het ook over de nieuwste mogelijkheden ga hebben: BBC BASIC for Windows.

In dit bulletin heb ik leuke tips voor u.

- Hoe noemen we de variabelen?
- De website Freeware, alle soorten gratis software.

Voor wie het leuk vind kan het onderwerp Sprites en objecten lezen. Hier leg ik u uit wat sprites zijn en hoe de objecten werken. Het programma Game Maker Studio laat ik dan als praktijkvoorbeeld zien. Er zit een scripttaal in die GML heet, maar het is helaas geen Basic script. Toch zit er wel wat van Basic in.

Marco Kurvers

BBC BASIC arrays, structuren en routines.

Gebruiker gedefinieerde routines en functies: PROC en FN

We hebben al gezien dat er ingebouwde opdrachten en functies zijn, dat de ruggengraat van BASIC wordt genoemd. Dergelijke voorbeelden zijn CLS die het scherm wist en LOG(X) die als resultaat de logaritme van X geeft. Wanneer uw programma's groter worden, zult u merken dat u steeds meer dezelfde regels gaat maken. Dit deel laat zien hoe we onze eigen opdrachten kunnen maken en hoe we ze kunnen aanroepen.

Stel dat u het scherm wilt schoonmaken en een titel wilt weergeven. Dat is simpel; twee regels code:

```
CLS
PRINT TAB(2);"BBC BASIC Tutorial"
```

Als we dit meerdere keren willen doen, zouden we de regels meerdere keren moeten kopiëren of we plaatsen het simpelweg in een blok met code die we dan kunnen aanroepen wanneer we willen. Hieronder ziet u hoe we het kunnen doen:

```
REM PROC demo
PROC_ScreenSetup
INPUT A$
PROC_ScreenSetup
END

DEF PROC_ScreenSetup
CLS
PRINT TAB(2);"BBC BASIC Tutorial"
ENDPROC
```

In andere BASIC dialecten is er geen groot verschil. Het enige is dat BBC BASIC andere sleutelwoorden heeft en niet de bekende SUB … END SUB kent.
Merk op dat de procedurenaam direct achter PROC staat, in dit voorbeeld met een onderlijning. PROC gevolgd door een spatie en dan pas de naam is niet toegestaan. U mag ook direct de naam opgeven zonder een onderlijning.

Een PROC kan zoveel waarden doorgeven als u zou willen. De regel is wel dat ze van hetzelfde type en van dezelfde volgorde moeten zijn als genoemd in de regel waarin ze zijn gedeclareerd. Hier ziet u bijvoorbeeld hoe u verschillende titels kunt weergeven door ze in de bovenste regels op te geven.

```
REM Passing a value to a PROC
PROC_ScreenSetup(5, "First screen")
INPUT A$
PROC_ScreenSetup(10, "Second screen")
END

DEF PROC_ScreenSetup(Col%,Title$)
CLS
PRINT TAB(Col%);Title$
ENDPROC
```

Wanneer een variabele wordt doorgegeven aan een PROC, is de procedure vrij om de lokale variabele te manipuleren in elke gewenste wijze. Als het werkt met een kopie, wordt de oorspronkelijke varia-

bele niet gewijzigd. Die manier van het doorgeven van gegevens wordt ‘bij waarde’ genoemd. Het bestaat in de meeste BASIC dialecten.

```
REM Call by value demo
MyString$ = "Hello, world"
PRINT "Value before PROC ";MyString$
PROC_DoSomething(MyString$)
PRINT "Value after PROC ";MyString$
END

DEF PROC_DoSomething(AString$)
PRINT "Value passed to PROC ";AString$
AString$="Goodbye, cruel world"
PRINT "Value after changing ";AString$
ENDPROC
```

Er zijn momenten wanneer we de inhoud van een variabele willen weten. Om dit te doen moeten we het RETURN statement in het PROC statement opgeven. Verander bovenstaande DEF regel in onderstaande regel en start de code opnieuw om het effect te zien:

```
DEF PROC_DoSomething(RETURN AString$)
```

Dit wordt ‘bij referentie’ genoemd.

Tip Zie deze mogelijkheden net zoals in andere BASIC dialecten, maar dan met de sleutelwoorden BYVAL en BYREF.

U kunt hele arrays en structuren in een PROC doorgeven. Om dit te doen gebruikt u gewoon de naam gevolgd door lege haakjes om aan te geven dat er niet een enkele variabele doorgegeven wordt. Als u vergeet DIM te gebruiken om de grootte van een matrix te vinden in de sectie op arrays, kunt u nu met een onmiddellijk gebruik dit doorgeven om te controleren dat er het juiste aantal dimensies van de juiste grootte zijn. Dit helpt grensfouten te voorkomen.

```
REM Passing an array
DIM IntArray%(2,3)
PROC_ArraySize(IntArray%())
END

DEF PROC_ArraySize(Array%())
PRINT "This array has ";
PRINT DIM(Array%());" dimensions."
ENDPROC
```

Structuren kunt u ook doorgeven in een PROC, maar de functie daarvan zal niet hetzelfde zijn als bij arrays. U moet de geassocieerde velden weten van de structuur.

```
REM Passing a structure
DIM Struct{A%,B$}
Struct.A%=23
Struct.B$="Twenty-three"
PROC_PrintStruct(Struct{})
END

DEF PROC_PrintStruct(St{})
PRINT St.A%
```

```
PRINT St.B$
ENDPROC
```

Arrays en structuren kunnen alleen maar doorgegeven worden bij referentie. De reden is dat een array of een structuur vaak groot kan zijn. Om een volledige kopie te maken zal duur zijn, zowel in termen van geheugen en de verwerkingstijd duurt het fysiek kopiëren van elke waarde lang. Dit betekent dat als u een waarde in een doorgegeven matrix of structuur wijzigt, de wijziging permanent is.

```
REM Passing a structure
DIM Struct{A%,B$}
Struct.A%=23
Struct.B$="Twenty-three"
PROC_PrintStruct(Struct{})
PROC_IncStruct(Struct{})
PROC_PrintStruct(Struct{})
END

DEF PROC_PrintStruct(St{})
PRINT St.A%
PRINT St.B$
ENDPROC

DEF PROC_IncStruct(St{})
St.A%=24
St.B$="Twenty-four"
ENDPROC
```

Lokale variabelen

Het idee dat we waarden in een PROC kunnen geven is een belangrijk onderwerp. Het betekent dat we routines kunnen schrijven die overdraagbaar tussen programma's zijn, zodra u een routine getest hebt kunt u het in uw volgende programma plakken, zet u het in een bibliotheek of geeft u het aan al uw vrienden van BBC BASIC zonder herontwikkeling. Om dit zo doeltreffend mogelijk te doen moeten we ervoor zorgen dat elke variabele alleen in de PROC gebruikt wordt en nergens anders. De doorgegeven waarden van buiten de procedure is al behandeld. Wat we nodig hebben is een manier om een variabele uitsluitend tot een routine te laten behoren. Het gaat nu om het idee dat we lokale variabelen tegenkomen.

Een lokale variabele is een variabele die bestaat terwijl de code binnen de procedure wordt uitgevoerd en alleen toegankelijk voor die procedure is. Er is een sleutelwoord nodig om BASIC te laten weten dat het variabele lokaal wordt behandeld. Het heet LOCAL. Een variabele als lokaal gedefinieerd komt tot stand wanneer de routine wordt aangeroepen, die kan dan overal binnen in de routine worden gebruikt en wordt verwijderd wanneer de routine wordt verlaten. Laten we een voorbeeld bekijken. Stel dat we een lijn van tekens willen trekken in onze PROC:

```
REM LOCAL demo 1
PROC_DrawLine("*")
END

DEF PROC_DrawLine(Char$)
LOCAL Count%
FOR Count% = 0 TO 79
  PRINT Char$;
NEXT Count%
ENDPROC
```

Lokale variabelen voldoen aan de regels van normale (global) variabelen met betrekking tot naamgeving en typen definiëren.

U mag ook arrays en structuren als LOCALs definiëren, maar om dit te doen wordt een extra DIM instructie vereist:

```
REM LOCAL array
PROC_A
END

DEF PROC_A
LOCAL MyArray%()
DIM MyArray%(10)
REM Do something ...
ENDPROC
```

Het LOCAL statement vertelt BASIC dat het ruimte moet reserveren voor een matrix. De DIM regel vertelt hoeveel. Een LOCAL structuur werkt op dezelfde manier:

```
REM LOCAL structure
PROC_A
END

DEF PROC_A
LOCAL MyStruct{}
DIM MyStruct{a%,b,c$}
REM Do something ...
ENDPROC
```

Private variabelen
Zodra u vertrouwd bent met lokale variabelen, zijn private variabelen eenvoudig. Een private variabele wordt op dezelfde manier gedeclareerd als een lokale variabele, maar in plaats daarvan wordt het PRIVATE sleutelwoord gebruikt. Het is geldig tijdens de uitvoering van de routine, maar net als een lokale variabele niet erbuiten. Het verschil is dat een private variabele niet vergeten wordt als een procedure voltooid is, zijn waarde wordt herinnerd en opnieuw gebruikt als de routine de volgende keer aangeroepen wordt.

```
REM PRIVATE variable
PRINT "Calling first time"
PROC_A
PRINT "Calling second time"
PROC_A
END

DEF PROC_A
PRIVATE MyInt%
PRINT "Value of MyInt% = ";MyInt%
MyInt%=MyInt% + 100
ENDPROC
```

Bij de eerste keer wordt MyInt% gemaakt en ingesteld op nul. Voor iedereen wordt de waarde afgedrukt. Vervolgens wordt MyInt% met 100 verhoogd. Als PROC_A voor de tweede keer wordt aangeroepen, heeft MyInt% de waarde van de vorige aanroep. Dit blijkt als de waarde opnieuw wordt afgedrukt.

U zult blij zijn te weten dat al de informatie over PROCs ook geldt voor door de gebruiker gedefinieerde functies – FN. Het enige verschil is dat FNs een waarde teruggeven naar de aanroepende regel, zoals de ingebouwde functies als SQR, kunnen zij ook in expressies worden gebruikt.

```
REM Squaring a number using FN
A=4
B=FN_Square(A)
PRINT A;" squared is ";B
END

DEF FN_Square(Num)
=Num^2
```

We zullen eerst kijken naar de definitie. Zoals kan worden gezien, vertellen we BASIC dat we een functie declareren met behulp van FN in plaats van PROC. Er is geen ENDFN. Het einde van de functie wordt aangegeven door de regel die begint met '='. De expressie wordt berekend en deze waarde wordt geretourneerd naar de aanroepende functie, regel 3 in ons geval.

In nummer 3 heb ik per ongeluk gezegd dat er wel een FN … ENDFN zou zijn. Foutje, u ziet: het is dus niet het geval.

In andere sommige BASIC dialecten moet het type van de retourwaarde opgenomen worden van de functienaam, zoals wanneer u een variabele declareert. BBC BASIC accepteert dat ook, maar het type dat bij de naam is opgegeven wordt niet gecontroleerd. Het heeft dus geen zin om dat te doen.

Zoals PROCs kunnen FNs ook lang en complex worden gemaakt. Zij kunnen andere PROCs en FNs aanroepen. De enige suggestie van beide routinetypes is, dat er een afsluitpunt is. Met meerdere = regels naar gelang van omstandigheden lijkt misschien een goed idee, maar het kan wel leiden tot problemen. Het is veel netter een lokale variabele te gebruiken om het resultaat te houden en vervolgens deze te retourneren aan het einde:

```
REM Vowel test
INPUT "Enter a letter: " Char$
IF FN_IsAVowel(Char$) THEN
  PRINT Char$;" is a vowel"
ELSE
  PRINT Char$;" is not a vowel"
ENDIF
END

DEF FN_IsAVowel(Ch$)
LOCAL Result%
IF INSTR("AEIOUaeiou", LEFT$(Ch$,1)) THEN
  Result%=TRUE
ELSE
  Result%=FALSE
ENDIF
=Result%
```

In plaats van dit, waar meer code wordt toegevoegd, zou in onderstaand voorbeeld elk punt in de rest van de code kunnen verdwalen:

```
REM ...
DEF FN_IsAVowel(Ch$)
IF INSTR("AEIOUaeiou", LEFT$(Ch$,1)) THEN
```

```
  =TRUE
ELSE
  =FALSE
ENDIF
```

U kunt alleen enkele waarden resulteren, geen arrays of structuren. (Er is een methode om dit te doen, maar dat valt zeker in het domein van geavanceerd programmeren.) Als u een waarde terug wilt geven, is het beter om het hele geval als een parameter door te geven en in de functie te manipuleren.

Tips!

1. In de meest triviale programma's zult u verschillende PROCs en FNs tegenkomen. Om naar de definitie van de routine te zoeken die u probeert te vinden, klikt u met de rechter muisknop ergens in de editor en de PROcs en FNs van het momenteel geopende programma worden vermeldt aan de onderkant van het popupmenu. Als u naar een PROC of FN wilt, kunt u er een aanwijzen en klikken.
2. Het zal vrij duidelijk zijn dat u twee PROCs of FNs niet met dezelfde naam kunt hebben, maar BB4W zal niet tegenstribbelen. Als BASIC twee routines met dezelfde naam vindt, zal het gebruik maken van de laatste ervan en de rest negeren. Met een klein programmaatje is dat gemakkelijk te herkennen. Wanneer de programma's groeien met meer dan één pagina, wordt het moeilijk om ze op te sporen en als u library's gebruikt waarmee u uw code verdeeld, krijgt u alleen maar meer hoofdpijn. Als uw code, die u aan het bewerken bent geen effect heeft, kijk dan eerst of u per ongeluk dubbele namen gebruikt.

Verder zal alles, zoals LOCAL en PRIVATE variabelen, ook in FN definities precies zo werken als in PROC definities.

BASIC programmeertips.

Variabelennamen

We gebruiken variabelennamen om aan te geven wat we bedoelen. Deze bedoelingen zijn niet altijd goed te begrijpen. We weten wel wat we bedoelen met X en Y voor coördinaten en I en J te gebruiken voor lussen, maar als er staat: 'Tekst', dan is het met de bekendheid van de naam snel gedaan.

Gebruiken we drie tekstvariabelen, dan hebben we gauw de neiging om deze Tekst1, Tekst2 en Tekst3 te noemen of we maken gebruik van een array die Tekst heet. Maar alsnog is de naam nihil en het gebruik ervan is dan ook sterk af te raden.

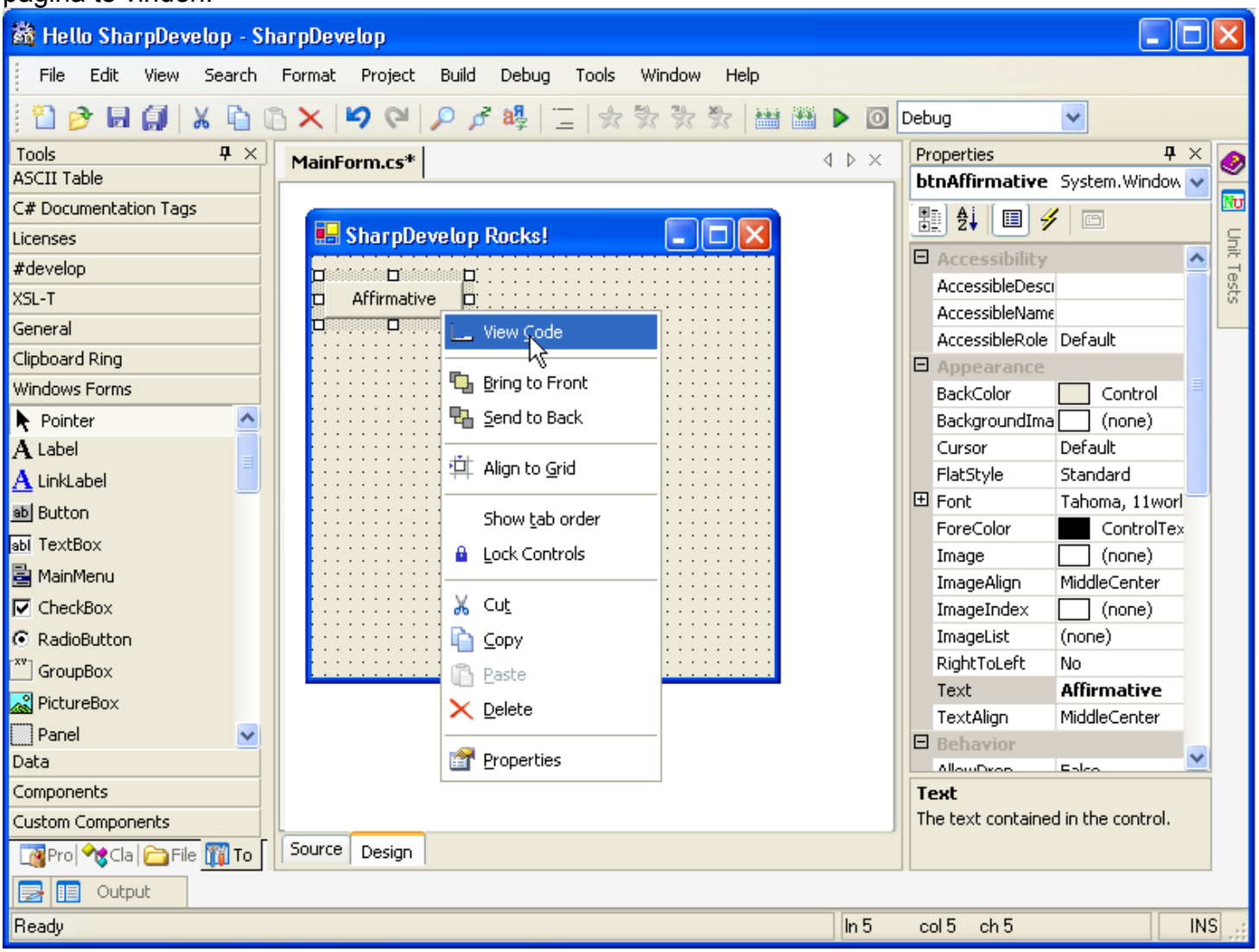
Hebben we drie variabelennamen die Naam, Adres en Woonplaats heten, dan zijn we al meteen een stuk verder. Nu is het voor iedereen te begrijpen wat er met deze variabelen bedoeld wordt. Declareer zulke variabelen ook niet los in het programma, maar pak het in een structuur. Een recordtype of klassentype die we bijvoorbeeld Persoon kunnen noemen.

De Freeware pagina

Op deze pagina kunt u allerlei gratis software vinden, ook programmeertalen. Just Basic is ook te vinden en SharpDevelop waar ik een afbeelding van zal laten zien.

Het adres van de site is: http://www.pkort.nl/freeware/programmeren.htm

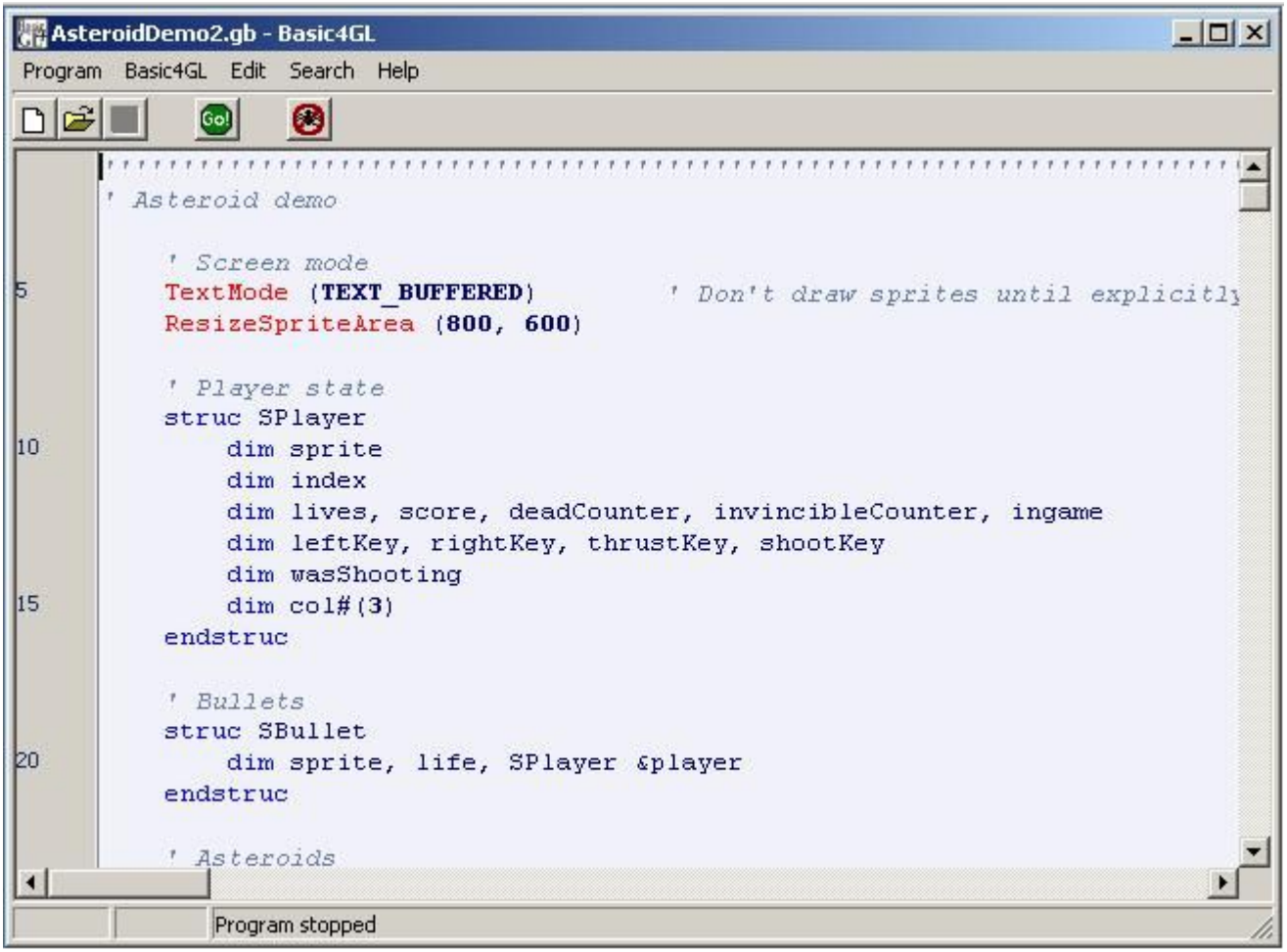
Doordat er ‘programmeren.htm’ staat, zal meteen de pagina over programmeertalen worden geopend. Links van de pagina ziet u een lijst van nog meer soorten pagina’s. Zo is er bijvoorbeeld een Game pagina te vinden.



SharpDevelop:
Een IDE om met C# en VB.NET projecten op het Microsoft .NET platform te kunnen programmeren.
OS : Windows (En natuurlijk het .Net Framework)

Het is dus mogelijk om met twee programmeertalen op één IDE te kunnen werken, en C# (C-Sharp) kan goed met Visual Basic .NET samen werken en het Framework library delen.

Hieronder is een afbeelding van een Basic programmeertaal speciaal geschikt om OpenGL Games te programmeren.



Basic4GL:
Een "basic" programmeertaal voor het maken van 2D en 3D spellen in OpenGL.
Een compleet pakket met een IDE, voorbeelden en uitleg.
OS : Windows

Basic4GL is een easy to use programmeertaal om grafische programma's, utilities, games en demo's te schrijven, zonder gebruik te hoeven maken van de achtergrond instellingen voor OpenGL en de Plug-Ins.

Meer weten?

Misschien vindt u het wel leuk om meer over games programmeren te weten, zoals wat het verschil is tussen sprites en objecten en hoe we om gaan met collisions (botsingen). Jammer genoeg zit er geen Basic dialect in Game Maker. Toch zou ik graag meer willen vertellen over Game Maker, want u kunt er leuke dingen mee doen. Het script taal is C#, maar het is ook mogelijk om zonder te programmeren games te maken door gebruik te maken van de muis en de IDE.

Bewaar ook de Freeware site in uw favorieten, want u kunt op die site veel mogelijkheden en tips vinden.

Ik kan programmeren.

In de voorgaande jaren zijn er af en toe stukken geweest over dit onderwerp. Ik heb toen gezegd: “Heeft u de lessen programmeren gevolgd? Prima, maar om dan gelijk van start te kunnen met grote projecten, is echter andere koek.”

Een programma schrijven is een route naar een doel die uit meerdere wegen naar Rome kan bestaan. Dat komt alleen maar omdat de programmeertaal u de kans geeft meer mogelijkheden te geven. Welke weg u kiest maakt niet uit. Een langere weg kost meer werk en tijd (en ook geld).

Waarschijnlijk weet u nu dat het leren programmeren niet iets betekent wat u daarna op het bord krijgt. Eerst wordt de programmeertaal geleerd en daarna “leert” u verder om een dergelijk programma te schrijven. De statements kent u al, maar de feitelijke structuren die in het programma moet passen staan er altijd los van.

Een voorbeeld dat ik kan laten zien is een lus. We leren daar de statements van, zoals FOR en NEXT, WHILE en WEND en nog meer lus-statements mocht het BASIC dialect er meer van hebben. Maar alleen maar de statements kennen is niet voldoende. We moeten weten wat we in een lus willen ‘stoppen’. Wat voor code moet er herhaaldelijk uitgevoerd worden … totdat het niet meer uitgevoerd mag worden.

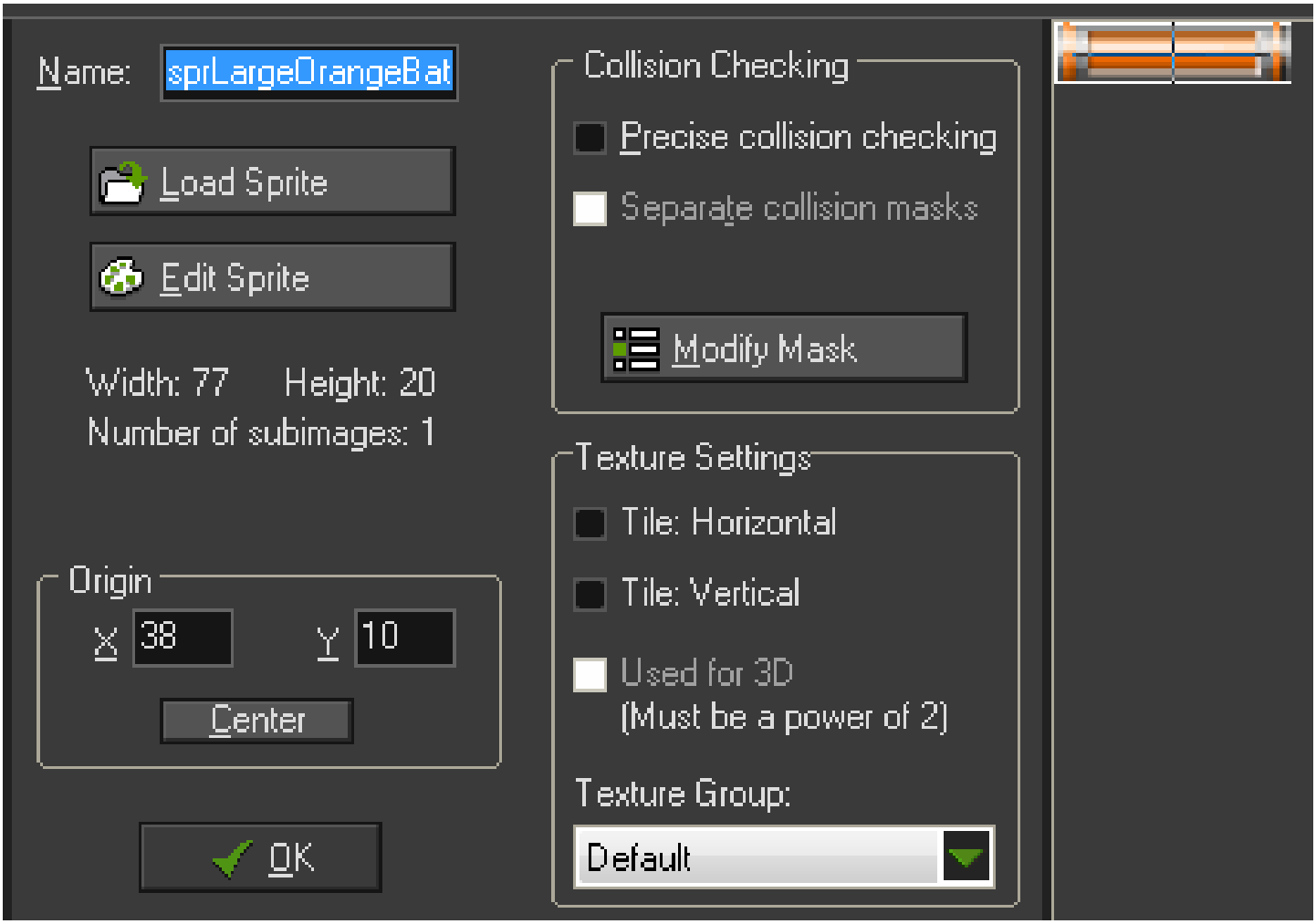
BASIC over de brug of in de sloot?
Er bestaan lus-structuren die kunnen controleren of er aan een bepaald doel is voldaan, bijvoorbeeld over de brug lopen. Is de overkant al bereikt? Zo niet, herhaal dan het lopen over de brug. Er is ook een lus-structuur die zelf bepaald hoe lang de brug is en waar dus niet op gecontroleerd kan worden. Die lus is de FOR … NEXT lus. Het TO statement achter de FOR bepaald met een waarde of expressie hoeveel keren er over de brug gelopen kan worden. We kunnen achter TO echter geen conditie gebruiken. De conclusie is dat de FOR lus alleen een beginwaarde en een eindwaarde kent.

Toch is er een mogelijkheid om de lus eerder te verlaten. Ook in een FOR … NEXT lus kunnen we controleren of er aan een voorwaarde (doel) is voldaan. Deze keer staat deze voorwaarde niet aan de FOR of NEXT statements, maar in een IF … THEN beslissing-structuur in de lus code. Is de voorwaarde waar (de overkant is bereikt), verlaat dan de lus. Dat kan met een GOTO statement, of als andere BASIC dialecten het ook kennen, met een EXIT FOR.

Zoals de titel van deze paragraaf al zegt, kan die manier om een lus te verlaten gevaarlijk zijn. Is bijvoorbeeld de conditie verkeerd geschreven waardoor het resultaat averechts werkt, dan zal BASIC in de sloot belanden, omdat de overkant niet bereikt wordt.
Dan zult u wel zeggen: Wacht eens, en als de lus wel goed werkt en zonder eruit te springen het doel bereikt wordt dan gaat de code toch ook gewoon verder na de lus?
Inderdaad, maar dan zal BASIC veilig op de overkant staan en niet in de sloot belanden. Daarom is het een goed idee om te bepalen welke soort lus u nodig heeft zodat de code correct herhaald wordt en de conditie of begin- en eindwaarde juist is.

Sprites en objecten.

Wat is een sprite?
Een afbeelding (image) die we in een spel gebruiken wordt een sprite genoemd. Een sprite heeft geen gegevens, daarom noemen we het een visueel gerepresenteerd plaatje.



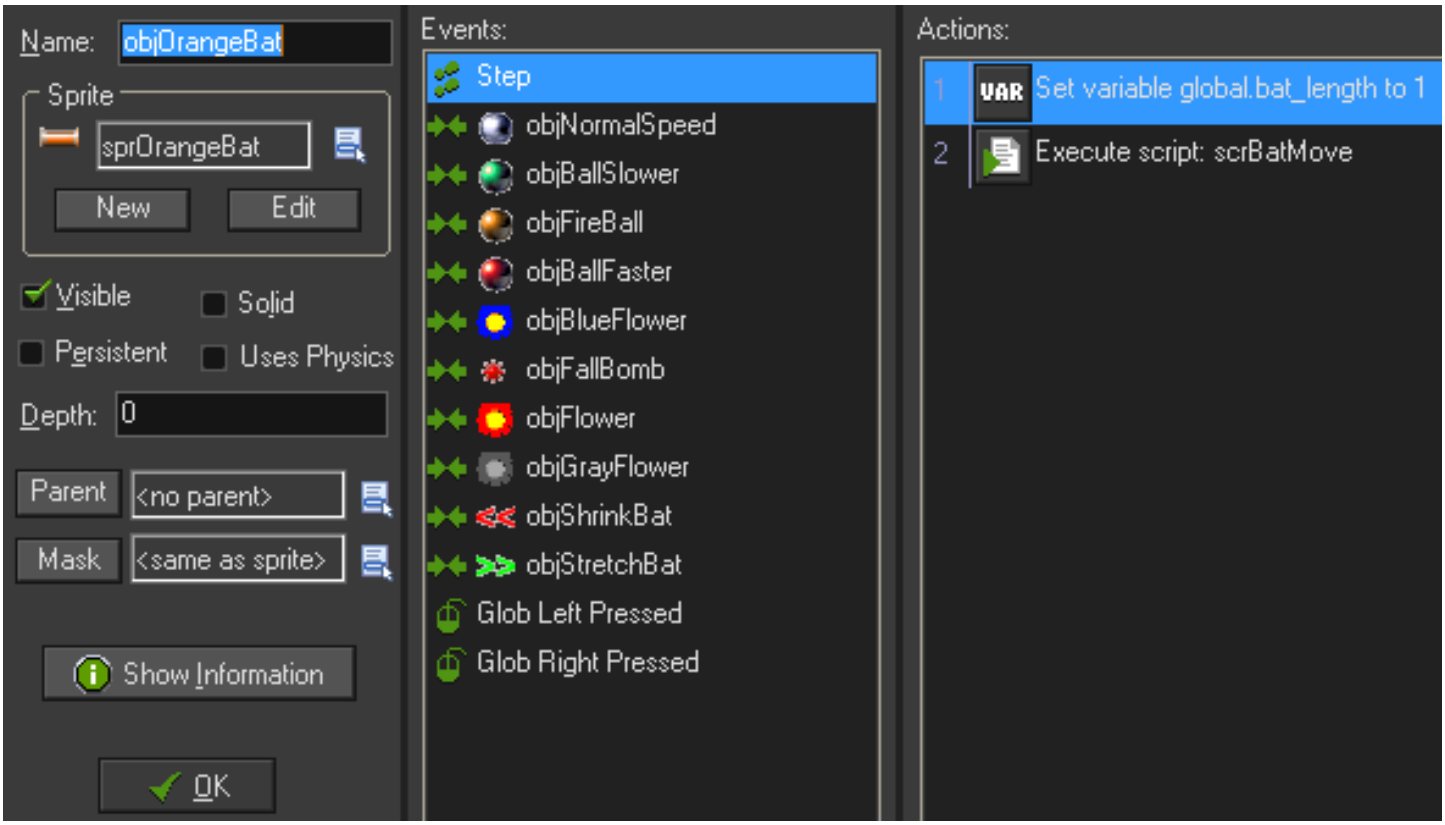
In Game Maker kunnen we sprites maken, inladen en bewerken.

We kennen ook een set plaatjes, die werken als een animatie. Onderstaande vier plaatjes vormen een sprite in Pacman waarbij het naar rechts beweegt.

Uit het voorbeeld van Game Maker kunnen we zien dat de Bat sprite uit één image bestaat. De *Number of subimages* bepaald uit hoeveel plaatjes de sprite bestaat. Mochten het er meer zijn, dan nog laat Game Maker één van de images zien. Gaan we naar *Edit Sprite*, dan zullen alle subimages weergegeven worden.

Heeft een sprite meerdere plaatjes, dan is het bestand een GIF bestand. Een GIF bestand moeten we niet zien als een echte video-animatie, want we maken alleen maar een set van plaatjes om ervoor te zorgen dat de sprite kan bewegen. Hoe meer plaatjes in een sprite, des te gladder of fijner de animatie wordt. Dit betekent dan wel dat het GIF bestand erg groot wordt.

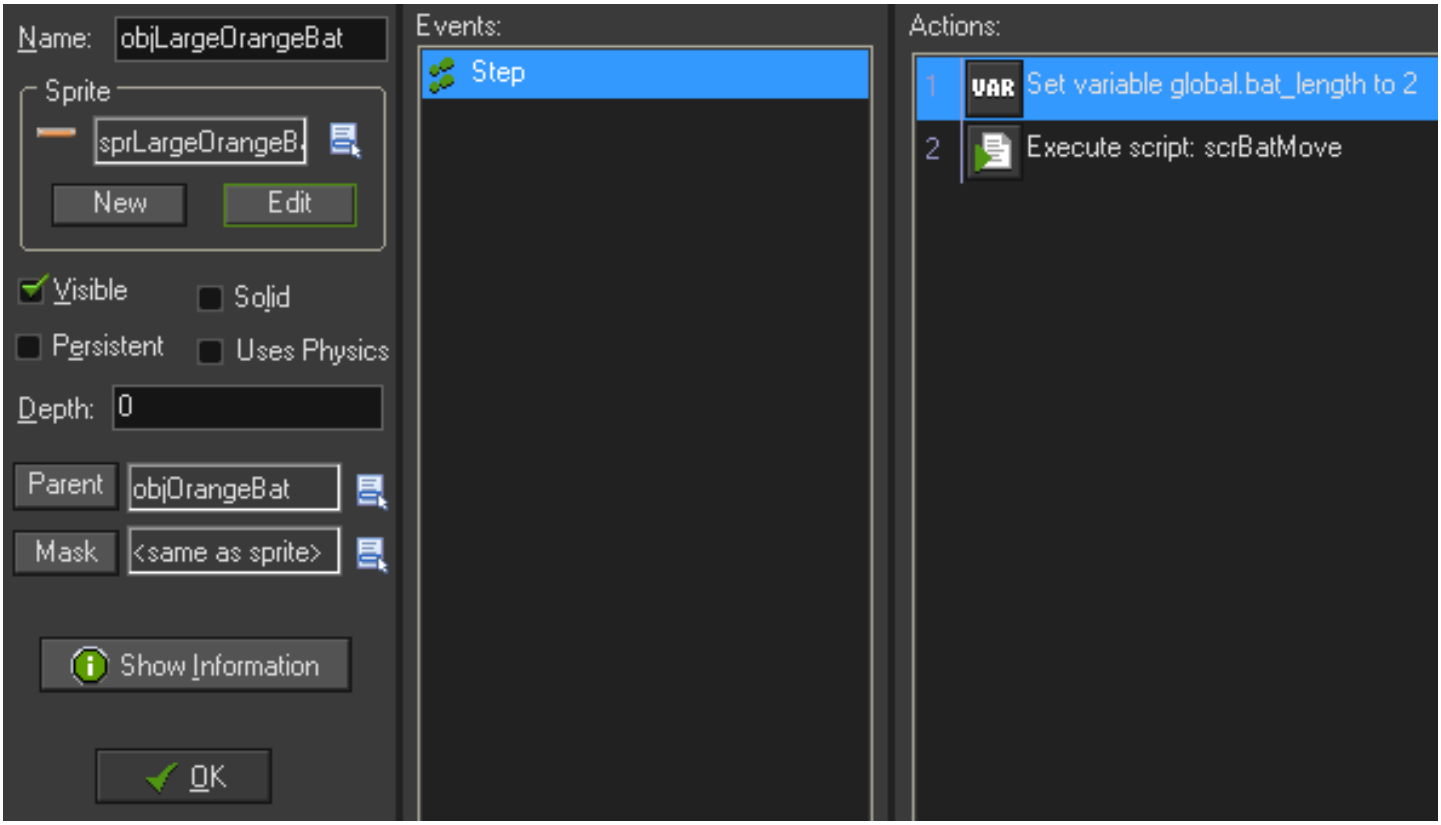
Omdat een sprite geen gegevens heeft, moeten we iets hebben om het te kunnen weergeven. Dat kan op twee manieren. We kunnen een sprite direct weergeven of er een object van maken. Het direct weergeven van de sprite betekent niets. Het kan dan ook niets en alles wat erbij hoort, zoals de besturing, kan niet aan een sprite gekoppeld worden. Om een besturing te kunnen maken, zouden we de sprites in variabelen of arrays op kunnen slaan en daarmee in scripts (zie later) het kunnen programmeren. In plaats van zelf de spritebesturing te programmeren, kunnen we het beter overlaten aan de objecten van de sprites en zelf de instanties van de objecten creëren.



Hier worden de objecten gemaakt en aan de juiste sprites gekoppeld. Elk object heeft zijn eigen besturing. De besturing bestaat uit events (gebeurtenissen), maar kan ook eigen geprogrammeerde scripts hebben.

Als u kijkt bij de knop *Parent*, dan ziet u er staan **<no parent>**. De objecten kunnen als een library werken. Meerdere objecten kunnen het Parent object (vader object) erven. Dat heeft voordelen, net zoals we de klassen in Basic maken.

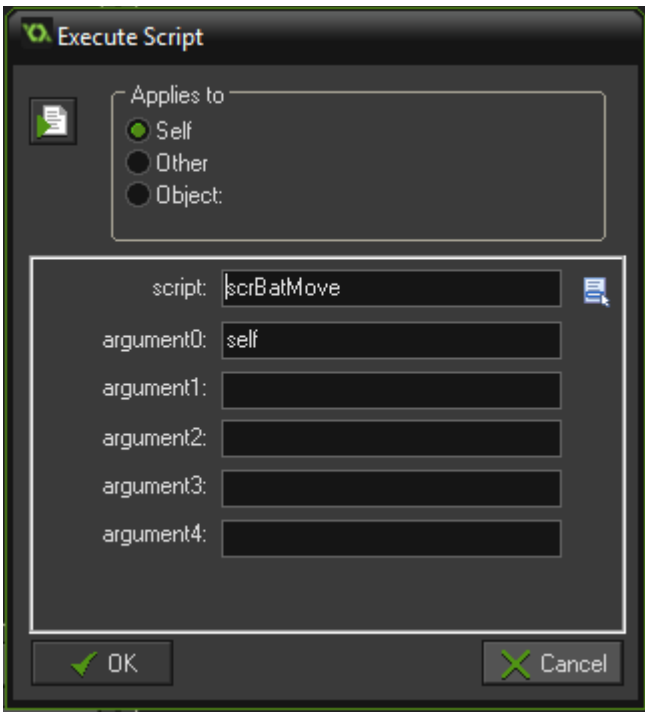
Deze Bat is echter niet die we van het sprite formulier zagen. Dat is namelijk onderstaand formulier: We zien dan ook dat het object wordt geërfd van het Parent object. We kunnen bij de Parent knop elk object als vader object gebruiken, als ze maar van elkaar overeenkomen. Probeer bijvoorbeeld niet het Bat object te erven van een Brick object.



Bij dit object ziet u een andere Bat sprite. Deze is langer (Large) zoals u bij Name ziet.

Dit object heeft maar één event. De rest van de events zijn ook nodig, maar dat is nou precies het voordeel van geërfde objecten. We hoeven alle events niet in elk object opnieuw te maken. Door bij Parent het vader object te kiezen, worden alle vader object events geërfd.

Bij de Step event, die hier niet geërfd wordt, wordt de waarde ingesteld die bepaalt om welk Bat het gaat. Het is een globale variabele *bat_length*. In de tweede regel wordt een script code aangeroepen. Een soort subroutine die ook parameters kan aannemen.

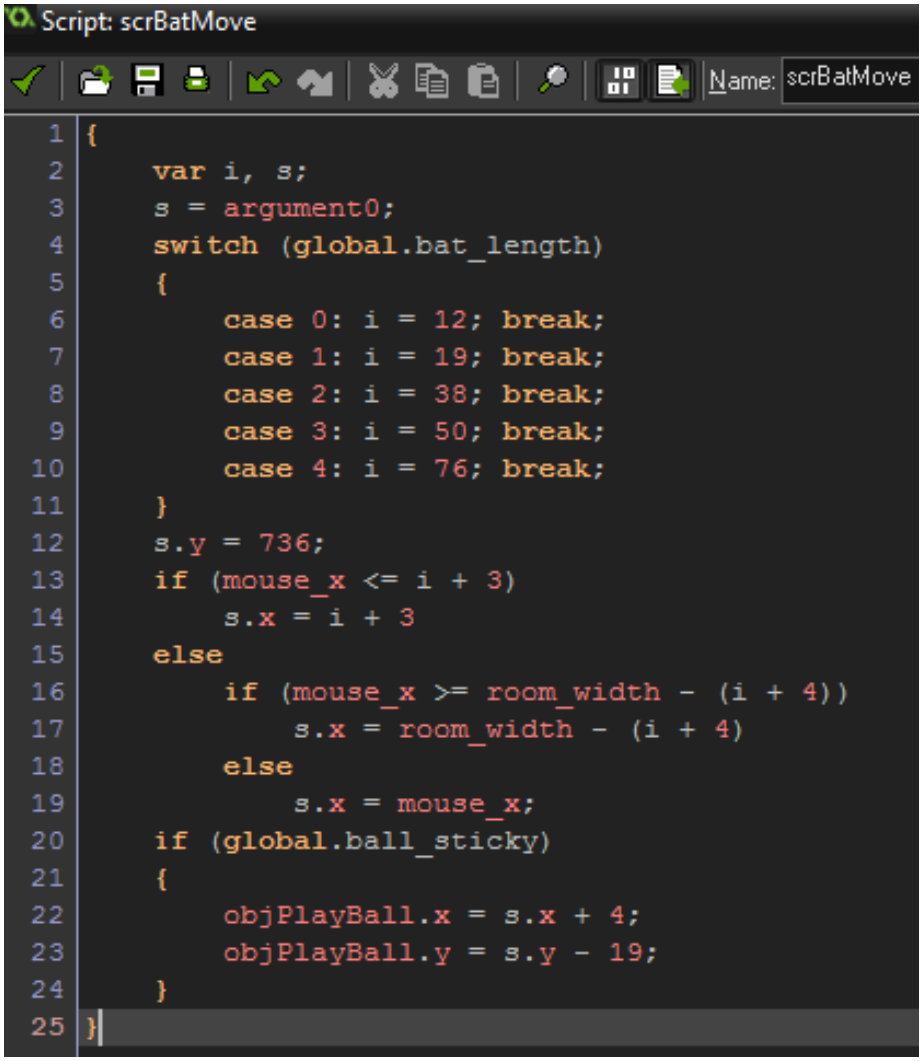


Het script **scrBatMove** wordt gekozen die voor de muisbesturing moet zorgen. De muisbesturing was echter geen code van Game Maker zelf. Daarom moest ik hem zelf schrijven.

Het script heeft ook het juiste Bat object nodig voor de besturing. Dat is de eerste parameter die als argument0 de objectnaam **self** meekrijgt. In Basic is dat het object **Me**.

Het volgende voorbeeld laat de script code zien. Hoewel het niet op Basic lijkt, kent GML (Game Maker Language) wel wat Basic sleutelwoorden en ook Pascal sleutelwoorden.

Zoals u ziet is het geen grote programmeertaal, vandaar dat het een script is. Het bestaat grotendeels uit C# code, maar het kent ook andere taal statements.



In de tweede regel worden twee lokale variabelen gedeclareerd. U kunt **var** vergelijken met **Dim**. Daarna wordt de doorgegeven parameter als **argument0** toegekend aan variabele **s**.

De lengte van de Bat wordt hier bepaald door een **switch** (in Basic kennen we het **Select Case** statement), maar we moeten achter elke case een **break** gebruiken. Doen we dat niet, dan zullen de andere cases na de juiste case ook uitgevoerd worden.

In regel 12 wordt bepaald op welke hoogte de Bat moet liggen. In regel 13 wordt gecontroleerd op welke plaats de muiscursor ligt. De eigenschappen **mouse_x** en **mouse_y** bepalen de mouse positie. We hebben voor de Bat besturing alleen mouse_x nodig om de Bat horizontaal te kunnen bewegen. Is de positie kleiner of gelijk aan variabele **i** (rechterkant van de Bat) plus 3 dan ligt de Bat aan de linkerkant van het scherm. Zo niet, dan moet daarna ook nog gecontroleerd worden of de positie groter of gelijk is aan de breedte van het scherm min de lengte van de Bat (gegeven door **i)** plus 4. Deze formule zorgt ervoor dat er vanaf de linkerkant van de Bat vergeleken wordt alsof het lijkt dat de Bat tegen de rand van het scherm komt. Zijn beide condities echter onjuist, dan zal de Bat ergens op het scherm zijn en zal er een nieuwe mouse_x positie aan **x** van de Bat worden toegekend.

Het laatste **if** statement is er een wat voor u een programmeur-geheim verklapt. Als u een dergelijk spel speelt en de bal staat op ‘plakken’, dan lijkt het net alsof de bal vast blijft zitten aan de Bat. De speler wordt met deze code natuurlijk voor de gek gehouden, want als de bal inderdaad op de Bat moet plakken (variabele **ball_sticky**) dan zal het bal object **objPlayBall** op de juiste positie van de Bat worden geplaatst en meebewegen met de muis net zoals de Bat doet.

Basic, Pascal en C(#)
De scripttaal GML (Game Maker Language) kent ook andere statements. In het voorbeeldscript ziet u geen **then** statements na de **if** regels. Toch wordt dat wel geaccepteerd. GML kent gewoon een coderegel met een **if** *conditie* **then**. Ook kunnen zowel taal C operatoren als Basic operatoren gebruikt worden. De C operator **&&** is hetzelfde operator als het Basic operator **and**. Ook de codeblokken kunnen vervangen worden in Pascal codeblokken. De accolades **{** … **}** werken op dezelfde manier als **begin** … **end**. Helaas kent GML geen **end if** statement om een blok te beëindigen. De puntkomma’s blijven in de code wel verplicht. Ze moeten er altijd staan om coderegels af te sluiten.

Wilt u meer weten van Game Maker? Kijk op www.yoyogames.com.

Sprite-strips en background-tiles
Om meerdere sprites in te kunnen laden die met een bepaald object te maken hebben, worden ook wel sprite-strips gebruikt. Een strip is een afbeelding die uit meerdere sprites bestaat. In Game Maker kunnen deze sprites eenvoudig eruit geknipt worden zonder dit handmatig te moeten doen. Wanneer het indelen van de sprite grootte en de afstanden van de sprites gebeurd is, worden de sprites geknipt en in losse images gezet. De images kunnen apart opgeslagen worden of als sprite indexen gebruikt worden. Tijdens het maken van het spel kan dan elke image bepaald worden door een indexnummer te kiezen.

Willen we Pacman maken, dan kunnen we het sprite object muren op een room (het speelveld) plaatsen. Het probleem is dan wel dat we elke muur (horizontaal, verticaal en een bocht) apart als objecten moeten aanmaken. Gelukkig hoeft dat niet. Net als bij sprite-strips bestaan er background-tiles. Dat zijn afbeeldingen waar we meerdere sprites in kunnen vinden, maar speciaal bedoeld zijn om ze als tegels op de room te kunnen gebruiken. We hoeven deze niet als objecten aan te maken, maar om de tegels te kunnen laten werken hebben we een mask nodig. Dit is een object dat we als een raampje ter grootte van de tegel nodig hebben. Plaatsen we geen maskers op de tegels, dan zou Pacman dwars door de muren kunnen bewegen. Voor het masker heeft u alleen de botsing event nodig, verder niet. Bovendien kunnen alle tegels het zelfde masker gebruiken. Daarom is het werken met background-tiles zo handig.

Samenvatting
Dit onderwerp ging even niet alleen over Basic. Ook al had ik het over een ander soort programma met een C# scripttaal; wat ik u hier uitgelegd heb was om te begrijpen hoe sprites en objecten in elkaar zitten en hoe men er mee omgaat. Voor de praktijk uitleg nam ik daarom als voorbeeld Game Maker.

Zou u meer willen weten? Via Google kunt u meer vinden over sprites, maar natuurlijk ook in yoyogames.com zelf kunt u informatie vinden.

Cursussen

Liberty Basic:
Cursus en naslagwerk, beide met voorbeelden op CD-ROM, € 6,00 voor leden. Niet leden € 10,00.

Qbasic:
Cursus, lesmateriaal en voorbeelden op CD-ROM, € 6,00 voor leden. Niet leden € 10,00.

QuickBasic:
Cursusboek en het lesvoorbeeld op diskette, € 11,00 voor leden. Niet leden € 13,50.

Visual Basic 6.0:
Cursus, lesmateriaal en voorbeelden op CD-ROM, € 6,00 voor leden. Niet leden € 10,00.

Basiscursus voor senioren, Windows 95/98,
Word 97 en internet voor senioren, (geen diskette). € 11,00 voor leden. Niet leden € 13,50.

Computercursus voor iedereen: tekstverwerking met Office en eventueel met VBA, Internet en programmeertalen, waaronder ook Basic, die u zou willen leren.
Elke dinsdag, woendag en vrijdag in buurthuis Bronveld in Barneveld van 19:00 uur tot 21:00 uur op de dinsdag en van 9:00 uur tot 11:00 uur op de woensdag en vrijdag. Kosten € 5,00 per week.
Meer informatie? Kijk op 'http://www.i-t-s.nl/rdkcomputerservice/index.php' of neem contact op met mij.

Computerworkshop voor iedereen; heeft u vragen over tekstverwerking of BASIC, dan kunt u elke 2de en 4de week per maand terecht in hetzelfde buurthuis Bronveld in Barneveld van 19:15 uur tot 21:15 uur. Kosten € 5,00.
Meer informatie? Kijk op ‘http://www.buurthuisbronveld.nl’ of neem contact op met mij.

Software

Catalogusdiskette,	€ 1,40 voor leden. Niet leden € 2,50.
Overige diskettes,	€ 3,40 voor leden. Niet leden € 4,50.
CD-ROM’s,	€ 9,50 voor leden. Niet leden € 12,50.

Hoe te bestellen

De cursussen, diskettes of CD-ROM kunnen worden besteld door het sturen van een e-mail naar penm@basic-gg.hcc.nl en storting van het verschuldigde bedrag op:

ABN-AMRO nummer 49.57.40.314
HCC BASIC ig
Haarlem

Onder vermelding van het gewenste artikel. Vermeld in elk geval in uw e-mail ook uw adres aangezien dit bij elektronisch bankieren niet wordt meegezonden. Houd rekening met een leveringstijd van ca. 2 weken.
Teksten en broncodes van de nieuwsbrieven zijn te downloaden vanaf onze website (http://www.basic.hccnet.nl). De diskettes worden bij tijd en wijlen aangevuld met bruikbare hulp- en voorbeeldprogramma's.
Op de catalogusdiskette staat een korte maar duidelijke beschrijving van elk programma.

Alle prijzen zijn inclusief verzendkosten voor Nederland en België.

De volgende personen zijn op de aangegeven tijden beschikbaar voor vragen over programmeerproblemen. Respecteer hun privé-leven en bel alstublieft alleen op de aangegeven tijden.

Waarover	Wie	Wanneer	Tijd	Telefoon	Email
Liberty Basic	Gordon Rahman	ma. t/m zo.	19-23	(023) 5334881	grahman@planet.nl
MSX-Basic	Erwin Nicolai	vr. t/m zo.	18-22	(0516) 541680	basic@lordthanatos.com
PowerBasic CC	Fred Luchsinger	ma. t/m vr.	19-21		f.luchsinger@kader.hcc.nl
QBasic, QuickBasic	Jan v.d. Linden				j.vd.linden@kader.hcc.nl
Visual Basic voor Windows	Jeroen v. Hezik	ma. t/m zo.	19-21	(0346) 214131	j.a.van.hezik@kader.hcc.nl
Visual Basic .NET	Marco Kurvers	do. t/m zo.	19-22	(0342) 424452	m.a.kurvers@hccnet.nl
Basic algemeen, zoals VBA	Marco Kurvers	do. t/m zo.	19-22	(0342) 424452	m.a.kurvers@hccnet.nl
Office					
Web Design, met XHTML en CSS