DS Game Maker

The next step in DS homebrew

For version 5.0

Written by James Garner

Welcome to DS Game Maker

Before we start to work with DS Game Maker and learn about developing homebrew, please read the following questions and relevant answers.

What is DS Game Maker?

DS Game Maker is a piece of software for Microsoft Windows that lets you design and make games for the Nintendo DS.

Can I Play the Games I Make?

Yes. Even with the Free Edition you can play your games on your computer and on your Nintendo DS console. This is covered in detail in "Playing the Game for Real".

Is this Game Maker?

No. DS Game Maker implements the same events and actions system as Game Maker (which is a very good system), but it is not related to Game Maker in any way, and past this similarity they are completely separate products, each with their own features, bugs, and ways of doing things and letting you make games. Please see the section "Relation to Game Maker".

Do I have to Pay to use This?

The short answer is No. DS Game Maker comes in 2 editions, Free and Pro. You can use the Free Edition to make games forever – there's no trial time limit. However, there are a few limitations. These are in place so you can try out DS Game Maker and see if it suits your needs, and make some small games. To make bigger games, and gain full usage of all features, you can buy the Pro Edition. This is a lifetime license which means that you will never have to buy the software again, even if we make major upgrades to the program. Despite the fact that we encourage you to buy the Pro Edition for our commercial interests, we encourage you to first use the Free Edition for a while. You can make good games without having to buy it, so you don't *have* to pay to use it.

Please see the section "Upgrading to Pro".

Who created DS Game Maker?

DS Game Maker is written by James Nathan Garner who is a 16-year old student living in West Yorkshire, England. He is also the creator of the website and administrator of the Forum.

James released the first version of DS Game Maker on 26th May, 2008. Since then the program has taken on many stages of appearance and functionality. James, Robert (creator of WiiGS) and Ondrej Blazek are working on a product which will allow you to make games for any platform.

All About Making Games

Here is a brief introduction to making games for the DS.

Playing computer games is great fun.

2D games still provide a wealth of entertainment for family and friends – think of Pacman, Space Invaders and Street Fighter.

Wouldn't it be great to make your own games like these?

Completing a computer game and having your friends and family play it gives immense satisfaction. Unfortunately, it is not easy to develop for the Nintendo DS - you need to know how to program and have knowledge of various tools and compilers. It is certainly not an easy task or a task for novices. Does this mean developing for the DS is impossible? …

No! Here comes DS Game Maker. With DS Game Maker, you are presented with an attractive and easy to use interface for designing and creating games.

Skill Levels

"We made the software simple to use but did not limit its use!" – V4 Manual

Like most things, there is an easy and a hard way to do things. Sometimes the hard way reaps long term benefits but who has the time or the effort for it? This is with reference to making games. DS Game Maker fits into the 'easy' category, but it does not possess limits that hinder you. You can take DS Game Maker as far as you want, making simple point and click games, or writing scripts and complex algorithms that use WiFi and C code. DS Game Maker has been designed so that it is easy to use at first but can be extended to the skill of the user.

DS Game Maker uses a similar events-driven system like that of Game Maker (for PC). If you have ever used Game Maker for PC, you will find similarities in the way objects interact and the way things happen.

Types of Games & Included Examples

Certain design programs present a range of templates to you, for example "driving game", "fighting game" or "shooter game". Whilst these are quick and exciting for beginners, they are incredibly limiting in terms of creating anything else or customizing the games at all. We have avoided this in DS Game Maker.

When you are experienced you can make your games completely from scratch. To help you with using DS Game Maker, a range of example games are included, so you can learn how to do a variety of things with DS Game Maker by example. You can look at and copy the parts that are useful for your project. Best of all, you can learn from these examples so that you can write great games yourself.

GETTING HELP!

At some point or another, you will probably need assistance with using DS Game Maker.

There are a variety of places at which you can get assistance and it comes in a variety of forms. The first place is naturally this help file. You can use the 'find' function in your PDF reader to look for certain keywords or the bookmarks tab which will list the contents of the document.

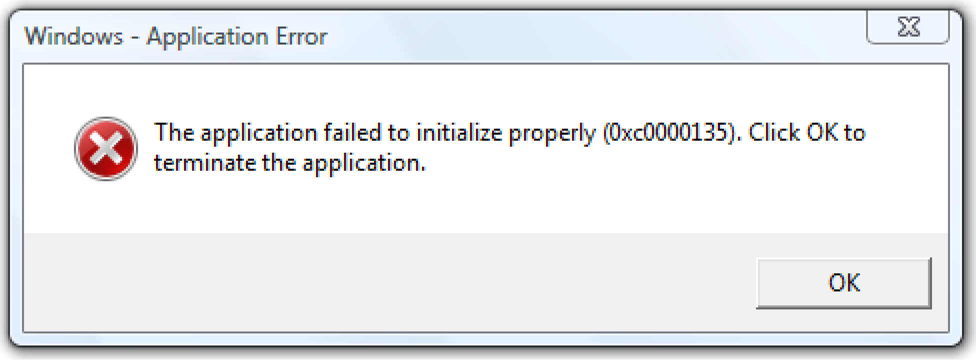
You may have question that is not covered in this manual – we cannot cover every situation or scenario.

For these questions you have 2 options. You can either contact us using the contact form on the website, or use the online Forum. With the contact form you will get a reply within 48 hours, but with the Forum the reply is normally much quicker. This is because there are users just like you reading about making DS games and helping other users (just like you!). So you should first consult the help file, and then post on the Forum: http://dsgamemaker.com/forum

Any issues relating to sales, homebrew kits or the Pro Edition should be raised with the contact form as they are private matters which relate to data that only we have.

INSTALLATION

Now that you are introduced to the software, you should install DS Game Maker (if you have not done so already). The setup process for the software is fast and simple. Before beginning, ensure that the "Microsoft .NET Framework" is installed. Windows 7 users have this by default but Windows XP or Vista users may not have it, or have an outdated version. This is so if you receive the following error:


Windows - Application Error

The application failed to initialize properly (0xc0000135). Click OK to terminate the application.

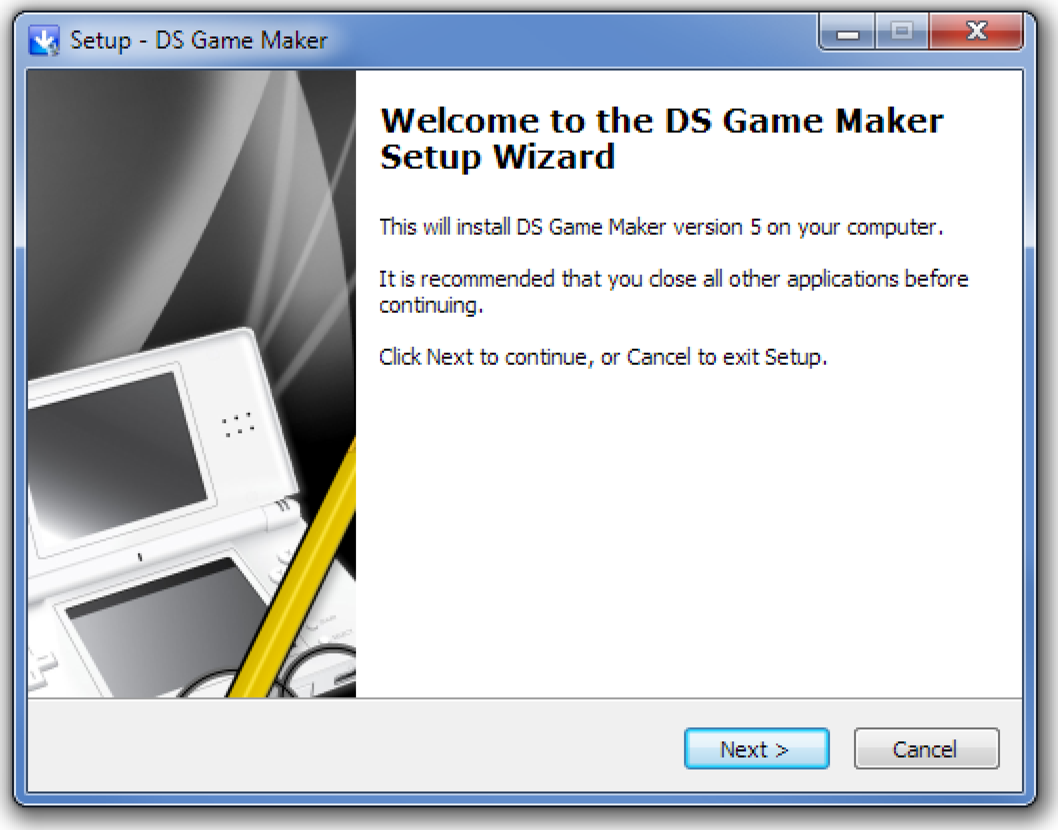
To correct this, install the framework from the Microsoft website:

http://bit.ly/MicrosoftDotNet

Next, run the DS Game Maker installer. If you have already downloaded it, it will be called "Install505.exe" and will probably be on your Desktop. If not, you can visit the website to download it:

http://dsgamemaker.com

You are presented with the following:



Welcome to the DS Game Maker Setup Wizard

This will install DS Game Maker version 5 on your computer.

It is recommended that you close all other applications before continuing.

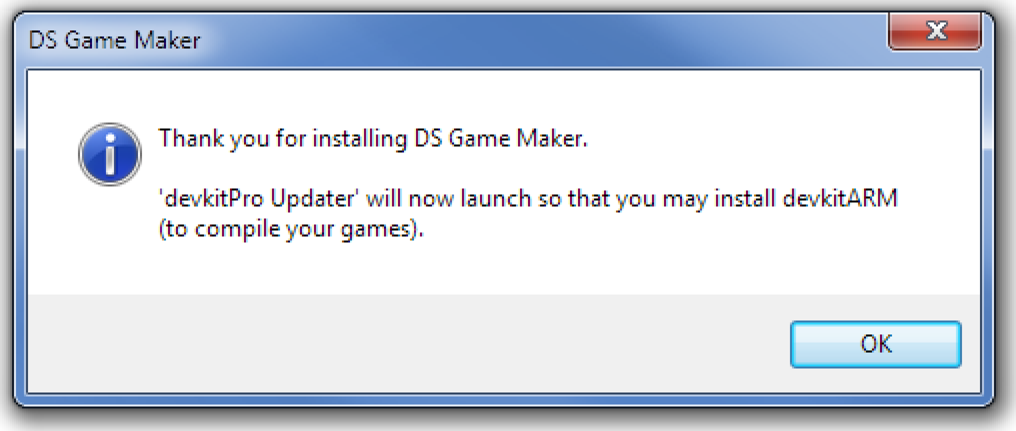
Click Next to continue, or Cancel to exit Setup.

You are given a variety of choices such as whether you would like to create a start menu shortcut. These do not affect the performance of DS Game Maker so you can choose what you like. If possible, leave the installation directory at default. Although the software will work and run from any directory, it is easier for us to help you if it is not changed, and something goes wrong. Once the wizard is complete you are given the chance to launch the software – do so.

When you start DS Game Maker and you have never used the software before or made any homebrew, you will need to install 'devkitARM'; an external toolchain which is used to compile your game.

DevkitARM – The Development Toolchain

DS Game Maker relies on this external toolchain to compile your game. It is not created by us, but it is easy to install. If the directory C:\devkitPro does not exist, it is presumed that you do not have the toolchain installed.


DS Game Maker

Thank you for installing DS Game Maker.

'devkitPro Updater' will now launch so that you may install devkitARM (to compile your games).

You will see the above dialog when you start the program. Click 'OK' and wait for the Updater to launch.

Specific instructions for using this are available online: http://help.dsgamemaker.com/updater.php

RELATIONSHIP TO GAME MAKER

Naturally people compare "DS Game Maker" to "Game Maker" by Mark Overmars (YoYo Games). In reality they are different products by completely different people.

DS Game Maker does not use any source code or components from Game Maker for PC.

DS Game Maker is entirely original.

The very obvious similarity is the events and actions concept that exists between both programs. This is an excellent concept, but it is also such a simple concept that could not possibly be copyrighted (or claimed to be Game Maker-exclusive, for example). It has already been duplicated for the Nintendo Wii with Wii Game Studio with relative success. The programs share interface layouts in areas, such as for the Object window. This is genuinely coincidental: the layout used in Game Maker is the best way to display the components; the best way to do it has been chosen in DS Game Maker, and these ways are similar.

Very simple concepts like having a menu bar and toolbar at the top of the main window like Game Maker, and having "floating windows" is certainly not copying anything. Such features have been part of Windows programs since before Game Maker was even created.

Finally, I DS Game Maker V5 admittedly has moved much close to the program by YoYo Games but it is still definitely a unique implementation that builds upon the good points of Game Maker and the great events/actions system, but does not harm their business or portray a direct copy.

GETTING TO GRIPS

Before diving into the possibilities of DS Game Maker it is important that you understand the key concepts behind games created using the software. Games created in DS Game Maker take place in one or more *rooms* (they are flat, not 3D, but may contain 3D looking graphics). In these rooms you place *objects* - for example walls, moving balls, monsters or the main character, and you also set *backgrounds* for the room (e.g. sky, grass, menu) which indicate the environment or add decoration. Every object has a *sprite*, which is basically its image. You can change the sprite of an object at any point. Some objects will move around and react to player input, for example from the stylus or DS' buttons. Some sprites trigger logic when they collide with other sprites, for example when the player meets a monster, he might die.

Once you have added objects it is time to define the *rooms* in which they are placed. Rooms can be used as levels in your game or as menu screens, or for anything in fact. Rooms have 2 *backgrounds* (one for each screen). You can place instances of your objects in the room. You can place multiple instances of the same object in the room (you only need to add 1 wall object to the project but you can use it in many places, such as in making a maze).

The most important element in your game is logic – making things happen. There is a simple *events* and *actions* driven approach. Firstly there are events, like "Collision with Ball" or "D-Pad Right button Press". When this condition happens (e.g. such button is pressed, or such collision occurs), the event is fired. When the event is fired, its list of actions is executed. Actions are things that happen, and they are arranged in a list. So each Object has a list of events, and every event for the object has a list of actions. Consider the following:

Player: Object_Player

Event: Collision with Object_Monster

The actions you may want to add would be:

```
Set Lives to Lives - 1
Go to Room Room_YouLose
```

Each room also has "View" settings. With this feature, you are able to add objects positioned off the screen and scroll them into view as the player navigates the room. See it as an overhead camera that is already programmed for you!

Even if the above is still quite vague, you will soon understand by following the below tutorial to create a basic and functional game.

TUTORIAL GAME: TOUCHING BALLS

You shouldn't misinterpret the above title, though it is accurate.

We are going to make a game with 5 rooms, so it is possible in the Free Edition. There will be some Balls arranged in these rooms, and the player must touch them all to advance to the next level. When a ball is touched it is deleted so it is clear which balls must still be touched. This is rather easy, so to add difficulty there will also be a countdown timer so that the player will have to work fast.

Firstly, we should plan what *Resources* are going to be needed. The Resources are going to include Sprites, Objects, Rooms, Backgrounds and Sounds. There is no need for Scripts or Paths in this game.

Before we can work out what Sprites to add, we must know what Objects will be using them. We are actually only going to need to have 2 objects:

- Ball Object
  - Event: Touch with Stylus (actions below):
    - Delete the touched Ball
- Controller Object
  - Event: Step (meaning always executed, every frame) (actions below):
    - Control the timer
    - If time is up (equal to 0), go back to the first level
    - If the instance count of Balls is 0, go to the next level
    - If the level is 5 and the instances of Balls is 0, go back to the start also.

So for the Ball object we are going to need a Ball sprite. The controller object will be invisible and therefore does not require a sprite.

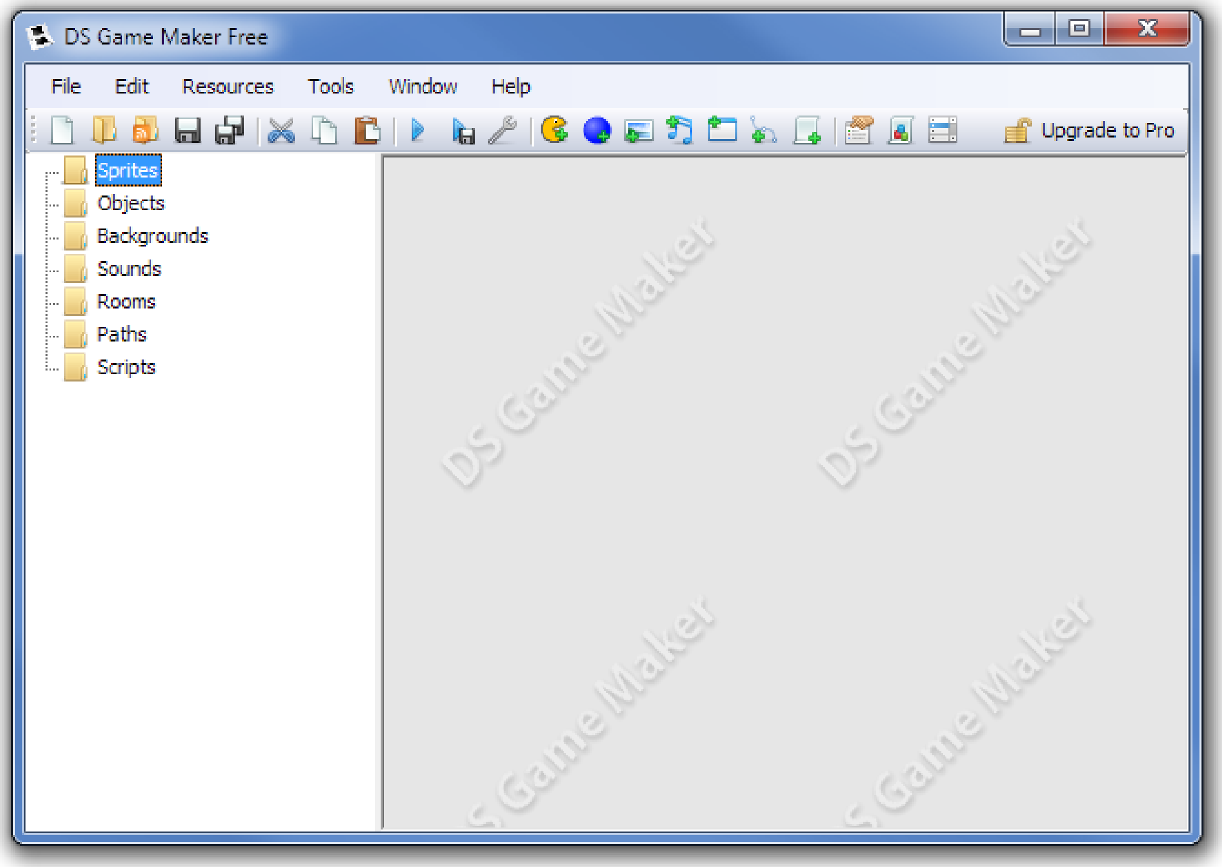
We are also going to need one background to place behind the balls. The top screen is not in use, so we do not need a background for it, but you may add one if you wish.

We are going to need 5 rooms. In each room we will firstly plot the controller object, and then instances of the Balls that the player must touch to complete the level and move to the next room.

When the above is done, we will add some Sound. We will add a sound effect for when you click a ball.

We have successfully planned our game. It is now easy to make as we have worked out what resources we will need and also thought about what logic will be used. Good work.

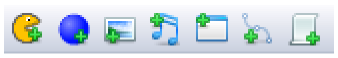
Making a Start & Adding Resources

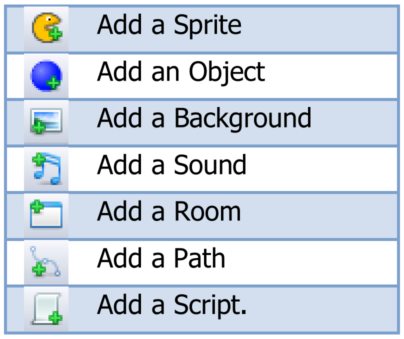


Welcome to DSGM! (DSGM is the abbreviation often used for Game Maker)

This is what DSGM looks like when you start it. We are now going to start making a game. The software is ready as soon as you start it.

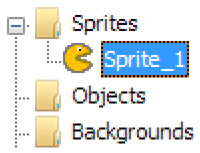
Now we will begin to make the game. We will first investigate how to add a resource like a Sprite or an Object. The best way to do it is by using the tool bar, though you may also use the menu bar (Resources > Add Sprite, etc.).



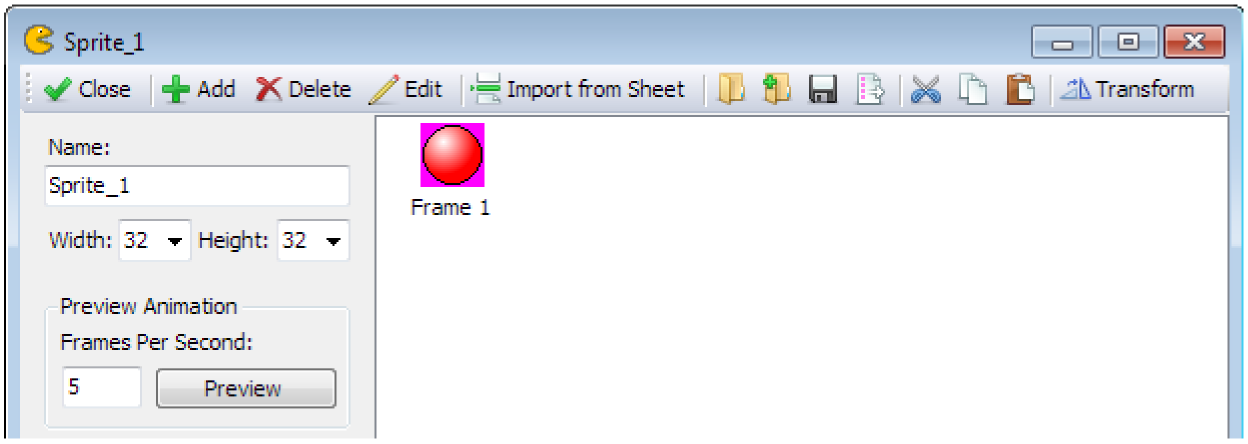


	Add a Sprite
	Add an Object
	Add a Background
	Add a Sound
	Add a Room
	Add a Path
	Add a Script.

Click the first icon shown above to add a Sprite. You will see 'Sprite_1' appear in the resources list:



Now we can begin to work with the sprite. To open it, double click 'Sprite_1'.



Name: Sprite_1
Width: 32 Height: 32
Frames Per Second: 5

We should change the name from Sprite_1 to a better informative name.

There are basic naming conventions that you must follow when naming any resource or variable. You cannot use any spaces, but you can use underscores (_) instead. The name cannot start with a number or be a number (one character). It also cannot be longer than 32 characters.

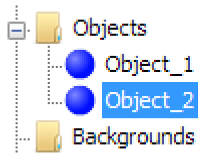
So in the text box where you see 'Sprite_1' put 'Ball_Sprite' instead. As it happens the default sprite in DSGM is a ball, so we do not need to change it.

Should you want to change it:

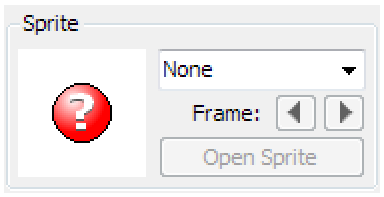
Click 'Frame_1' so it is selected. Now click 'Edit' on the tool bar (the pencil). The frame will open with MS Paint, or Paint.NET if you have it installed. Make any changes in the editor and use the editor's save feature. Now close the editor. Click 'OK' at the dialog presented by DSGM, and the frame shown in the sprite will appear updated.

Now that we are finished with the resource, click 'Close'. This will save the new name and close the dialog. You will see that 'Sprite_1' has changed to 'Ball_Sprite' in the resources list.

Now we will add 2 objects:

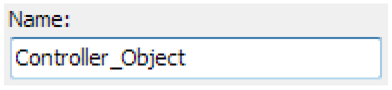


Double click 'Object_1' to open it. This will be our invisible controller object, so we do not need to assign it a Sprite in the project. Look at this:



Here you may choose the sprite of an object (for when the object is created in the room). The question mark reflects that 'None' is selected. In the game this icon is not shown, nothing is visible.

Change the name to Controller_Object (don't click 'Accept' yet):

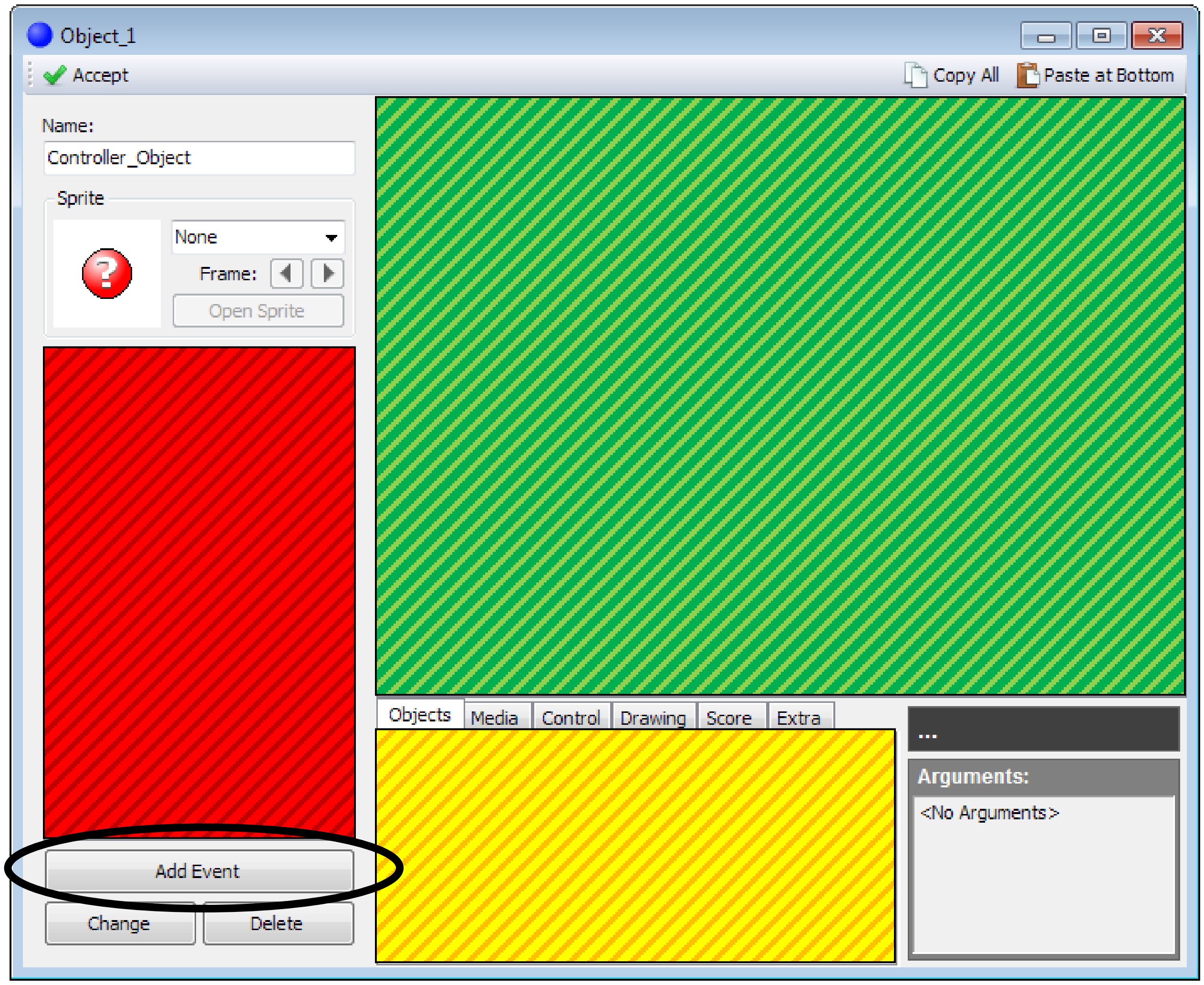


Name:

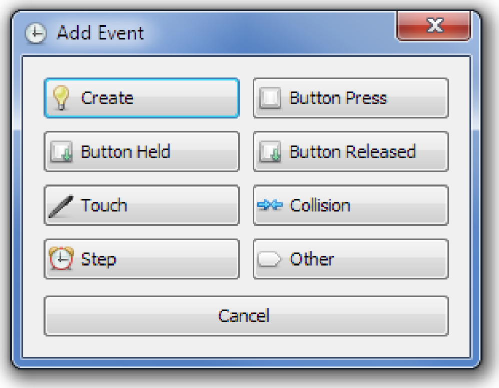
Controller_Object

Working with an Object

Now we will familiarize ourselves with events and actions. Take a look at an Object window:



- RED: The list of events for the object, such as Collision events, Button press events
- GREEN: The list of actions for the selected event (in the red list)
- YELLOW: The actions you can add to the list for the selected event. Drag them in.



Now we are going to add an event. To this click the 'Add Event' button (circled above). You will see the dialog to the right.

Here we choose the type of event.

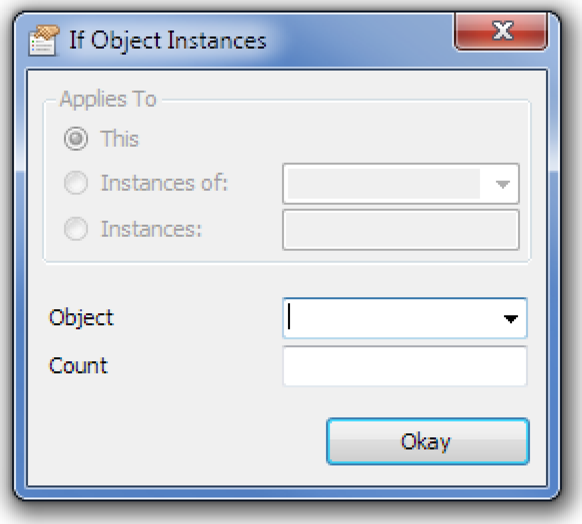
Events like 'Create' don't need a setting, because no customization can be performed on creation of an object. However, for 'Collision', naturally the setting is which object the collision should be detected against (which collision will fire the event). For 'Button Press' the setting is which DS button e.g.// L, R, A, B, Start and so on.

We are going to add a 'Step' event, so click 'Step' (with the little alarm clock). Now this event appears in the events list. It should be already selected for you – showing a blue background behind it – if not, click it to select it. Actions can now be added to the actions list for the 'Step' event.

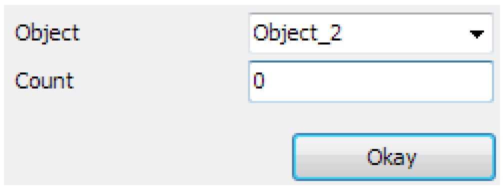
The Step event is an event which happens every frame the game is running, most useful on the controller object. We use this event if we want to constantly monitor a variable or perform timing operations. For such timing, we need an event which is constantly fired, and not when something happens.

We can use the action 'If Object Instances' to check for the number of Balls left present on the screen. You can find it in the 'Objects' tab (the first one). Hover over the icons until the information panel to the right of the action chooser shows 'If Object Instances'. The icon is an octagon with 2 blue balls.

Hold the mouse over the icon and drag it into the empty actions list. You see:



Every action has *arguments* or *parameters.* These provide the extra information that the action needs to function. In this case it is the Object of which to count the instances, and the Count against which to compare. We need to check if there are no instances are left, and therefore the object will be Object_2 (soon to be renamed to Object_Ball), and the Count will be 0. Execution therefore continues when there are 0 instances of the balls left! **Fill in the values like below, then hit 'Okay'.**



Object	Object_2
Count	0

You have added your first action – congratulations.

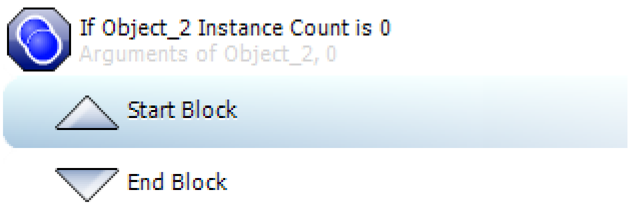
Actions execute one after another from top to bottom. So for example, if you set a variable at the top of the list, you will use the changed value in all operations below this.

BLOCKS OF LOGIC – IMPORTANT!

Creating blocks of logic is crucial to making any kind of game. A block can be described as a bit of code that is indented or separate from the rest, and may not run in the logical order of top to bottom. A simple example is the 'If Object Instances' action. This action opens a block. It opens a block which *only* executes when the condition is met. So when the instance count is 0, the block executes. You must *always* start and end a block, or you will receive an error. You create a block whenever actions after one action begin to indent/move right.

Go to the 'Control' tab and add 2 actions – 'Start Block' and 'End Block'.

You will brilliantly see the formed block now:


If Object_2 Instance Count is 0
Arguments of Object_2, 0
Start Block
End Block

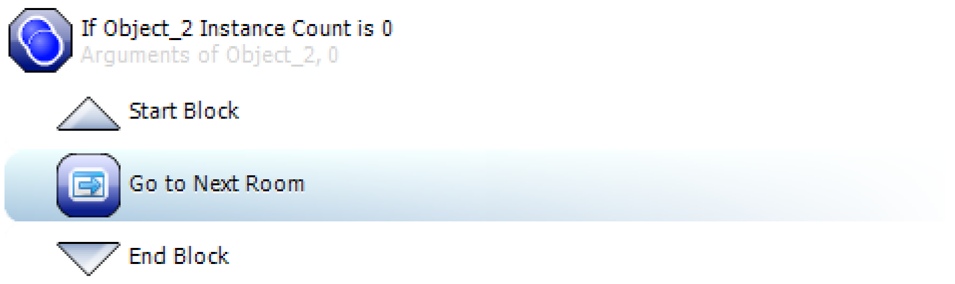
Actions between 'Start Block' and 'End Block' ***only*** **execute when the If condition is satisfied, i.e. there are 0 instances present.**

Now add the action 'Go to next Room' also from the 'Control' tab. No dialog appears this time because the action needs no additional information (no arguments). However, the action is in the wrong position. It needs to be between the 'Start Block' and 'End Block' actions.

An action is always added to the bottom of the list, at least as of V5.0.

REARRANGING LOGIC

To rearrange the actions so that the logic works, hold the mouse over 'Go to Next Room' and drag it above the 'End Block' action in the list. The indentation will update once you release the mouse. You should now see:


If Object_2 Instance Count is 0
Arguments of Object_2, 0
Start Block
Go to Next Room
End Block

If you have followed successfully so far you are well on your way to totally understanding how the DSGM system works.

Now we can add the time limit.

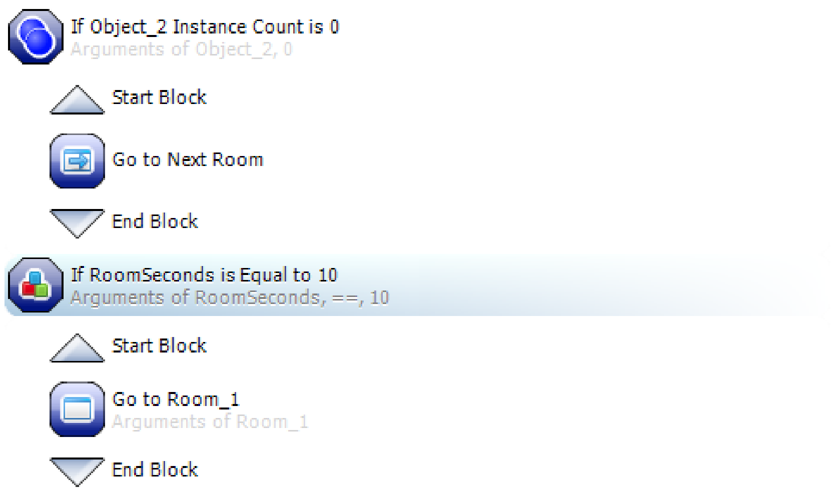
VARIABLES!

'Variables' are covered in the section titled 'Variables' in this document. For this game, you must just know that a 'Variable' is a name that represents a value that can change, like in algebra.

Here are some variables you can use:

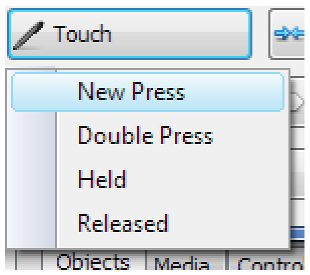
- RoomSeconds – how many seconds have passed since the Room started
- RoomFrames – how many frames have passed since the Room started (60 frames per second)
- Seconds – how many seconds have passed since the Game started
- Frames – how many frames have passed since the Game started

The variable we need to use is 'RoomSeconds'. Let us give the player 10 seconds to touch all of the Objects. So we need to see if 'RoomSeconds' is equal to 10, and if it is, reset to level 1 because they failed. Under the 'Control' tab use the 'If Variable' action. For 'Variable' argument put into the box RoomSeconds. For the last field put 10, hit 'Okay'. Add 'Start Block' and 'End Block' actions. Add the action 'Go to Room'. 'Room_1' is the first level so hit 'Okay' at the dialog. Move the action up within the block, resulting in the following:



```
If Object_2 Instance Count is 0
Arguments of Object_2, 0
    Start Block
    Go to Next Room
    End Block
If RoomSeconds is Equal to 10
Arguments of RoomSeconds, ==, 10
    Start Block
    Go to Room_1
    Arguments of Room_1
    End Block
```

We are finished with the Controller object, so hit the 'Accept' button. The dialog will close.

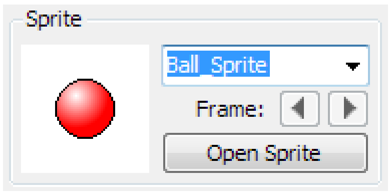


Now open 'Object_2'. Give it the name 'Ball_Object' in the name box. Click 'Add Event', and select 'Touch'. In the drop down menu select 'New Press' (shown left).

Applying An Action

Add the action 'Delete Object' in the 'Objects' Tab. Don't click 'Okay' yet though. 'Delete Object' has no actions but a dialog has still appeared. This is because the action can be *applied* to different objects or *instances* of objects.

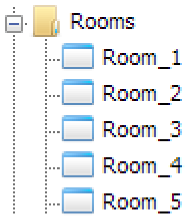
- **THIS** is the current instance – the specific ball that is touched. So, if 'Delete Object' is applied to itself, the touched object will be removed, but only that touched instance
- **INSTANCES OF** – this lets you apply an action to all instances of an object. So for example you could delete all 'Monster' objects if you were making a game involving monsters. In our game there is no need to apply an action to all instances.
- **INSTANCES** – (this is covered in detail later) Whenever you plot an object in a room, starting from 0, it is given an *Instance ID*. So you can apply an action to specific instance numbers which could hold any object in the room.



Leave 'This' selected and hit 'Okay'. Now we must apply the 'Ball_Sprite' sprite we created a while ago to this object. Next to where you see the red ball with a question mark there is a drop down box. Select 'Ball_Sprite' in that box, and the preview will refresh (shown left).

Now we are also done with the ball so click 'Accept' to close the dialog.

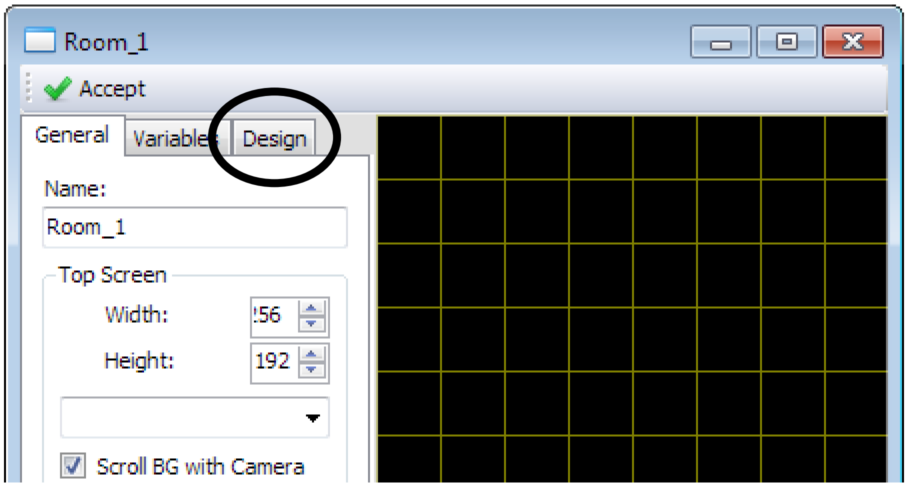
Creating a Scene with Rooms



Now that our actions are prepared, we can start plotting the objects in our levels.

Add another 4 rooms so that the resources list is populated like shown left.

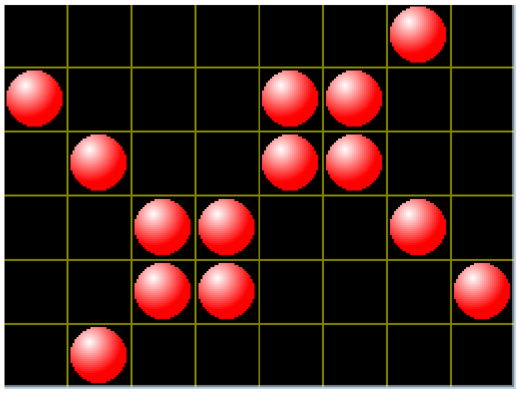
Double click 'Room_1' to open the room editor.



The room editor is divided into 2 sections. The left most section is a tab display that lets you manipulate the various settings for the room. The right section shows the 2 Nintendo DS screens, on which we plot backgrounds and objects.

Go to the 'Design' tab.

There is a drop down box that lists the Objects in your Project, underneath the label 'Object to Plot:'. Select 'Object_Ball' from the list, and click at various places on the bottom screen to plot instances of the ball:



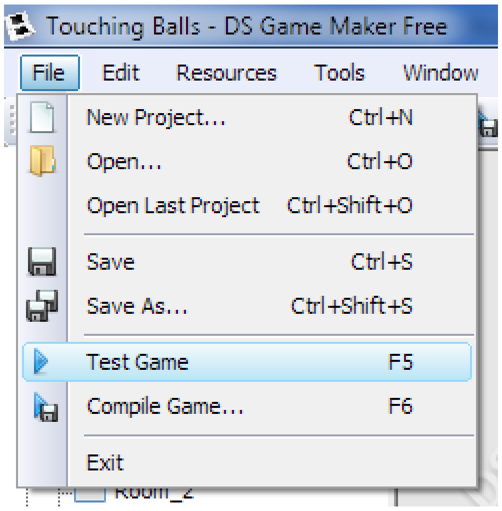
Balls to touch.

We also need to plot the controller object, so select 'Controller_Object' in the drop down box then plot it somewhere. **Only plot it once.**

The same process needs to be repeated for Room_2, Room_3, Room_4 and Room_5. In these rooms plot 1 controller object and then the balls that must be touched.

Testing our Game

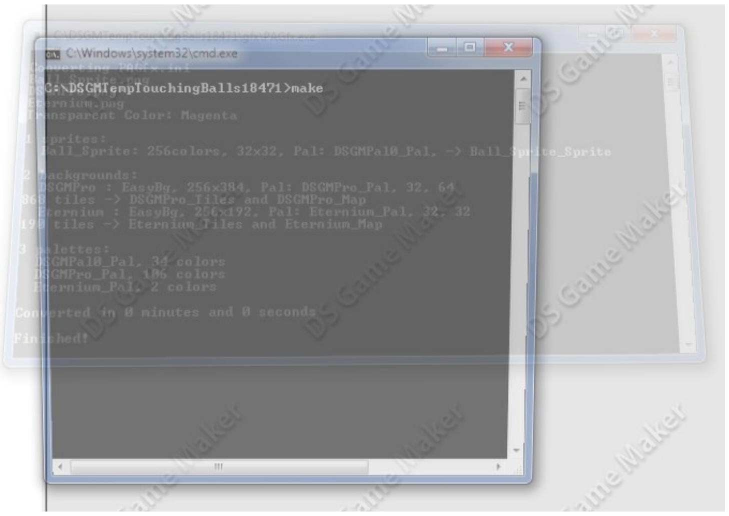
Providing you followed the instructions, our game is ready to play:



F5 is a useful shortcut that you will use quite often.

In compiling the game, there are 2 stages:

1. **PAGfx** – Converting graphics
2. (Actual) **Compiling** – Turning the generated C code into a binary (NDS file).

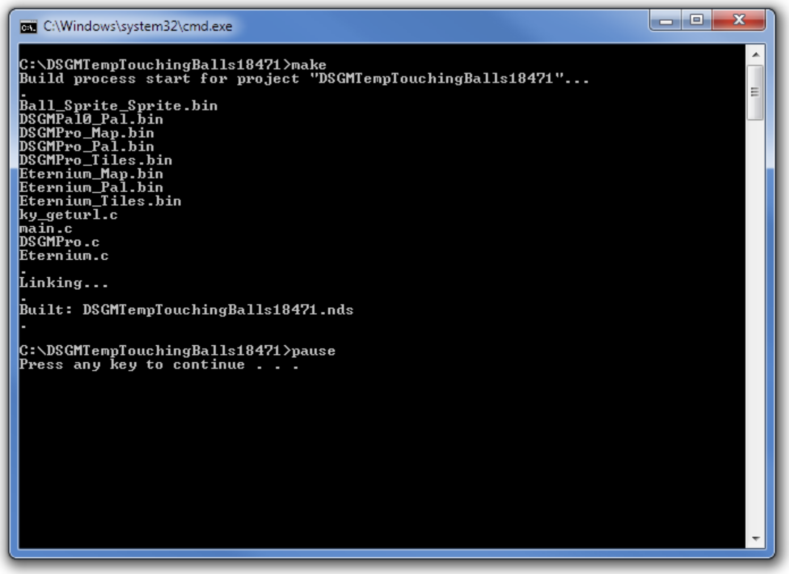


```
C:\Windows\system32\cmd.exe
C:\DSGMTempTouchingBalls18471>make
```

```
Converting PAGfx.ini
Ball_Sprite.png
DSGMPro.png
Eternium.png
Transparent Color: Magenta

1 sprites:
 Ball_Sprite: 256colors, 32x32, Pal: DSGMPal0_Pal, -> Ball_Sprite_Sprite

2 backgrounds:
 DSGMPro : EasyBg, 256x384, Pal: DSGMPro_Pal, 32, 64
868 tiles -> DSGMPro_Tiles and DSGMPro_Map
 Eternium : EasyBg, 256x192, Pal: Eternium_Pal, 32, 32
190 tiles -> Eternium_Tiles and Eternium_Map

3 palettes:
 DSGMPal0_Pal, 34 colors
 DSGMPro_Pal, 106 colors
 Eternium_Pal, 2 colors

Converted in 0 minutes and 0 seconds

Finished!
```

PAGfx is very fast. If there's no error, the dialog will disappear almost instantly. It can be seen disappearing in the above screenshot, whilst the compiler opens.

The second dialog can help you solve errors.



```
C:\DSGMTempTouchingBalls18471>make
Build process start for project "DSGMTempTouchingBalls18471"...
.
Ball_Sprite_Sprite.bin
DSGMPal0_Pal.bin
DSGMPro_Map.bin
DSGMPro_Pal.bin
DSGMPro_Tiles.bin
Eternium_Map.bin
Eternium_Pal.bin
Eternium_Tiles.bin
ky_geturl.c
main.c
DSGMPro.c
Eternium.c
.
Linking...
.
Built: DSGMTempTouchingBalls18471.nds
.

C:\DSGMTempTouchingBalls18471>pause
Press any key to continue . . .
```

You can see that there are no errors. The instruction says 'Press any key to continue' so follow it.

Now the emulator will appear.

EMULATION WITH NO$GBA

NO$GBA is a Microsoft Windows program that lets you play Nintendo DS homebrew on your computer. It is excellent for testing because you do not have to transfer the file onto a memory card every time you make a change. When you write larger games you may have to test the game 20 times to solve a bug in your logic.

The game works well.

If you would like to test any saving logic, you cannot use an emulator. Saving and Loading variables only works on a real Nintendo DS.

ADDING A BACKGROUND

Go to Resources > Add Background on the menu bar. Double click 'Background_1' in the resources list. In the new window, click 'Load from File'. You are presented with a dialog where you may select an image file to import.

Navigate to the folder C:\Program Files\DS Game Maker\Resources\Backgrounds

Select the file 'Wooden.png'. Double click it, and the background preview in the window will change. Using the name box at the bottom of the window you can rename the background to 'WoodenBG' if you wish, then click 'Accept'.

Open up Room_1 again, and make sure you are on the 'General' tab. There are 2 boxes, one for each screen. In each box there is a drop down box with a list of the backgrounds in the project. There is also a blank item, which is currently selected, which represents no set background and is therefore black during production and testing. For the drop down box in the bottom screen container, select your added

Background. For display we will also turn off the grid lines. Go to the 'Design' tab and uncheck the box 'Show Grid'. You must apply the background in a similar way to the other rooms.

Test the game again to see the change made.

FINISHING TOUCHES

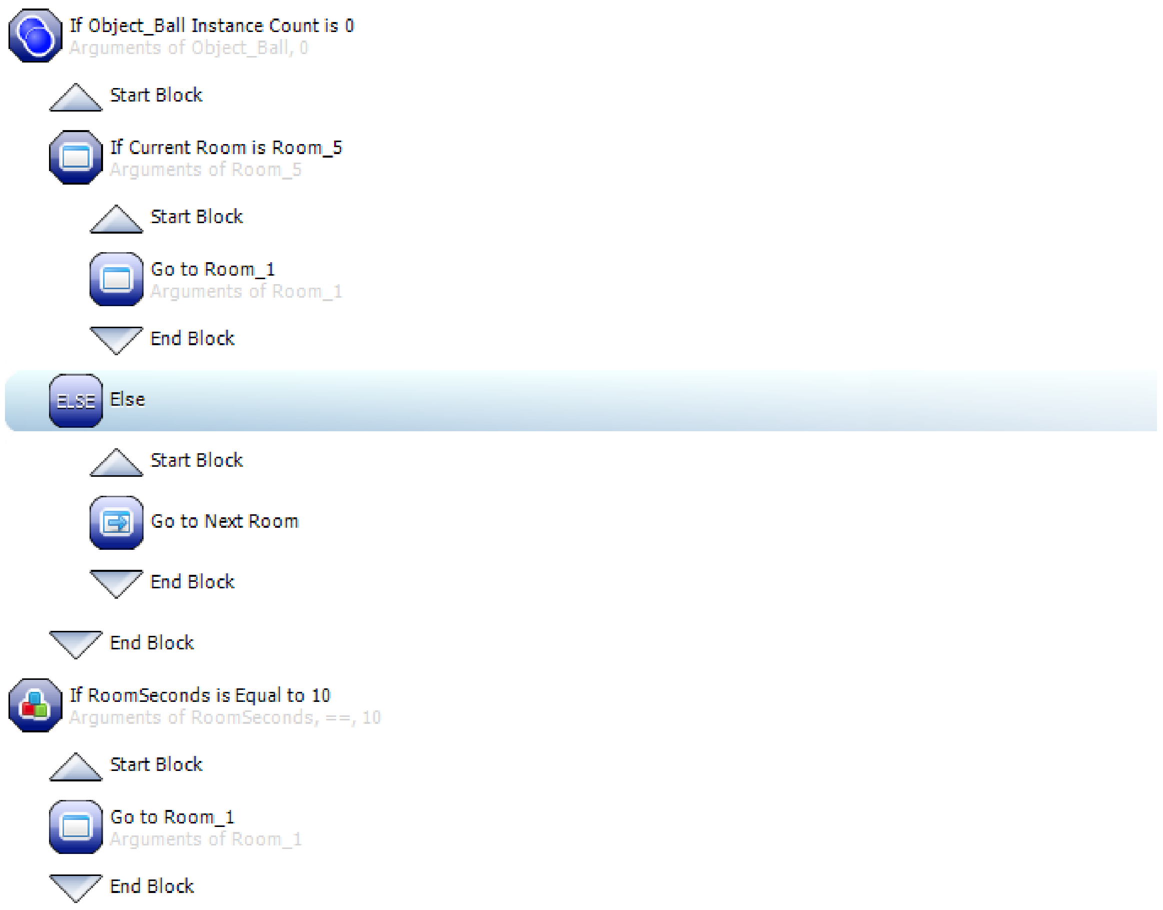
The game is good, but there are a few things left that we can do.

"If the level is 5 and the instances of Balls are 0, go back to the start also."

This part of our brief is not complete yet, but it is very easy. There is an action called 'If Current Room' under the 'Control' category. We are going to need this as well as the 'Else' action. We must do some more exciting logic now. We must get the DS to work out what to do:

- If instances of Ball = 0
  - If the room is 'Room_5', go back to Room_1.
  - Otherwise, go to the Next Room.

You should see that there is a block inside a block here. This is perfectly acceptable. Here is the modified actions list:



```
If Object_Ball Instance Count is 0
Arguments of Object_Ball, 0
    Start Block
    If Current Room is Room_5
    Arguments of Room_5
        Start Block
        Go to Room_1
        Arguments of Room_1
        End Block
    Else
        Start Block
        Go to Next Room
        End Block
    End Block
If RoomSeconds is Equal to 10
Arguments of RoomSeconds, ==, 10
    Start Block
    Go to Room_1
    Arguments of Room_1
    End Block
```

Adding Sound

Sound is very easy to add to a game. All you need is a WAV (Wave) file, and one action to play it. Go to Resources > Add Sound then double click 'Sound_1' in the resources list. The predefined sound effect will do (don't trouble yourself with finding another sound file). You can rename the sound if you wish.

Open up 'Object_Ball' and ensure the first event is selected. Go to the 'Media' tab of the actions panel this time. Drag 'Play Sound' into the list. Select 'Sound_1', Hit 'OK' and then 'Accept' on the dialog. That is all that is needed to play a sound file.

Playing the Game for Real

We will finally discuss how to play the game on a physical Nintendo DS console.

To do it, you will need a small adapter cartridge, called a **"flashcart"** (also commonly known as an R4, or slot-1 solution). This is an adapter from a memory card (that you might find in your cell phone) to the DS game slot. **It enables the Nintendo DS to play games that are stored on the memory card.**

You *may* already have one. They are commonly used for pirating commercial games (questionably the intended purpose) and are therefore popular. Don't pirate games, folks.

We will be using them just for homebrew, which is completely legal and also great fun.

I have a Flashcart ...

Most flashcarts are very compatible with DS Game Maker games. If your flashcart does not work with DSGM (after you have followed the below troubleshooting steps), it is a clone or has a problem.

Some flashcarts do not use a memory card, but instead store the files internally, and such files can be changed by connecting the flashcart to a PC with a USB cable. If this applies to you, do so.

Most flashcarts though have a removable memory card. You should connect this to your PC like you have done in the past. You can use a memory card slot on your computer or use a USB card reader.

I Don't have a Flashcart ...

You now have 3 options. The first is that you don't have to buy a flashcart. If you want to just test your games on your computer, you can do this entirely without buying this extra hardware. You could give the NDS file to your friends by e-mail and they could play it on their DS, should they own a flashcart.

A flashcart is only needed if you want to play the games on your physical DS console.

The second option is to purchase a flashcart and memory card yourself. This is best done online but can be done in town or whatnot. In retail you will often find the price is much greater that on the internet.

Buying a Flashcart

A reliable place to shop is http://shoptemp.com/?gbatemp. You will need to do your own research into which flashcart is best based on price and functionality. Note that the $6 'R4 DS Revolution card' shown does not support the DSi or later console releases.

In particular you will want to explore the category 'DS Flashcarts': http://shoptemp.com/categories/DS-Flashcarts (you need one of these)

Then you should visit the category 'Flash Memory': http://shoptemp.com/categories/Flash-Memory

Once you receive the products in the mail you should use the internet to find the firmware files for your flashcart and put them onto the memory card. Then you are able to also load any games you have made onto the flash cart to play them.

Buy a DSHomberew Kit!

You can find a few guides online for setting the above up, and I would like to confirm that you are not forced at all to use a Homebrew Kit, any other flashcart should work just the same.

However, some people do not want to have to order a memory card and flashcart separately and then load the firmware on etc. Some people also want to buy a flashcart which is guaranteed to run their games without compatibility issues.

Some people also would like to know that their flashcart has been tested before it was sent to them, and includes a variety of sample DSGM homebrew games for them to play.

These things are only generally available with a Homebrew Kit from us.

http://kit.dsgamemaker.com

With a DS Homebrew Kit, you receive a flashcart and 2GB memory card. This is all you need to play the games you make. The memory card already includes the firmware for you and also many games from the DSGM Website, so you can use the flashcart with your DS as soon as you receive it.

The Homebrew Kit is guaranteed to run DS Game Maker-produced homebrew games.

There is also a 50 day return period:

If the product does not work, we will help you try to fix it. If it is determined that there is a physical problem with the Kit then we will send you a replacement Kit or broken part for free until there is no more issue. Instead of receiving this help you can also request a Refund, in which case you will have to post the Kit back to us (at your cost), and we will then refund all of your money.

Should the product work but you decide that you do not want it, you can send the product back to us within 60 days (at your cost) for a complete refund.

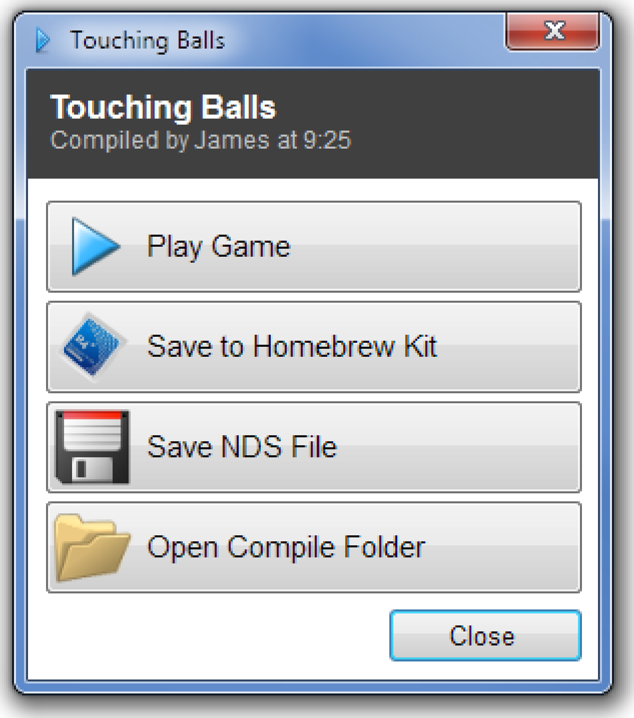
If you bought a Serial code and then receive a refund, the Serial code will be removed and you will not be able to use it any more. We request that you send the product back to us so that you cannot lie about *it not working to receive a refund but continue to use it, and so that we can investigate any technical faults.*

So if you want to play your games on your DS, buying a DS Homebrew Kit is the recommended option. If you have not thought about purchasing the Pro Edition yet, you can order a DS Homebrew Kit without Pro included. Otherwise there is a discount for people who already have Pro.

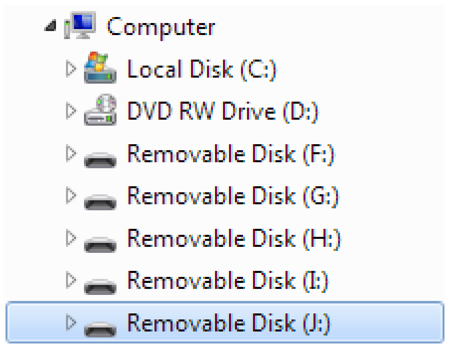
LOADING A GAME ONTO ANY FLASHCART

Make sure that the memory card is connected to your computer, or that the flashcart is connected to your computer by its USB cable. A new removable device should appear in 'My Computer'.

Now we are going to use the *second* compile button. The first, that we used before (F5), simply let us test the game, but it did not give us the option to save it somewhere. So we use the other button, which looks almost the same but has a small floppy disk icon on it. You can also perform this command by pressing F6:



Click the button 'Save NDS File'. You are presented with a save file dialog. On the left (if you use Windows Vista/7) or in the top drop down box, select the drive that represents your memory card:



Now just click 'Save'. You can now (safely) remove your memory card/flashcart.

Using a Homebrew Kit

You can follow the above for using a Homebrew Kit and it will also work, but there is actually an easier method. In the dialog presented by DS Game Maker we chose the option 'Save NDS File'. Above this was a button entitled 'Save to Homebrew Kit'. If you have a Homebrew Kit, you can use this button with faster results.

Connect the Kingston memory card to your PC with the included USB card reader. Now click the 'Save to Homebrew Kit' button. If the game already exists on the homebrew kit (the same name NDS file), it will be overwritten. When it is done you will receive a confirmation message. If you click 'Yes', the card will be safely removed so you can unplug it. If you click 'No', it will remain mounted so that you can put other games or files onto it.

Now you can play the game on your DS just like the included sample games.

As a reminder, here is the address where you may buy a homebrew kit: http://kit.dsgamemaker.com

PRO EDITION

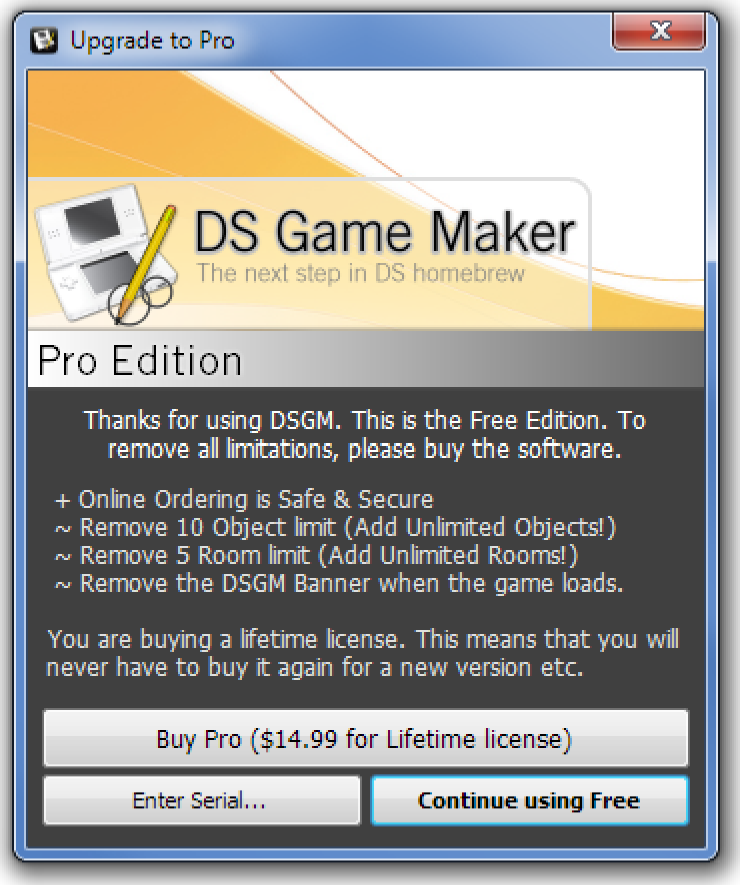
Once you are more experienced with DS Game Maker, you will want to make bigger games – games with more objects and rooms, and games that do not show the splash screen.

To remove the splash screen and add unlimited objects and rooms, you should buy the Pro Edition:

- Remove the 10 Object limit
- Remove the 5 Room limit
- Remove the DSGM Splash screen when the game loads.

The Pro Edition costs $14.99 for a single user, life time license. This means that no matter how much the program advances or how many new versions are released, you will not need to buy Pro ever again.

To upgrade to Pro you can use the dialog shown when the program starts:



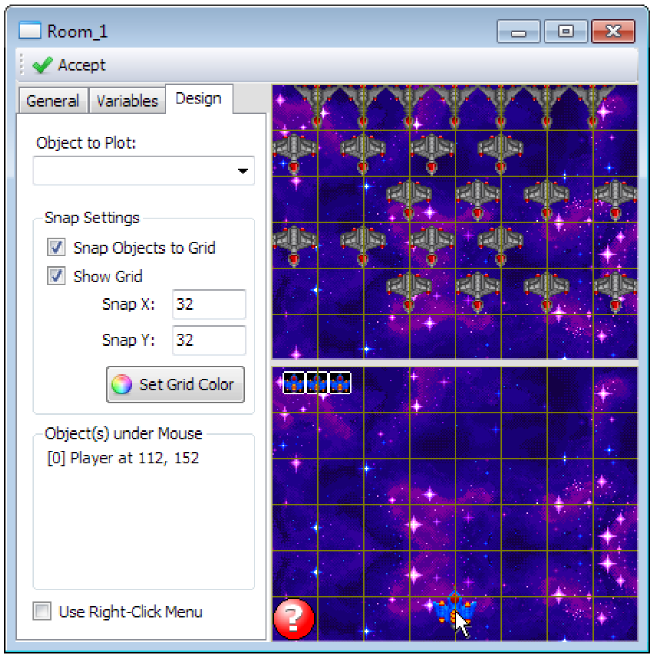
Thanks for using DSGM. This is the Free Edition. To remove all limitations, please buy the software.

+ Online Ordering is Safe & Secure
~ Remove 10 Object limit (Add Unlimited Objects!)
~ Remove 5 Room limit (Add Unlimited Rooms!)
~ Remove the DSGM Banner when the game loads.

You are buying a lifetime license. This means that you will never have to buy it again for a new version etc.

You can use the following address also: http://order.dsgamemaker.com

INSTANCING



Object to Plot:

Snap Settings
[✓] Snap Objects to Grid
[✓] Show Grid
Snap X: 32
Snap Y: 32

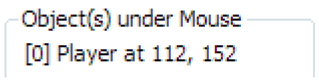
Object(s) under Mouse
[0] Player at 112, 152

[ ] Use Right-Click Menu

In following the above tutorial, we already know that in the project there are objects like a Ball or a Monster. Now we must understand that whenever we 'plot' or 'place' one of these objects in a room, we are creating an 'instance' of it. Each plotted instance also has an *ID* or *number* which is its index in all of the created instances. This index begins at 0, therefore the first instance plotted has an ID or number of 0, the second 1, the third 2 and so on.

In the screenshot shown left, you can see that the mouse is hovering over the 'Player' object in the 'Asteroids X' example game. This was the firstly plotted object in the room, and has an instance ID therefore of 0. DSGM doesn't expect you to remember the order in which you plotted objects though.

On the 'Design' tab there is the following display:



Object(s) under Mouse
[0] Player at 112, 152

The number in the square brackets represents the ID of the instance in all created instances. The other 2 numbers are the co-ordinates of the object (in pixels): the first is X (how right-wards or horizontally positioned an object is) and Y (how down-wards of vertically positioned an object is).

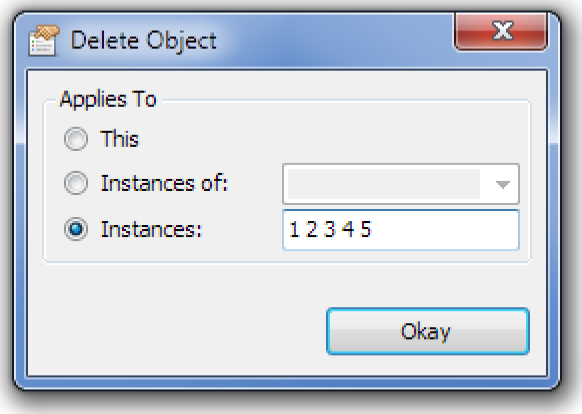
You can plot 128 instances in a room from 0 to 127 IDs.

WHERE THIS IS USEFUL

In 'Asteroids X' it isn't important to know the instance IDs. This is because an action is only ever applied to 'this' (itself) or all instances of an object (e.g. move all instances of an enemy ship). When an action is applied to all instances of an object, DSGM computes which instance IDs are of that object and performs the action on those instances.

However, there are occasions where it is useful to apply an action to a specific or some specific instance IDs in the room, and it doesn't matter what object is any of these instance slots.

For example let us say that we are building an inventory. The first 2 instances (0 and 1) are for interface controls ('back to menu', 'use item'). Instances 3, 4, 5 and 6 hold objects which are in the inventory. We do not know when designing the game what objects will be in these instance slots but we can presume they will always be full. Therefore without knowing what objects are in the slots we can apply an action to them. You may also want to manipulate the position of instance 0 (in a different game), whatever object is actually in this ID, or delete a random instance of an object. To do this, find an action that supports application, we will use 'Delete Object':



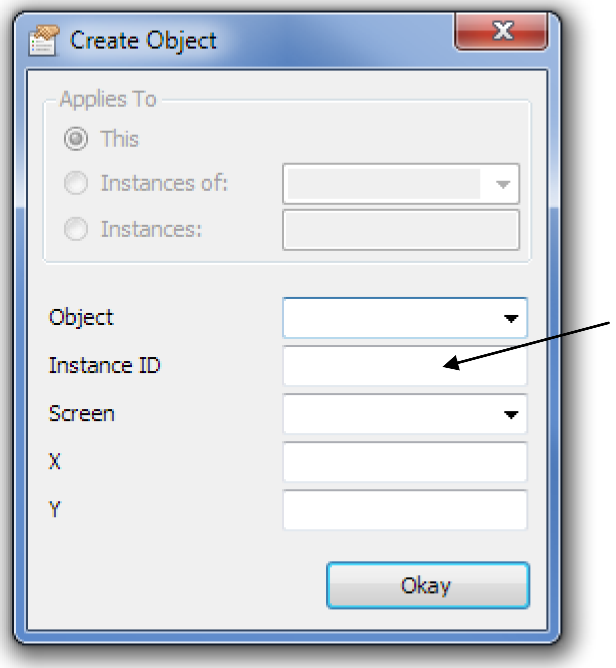
Instances: 1 2 3 4 5

Input the list of instance IDs separated by spaces.

Now whatever objects are in the 1, 2, 3, 4 or 5 slots, the action will be applied to them.

Creating Instances of Objects

Plotting objects in rooms is fine but often you are required to create objects whilst the game is running, like bullets. These have to be created when an action like pressing A occurs, and so cannot be plotted already in the game. Therefore, we must use the action 'Create Object':



For 'Instance ID' you can fill in any free or occupied ID. However, it is not always possible to know what instance ID will be free unless you build a very complicated logging system.

This is not necessary. To use the next available instance ID, put in the box:

NextInstance()

This function will compute the next free instance slot for the object.

You can use this to rapidly create objects without keeping track of or knowing their instance IDs. For example in 'Asteroids X', see the actions for creating the bullets.

Designing Rooms

We have briefly designed some rooms in the 'Touching Balls' tutorial but we have not explored all of the possibilities available with the room editor.

Snapping to Grid

If you want to arrange objects in a linear or organized fashion, this will be useful. When you click to plot an object, the software will round the clicked co-ordinate up or down according to the 'Snap X' and 'Snap Y' provided values. If the 'Snap X' is 16 then the object will 'snap' to the grid of 16 when plotted.

These settings are available under the 'Design' tab of the room editor. When designing a game having a visible grid can be obstructive but you still want the snapping ability. To achieve this, check 'Snap Objects to Grid' but uncheck 'Show Grid'. There is also a button for changing the grid color.

Right-Click Menu

To remove a plotted object, right click it. You can also enable a Right-click Menu which will give you 2 additional options:

- Open the object to edit its events, actions and settings
- Set the precise co-ordinates of the object.

To enable this, use the check box titled 'Use Right-Click Menu'.

DSGM "Views" System

In DSGM 5 there is a Views system which is just the same as in V4. In the game you can control a virtual camera that can move around the screen. The objects and backgrounds scroll in the opposite direction. The Views system is without complicated controls and can be used very easily.

To experiment with the Views system, change the width and height of the room from the DS screen size of 256x192 pixels. For example you may wish to make it 768x192 pixels for a side-scroller game.

VARIABLES

'Variables' have already been mentioned a few times, and they are extremely important. There are 2 types of data, constant data, and variable data. Constant data are things like "James", 5 or true. They are fixed values that cannot be altered. Variable data is much more important and useful. With a variable you take a value like 5 and give it a 'name' so that you can change the value, and use its value at many places. This is not easy to explain but 'score' is the best example. The term 'score' is used to represent a number that can be changed and used in the game. There are 3 predefined variables for you to use:

- score
- lives
- health

Variable names are case sensitive! Be careful with this, it's easy to make a mistake. For example 'Lives' will generate an error because 'Lives' does not exist, but 'lives' will work just fine.

VARIABLE SCOPE

The 'scope' of a variable defines how visible it is to the rest of the game. In DSGM there are 2 possible scopes: Eventual, and Global.

A Global variable can be accessed and changed by all objects and their events. An Eventual variable can be accessed and changed only by the event in which it is declared using an action.

VARIABLE TYPES

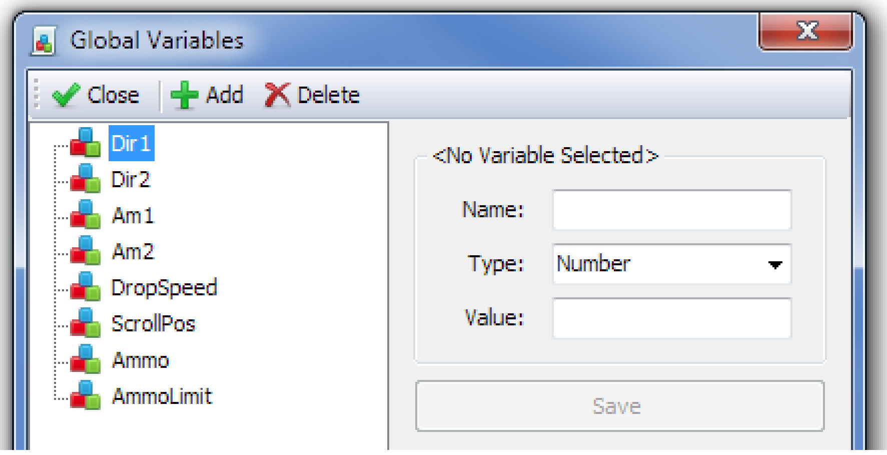
A Variable also has a certain 'type' which dictates what kind of data it can hold. Really there are only 2 types of variables you will need to use and they are 'Number' and 'TrueFalse'. A number holds (..) a numeric value like 500 or 3. A 'TrueFalse' (Boolean) holds either true or false, and therefore consumes much less memory.

You *could* declare 'Invincible' as a number and use 1 for true and 0 for false but then the DS still stores the possibility of using many other numbers, so when you can use a 'TrueFalse', you should.

GLOBAL VARIABLES

To begin we will look at using Global Variables. As previously mentioned, Global variables can be accessed and changed anywhere in the game. There are already some Global variables declared for you – score, health and lives. You probably would not think of these but they can be modified or accessed at any time by any object so they are Global.

To manage Global Variables there is an inbuilt tool. Go to Tools > Global Variables:



The Global Variable Manager

The variables are listed on the left. To edit a variable, double click it in the list.

If you have read carefully you will have already seen this:

There are basic naming conventions that you must follow when naming any resource or variable. You cannot use any spaces, but you can use underscores (_) instead. The name cannot start with a number or be a number (one character). It also cannot be longer than 32 characters.

So when you change the name of a variable you must follow the above rules or you will get errors when you compile your game. If you choose 'TrueFalse' for the variable type you will have to put 'true' or 'false' in the 'Value' field. If you use a 'Number' you can put in any value.

ADD A VARIABLE

To add a variable, click the 'Add' button on the tool bar. Double click 'Variable_1' which has appeared in the list. Put Coins in the name field instead. Make sure the 'Value' is 0 and then click 'Save'. You can now close the window.

EVENTUAL VARIABLES

An Eventual Variable has only the scope of the event in which it is declared. Therefore if you declare 'Coins' within the Creation event of the player, no other events will be able to use this variable so it is quite pointless. However there will be situations where eventual variables are really useful because you want to have a variable of the same name for some logic for many events.

To declare an Eventual variable, use the action 'Declare Variable'. This action has 3 arguments. The first is the name and we know the rules for the name (no spaces, etc.). The default value is also quite self-explanatory but the 'C Type' is explained below.

'Number' and 'TrueFalse' are just human-like DSGM-created terms for programmatical terms. A 'Number' is actually an 's16' and a 'TrueFalse' is actually a 'bool'.

So for C Type if you want a numeric variable, put in the box s16. If you want a TrueFalse variable, put bool.

Only actions *below* this declaration action can use the variable, so you are best off putting it at the top of the list.

SETTING VARIABLES

The following applies to all variables; they are all set in the same way whether Global or Eventual.

Setting a Variable is very simple. The action to use is 'Set Variable' and there are 2 arguments – the variable name and the value you would like to give it.

To increase a variable, set the value to the name + whatever, for example:

```
Set Coins to Coins + 1
```

Divisibility

The action 'If Variable Divisible' will continue execution if a certain variable is divisible by a number or other variable. Example:

```
If Coins is Divisible by 2
  Start Block
  (Player has an even number of coins)
  End Block
```

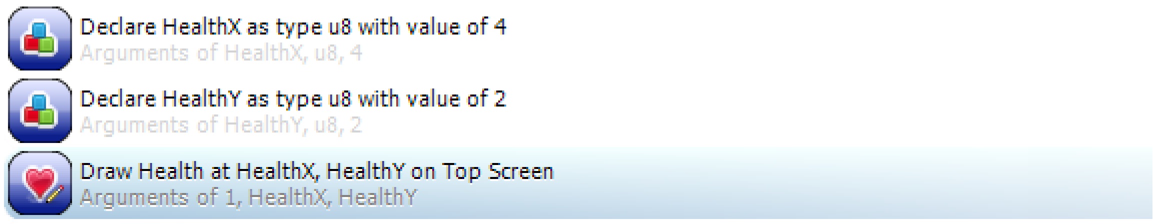
Variable Actions

Here are the other variable actions you should try:

- If Variable (Control)
- Draw Variable (Drawing), also Draw Lives, Draw Health, Draw Score

Variables in Other Places

You can use Variables anywhere (providing the scope is correct), and not just in variable-related actions. You can have a variable for the position of some text or the position at which to create an object. Instead of putting in a fixed value like '3' into an argument box, you can put in the name of the variable, like so:



Declare HealthX as type u8 with value of 4
Arguments of HealthX, u8, 4

Declare HealthY as type u8 with value of 2
Arguments of HealthY, u8, 2

Draw Health at HealthX, HealthY on Top Screen
Arguments of 1, HealthX, HealthY

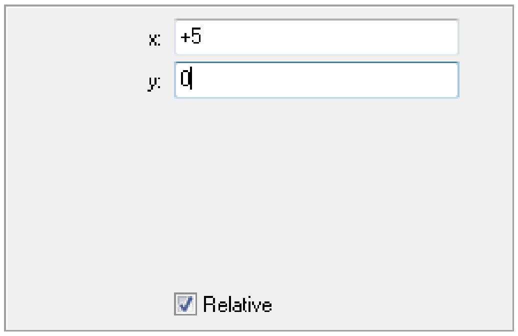
Relativity

Relativity is important in designing games. Sometimes you do not want to use a fixed value for the position of something, and in such occasions you want to increase or decrease the current value.

And so we make a 'relative' change which means that it is a change upon the existing value and not a completely different value.

In Game Maker…

In Game Maker there is a 'Relative' check box, but this is actually quite limiting. For example adjusting many arguments relatively is quite cumbersome. Also one argument cannot be adjusted relatively to another. See this:


x: +5

y: 0

[✓] Relative

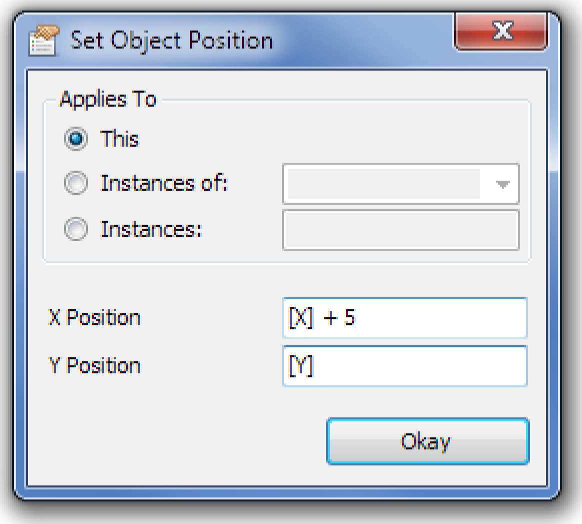
What is happening here is that the X and Y are being adjusted based upon themselves. So whatever change is supplied in the textbox is done *on top of* the existing co-ordinate. E.g. X is set +5 on top of X so X = X + 5, increasing X by 5 and moving it right. Y is adjusted by 0 so there is no change.

In **DS** Game Maker…

In DS Game Maker there is no relative check box. Instead whenever you want to perform a relative operating on a co-ordinate or other object property, you can use one of the following replacement indicators:

- [X] – X Position (how right-wards)
- [Y] – Y Position (how down-wards)
- [Frame] – Set frame of the instance (of the Sprite)
- [Screen] – Screen (as Boolean/TrueFalse)
- [OX] – Original/Creation X Position
- [OY] – Original/Creation Y Position
- [Width] – Width of the instance e.g. 32 pixels
- [Height] – Height of the instance e.g. 64 pixels

Therefore to move the instance right 5 pixels, we do:


Set Object Position

Applies To

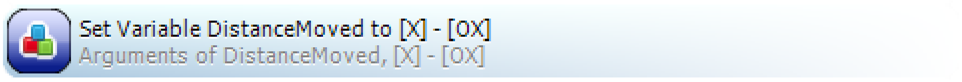
(● This) Instances of: Instances:

X Position: [X] + 5

Y Position: [Y]

Use [Y] so the Y position is retained when changing the position

A clever trick is to use [X] and [OX] to know how much an object has moved since it was created, like so:



Set Variable DistanceMoved to [X] - [OX]
Arguments of DistanceMoved, [X] - [OX]

E.g. X of 65 minus creation position of 30 shows movement of 35 pixels right.

SOUND

TYPES OF SOUND

As you will have seen when adding a sound in the above tutorial, there are 2 types of sound, or rather, 2 types of which a sound can possibly be. These are *Sound Effect* and *Background Sound*.

You have to choose which one you would like to use for each sound you add.

SOUND EFFECTS

Sound Effects are generally short and snappy. They are played when a specific event occurs like a collision or button press. They are not generally longer than 3 seconds, and they never loop.

BACKGROUND SOUNDS

A background sound is usually longer and played behind sound effects. It can also loop. A background sound can take around a second to load into memory and start playing, so they are no use for sound effects.

A background sound is also *streamed*. This is like internet radio and it means that whilst the game is running the next bit of the music is loaded as it plays. The DS has to do this because it does not have enough temporary memory to store the whole music file at one time. This means that you have to put the action to play a background sound in the step event of a controller object, or at least of an object that has a step event that executes every frame.

WORKING WITH THE SOUND PLAYER

DS Game Maker has a built-in implementation of Windows Media Player so that you can play the sound without running the game and so forth. You do not have to do anything else; when you open the sound by double clicking it in the resources list, the media player is ready.

The far button to the left is the play button, and this will play the sound. Note that all functionality here is the same for both background sounds and sound effects.

There is also a volume slider, which you must set every time you open a new sound window (we cannot solve this).

Background Sounds, Bitrates and Compression

Warning: Nerdy stuff that doesn't really matter, but it's good to know.

The DS has a tiny and slow processor. It also has very little temporary memory (RAM). On top of this the DS is also performing hundreds of calculations to make the rest of your game work, such as events & actions, graphics and scripts.

Hence it is not possible to play high quality music whilst the game is running. The DS can play music at a maximum of 96kbps (96 kilobytes of sound data per second), or at a recommended 64kbps. In DSGM version 4 this was a large concern as you had to reduce the bitrate of any sound you wanted to play before adding it into the software.

The above now is not necessary. You can add any high quality sound as either a sound effect or background sound. You can take music directly from an iTunes folder (providing it is an MP3 file) or from sound banks in other pieces of software like Game Maker.

When you compile your game, sound effects, stored as WAV files, are converted to RAW files for playing on the DS. This is done so that the DS does not have to work out any additional information about the sounds, RAW contains only the sound data, but you don't need to worry about this.

Background Music files are transmogrified into a lower bitrate of 64kbps. This doesn't make much difference to you, but it is the reason why the music does not sound as clear when you play your game. With reference to this, I believe the size of the DS speakers is so small that the difference is hard to tell, and regardless it's impossible to make the DS better.

Issues with Sound Files

In many scenarios, certain programs cannot handle certain files generated with certain other files and so on. DSGM is generally compatible with all varieties of something it supports (for example it loads PNG, GIF and BMP images), but you may experience difficulties when working with sound.

This is because there are many different ways to *encode* some sound into a file. Quality, compression and playback are things that are considered when making a sound file.

Unfortunately, it isn't possible for DSGM to support any sound file you find on the internet, and this is more-so relevant with sound effects. DSGM supports generic WAV files without issues, but if the WAV file is created in a weird or old way DSGM will simply not play nice with it. This issue cannot be avoided, if it happens, you have 2 options:

- Find another sound file
- Use a program like Audacity or Switch to convert the sound to another format and back.

There can also be issues with background sounds (MP3 files) though this is less common. Generally there are issues with very large sound files (8mb+), very high quality sound files (320+ kbps) and sound files with DRM protection. Again you can use an audio editor like Audacity to fix these things.

Playing Sounds

Sound Effects

Sound Effects are very easy to play in DSGM. First of all you must have added a sound and then selected 'sound effect' as the type. Once you have done this, you should select an appropriate event for an object, and go to the 'Media' tab of the actions pool.

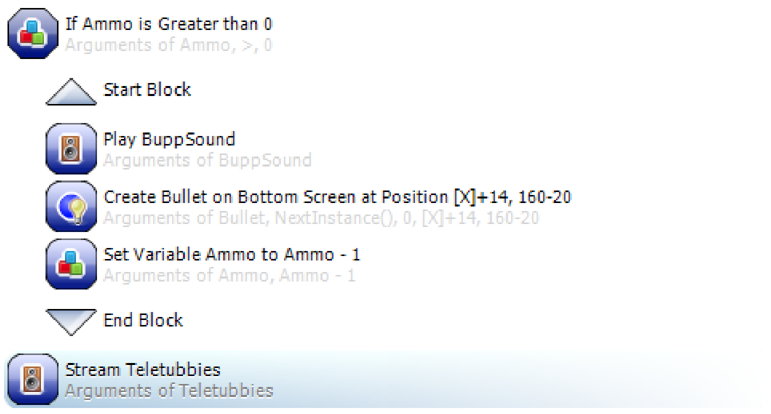
There are 2 sound actions and we need the first, titled merely 'Play Sound'. There is only one argument - the sound you would like to play. So use the drop down box to select a sound from the resources list.

Now when that event fires the sound will play right away. Easy!

Background Sounds

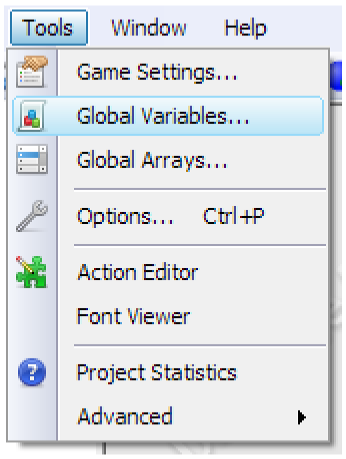
As mentioned above, a background sound is not played, it is streamed. This means that an action to stream the sound must execute all of the time. Thus, it is compulsory that you place the 'Stream Background Sound' action in the step event of an action in an object that is plotted in a room.

You can still run other logic in the step event, but the 'Stream Background Sound' must not be within a control block. For example the following is fine:



If Ammo is Greater than 0
Arguments of Ammo, >, 0
Start Block
Play BuppSound
Arguments of BuppSound
Create Bullet on Bottom Screen at Position [X]+14, 160-20
Arguments of Bullet, NextInstance(), 0, [X]+14, 160-20
Set Variable Ammo to Ammo - 1
Arguments of Ammo, Ammo - 1
End Block
Stream Teletubbies
Arguments of Teletubbies

Playing Sound "Through" another Event

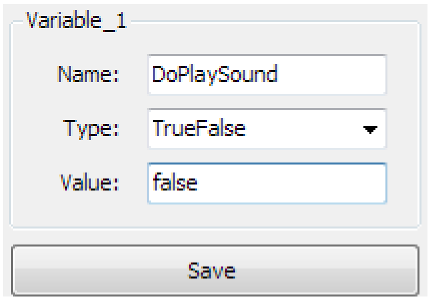


So, you ask, if the action for playing sound must be in the step event, how do I play the sound when an object is touched or a button is pressed? With a little creative thinking it is not difficult.

Here is the answer: To do it we will create a global variable which indicates whether the sound should be streamed or not. By default the value will be false, but when the object is touched, it will be set to true. The step event with the sound action will only execute the action if this variable is true.

To create this global variable we will need to use the 'Global Variable Manager', as shown left.

Click the 'Add' button on the toolbar. Double click 'Variable_1' in the list to edit it. Firstly it does not need to be a number. A 'number', like a human number, can hold many values such as 1, 45, 14585. We do not need this flexibility, we only need to store a true or false value, so we use 'TrueFalse' in the 'Type' drop down box. We should also change the value in the 'Name' text box to 'DoPlaySound'. The Value should be 'false', as shown right. Click the 'Save' button, and then 'Close' on the tool bar, to close the dialog.

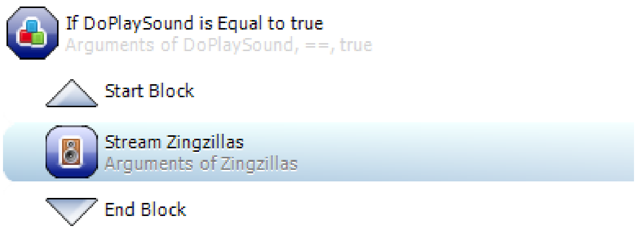

Variable_1

Name: DoPlaySound

Type: TrueFalse

Value: false

Now add the following logic to your controller object step event (The first action is 'If Variable'):


If DoPlaySound is Equal to true
Arguments of DoPlaySound, ==, true

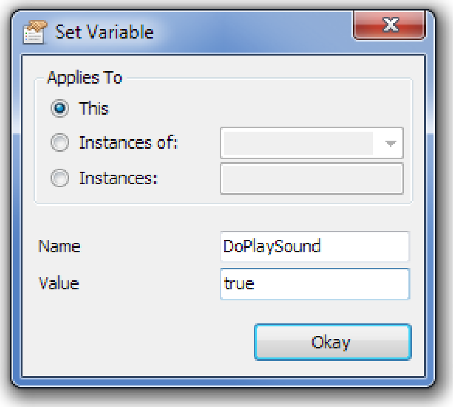
Start Block

Stream Zingzillas
Arguments of Zingzillas

End Block

For the touch event of an object, add the action 'Set Variable' like shown right.

That is all you need to do to make it so that when an object is touched background music starts to play.


Set Variable

Applies To

[✓ This]
Instances of:
Instances:

Name: DoPlaySound

Value: true

TROUBLESHOOTING SOUND

SOUND EFFECT ERRORS

If you use the action 'Play Sound' for a sound certainly existing in the project but you get a compile error particularly pertaining to this, it is likely that the WAV file used for the sound is not compatible with SOX (the audio convertor used by DSGM). You can check this is the issue:

1. Compile/Test your game and get the error
2. Go to Tools > Advanced > Open Compile Temp on the menu bar
3. In the Explorer window, open up the 'source' folder and then the file 'main.c'
4. (If Windows does not have a program to do this, use *Notepad*)
5. At the start of the file, there are lines beginning with "#include
6. If you see "// Error converting Sound_Name_Here with SOX! Sorry folks, use a proper WAV next time", this is the issue!

To fix this use a different WAV file or convert it using a program like Audacity.

Streaming Background Music

If your background music simply isn't streaming then check that the 'Stream Background Sound' action is in a step event of an object that is plotted in the room. For example, you can't put it in the 'Touch' event or a 'Button Press' event.

To be able to do this see the above section entitled 'Playing Sound "Through" Another Event'.

Taking Sound Further

There are more possibilities than merely playing sounds like described above, which are discussed here.

Stopping a Sound Effect

To stop a Sound Effect, use the action 'Stop Sound Effect'. This will stop all playing Sound Effects at the moment it is executed.

Stopping a Background Sound

Stopping a Background Sound is very easy, use the action 'Stop Background Sound'. There are no arguments, the action stops whatever sound is playing for you.

Pausing a Background Sound

Like above there is also an action to pause a Background Sound called 'Pause Background Sound'. This should be used instead of not streaming the sound with the action 'Stream Background Sound' which may produce unexpected results.

You can unpause a background sound which has been paused with the action 'Unpause Background Sound'.

Looping

Background Sounds can loop, meaning that when the sound ends it starts to play from the beginning again. This can be enabled and disabled with the action 'Set Looping'. Use the option 'True' to make the sound repeat indefinitely or 'False' to make the sound play only once.

Panning

The panning of a Background Sound can be adjusted when the game is playing. The action is 'Set Panning' and the 'Amount' argument takes a value between -64 and 64:

- -64 is full Left panning
- 0 is normal/balanced panning
- 64 (+64) is full Right panning.

THE INTERNALS OF DSGM

In working with DSGM and particularly for advanced users, knowing the internals leads to more efficient games and maximum extent of use of the systems used behind the scenes. Please read the above section 'Instancing' before this section.

INSTANCES

In C there is an array of all of the instances. Each element has properties which are things like the X, Y and Width of the object, which you can get and set.

You can use indexes from 0 to 127 in this array as there are 128 possible instances in a room.

To set the X position of an instance in C, you would do:

```
Instances[5].X = 64;
```

You can also do this using the DSGM actions. Use 'Set Variable'

```
Set Variable Instances[5].X to 64
(Arguments of Instances[5].X, 64)
```

Equally anywhere else in actions, you can read the positions and properties of instances. To make one object share the X position of another (horizontally move together), you can do this:

```
Set Object to Position Instances[5].X, [Y]
```

(To understand the concept of [Y], read the 'Relativity' section).

Otherwise the X of our object is being set to the X of instance 5, so they will move together horizontally.

PROPERTIES OF INSTANCES

Above we worked with .X and .Y properties of instances. Here are all of the properties you can get and set of instances whilst the game runs:

- Instances[5].X – X Position (how right-wards)
- Instances[5].Y – Y Position (how down-wards)
- Instances[5].Frame – Set Frame of the instance (of the Sprite)
- Instances[5].Screen – Screen (as Boolean/TrueFalse)
- Instances[5].OriginalX – Original/Creation X Position
- Instances[5].OriginalY – Original/Creation Y Position
- Instances[5].Width – Width of the instance e.g. 32 pixels
- Instances[5].Height – Height of the instance e.g. 64 pixels

Extended Instance Work

Here is the C code for an instance within the Instances[] array.

```
typedef struct {
  s16 OriginalX;
  s16 OriginalY;
  s16 X;
  s16 Y;
  bool Screen;
  bool InUse;
  u8 Width;
  u8 Height;
  u8 ObjectID;
  u16 Frame;
  bool FrameChanged;
  u8 SpriteID;
} Instance;
```

You will see that there are properties of this structure that aren't covered above:

Property	Explanation	Can Read	Can Write
InUse	Whether there is an object in this element	🟩	🟥
ObjectID	Which object is in this element	🟩	🟥
Frame	Which frame is set by the user (applies to the sprite)	🟩	🟩
FrameChanged	Whether the .Frame property has been changed. If true DSGM will update the sprite's frame in the VBL	🟥	🟥
SpriteID	Which sprite (in PAlib's instancing) is for this instance	🟩	🟥

What Object is in an Instance?

To work out whether a Ball/Monster/etc. object is in an instance, we can use the 'If' action and an internal DSGM function. The above property 'ObjectID' specifies what object is in the instance element, but it is a number. However, we can use a function to find this number of an object by name and then compare these 2 values.

The function is ObjectToID(char* ObjectName):

```
If Instances[5].ObjectID is Equal to ObjectToID("Ball")
  Start Block
  (Object is a Ball)
  End Block
```

Inbuilt Functions

There are a few inbuilt functions in DSGM. These are used mainly by actions in their code but you can use them should you wish.

IsBgCollision

Arguments: Screen (Boolean), X Position (Number), Y Position (Number), Layer (Number).

Returns whether a pixel on that background is occupied (not magenta). The layer argument is 2 for plotted backgrounds in a room. Example:

```
If IsBgCollision(false, Stylus.X, Stylus.Y, 2)
  Start Block
  (Position under Stylus is occupied)
  End Block
```

IsSpriteCollision

Arguments: Screen (Boolean), Sprite 1 (Number), Sprite 2 (Number)

Returns whether a midpoint based collision is occurring between the 2 supplied sprites in the PAlib sprite system (usually matching instance IDs also). Example:

```
If IsSpriteCollision(true, 0, 1)
Start Block
(Collision occurring between instances 0 and 1 on top screen)
End Block
```

Note that instances must be on the same screen for this function to work.

HowManyInstances

Arguments: Object ID (number) (find using ObjectToID(char* ObjectName))

Returns the number of instances of the object in the room. Useful for seeing if there are 0 instances of pickups left. Example:

```
If HowManyInstances(ObjectToID("Burger") == 0)
Start Block
  (No more burgers)
End Block
```

DSGM_ObjectsSync

No Arguments. This object syncs the positions of the on-screen PAlib sprites with the Instances[] array. It is performed every frame in the VBL. This is rarely useful, but you may need to update the sprite positions on screen for collision detection before the screen is refreshed. Also note that the display is only updated in the VBL anyway, so calling this function early will not change the display. Example:

```
PAlib DSGM_ObjectsSync()
```

ADVANCED

'FIXING' A FLASHCART

If your flashcart will not play your game (most common problem is 2 black screens), there are a few things you can try:

- Put the game in a sub-folder, e.g.// Games/Test.nds (fix for Acekard)
- Manually DLDI patch the game (most cards auto patch): http://chishm.drunkencoders.com/DLDI
- Run the game with Homebrew Menu: http://wiki.devkitpro.org/index.php/Homebrew_Menu

WORKING WITH YOUR GAME SOURCE CODE

When you test/compile your game, DSGM generates its entire source code and related files. You can access this at any time by going to Tools > Advanced > Open Compile Temp on the menu bar.

Until you test/compile a game within DSGM, the source code file will be blank at 0 bytes!

As soon as you test/compile your game again, the files in this folder will be overwritten with newly generated ones. So if you want to further edit the source code that DSGM has generated, please copy all of the files into a new directory. (Tip: it won't compile if the full path has a space in it, e.g. C:\Back Up).

MAKING CHANGES

If you make any changes to this source code and you want to recompile it outside of DSGM, run the file 'build.bat'. If you make any graphical changes, or a change you made isn't reflected, run 'clean.bat' which will erase any cached/precompiled material, and then run 'build.bat'.

Before you do the above it is recommended that you delete the already generated NDS file in the folder, e.g. TouchingBalls.nds. This is to avoid confusion between DSGM-generated NDS binaries and the latest binary you are generating with the batch file. After successfully compiling the game with build.bat you will see a new NDS file in that directory of name starting with 'DSGMTemp …'.

To access the bulk of the source code (the exciting bits) see source\main.c.

FREEING HARD DISK SPACE (EXPLANATION OF DSGM SESSIONS)

In DSGM V4 you could only have open one instance of a project at once. This limitation occurred because the program used just one directory for compiling the game (C:\DSGameMaker\Temp). This directory was used for every compilation which meant it was too risky for 2 instances to attempt to use this directory concurrently.

Under DSGM 5 however, a new directory for each project is created for compilation, and also for temporary storage of resources whilst you work with your game in the program. On closing, DSGM is supposed to clear the 2 directories it creates, for example:

- C:\DSGMTempTouchingBalls0561

- C:\Program Files\DS Game Maker\ProjectTemp\DSGMTempTouchingBalls0561

If the program crashes or quits unexpectedly, these directories will remain on your system until they are either manually deleted or cleared by the inbuilt tool.

So to clear these directories, go to Tools > Advanced > Clean Up on the menu bar.

Editing the Internal XDS

Users of DSGM 4 will know that 'XDS' is the extension of projects made with this version. It is a non-inclusive format which means that it does not include the graphics or any external resources like game include files or sounds. It is a plain text file that contains all data about the game, including:

- Events, actions and their arguments
- Game info like title, text 2 and text 3
- Sprite sizes, sound modes
- Room info and all object plots

In DSGM 5, this file is still used, it is stored within the inclusive .DSGM archive. If you want, you can tinker with this file. There isn't room to discuss the XDS format but it is quite easy to pick up should you take a look.

To edit it, close all open windows in DSGM and then go to Tools > Advanced > Edit Internal XDS

Warning: There is no protection against making mistakes. If you do not fully understand the XDS format, you should not edit it. DSGM will try to interpret the file as if the syntax is correct – by default behavior only DSGM edits the internal XDS regardless. If you make mistakes in editing the XDS it could lead to (major) losses of parts of your game. However, I do encourage you to investigate this capability.

Transferring Options

If you want to move your settings from one computer to another, you can do it by moving a file in the installation directory.

Go to C:\Program Files\DS Game Maker and find the file 'data.dat'.

You can *copy* (not move) this file to another computer to use the settings of the previous installation (from which you took the data.dat file). You may open this file with Notepad to examine it, but any changes should be made through the DSGM 'Options' dialog to prevent problems. Note that upon examination you will find your E-mail and Serial code should you use the Pro Edition – these cannot be transferred using this method. You will need to use another activation on the new machine.

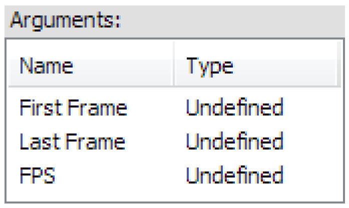
EDITING ACTIONS

There are lots of included actions with DS Game Maker that should let you do most things. However advanced users will find that they need to make their own actions sometime to perform some C code that isn't already in one of the included actions. Advanced users will also want to tweak the included actions to suit their own needs. To do this there is a tool included called the 'Action Editor' which can be found underneath the 'Tools' button on the menu bar. The Action Editor is split into 3 sections:

The first and left most section is a list of all actions in the specific DSGM installation. If you have not created any actions this will be the same list as anyone else who downloads the software. So use this list to 'open' one of the actions to change it or view it. For example to edit or change 'Draw Variable' double click 'Draw Variable' in the list. Once you have done this you will see that the middle section has changed. It now shows the C source code of the action that you opened. You will also see that the right most and last section has also changed to reflect the settings of the newly opened action. The middle and right sections in this window let you edit the action selected in the left section.

ARGUMENTS

If you followed the tutorial to make the 'Touching Balls' game or if you have in fact made anything really with DSGM then you will see that every action has 'arguments' which are provided to give the action information that it needs to function. So 'Animate' needs to know the FPS, First Frame and Last Frame to do function, so these are the 3 arguments.

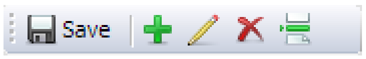

Arguments:

Name	Type
First Frame	Undefined
Last Frame	Undefined
FPS	Undefined

An argument also has a 'type'. This is because in certain cases an argument will want to take a different kind of value. For example an action which turns something on or off will need an argument that can only accept those 2 possible values (programmatically, True or False). In other cases like for 'Change Sprite' an argument for the 'Sprite' should let the user choose a sprite from the sprites in the project.

Type	**Description**
Undefined	Anything can be put into the argument box
Screen	Allows selection of the Top or Bottom Screen
TrueFalse	The user has to select 'True' or 'False'
Variable	The user can still put in the name of a variable they have declared with the 'Declare Variable' action, but a drop down box is pre-populated with the project's Global Variables
Object	The user must pick an Object
Background	The user must pick a Background
Sound	The user must pick a Sound
Room	The user must pick a Room
Path	The user must pick a Path
Script	The user must pick a Script
Comparative	A list is generated with comparative terms like 'Greater than' and 'Equal to'.
Font	The user chooses a font from all of the available fonts in DSGM
Unrestrictive	Basically the same as Undefined, but the user doesn't have to fill in the box.
Sprite	The user must pick a Sprite.

The toolbar shown below is used to work with the action's arguments:



The 'plus' icon lets you add an argument, whilst the 'red cross' symbol deletes the currently highlighted argument. Note that deleting an argument does not remove references to it from the C code. To edit the selected argument, use the pencil.

Arguments in the Code

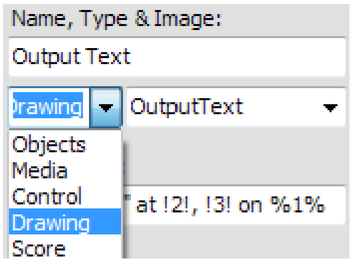
Now that we have seen how to change these arguments, we can explore how to use them in the C code. To help, open the action 'Output Text'. All of the 4 arguments are used in the same order within the code. To use an argument in the code, firstly put a single exclamation mark - !. Now add the number of the argument in the list starting from 1, and finally add another exclamation mark.

So to utilize the value the user has given for the first argument (Screen) in the C code, you put !1!.

To utilize the value the user has given for the third argument (Y tile), you put !3!.

Editing An Action

You can make changes to the C code of an action at free will. You can also edit the arguments and so on but when you are done, you need to save these changes. To do this use the 'Save' button near the top right of the window. If you open another action without saving the one with which you were working, the changes will not be saved.



Action Categories

Every action is also categorized. These categories are: Objects, Media, Control, Drawing, Score and Extra. These are the categories also shown in the actions pool when editing an object (the tabs). So now it's clear where an action appears within these tabs. The drop down to change the category is shown left.

Action Icon

Every action also has an icon which you can choose from the included DSGM action icons using the second drop down box. This icon is shown in the actions pool in the object window and also in the actions list on the left.

The action icons are stored in C:\Program Files\DS Game Maker\ActionIcons.

If you plan to create an action icon, please meet these requirements:

- Keep the styling consistent with other icons. Use 'Generic.png' as a template
- Use 32-bit transparency and save as a PNG file
- 32x32 pixels is the size you must use.

Once you've made an action icon, close the action editor and reopen it. You will now find the icon is selectable in the drop down box.

List Display

In DSGM 4, an action's arguments were shown after the name in the list, as such:

```
Output Text 0;1;1;Hello World
Create Sprite 0;0;1;2;Sprite_1;0;50;50 (theoretical...)
```

This is fine but it is not very visual or human-like, so in DSGM 5 actions can now be made to show a sentence like phrase in the actions list whilst retaining a logical name. For example:

```
Output "Touch to Play On" at 8, 20 on Bottom Screen
If X is Equal to 5
…
```

To change this, use the 'List Display' field in the action editor. Here is an explanation of the replace-symbols you can use. Note that the argument each time is 1, or theoretical:

Replacer	Description	Output
!1!	Replaced with the argument value	1
%1%	Replaced with a string description of a Screen argument. E.g. 0 = "Bottom Screen"…	Top Screen Bottom Screen
1	Replaced with a string description of a non-string-stored argument. Usually a comparative. For this the value is stored like == as in C. So !1! shows == but 1 shows 'Equal to'.	Equal to True (if 1) False (if 0) Greater than …

Application

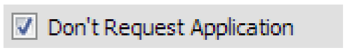
An action can 'apply' to an object. If you followed the tutorial you would have seen this:

"This is because the action can be *applied* to different objects or *instances* of objects.

- **THIS** is the current instance – the specific ball that is touched. So, if 'Delete Object' is applied to itself, the touched object will be removed, but only that touched instance
- **INSTANCES OF** – this lets you apply an action to all instances of an object. So for example you could delete all 'Monster' objects if you were making a game involving monsters. In our game there is no need to apply an action to all instances.
- **INSTANCES** – (this is covered in detail later) Whenever you plot an object in a room, starting from 0, it is given an *Instance ID*. So you can apply an action to specific instance numbers which could hold any object in the room."

If an action is to delete an object, it must know what instance or what object to delete. However, if an action is to output text, this doesn't apply to a specific object or instance (text is just generally drawn, overall).

When this is the case, it is important to check the box 'Don't Request Application'. This is so that a variable isn't created for the application when it isn't needed:


[✓] Don't Request Application

Using what the action is applied to in the C code is essential if you require application (think about it!). The 'Applies To' will *always* come back as an instance ID, even if the user applies the action to many instance numbers or one object that has many instances. DSGM handles this for you. Therefore, whenever you want to know the instance ID for the current application, use AppliesTo in the code.

Please see the section titled 'The internals of DSGM' if the following makes little sense.

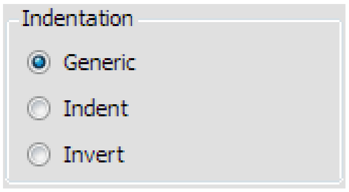
Since all instances are stored in the 'Instances[..]' array, the most useful place to use AppliesTo is to read and write to the properties of instances, by referencing the element within such array with AppliesTo. Consider the following:

```
Instances[0].X = 5;
```

This sets the X position of the first instance in the room to be 5.

To do this to whatever instance the action is being applied to:

```
Instance[AppliesTo].X = 5;
```


Indentation
[● Generic]
Indent
Invert

Indentation

You can get an action to start to indent the list of actions. This is *only* used currently for the 'Start Block' and 'End Block' actions, but you can use it on any action to get visually weird results.

Finding Functions & Learning C

A lot of the functions you will find being called in the actions are from a library called 'Programmer's Arsenal Library', hence why they begin with PA_. There is a reference for this library available online:

http://eldudeds.webs.com/html/modules.html

You will probably also want to learn C if you want to make anything more than simple actions. To do this my best advice is that you buy a book. It is the best way to learn how to program in C. Otherwise look at all the existing actions and follow an online C tutorial. The downside of books or tutorials is that they will not be DS specific and some may even be Windows specific, so watch out.

Note: Don't buy a C++ book or follow a C++ tutorial. C++ is *not* the same as C.

Sharing Actions

If you have made an action that is not included with DSGM, which you also think other people may like to use in their games, you are very much encouraged to share it!

We have a Forum just for this purpose: http://dsgamemaker.com/forum/viewforum.php?f=64

Good actions will be included in future versions of DS Game Maker, and you will be credited in this file.

CREDITS & LEGAL NOTICES

DEVKITARM & COMPLIANCE WITH THE GPL

(PAlib is unlicensed and no longer being developed, so the license of it is not discussed below).

The vendor devkitPro is naturally keen to protect its toolchin – devkitARM. To do this a sensible license has been chosen for it (it is open source). This is GPL. It allows people to make modifications of the toolchain, make it better, and re-release it. The condition is that the resulting product, or any product that uses the toolchain, must echo this sentiment, and be open source.

To confirm; DS Game Maker does not use the toolchain, and must therefore not be released under the GPL. The compiling process is merely initiated by our software and handled entirely externally – it is only called and the result used.

DEVKITARM, LIBNDS

C:\devkitPro, C:\Program Files\Toolchain.7z

http://devkitpro.org

devkitARM by the vendor devkitPro is an absolutely excellent toolchain for creating NDS binaries. Truly DS Game Maker would not exist without it. Thanks go to Dave J Murphy from 'down south'.

MP3ENC

C:\Program Files\DS Game Maker\mp3enc.exe

http://mitiok.ma.cx

"LAME version 3.82 (www.sulaco.org/mp3)
GPSYCHO: GPL psycho-acoustic and noise shaping model version 0.77.
Win32 binaries from www.chat.ru/~dkutsanov/

USAGE   : mp3enc [options] <infile> [outfile]

<infile> and/or <outfile> can be "-", which means stdin/stdout.

Try "mp3enc --help"   for more information
 or "mp3enc --longhelp" for a complete options list"

SOX (SOUND EXCHANGE)

C:\Program Files\DS Game Maker\sox.exe, pthreadgc2.dll, zlib1.dll (covered below)

http://sox.sourceforge.net

Used to convert WAV to RAW at compile time

"SoX is a cross-platform (Windows, Linux, MacOS X, etc.) command line utility that can convert various formats of computer audio files in to other formats. It can also apply various effects to these sound files, and, as an added bonus, SoX can play and record audio files on most platforms."

Browse CVS here: http://sox.cvs.sourceforge.net/viewvc/sox/sox

Z-Lib

C:\Program Files\DS Game Maker\zlib1.dll

http://gnuwin32.sourceforge.net/packages/zlib.htm

Requirement of SOX

PALib

C:\devkitPro\PAlib

http://palib-dev.com

PAlib (Programmer's Arsenal library) is a retired but easy to use library for creating 2D games. It was created by Mollusk but maintained by fincs (and in the future possibly James Garner).

In particular see the PAlib forum: http://palib-dev.com/forum

Scintilla

http://www.scintilla.org

7-Zip

C:\Program Files\DS Game Maker\zip.exe

We are incredibly grateful to the developer of 7-Zip!

EjectMedia, RemoveDrive

C:\Program Files\DS Game Maker\RemoveDrive.exe, EjectMedia.exe

Freeware by Uwe Sieber - www.uwe-sieber.de