(easy-to-learn) Game Maker Language Tutorial

version 6

Made by General_Leo (Pixel Perfect Games)

Index

Hold Ctrl and press F. Enter the keyword (in pink) to find that section of the tutorial

---

ONE1

You may think this is yet another confusing tutorial attempting to explain how game maker's coding works. Well let me just say that I'm here to help all of you who REALLY want to learn it but don't want to take classes or attempt to understand some of the mumbo jumbo of the other GML tutorials out there.

I promise I will do what I can to stay away from using technical terms. I'll talk as if you guys have no idea what your doing. This should help you understand things alot better. I DO recommend you at least know the basics of D&D (Drag and Drop) to understand my tutorial.

The **BOLD** text is commands and the non-bold text is the parts you need to change. (for TWO2 and THREE3)

P.S. This tutorial is ment to teach you GM6 or above and may not be completely accurate for GM5.3 or below.

TWO2

Starting with the very basics here. You'll notice that the basic setup is the same as D&D (Drag and Drop) if you just read through it like a normal sentence. Also, note that capital letters DO matter! The built-in commands will always be lower case.

if something
do this

By replacing "something" with something such as a variable, you can tell GM what to do but only if it meets a requirement. Lets start by worrying about "something" rather then the "do this" part for now.

if Timer**=**1
do this

That code will do whatever as long as the variable "Timer" equals 1. You can also use < (less then) or > (greater then) in the place of the = sign. Remember "timer" and "Timer" are different so be sure to use correct capitalization!

if Timer**>**1
do this

if Timer **>** 1
do this
Here I just want to point out that spaces between everything doesn't really matter. The above code is the EXACT same as the one below it. They read the same so it's just a matter of what is easier to read for you.

if Timer **>=** 1
do this
Another thing you can do is use two >, =, or < signs. GM understands >= as being "greater than or equal to". Also, instead of using a variable, you can use a command. The "if" commands are listed in the codeing box at the bottom when you start to type. All of which are explained in the help file. (GM manual)

if distance_to_object(me**) =** 10
do this
This may look like a big jump from the above codes but lets take a look at it. If the distance to the object "me" is equal to 10, it will do the code. (thats 10 pixels by the way) For this command, you probably would want to use > or < instead. This command is often used in enemy AI to detect how far from the player the enemy is or how far from a certain object. There is a section of this tutorial where I explain a bunch of commands such as this one. Check the index.

if distance_to_object(me**) >** 50 **&&** Timer **>** 1
do this
Ok, this code is a little harder to understand but still easy to use. "&&" means just that... it means "and". So this code means, if the object "me" is further then 50 pixels away from this object AND this object's variable "Timer" is equal to 1, it will run the "do this" part of the code. Remember it will NOT do anything unless it meets both requirements. You can add as many requirements as you want by just putting more and more &&'s in there...

if Timer **>** 1 **&& <** 10
do this
Can you see anything wrong with this code? Yes it is WRONG and GM will point that out. When using &&, it's kinda like using "if" only it's mixing it with another. When using &&, you must repeat the command or variable as if it were separate. Doing the above code makes GM think "if what is less then 10???"

if Timer **>** 1 **&&** Timer **<** 10
do this
This code is helpful because it will only do the code when the variable is within the range of 2 to 9.

if Timer **>=** 1 **&&** Timer **<=** 10
do this
This is basically the same as the one before it only you can see I added some = signs in there. Instead of checking if it's only greater then or less then, now it will also accept it if it's equal to it which makes it give you a range from 1 to 10. VERY important you remember that if one of the symbols are going to be an = sign, that it's second. <> also would work. Basically checking to make sure it is not the amount.

if not Timer **=** 1
do this
Now this code looks a little different but it is just like using <> only it looks better and may be a

little easier to read. (if Timer <> 1) There is yet another way to use not...

if Timer **! =** 1
do this
That code is the exact same as the last. placing a ! in there works EXACTLY like putting "not". Using this and && in there you can make your codes looks pretty complicated. :) Also, note that functions (almost anything but variables) need the ! before it instead of after. An easy way to know when it's before or after is if it has an = sign or not. If it does, it comes before the = sign. If not it comes before the function itself.

if Timer = 1 || Timer = 2
do this
This is the last of the shortcuts i'm gonna tell you about. Putting || in your code is like placing a mirror in there. This code will activate if Timer is equal to 1 OR equal to 2. Yes, this is a replacement for putting in "or". The | lines are made by holding shift and pressing the button above your enter key.

Now lets look further into the same codes as before but mess with the "do this" part.

if x = 255
x = 100
This code would check to see if the object's x value is exactly equal to 255. If it is then it will change the x value to the value 100 ; jumping your object to a different spot in the room.

if Timer **>=** 10
move_towards_point(10,20,30**)**
That code will check if Timer is equal or over 10. If it is, it will move the object towards the point 10x20 at the speed of 30. How about we add more then one effect?

if Timer **! =** 2 **{**
move_towards_point(10,20,30**)**
Timer = 2 }
Ok, now it's starting to look like code huh. Just break it down and see what it says. If timer is NOT (notice the ! in there making it mean not) equal to 2, it will move towards the point AND change Timer equal to 2 making the code not do anything else. Using { and } in your code is just like using brackets in D&D. If you have more then one effect, you will need to add { after the "if" part and then add a } at the end if the entire effect list. If you are using &&, you still only need one set of brackets.

if Timer **=** 1 **&& distance_to_object(**me**) <** 10 **&& x >=** 10**{**
x = 1
y = 2
}
Notice I used 3 ifs but I still only need 1 set of brackets. You'll never need more then one set unless you have a requirement inside of a requirement like this.

if Timer = 1 || Timer = 2**{**
x = 300
y=200
if Timer = 2**{**
Timer = 3
}}
If Timer is equal to 1 or equal to 2, it will move the object to 300x200 and if Timer was 2, it changes Timer to 3. Remember this code would still activate if Timer is 1 but it just will remain 1 at the end of this code.

Also, remember you don't always need to use an "if" in your code.
Timer **+=** 1
If your code is in the step event of an object, putting this code will make it constantly raise the value of "Timer" by 1.

Timer = 1
This IS different because it would change the value of "Timer" to the value of 1 meaning it will just keep changing it to 1 (if put under step). If your going to do something like that, (for lag reasons) you may want to use something like this...

if Timer **!** = 1
Timer = 1
Now again, assuming this code is under the "step" event of an object, this code will ONLY change Timer to 1 if it isn't already 1. Using the last code will be changing Timer to 1 every step of the game. Doing that kind of thing too many times WILL cause lag eventually because it's constantly loading those.

Also, another helpful thing to learn is how to effect an object with the code being in another object.

x = Me**.x**
If there is an object named "Me" in the room and you run this code from another object, it will change the x value of this object to the object "Me"'s x value. Still with me?

x = Me**.x**
y = Me**.y**
This code will make the object 'stick' to the object "Me" by keeping the x and y values the same Keep in mind it will only update every time you run the code. If in step and you havent changed the game speed, it will "jump" to the object 30 times each second. It's common to see it trail behind you by a few pixels before updating again. I'd recommend drawing a sprite for this instead of using objects but anyways...

x += 1
This code is most likely the type of code used in most platformers or RPGs. If put into the "keyboard left" event of an object, it will rapidly move the object to the right. using a - (minus) sign instead of the plus sign will cause it to go backwards making it (in this case) go left.

You should now know some of the major parts of coding. Now all that is left is learning the built-in codes from D&D and maybe a few more even. I'd recommend checking the help file (Game Maker Manual) and click the index tab. Then start to type some stuff in or use the search tab if you have no idea of the code you need. Anyways, here are some important ones...

THREE3

These are only the most common commands. PLEASE use the GM manual to look up all of them. Simply open GM and go to "help". Also note that all commands will NOT work if you use capital letters.

SETTING UP AN OBJECT
---

1. **image_alpha**

This controls how transparent (see-through) your object is. 0 would be completely non-visible and 1 would be completely visible. Usually 0.4 is a good setting making it see-through but not too much so you can't see it at all. GM6 can do alot of alpha values with very little or no lag at all. GM5.3 had alot of trouble doing this.

2. **image_index**

This controls what frame your object is on. (of the sprite your object currently has). Remember sprites start with frame 0 so if your sprite has four frames, they are numbered 0-3. If you use this, you probably want to also set your image_speed to 0 so your image_index isn't cycling through the frames after you set it.

3. **image_speed**

Setting your image speed is setting how fast it would play through the frames of a sprite. Often times, putting an object in a room will make the sprite appear really fast and it will need to be slowed down. in the create event of an object, put this "image_speed=0.5" this will make your sprite play half as fast which helps alot usually. 0 would make it stay on one frame; letting you use image_index to tell it which frame to use.

4. **sprite_index**

This command changes the sprite of the object. If you are making a platformer, you may want your character to use a different sprite when jumping. When you press up, put this code in there and put the name of the jumping sprite.

5. **solid**

This will change if the object is solid or not. You will need to change it to "true" or "false". solid=true will make the object solid...ect.

6. **visible**

This will change if the object is see-able or not. You will need to change it to "true" or "false". visible=true will make the object see-able by the player...ect.

7. **persistent**

This will change if the object is persistent or not. You will need to change it to "true" or "false". persistent=true will make the object always be in every room at the same location unless told otherwize.

8. **speed**

Setting the speed will change how fast the object is moving. This is NOT the same as changing the image_speed. speed is used anytime an object is moving in any direction.

9. **x**

An object's x value is how far over it is. It can be changed to "jump" the object left or right. Platformers use x+3 to move right and/or x-3 to move left. (changing the "3" of coarse for speed.)

10. y

An object's y value is how far up or down the screen it is. It can be changed to "jump" the object further up or down.

11. hspeed

hspeed is kind of like changing the x value only it's better for acceleration. hspeed+=1 would constantly raise the horizontal speed (going to the right) making it speed up slowly. adding an if code before it can be used to limit the speed.

12. vspeed

vspeed is kind of like changing the y value only it's better for acceleration. vspeed+=1 would constantly raise the horizontal speed (going downward) making it speed up slowly. adding an if code before it can be used to limit the speed. vspeed is usually a good way to setup gravity in a platformer.

13. gravity

Anytime gravity is greater then 0, you will be pulled down. To make an object stop on a solid platform, you must set the gravity back to 0. You can use gravity for jumping by slowly increasing it but vspeed is more widely used in this case.

14. image_angle

I'm pretty sure you need a registered version to use this but it is really helpful. If you do something like `image_angle=point_direction(x,y,mouse_x,mouse_y)` any object will face the mouse automatically without you needing to make the sprites have a bunch of subframes of it at different angles. This only needs 1 sprite frame and uses the right side as the "front" of the sprite. The graphics quality drops a little if you use this so it's not always the best way to go. Just remember you **can** have an animated sprite that can turn and everything using this!

Changing things IN-GAME

---

1. `instance_create(x,y,obj_whatever)`

This is used VERY often to put an object into the game that wasn't already in the room when you started the game. x and y are the coordinates of where you want the object to be created at. 0 for x and 0 for y would put the object at the very top left corner of the room. obj_whatever should be replaced with the name of the object you want to create.

2. `distance_to_object(obj_whatever)`

This will tell you how far "obj" is (in pixels) from the object running this code. This is mostly helpful if used in a requirement such as "if distance_to_object(OBJECT) < 100". It helps you change things according to where an object is at the time.

3. `distance_to_point(obj_whatever.x,obj_whatever.y)`

Similar to distance_to_object. Instead of using an object, it uses any point in the room. Change x and y to the point you want it to measure to.

4. `move_towards_point(x,y,speed)`

This will move it's object to the x and y values you put in. replace speed with the speed you want it to move to the point with.

5. `keyboard_check(vk_left)`

This code will check if an arrow key is being held down or not. You can replace "left" with any of the following: left, right, up, down, alt, anykey, backspace, control, nokey...ect You can also change it to keyboard_check_pressed() or keyboard_check_released() to check ONLY when they button is pressed down (once at a time) or when the key is released (once at a time).

6. `instance_place(x,y,obj_whatever)`

This helpful code checks if "obj" is at a certain point. This can be used alot in just about all types of games.

7. `sleep(ms)`

Sleep is like a built in timer where GM completely pauses for the amount of milliseconds you put in the place of "ms" (which stands for milliseconds). Remember NOTHING can change during this pause time. I think the only thing it doesn't stop is sounds.

8. `sound_play(snd_whatever)`

This is the basic code for playing a sound effect or background music of some kind. You can run sound_loop(snd) after this code to make the sound play over and over again till the room ends or until you put sound_stop(snd). Replace snd with the name of the sound.

9. `random()`

Making a variable random is something lots of people have questions on. random(3) would give you ANY number between 0 and 3 including decimal numbers. If you want it to pick a rounded number, you can do this... round(random(3)) this will pick: 0, 1, 2, or 3. Also, you can put a number after it to tell it which number is the lowest that should be picked. round(random(3)+2) would give you 2,3,4, or 5. Also, you don't have to use round(), you can also use ceil() to round upward or floor() to round downward. To use random, you most likely need to set it as a variable. Do something like this... selection = round(random(3)+1).

10. `instance_nearest(x,y,obj_whatever)`

This code finds the nearest of the object you put into it. Using this code, it will take it's own x and y and find the nearest object of "obj_whatever". Finding the nearest enemy which could be many different objects is ALOT more complicated...

FOUR4

Ok, I'm going to try to explain the very basics of arrays. Hopefully you know exactly what global variables are by now. The arrays all start with 0 but I like to start with 1 just because it looks a little more normal.
So when creating a new one, the first would be arrayname[0,0] ...ect

arrayname[#]

This is the basics of an array. Just like any variable, you give it a name and set a number for it.

Item[1]

Ok, now the number 1 is NOT it's value, it's merely the number of the array. Now your ready to set this array to something, instead of a number, lets give it a sentence.

Item[1]="This item is used for healing"

Now that we have set all 3 parts, it's ready to be used. It can be used EXACTLY like a variable can.

draw_text(x,y,Item[1])

Now this part of a code will draw the text "This item is used for healing". Understand? This array setup is almost exactly like a variable. Lets do a few now.

```
Item[1]="Potion"
Item[2]="Ether"
Item[3]="Coins"
```

Now this could be used to draw an inventory list of you'd like. By setting each one as a different number, your setting them all different. I think you know how to draw them from here so now lets try something a little more complicated with these.

```
Item[1,0]
Item[1,1]
```

Now your probably looking at this thinking "wtf is he doing now?". This is where arrays become very helpful (and maybe a little complicated). You can set a sort of 2nd layer to each number. Kinda like keeping multiple parts on one variable. Lets say you want to set up a potion for your game.

```
Item[1,0]="Potion"
Item[1,1]="Heals 50 HP"
Item[1,2]=1
Item[1,3]=100

Item[2,0]="Apple"
Item[2,1]="Heals 75 HP"
Item[2,2]=3
Item[2,3]=200
```

Ok, first I have made parts of a potion. Here, I'm using 0 as the name, 1 as the info, 2 as the quantity you have, and 3 as the selling price. Now your free to draw these each where ever they

are needed. Now if you want to draw parts for the apple, all you need to change is the first number of each. If you still don't understand, let me show an example to make multiple items. Make a script and run it once in the beginning of your game to create these items so you can refer to them in your game.

```
Item[1,0]="Potion"
Item[1,1]="Heals 50 HP"
Item[1,2]=2
Item[1,3]=100

Item[2,0]="Ether"
Item[2,1]="Heals 30 MP"
Item[2,2]=3
Item[2,3]=125

Item[3,0]="Coins"
Item[3,1]="Currency of the island"
Item[3,2]=12000
Item[3,3]=100

Item[4,0]="Life up"
Item[4,1]="Revives a fallen team member"
Item[4,2]=1
Item[4,3]=1000
```

Do you understand now? You can go on with this for as long as you like. Also, remember arrays aren't only for drawing, they can be used ANYWHERE a normal variable can be used. Last but not least, you can make arrays global too. Yeah, you guessed it...

```
global.Item[4,2]=1
```

...would allow you to refer to it anywhere and would make it keep it's value from room to room. This part of the code being the quantity of the item "life up" btw.

Now if you really want to make your game impressive and keep things orderly, check this out...

```
draw_text(x,y, global.Item[Numb,0])
```

Using "Numb" as your variable and putting it as the first number in the array, you can change what text shows up according to what you set "Numb" to. Setting it to 1 (using the above list as an example) it would show the text "Potion" and changing "Numb" to 2 would change it so it writes "Ether" instead. Note that you can put a variable in the place of either of the 2 numbers in the array to allow changes. This helps a TON with making inventorys, shops, status screens ...ect

---

FIVE5

---

1. Custom alarms not using alarms

```
timer+=1
if timer>50{
do this
do this
do this
timer=0}
```

Using this is a lot more accurate then using alarms. It's also better because it's in-code and requires less of a mess in your objects. Just make sure you put it in the step event of an object. You can also add another "if" before the timer+=1 so it doesn't just keep going after your done with it.

2. Automatic adjusting screen

```
view_hborder=view_wview/2 -obj_player.sprite_width
view_vborder=view_hview/2 -obj_player.sprite_height
```

This is probably one of the best "smart codes" I know. It resets your view so that "obj_player" is always perfectly in the middle. If you have a game where the player's sprite changes sizes or shapes, all you gotta do is run this code once after it changes sprites and it is perfect everytime! Just make sure you have views turned on.

3. Perfect landing (platformers)

```
if vspeed > 0 && !place_free(x,y+vspeed)
move_contact(270)
vspeed = 0
```

You may notice in your platformer that when you land, your character stops just above the ground, before landing, then drops the last few pixels. It can get very annoying at times. Use this code on your character when he comes in contact with the ground. It tells GM the exact amount of downward speed it needs to land perfectly. You won't notice any difference other then landing will be nice and smooth.

4. Easy gravity (platformers)

```
if !instance_place(x,y+1,o_Solid)
gravity=0.5
if instance_place(x,y+1,o_Solid)
gravity=0
if vspeed>10
vspeed=10
```

This code helps because it creates gravity ONLY if there is space under you that doesn't have a solid object. It also limits your vertical speed which helps with falling through floor bugs. Lowering the number helps more but may make it look less realistic. Try using different limit amounts.

5. Rotate / orbit object

```
spin+=10
x=obj_Player.x+lengthdir_x(obj_Player.sprite_width*1,spin)
y=obj_Player.y+lengthdir_y(obj_Player.sprite_height*1,spin)
```

This is a nifty little code that automatically makes the object go in circles around whatever object you put in the place of "obj_Player". It uses the objects sprite to find out how large of a circle it needs to make to go around it. Also, it will automatically jump into place so it doesn't matter where you create the rotating object in the room. It will jump into place and start rotating at the speed of the variable "spin" in this case. Just make sure this goes in your step event to keep it circling.

6. 360 degree turning

```
image_single=facing/10
facing = point_direction(x,y,mouse_x,mouse_y)
```

This is a very commonly used code for creating an object that can turn all the way around pointing tword the mouse at all times. It sets 1 subsprite (frame) for each direction. Works best when the sprite's first subsprite is facing right and the sprite has 36 frames and spinning counter clockwise. Now this code does that as well as makes the object face toward the mouse at all times. The 2nd line can be changed or taken out depending on where you want the object to face.

7. Setting a sprite as your font

Create:
```
global.SpecialFont=font_add_sprite(s_FontSprites,ord(' '),false,0)
```
Draw:
```
draw_set_font(global.SpecialFont)
draw_set_color(c_white)
```

I've seen alot of people ask how to do this. This lets you make a sprite and use it when you use draw_text() instead of a normal font. You can name "SpecialFont" as whatever you want and set draw_set_color() as whatever color you want the text to be. (white makes it use the sprites actual colors) the "ord" you see tells it you have numbers, letters, and symbols included in your font. you can put "ord('0') and have frame 0 starting as the number 0 if you only want to have numbers. It took me quite a while to find the order game maker wants these to be in so I included one of the fonts I put together so you can know the order too. There may still be some more symbols I havent found but all the most common (and some of the lesser common) ones, I've already found for you.

sprite font.gif

8. Good Healthbar

```
//DRAW:
draw_set_color(c_red)
draw_line(x,y-10,x+linelength,y-10)
if hpamount > hpmax hpamount = hpmax
if hpamount < 0 hpamount = 0

//STEP:
linelength= hpamount / hpmax * sprite_width
```

I've been asked many time on how to make a healthbar and many times the person wants it to only be so long even if it's alot of HP. (just goes down slower...ect) Well this code does it all. This one uses a line which is smaller and cleaner looking rather then a rectangle. Just set hpamount, hpmax, and make sure linelength=0 is in create. Also be sure to put the first part of the code in draw and the second part in step. You can mess with the numbers like the 10s for the height (raise number to raise bar). Now you can set any objects health and not worry about line lengths! :)

SIX6

Lastly, I'd like to say thank you for reading this and I hope it helps you get started using GML. Basically try to use the GM manual (help file) as much as possible till you learn alot more codes. Experiment using a mixture of different codes. **The only real way to learn this is by using it alot.**

Any questions or comments?
Email me or IM me with any questions or comments you may have...

PM: **General_Leo** (http://forums.gamemaker.nl/index.php?showuser=3133)
My Site: **PixelPerfectGames.com** (has all my released games and alot of info on other stuff)
My Forum: **http://s6.invisionfree.com/Pixel_Perfect_Games/index.php?act=idx** (feel free to post without signing up. I mostly check it daily.)
Email: **OutlawCecil@gmail.com**
AIM: **Outlawed User**
MSN: **General_Leo@hotmail.com**

I don't mind being Emailed or PMed simple questions. But please try not to email me asking me to make practically your entire game. I might help if you ask but your going to learn alot better if you try it yourself first then email me and ask what you did wrong.