Tutorial for Platform Engine v3.0

By Martin Piecyk

Audience: Intermediate and Advanced users

Version: For Game Maker Studio
Note: This tutorial was written for the original version in GM6 and may have some minor differences to GM: Studio.

Introduction

Welcome to my Platform Engine tutorial. After completing this tutorial, you should be able to make your own platform engine with features like smooth movement for the platform character, exact collisions, moving solids, jump-through platforms, ladders, moveable blocks, and water.

Making a new room

First off, we need to make a new room. Click Add > Add Room (or any other variation you like) to make a new room. Rename this room to "*level1*" and close it.

Creating the game object

Click Add > Add Object, and give it the name "oGame" (without the quotes). Click "Add Event" and select "Create." Then, for the action, click the "control" tab, and drag "Execute a script." Find and pick this script: *gameCreateEvent*. Then click "Add Event" and select "Step." Find and pick this script: *gameStepEvent*.

Creating the character object

Click Add > Add Object, and give it the name "oPlayer1" (without the quotes). Make the sprite: Character > "sStandRight." Make the parent object Basis > "oCharacter". Click "Add Event" and select "Create." Then, for the action, click the "control" tab, and drag "Execute a script." Find and pick this script: *characterCreateEvent*. Click "Add Event" > Step > Step. Add the following script: *characterStepEvent*. Now click "Add Event" > Draw. Add the following script: *characterDrawEvent*.

Placing the objects in *level1*.

Open up the room *level1*. Add "oPlayer1" somewhere near the middle of the room. Then add "oGame" at the top-left of the room. Now click the green check to save and close the room. Then run the game by pressing F5.

If all went well, the game will run with no game errors. Mario should fall downwards and off the screen. However, that's no fun. We need to make some solids for Mario to land on.

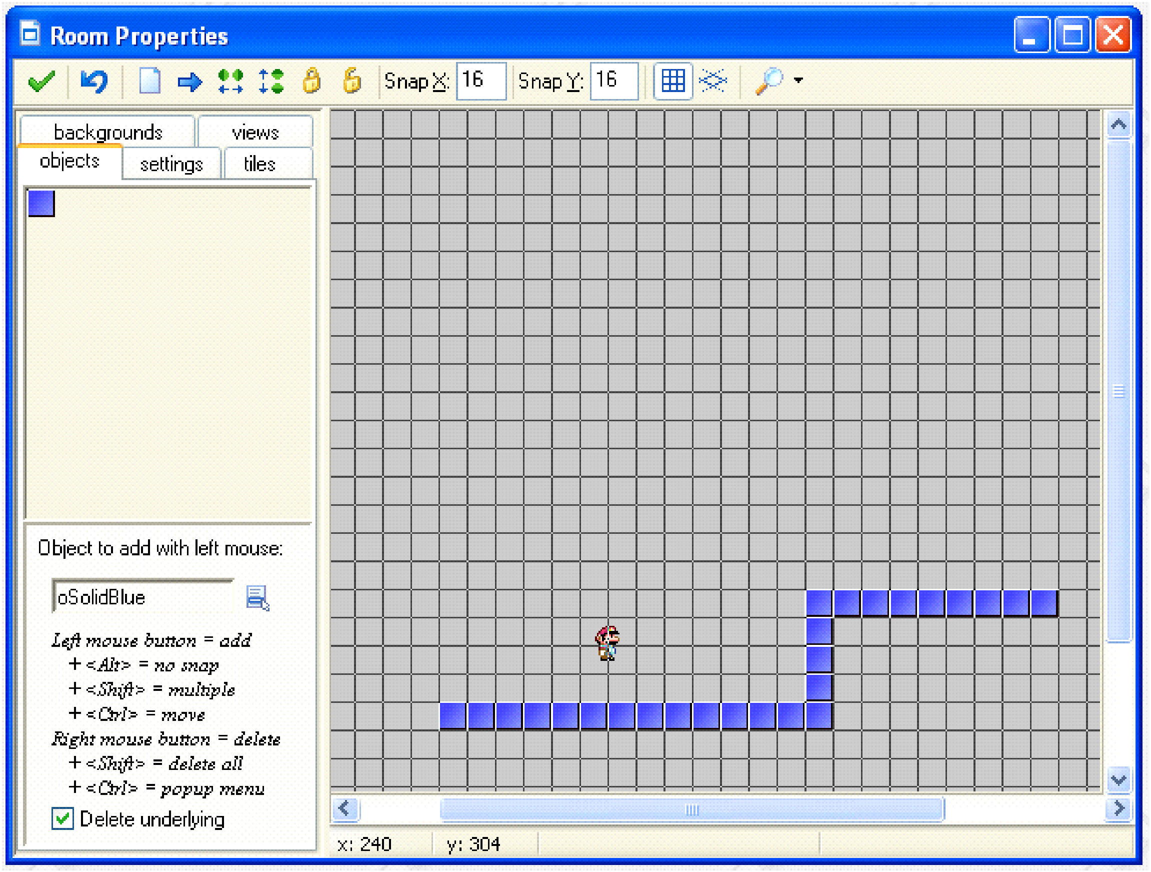
Making solids

Close out of the game and make a new object. Name it "oSolidBlue." Make the sprite “sSolidBlue.” Make the depth 2, so the solid appears underneath the platform character.

Here is the important part: Make the parent Basis > “**oSolid**.” It does not matter if you check the “Solid” checkbox in this platform engine. When you make an object have a parent “oSolid”, the platform character will be able to walk on it. Now you are set. You don’t have to make any events for the solid, so press OK to close out of the object.

Placing solids in *level1*

Open *level1* and create some “oSolidBlue” underneath Mario. It may be easiest to hold down the shift button to make multiple solids. Your level should look like this:



Run the game and if you followed the directions correctly everything should work.

Restarting the level when Mario falls off the edge

When Mario falls off the edge, the room does not restart. We should fix that. Open up the “oPlayer1” object and click on the “Step” event. Drag and drop Control > “Execute a piece of code” after the “characterStepEvent” script. Add the following code:

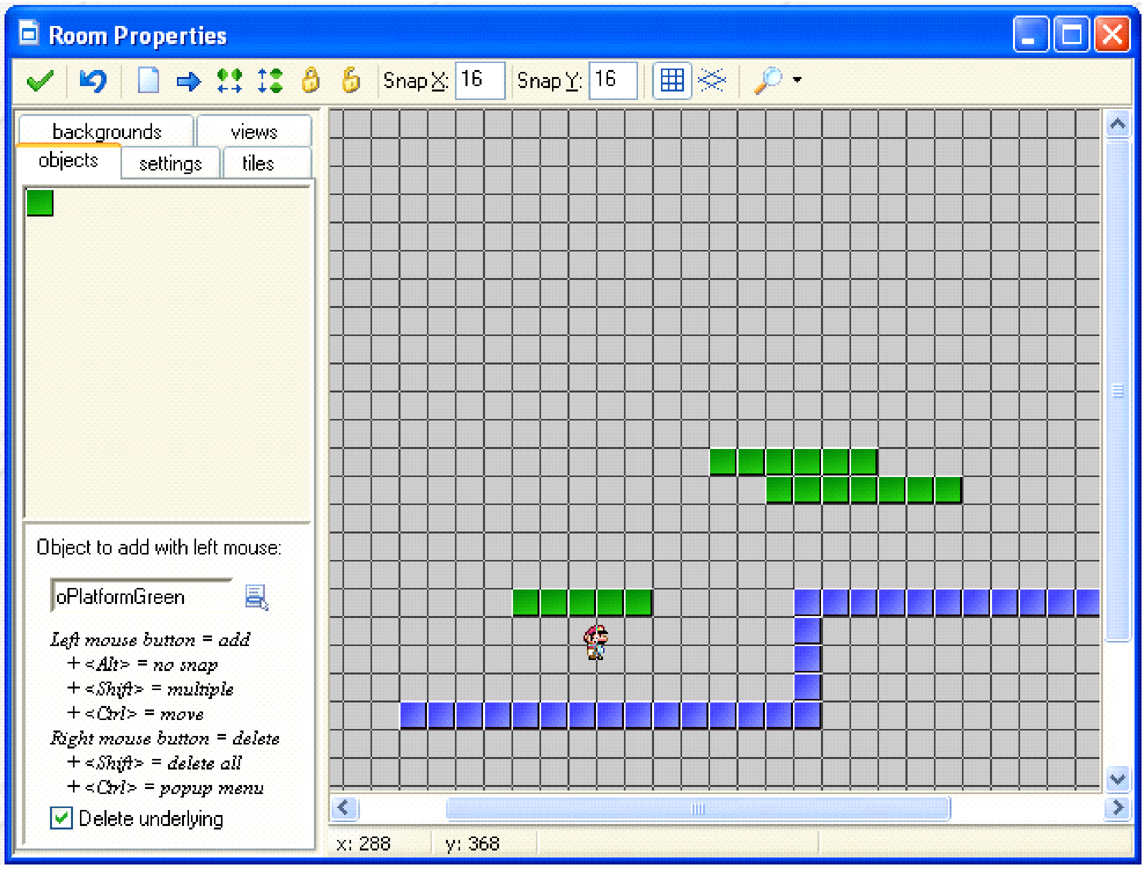
```
if y>room_height
  room_restart()
```

Run the game. When Mario falls off of the edge, the room is restarted. You can make a more advanced death sequence, but I'm just going to keep it simple here.

Making jump-through platforms

Create an object with these characteristics:
Name: oPlatformGreen
Sprite: sPlatformGreen
Depth: 4
Parent: Basis > oPlatform

Then create some oPlatformGreens in *level1* like so:



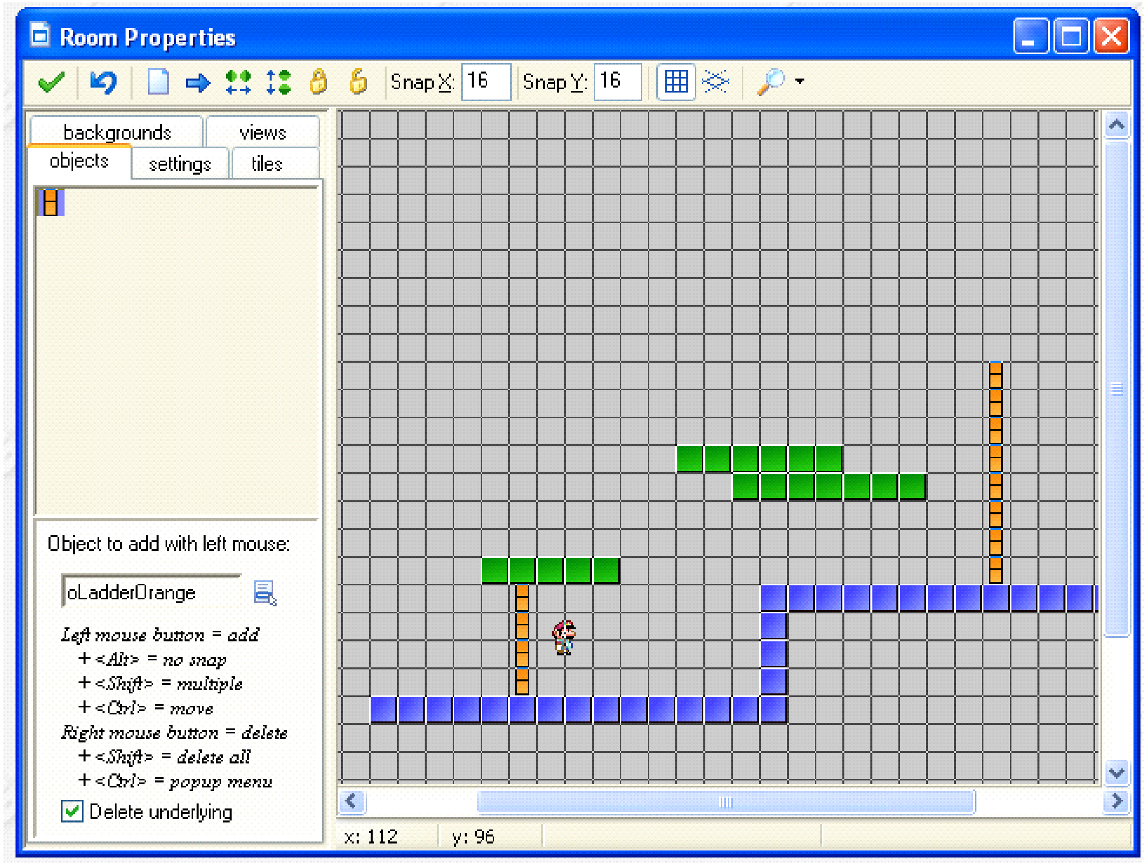
Object to add with left mouse:
oPlatformGreen
Left mouse button = add
+ <Alt> = no snap
+ <Shift> = multiple
+ <Ctrl> = move
Right mouse button = delete
+ <Shift> = delete all
+ <Ctrl> = popup menu
Delete underlying [✓]
x: 288 y: 368

Make sure that there are at most three spaces in between the ground and the platform. If you make a platform too far away from the ground, Mario will not be able to reach the top of the platform. Run the game and make sure the platforms work.

Making ladders

Create an object with these characteristics:
Name: oLadderOrange
Sprite: sLadderOrange
Depth: 4
Parent: Basis > oLadder

Then create some oLadderOrange objects in *level1* like so:



Then run the game and make sure the ladders work.

Making moveable solids

Moveable solids are solids that can be pushed by the platform character and be used to create puzzles. Create an object with these characteristics:

Name: oMoveableSolidBrown
Sprite: sMoveableSolidBrown
Depth: 2
Parent: Basis > oMoveableSolid

Create an event, Other > Outside Room. Add the action, main1 > Destroy the instance. This will destroy the moveable solid when it is pushed outside of the room.

Put two or three “moveable” solids near Mario in *level1*. Make some of them in the air so you can watch them fall from the sky when you run the game.

Making moving solids

Finally, we make the moving solids, one of the main features of the Platform Engine! They are the most difficult to make, compared to the non-moving solids and ladders.

In this platform engine, *moveable* solids are pushed by the character and fall downwards whereas *moving* solids push the character, do not fall downwards by default, and cannot be pushed by the character.

First, create an object with these characteristics:
Name: oMovingSolidPurple
Sprite: sMovingSolidPurple
Depth: 2
Parent: Basis > oMovingSolid

Now we need to decide where we want the object to move. Let's just make it move up-right for a certain amount of time (give it positive horizontal speed and negative vertical speed) and then reverse its motion every 60 steps (that is, every 2 seconds at 30 fps).

Creation event

In the creation event, choose the action "Execute a piece of code" and add this code:

```
viscidTop=1
makeActive()
xVel=2
yVel=-1
alarm[0]=60
```

"viscidTop" tells the moving solid whether or not it is "viscid." `"viscidTop=1"` makes the solid top viscid and `"viscidTop=0"` makes the solid top not viscid. The definition of "viscid" is "Having the properties of glue." A moving solid that has a viscid top allows Mario to stand on top of it without slipping. The moving solid could move left, right, up, or down, and Mario would still be standing on it, and move with it. On the other hand, a non-viscid top will cause Mario to slip, and not move with it.

"makeActive()" makes the object an "active object." An "active object" uses the variables xVel, yVel, xAcc, and yAcc. "vel" stands for velocity and "acc" stands for acceleration. Active objects use these four variables instead of the traditional "hspeed" and "vspeed" variables of Game Maker.

For your convenience, here are the contents of the script "makeActive." It's a pretty simple script:

```
/*
Makes the object "active." An "active" object is one that has the
variables xVel, yVel, xAcc, and yAcc.
*/
xVel=0
yVel=0
xAcc=0
yAcc=0
```

Anyway, let's look at the code from the creation event again:

```
viscidTop=1
```

```
makeActive()
xVel=2
yVel=-1
alarm[0]=60
```

“xVel=2” will cause the object to move 2 pixels to the right every step, and “yVel=-1” will cause the object to move 1 pixel up every step.

“alarm[0]=60” sets Alarm 0 for 60 steps.

Alarm 0 event

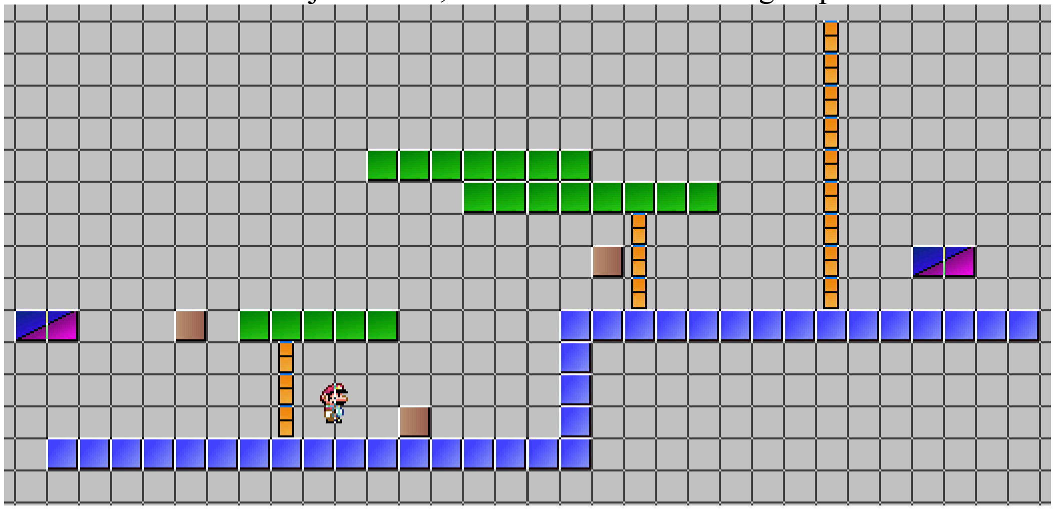
Make an Alarm 0 event. In the Alarm 0 event, choose the action “Execute a piece of code” and add this code:

```
xVel*=-1
yVel*=-1
alarm[0]=60
```

So that should make our object reverse direction every 60 steps.

Placing the moving solids

Now close out of the object editor, and make two “oMovingPurpleSolids” in *level1*:



They should move correctly when you run the game.

Adding a monster

We still never made a monster in this tutorial! In this tutorial, we’ll make a Goomba that conforms to our Platform Engine “standards”. What are the standards? An object that follows the standards will be an active object, using these four variables to move: xVel, yVel, xAcc, yAcc, and must set a collision bounds using the script “setCollisionBounds”. You could make a Goomba any way you want but this tutorial will show you how to stick with the standards I made up.

First, create an object with these characteristics:

Name: oGoomba
Sprite: sGoombaRight
Depth: 3
Parent: No parent

Make the creation event and drag and drop “Execute a piece of code” and add the following code:

```
makeActive()
setCollisionBounds(2,0,sprite_width-2,sprite_height-1)
xVel=2.5
image_speed=0.1
```

makeActive()

Again, “makeActive” gives the object these four variables: xVel, yVel, xAcc, and yAcc. Just because an object calls this script and gets the variables “xVel” and “yVel” doesn’t mean anything will happen. Earlier, we created a moving solid object and set “xVel” and “yVel” to non-zero numbers. These solids moved because their movement is controlled in the Step Event of the platform character. The platform character also uses the variables “xVel” and “yVel.” The movement for the character (using these variables) is also controlled in the Step Event for the platform character. For your convenience, I coded it in for you. However, when you make an enemy that uses the “active” variables “xVel” and “yVel”, you write the code yourself. It’s simple. I will teach you how in the Step Event for “oGoomba”.

“Collision” objects

“setCollisionBounds” sets the collision bounds of the object. The first argument is the left bound (x1), the second, the top bound (y1), the third, the right bound (x2), and the fourth, the bottom bound (y2).

Making the Step Event

Now that you learned important details about the Platform Engine, we can continue.

Add a step event (in oGoomba). Add the following code:

```
yVel+=0.2
if isCollisionBottom(1)
  yVel=0
if isCollisionLeft(1)
{
  xVel=2.5
  sprite_index=sGoombaRight
}
if isCollisionRight(1)
{
  xVel=-2.5
  sprite_index=sGoombaLeft
}
moveTo(xVel,yVel)
```

“yVel+=0.2” makes the Goomba fall downwards.

isCollisionBottom(1)

“isCollisionBottom(1)” checks one pixel below the Goomba’s collision rectangle to see if there is a collision with a solid. Checking one pixel below the Goomba is the same thing as checking whether the Goomba is standing on the ground. If we changed the number “1” to “2” the script would check exactly two pixels below the Goomba’s collision bounds.

If the Goomba falls on top of a solid, its y-velocity should be reset to zero, as described by the condition in the code.

isCollisionLeft(1)

“isCollisionLeft(1)” checks one pixel left of the Goomba’s collision rectangle to see if there is a collision with a solid. If there is a collision one pixel away, we want the Goomba to walk towards the right.

isCollisionRight(1)

“isCollisionRight (1)” checks one pixel right of the Goomba’s collision rectangle to see if there is a collision with a solid. If there is a collision one pixel away, we want the Goomba to walk towards the left.

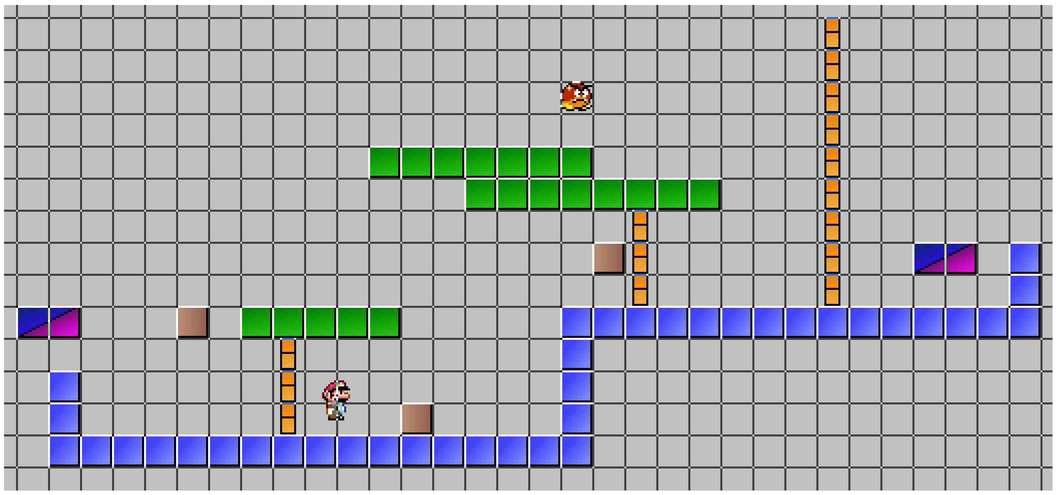
moveTo(xVel, yVel)

moveTo(xVel,yVel) moves the object “xVel” pixels to the right and “yVel” pixels down, stopping precisely at any solids. Conversely, a negative “xVel” would make it move to the left and a negative “yVel” would make it move up. If we didn’t put this function in the code, the Goomba wouldn’t move anywhere!

Note that even though moving solids use the “xVel” and “yVel” variables, they should not have this function in their step event. (Actually, they don’t even need a step event.) As a technical note, the character object contains the code that causes moving solids to move. So if the character object is not present, the moving solids will not move.

Placing the Goomba in *level1*

Now place “oGoomba” in *level1* here:



Make sure that you make some edges so the Goomba doesn't walk off of the solids. When you run the game, notice that the Goomba falls right through the jump-through platforms. That is because they are not genuine "solids." Also notice that you can push the moveable solid on top of the Goomba. Alternatively, you can push the moveable solid to the left and give the Goomba a new path.

Making the Goomba a threat

Mario can walk through the Goomba and can't squish it. So let's fix that. Open up "oGoomba" and add the event "Collision with object oPlayer1."

Add the following code:

```
if other.yVel>0 and other.y<y+5
{
  other.yVel=-6-0.2*other.yVel
  instance_destroy()
}
else
  room_restart()
```

The code basically says: If Mario falls on top of the Goomba, then the Goomba should die and Mario should bounce off. Otherwise, Mario should die, and the room should restart.

Note that this tutorial does not cover the creation of fancy dying effects, although the demo shows some interesting effects when the character dies and when the Goomba dies.

Allowing the Goomba to move up moveable blocks

When you push moveable blocks into the Goomba to make it stuck into a wall, the block moves on top of the Goomba, the Goomba kills Mario, and it basically doesn't look right. To fix this, go to the step event of "oGoomba," and add the following code at the very bottom (after `moveTo(xVel,yVel)`).

```
if isCollisionSolid()
  y-=2
```

“isCollisionSolid()” returns whether or not the Goomba is colliding with a solid (which could be a moveable block, non-moving solid, or moving solid that follows a path). “y-=2” moves the Goomba up. So the code generally says, if Mario pushed a moveable block on top of this Goomba, move the Goomba upwards.

Try setting up the blocks to squish the Goomba to verify that the code works.

Editing the “characterCreateEvent” script

Many variables of the platform character can be changed from within “characterCreateEvent.” Open up the script, “characterCreateEvent” and look for the “user variables” section. There is documentation on the right side of each variable.

Let’s change a few variables. If you want the running acceleration to be higher, you could change “runAcc=3” to “runAcc=5.” If you want to lessen the gravity, you can change “grav=1” to “grav=0.2.” If you don’t want the character to be able to run (while holding down shift) in your game, you could change “canRun=1” to “canRun=0.” (In case you didn’t know, 1 means true and 0 means false.) If you wanted your character to be able to perform an initial high jump when running at max speed, you can write “canFly=1.” If you want your character to be able to jump many times while in the air, then you could write “canFlyJump=1.” The list goes on and on. By the way, you can change almost all of these variables while the game is running in almost any part of your code. For example, you could make a leaf that sets “canFly” to 1. (That is what I did in the demo.)

Using your own sprites

If you have your own sprites, of Sonic for example, and they are bigger than the Mario sprites I provided, you would replace all of your sprites on top of the Mario sprites and change “setCollisionBounds(-5,-7,5,8)” to something like “setCollisionBounds(-20,-30,20,30).” You will have to experiment with the arguments in the “setCollisionBounds” script to find the correct values for your sprites.

Conclusion

Congratulations on making your way through the tutorial! Now you should know how to make your own platform games using this engine. But wait! Don’t stop here. If you wish to make your game more advanced, you may want to look at some more of the scripts. All of the scripts are documented, and their names most often explain their purpose. It will take careful observation to understand the code in characterStepEvent and moveTo, but as long as you have a general idea of what they do, it will really help you out if you want to use this engine.

Anyway, good luck with making your platform game if you plan on making one! May you come up with many insightful ideas and produce multitudes of high-quality games!

Last updated: 2017, January