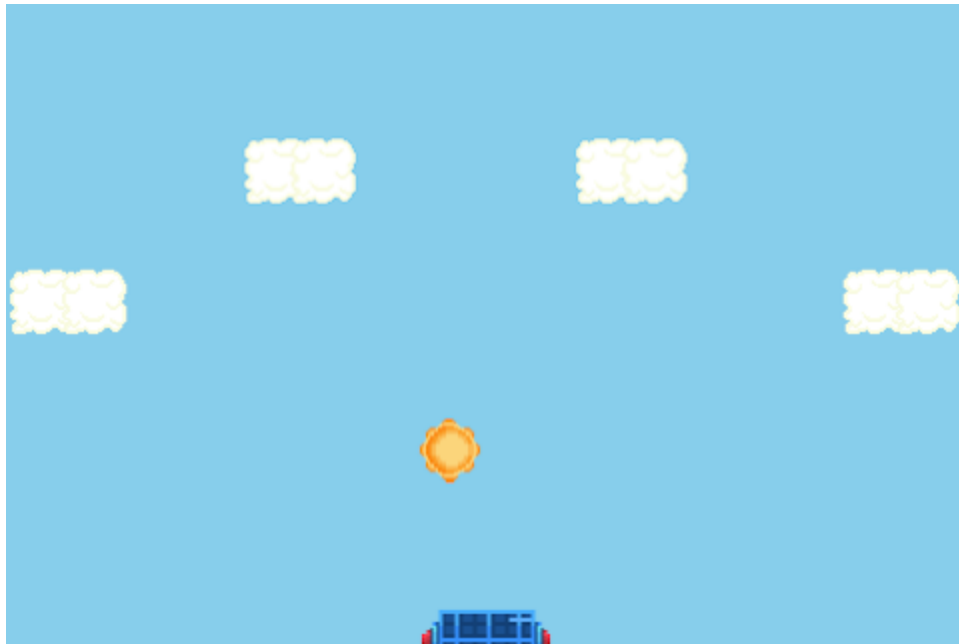


How to Make “Solarball” in GameSalad

by J. Matthew Griffis

*Note: this is a Beginner-level tutorial. It is recommended, though not required, to read the separate PDF **GameSalad Basics** and go through the **Dropcycle** tutorial before continuing.*



In Solarball, you have one job: keep the solarball in play! It's bouncing around the screen and plummeting toward the bottom; move the solar panel to intercept it or it's Game Over! This simplicity makes Solarball a great introduction to creating games in GameSalad and shows how easy it is to add sound and increasing challenge.

In this tutorial, you'll learn how to recreate Solarball. Make sure to download the folder of Resource Files for the game, and play the game on the website to see it in action.

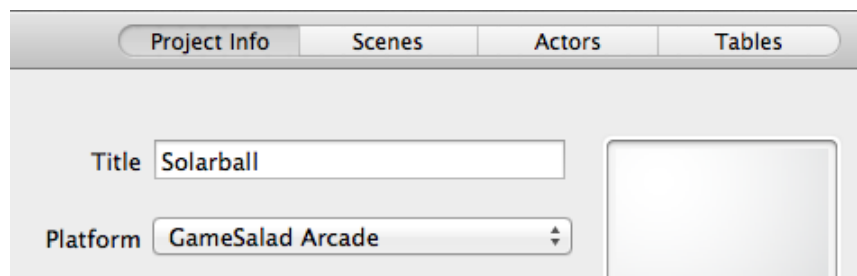
OK! Let's get started.

Open GameSalad and create a blank project. (Note that while this is intended as a Beginner-level tutorial, it does not duplicate the **Dropcycle** tutorial's coverage of some of the most basic interface functions in GameSalad, so if you start feeling lost, consider working

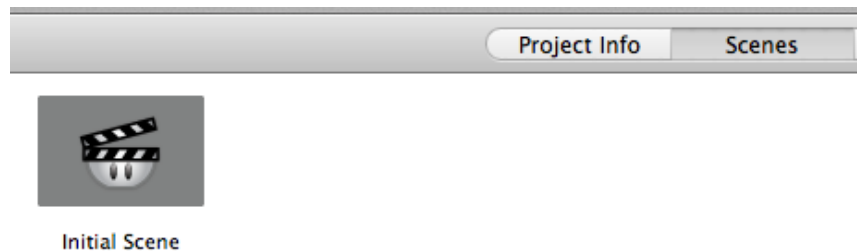
through that tutorial first.)

The blank project should open on the “Project Info” tab. Give your game a title and choose a Platform. Remember that the Platform is the device for which you’re making your game, whether computer, tablet, smartphone, etc., and choosing it changes your project’s screen size to match that of the device, so it’s important to do this right away. As usual for this tutorial series, we choose GameSalad Arcade so we can publish the game online, but you can choose a different Platform if you want. Feel free to fill in the Description and Instructions as you like (you can always do this later).

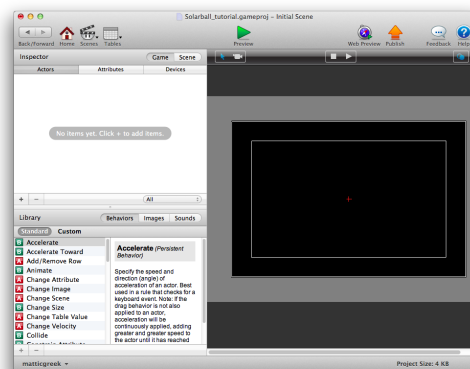
Alright, enough setup! Click the “Scenes” tab to the right of “Project Info”:



Then double-click on “Initial Scene”:

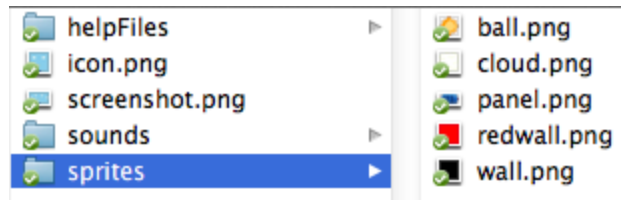


And here we are at the Scene view (hopefully bigger than this image):

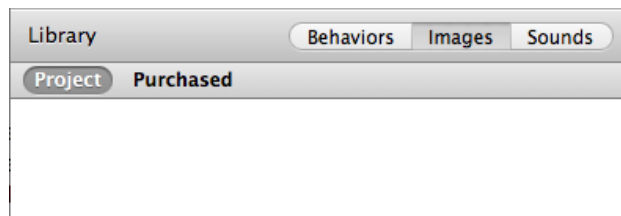


Step 1: Add the images.

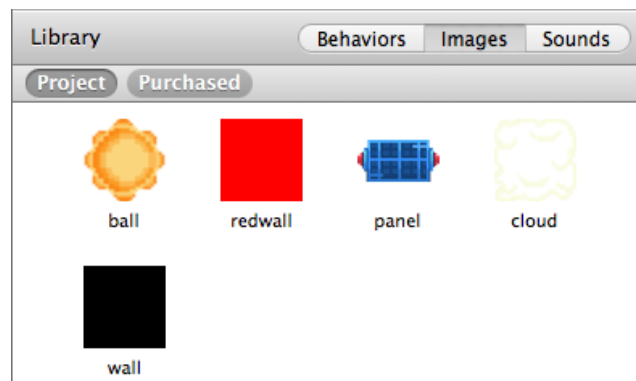
First thing is to import the images from the Resources Files folder. Open it up and look for a “sprites” sub-folder containing images. It should look something like this:



Return to GameSalad and look in the lower-left to see the “Library.” Click on the “Images” tab:

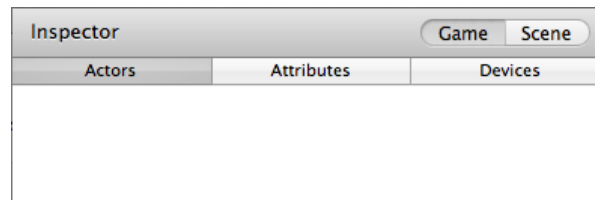


This is where we store our images so we can use them. Adding them to GameSalad is as easy as dragging and dropping them from the folder into the white box. You should end up with this:

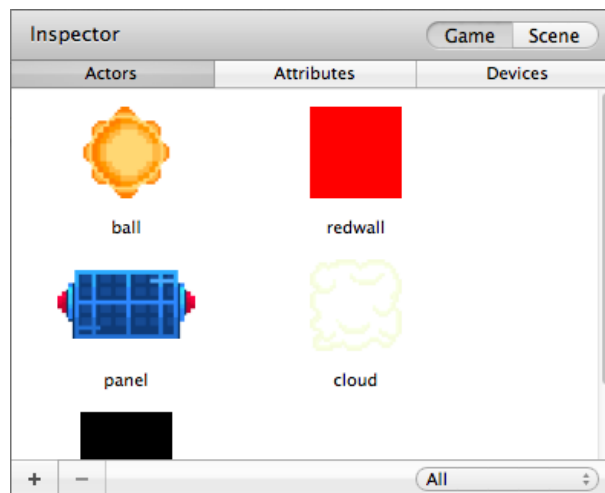


Step 2: Add the Actors.

On their own, the images are not useful--we need to attach them to objects, which we can place and use in the Scene. GameSalad calls them “Actors” and you can see the tab for them in the upper-left of the interface, called the “Inspector”:

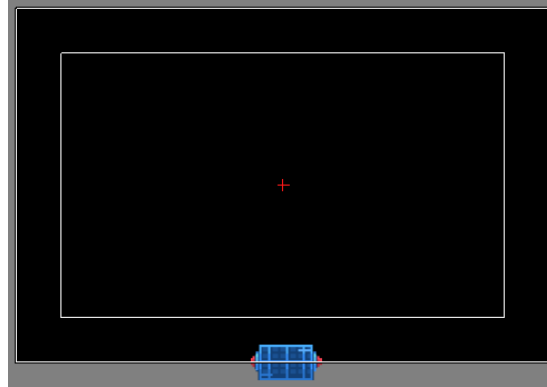


It's easy to make an Actor with an image attached--simply drag the image from the Library into the "Actors" tab of the Inspector. Do so with the images. The "Actors" tab should look like this:



Step 3: The solar panel.

Now that we have Actors, we can put them in the game! Let's start with the solar panel. Drag it from the "Actors" tab into the black rectangle on the right. The black rectangle is the game screen--anything you put there will show up exactly like that when the game is run. Drop the panel somewhere along the bottom of the screen. Note that you can put it partly offscreen, if you want (probably a good idea if you're working with a smaller screen like GameSalad Arcade); that will just mean that only the part within the black rectangle shows up in-game.

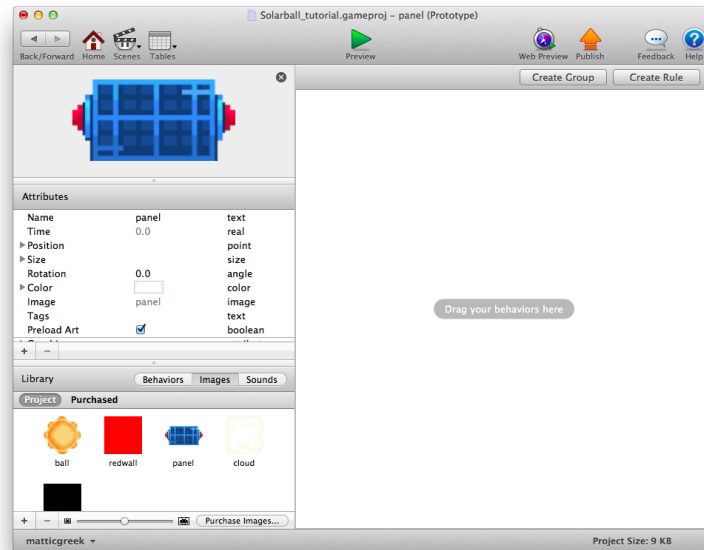


You’ve just created an “instance” of the Actor “prototype.” What does that mean? It means that when you dragged the panel image from the Library into the Inspector, you created a *type* of Actor called “panel,” which still doesn’t appear in the game until you make a copy of it by dragging it into the Scene. You can make as many copies (“instances”) as you like. It doesn’t make sense in our game to have more than one instance of the panel, but you could add more if you wanted to. This is useful because you can tell the prototype to behave in a certain way, and it will apply to all the instances, whether you have 1 or 100.

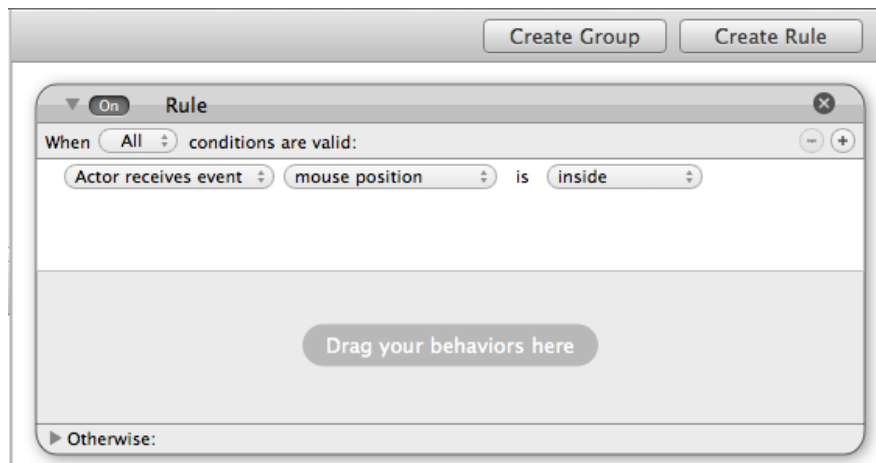
Step 4: Move the panel.

If you click the large green “Preview” arrow at the top-middle of GameSalad, you’ll see that the game doesn’t do anything yet. We’ve placed an Actor in the Scene, but we haven’t told it how to Act, so the panel just sits there. Let’s allow the player to move the panel to the right and left.

If you’re still in Preview mode, click the Back button in the upper-left to return to the Scene view, then double-click on the panel prototype (remember, that means double-click on it in the Inspector, not in the Scene). You’ll see the details of the prototype:

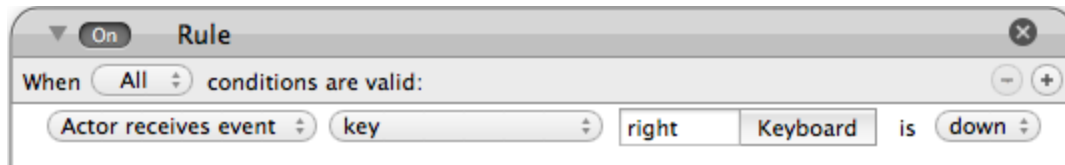


The big empty white box on the right is where we can tell the panel what to do. The first thing is to tell it to act only when we press a button. For that, we need to create a Rule. In the upper-right of the currently-empty Behavior window, click “Create Rule” to produce this:

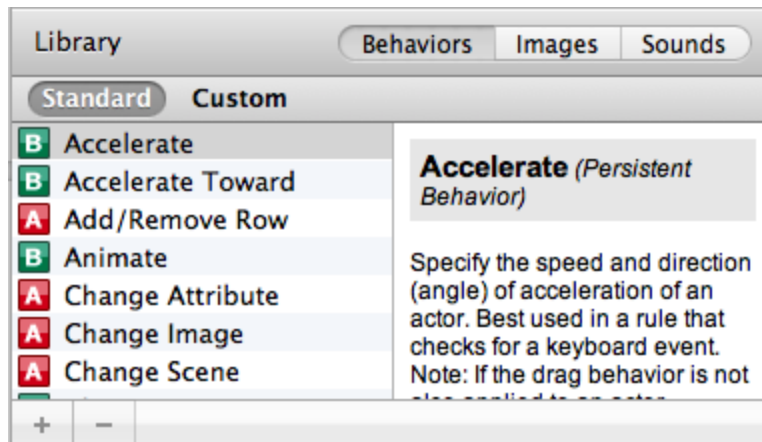


As you can see, a “Rule” is a conditional statement. It tells the game to check if a certain condition is met, and if so, what to do. “Actor receives event” sounds fancy but simply means that the Actor is paying attention to the state of the game and will notice if the condition is met.

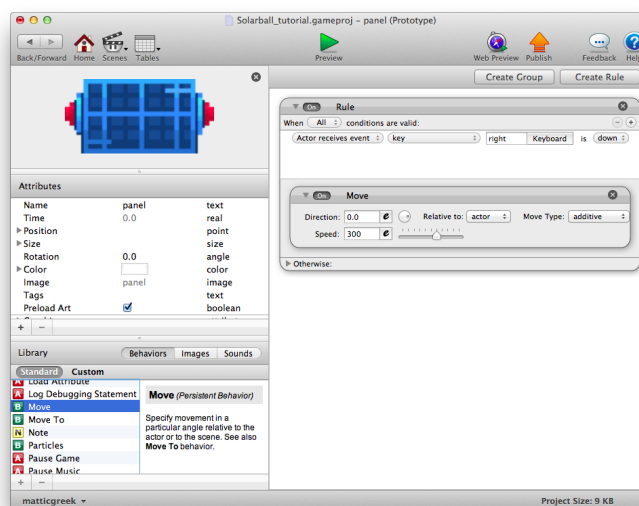
The default condition is that the mouse is inside the bounds of the Actor, but that’s not what we want, so click on “mouse position” and change it to “key.” You’ll see that a new field to choose the key appears. Click on “Keyboard,” then click on the RIGHT arrow key. The statement should now look like this:



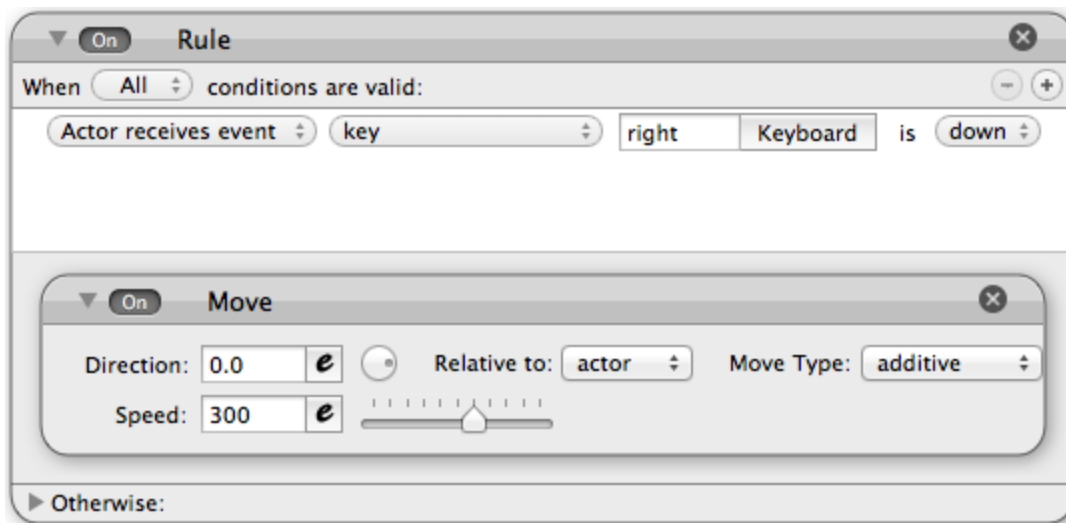
This says that if the RIGHT arrow key is pressed, do something, which is what we want. Now we need to say what to do, which goes in the space below the condition. In the Library, click the “Behaviors” tab:



You can scroll through the list of Behaviors on the left, and read a description of the selected one on the right. The Behaviors are alphabetized. We want to move the panel, so look for “move.” You’ll see we have two options: “Move” and “Move To.” Which should we use? Well, the “Move” Behavior moves in a certain direction with a certain speed, so let’s choose that. Drag it from the list into the Rule we created. Now it looks like this:

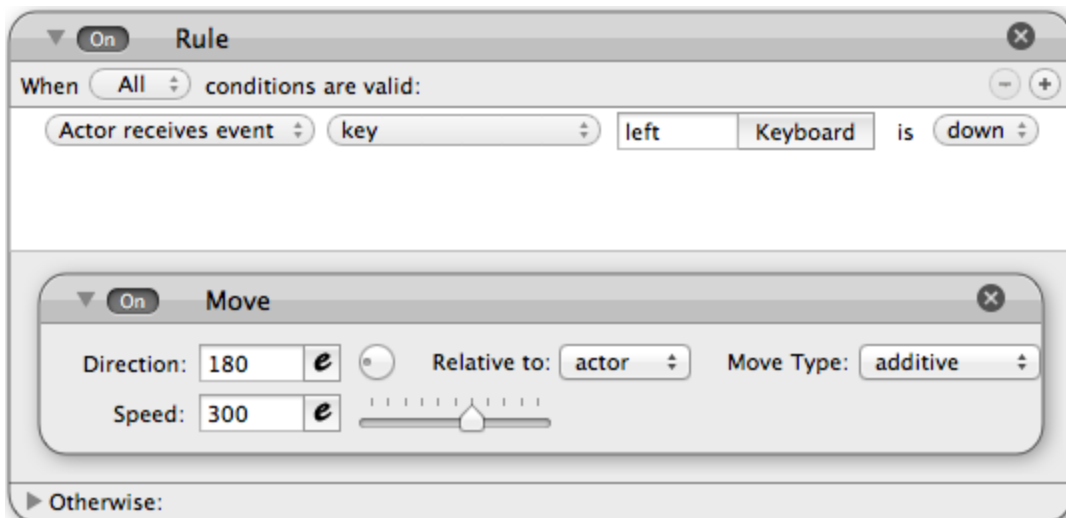


Here's a closer view:



We need to set the direction and speed. Or, actually, we don't, because direction is measured as an angle against the x-axis (which runs positively to the right), so zero degrees is to the right, which is what we want. The default speed of 300 is fine. Click Preview and press RIGHT. The panel should slide to the right!

Now create a second Rule (make sure it is independent of the Rule we just made) and set it up to move to the left. In this case we use a direction of 180 degrees, which is straight to the left:

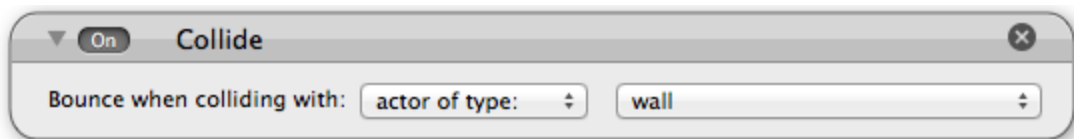


Step 5: Keep the panel on-screen.

Great! The panel now moves right and left at our whim. However, you may notice that when it reaches the right or left edge of the screen, it keeps going (if we hold the button down) and cheerfully slides right out of view. This is not ideal. We'd prefer to keep the panel on-screen at all times.

There are any number of ways to achieve this, but the simplest is to have the panel collide with an off-screen object, preventing the panel from moving farther. Remember that Actors can exist off-screen, outside the bounds of the black rectangle; the player just won't see them. So, let's make some walls just outside the bounds of the screen.

First, let's give the panel one more Behavior. Scroll through the list until you find Collide, and drag it into the panel's Behavior, outside of any Rule. Make sure the type of Actor the panel will collide with is set to "wall." It should look like this:




We are now done with the panel! Huzzah! For reference, its complete Behavior looks like this:



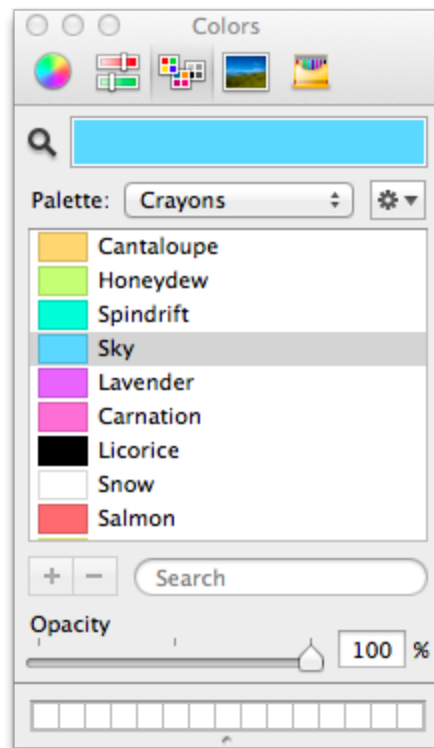
Of course, if we run the game, the panel still goes offscreen, because we haven't actually placed any walls yet. Let's do that. But first, that black background is a downer. How about some color?

Step 6: Add a background color.

Click Back if necessary to return to the Scene view, then click on the *Scene* tab, in the upper-right of the Inspector. Make sure the *Attributes* sub-tab is selected:

Inspector		
		Game Scene
Attributes		Layers
Name	Initial Scene	text
Time	0.0	real
► Size		size
Wrap X	<input type="checkbox"/>	boolean
Wrap Y	<input type="checkbox"/>	boolean
► Gravity		point
► Color		color
► Camera		rect
► Autorotate		attributes

Look, there's Color! Click on that black rectangle to open a pop-up. Make sure the middle icon across the top is selected ("Color Palettes" appears if you hover over it), then choose the "Crayons" palette from the drop-down and look for "Sky." Click it to make the change. So easy!



Step 7: Add walls.

Close the Colors pop-up, then click the *Game* tab in the upper-right of the Inspector to return to the familiar view. To add a wall, simply drag the wall Actor from the Inspector into the Scene. Drop it somewhere just outside the left edge of the game screen, on a level with the panel:



If you run the game and run the panel off the left edge, you'll notice that not only does it not stop moving, but you may not be able to get it back once it goes off-screen! Though it's hard to tell since it's happening off-screen, this is happening because Actors in GameSalad come with basic physics rules attached by default. That means that when you run the panel into the wall and the panel is set to collide with the wall, it gives the wall some of its momentum and pushes the wall to the left! See this for yourself by moving the wall on-screen and then moving the panel into it. You may notice the panel rotate, too. So weird.

Well, that's not a very useful wall, is it? Fortunately, this is easy to fix. Double-click on the wall prototype in the Inspector. We are not going to give the wall any Behavior (it's...a wall), but on the left underneath the image you'll see a list of Attributes. These properties define the wall:

Attributes		
Name	wall	text
Time	0.0	real
► Position		point
► Size		size
Rotation	0.0	angle
► Color		color
Image	wall	image
Tags		text
Preload Art	<input checked="" type="checkbox"/>	boolean
► Graphics		attributes
+ -		

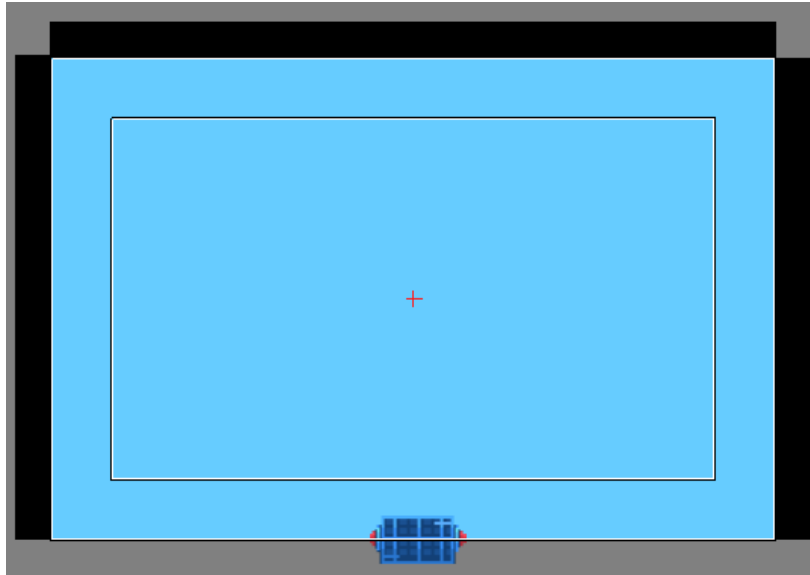
Scroll down until you get to "Physics" and click the arrowhead to the left to expand the category. Keep scrolling until you get to "Movable," which you'll see is checked. Uncheck that box!

Attributes		
► Motion		attributes
▼ Physics		attributes
Density	1	real
Friction	3	real
Bounciness	1	real
Fixed Rotation	<input type="checkbox"/>	boolean
Movable	<input type="checkbox"/>	boolean
Collision Shape	Rectangle	enumeration
Drag	0.0	real
Angular Drag	0.0	real

Now if you run the game, you'll find the wall most immovable, as a good wall should be. Place instances just outside the left and the right edges of the screen. If you have any trouble, you can place an instance on-screen, then click it and use the arrow keys to move it off-screen (SHIFT moves it faster). Now the panel can't go off-screen!

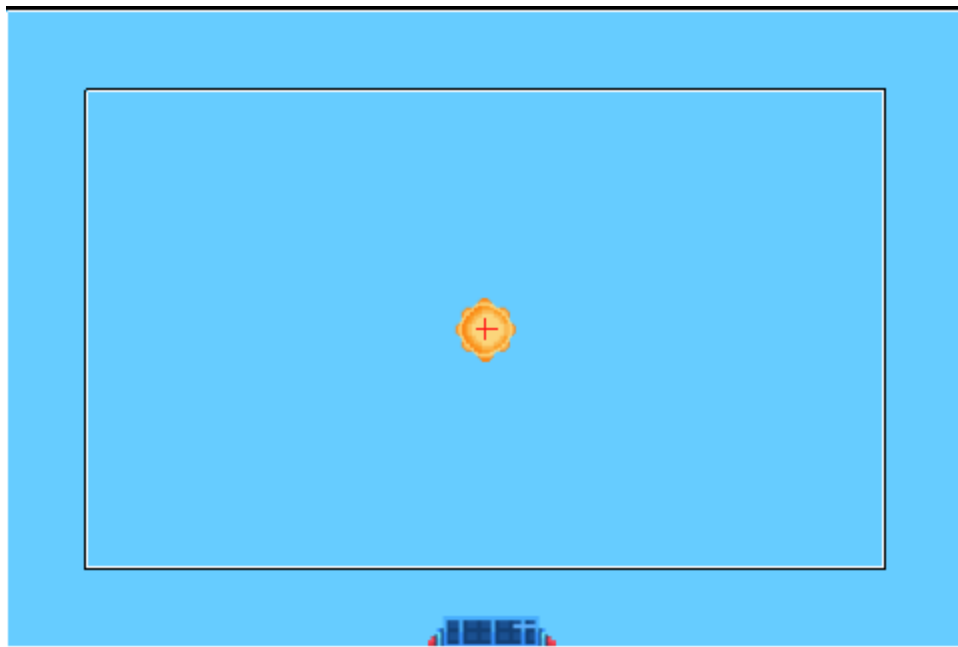
To prevent the panel from rotating, open up its own Physics Attributes, like we did for the wall, but this time check the "Fixed Rotation" box just above "Movable" (leave the latter checked).

We're done, right? Not quite. If we think ahead, we know we'll also want to prevent the solarball from going off-screen (except the bottom edge), and it moves in all directions, not just left and right. So, let's build walls around the entire left, top and right edges of the screen. Don't worry, you don't have to create a million instances of those tiny black boxes. Simply click on a wall instance, then click and drag the circles on its edges to stretch it. You should only need a single instance for each edge of the screen:



Step 8: The solarball.

Speaking of the ball, isn't it about time we add that? Yes, yes it is. Create an instance of the ball somewhere around the middle of the screen.



Right now it's more like the sun than a ball, just chilling (heating?) there in the sky. Let's make it move. Open up the ball prototype in the Inspector.

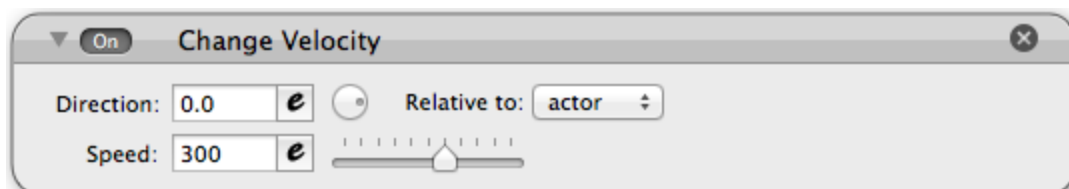
Up to this point we've used the Move behavior, which moves the Actor in a specific direction with a specific speed. We know the Collide behavior transfers force. So it would seem logical to Move the ball and set it to Collide with the walls and the panel, causing it to bounce off.

However, there is a problem with this. Remember what happened with the panel running into the (movable) wall. It pushed it right out of the way. Try Moving the ball downward and setting it to Collide with the panel. The same thing happens! The ball just pushes the panel out of the way. We can set the panel to be immovable, but...then we can't move the panel ourselves. And even if we do that, Move constantly tells the ball to go downward, while the Collide behavior tells it to go upward, causing the ball to get stuck when it hits the panel. Further thinking is required.

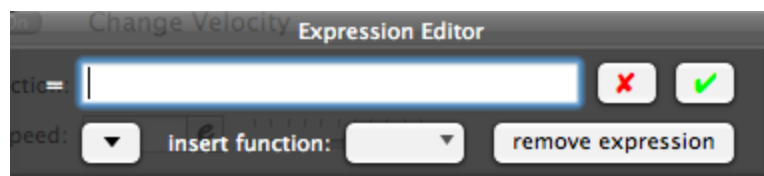
Additionally, if you've played the game on the website, you know that two things happen when the ball collides with the panel. One is that the ball's speed increases, and the other is that the ball bounces away in an unpredictable direction. True to life? Not exactly, but it does make for a more challenging game, which is the point.

What does this mean? It means that every time the ball bounces off the panel, it changes both its speed and its direction. "Move" just isn't going to cut it here. Instead, we'll use "Change Velocity."

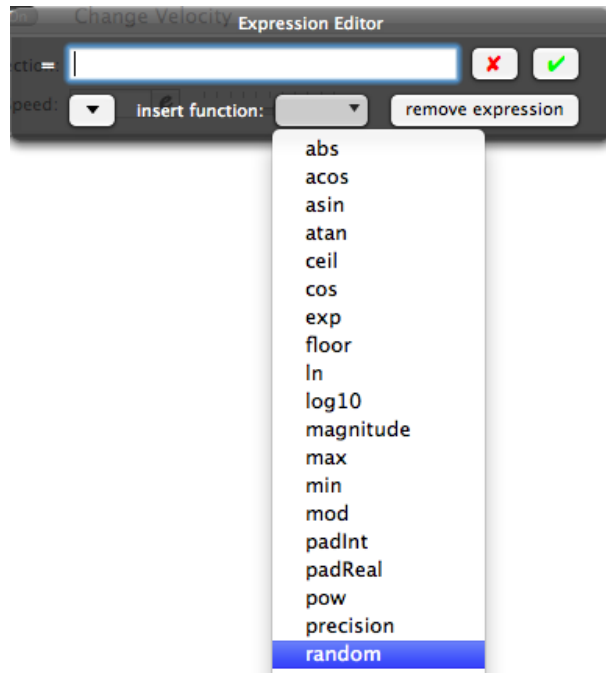
Drag it into the ball's Behavior. This will be the initial velocity:



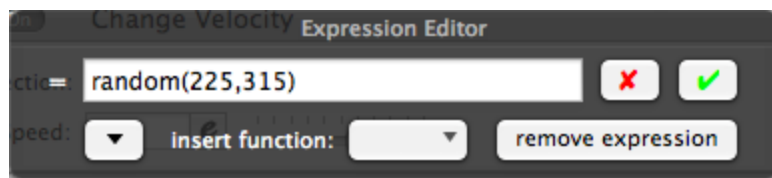
What for direction? Just to make things interesting, let's start the ball going in a random direction. It should still be downwards, though. Straight down is 270 degrees, so how about a range on either side of 270? To achieve this, click on the letter "e" to the right of the Direction field to open the Expression Editor:



Click the drop-down to the right of "insert function" and choose "random":



Replace “min” with 225 and “max” with 315. That’s 45 degrees on either side of 270, a good range.



Finally, click the green checkmark to save the expression into the Direction field. Now, if you run the game several times, you’ll see the ball chooses a random angle somewhere in that range.

OK, what about the speed? 300 is rather fast. We want to start out slowly, but more importantly than that, we are going to want to increase the speed every time the ball bounces off the panel. A set speed won’t do. We need to create our own Attribute.

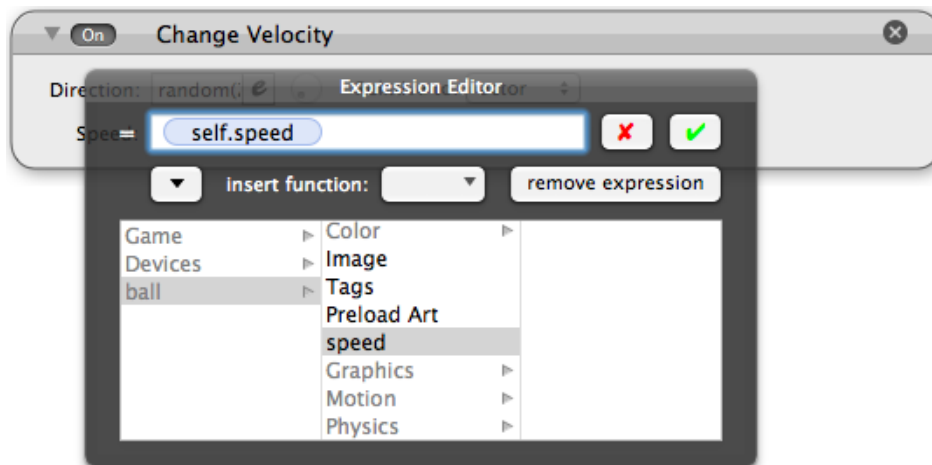
To do so, look in the lower-left of the list of the ball’s Attributes and click the plus button to create a new one. You’ll be asked to choose a type. This will be a number so choose “Integer.” The New Attribute will appear near the bottom of the list:

Preload Art	<input checked="" type="checkbox"/>	boolean
New Attribute	0.0	integer
► Graphics		attributes
+	-	

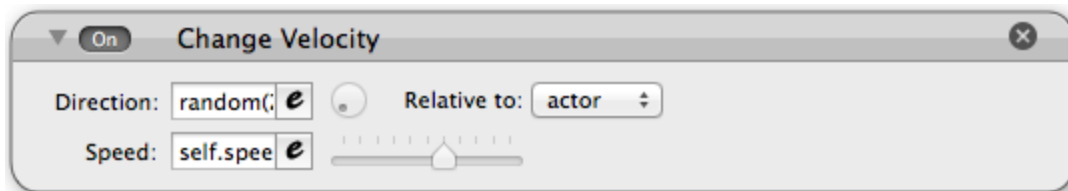
You can double-click on “New Attribute” to rename it something better, like “speed.” Also, double-click on the 0.0 to change the initial value. 100 is a good speed at which to start.

Preload Art	<input checked="" type="checkbox"/>	boolean
speed	100	integer
► Graphics		attributes

Finally, back in the ball’s Change Velocity behavior, open the Expression Editor for the speed. This time, click the arrowhead to the left of “insert function.” This opens the Attribute browser. We want to find the “speed” Attribute we just created. We made it in the ball’s Attributes, so choose “ball,” then find “speed” near the bottom. Double-click on it to select it:



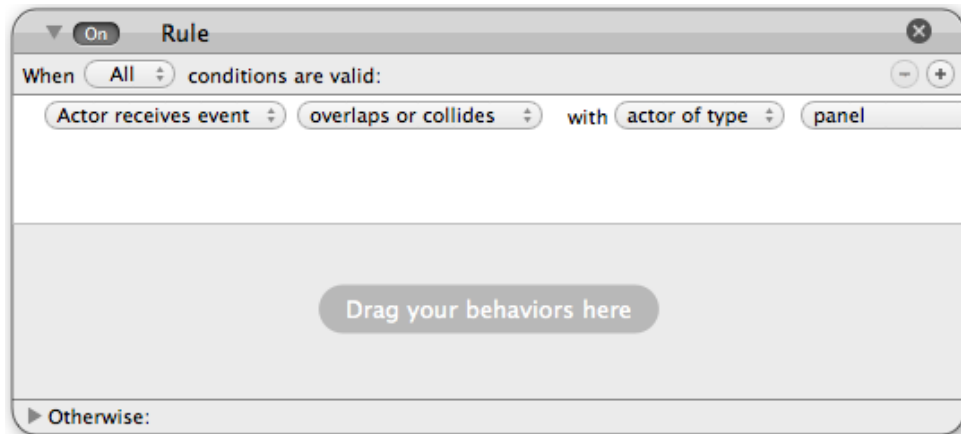
Finally, click the green checkmark. Now the Behavior should look like this:



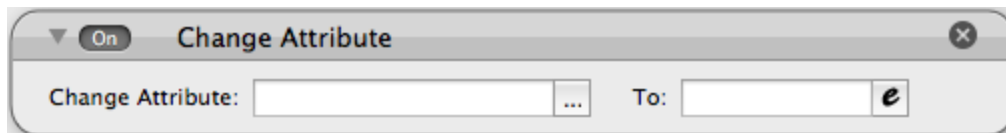
Step 9: Make the solarball bounce off the panel.

Whew! That was a lot of work! It will pay off though. If you run the game, you'll see the ball moves slowly in a random downward direction. So far so good. It doesn't collide with the panel though. So let's take care of that now.

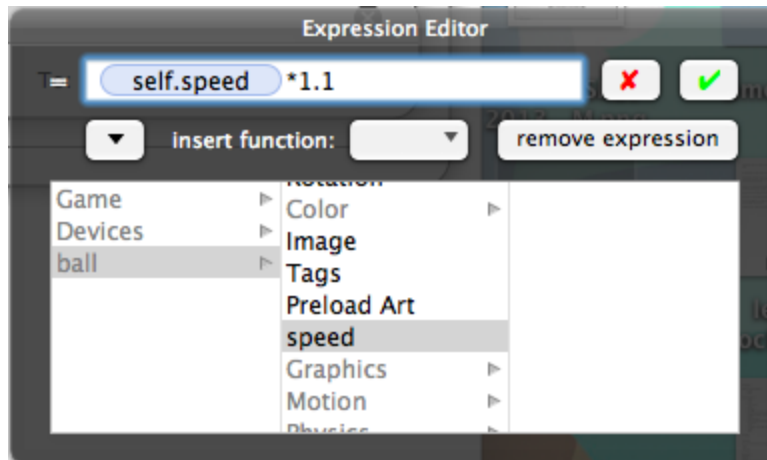
As mentioned before, we are not going to use Collide. So, we'll need to create a Rule for the solarball instead. Do so, then set the parameters to be "overlaps or collides" with the panel:



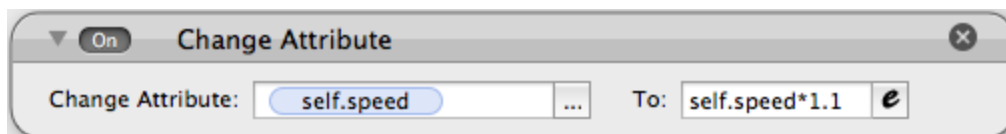
First of all, let's increase the speed. We can do that by multiplying our speed Attribute by some number greater than 1. Drag the "Change Attribute" behavior into our new Rule:



Click the triple-dot to the right of the "Change Attribute" field to open the same Attribute browser as before, then double-click on "ball"-->"speed." Then, open the Expression Editor to the right of the "To" field, click the arrowhead in the lower-left, and select the speed Attribute once more. This time though, after you double-click it, click to the right of it in the field and type " *1.1. " (the asterisk character followed by "1.1"). The result looks like this:



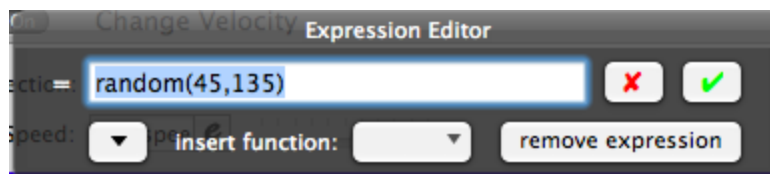
Finally, click the green checkmark. The Behavior now looks like this:



That was pretty elaborate but it simply multiplies the current value of “speed” by 1.1 and makes the result the new value of “speed.” So, everytime the ball collides with the panel, “speed” grows.

There's one more thing--we need to change the velocity! Drag the Change Velocity behavior into the Rule, beneath the Change Attribute we just set up. For the speed, open the Expression Editor and select our “speed” Attribute (which we just increased) using the Attribute browser, as per usual.

For the direction, open up the Expression Editor and select the “random” function again, but this time we want the ball to go upward, so we need a range on either side of straight up, which is 90 degrees. Try 45 for the min and 135 for the max (again, 45 degrees on either side of the straight direction).

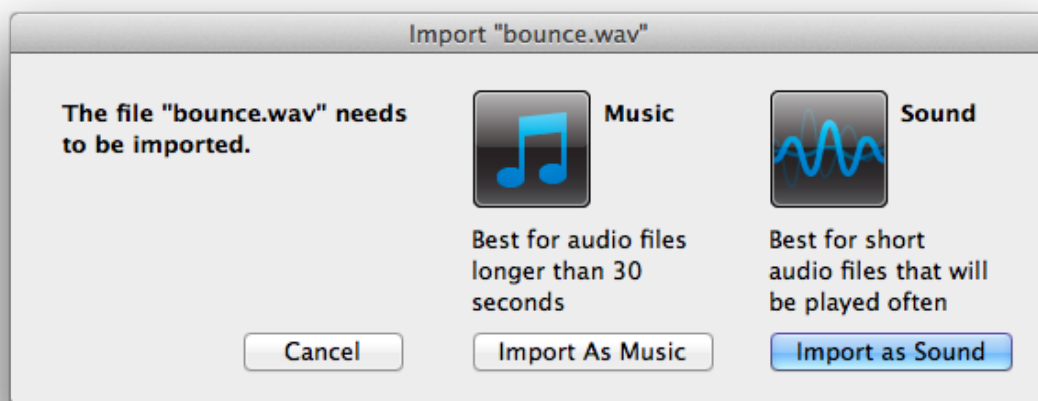


Just to be on the safe side, open up the ball's Physics Attributes and set it to Fixed Rotation, so the ball doesn't spin and potentially screw up the directional changes.

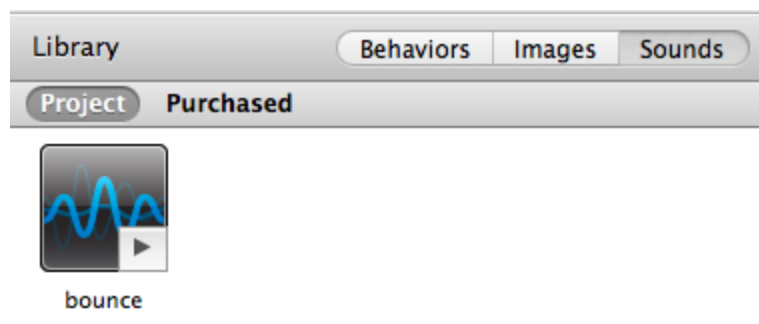
Run the game. If everything is correct, the ball should now bounce off the panel! Nice!

Step 10: Add sound, and collide with the walls.

Take a deep breath and pat yourself on the back. All the hard work is out of the way. To reward ourselves, let's add a sound effect for when the ball bounces off the panel. This is really easy. In the Library, click on the *Sounds* tab. Now, look in the Resource Files folder for a subfolder called "sounds." There should be one called "bounce.wav." Drag it into *Sounds* in GameSalad. A pop-up menu should appear asking how you want to import the audio file:

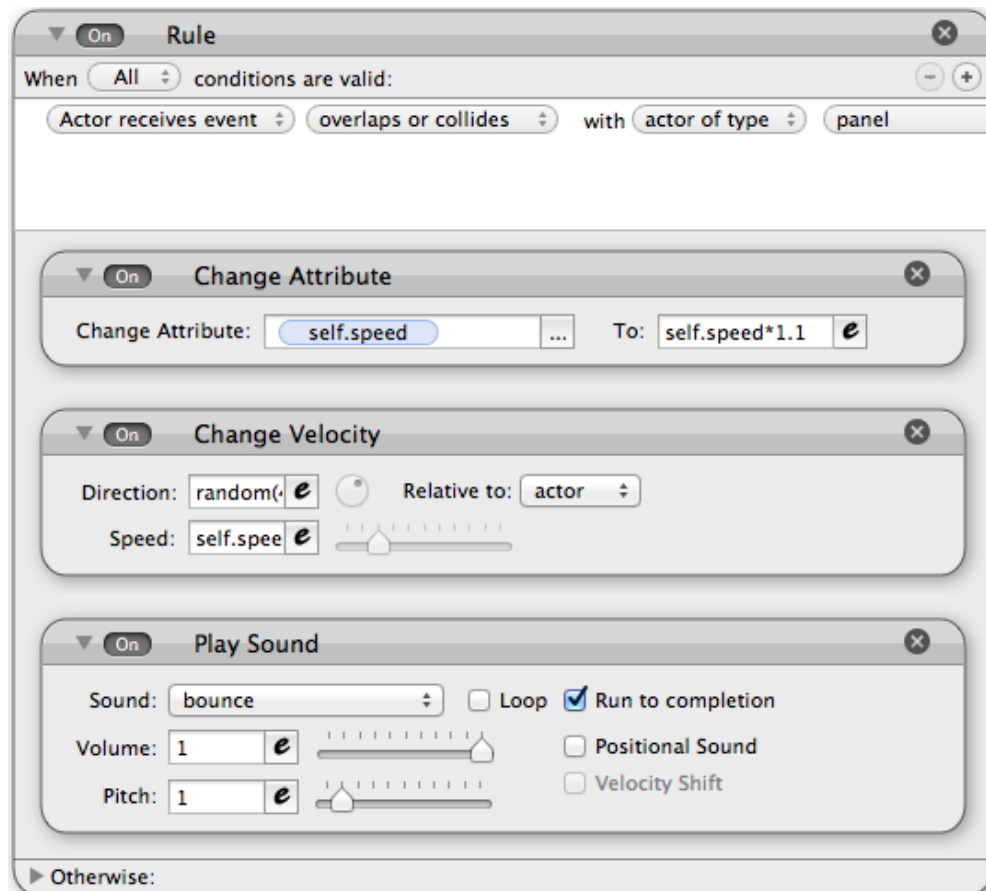


This is a sound effect, so click "Import as Sound." GameSalad will import the file and the *Sounds* tab should now look like this:

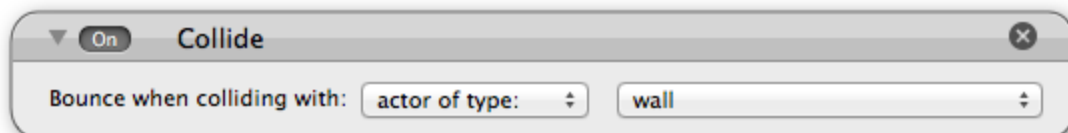


Finally, click on the *Behaviors* tab again, and (inside the ball prototype's Behavior) drag the "Play Sound" behavior into the Rule for collision with the panel. You can place it wherever you like, before or after the Change Attribute and Change Velocity behaviors. Play Sound has a

lot of parameters that can be adjusted, but the only one we need to change is which sound to play, which should be the “bounce” that we just imported. The complete collision rule should now look like this:



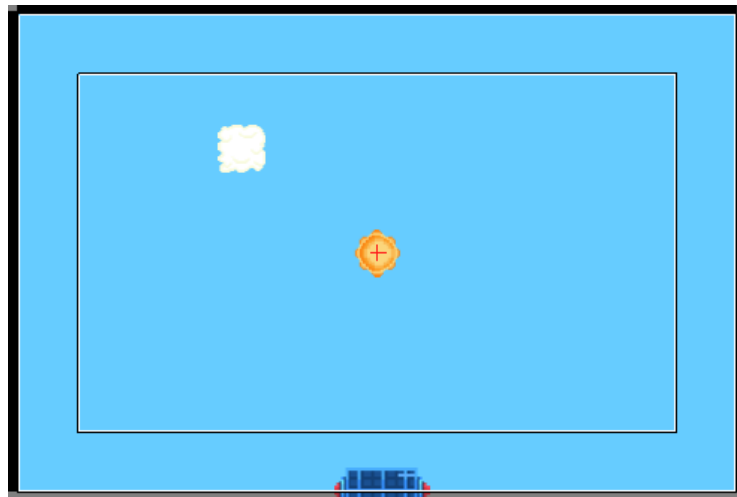
As a final touch (unrelated to sound), we will use the Collide behavior for when the ball collides with the wall, since the wall is immovable. Drag Collide into the ball’s Behavior (independent of any other Rules and Behaviors we already applied), and set the Actor type to the wall:



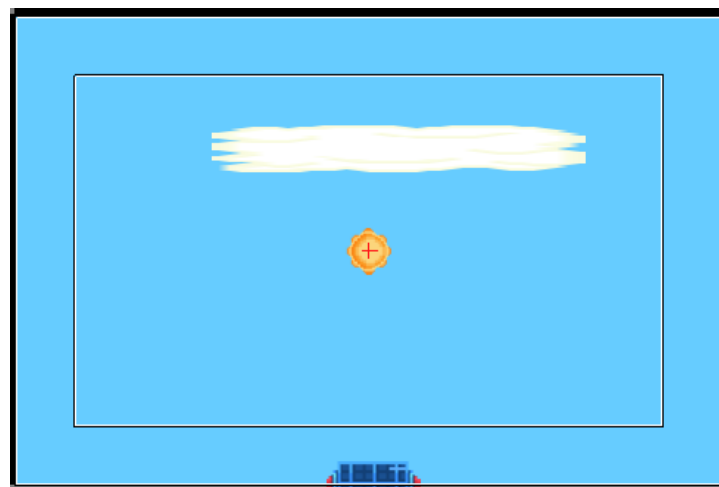
Step 11: The clouds.

The ball now bounces off both walls and panel, picking up speed and randomly changing direction (and playing a pleasing sound effect) when it bounces off the panel. You can play the game like this!

It's not very interesting, though. Yes, it does get quite challenging if you play it long enough, but there's no unpredictability other than when the ball hits the panel, and the field of play is just an empty space. It could use some obstacles. Let's add clouds. The ball will bounce off them (these are very dense rain clouds!). Drag the cloud Actor prototype into the Scene to create an instance. Place it anywhere you like.

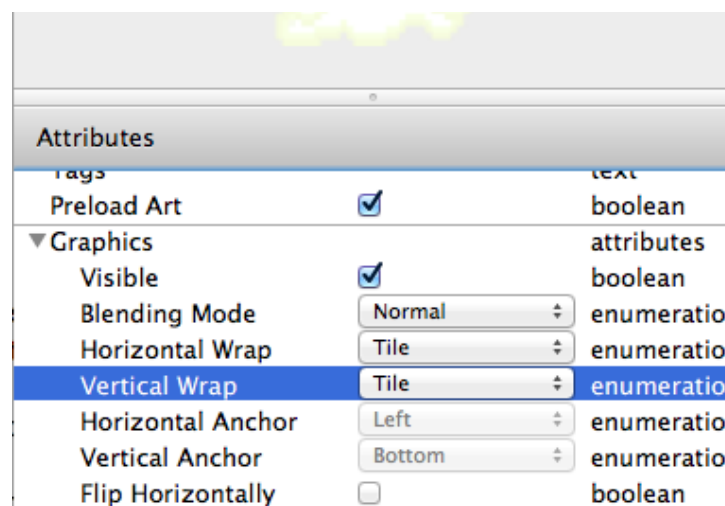


That cloud is small enough that it may be difficult to hit it with the ball. You can create more cloud instances and put them next to each other to make a “long” cloud, or you can take advantage of “tiling.” What does that mean? Try selecting the cloud instance and then stretching it (as before, by dragging the circles on the side):

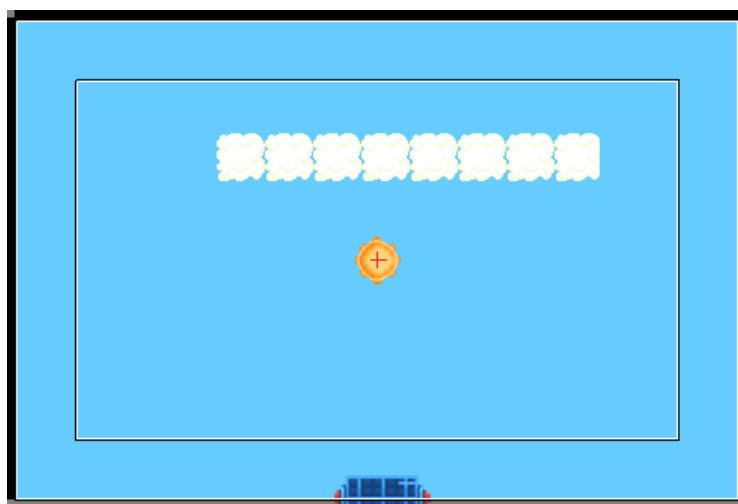


Well, it sure is stretched, isn't it? It doesn't look very good though. That's because we've taken an image that was a certain size and stretched it horizontally without proportionately stretching it vertically, so it ends up distorted (this wasn't a problem with the wall because its image is just a solid black square).

However, remember that the cloud image is attached to the cloud Actor. They are not the same thing. We've stretched the cloud Actor, and by default the image stretches to fit the Actor's new dimensions, but it doesn't have to! Double-click on the cloud Actor prototype in the Inspector. Scroll through its Attributes and open up the "Graphics" category. Look for "Horizontal Wrap" and "Vertical Wrap." These control how the image covers the Actor. By default they are set to "Stretch." Change them to "Tile."



While we're here, open up the Physics Attributes and uncheck "Movable." As with the wall, we want the ball to collide with the cloud without moving the cloud. Now, return to the Scene view:



Cool! If you click on it, you'll see it's still just a single cloud Actor, but now the image is *tiled*, meaning it's copied at the original size as many times as needed to fill the bounds of the Actor. Much better looking! Note that this does mean you can end up with part of an image:



That's because the image ignores whether it has enough space left to complete another copy or not. So, watch out for that. Still, it beats placing multiple instances side-by-side. Tiling is great.

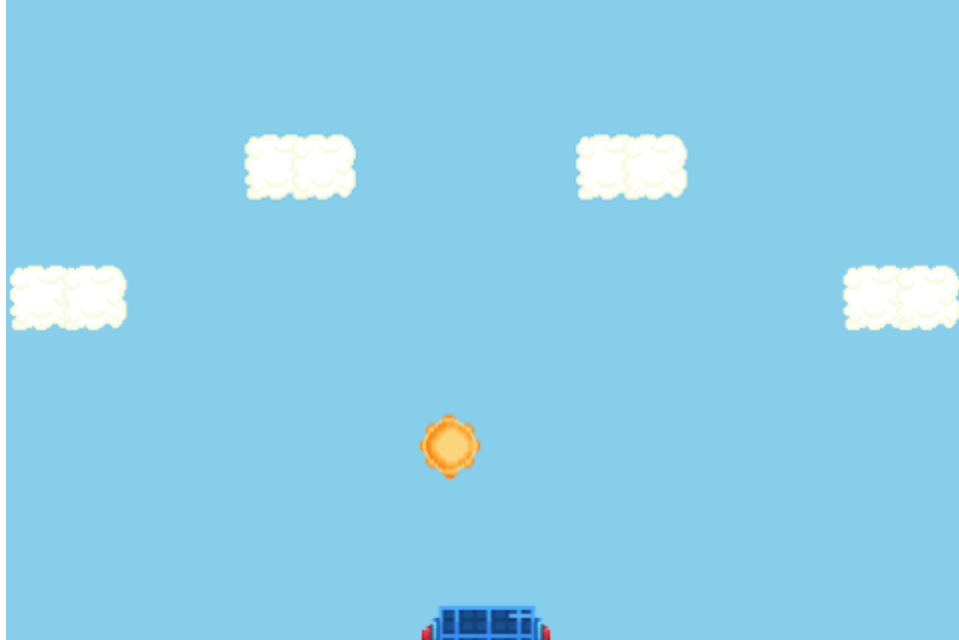
Finally, open up the ball's Behavior once more and add another Collide behavior, this time set to the cloud Actor. Now, the ball should bounce off the cloud!

Step 12: Make a level.

We are almost done! Now it's time to design a proper level, i.e. the playfield. Even though the clouds function identically to the walls, you can choose where to place the clouds. By placing them in various places and sizes on the playfield you can radically change the flow of play.

For instance, that long cloud we created above is too long. If the ball gets into the space between the cloud and the top of the screen, it will bounce around up there on its own for a long time, requiring no assistance from the player, which will quickly become boring.

Furthermore, because of the way the default physics work in GameSalad, the ball may get stuck in that space, bouncing vertically forever without enough horizontal speed to escape. We could tweak things to change that, but it probably makes more sense in this case simply to use shorter clouds. For instance, the layout from the website version looks like this:



There is enough space between the clouds that the ball has a good chance of getting through them to the top, but an equally good chance of bouncing off their underside and requiring a quick response from the player. The clouds are long enough that the ball probably will bounce off their top but short enough that it's unlikely to get stuck there.

As with other games in this series, this is one that would benefit from a larger screen size, which unfortunately is not currently an option for GameSalad Arcade. Still, you can be creative even within this limited space. Make your own level layout and test it to make sure it's fun to play. Use as many or as few clouds as you like!

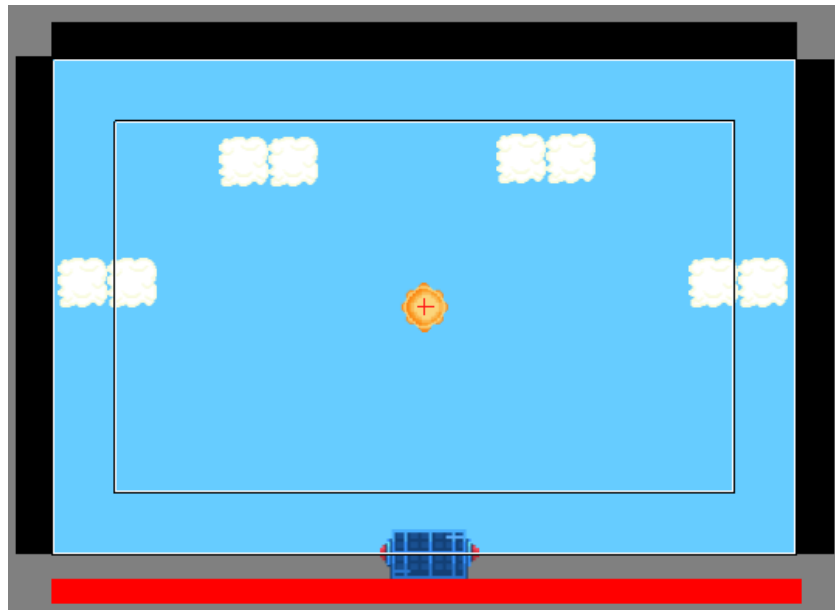
Step 13: Losing the game.

We're missing one final and essential component of our game. There's no way to lose!

Well, technically there is. If the player misses the ball and it goes off the bottom of the screen, that's it. The ball is gone and play comes to a halt. But it's not very satisfying because the game just sits there. There's no message to say the player lost, and no restart of the game. We should add those things.

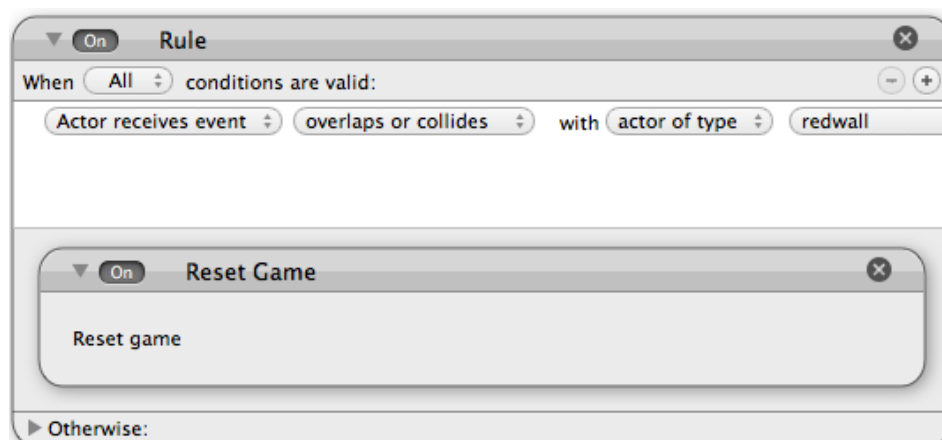
First of all, we need a way to determine when the player has lost, which means notifying the game that the ball has gone off the bottom of the screen. We used the wall to prevent the ball from going off the sides and top; similarly, we'll use another type of wall, and the ball's collision with it, to indicate the game is over. Create an instance of the "redwall" Actor

prototype, place it underneath the bottom of the game screen, and stretch it to the width of the screen:



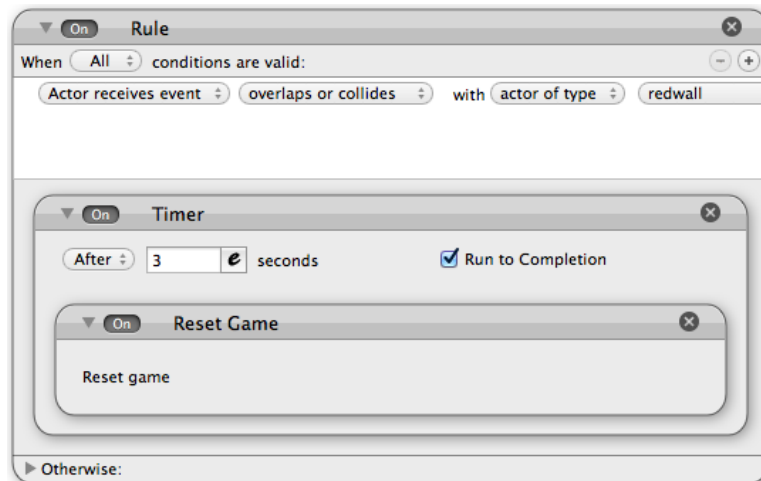
You could put it directly under the screen edge, but leaving a little space between the edge and the red wall lets the ball get off-screen before it triggers Game Over, which just looks nicer. Also be sure to open up the red wall's Physics Attributes and uncheck "Movable," as per usual. It doesn't really matter in this case since game play will stop as soon as the ball hits the red wall, but it's a good habit to get into for Actors that shouldn't move.

Now, open up the ball's Behavior once more. We don't want the ball to bounce off the red wall, so we won't use Collide or Change Velocity. We do care if the ball collides with the red wall though, so create a new Rule for the ball and have it check for "overlaps or collides" with the redwall Actor. Then, drag the Reset Game behavior into the Rule:



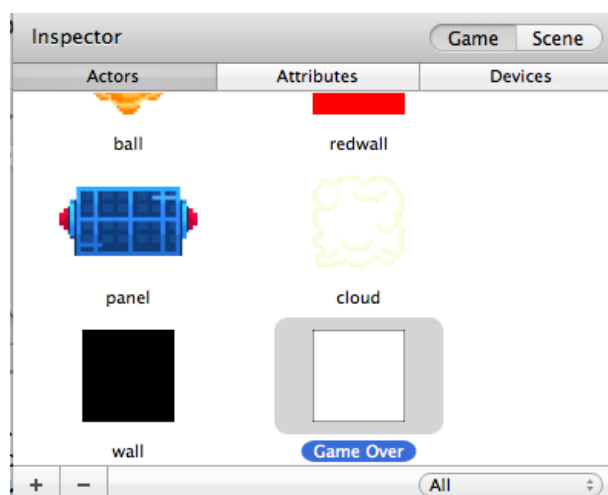
Run the game and let the ball fall off the bottom of the screen. The game should restart.

That's fine, but it restarts right away. We could add a Timer. Drag the Timer behavior into the Rule, change "Every" to "After" and change the number of seconds if you like, and move our Reset Game behavior into the Timer. Make sure to check the "Run to Completion" box, which ensures the timer will finish counting even if the ball is no longer colliding with the red wall:

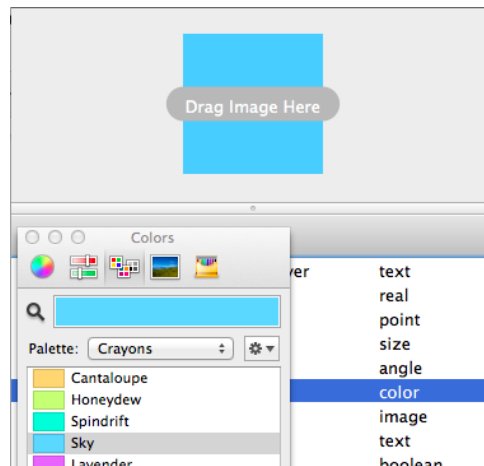


That's better. There's now a pause before the game restarts. But we also want to tell the player the game is over. It would be nice to display some text to that effect. The thing about displaying text is that, like anything else in GameSalad, the text is attached to an Actor. So, we can't display the text in the ball's Behavior, because then it would move with the ball.

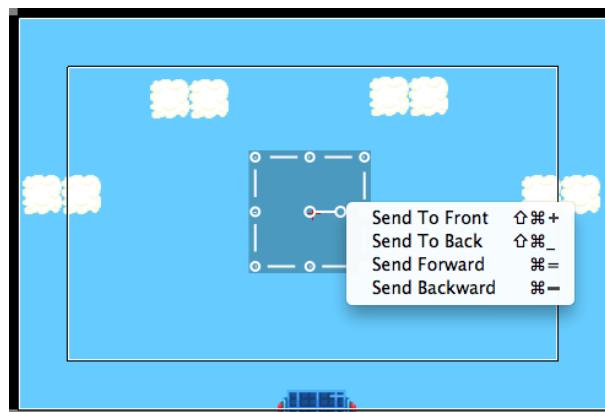
Let's create a new Actor to display the text. Return to the Scene View, and in the Inspector click the plus button in the lower-left to create a new Actor, with no image attached. It's just a blank white square. Click on the name (probably "Actor 1") and change it to "Game Over" or the like.



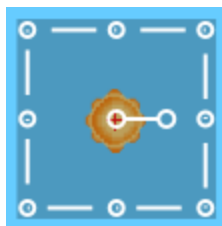
We'll have to place this in the Scene, and we want it to blend in, so double-click on it and then in its Attributes change its color to match the background.



Finally, return to the Scene view and drag the new Actor into the Scene to create an instance. You'll notice immediately that, thanks to our color change, it's invisible. That's good. What's not good is that it covers over any other Actors, like the solarball. That's because by default GameSalad displays images in the order we added them to the Scene. We can easily change this though. Click on the instance to select it, then right-click on it and choose "Send To Back":

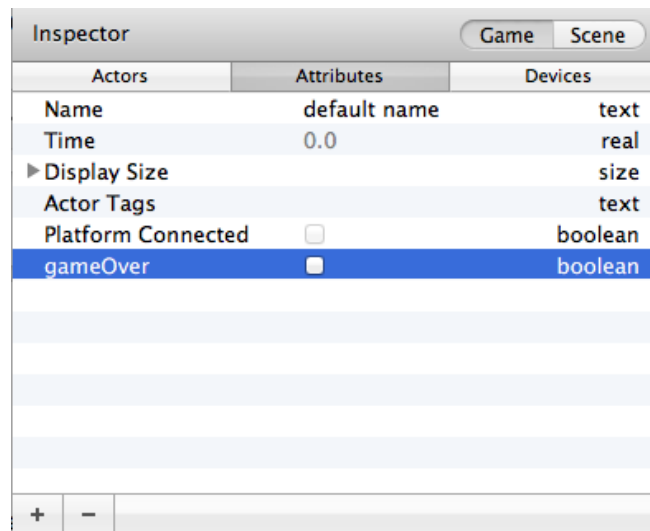


Now it will appear behind everything else (you can see the solarball on top):

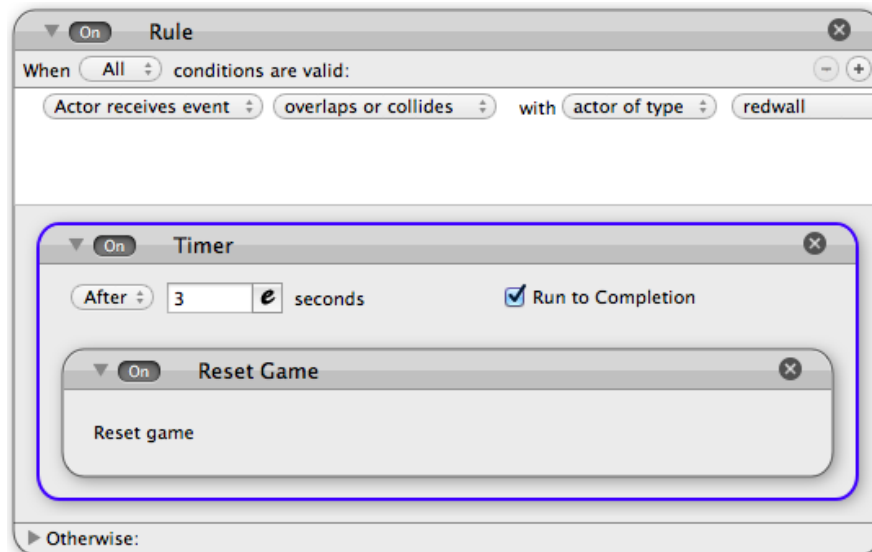


Now that we have the Actor in place, we can make it display the text, but to do so we need to tell it when the game is over. It doesn't have access to the collision information from the ball, so we'll create a new Attribute, called a "boolean," which can be set to either true or false, and use that.

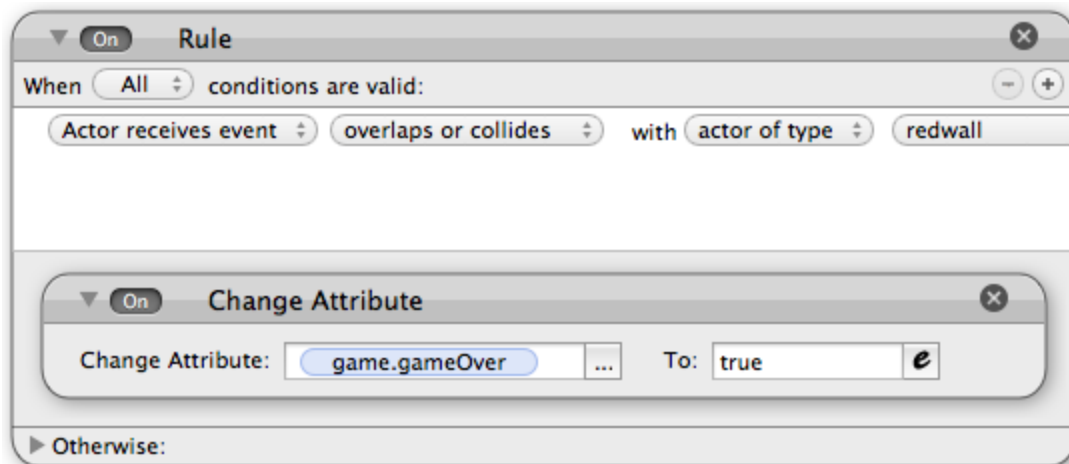
In the Inspector, click the *Attributes* tab, then click the plus in the lower-left to make a new Attribute. Choose "boolean" as the type, then rename the result "gameOver" or something along those lines and make sure the checkbox is unchecked (this means it starts set to "false"):



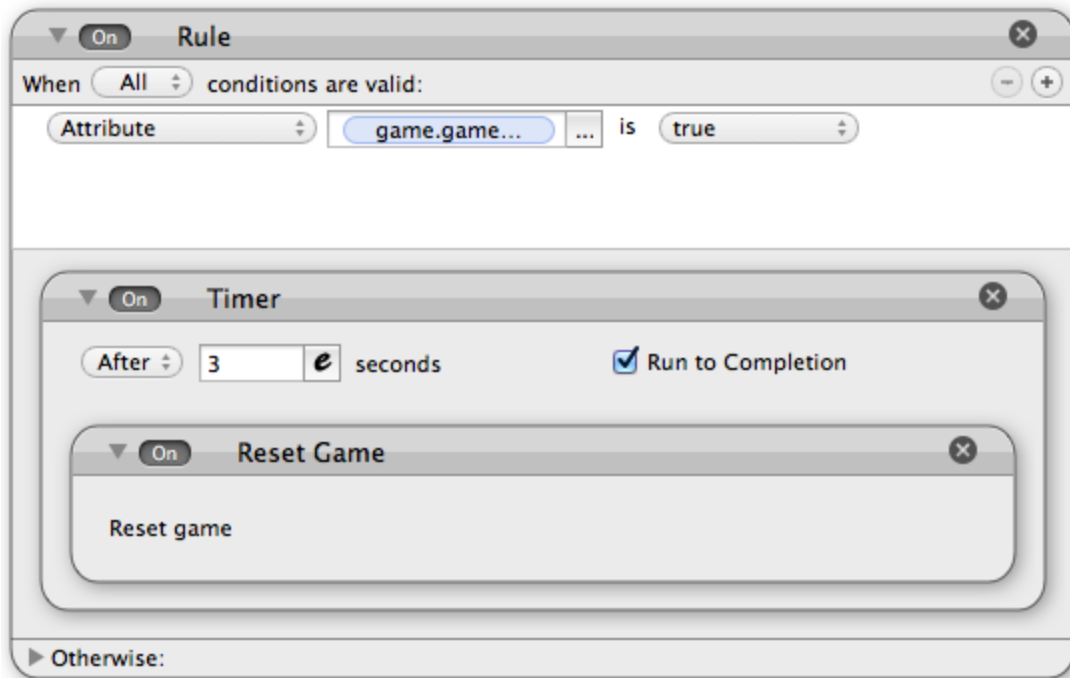
Now, click on the *Actors* tab, then open up the ball's Behavior and return to the Rule regarding collision with the red wall. Instead of running the Timer and restarting the game, we'll set gameOver to "true," then run the Timer and restart in the Game Over Actor instead. So, we'll do several things here. First, click the Timer inside the Rule to select it. A blue border will appear:



Choose “Edit,” then “Cut.” This will remove the Timer from this Rule and save it so we can apply it to the other Actor later. Then, drag the “Change Attribute” behavior into the now-empty Rule. Choose “Game”-->”gameOver” for the Attribute and type “true” in the “To” field:

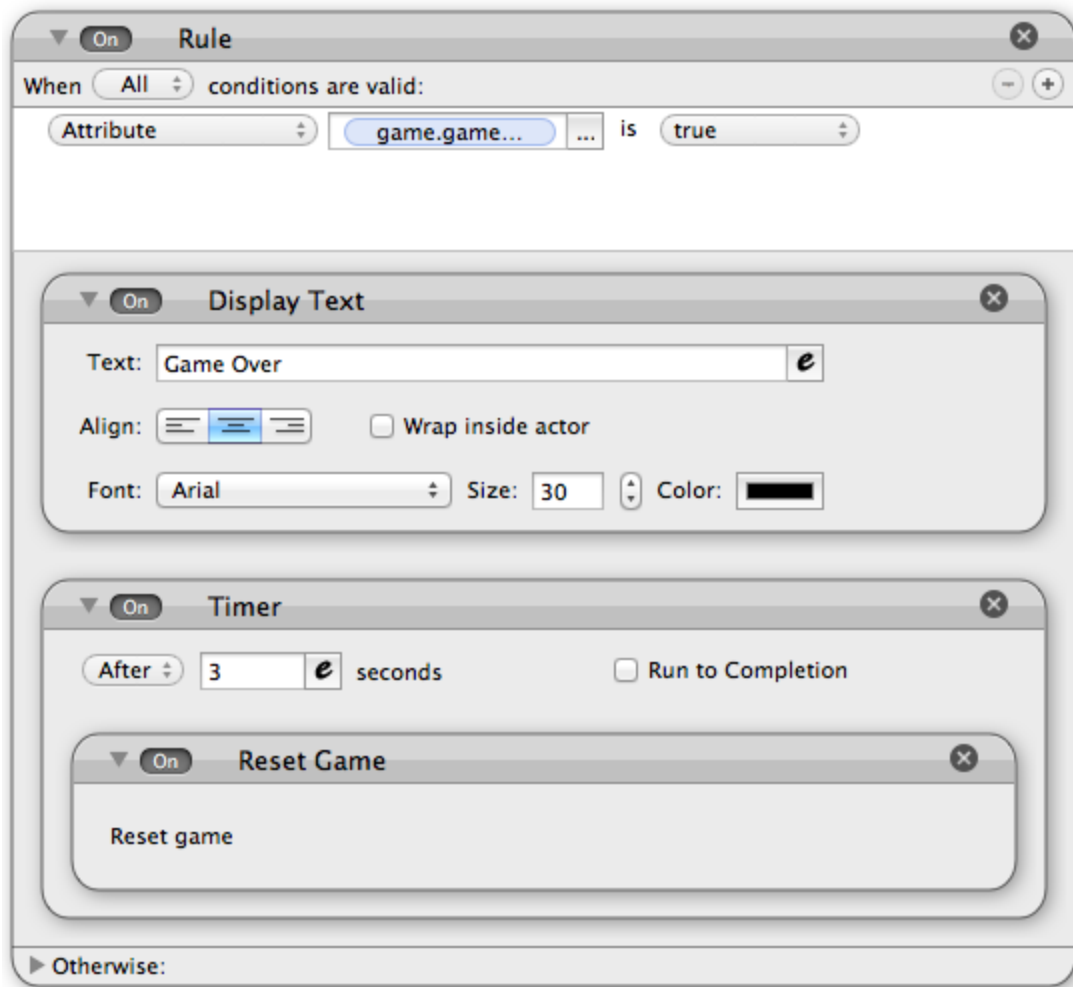


Now, return to the Scene view, then open up the Behavior for the Game Over Actor. Create a new Rule. This time though, change “Actor receives event” to “Attribute.” Once again, choose “Game”-->”gameOver” as the Attribute, and make sure “true” is selected in the drop-down. Now, click inside the Rule, then choose “Edit” then “Paste.” The Timer with its embedded Reset Game behavior should appear!



Incidentally we can now uncheck “Run to Completion,” since gameOver will stay set to true.

If we run the game, it should work the same as before. Indeed, up to this point we haven’t gained anything by changing what Actor resets the game. However, now that we have a separate Actor handling the Game Over event, one who doesn’t move, we can do one more thing, which is to display text. Drag the Display Text behavior into the Rule and place it above the Timer. Change the text to “Game Over” or whatever you like. Note that the default text color is white, which could be hard to see, so change it to black or whatever other color you prefer. The whole Rule (which makes up the entirety of the Game Over Actor’s behavior) should now look like this:



Now, run the game and let the ball past the panel. The game displays text before it resets!



Victory! What now?

Hey, that's it! Great job! You've made Solarball!

Whew, that was a lot of work, huh? Especially for a game that seems so simple. However, we've actually used an impressive number of advanced techniques. Things like sound effects, increasing speed, randomly-determined angles, multiple types of collision, invisible and off-screen objects, manipulation of physics properties, booleans, special Actors to control text...you are definitely ready for the Intermediate-level tutorials!

What now? Playtesting is an important part of game design, so you should ask people to try the game out, to see how you can improve your level layout. You don't want the play session to be too short but you also don't want it to be too long (when the game might become boring).

Think about how you might improve the game further. Many games like this use a scoring system that gives the player points for keeping the ball in play and making especially tricky shots. You could use a clock to track how long the play session lasts, too (that is a kind of score). Perhaps the player could destroy the clouds by hitting them enough times. Perhaps there are additional "screens" or levels the player could reach by playing well.

With this tutorial you've learned the basics (and some of the advanced techniques) of game creation in GameSalad. Feel free to build this game further, work through the other tutorials in the curriculum, and start your own projects. The possibilities are limitless!