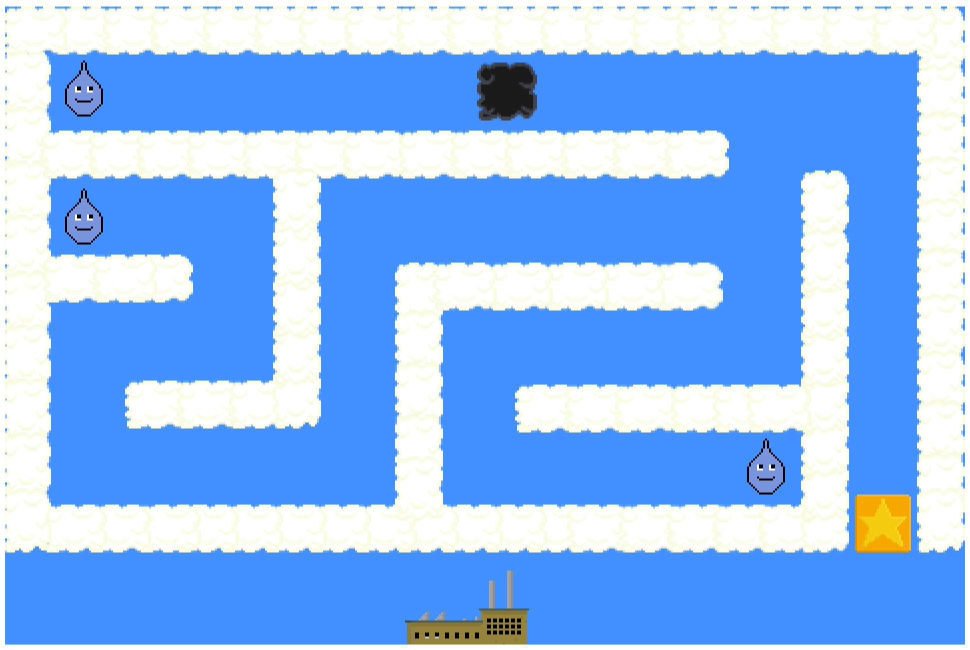
How to Make “Smog Cloud Madness” in GameSalad

by J. Matthew Griffis

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



In Smog Cloud Madness, you control a group of raindrops as they try to escape the clouds while avoiding the smog from the factories below. The game is simple but provides an introduction to designing and transitioning between different “levels” of gameplay. Level design is one of the most important skills for any game designer!

In this tutorial, you’ll learn how to recreate Smog Cloud Madness. 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.

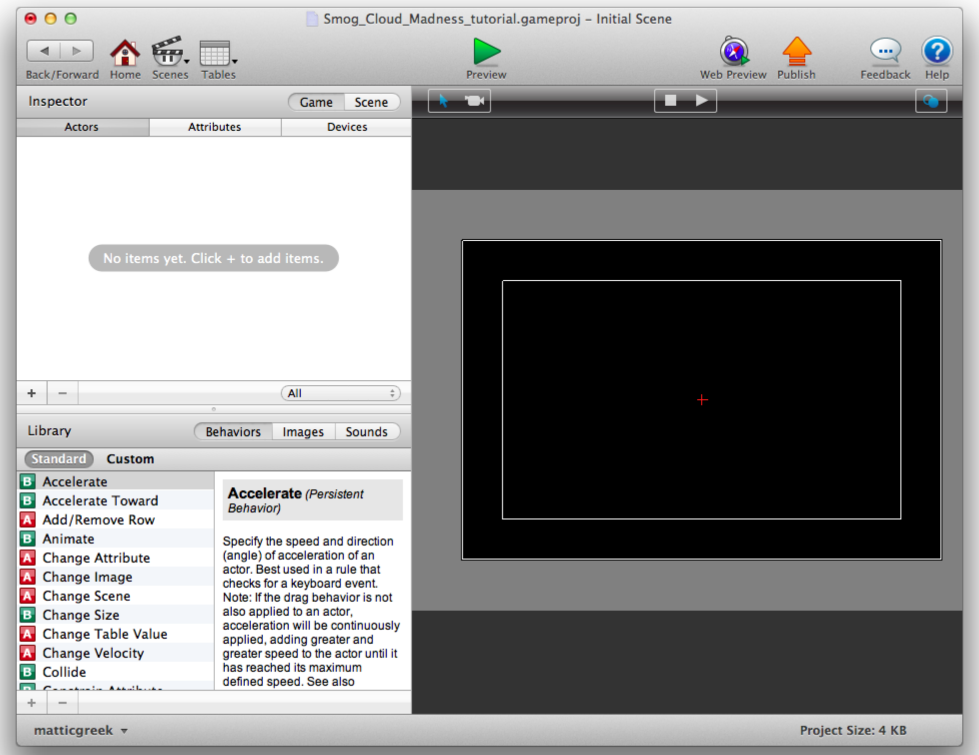
Create a blank project and fill in the Project Info, making sure to select “GameSalad Arcade” as the Platform if you want to publish the game online. If you don’t, feel free to select another

Platform but be advised that your screen may be a different size than the one in this tutorial.

Save your work. Do this often. Saving is your friend.

Alright, let's make some stuff and put it in our game! This will be a game with three *levels*, or scenarios for the player to play through. GameSalad divides its game settings into "Scenes," so it makes the most sense to create a Scene for each level.

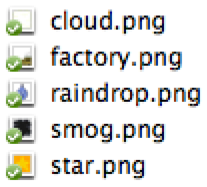
For now, we'll work on the first Scene. Click the *Scenes* tab, then double-click on "Initial Scene" to see this:



Accelerate *(Persistent Behavior)*

Specify the speed and direction (angle) of acceleration of an actor. Best used in a rule that checks for a keyboard event. Note: If the drag behavior is not also applied to an actor, acceleration will be continuously applied, adding greater and greater speed to the actor until it has reached its maximum defined speed. See also

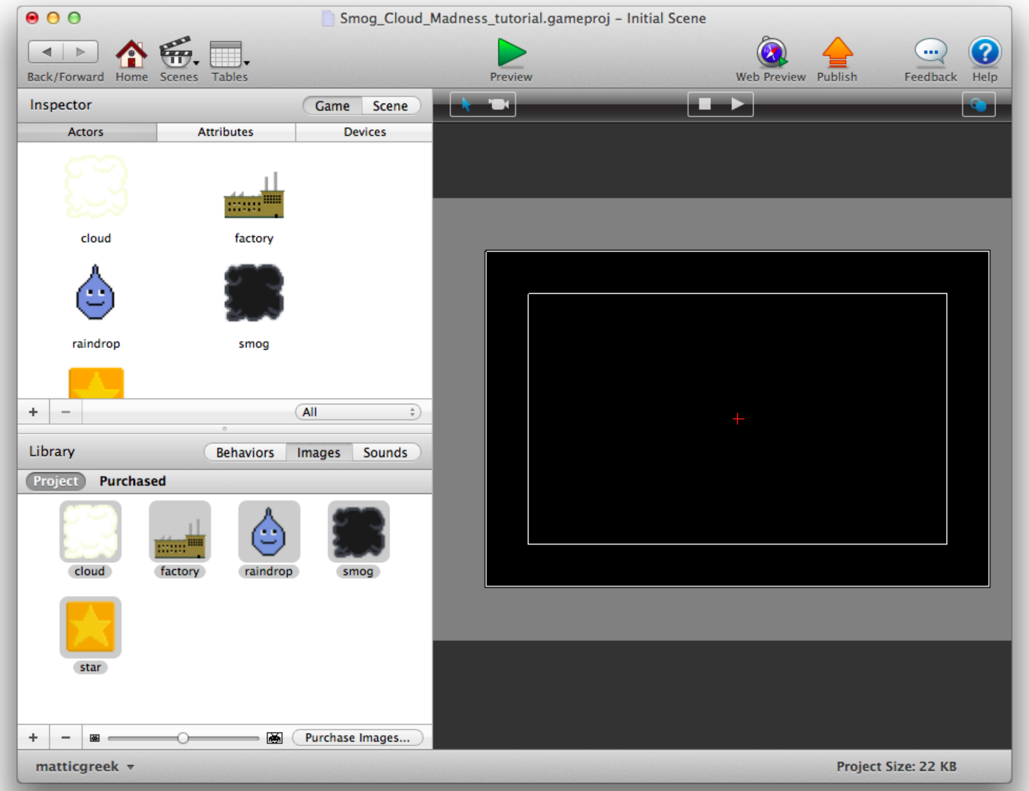
An empty stage, waiting for us to transform it into an exciting show. Let's start with hiring...sorry, *creating* some Actors, the elements that will make up the game environment and characters. To do so, look at the Library in the lower-left of the interface and click on the *Images* tab. Now open the Resource Files folder you downloaded and open the Sprites folder. You'll find five images:



We are going to need all of these, so go ahead and drag all five into the *Images* box. GameSalad will import the sprites. Finally, drag all the images from the Library into the *Actors* box just above in the Inspector. Poof! Five brand spanking new Actors with images, ready to rock and roll.

What have we just done? We've imported images into GameSalad so we can use them for our game. Then we used those images to create Actors, which are the actual elements that make up a game in GameSalad. An image can't be used in a game without being attached to an Actor, and an Actor without an image is just a blank white box.

Your screen should now look something like this:

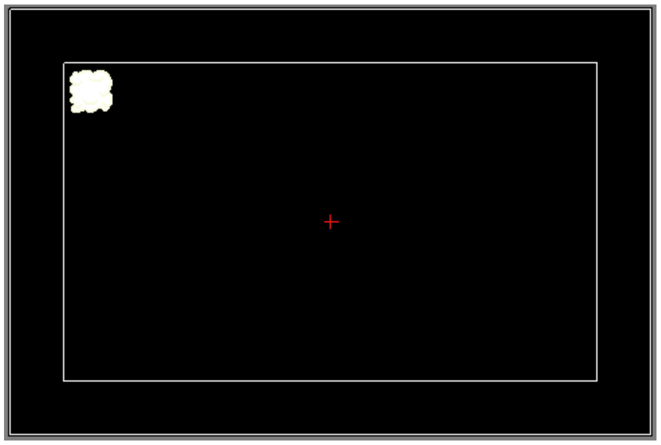


Go ahead and click the *Behaviors* tab in the Library; we won't be needing to add any further

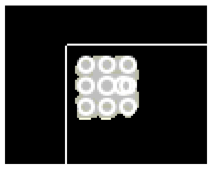
Images, but we will be needing to add *so many* Behaviors. Oh yes.

First things first. We need to build a *level*. Generally speaking a level is an environment or specific scenario that the player must successfully navigate/overcome. It has a definite start point and a definite end point. What constitutes a level is pretty flexible and varies from game to game.

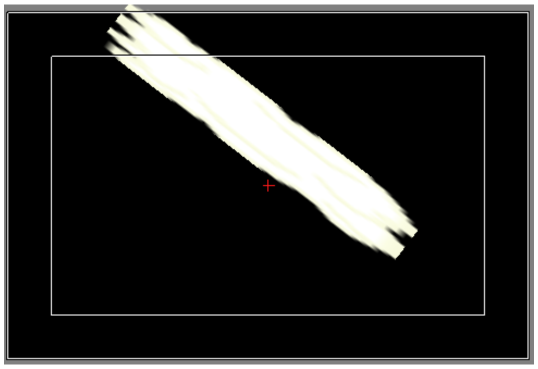
In this game we move the raindrops through a maze made of clouds. The raindrops start in a specific place and must reach the star at the end of the maze. In this game, then, a level is a single maze, with the star representing escape. So we need to create the paths that make up the maze. Drag the cloud Actor into the Scene on the right (remember, the black rectangle is the game's visible area):



Alright! We've created an *instance* of the cloud Actor. Instances are what actually appear in the game. Click on the instance:



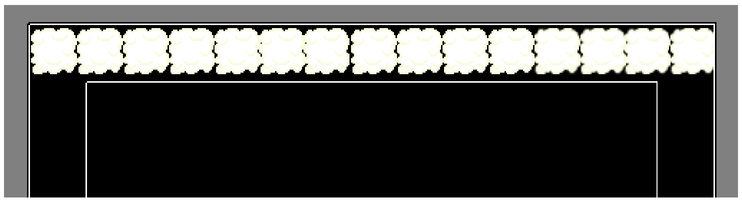
The instance is now *selected*. The circles are points we can click and and drag to manipulate the cloud in various ways. Drag from the center circle to move the cloud. Drag from any of the circles on the outside to change the shape of the cloud. If you make the cloud bigger, you can see another circle in the center that allows you to rotate the instance. Try it now:



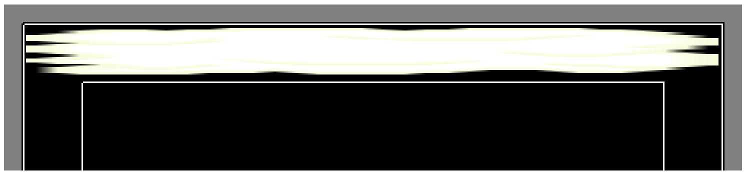
This is a good time to mention that if you don't like a change you've made, you can undo it by choosing "Undo" from the Edit menu. You can also click an Instance and press DELETE to remove it. See **GameSalad Basics** for more tips on advanced use.

We want to prevent the raindrops from moving offscreen, so the first thing we should do is create a boundary of clouds around the edges of the screen. We will make it so that the raindrops stop moving if they collide with a cloud.

Build a ceiling of clouds across the top of the screen. How you do this is up to you. You could create many cloud instances and carefully place them:



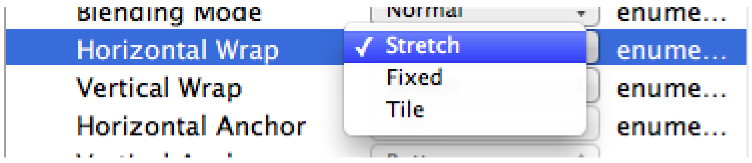
Or you could create one cloud and stretch it to fill the screen:



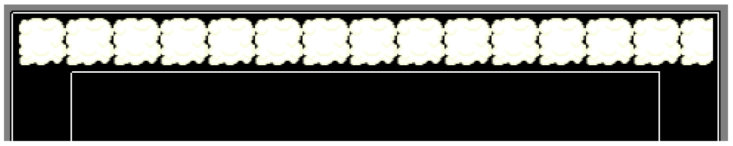
Either of these will work and they are both valid. The first takes a long time to create but looks very nice, while the second looks bad but takes only a moment to generate.

This illustrates well a general principle of design: **take only the time you need to achieve your goal, but remember that the quality of the final result depends on the time and effort you put in**. If you're making a game for release, you'd probably want to make the first version of the cloud ceiling, but if you're testing something for yourself, don't waste time on polish you don't need: use the quick and dirty single-cloud method. This tutorial uses the latter method, but you can play the game on the website to see how much of a visual difference it makes to put in the time. Feel free to create your own version however you like; there is no right or wrong answer.

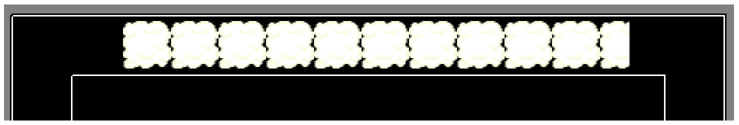
Side note--there is one more approach, which strikes a nice balance between the fast-and-ugly and slow-and-pretty methods. You may wonder why *it looks so bad when we stretch a single cloud. The reason is that the cloud image is a certain size, and when we stretch the cloud Actor, to be a different size, the image gets stretched too. But remember that the Actor is not the* same *as the image. The image is a "cover" for the Actor, nothing more. And that means we have options for* how *the image covers the Actor. Double-click on the cloud prototype again and scroll down through the list of its Attributes. At the very bottom, you'll see "Graphics," "Motion" and "Physics." Click the arrow to the left of "Graphics" to expand that category, and scroll down until you come to "Horizontal Wrap" and "Vertical Wrap." These control how the image covers the Actor. By default, they are set to "Stretch," which means a single copy of the image stretches to fill the full dimensions of the Actor. That's fine if image and Actor are the same size. But if you click on "Stretch," you get other options:*



Change both Horizontal Wrap and Vertical Wrap to "Tile." Click the Back button to return to the Scene view and...

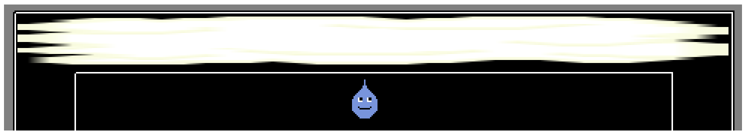


Amazing! The image keeps its size but repeats as needed in order to cover the single Actor fully. Tiling is a great technique so don't be afraid to use it. It does have its downsides, though:

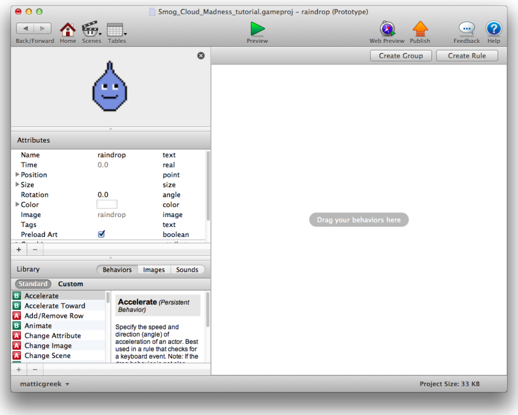


As you can see, the rightmost cloud is cut off and looks rather awkward. There was enough room left on the Actor to start another copy of the image but not enough to finish it. Also because the spacing between the "tiled" images is set, it doesn't look quite as much like a *continuous cloud. Compare that to the website version. On the whole, tiling is a great if imperfect technique.*

We'll need to build much more than just the cloud ceiling to finish a level, but right now let's set up the interaction between the raindrop and the cloud. That means we need a raindrop. Drag the raindrop Actor into the Scene:



If you Preview the game right now (by clicking the large green arrow at the top of the interface), nothing will happen. That's because we need to add Behavior. Click Back if needed to return to the Scene view, then double-click on the raindrop Actor. Make sure to do this in the Inspector, rather than in the Scene. That way any changes you make will apply to any further instances of the raindrop that you create. We are now in the settings for the raindrop *prototype*:



Attributes

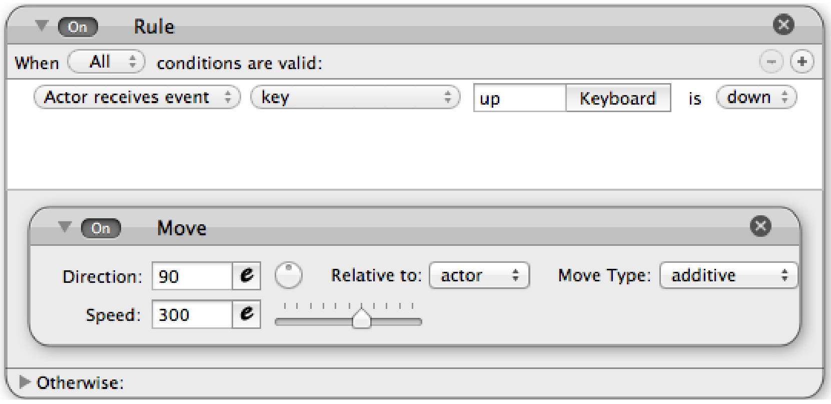
Name	raindrop	text
Time	0.0	real
Position		point
Size		size
Rotation	0.0	angle
Color		color
Image	raindrop	image
Tags		text
Preload Art	✓	boolean

Accelerate *(Persistent Behavior)*

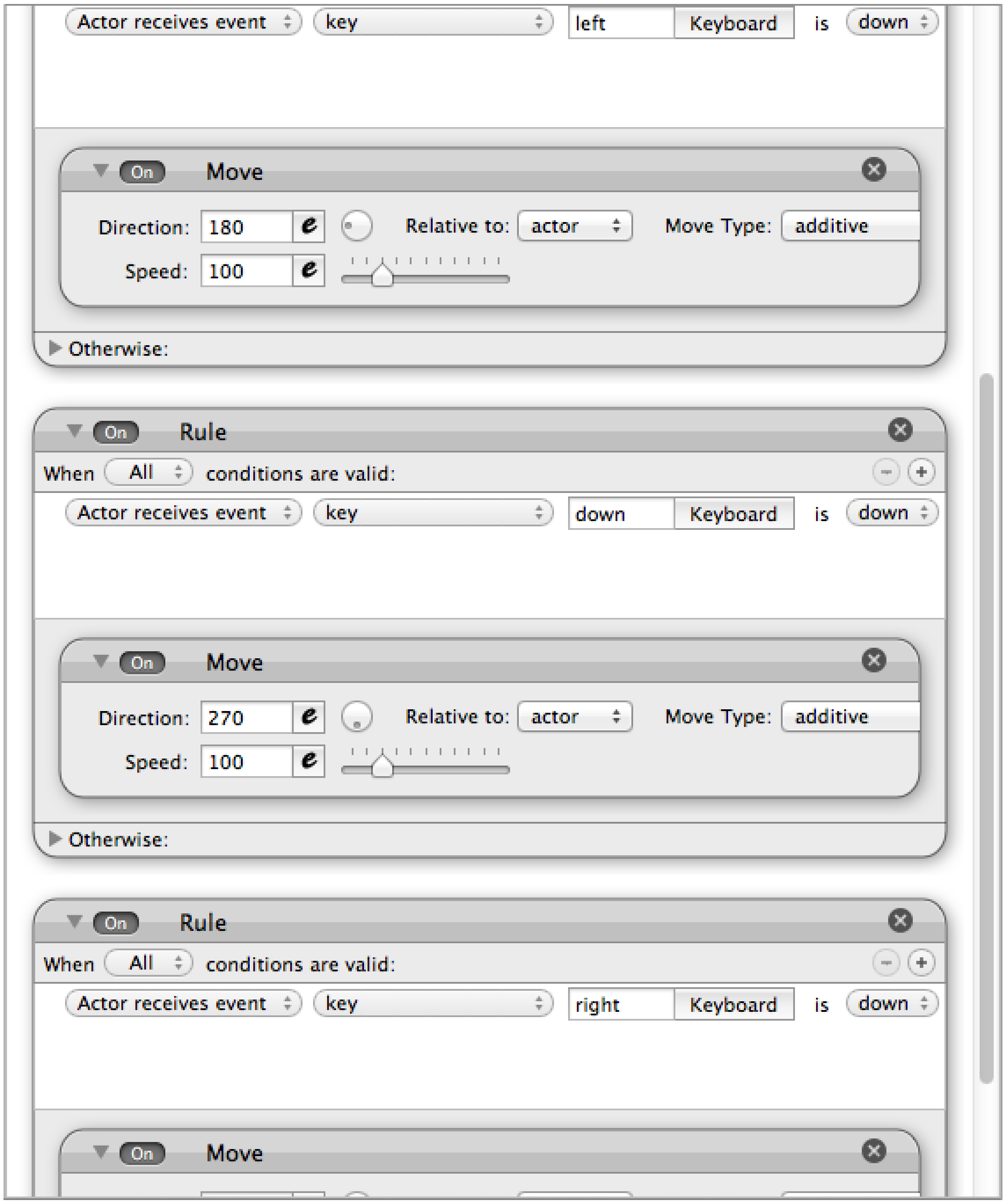
Specify the speed and direction (angle) of acceleration of an actor. Best used in a rule that checks for a keyboard event. Note: If the

We want to be able to move the raindrop using the keyboard. That means we need a Rule to check for keyboard input. Click "Create Rule" in the upper-right of the blank white box. As we see, the default settings check for the position of the mouse. Click on "mouse position" and choose "key" instead. Click on the word "keyboard" that appears to the right to summon a virtual keyboard, then click on the UP arrow key. Now we have a Rule that runs whenever the UP arrow key is down, i.e. pressed.

Now, scroll through the Behaviors in the Library in the lower-left until you come to Move, then drag it into the Rule we just made. The default direction is to the right, as we see from the circle representation. Direction is calculated as an angle, with to the right being 0 degrees and the angle increasing clockwise, so going up would be 90 degrees. The Rule should look like this:



Click Preview. The raindrop moves up when you press UP! Of course, it also passes right through the cloud. Still, progress! Go back and add three more Rules to make the raindrop move left (180 degrees), down (270 degrees), and right (0 degrees, which is the same as 360 degrees). Considering the size of our game screen relative to the raindrop, it also makes sense to slow down the movement speed. Feel free to experiment with the speed and use Preview frequently until it feels right to you. The Behavior is now lengthy enough that it won't all fit on one screen, but the new content should look something like this:


Actor receives event | key | left | Keyboard | is | down

On Move
Direction: 180 | Relative to: actor | Move Type: additive
Speed: 100

Otherwise:

On Rule
When All conditions are valid:
Actor receives event | key | down | Keyboard | is | down

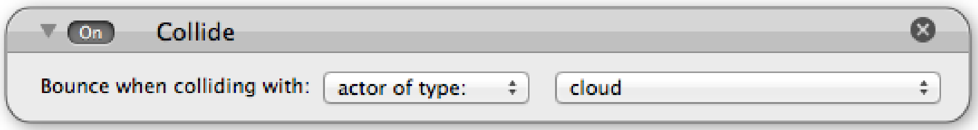
On Move
Direction: 270 | Relative to: actor | Move Type: additive
Speed: 100

Otherwise:

On Rule
When All conditions are valid:
Actor receives event | key | right | Keyboard | is | down

On Move

If the raindrop can move through the cloud walls then we don’t have much of a game, so let’s fix that next. We need to tell the raindrop to *collide* with the cloud. This is easy to do. Return to the raindrop prototype’s Behavior screen. Scroll through the Behaviors in the Library until you find “Collide,” and drag that into empty space in the Behavior list. You don’t need to create a Rule first (and make sure that you don’t add the Behavior to any existing Rules). Here’s the result:


Collide

Bounce when colliding with: actor of type: cloud

If yours did not default to “cloud,” then click the drop-down menu on the right and select it.

Hit Preview again and make the raindrop run headfirst (it’s all headfirst, really) into the cloud ceiling. You’ll almost certainly notice something very strange. The raindrop pushes the cloud out of the way, and depending on how you ran into the cloud, it and/or the raindrop may rotate. It’s rather fun, actually. Try it a few times. You can click the curly arrow just above the game window to reset the preview.

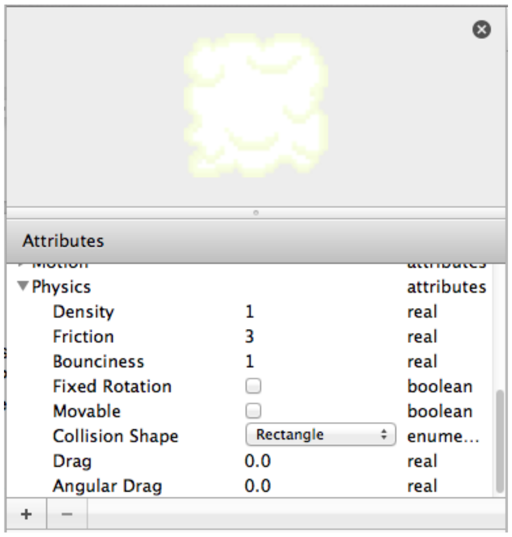
“Cloud, is there a wind blowing or what?”

As entertaining as this is, it is not what we want for our game. We didn’t give the cloud any Behavior! What’s going on? Well, GameSalad includes a *physics* system that simulates what would happen in an actual collision. Because we’ve told the raindrop to bounce off the cloud (instead of passing through it), the collision imparts some momentum to the cloud. That’s what causes the cloud to float away. Furthermore, as per Newton’s Third Law, the cloud gives some force back to the raindrop--not enough to actually move it, perhaps, but enough to make it rotate.

Yeah. It’s kind of weird.

Well, forget that! This is no physics game; it’s about raindrops faceplanting into walls of cloud, just like real life. We need to tell the cloud not to move. This, too, is easy to do. Click the Back button to return to the Scene view, and double-click on the cloud prototype.

In this case we are not going to use Behaviors. Instead, look on the left underneath the image of the cloud. You’ll see a list of its *Attributes*. These are properties of the cloud, such as position and color. If you scroll to the bottom, you’ll see three categories of Attributes in a list by themselves: *Graphics*, *Motion* and *Physics*. Guess which one we want? Yes, click the arrow to the left of *Physics* to reveal its selection of Attributes. If you scroll down further, you’ll see there is one called “Movable,” with a checked box next to it. Uncheck that box!

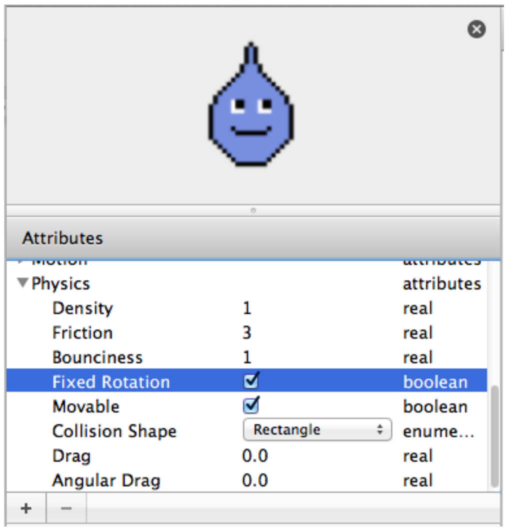


Attributes

▼Physics		attributes
Density	1	real
Friction	3	real
Bounciness	1	real
Fixed Rotation	☐	boolean
Movable	☐	boolean
Collision Shape	Rectangle	enume...
Drag	0.0	real
Angular Drag	0.0	real

Now if you click Preview you'll discover that the cloud is as solid and immovable as a boulder, which for a raindrop is probably appropriate.

Depending on how you bump the raindrop against the cloud, you may discover that the cloud can still cause the raindrop to rotate. Return to the details of the raindrop and open its own *Physics* Attribute list. Now, if you uncheck Movable for the raindrop and then run the game, you'll notice you can't move the raindrop, despite the Behavior we assigned it. Well, that's not surprising. Movable is Movable. However what we are concerned about is rotation, and just above Movable there is another Attribute called "Fixed Rotation." Check that box. Now the raindrop is restricted to its default orientation.

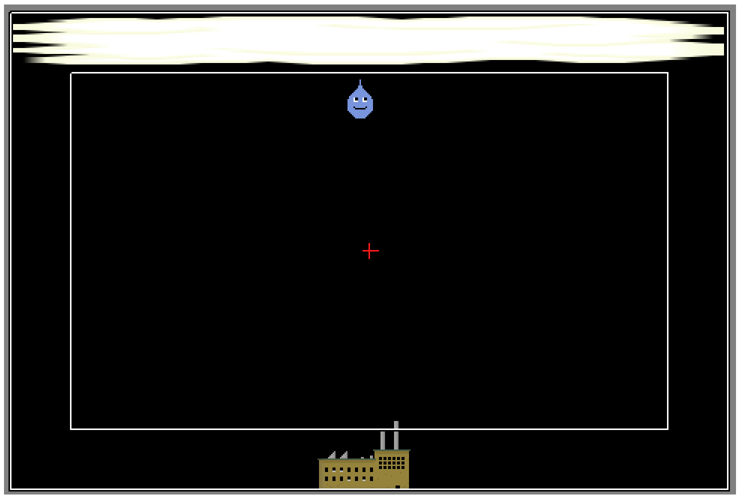


Attributes

▼Physics		attributes
Density	1	real
Friction	3	real
Bounciness	1	real
Fixed Rotation	✓	boolean
Movable	✓	boolean
Collision Shape	Rectangle	enume...
Drag	0.0	real
Angular Drag	0.0	real

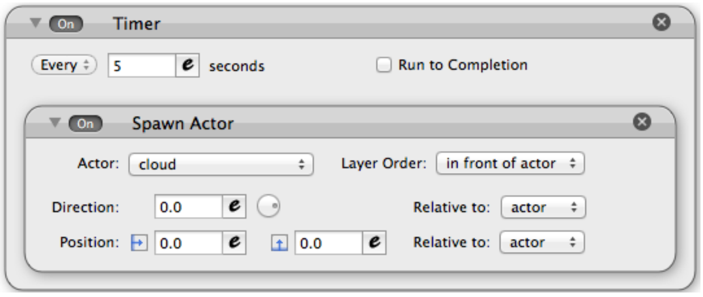
Alright! We've got a raindrop we can move around and a cloud to block its way! These are the basic ingredients of our game. But it's not a game yet. We are missing three important things: the remaining walls, an endpoint, and a hazard (i.e. a threat to the raindrop). It might be tempting to build the rest of the level first, but the way the hazard works will influence the way you design the level, so let's take care of that hazard.

In our game, the raindrops must avoid the smog belched from the factory's chimney. That means we need a factory. Drag the factory Actor from the Inspector into the Scene. Unless this is a floating factory you'll probably want to put it somewhere along the bottom of the screen.

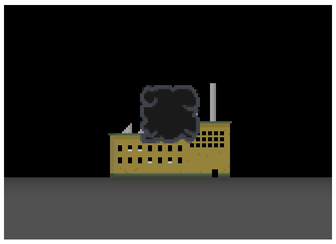


Hmm...the scale looks a *little* off, but you know what, don't even worry about it. Video games! Anyway, we are going to do something neat next. Since the factory should be producing smog, we are going to make the factory Actor produce instances of the smog Actor. That's right--you do not need to place any smog in the Scene! The game will do it for you! Once you tell it to.

Double-click on the factory prototype. We will set the factory to create a new smog instance every few seconds. In the list of Behaviors, find Timer and drag it into the factory's Behavior. You'll see that, as with a Rule, the timer establishes a condition (in this case every certain number of seconds) to run additional Behavior(s). Now find Spawn Actor and drag that into the Timer you just created. You'll see there are many parameters to set. Let's take a look:

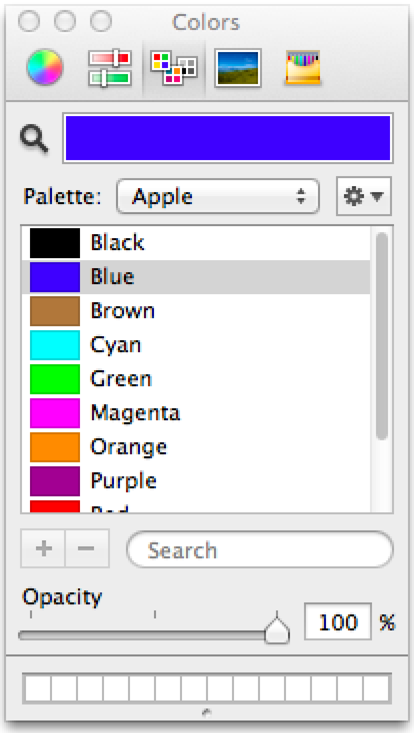

Timer
Every 5 seconds
Run to Completion
Spawn Actor
Actor: cloud
Layer Order: in front of actor
Direction: 0.0
Relative to: actor
Position: 0.0 0.0
Relative to: actor

First of call, click on the the Actor drop-down and set it to smog, if it's not that already. The Layer Order is fine. The Direction is a little confusing since we are not setting movement here. Essentially it functions as the direction we want the smog instance to be "facing." Leave it at zero. We'll probably want to edit the Position coordinates but for right now try clicking Preview.

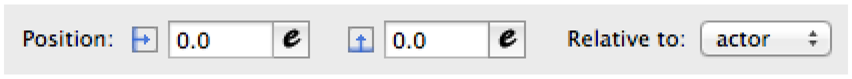


We notice a few things. 1) The smog does spawn (hurray!), but it spawns in the middle of the factory's ceiling. That's not where smog comes from! 2) Once spawned, the smog just sits there (sadly that is pretty true to life, but not good for our purpose). 3) It is rather hard to see the black smog against the black background. What is this, anyway--a night game?

Let's give the game a nice blue sky backdrop. Return to the Scene view. In the Inspector, click on the *Scene* tab (above *Devices*), then make sure *Attributes* is selected. You'll see there's an Attribute for Color, with a black box. Click on the black box to make a color window pop up. At the top, just under the word "Colors," are five icons. Hover over the middle one and you'll see a popup saying "Color Palettes." Click that icon, then choose "Blue" from the palette.



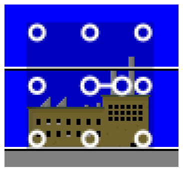
If you Preview the game again, you'll notice that the smog is now much easier to see, but the other problems remain. Now we will adjust the Position settings we saw earlier. Return to the factory Actor's Behavior. Take a closer look at this section of the Spawn Actor behavior:


Position: 0.0 0.0 Relative to: actor

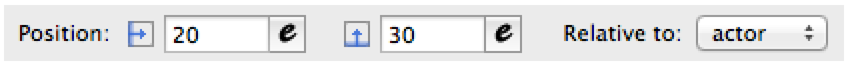
What does this mean? The arrows to the left of each coordinate indicate axis. The arrow pointing to the right represents the x-axis while the arrow pointing up represents the y-axis. "Relative to actor" means the position of the smog is relative to the position of the factory.

OK, but what does this really mean? Well, the entire Scene is drawn using a coordinate system that originates in the lower-left corner. So the lower-left corner is point (x=0, y=0), with x increasing going right and y increasing going up. However, GameSalad draws sprites from their *center* outward. So if you imagined the factory as its own little coordinate system, the origin point (0,0) would be the center of the factory sprite. Just so, the origin point of the smog would be the center of its sprite.

In other words, the smog would be drawn from its center regardless, and we are telling it to be drawn at the factory's center, point (0,0). Yes, the roof doesn't look like the center of the factory, but if you click on the instance, you'll see that in fact the center of the sprite is at that exact point:

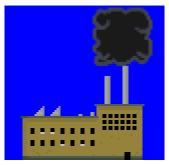


Enough explanation; what do we do? The smog should appear from the factory's chimneys, which are to the right and up from the sprite's center, so we need to replace those zeros in the position settings with positive numbers. You should experiment to get just the positioning you want, but you'll probably end up with something like this:

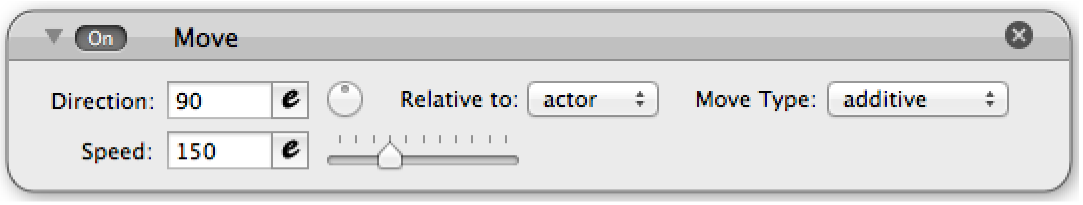


Position: 20 30 Relative to: actor

which produces this:



Now we need to make the smog move. For that, we have to set its own Behavior. Return to the Scene view and then double-click on the smog prototype. Give it the Move behavior. We don't need a Rule for this; the smog will start moving as soon as it's created. Set the Direction to 90 (straight up) and set the speed to whatever you like. The result should look similar to this:



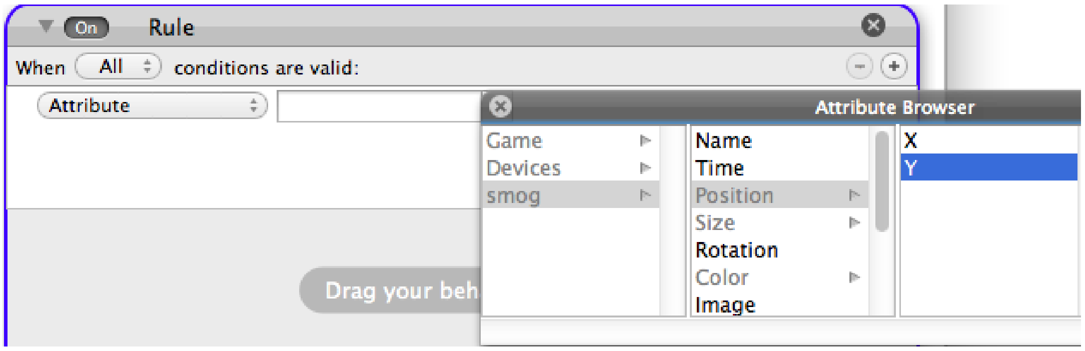
Move

Direction: 90 Relative to: actor Move Type: additive

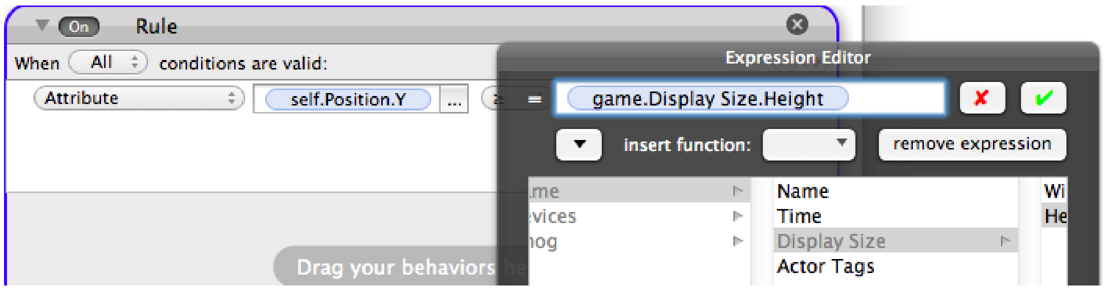
Speed: 150

Now if you Preview the game, you'll see that every few seconds a new smog instance appears and moves upward. Huzzah! At this point you may wish to adjust the Timer on the factory to make the smog appear more (or less) frequently.

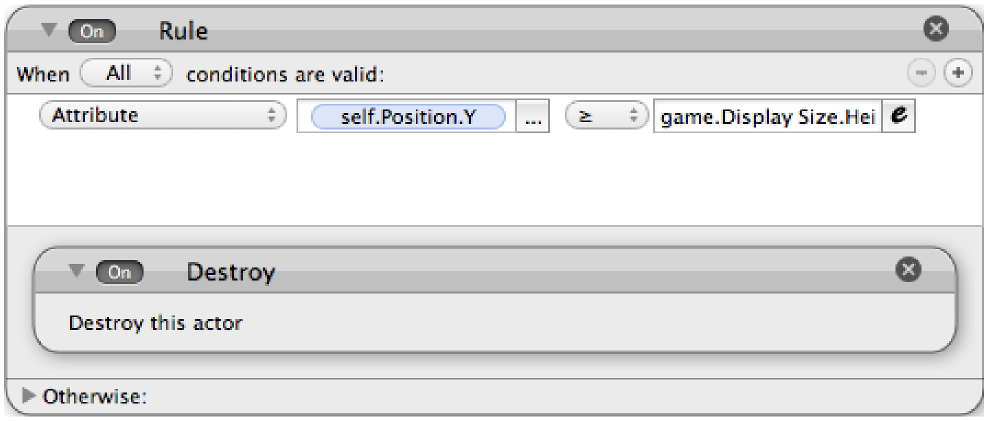
We should tell GameSalad to destroy the smog instance if it goes offscreen, to minimize the number of things the game has to keep track of simultaneously. Since the smog travels up the y-axis, we need to check its y-position. Return to the smog's Behavior and click Create Rule, then change the condition from "Actor receives event" to "Attribute." Click on the empty box to the right, then choose "smog → Position → Y" like so:



Double-click on “Y” to fill the box. An equal sign will appear to the right. Click on it and change it to “greater than or equal to,” which looks like an arrowhead pointing to the right over a single horizontal line. Finally, click on the “e” symbol on the far right to open the Expression Editor. Click on the down-arrow on the left to open the Attribute list like before, but this time choose “Game → Display Size → Height.”

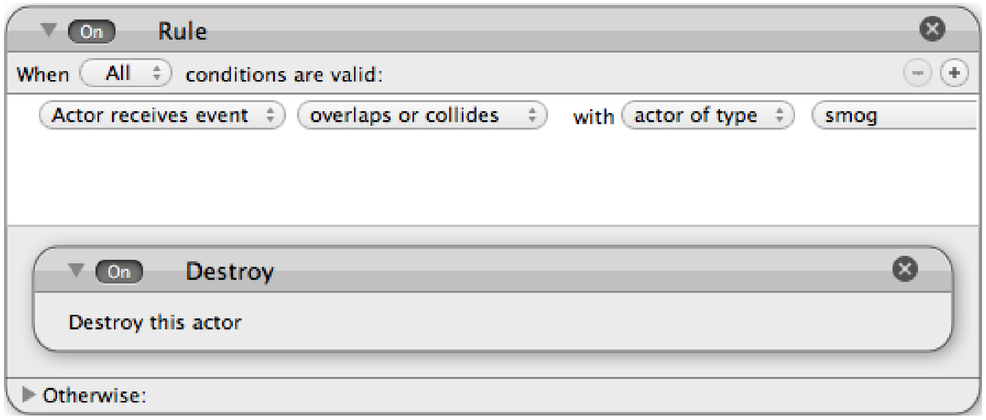


Double-click on “Height” to fill the Expression Editor’s box, then click the green checkmark to save the result. Finally, drag the Destroy behavior into the Rule. The result should look like this:



What have we done? This says that if the y-position of the smog--its center--reaches the top of the game screen, GameSalad should destroy the smog.

We're doing well! However right now the smog does nothing to the raindrop. Let's change that. Open up the raindrop's Behavior once again and create a Rule. This time we will check for an event, but instead of the default mouse position, change it to "overlaps or collides," and then set the Actor type to "smog." Can you guess what comes next? Yes! Drag the Destroy behavior into the Rule.



On Rule

When All conditions are valid:

Actor receives event | overlaps or collides | with actor of type | smog

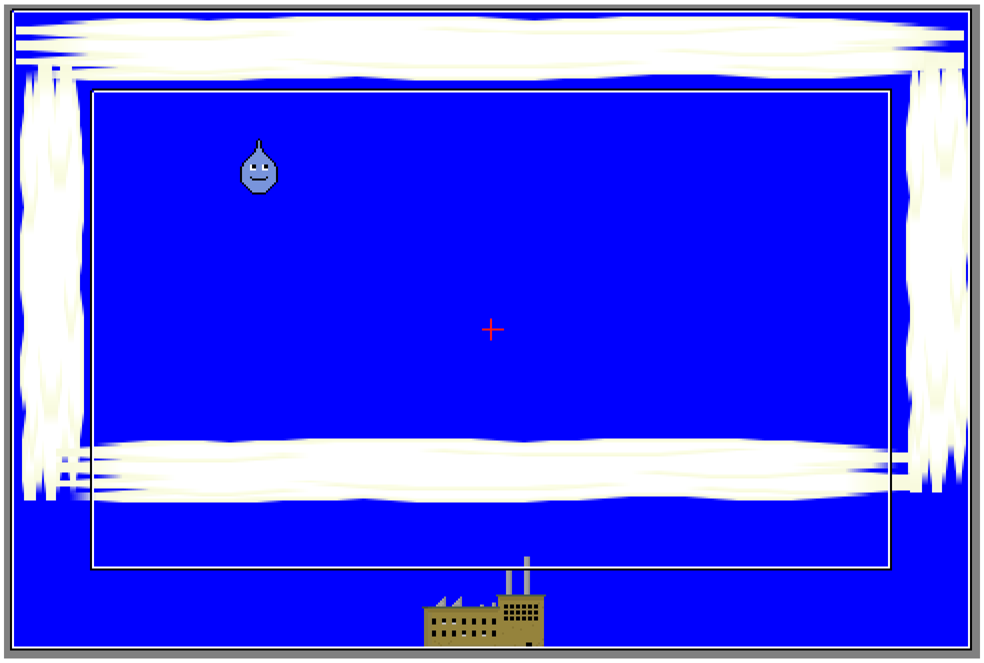
On Destroy

Destroy this actor

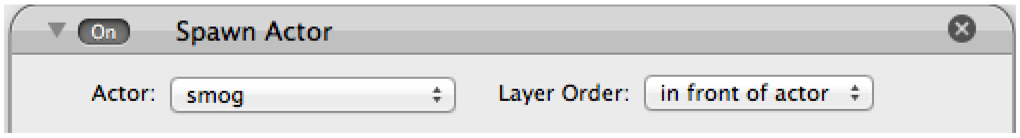
Otherwise:

Hit Preview and run the raindrop into the smog. The raindrop disappears! That darn smog.

Great work so far! Take a deep breath. You've earned it! We are very close now to establishing all the basic mechanics and finishing the first level. Victory is so close! But there is a little more work to do. First, build cloud walls along the sides of the screen and a "floor" to complete the cloud box. Make sure to place the cloud floor above the point where the smog appears. Feel free to do this however you like, but the result should look something like this:



If you Preview the game, you may discover that the new clouds you've added cover over the smog. This is because by default GameSalad displays things in the order you placed them in the Scene. We added the new clouds after we added the factory that generates the smog, so the new clouds take precedence. Fortunately, we can change this. Open the factory Actor's behavior again and look at the Rule for Spawning the smog Actor. You'll see the option in the upper-right is "Layer Order":



Actor: smog
Layer Order: in front of actor

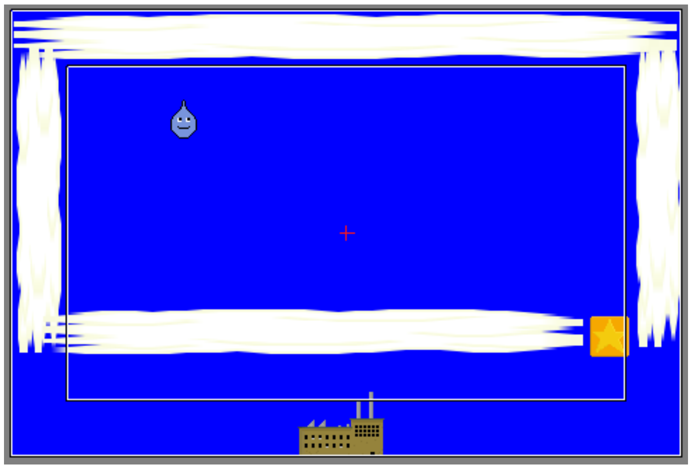
Think of a *layer* as a single sheet of transparent paper. You could draw a tree and a house on one layer and a family on another, then put one layer atop the other to see the combined picture. Within the first layer, you might decide to draw the tree in front of the house, or vice-versa.

GameSalad works the same way. By default, the Spawned Actor appears in front of the Actor that Spawned it; that is, the smog appears in front of the factory. But anything further you add to the Scene (within the same layer) will appear in front of the smog. Change the Layer Order on the Spawn Actor behavior to "front of layer." Now the smog will appear in front of everything, even if you add more Actors to the Scene later.

Note: any GameSalad project starts out with a single layer, and anything you add to the Scene will be added to that layer. However, you can create new layers and distribute your Actors among them. We will not do so for this tutorial, but feel free to play around by going to the Scene tab in the Inspector, then clicking the Layers subtab.

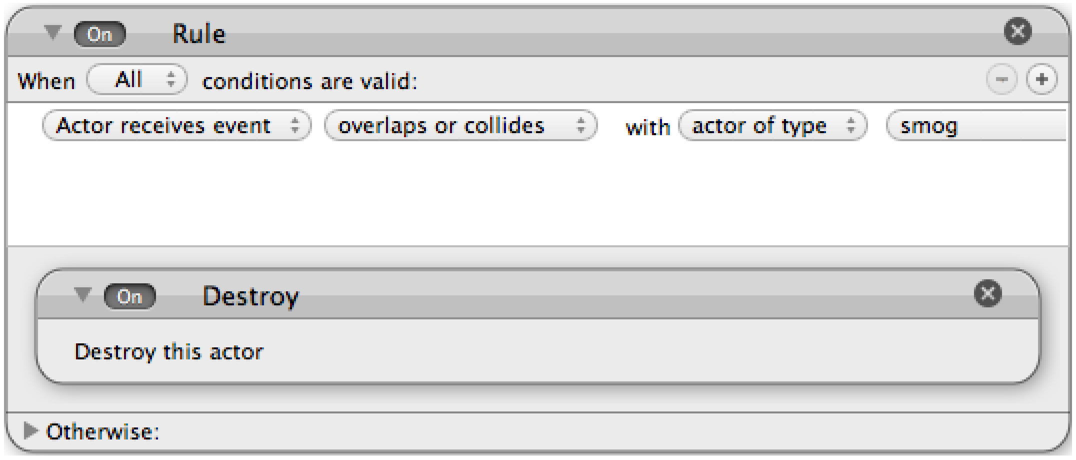
We have a (very bare-bones) level, a playable character and a hazard, but there's no way to win! Let's add a goal. Place an instance of the star Actor in the Scene. This will be the goal the player has to reach. Remember that it represents escape from the maze (and freedom for the raindrop to fall to the ground below), so you want to put it on the outside of the maze. It probably makes the most sense to put it along the bottom, although you don't have to.

Make sure to remove or stretch the clouds as needed to make room for the star. The raindrop bounces off the cloud but it needs to be able to move into the star. However you do it, the result should look something like this:



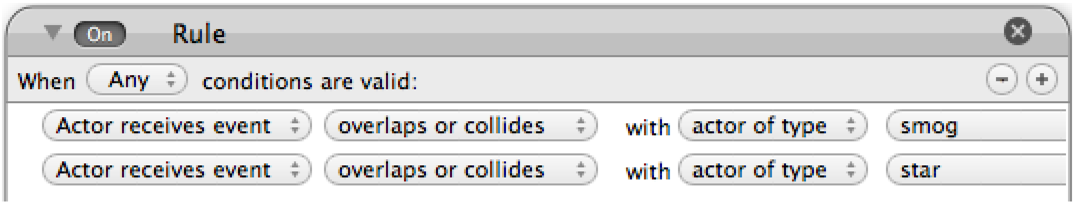
Now, what should happen when the raindrop collides with the star? For one thing, we should destroy the raindrop, since we don't want the player to be able to control it after it has reached the goal. Eventually we'll also want to move to the next level, but for now let's restart the game.

Open up the raindrop prototype's Behavior. You may recall that we already created a Destroy behavior for the collision between the raindrop and the smog:



We can apply this same Rule to the collision with the star. Click the plus button in the upper-right of the rule to create an additional condition. Change the event type to "overlaps or collides" and change the Actor type to "star."

If you Preview the game now you'll notice that the collision with the star doesn't have the desired effect--and not only that, suddenly the collision with the smog doesn't work, either! Why? Look at the top of the Rule, just above the first condition. It says "When All conditions are valid." That means the revised Rule requires the raindrop to be touching both the smog and the star simultaneously, which will never happen in our current setup. Click on "All" and change it to “Any.” Now run the game. The raindrop gets destroyed if it collides with either object. Success!


Rule

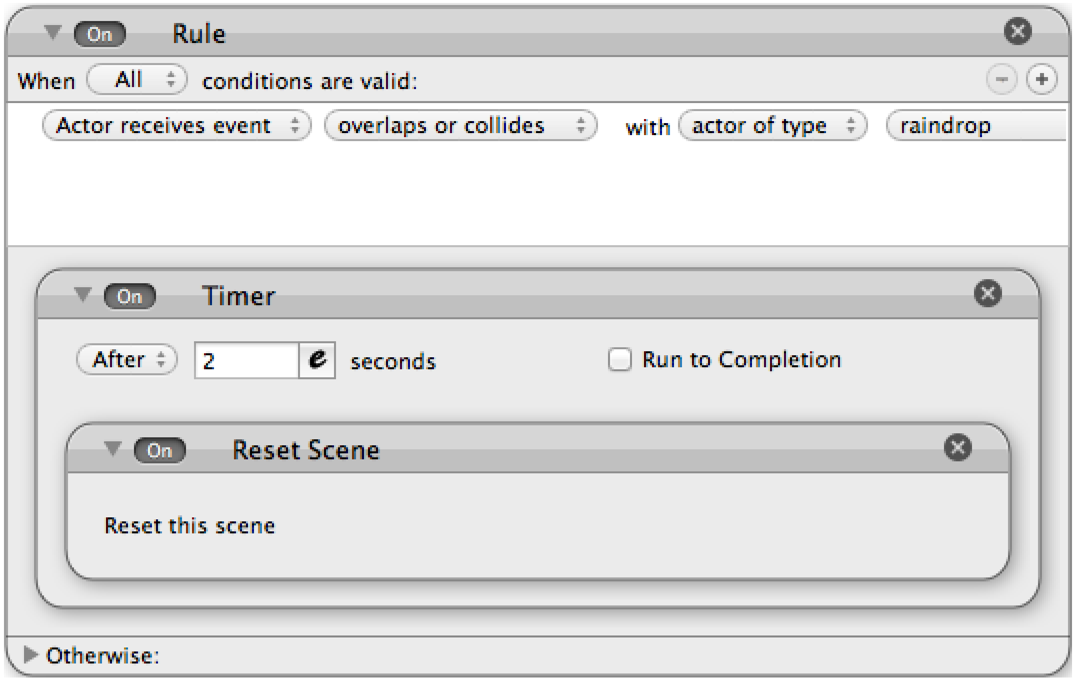
When Any conditions are valid:

Actor receives event | overlaps or collides | with | actor of type | smog

Actor receives event | overlaps or collides | with | actor of type | star

Finally, open up the star prototype’s Behavior. Create a Rule and set it to a collision with the raindrop. Now, drag the Reset Scene behavior from the Library into the Rule. Preview the game. It works!

It works right away though; it would be nice to put in a little delay. We can do that! Go back to the star and drag the Timer behavior into the Rule. Click on “Every” and change it to “After.” Change the default 5 seconds to 2 seconds, or any other number you like. Now, drag the Reset Scene behavior that is already in the Rule into the Timer, so that the Rule contains the Timer and the Timer contains the Reset Scene behavior. It should look like this:


Rule

When All conditions are valid:

Actor receives event | overlaps or collides | with | actor of type | raindrop

Timer

After 2 seconds ☐ Run to Completion

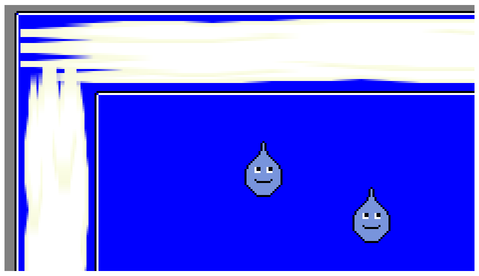
Reset Scene

Reset this scene

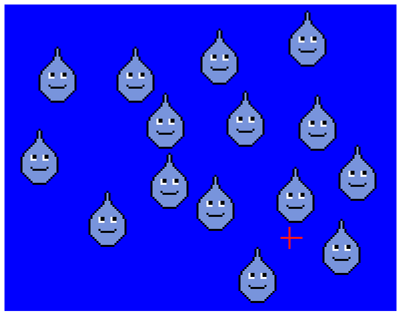
Otherwise:

Preview the game. Hmm...that’s weird, it doesn’t seem to work. That’s because the Timer event only runs when the star is colliding with the raindrop. When the raindrop destroys itself, there is no longer a collision and the Timer stops running, so it doesn’t reach 2 seconds and doesn’t reset the Scene. But wait! What’s that “Run to Completion” checkbox on the right of the Timer? Check it. Play again, and...it works! Use that feature to finish an event even if conditions change.

If you played the game on the website you'll have noticed that you sometimes control multiple raindrops simultaneously. That must be difficult to implement, right? Wrong! It couldn't be easier. Drag the raindrop Actor prototype into the Scene a second time.



Now Preview the game. Amazing! Both raindrops move when you press a direction. This is because we gave the Behavior to the *prototype*, and then used the prototype to create multiple *instances* in the Scene. The Behavior of the prototype applies to all the instances. Still, remember that the instances do exist independently of each other. Destroying one will not destroy the other(s). If you move the two in the picture above to the left, the higher one will collide with the wall first and remain in place while the lower one keeps moving until it collides with the wall as well. Create as many (or as few) raindrops as you like!

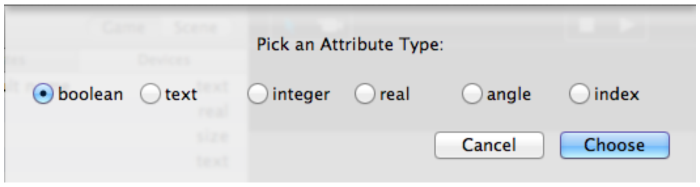


Uh-oh.

Note that the introduction of additional raindrops does cause one new problem. There's always a catch, right? You'll notice that if one of the raindrops touches the star, the game restarts the Scene as we previously established. That may be just fine with you. But what if you want to let the player continue to control the remaining raindrop(s)? Perhaps you want to require the player to get every raindrop to the exit safely, or note how many raindrops the player is able to save.

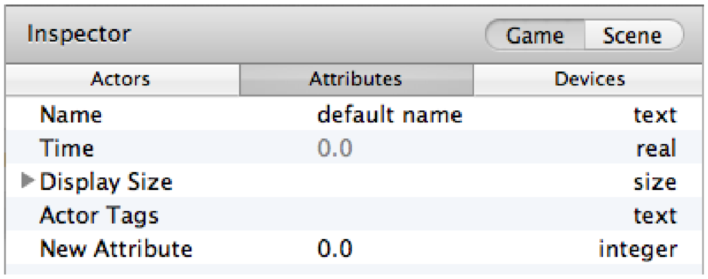
Either way, we need something to keep track of how many raindrops are in play. We can use that to control when the game advances or restarts a level.

Let's start with a simple count of how many raindrops there are. For this we'll need to create a new Attribute. We should be able to use this Attribute regardless of the level (i.e. Scene), and every Actor should be able to see its value. That means we need to create a *Game*-level Attribute. Return to the Scene view. Look in the Inspector and make sure the *Game* tab is selected, then click on *Attributes*. Click the plus button in the lower-left of the Inspector and...


Pick an Attribute Type:

[● boolean] ○ text ○ integer ○ real ○ angle ○ index

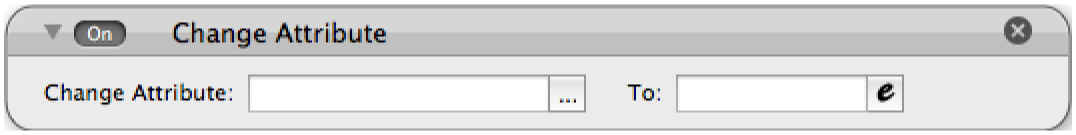
...you'll see that. For more guidance on the different types of Attributes, refer to **GameSalad Basics**. We are just trying to count the raindrops, so pick "integer" and click "Choose."



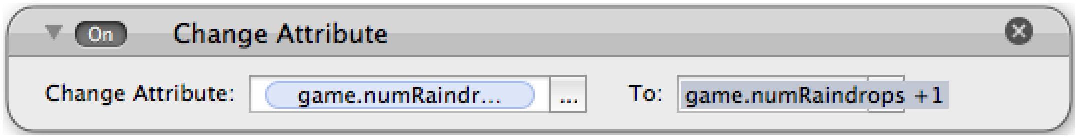
Name	default name	text
Time	0.0	real
▶ Display Size		size
Actor Tags		text
New Attribute	0.0	integer

There it is, at the bottom of the list! However, "New Attribute" isn't very descriptive. Double-click on "New Attribute" and rename it to "numRaindrops" (i.e. "number of raindrops).

Of course, that Attribute won't do anything on its own; just because we called it "numRaindrops" doesn't mean it knows how many raindrops there are! Open up the raindrop prototype and look at the Behaviors in the Library. At the top you'll see one called "Change Attribute." Drag it into the raindrop's Behavior. Make sure that you put it outside of any of the other Behaviors and Rules already applied to the raindrop. Now we have this:



Click in the "Change Attribute" box, then choose "Game" and double-click on "numRaindrops" to select it. Now, click the "e" to the right of the "To" box to open the Expression Editor. We need to add 1 to numRaindrops. Click the down arrow on the left of the Editor and select numRaindrops as before, then type "+1" (without the quotation marks). Click the green checkmark to save the expression. Yes, it really is that easy.

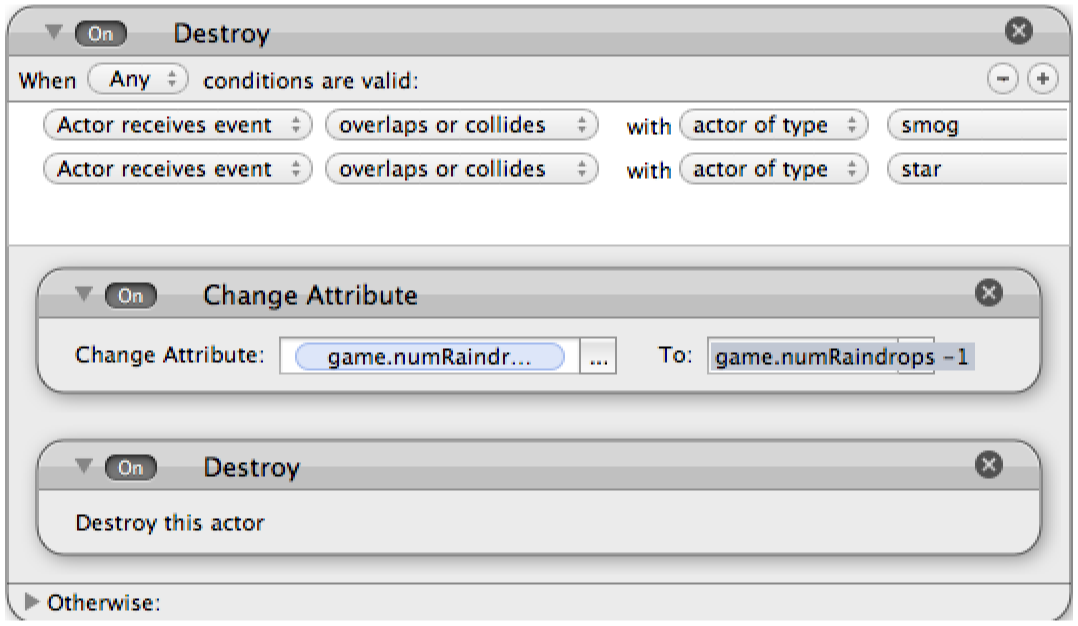

Change Attribute

Change Attribute: game.numRaindr... To: game.numRaindrops +1

You may wonder why assigning this as Behavior doesn't keep adding 1 to numRaindrops, over and over again. By default, Behavior like this runs just once, upon the Actor's creation. It would only run repeatedly if we assigned it to a Timer or a Rule with a repeating condition.

Great, we have a count of the total raindrops! You can probably guess what comes next. If we used Change Attribute to add 1 to numRaindrops for every raindrop created, we can use it again to subtract 1 from numRaindrops for every raindrop destroyed. Try to do that on your own.

If you need help, look at this next image for guidance (hint: modify the raindrop prototype's existing Behavior):


Destroy

When Any conditions are valid:

Actor receives event overlaps or collides with actor of type smog

Actor receives event overlaps or collides with actor of type star

Change Attribute

Change Attribute: game.numRaindr... To: game.numRaindrops -1

Destroy

Destroy this actor

Otherwise:

Note: you may wish you could see the value of numRaindrops yourself, to make sure it's working properly. There are several ways to do this. When you are testing a game, often while trying to figure out why something isn't working (this is called "debugging"), a common technique is to use the "console," a special window that appears outside of the game and lets you see data from the game as it's running. You can tell the game to "print" or "log" (i.e. display) specific values within the console. This is a great technique (and essential for any kind of software development), but it's a little more advanced than this tutorial requires. If you're curious, read more about debugging and the console at the end of ***GameSalad Basics****.*

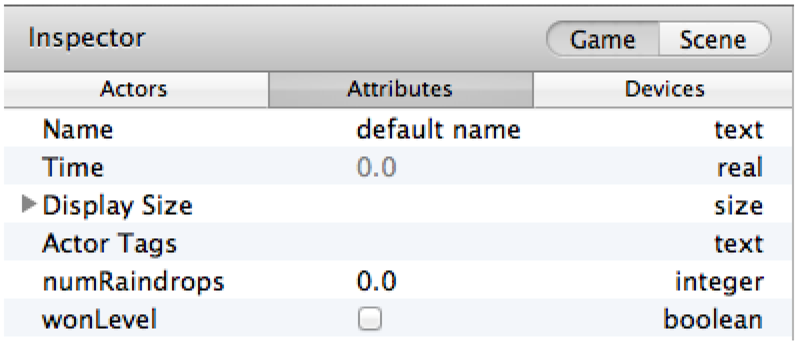
Now we have to tie when the level restarts (or advances) to numRaindrops. Open up the star prototype. We can use the Rule we created to reset the Scene, but we no longer need it to check for a collision with a raindrop. Instead, change “Actor receives event” to “Attribute.” Set the Attribute to numRaindrops (Game → double-click on “numRaindrops”). By default it checks if the Attribute equals zero, which in this case is exactly what we want. (We can also uncheck “run to completion” on the Timer, since the condition of zero raindrops doesn’t change.) Preview the game. Now the Scene won’t reset until all the raindrops have been destroyed!

Of course, if you think about that last sentence you can probably anticipate a new problem. The star only checks whether all raindrops have been destroyed--not whether any of them has collided with the star. They could all be melted by smog and the level would still reset. That means the player no longer has to reach the goal. Boo.

How could we solve this? Many ways! (This is always true in design. Take it to heart.) We *could* make it so that numRaindrops only depletes when the raindrop collides with the star. On the other hand, that would not allow us to reset the level if the player “lost” (presumably by running all the raindrops into smog), and it would mean the player has to save every raindrop.

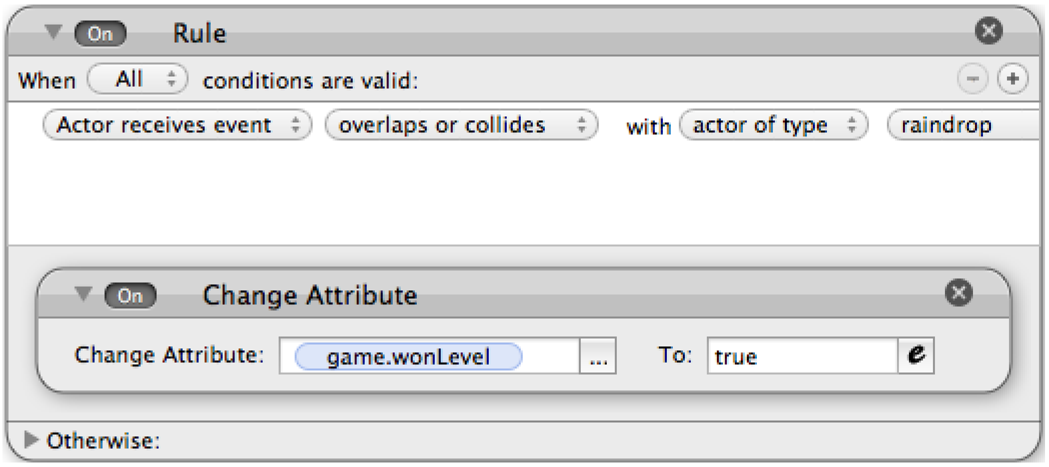
It would be more interesting to let the player win the level as long as one raindrop makes it to the goal, but let the player play out every raindrop (i.e. not end the level as soon as the first raindrop hits the star). This means we need another condition for “winning” the level, one that checks whether any raindrop made it to the end of the maze.

It’s time to make another new Attribute. Create another one just like you did numRaindrops, in the same place (*Attributes* under the *Game* tab in the Inspector), but this time leave it on the default type of “boolean.” This is an Attribute that can only be set to “true” or “false.” Rename this one “wonLevel.” Note the checkbox to the right. It should be empty. That means the default state of wonLevel is “false.” Make sense, right? Leave it unchecked.



Name	default name	text
Time	0.0	real
▶ Display Size		size
Actor Tags		text
numRaindrops	0.0	integer
wonLevel	☐	boolean

Open up the star prototype’s Behavior again. We need to add two things. First, create a new Rule that checks for a collision with a raindrop. Give it the Change Attribute behavior, and set that to change wonLevel to “true” (you’ll need to type it out). The raindrop will still be destroyed when it runs into the star, but the star will “note” that the player has saved at least one raindrop and therefore has “won.”


Rule

When All conditions are valid:

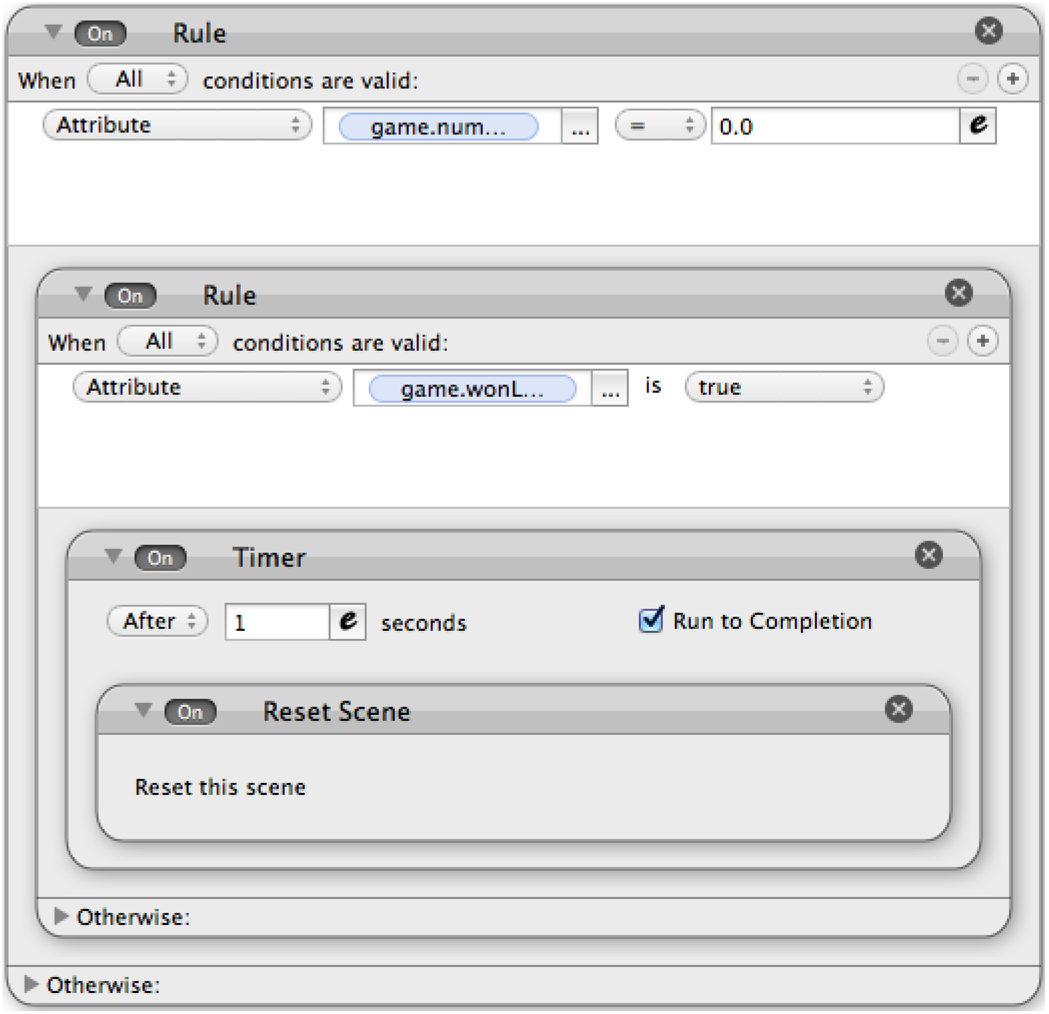
Actor receives event | overlaps or collides | with | actor of type | raindrop

Change Attribute

Change Attribute: game.wonLevel To: true

Otherwise:

Now, create a new Rule with the condition that the wonLevel Attribute is true. Drag this Rule inside the Rule checking for zero raindrops, making sure to place it outside the Timer behavior already assigned. Then drag that Timer behavior inside the new Rule. The result looks like this:


On Rule

When All conditions are valid:

Attribute game.num... = 0.0

On Rule

When All conditions are valid:

Attribute game.wonL... is true

On Timer

After 1 seconds ☑ Run to Completion

On Reset Scene

Reset this scene

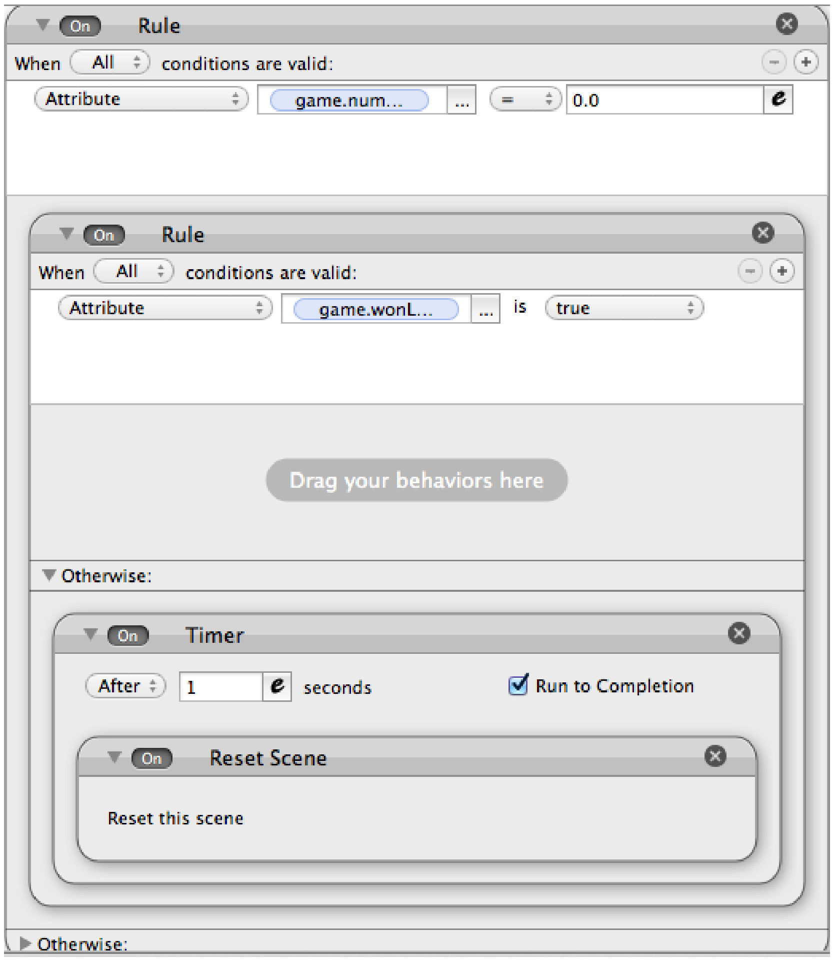
Otherwise:

Otherwise:

Note that we could have created another condition in the outermost Rule to check for whether wonLevel is true. Then we wouldn't need the inner Rule. But remember that the reason we need to check whether wonLevel is true is that numRaindrops is insufficient, because it will equal zero whether the player wins or loses the level. Probably we want to do something different in each situation. Setting up the Rule(s) the way we did allows us to achieve just that.

For instance, it makes the most sense to reset the Scene if the player loses, and advance to the next Scene if the player wins. One thing we haven't looked at yet is the "Otherwise" that appears at the bottom of every Rule. If you click the arrow to the left you'll see that it expands to provide space for Behavior. This is Behavior that will activate if the condition for the Rule is not satisfied.

Drag the Timer with the Reset Scene behavior into the Otherwise of the Rule checking if wonLevel is true (what a sentence, eh?). Now the whole thing looks like this:


Rule

When All conditions are valid:

Attribute game.num... = 0.0

Rule

When All conditions are valid:

Attribute game.wonL... is true

Otherwise:

Timer

After 1 seconds ✓ Run to Completion

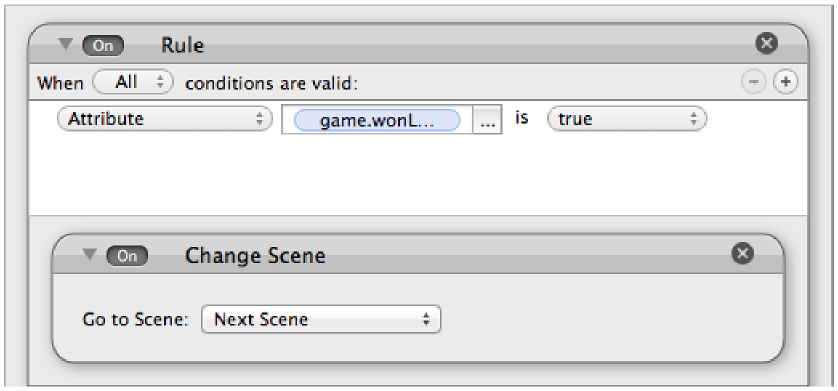
Reset Scene

Reset this scene

Otherwise:

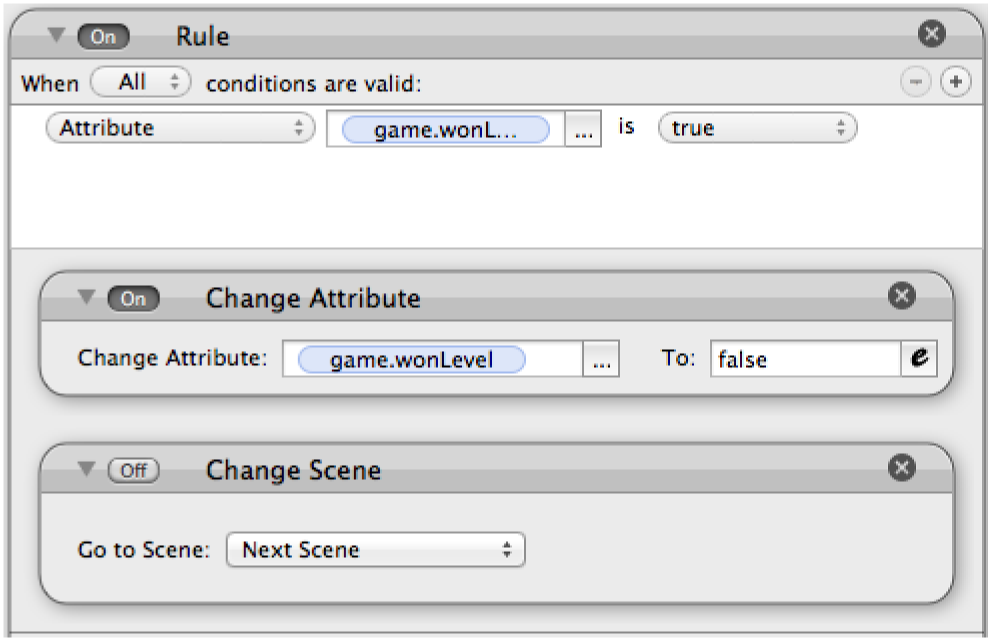
This says that if the number of raindrops is zero, the game will check if the player has “won” the level (i.e. saved at least one raindrop). If so, nothing will happen. If not, the Scene will reset.

Obviously, we want to make something happen if the player does win the level, which typically would be to advance to the next level. GameSalad has a Behavior called “Change Scene.” Drag it from the Library into the Rule for when wonLevel does equal true. You’ll see the Behavior lets you decide which Scene to change to, and defaults to the next Scene:


Rule
When All conditions are valid:
Attribute game.wonL... is true
Change Scene
Go to Scene: Next Scene

We don't have any more Scenes yet, so the game would just get confused if we tried to run that Rule. Fortunately GameSalad lets you easily switch Rules and Behaviors on and off. Where it says "On" to the left of "Change Scene," click it to change it to "Off."

There's one other important thing to note. Unless instructed otherwise, Attributes keep their most recent value. That means that wonLevel, which was set to true when a raindrop collided with the star, will *stay* set to true even when the game advances to the next level. The player could lose all the raindrops in that next level to smog and the game would still think the player had won. That won't do! Fortunately this is very easy to fix. Simply add a Change Attribute behavior to set wonLevel back to false just before the Scene changes. It looks like this:


Rule
When All conditions are valid:
Attribute game.wonL... is true
Change Attribute
Change Attribute: game.wonLevel To: false
Change Scene
Go to Scene: Next Scene

Always make sure to reset your Attributes where needed and save yourself much frustration.

So, what's left? Our level is not much of a level. There's hardly any challenge--just a straight path to the star and some slow-moving smog to avoid. This is supposed to be a maze; where's the maze? Well, put on your level designer hat, because that's where you come in. Create more cloud instances and build paths inside the cloud box. We set the Behavior already; all you have to do is place more walls. Feel free to create more raindrops and even factories too. Here are some tips:

***Preview often.** Don't spend a lot of time constructing an elaborate path only to discover that the raindrop doesn't fit through it, or can't get around a corner. This will happen to you. The collision is a little weird, especially with something like the cloud where the edges are curved. If you try to make a gap just wide enough for the raindrop to fit, you'd better make sure it does.

***Multiple characters is fun and challenging.** The game changes *dramatically* when you have to control more than one raindrop simultaneously. Every move you make affects every single raindrop. Moving one raindrop out of the way of the smog may move another right into it. *Take advantage of this!* Create a design that requires the player to think carefully about every move, and to utilize the game's rules. Recall the two-raindrop example above where one collided first?

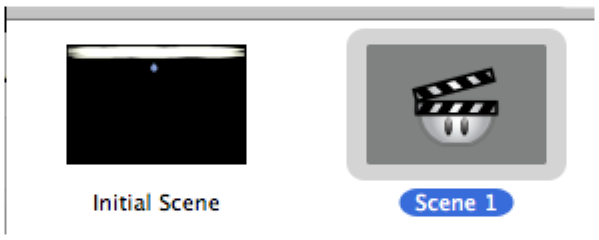
***Create tension.** Mazes are all about bendy paths, and paths that double back on themselves and require the raindrop to cross the line of fire (i.e. smog) multiple times are interesting. At the same time watch out that your paths don't become so convoluted that they become tedious.

***Preview often.** Yes, again. This time though the focus is on playing your game to make sure it's fun...and more importantly, manageable. Think you've just created the most challenging path ever? You'd better make sure you can solve it yourself, because if you can't, no one else can. Better yet, give it to other people and see if they can do it.

*If you're stuck, look to the game on the website for inspiration.

Your challenge is to create a three-level version of the game.

"Hang on," you say. "You haven't explained how to do multiple levels!" Good point. It's very simple. Click the Home button in the upper-left of GameSalad, then click the *Scenes* tab. You'll see the lonely Initial Scene. Click the plus in the lower-left. Ta-da! A brand new Scene!



Double-click on the new Scene to open up its own view, and go to work! We've already created Behavior for the star prototype to move to the next Scene; remember to go back and turn it On.

Your challenge is to create a three-level version of the game. For real this time.

Great job! You've made Smog Cloud Madness!

Think about how you might improve the game further. Many more levels? New hazards? New raindrop types? Rewards for rescuing all the raindrops instead of just one?

With this tutorial you've learned how to make a multi-level, multi-character game in GameSalad, and you've dipped into the infinite depths of level design. This is a good opportunity to think about how gameplay affects level design and vice-versa, and how external factors such as the Platform play a role. You probably ran up against the frustration of trying to build complex and interesting mazes within the relatively small bounds of a GameSalad Arcade-sized Scene. Furthermore, the factory at the bottom of the screen creates smog that floats upward, but the screen is wider than it is tall (and it's not very wide), so it's quite difficult to make passages that cross the smog's path multiple times, which minimizes the threat posed by a single factory.

For a game like this, it might make sense to choose a Platform other than GameSalad Arcade, one which gives you a lot more space to work with and is taller than it is wide--perfect for vertically-moving hazards and big, beautiful mazes. On the other hand, then you couldn't publish it on a website, because that requires the GameSalad Arcade format. It's very difficult if not impossible to make the same game for multiple Platforms, because they all have different properties. Know the strengths and weaknesses of your chosen Platform and design your game accordingly.

Feel free to build this game further, work through the other tutorials in the curriculum, or even start your own project. The possibilities are limitless!