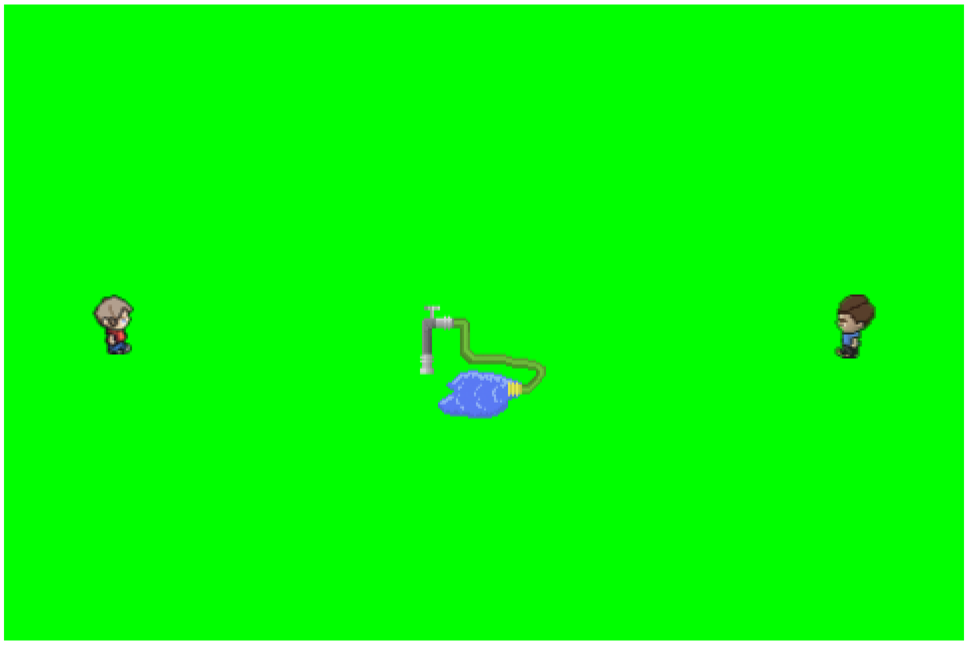
How to Make “Water Saver Sprint” in GameSalad

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 Water Saver Sprint, someone left the hose running and it’s a race between you and a friend to turn it off! This simplicity makes Water Saver Sprint a great introduction to creating games in GameSalad and shows how easy it is to include multiple players.

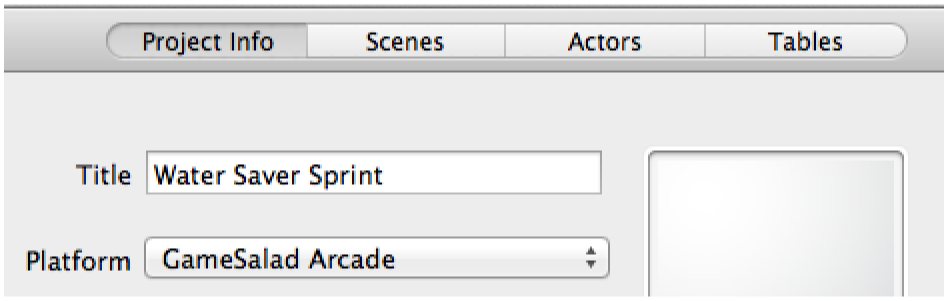
In this tutorial, you’ll learn how to recreate Water Saver Sprint. Make sure to download the folder of Resource Files for the game, and play the game on the website to see it in action.

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).

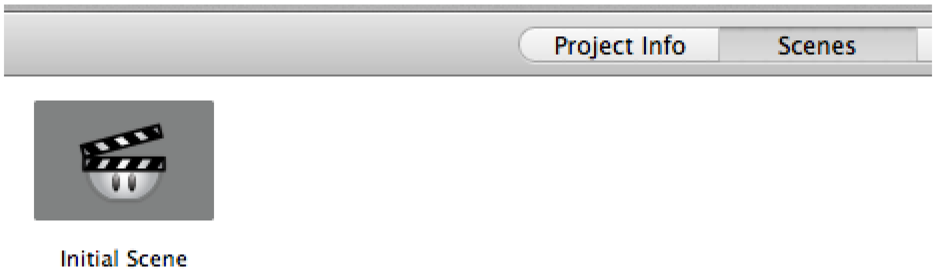
Alright, enough setup! Click the "Scenes" tab to the right of "Project Info":



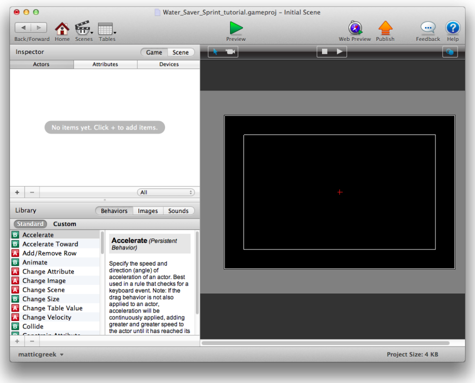
Title: Water Saver Sprint

Platform: GameSalad Arcade

Then double-click on "Initial Scene":

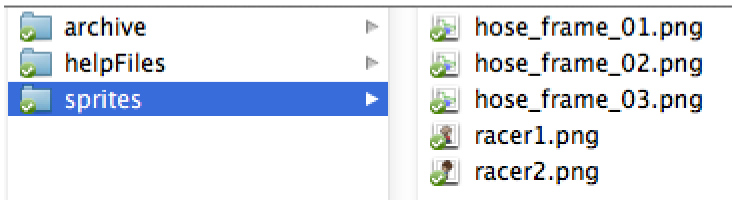


And here we are at the Scene view (hopefully bigger than this image):



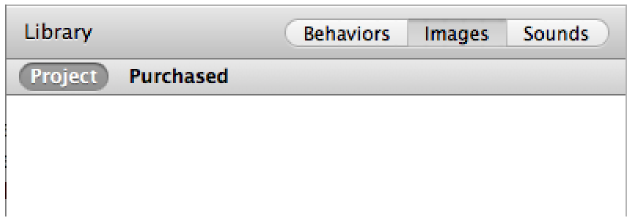
Step 1: Add the images.

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:

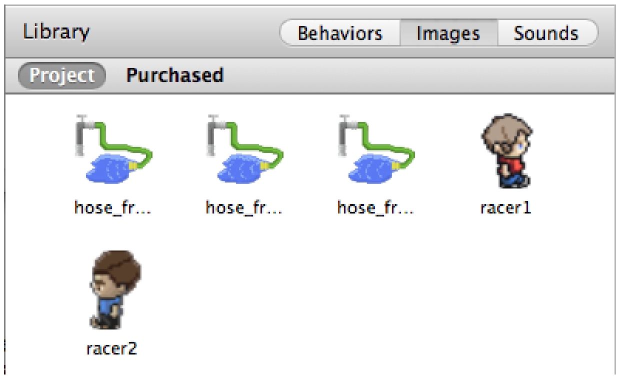


Note that there are three separate "frame" images for the hose. You'll notice each looks slightly different from the others. That's because we are going to animate the hose, which simply means that the image changes during gameplay, frequently enough that it looks like the water is moving. This is analogous to flipping through the pages of a flipbook. Sometimes you may encounter a "spritesheet"--this is a single image containing all the "frames" of an animation, side by side, which some programs can turn into an animation. GameSalad requires the frames to be separate images though, so that is what we are using.

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

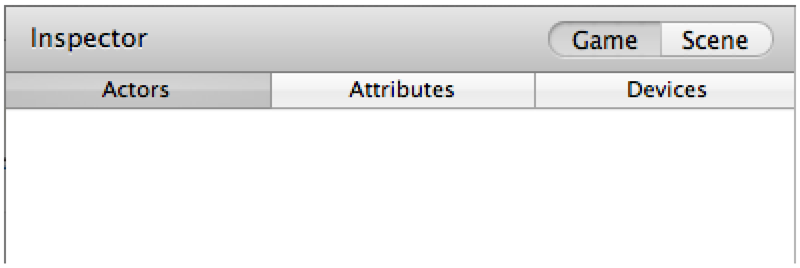


This is where we store our images so we can use them. Adding them to GameSalad is as easy as dragging and dropping them from the folder into the white box. You should end up with this:

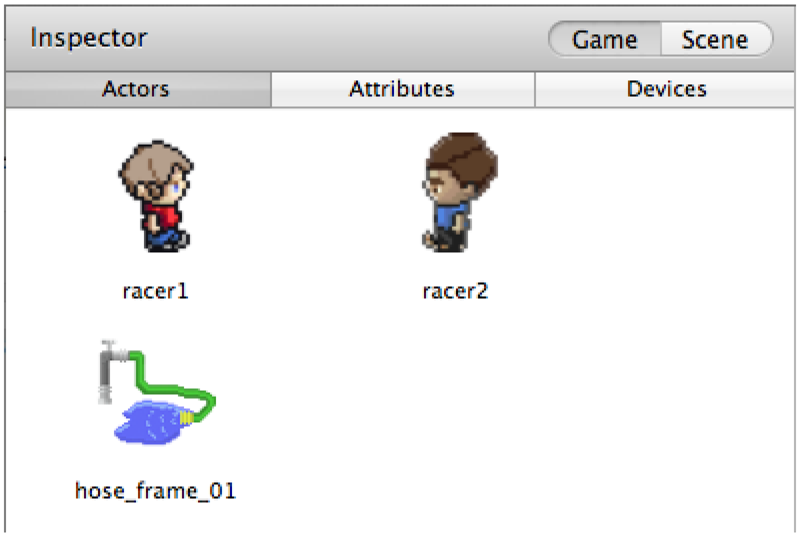


Step 2: Add the Actors.

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":

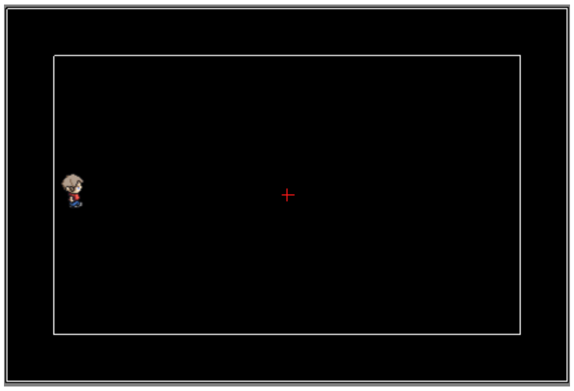


It's easy to make an Actor with an image attached--simply drag the image from the Library into the "Actors" tab of the Inspector. Do so with the two racers and the first frame of the hose (we don't need separate Actors for the other two frames). Now the "Actors" tab should look like this:



Step 3: The first racer.

Now that we have Actors, we can put them in the game! Start with “racer1” by dragging 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 “racer1” somewhere on the left side of the screen:

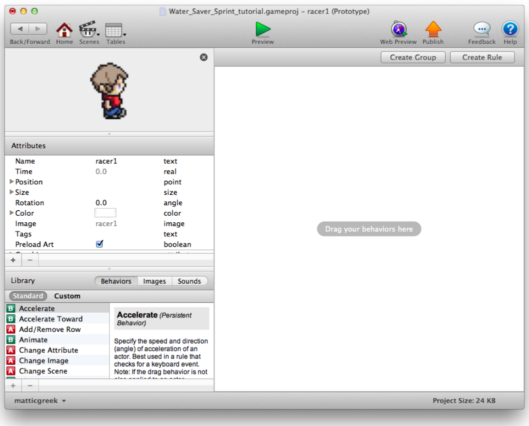


You’ve just created an “instance” of the Actor “prototype.” What does that mean? It means that when you dragged the “racer1” image from the Library into the Inspector, you created a *type* of Actor called “racer1,” which still doesn’t appear in the game until you make a copy of it by dragging it into 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 “racer1,” 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 racer.

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 him how to Act, so “racer1” just stands there. Let’s allow Player 1 to move “racer1” to the right.

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 “racer1” prototype (remember, that means double-click on him in the Inspector, not in the Scene). You’ll see the details of the prototype:

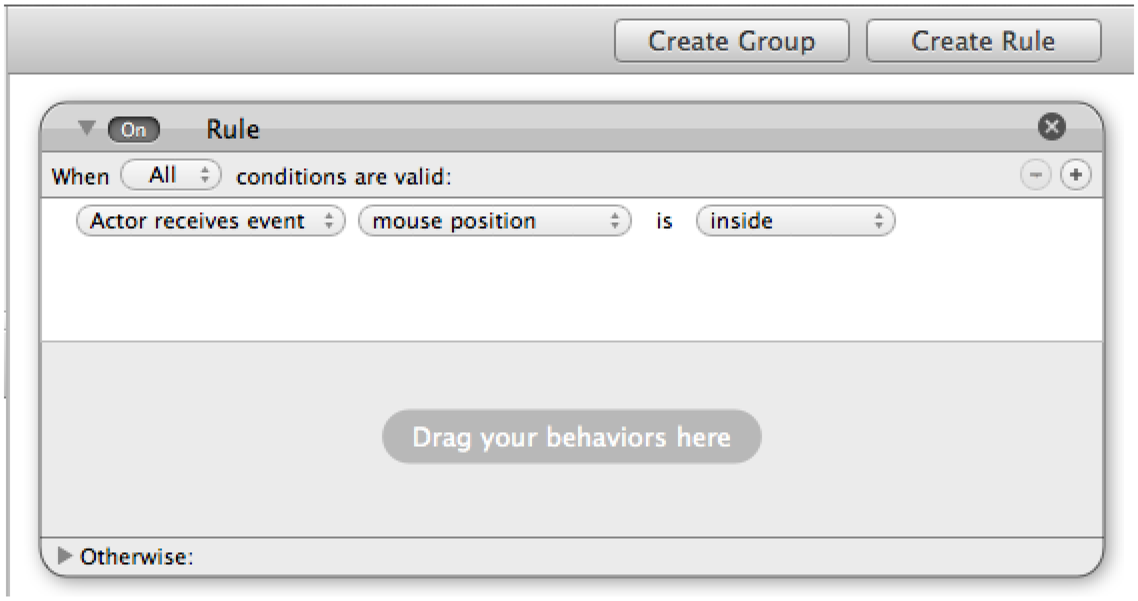


Name	racer1	text
Time	0.0	real
Position		point
Size		size
Rotation	0.0	angle
Color		color
Image	racer1	image
Tags		text
Preload Art	✓	boolean

Accelerate *(Persistent Behavior)*

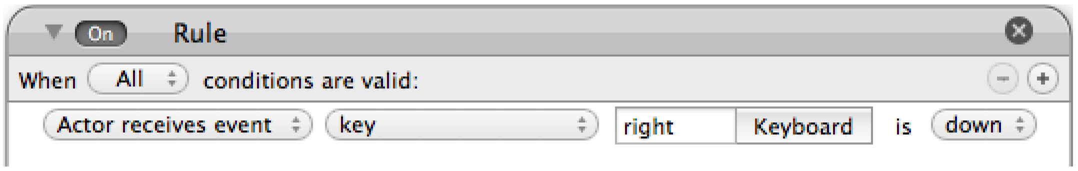
Specify the speed and direction (angle) of acceleration of an actor. Best used in a rule that checks for a keyboard event. Note: If the drag behavior is not

The big empty white box on the right is where we can tell “racer1” what to do. The first thing is to tell him to only act 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:

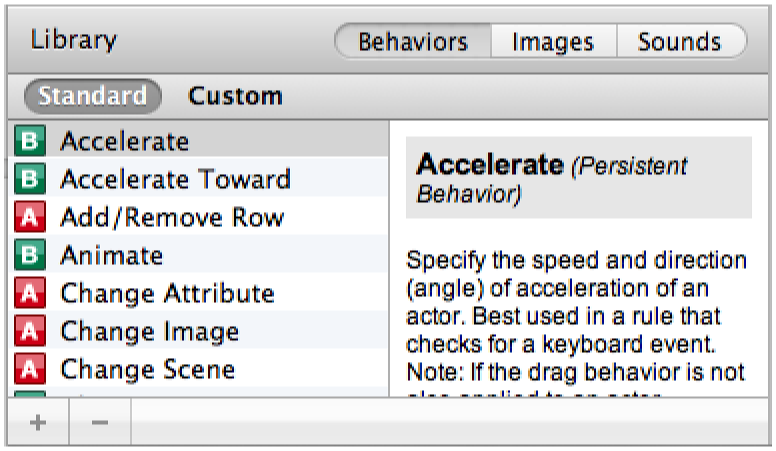


As you can see, a "Rule" is a conditional statement. It tells the game to check if a certain condition is met, and if so, what to do. "Actor receives event" sounds fancy but simply means that the Actor is paying attention to the state of the game and will notice if the condition is met.

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:



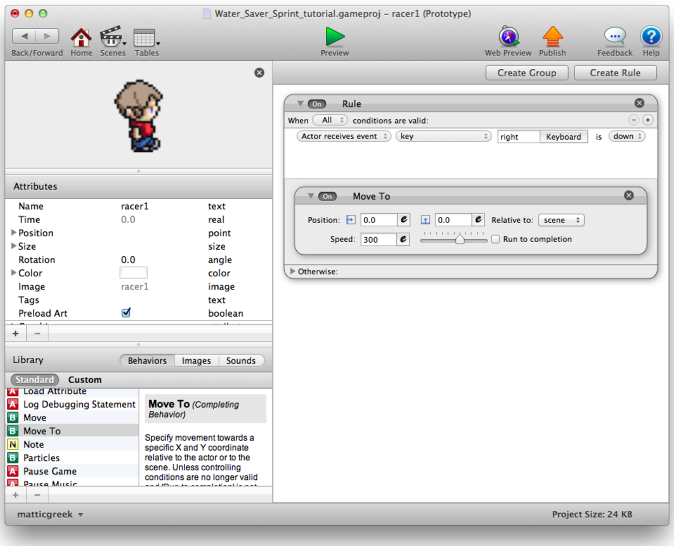
Accelerate *(Persistent Behavior)*

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

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 “racer1,” so look for “move.” You’ll see we have two options: “Move” and “Move To.” Which should we use?

Let’s think about how each Behavior works. The “Move” Behavior moves in a certain direction with a certain speed. “Move To” moves toward a certain point with a certain speed. Direction vs. destination. Consider how we want the movement of “racer1” to work. If you’ve played the game on the website, you know that the racers don’t move like race cars, driving smoothly towards the goal--they move incrementally, once per button press, as though they were taking steps. It makes sense to use “Move To,” since each racer moves to a point one “step” closer to the goal with each press of the button.

To give the “Move To” Behavior to the “racer1” prototype, simply drag it from the list into the Rule we created. Now it looks like this:



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

Move To — Position: 0.0 0.0 Relative to: scene; Speed: 300; Run to completion

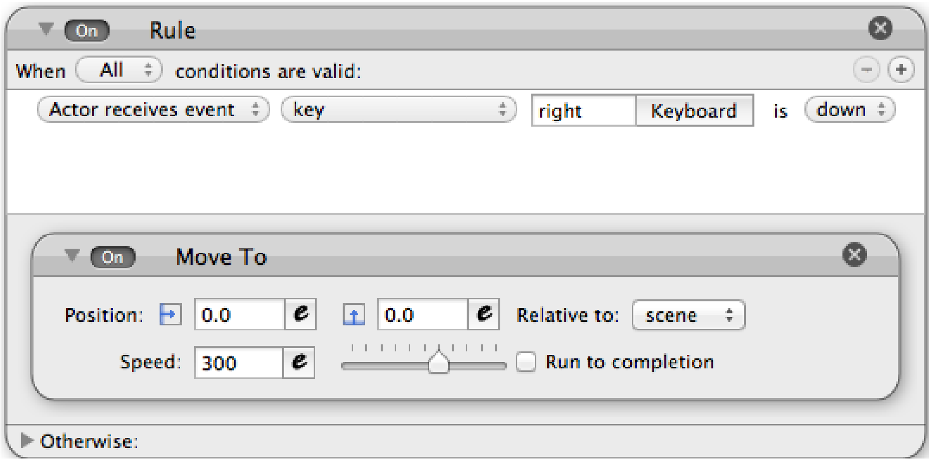
Otherwise:

Name	racer1	text
Time	0.0	real
Position		point
Size		size
Rotation	0.0	angle
Color		color
Image	racer1	image
Tags		text
Preload Art	✓	boolean

Move To *(Completing Behavior)*

Specify movement towards a specific X and Y coordinate relative to the actor or to the scene. Unless controlling conditions are no longer valid

Here's a closer view:



When All conditions are valid:

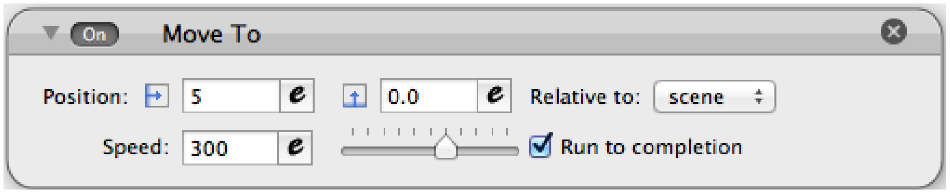
Actor receives event | key | right Keyboard | is | down

Position: 0.0 | 0.0 | Relative to: scene

Speed: 300 | Run to completion

We need to set the position and speed. Just to the right of "Position" is an arrow pointing to the right; this indicates the coordinate on the x-axis, running from the left edge of the screen to the right. To its right is an arrow pointing upward, which indicates the coordinate for the y-axis, running from the bottom edge of the screen to the top.

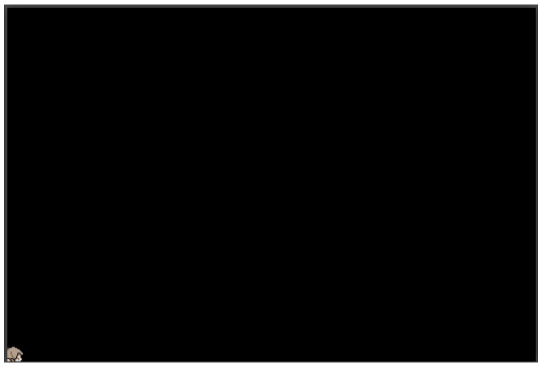
Let's make "racer1" move 5 units to the right. Change the x-coordinate from "0.0" to "5." We don't need to move vertically so leave the y-coordinate at "0.0." Check the "Run to completion" box, just in case--this simply means that if the Rule's condition is no longer met but the Actor hasn't finished the Behavior (i.e. hasn't gotten to the destination), the Behavior will complete anyway.



Position: 5 | 0.0 | Relative to: scene

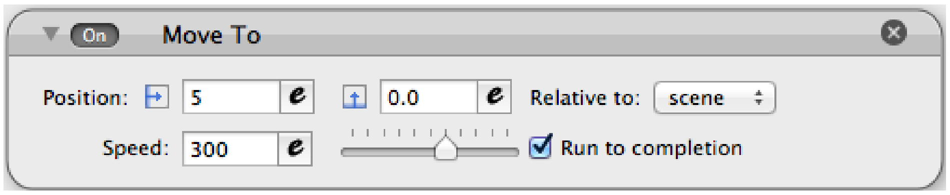
Speed: 300 | ✓ Run to completion

OK, let's try it! Click the Preview button and press the RIGHT arrow key.



What happened?! He moved all the way to the lower-left corner. That's not what we wanted. What did we do wrong?

Well, we need to change one more thing in the Behavior. Let's look at it again:



Position: 5 0.0 Relative to: scene
Speed: 300 ☑ Run to completion

To the right of the condition, we see "Relative to: scene." This means that the coordinates we specified for the destination, (5, 0.0), refer to the coordinate point (5, 0.0) in the Scene itself, which is just 5 units to the right of the origin point, (0, 0), in the lower-left corner. In other words, we told "racer1" to move to the location (5, 0.0) in the Scene. But what we want is to move 5 units to the right of wherever the Actor is, which means our movement is relative to the *Actor*, not the Scene. Click on "scene" and change it to "actor." Preview the game again.

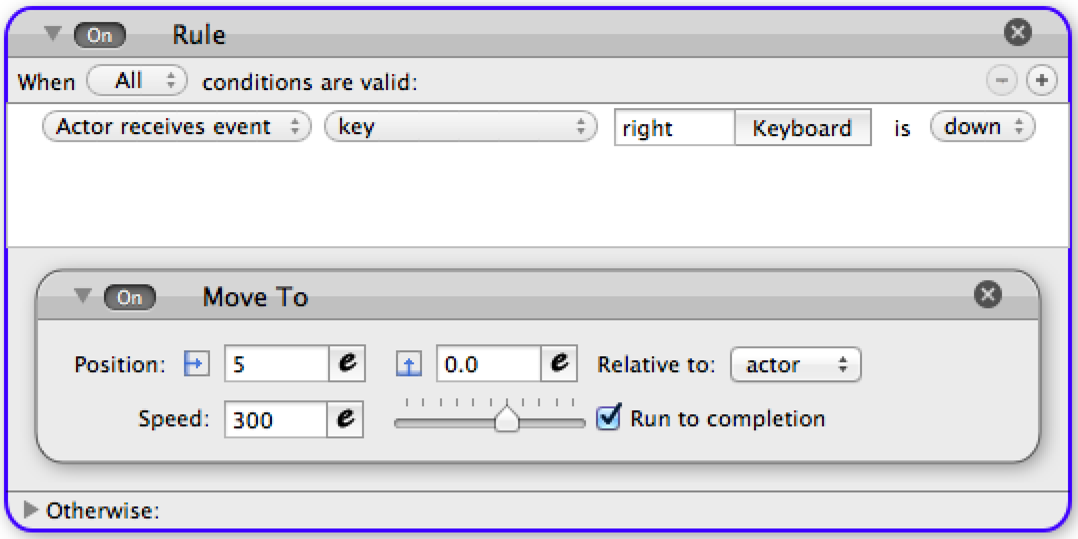
It works! When we press the RIGHT arrow key, "racer1" moves 5 units to the right. If we release the key and press it again, he moves another 5 units. If we tap the key, he races forward. This is exactly the effect we wanted.

Take a moment to think about how this differs from what would have happened if we had used the "Move" Behavior instead of "Move To." Since "Move" specifies a direction instead of a destination, we could press and hold the key and "racer1" would zoom across the screen. But we want the player to have to tap the button, so we chose "Move To," which allows us to limit how much "racer1" moves with each press of the button.

Step 5: The second racer.

Whew! It took some effort to get that first racer working properly. Now we want to add a second racer, controlled by a different player and moving the other way. This is going to be hard, huh?

Actually, no! Quite the opposite. If you think about it, the second racer will behave almost identically to the first--the only differences are the direction and key to press. That means we can reuse all the work we already did, using our friends Copy and Paste! Return to the prototype view for “racer1” and click the Rule to select it, indicated by the blue highlight:



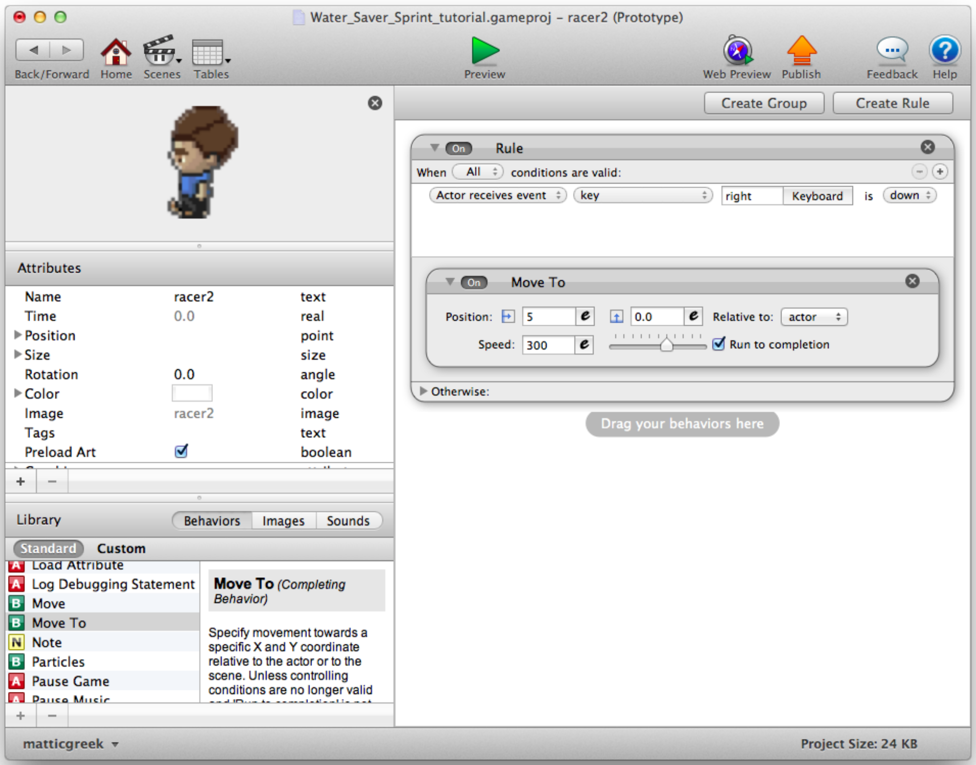
When All conditions are valid:

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

Position: 5 | 0.0 | Relative to: actor

Speed: 300 | ✓ Run to completion

You should have the usual set of menus (“File,” “Edit,” “View,” etc.) across the top of the screen. Choose “Edit,” then “Copy.” Now, click the Back button to return to the Scene view, then double-click on the “racer2” prototype. Click inside its empty Behavior window, then choose “Edit,” then “Paste.” You should see this:



Attributes

Name	racer2	text
Time	0.0	real
▸ Position		point
▸ Size		size
Rotation	0.0	angle
▸ Color		color
Image	racer2	image
Tags		text
Preload Art	✓	boolean

Library — Behaviors | Images | Sounds — Standard | Custom

- A Load Attribute
- A Log Debugging Statement
- B Move
- B Move To
- N Note
- B Particles
- A Pause Game
- A Pause Music

Move To *(Completing Behavior)*

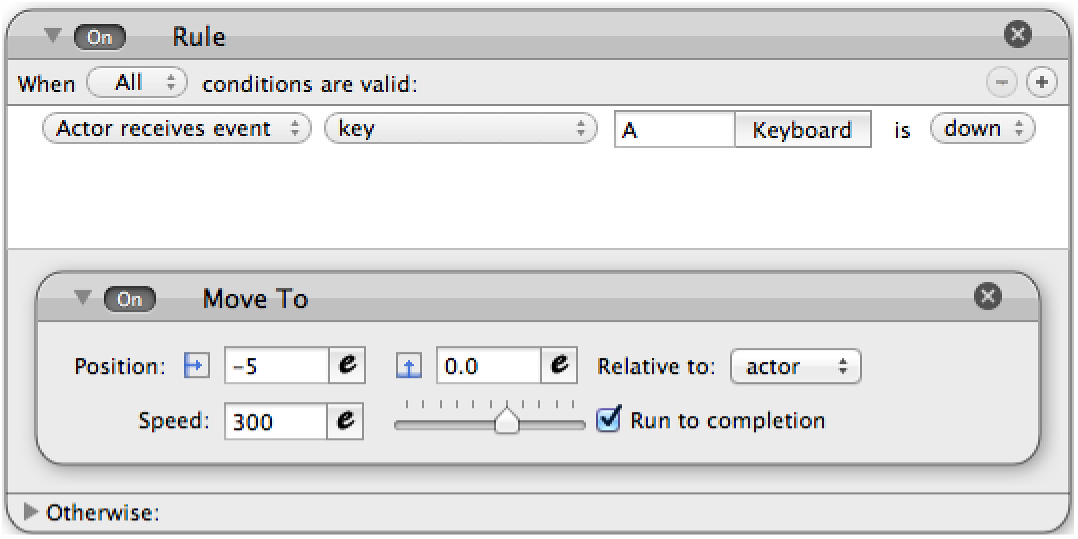
Specify movement towards a specific X and Y coordinate relative to the actor or to the scene. Unless controlling conditions are no longer valid

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

On Move To
Position: 5 | 0.0 | Relative to: actor
Speed: 300 | ✓ Run to completion

▸ Otherwise:

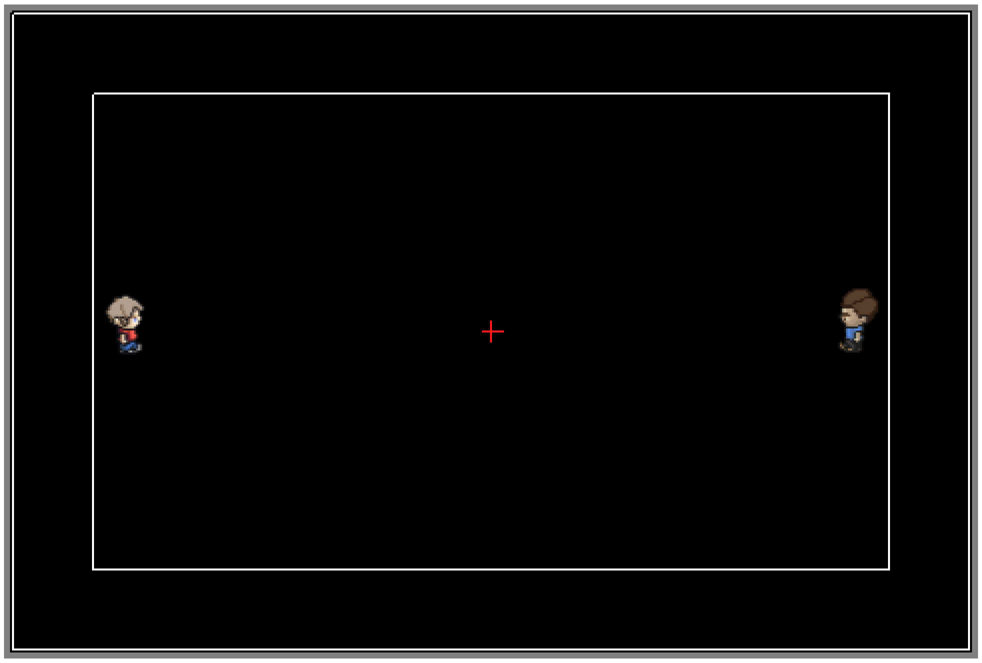
Alright! It’s the same as “racer1”! Simply change the key in the Rule to “A,” and change the x-coordinate in the “Move To” Behavior to “-5.” Since the x-axis increases going to the right, making the coordinate negative moves to the left. Now the Behavior should look like this:


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

On Move To
Position: -5 | 0.0 | Relative to: actor
Speed: 300 | ✓ Run to completion

▸ Otherwise:

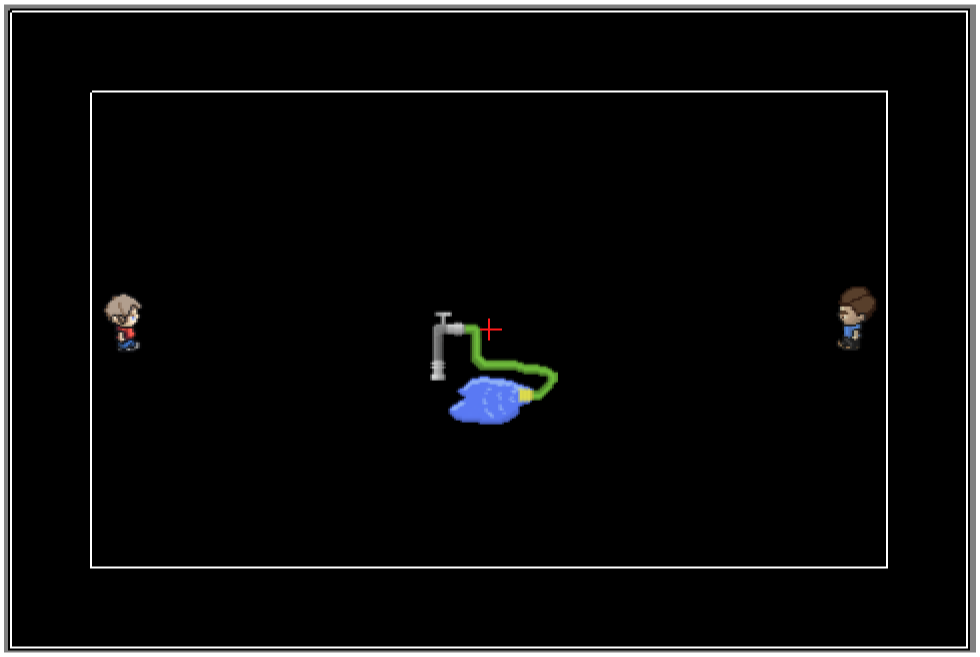
Wait, we haven’t added the Actor to the Scene yet! Click “Back” to return to the Scene view, then drag the “racer2” prototype into the Scene to create an instance. Drop him on the right side:



Preview the game. Success! Tapping RIGHT moves “racer1” and tapping “A” moves “racer2”! Put one person on each button and you have multiplayer!

Step 6: The hose.

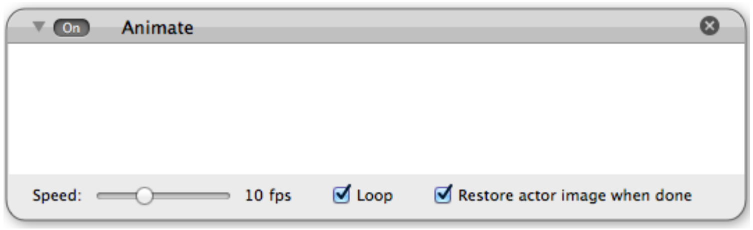
It’s still not really a game, though--there’s no goal! Right now those racers are racing toward a head-on collision. We can’t have that, so it’s time to add the hose. Speaking of which, the prototype is currently named ““hose_frame_01,” which is very cumbersome, so click on the name and change it to “hose.” Now, drag the prototype into the Scene to create an instance and drop it in the center, between the two racers:



The water is supposed to be flowing (turning it off is the point of the game!) but that pool of water isn't very convincing. It just sits there, even if we run the game. We need to animate it!

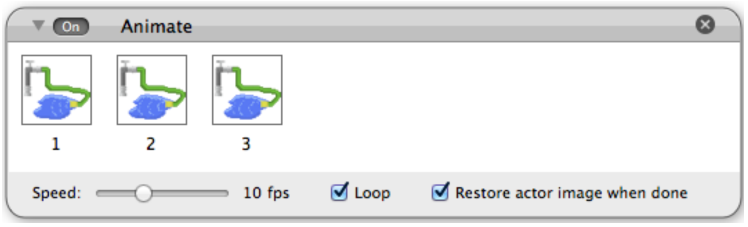
Animation is just motion. With the racers, we created the illusion of motion by changing the position of the image whenever the player presses the button. With the hose though, it stays in the same place. So, instead of moving the image to animate it, we need to *change the image*. We have three images, all very similar but with slight differences in the ripples in the pool of water. By quickly changing from one image to the next during gameplay, we trick our eye into thinking the water is flowing.

That sounds complicated! Good news: it's not. GameSalad does the work for us. Double-click on the "hose" prototype so we can give it some Behavior. Guess which one? If you guessed "Pause Game," you're wrong! Why would you guess that? No, the one we need is…"Animate." Drag it into the Behavior window for "hose."



Speed: 10 fps ✓ Loop ✓ Restore actor image when done

The blank space is for us to indicate what images we want to use. Click on the "Images" tab in the Library, then drag the three frames of the hose animation into the Behavior. Voilà:


Animate

1 2 3

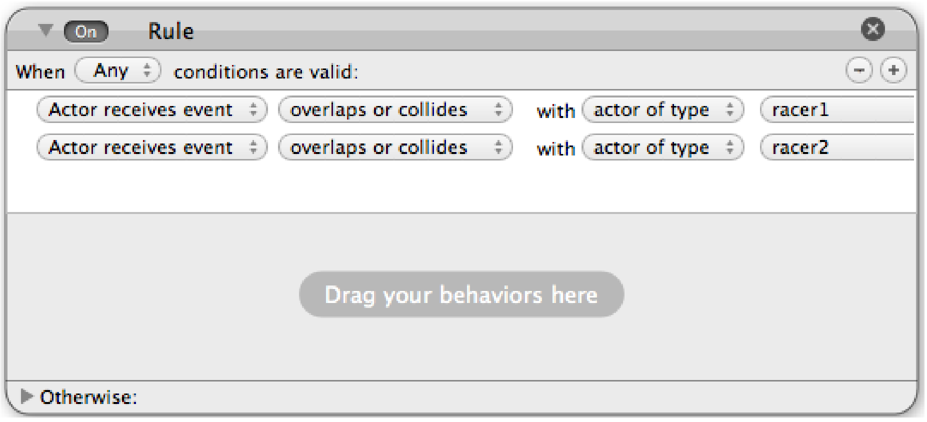
Speed: 10 fps ✓ Loop ✓ Restore actor image when done

We don't need to adjust any of the settings. Preview the game. Cool! The water is now animated!

Step 7: Turn off the water.

There's one more thing we need to do, and that's to enable the players to turn off the water. We'll create that effect by stopping the animation when one of the players touches the hose. That means we need to detect a collision between racer and hose.

Still in the "hose" prototype's Behavior, click "Create Rule." Change "mouse position" to "overlaps or collides" and set the "actor of type" to "racer1." Then, click the plus button in the upper-right of the Rule to create another condition, and set it up the same way but for "racer2." Finally look in the upper-left of the Rule and change "All" to "Any." This is necessary because we want to stop the animation if either racer touches the hose. Now the Rule looks like this:

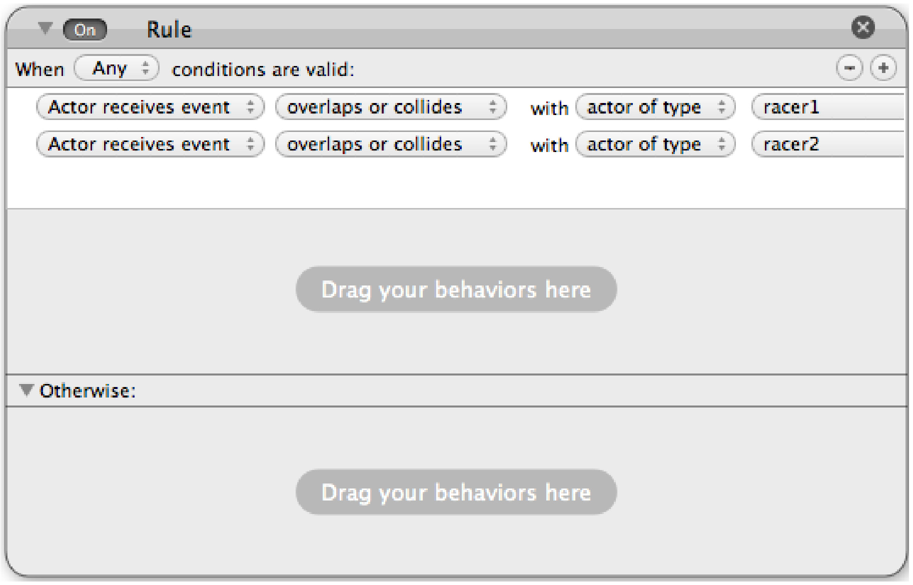

Rule

When Any conditions are valid:

Actor receives event · overlaps or collides · with actor of type · racer1

Actor receives event · overlaps or collides · with actor of type · racer2

OK, but we want to use this Rule to stop the animation. Is there a "Stop Animation" Behavior? No, there is not. How do we do this? Well, we're actually going to do something strange, which is not to put any Behavior in the Rule...sort of. Instead, we're going to use the "Otherwise" at the Rule's bottom. Click the arrow to the left to expand it. Oh, look! A second place to put Behavior:

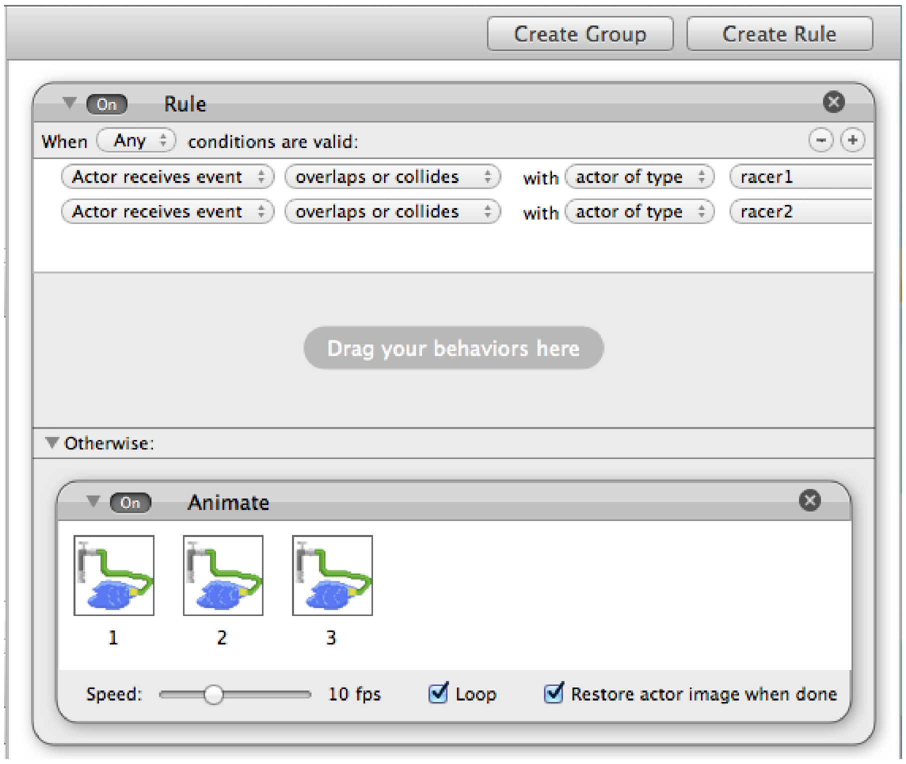

Rule

When Any conditions are valid:

Actor receives event overlaps or collides with actor of type racer1

Actor receives event overlaps or collides with actor of type racer2

Drag our “Animate” into this space. The hose’s entire Behavior should now look like this:



When Any conditions are valid:

Actor receives event overlaps or collides with actor of type racer1

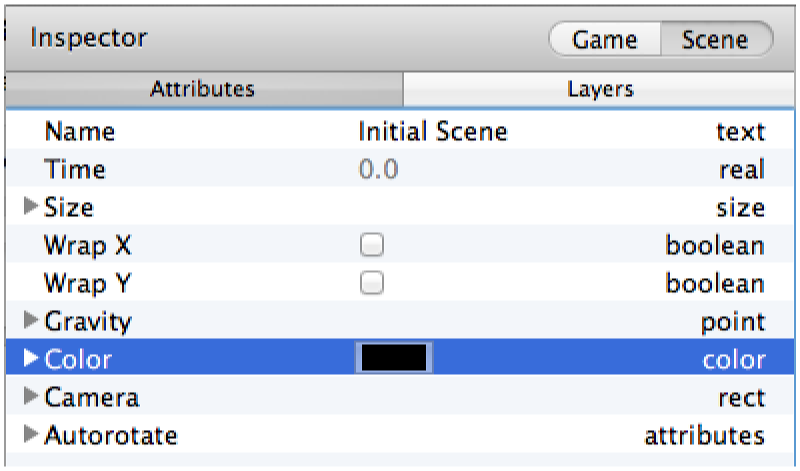
Actor receives event overlaps or collides with actor of type racer2

Speed: 10 fps ✓ Loop ✓ Restore actor image when done

Think about what this says. It checks for a collision with either racer, and if it sees one, it does nothing; otherwise, it animates the hose. The hose only animates when the racers haven’t reached it, so when one does reach it, that has the effect of stopping the animation. Easy!

Step 8: The background.

Up to this point we've left the background black, but the hose is supposed to be watering the yard, so let's change the background to be grass-green. This couldn't be easier. Return to the Scene view, then click the "Scene" tab in the upper-right of the Inspector:

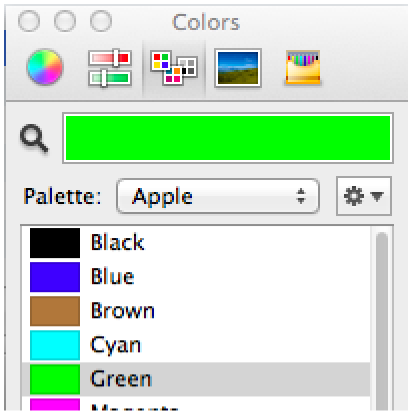


Inspector — Game | Scene

Attributes | Layers

Name	Initial Scene	text
Time	0.0	real
▶ Size		size
Wrap X	☐	boolean
Wrap Y	☐	boolean
▶ Gravity		point
▶ Color	■	color
▶ Camera		rect
▶ Autorotate		attributes

Click the black rectangle next to "Color" to bring up the "Colors" pop-up, then click the middle tab (Hover over it to see "Color Palettes") and choose "Green."

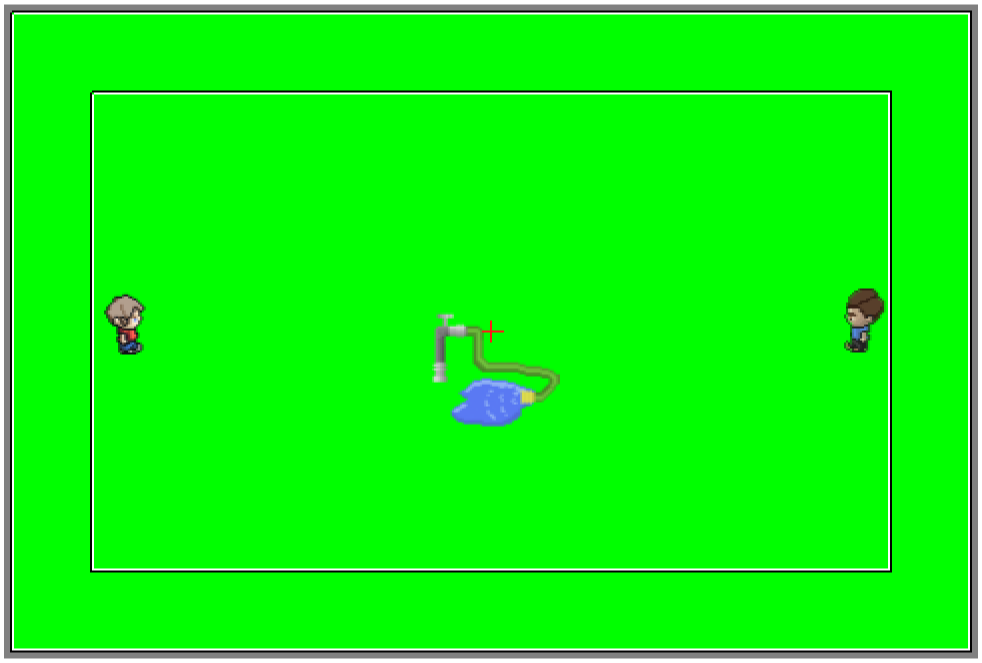


Colors

Palette: Apple

⬛	Black
🟦	Blue
🟫	Brown
🟦	Cyan
🟩	Green

That looks much better:



Victory! What now?

Hey, that's it! We've got rules (move toward the hose), a goal (reach the hose), challenge (get there before the other player), and feedback (the hose "shuts off" when a player reaches it)--we're done! Great job! You've made Water Saver Sprint!

What now? Playtesting is an important part of game design and you should definitely ask people to try the game out. At the least you should ensure that it takes the same number of button presses for either racer to reach the hose--otherwise the racer who is closer starts out with an advantage.

Think about how you might improve the game further. For one thing, we haven't created any kind of victory message when a player wins. Many races incorporate multiple rounds; how about best two out of three? Perhaps the player has to press another button upon reaching the hose to actually turn it off? Maybe the hose's position changes between rounds, and the racers can move vertically as well?

With this tutorial you've learned the basics of game creation in GameSalad. You even realized that "multiplayer" doesn't have to be any more complicated than asking two people to use different buttons on the same keyboard. Feel free to build this game further, work through the other tutorials in the curriculum, and start your own projects. The possibilities are limitless!