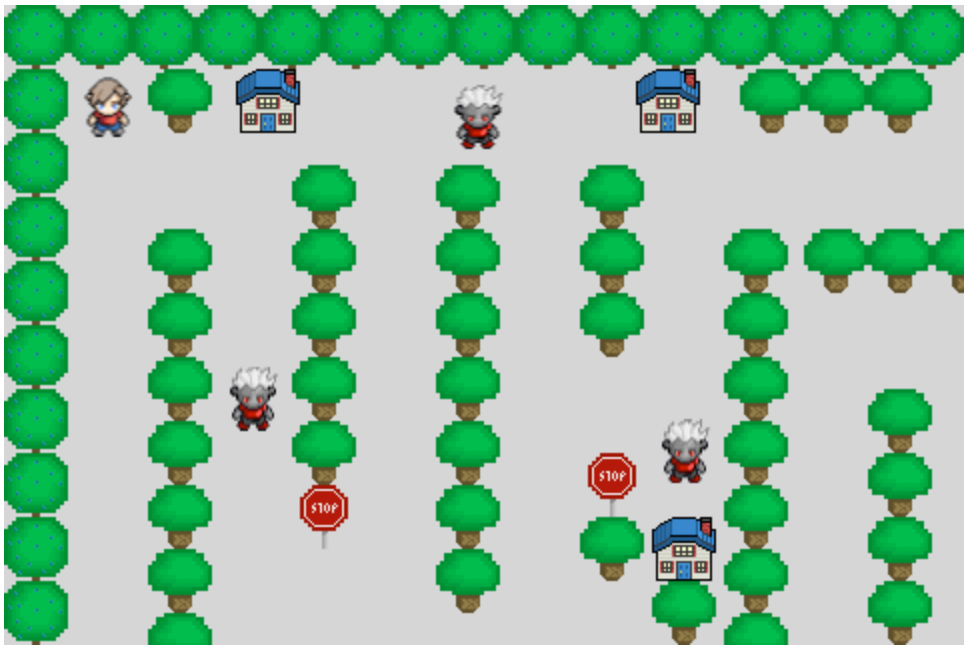
How to Make “Lights Off” in GameSalad

by J. Matthew Griffis

*Note: this is an Expert-level tutorial. It is serious business. If you have not already read the separate PDF **GameSalad Basics** and done the other tutorials, you are asking for trouble.*



In Lights Off, your neighbors have become used to leaving their lights on all day long, so it is up to you to help them change their behavior and learn how to use energy more efficiently. In order to do this, you must spread the word through meeting houses your neighbors visit. But be careful! As you walk around the neighborhood you must avoid the neighbors who are still using too much energy. Their bad habits have transformed them into monsters who cannot control their desire to keep using energy. Can you save your neighbors by convincing them to act more responsibly and turn off all the lights?

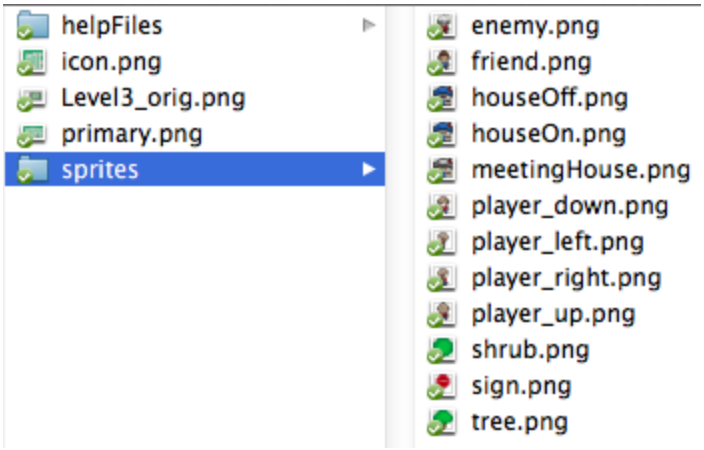
Lights Off uses more-complex interactions than the previous games in this series, and serves as a great introduction to creating more-sophisticated games in GameSalad. In this tutorial, you’ll learn how to recreate Lights Off. Make sure to download the folder of Resource Files for the game, and play the game on the website to see it in action.

OK! Let’s get started.

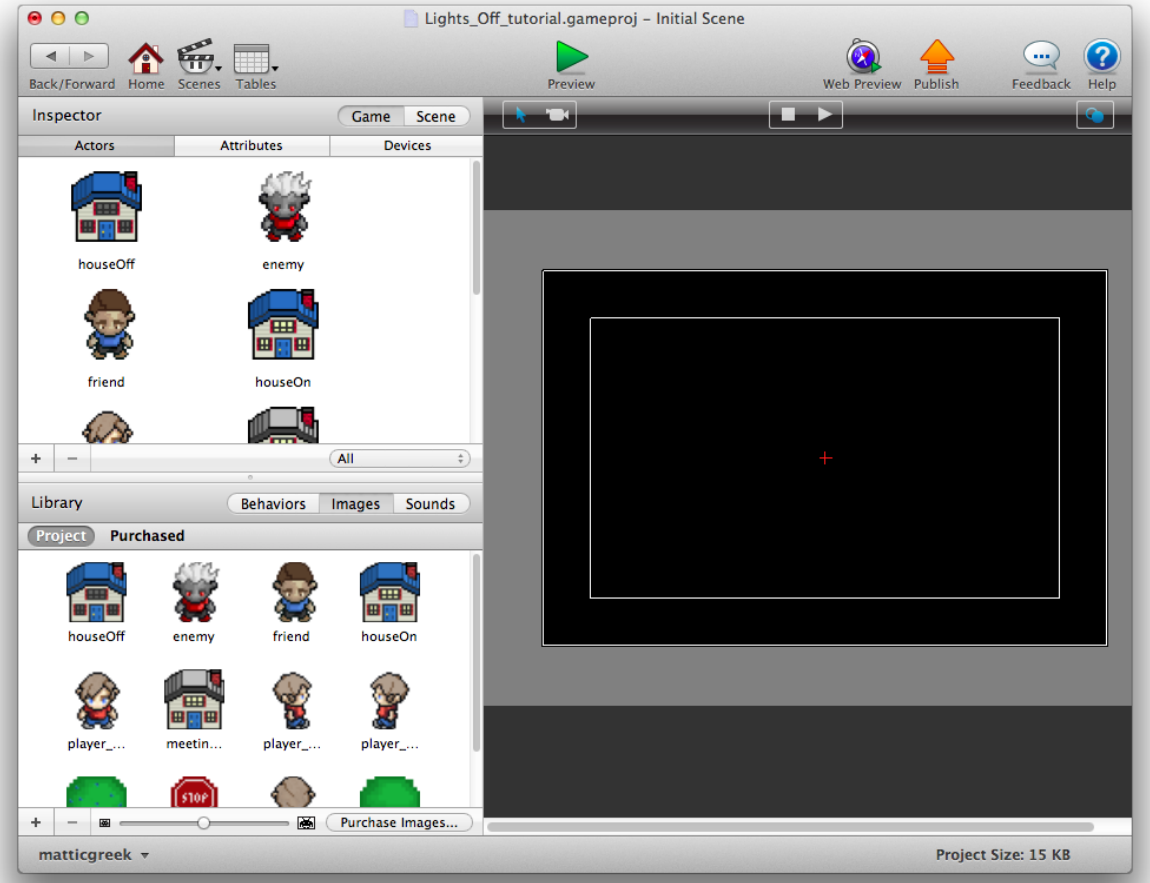
Create a blank project in GameSalad and fill in the Project Info. Choose whatever Platform you like, but remember that only GameSalad Arcade can be published online (so that’s what this tutorial uses). Now, open up the first Scene.

Step 1: Add the images and Actors.

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:

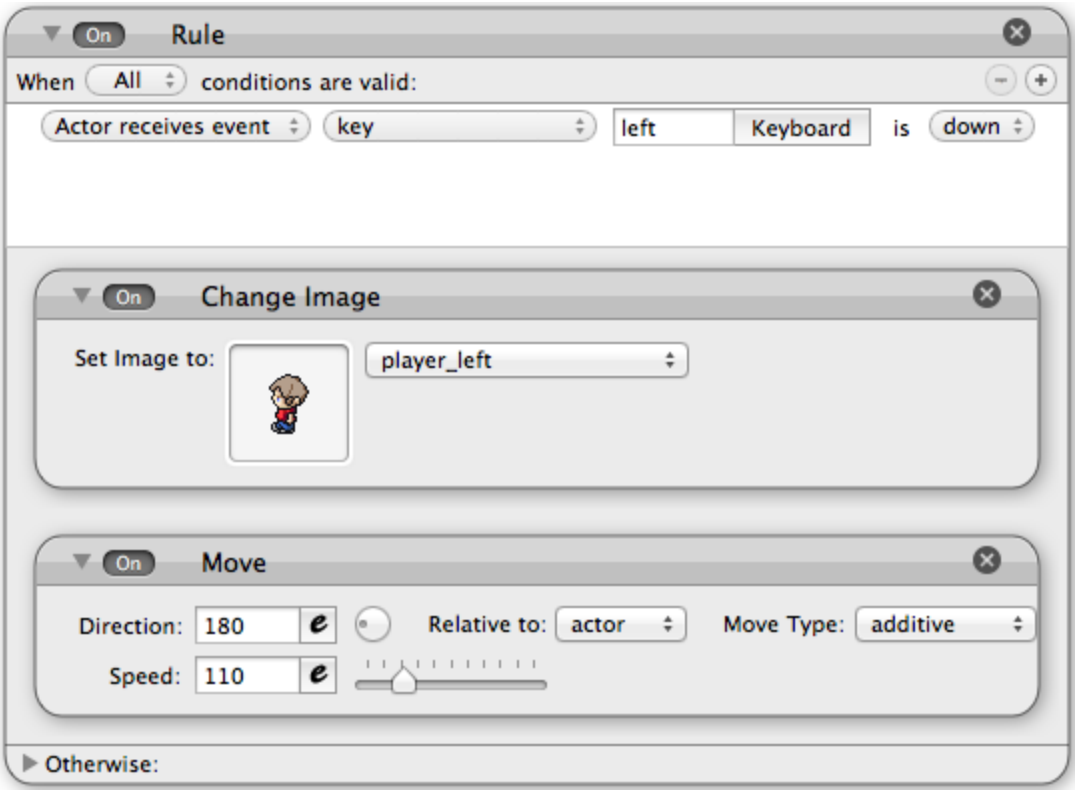


We’ll need all of them, so drag them all into the “Images” tab of the Library to import them into GameSalad. Then, drag the imported images from the Library into the “Actors” tab of the Inspector to make Actors with the images attached. Note that we we have four images for the player but only need one Actor, so use “player_down.png” to create it. The whole thing should now look something like this:



Step 2: Set up the player.

Drag the player Actor into the Scene to create an instance. Let's set it up so we can move it with the arrow keys. This should all be very familiar by now. Double-click on the player prototype, then create a Rule for pressing the left arrow key, and set the player to move to the left. Note that we also have a different image for the player facing left. So, add the "Change Image" behavior to the Rule and choose the left-facing player image. The Rule should look like this:

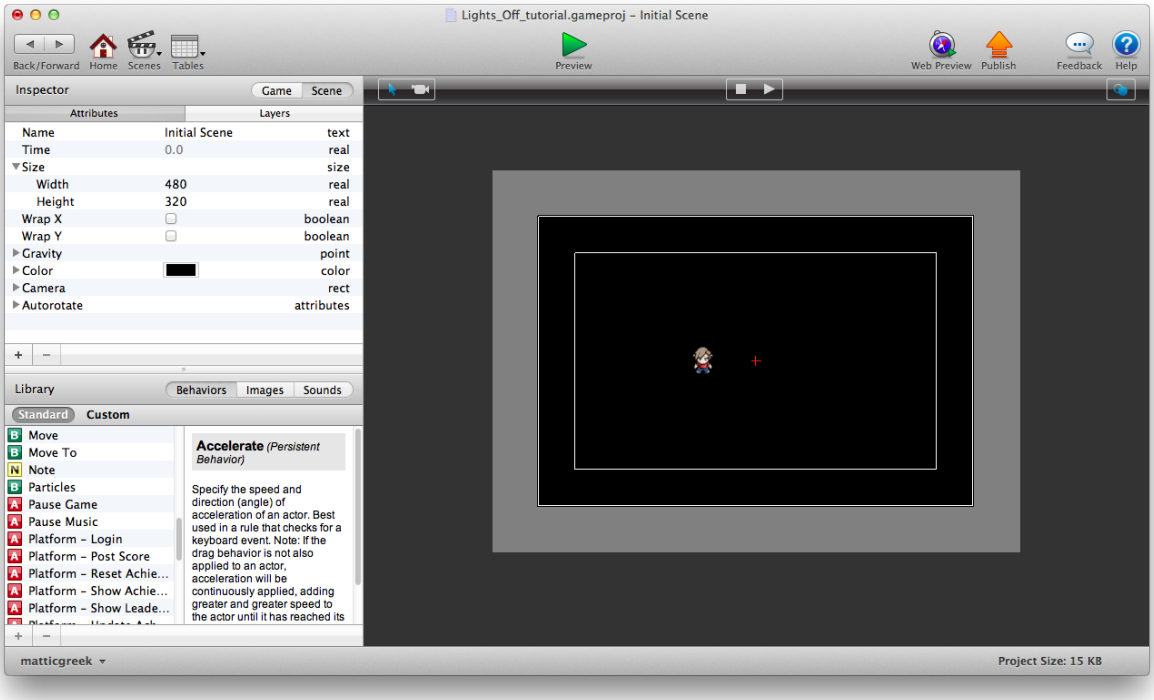


Add similar rules for the other three arrow keys. The upward direction is 90 (degrees), right is 0 and down is 270. Preview the game and test the controls to make sure they work.

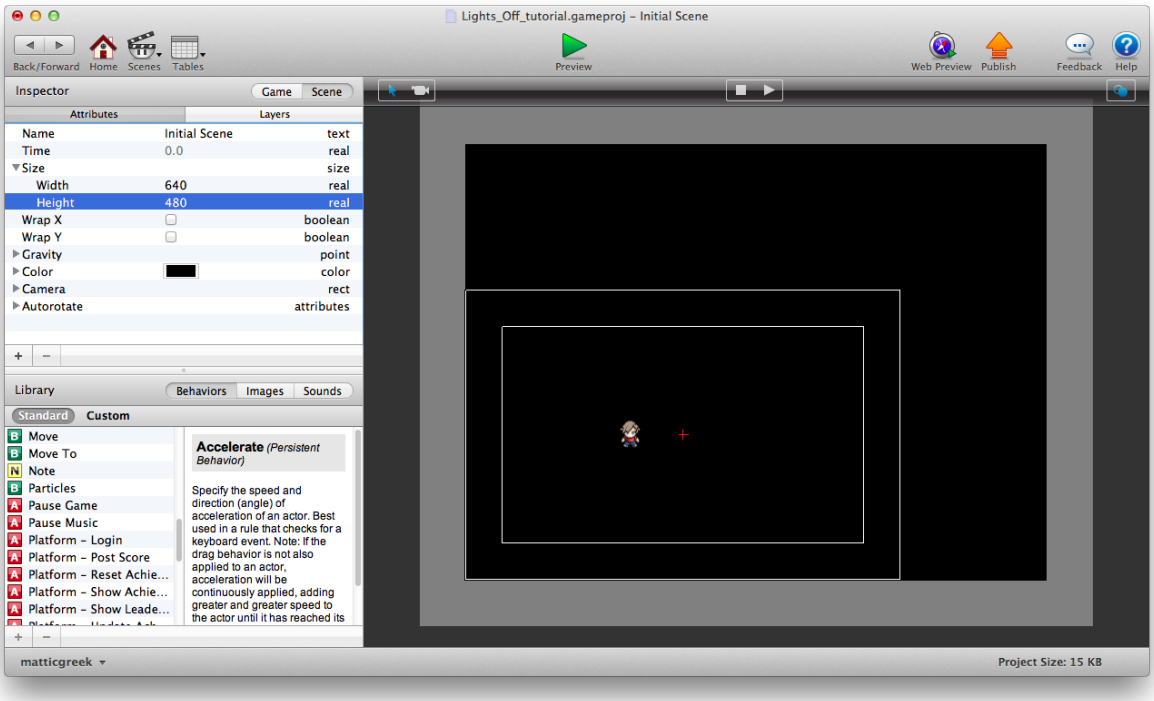
Step 3: Set up the camera.

We're going to do something special with this game, which is to only show part of the Scene at a time. That means that the entire Scene will be larger than what shows on-screen--in fact, the Scene can be as large as we want, regardless of our Platform screen size. How do we do this? We will use what's called a "camera view," meaning that the game screen is like the view of a camera that follows the player. Moving the player to the edge of the screen will cause the camera to scroll in that direction, revealing more of the Scene that way while hiding things in the opposite direction.

This is very easy to set up in GameSalad. The first thing is to increase the size of the Scene, which by default is the same size as the Platform screen size. To do this, return to the Scene view, then click the "Scene" tab in the upper-right of the Inspector. It should default to the "Attributes" sub-tab. There's an entry for "Size," which we can expand. Here's how the Scene looks at the default GameSalad Arcade size of 480x320:



Let’s increase the Width to 640 and the height to 480. Double-click on the values to edit them.



That's more like it! The black itself is the entire Scene, while the duo of rectangles inside it reflects the camera, which defaults to the game screen size. We can see that too by expanding the Camera attribute, and its internal Size and Tracking Area attributes:

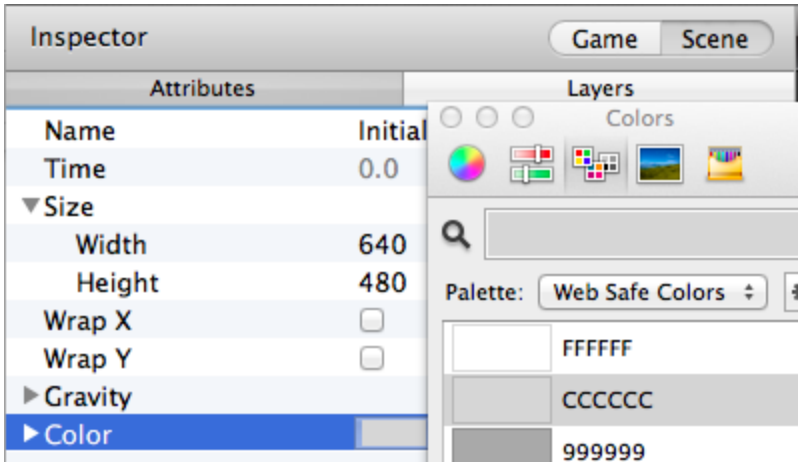
▼Camera		rect
▶Origin		point
▼Size		size
Width	480	real
Height	320	real
▼Tracking Area		size
Width	400	real
Height	240	real
Rotation	0.0	real

We want the camera's size to be the same as the screen, so we leave that alone. Take note of the "tracking area" (which is the internal of the two rectangles inside the Scene). This controls when the camera actually moves. The player has to reach the bounds of the tracking area before the camera starts scrolling. The default size of 400x240 is too big for our needs--there's not a lot of distance between the edges and the edges of the camera itself. Change the width to 240 and the height to 144 (this is the same ratio as the camera size). Of course feel free to tweak this and find what feels right to you through playtesting. The revised size:

Of course, none of this will do anything until we actually attach the camera to the player. This is as simple as adding the “Control Camera” behavior to the player’s Behavior. At this point you can try running the game, but it will be difficult to tell if the camera is working, since there’s nothing else in the Scene to be a reference point. So, add a shrub or something so there’s something to compare the player’s position to, then try moving about to see the camera in action.

Step 4: Set the background.

Before we do anything else, let’s set the background color to something other than black. To do so, return to the Scene’s attributes list, but this time click on the “Color” box and choose a color, e.g. “CCCCCC” (a lovely shade of gray) from the “Web Safe Colors” palette:



Important note: any changes made to a Scene’s attributes must be made again for any additional Scene(s). That means the background color, but also the dimensions of both the Scene itself and the camera (which we established in the previous step).

Step 5: Make the player collide with objects.

Besides the player and the enemies/friends, Lights Off has many inanimate objects, including shrubs, trees, signs, and three varieties of houses. Some of them are nothing more than obstacles (shrubs, trees and signs) while others interact with the player (houses), but the player shouldn’t be able to move through any of them, so let’s set up collision with those objects.

//

Notes:

Enemy and building logic:
NPCs bounce between buildings as well as obstacles.
If house on and player collides and house blue, unchanged.
If house on and player collides and house gray, house off (player must press against house).
If house on and enemy collides (gray or blue), unchanged.
If house on and ally collides, house off.
If house off and enemy collides, enemy-->ally.
If enemy collides with player, game restarts.
If ally collides with player, unchanged.

COLLISION IS A HUGE PAIN IN THE BUTT. Have to provide more width than the sprite or else you get brush-against collisions (play havoc with enemies but also player movement). Also need to offset objects like the house to ensure the player must move in that direction to trigger the collision.

72/40 px between columns. 80/48px between rows ONLY ON LEVEL 1 THO.
Have to offset dudes by 8px to prevent excess collision with adjacent objects. E: Thanks, GameSalad.