GameSalad Basics

by J. Matthew Griffis

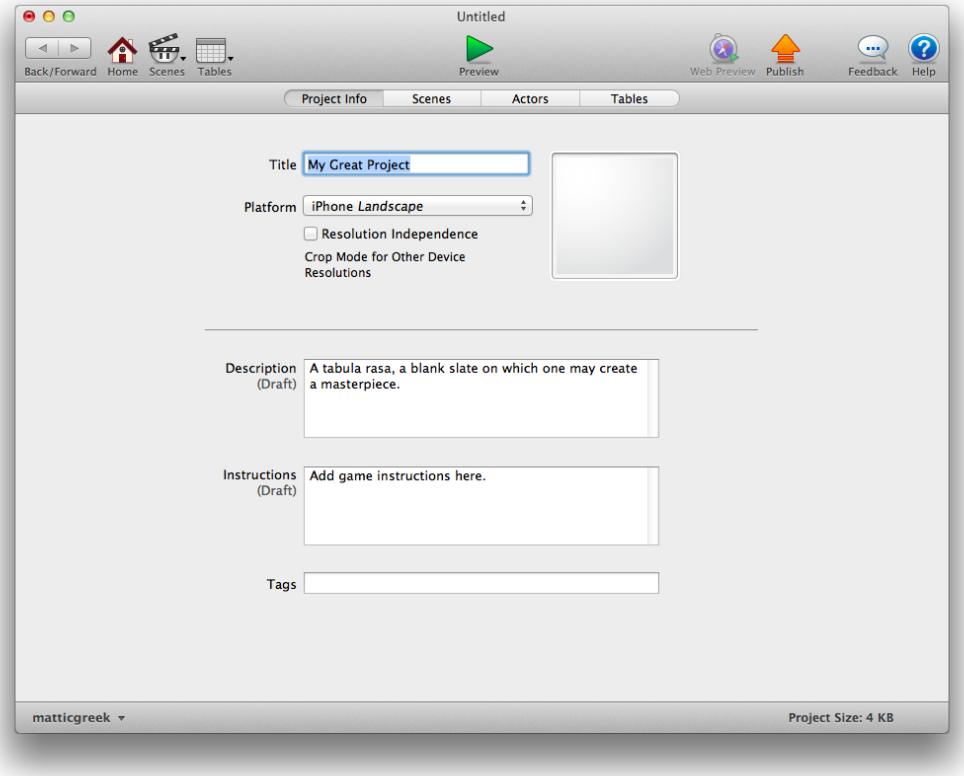
[Click here to jump to Tips and Tricks!]

General usage and terminology

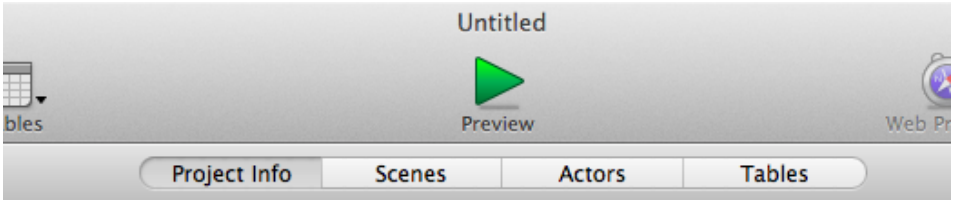
When we first open GameSalad we see something like this:

Templates: GameSalad includes templates (pre-built versions of various types of games), which we can use as the foundation of a new project. We select from the templates--or start from scratch by clicking “Blank Project”--under “Create New Project.” Studying these templates is a great way to learn how to achieve various effects.

Once we open or begin a project we see the main interface:

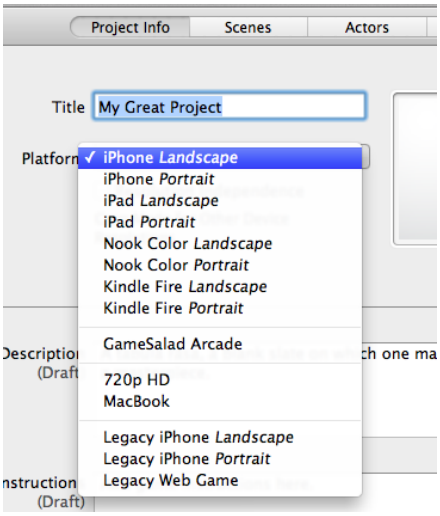


Navigation tabs: at the top of the interface when we first open GameSalad, just below the large green arrow to Preview the game, are four tabs: Project Info, Scenes, Actors, and Tables:



For the purpose of the Activate! curriculum we ignore Tables, which is a more advanced feature.

Project Info: we set the name and description of our project here. Most importantly, we choose a platform. “Platform” refers to the type of device on which we expect people to play our game, and sets the size of our game window to match the device’s screen:



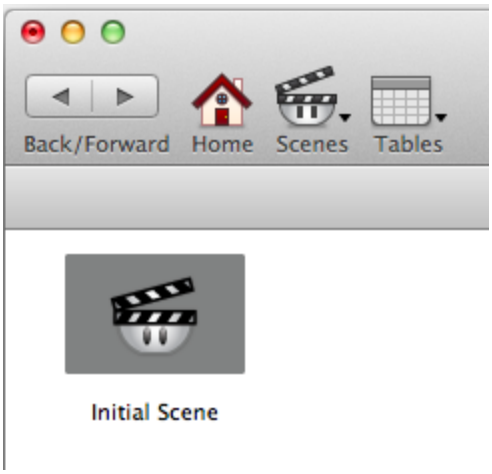
If we want to publish our game online (which uses HTML5 to embed the game in a webpage), we must choose “GameSalad Arcade,” which itself sets the game window to a specific size. All Activate! games use the GameSalad Arcade platform.

Scenes: “scenes” don’t have a defined nature in GameSalad, but it might be easiest to think of them as the different game screens and “levels” that make up a game. For instance we might make our game’s title screen the first scene, then make the first gameplay level the next scene, followed by a menu as the third scene, etc. But there’s nothing stopping us from making the title screen transition into the first gameplay level, all within the first scene. A scene is a canvas - it’s up to us how to use it. For this curriculum each scene is a level of gameplay.

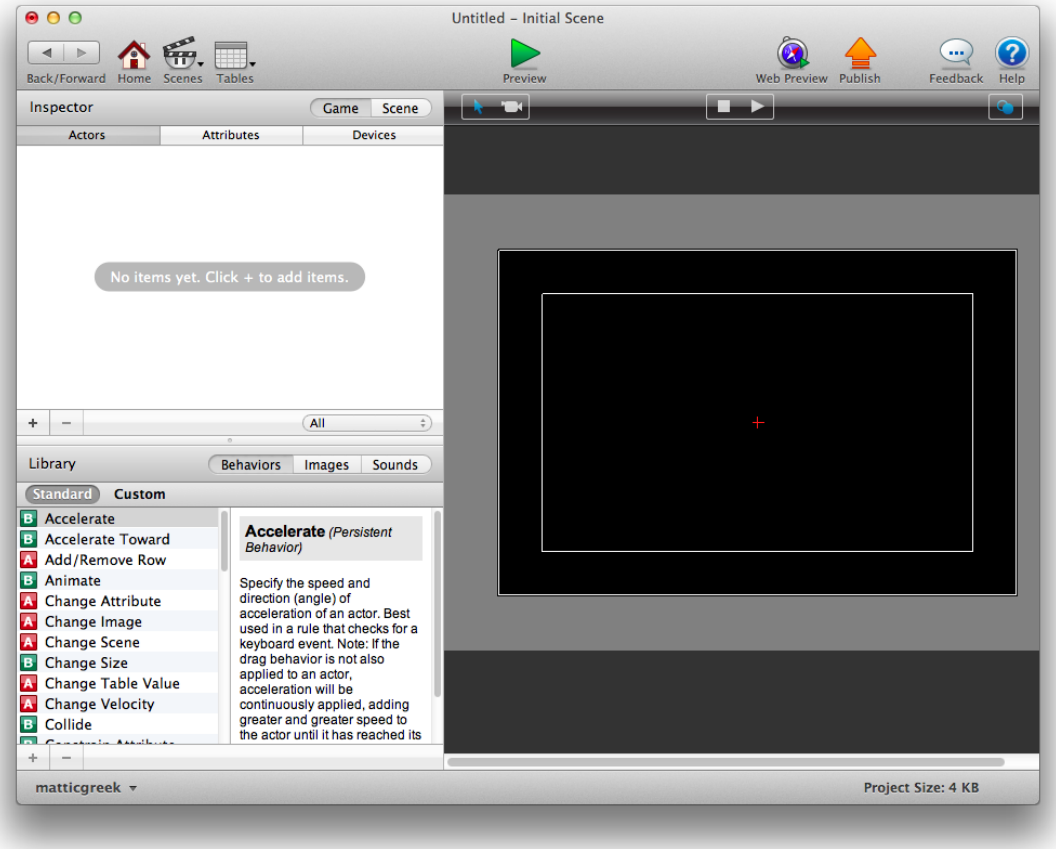
Actors: to GameSalad anything we put in a scene is an “actor,” whether it “acts” in the conventional sense or not. Things that have no impact on gameplay (such as images we drop in to make the environment more convincing) are actors no less than the character(s) the player controls. We can give actors behavior but we don’t have to. If it serves a purpose, it’s an actor.

Add/Remove: to create a new element of any type (scene, actor, image, sound, etc.), we click the “+” button in the lower-left of the type’s window. To remove an existing element, we click on it then click the “-” button:

Let's open up a scene. Click the *Scenes* tab, then double-click on "Initial Scene":



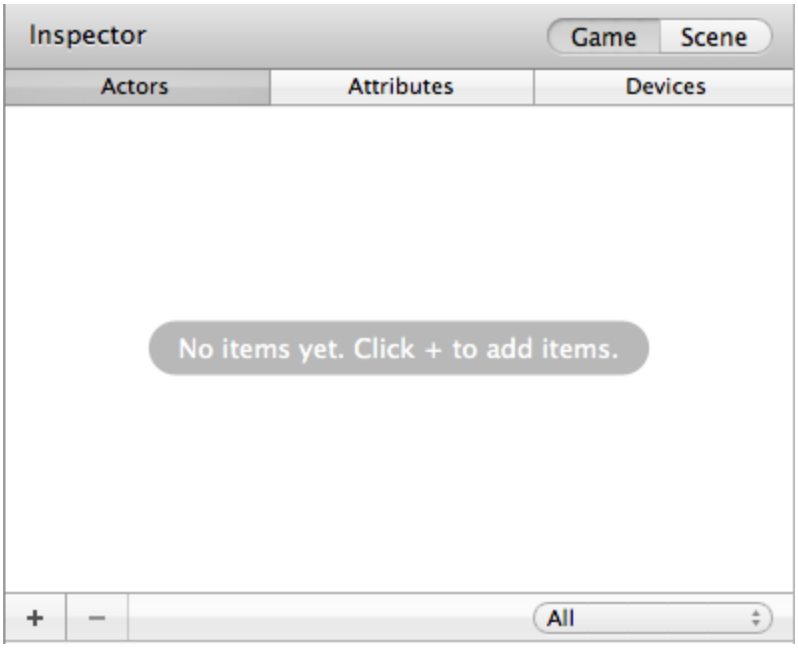
We should see this:



The very top looks the same as before, except that we no longer have the four navigation

tabs. However, clicking the Home button in the upper-left will return us to the main interface. Meanwhile, the Scene dropdown button provides a handy shortcut for switching between scenes. Finally, we can click the Back and Forward buttons to retrace our steps.

The area just under the home button is called the Inspector:



We see two sets of tabs. *Game* and *Scene* in the upper-right let us switch between details that apply to the entire game and those specific to an individual scene. Within each tab is another set of tabs. *Scene* lets us set *Attributes* (more about these momentarily) and *Layers* (which control what appears in front of what). *Game* lets us work with *Actors*, *Attributes* and *Devices*. We already know what Actors are; Devices refers to any available hardware, such as the mouse.

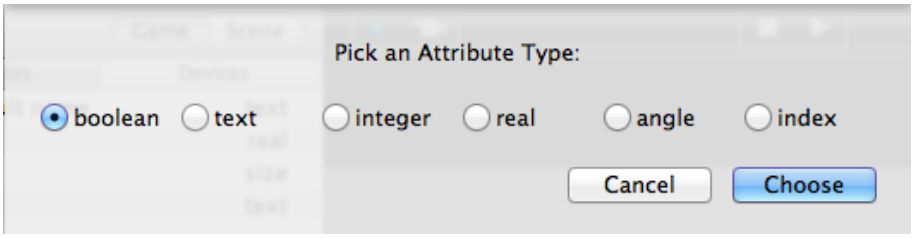
Attributes: “attribute” is the name GameSalad gives to a fundamental concept in programming and mathematics--the “variable.” A variable is a sort of container that we make to store changing or unknown information. For instance, in algebra we might see the equation “y + 25 = x.” In this equation “y” and “x” are variables: we’re not really adding the letter y to 25, but we don’t know the numerical value of what we’re adding, so we give it the label “y” and attempt to solve for its actual numerical value. Also, the value of y is dependent on the value of x, and vice-versa. If we set x equal to 28, y must equal 3, but if we set x equal to 20, y must equal -5, and so on. The equation is an input-output machine: if we input a value for x or y, the machine outputs the necessary value for the other variable.

In software, the data that make up the program are changing all the time. Think about all the things that move, process, react to your input, etc. So, when making software (such as a

video game in GameSalad) we use variables to contain the information that controls the software's behavior.

For instance, perhaps the player moves a character to the right by pressing the right arrow key. To draw the game screen, GameSalad uses a coordinate system with x and y axes, just like in algebra. The origin point (0,0) is the lower-left corner. So "moving to the right" means adding to the character's position on the x-axis. The character might be moving from x = 100 to x = 110. But it would be extremely tedious to remember and input all the specific numbers. So we use a variable, which we might call "xPos," instead, and let GameSalad do the calculations for us! We might say that moving to the right means adding 10 to the xPos. We don't have to write equations in GameSalad (although we can if we want to), but if we did it would look something like this: y = x + 10, where x is the old x position and y is the new one.

GameSalad includes many attributes (variables) by default, including ones to track xPos, yPos, etc. We'll see examples of using these in this curriculum. Sometimes though we need to track something not covered by the defaults, which means creating our own attributes. This is easy to do. On the *Attributes* tab (make sure to select *Game* or *Scene* first) we click the "+" button and see this:



This has a lot of terms on it, but it's not so bad. A variable can store any kind of data--not just numbers. Let's consider these four:

Boolean: sometimes we just want to know if something is true or not. We can turn to philosophy, or we can use a boolean, which only takes the values "true" and "false" and functions as an on/off switch. For instance, we might want to make a character do something if it's on the left side of the screen. We could make a boolean, call it "onLeftSide," and set it to "true" whenever the character's x position is left of the center of the screen. Then we would set up our desired behavior and make it activate if onLeftSide were set to "true."

Text: maybe we want to display different messages depending on how many items the player has collected. We could create a text variable and set the game to display a message using that text. Then, we'd set the variable's value to different text depending on item count.

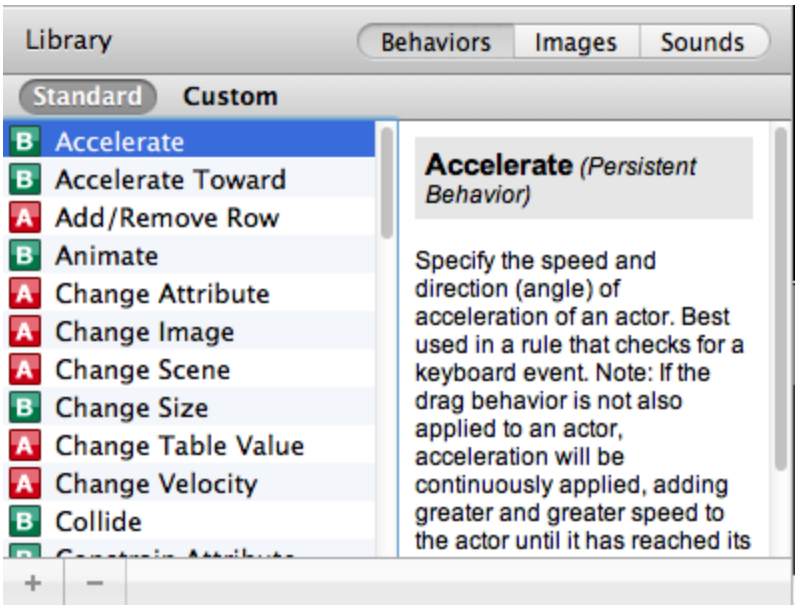
Integer: integers are the counting numbers: 0, 1, 2, 3, etc., but also -1, -2, -3, etc. We use integers the most because they are ideal for many purposes, such as...counting.

Real: don't be alarmed by the word "real." This just means fractions and decimals. If we need more precision than integers offer, we use a "real" attribute.

If this is overwhelming, don't worry. Remember, there is no one way to do things in GameSalad (or in programming!). Often there are many valid ways to do the same thing. What matters is finding something that makes sense to you (and making sure it works!).

For more information about attributes and their types, click here.

OK, enough of attributes! Below the Inspector, we have this:



As we see, this is the *Library*. It stores all the assets we use to make our game. These include *Behaviors*, *Images* and *Sounds*. We see the tabs in the upper-right. As you might imagine, we use image files to make the graphics in our game, and we use sound files to make the music and sound effects.

To add an image or sound, we simply click on the tab, then drag an image or sound file from our computer and drop it into the window. GameSalad will import the file and make it available for our use. Note that this is not the same as using the file in our game. We are simply storing it in a place where GameSalad can find and reference the file. Later we will actually use the file.

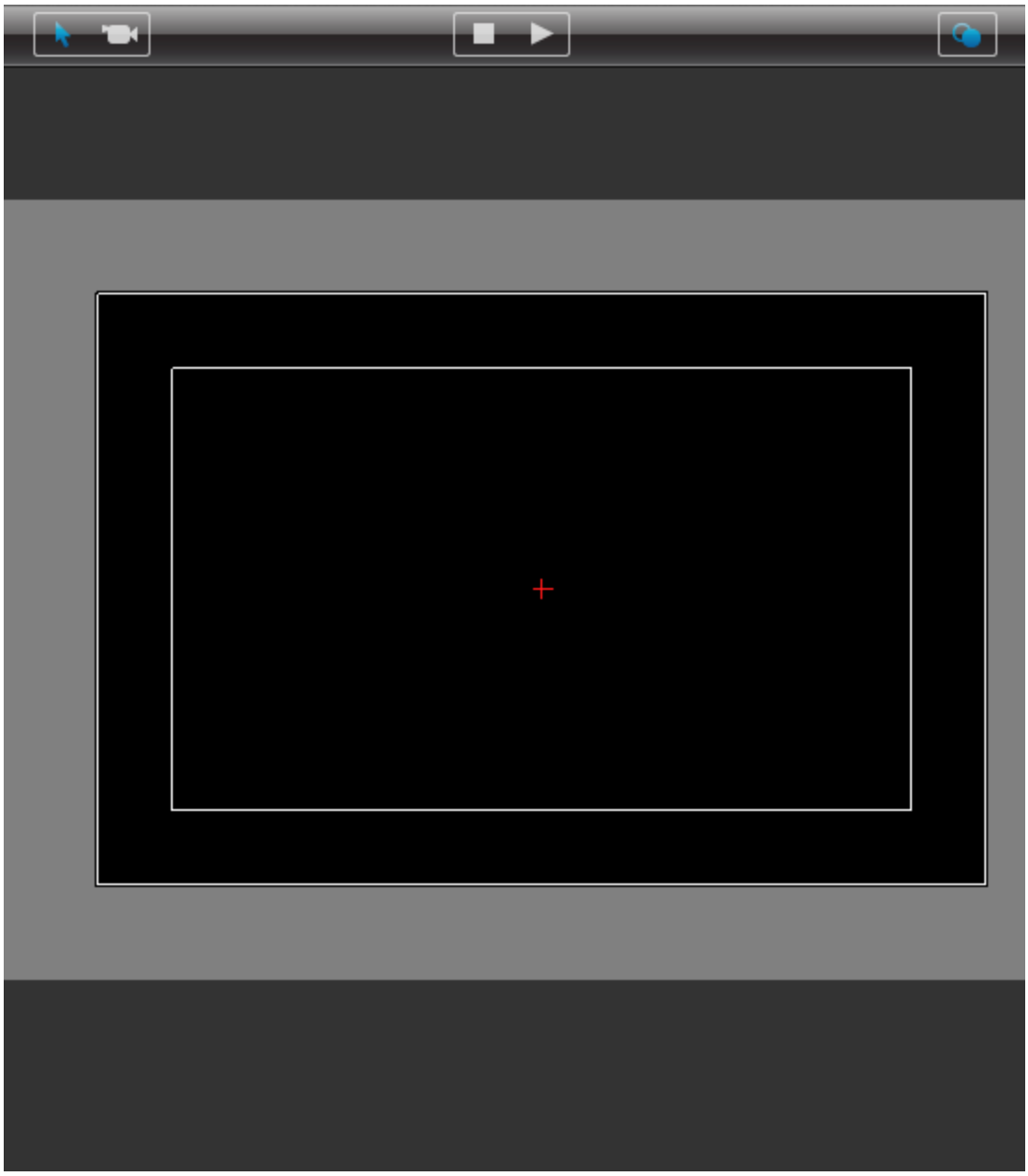
GameSalad also includes options for purchasing pre-made images and sounds, if we so desire.

Behaviors are the key to using GameSalad. These are the mechanisms by which we make

objects in our game (actors) do things. *Behaviors* includes subtabs for *Standard* and *Custom*; in this curriculum we will stick with *Standard*.

On the left side of the *Behaviors* window is the list of behaviors; on the right side is a very handy description of whatever behavior is currently selected. It is useful to scroll and click through the various behaviors, reading what each one does. Not all of them will necessarily make sense right away but the description usually conveys the general idea.

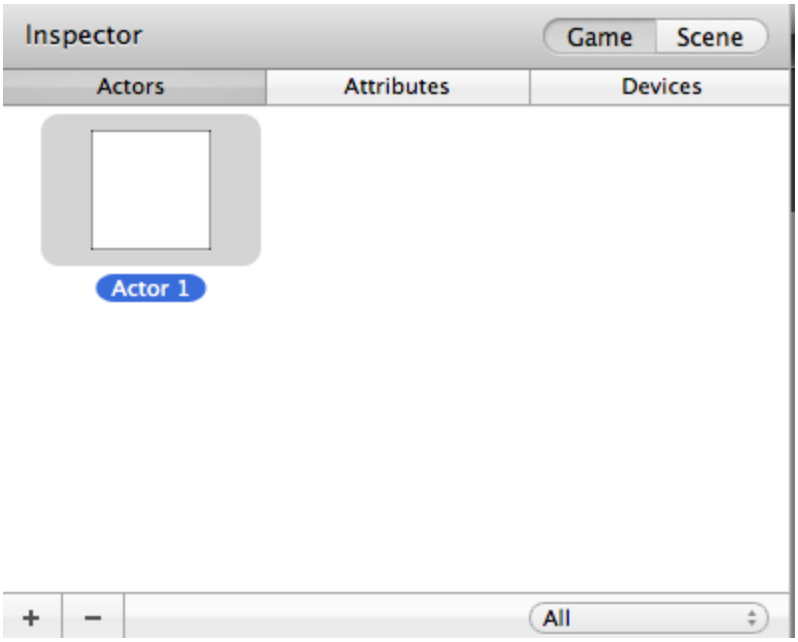
As with the Images and Sounds, the Behaviors don't do anything on their own. It is now finally time to look at the right side of the interface:



This is the game window. Here we will place and arrange the elements of our game. The black rectangle is the size of the target device's screen (determined when we chose a Platform). The red cross and the inner white line tracing a smaller rectangle are guides to help focus the presentation, but the boundaries of the game screen go all the way to the edge of the black.

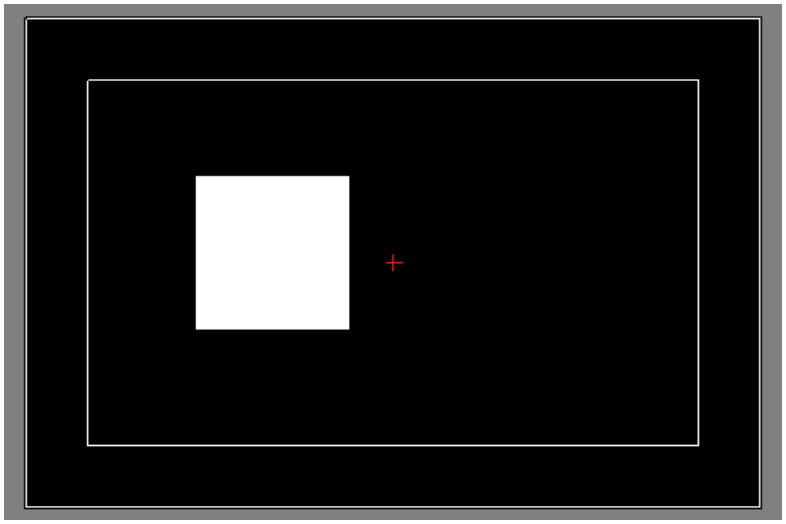
We can ignore the buttons along the top, except for the Stop and Play buttons in the middle, which allow us to test-run this particular Scene without loading the full-blown Preview mode. This is helpful, although behavior in this mode does not always reflect the actual behavior, so it's best to use the Preview mode whenever possible.

Well, this scene is pretty dull without anything in it. Let's make a box move across the screen. To do so, we need to create an Actor. We go to the Inspector and click the Actors tab, then click the + button in the lower-left. Voilà--we have an actor:

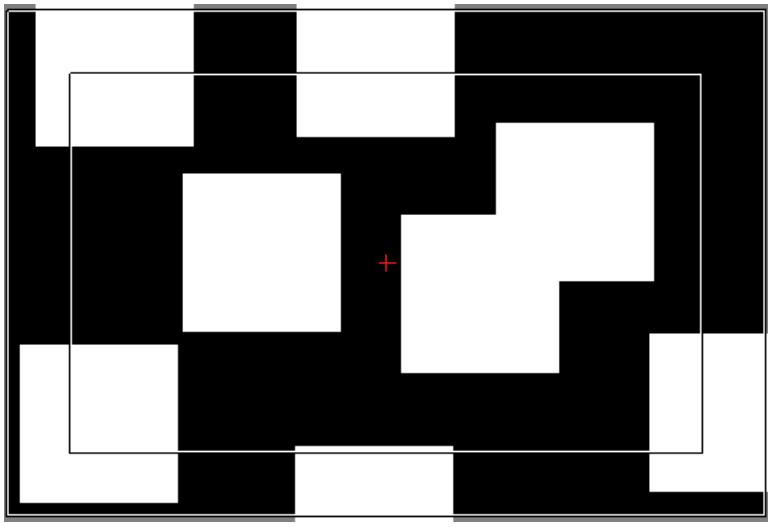


It's not very exciting but it is a white square, just like we wanted. If we had an image, we could click on the Images tab in the Library, then drag the image from the Library into the Inspector and drop it onto our white square. That would assign the image to the actor. However we like this white square so we will stick with it.

But wait--the game window is still empty. That's because we haven't placed the Actor in the Scene. To do so, we click on the Actor and drag it into the Scene:



Success! Note that “Actor 1” remains in the list of Actors in the Inspector. Maybe we decide we want two white squares: we can click on “Actor 1” and drag it into the game window a second time...or a third time...or a fourth...

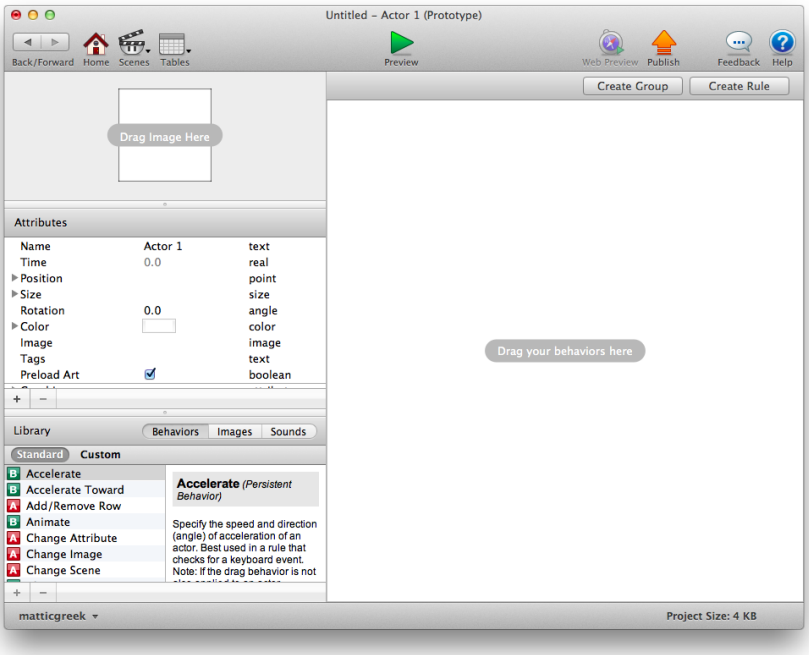


This brings up an important concept: the *instance* vs. the *prototype*. When we create an Actor, what we are creating is a *prototype*: a model for how a specific type of object in the game should look and behave. As we saw, creating an Actor doesn’t cause the Actor to appear in the Scene. We must drag the Actor into the Scene. When we do so, we create an *instance* of that Actor (which *does* appear in the Scene). Technically the instance is also an Actor, but it is not a prototype and only exists within the specific Scene.

This is very powerful. Not only can we can create as many more instances as we like--all of which will look and behave according to the prototype--but we can go back after creating instances and make changes to the prototype, and they will automatically apply to all the

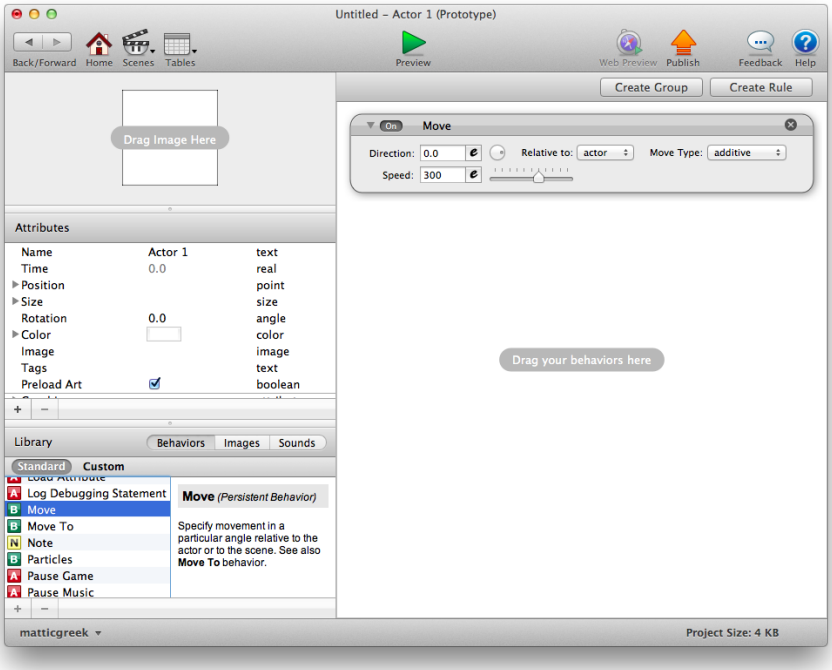
instances!

OK! We have at least one white square in place. Let's click the green Preview arrow at the top-center and...oh, the square just sits there. Well, we haven't told it to do anything yet. Let's change that. In the Inspector, we double-click on the Actor 1 prototype and see this:

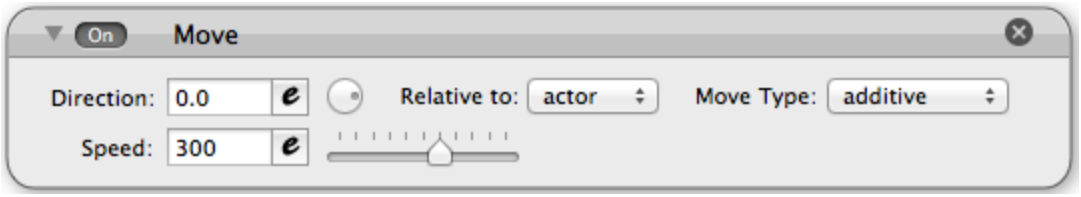


This looks similar to what we had before, but now the Inspector is replaced by the individual Attributes of this Actor, and where the Scene was on the right we have a big, inviting space to assign behavior!

We want the white square to move, so we scroll through the list of Behaviors in the Library until we find Move! As the text on the right prompts, we drag Move into the empty box and get this:



Let's take a closer look:

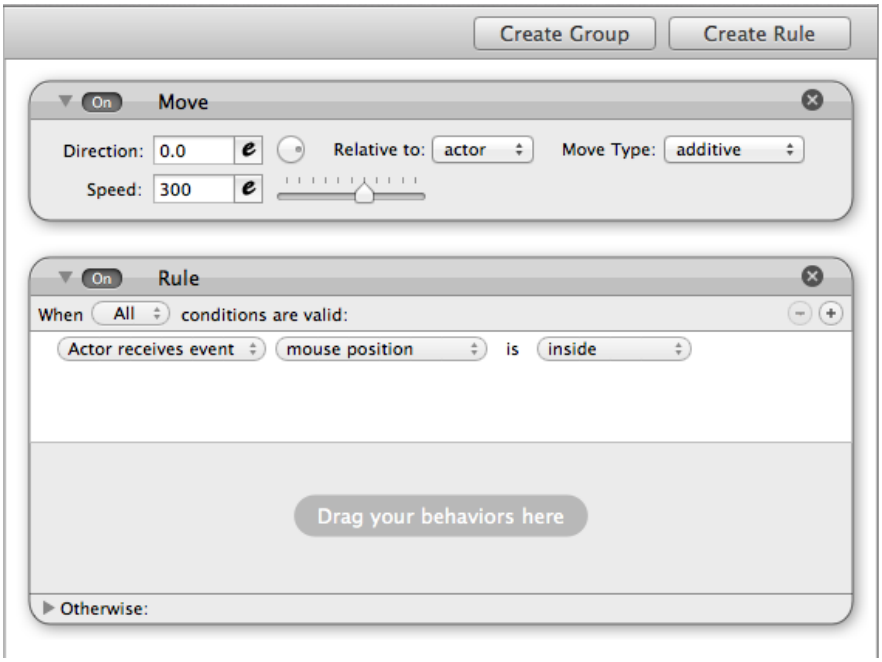


As expected, we must set a direction and speed. Direction is in degrees. The default of zero degrees means to the right, so we'll leave it alone. The default Speed is fine as well, and we can ignore the other options for the time being. Now, click the Preview arrow again.

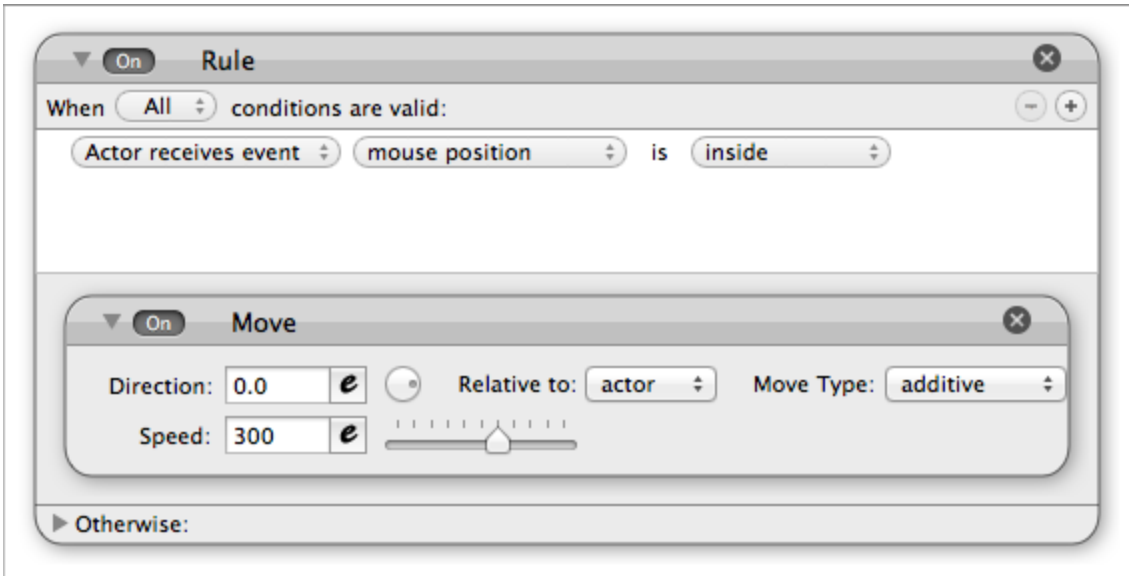
Woohoo! The white square moves to the right and vanishes offscreen. If we click the curly arrow above the preview while it's running, we can reset the game and watch it again.

It's not much of a game, though--it's not even interactive. All we're doing is watching. This is because right now we've only told the white square to move right, all the time. It's not dependent on any other factor.

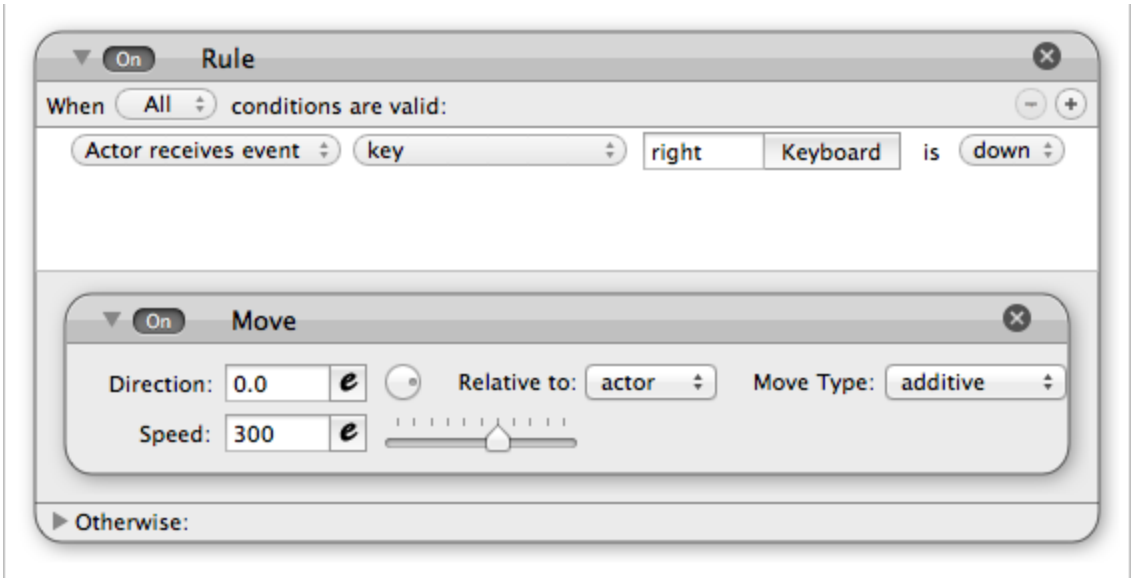
Let's make the square move when we hold down the right arrow key. We click the Back button to get out of Preview mode and back to Actor 1's Behavior. What we want to do now is set a condition to determine when behavior takes place. This is called a Rule. We click "Create Rule" in the upper-right and see this:



Well, we have a Rule, but there's nothing in it! We click on our Move behavior and drag it into the Rule:



We need to set the condition. We click "mouse position" and change it to "key." A "keyboard" field appears and we click on the word to see a handy virtual keyboard, then click on the Right arrow key. Now our behavior should look like this:



We click Preview again. Now when we hold down the Right arrow key, the white square moves right, and when we release the key, the square stops moving. Huzzah!

Congratulations! You've created your first interaction in GameSalad. There's much more to the program, but this should get you started. Move on to the curriculum to learn the specifics of creating multiple different games in GameSalad. **Scroll down to see some tips for using GameSalad like a pro!**

Tips and Tricks

Rename

Double-click on the name of any element to rename it. This works not only for actors and attributes but also scenes, images, rules and behaviors you've added to an actor, etc. This is very helpful for remembering what things do (especially for groups, rules and behaviors).

Duplicate

Hold ALT/OPTION and click and drag on something to duplicate it (actor, scene, etc.).

Place Precisely

Click an actor within a scene, then use the arrow keys to move it with precision. Hold SHIFT for faster movement.

Scale Proportionately

Hold SHIFT while dragging the edges of a selected Actor to keep its current proportions.

Edit an Instance

Double-click an actor within a scene to edit that specific instance of the prototype, but be warned that it won't be affected by future changes to the prototype. Click "Revert to Prototype" to restore the relationship but lose any individualized changes made.

Switch Behaviors On/Off and Collapse/Expand their Details

In the upper-left of every group, rule and behavior you've added to an actor are an arrow and an on/off button. Use the on/off button to turn behaviors on and off prior to starting gameplay (great for testing, because you don't have to delete the behavior to disable its effects). Use the arrow to collapse/expand the details of the behavior. This is very useful when you create more complex behavior and the window gets cluttered.

Reorder, Copy and Paste Behaviors

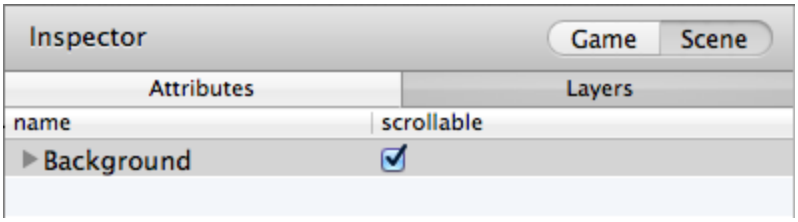
Click and drag any group, rule or behavior you've applied to an actor to move it around. You can reorder elements and move them into (and out of) containing elements. You can also click to select any element and then use the Edit menu (or keyboard shortcuts) to copy the element and paste it elsewhere. You can easily duplicate entire behavior sets this way.

Control Display Order with Layers

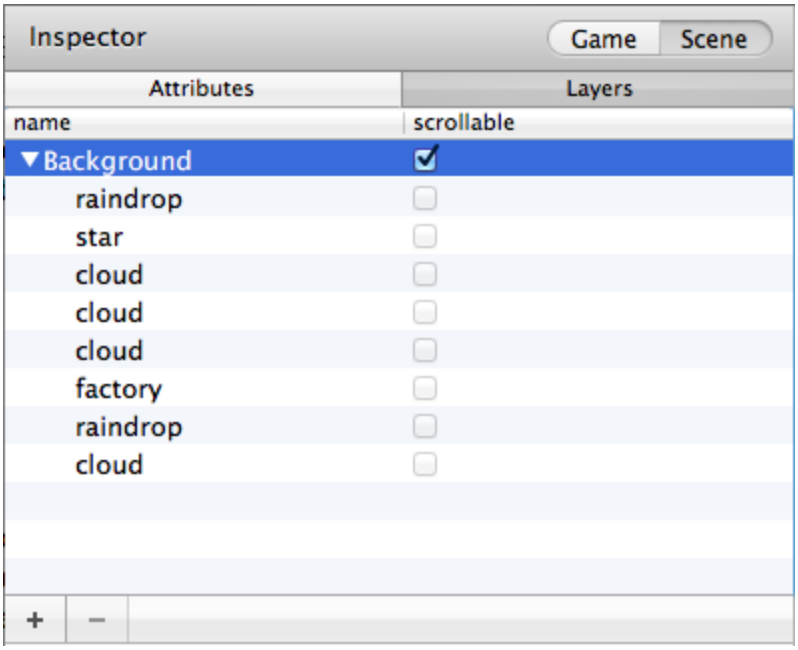
It is inevitable that at some point you will discover that one Actor is covering over another Actor and it should be the other way around. To correct it, you will need to manipulate *layers*.

If you've done any visual design and/or used programs such as Photoshop you may be familiar with layers already. 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, when you add a new Actor to the Scene it appears in front of everything already there. Certain Behaviors like Spawn Actor allow you to set the Layer Order of certain Actors, but in general you'll need to make changes manually. To do so, open the Scene view (note that layers are Scene-specific). In the Inspector, click the *Scene* tab in the upper-right, then click the *Layers* subtab. You'll see something like this:



Not much there right now. Every GameSalad project starts out with a single layer, and anything you add to the Scene will be added to that layer. The default layer is called "Background." Click the arrowhead to the left of Background to expand it and show its contents:



This is from the tutorial for Smog Cloud Madness. Here are all the Actors in the Scene, in ascending display order. In other words, the raindrop at the top of the list is drawn last, which puts it at the front of the layer (nothing in the list will obscure it), while the bottom-most cloud is drawn first, so every successive thing would cover it over. Changing the order is as simple as dragging the Actors higher or lower in the list.

You can create new layers and distribute your Actors among them. It isn't necessary but makes organizing Actors much easier, especially once you have a lot of Actors. For instance if you were creating a lot of clouds you might put them all in one layer, then put your raindrops in a separate layer. If you do that then just make sure to select the appropriate layer before adding new Actors; otherwise you'll have to move them around manually.

PROTIP: you can right-click on any instance to get immediate layer options, including "Send To Front," "Send To Back," "Send Forward," and "Send Backward."

Fix Problems by Debugging and Logging Statements

Sometimes things go wrong and you need to figure out why. This is known as "debugging," because every problem is a bug waiting for you to squash it. In that battle, "Log Debugging Statement" is your best friend. This is a Behavior in the Library that lets you print (i.e. "Log") something to a special screen. For instance you could add this Behavior to a Rule and Log any text you like (such as "I'm working!") to test whether the Rule is working. Or you could Log the value of an Attribute to see if it is updating correctly.

Where is this magical special screen? In the GameSalad menu at the very top of the interface, choose "View → Debugger." A new window pops up. Leave this open while running

your game and use Log Debugging Statement to identify where the problem lies. Press the Clear button if the Debugging window gets too cluttered. Learn to use this tool. It is invaluable.

Use Web Preview

Games (and all other software as well) perform differently depending on the machine running them. Every computer / game system has its quirks that affect how smoothly and quickly the game runs, how the timing between events works, and even how graphics are displayed. For this reason it's very important to test your game on your target device(s).

This is especially true for publishing to the Web via GameSalad Arcade. Your game may look and run differently in the web browser than it does in the GameSalad interface using the Preview function. All the graphics you so carefully aligned may no longer fit together correctly. The timing between events that you spent so much time tweaking may be off. Your beautiful game that you poured so much work into may look like a disaster area.

Fortunately, GameSalad includes a "Web Preview" function. The button is in the upper-right of the interface. As opposed to the regular Preview function--which runs the game using your computer's hardware--Web Preview runs the game using the same HTML and CSS that GameSalad Arcade uses. There's even an option to open the game in your browser. Use Web Preview. It loads almost as quickly as Preview, and it lets you see immediately any differences in placement, timing, and other performance, rather than giving you a nasty surprise after you've taken the time to publish the game online.

Be aware that the same changes you make to your functioning game in order to make it function correctly online may make it stop functioning correctly on your computer. For instance you might discover you have to change the timing of an enemy's movement, which was fine in regular Preview but is wrong in Web Preview. Once you've tweaked the timing, the movement now looks correct in Web Preview, but it's no longer correct in regular Preview. Unfortunately there is no avoiding this and similar consequences of the performance differences between systems. For that reason you may want to make multiple copies of your game file, each tweaked to run correctly on a different platform.