



GameSalad Creator

(Windows Version 0.9.92)

Written by Jack Reed Layout by Anne Austin

Table of Contents

Windows Creator Walkthrough	3
Getting Started	3
System Requirements	3
Intro	3
First Look	3
The Library	4
The Backstage	6
The Stage	9
Making a Game	10
Scenes	22
Scene Attributes	22
Actors	24
General Attributes	25
Graphics-Related Attributes	25
Motion-Related Attributes	26
Physics-Related Attributes (Non-Motion Physics)	26
Behaviors	27
Attributes	31
Tables	32
Creating a Table	32
Editing a Table	33
Using Data in a Table	33
Additional Table Functions	33
Publishing With GameSalad	34
Tips and Best Practices	40
Frequently Asked Questions	40
Troubleshooting	42
Further Assistance	43
Definitions	43

Windows Creator Walkthrough

Getting Started

System Requirements

Windows 7 or Windows Vista

Intro

GameSalad Creator is a 2D object oriented tool that allows you to create completely original games using a drag and drop interface, enabling you (the user) to create applications for iOS, Android, HTML5, and even for the Mac Platform without typing a single line of code!

Tip: Currently, Windows Creator supports Android and HTML5 publishing while Mac Creator supports iOS, Android, HTML5, and Mac Platform. A single Professional GameSalad Membership subscription for Windows and Mac is required for Android publishing.

This is possible by using Creator's unique design and powerful features to turn logic and assets into finished high-quality products. For our purposes, 'logic' refers to the combination of **Rules**, **Behaviors**, and **Attributes** that jointly define how a project operates, and 'assets' are the images and sounds imported into your project.

Tip: Throughout this manual we'll be introducing many new terms that we have identified in bold. For definitions, please refer to page 43.

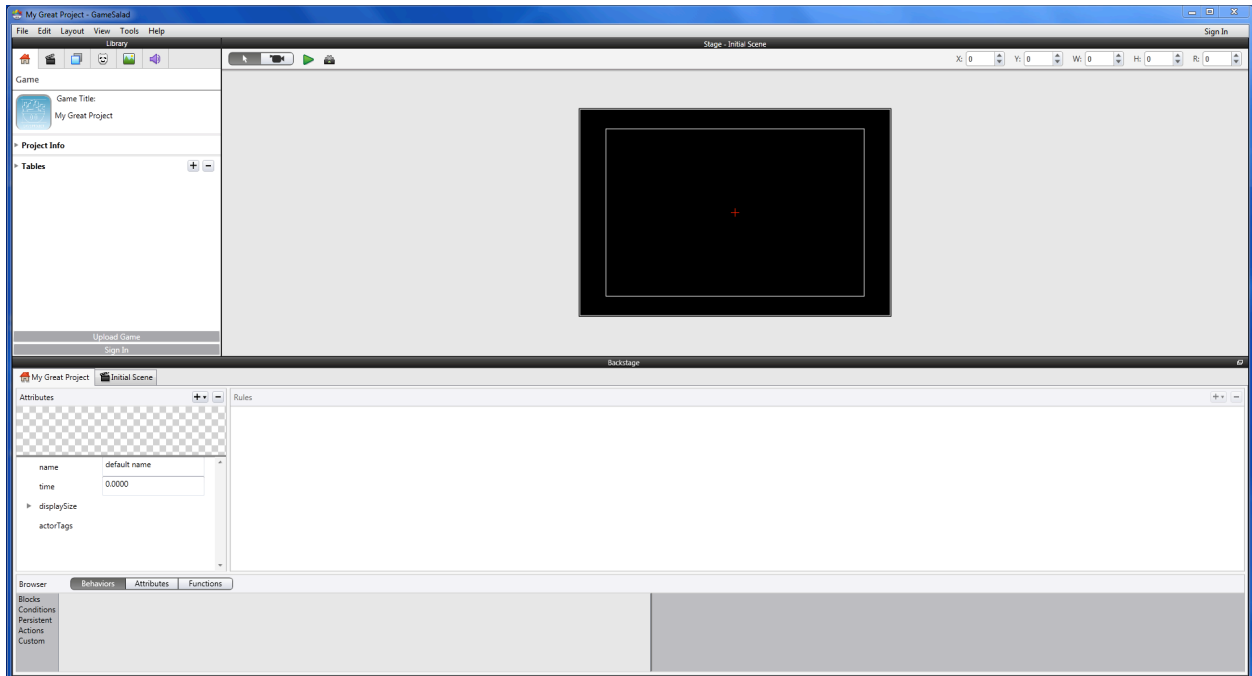
Over the next several pages we'll cover what to do first, as well as some essential background information, so you can begin to experiment and create on your own. Before that, you'll need to download the GameSalad Creator itself.

To download Creator for Windows, head over to <http://gamesalad.com/get-windows-creator> to get the most recent version. Afterwards, just **install** it and you're good to go!

Tip: Though you will be able to download and use the Creator without registering for a GameSalad account, you will not be able to publish without logging in. We highly recommend going ahead and taking the time to register at this point in the process, if you haven't already created an account.

First Look

Whether this is your first time using the GameSalad Creator or you're a long-time user of the Mac Creator, as soon as you run the Windows Creator you'll be presented with the following screen where you'll see numerous buttons, fields, and tabs.

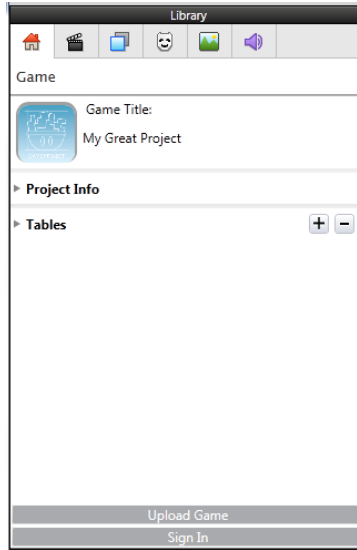


While it may seem like a lot to take in at first, don't worry! Keep in mind that everything you see is intended to give you control over the creation of your game, and, even better, you're reading this manual which will walk you through everything you need to get started. For now, let's jump right into a new project (our word for unpublished games), which is automatically brought up the moment you run Creator.

The Library

You are now officially using the Creator! One of the first things you'll notice is that the Creator's user-interface is divided into three primary sections: **'Library'**, **'Stage'**, and **'Backstage'**.

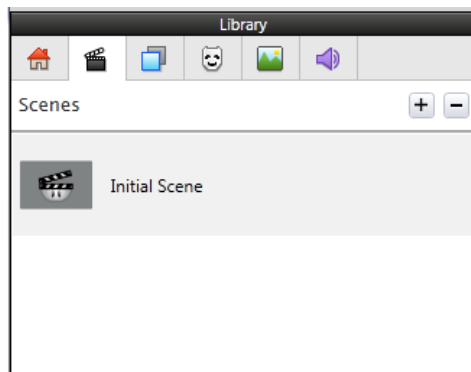
Let's focus on the Library for the moment, which can be found in the upper left-hand corner of the window. The Library panel will help you stay organized by placing various key components of your game into separate, but easily accessible, tabs. Since we're looking at a brand-new project these tabs will be mostly empty for the moment, but will quickly populate as you make progress.



The organization tabs found in the Library panel include: **'Game'**, **'Scenes'**, **'Layers'**, **'Actors'**, **'Images'**, and **'Audio'**. As mentioned above, these tabs are where you'll find most of the fundamental components for your project. For example, you can give your project a title in the 'Game' tab and image files can be imported into your project using the '+' icon in the 'Images' tab.

Tip: Image files can be in .png, .jpg, .jpeg, .gif, .tiff, .wdp or .bmp format. Acceptable sound formats include .ogg for sounds and .m4a for music.

Select the 'Scenes' tab and you'll see a single scene with the title of "Initial Scene" (which you can edit by double clicking). It's defined by some default scene attributes in the **'Backstage'** section of the Creator, which we'll go over in more detail further on in the manual. You can create additional scenes by pressing the '+' icon, a process similar to the importing images and sounds.



Scenes are the primary way of dividing up your project, and are often thought of and used as **"Levels"** or **"Stages"**. Different scenes are able to have unique configurations, including **Scene Attributes** (such as size, gravity, and X/Y wrap) and actor placement.

Changing between scenes is not something that happens automatically with GameSalad, but is instead triggered by a Behavior (appropriately named ‘**Change Scene**’). We’ll come back what Behaviors are, how they work, and where they can be found later on, but for the moment go ahead and click on the ‘Actors’ tab in the Library panel.

Tip: The two main factors you’ll want to weigh when considering adding a new scene are “Am I okay with the player encountering a brief pause while the next scene is loading?” and “Am I comfortable with a potential performance decrease if I make the current scene larger by continuing to add actors and behaviors?”.

Actors are the primary agents in all projects – everything from your game’s central protagonist, to the platforms he or she might run across, to the countless shots fired throughout their epic journey. Actors can be tailored to fulfill diverse roles depending on your game’s genre and needs. You might even have a non-moveable, invisible actor that you make responsible for spawning all enemy actors into the scene.

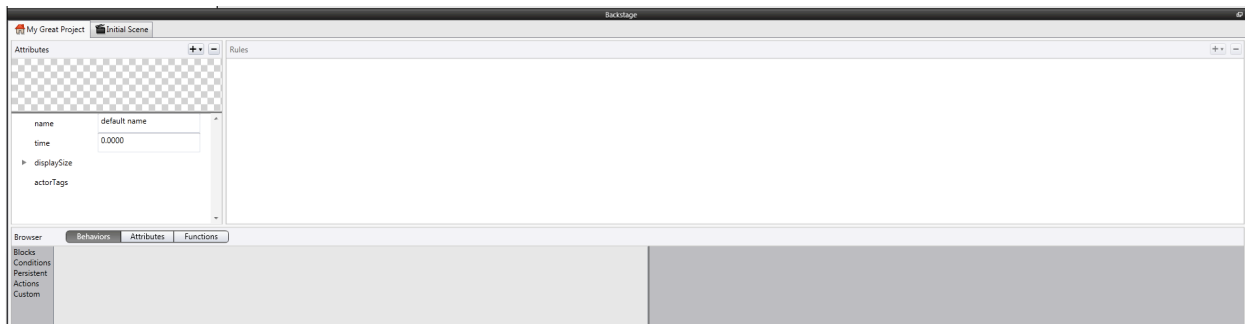
You’ll find every actor that currently exists in your project in the ‘Actors’ tab (not just those used in a specific scene). This is also where you will be able to create new prototype actors or delete existing actors.

When a prototype actor is dragged into the scene (found in the Stage section of the Creator) it creates an instance actor, which is essentially a copy of the prototype actor. To clarify the prototype/instance relationship further: new instance actors include all logic and most attribute settings of the prototype actor that created them, and initially start out as “locked” to the originating prototype. While the instance actor is in locked state, any change (with a few exceptions) to the prototype actor will also affect every instance actor locked to it.

In this way you can create many instance actors all tied to a single prototype. This allows you to save an enormous amount of effort by only needing to edit one prototype, instead of manually making changes to every instance actor individually, which can be prohibitively time consuming.

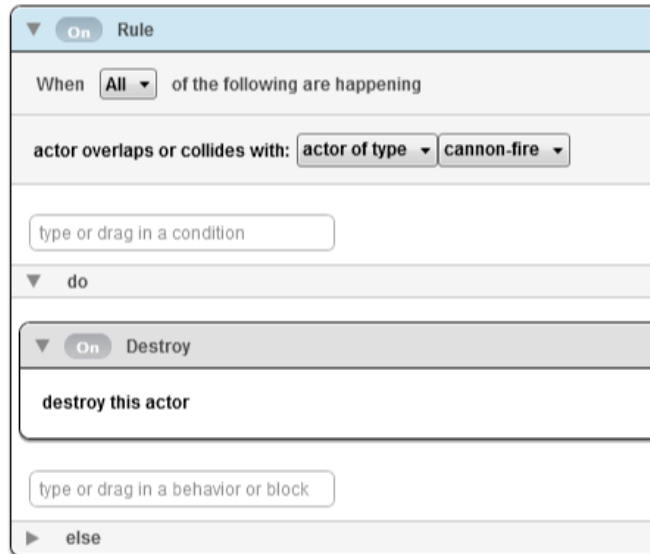
The Backstage

From the Backstage you can access Game Attributes, Scene Attributes, actor configuration (logic and Self Attributes), and the content of any **Tables** you’ve set up. For this reason, it’s also where the majority of your design implementation will be accomplished.



When we talk about logic, we're for the most part we're talking about Rules and Behaviors. Rules are, at their simplest, direct if/then statements in that they will consider whether a given condition is true or valid and, based on the answer to this question, they will then take one or a series of pre-determined actions, known as Behaviors.

To illustrate, imagine a Rule that states than an actor should destroy its self when it overlaps or collides with cannon-fire.

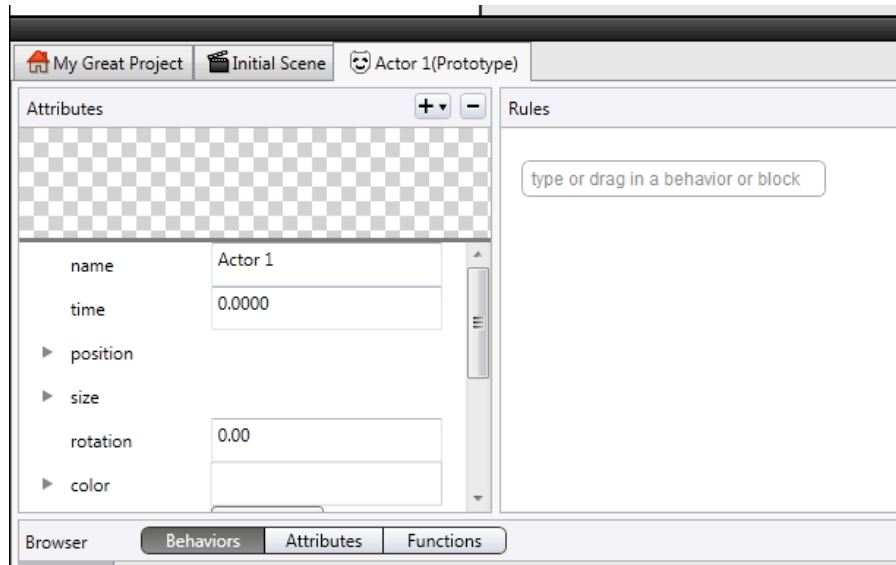


We'll walk through the specific steps for setting up actors to accomplish a specific tasks or design objects in the 'Making a Game' section of this manual, but the above should give you a general concept of how Creator is able to bring your ideas to life without requiring you to learn and type out code.

If you're looking at a new (and so, almost entirely empty) project, you've most likely noticed that the only two tabs currently available in the Backstage are 'My Great Project' and 'Initial Scene'. Using the Library panel, click the Actors tab and press the '+' button to create a new actor.

Tip: Note that only instance actors exist within a scene. For example, dragging a prototype actor into a scene will create a new instance of that actor. A list of all instance actors for the currently selected scene can be found under the 'Layers' tab, while all actors listed in the 'Actors' tab are prototypes.

A new actor named "Actor 1" should appear in under the organizational header 'All'. Click "Actor 1" and a new tab named 'Actor 1(Prototype)' will automatically appear and become the focus of the Backstage.



Tip: You know you're editing a prototype actor when you see (Prototype) in its Backstage tab name. Similarly, you know you are editing an instance actor when you see (Instance) instead. While editing an unlocked Instance actor, press the 'Revert to Prototype' button to revert back any logic that differs from the prototype actor. This will also re-lock the instance actor to its prototype.

As you can tell from the blank 'Rules' section, this newly created actor has no logic and is ready for you to tell it how to behave. We'll be using the 'Browser', 'Attributes', and 'Rules' sections of the Backstage to achieve this.

The Attributes panel will give you access to some of the core configuration settings of the tab you currently have selected. So, clicking on the tab 'Actor 1(Prototype)' when you were previously had tab 'Initial Scene' selected will result in different attributes being listed, as well as different values for those attributes. Note that both these tabs have a "name" and "size" attribute, but unlike 'Initial Scene' the 'Actor 1(Prototype)' also has a 'Graphics' category, since actors and scenes are distinct from each other, which is reflected by some unique attributes.

Attributes are one of the most powerful aspects of Creator, and are essentially value placeholders. They are divided into three main categories: Scene Attributes, Game Attributes, and Actor Attributes (also known as Self Attributes).

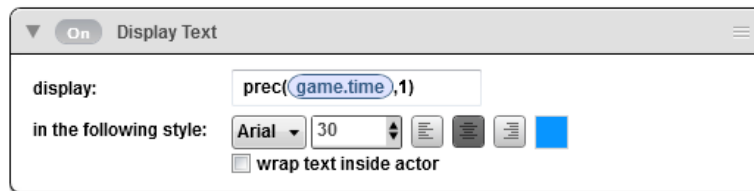
If you hit the '+' icon in the Attributes panel, Creator will prompt you to pick a type for your new Attribute for the Backstage tab that is currently selected: **Boolean**, **Text**, **Integer**, **Real**, or **Angle** (the differences between these are covered in the Attributes section on page 31). While you'll be using some of these types more frequently than others, each of them are incredibly powerful due to the fact that actors can reference, change, and react to these values.

Attributes are how actors in your project will know when to spawn new enemies, how much life or ammo the hero has left, cause the game to show the Game Over screen, tell the player they have obtained a new high-score, and much more.

Later, I recommend going through these more thoroughly, but for now let's move on to the 'Browser' section of the Backstage. The 'Browser' section is divided into three tabs: 'Behaviors', 'Attributes', and 'Functions'.

We've already briefly covered Behaviors (but we'll go into even more detail soon), so go ahead and click the 'Attributes' tab. As its name implies, the 'Attributes' tab has ties directly back to the 'Attributes' panel. You'll often find yourself creating logic that needs to reference game, scene, or actor attributes. In cases like this, simply click the field in the Rule or Behavior you want to add the reference to, and find the relevant attribute in the 'Attributes' tab.

Functions are potent tools for ensuring that the math behind your game is working for you, rather than against you. For example, the function 'prec' (short for precision), allows you to decide the maximum number of decimal places you want a value to be able to go out to.



This screenshot incorporates elements from all three tabs: It has a Behavior (Display Text), an attribute reference (game.time), and a function (prec, or precision). When this actor is placed into a scene and that scene is previewed using Play, it will now display the amount of time that the project has been running for to the tenths decimal place.

There are countless other possibilities we could delve into, but let's move on to the Creator's third primary user-interface section, the 'Stage'.

The Stage

The Stage is where you'll place your newly created actors to give them life and to allow them interact with each other. You can also preview your project from here using the green 'Play' button, to allow for working in rapid iterations. This means making a change and then having the ability to immediately see how that change affects your project, giving you real-time information on how your project is shaping up.

Tip: Working with a particularly large scene or using a monitor that has a lower resolution? You can use the 'Toggle Backstage' button to temporarily give you more room to work with for the Stage.

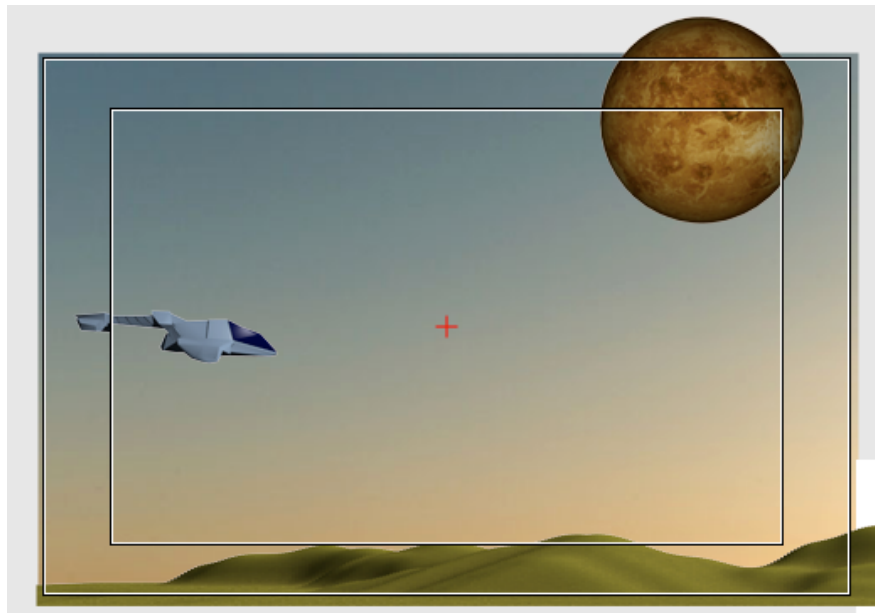
Now that we've established some of the fundamental aspects of GameSalad Creator, let's go over how to make a simple project.

Tip: At this point you should have enough information to begin experimenting with Game-Salad Creator if you learn better by doing than by reading. Of course, this is not meant to discourage you from continuing to read on! There's a lot more to cover and we're happy to have you.

Making a Game

There are plenty of menus, options, and features that we haven't gotten a chance to cover, but rather than attempting to memorize everything about Creator this very moment, let's make a game instead! We'll start by using a template project so that we have some art and sound assets available, though.

Head over to the [Cookbook](#) and download the 'Basic Shoot Em Up' template, then follow the instructions provided for opening it in Creator.

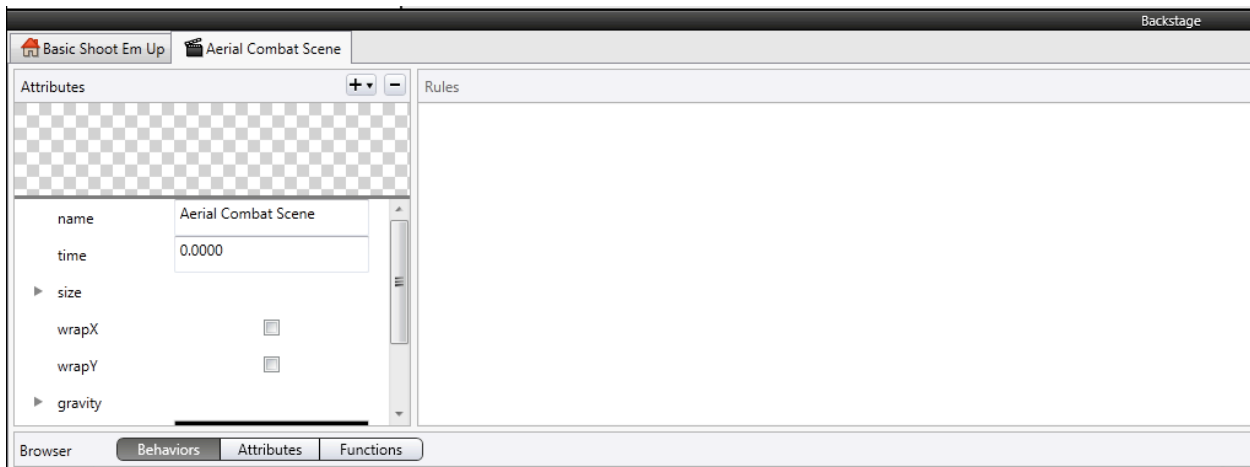


We'll leave the actors as they are (for added convenience later in this tutorial), but let's go ahead and delete the scene titled "Initial Scene" by selecting it in the 'Scenes' tab of the Library panel, then hitting the '-' icon.

Tip: You can always re-download the original version of the template; so don't worry about making your own changes to these projects. After all, experimenting and making mistakes is a great way to learn. That said, save often and to multiple files! A common issue reported by users is getting into trouble when saving to a single file.

Now we'll click the '+' to create the new scene which will serve as our starting point. Double click the default name (Scene 1) and let's name it something that reflects what is planned for this scene: "Aerial Combat Scene" (minus quotes). Also, drag the new scene it to the top of the Scenes list so that later when we try our game we're sure it's playing the right scene.

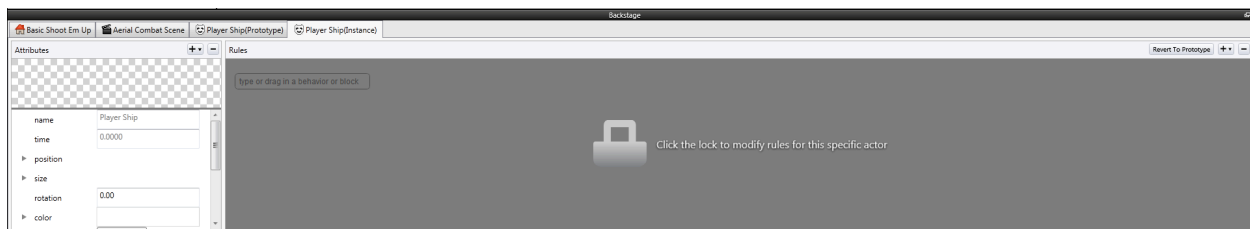
Click the scene in the list to bring it into focus in the Backstage, and you'll be looking at this:



Now, click the ‘Actors’ tab in the Library panel. Again, let’s not delete the pre-made actors hidden under the ‘All’ category, but we won’t be using them for the moment either. Instead, click ‘+’ to create a new actor. You’ll notice that ‘Actor 1’ is now at the very bottom of the Actors tab list.

Double click ‘Actor 1’, rename it to “Player Ship”, and then drag it from its list on the left side of the screen to the scene in the Stage.

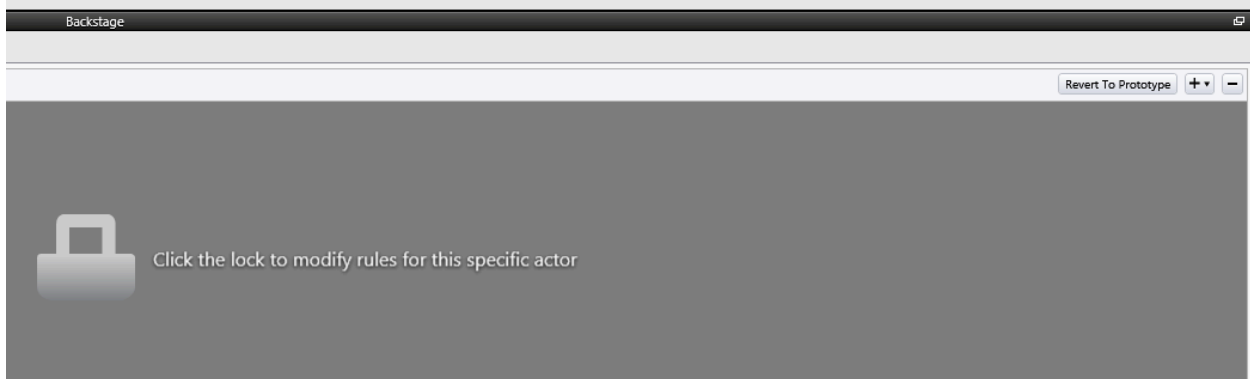
It’s not much to look at right now, but give us a few more minutes and you’ll see us turn into a core component of our new game. What we have in front of us is a lone instance actor that has been dropped into a scene, and that instance actor is still locked to a prototype actor of the same name.



Tip: If we dropped a second “Player Ship” actor into the scene, they would have the same name and essentially be identical in every way, except for their position in the scene. These X and Y positions are tracked by two Self Attributes, which are included in every actor by default. Self Attributes are unique from Game Attributes in that each instance actor independently keeps track of their own Self Attributes. This means that if we added the Self Attribute “Current Health” to a prototype actor representing enemy ships, each new instance actor spawned based on that prototype would keep track of the value of their own version of that Self Attribute. If we attempted to do this with a Game Attribute, hurting one enemy would affect all enemies. By using Self Attribute, each enemy ship’s health is separately tracked, so it’s possible to severely damage one ship while leaving another untouched.

There are several items worth highlighting about this particular actor’s tab in the Backstage panel. Locked instance actors have been mentioned a few times early on in this manual (actors whose logic and attributes are informed by the prototype actor that created them), but here we see the hard-to-

miss notification that lets us know we are looking at an instance actor who is still locked to its prototype.



By clicking the lock symbol, we can now make changes to this instance actor that will not affect the prototype actor. Likewise, while unlocked, changes to the prototype will no longer affect the instance actor. To re-lock the instance actor, select the ‘Revert to Prototype’ option.

Tip: While locked actors are in most ways exact duplicates of their prototype, its important to note that they are not exactly identical. For example, each instance actor can have a unique X and Y position attributes, as well as width and height attributes for size. Changing these and comparable attribute values in the instance actor will not break the prototype/instance lock.

For now, let’s leave this Player Ship instance actor locked, and instead click the prototype tab ‘Player Ship(Prototype)’. Notice that there’s no lock symbol, since it’s only instance actors that are locked to prototypes, rather than the other way around.

In just a short while, we will be adding rules and behaviors to this actor, which is a big step towards letting our game know how to operate. But let’s first take a moment to change this actor from a generic white box to a ship art asset. Open the ‘Images’ tab in the Library, expand the ‘All’ category, and drag the PNG file titled “Ship” over to the white box representing “Player Ship” in the Stage, then let go.

Much better! However, its current dimensions of 100 Width by 100 Height are making our ship look a little squished. Find the attributes that control these dimensions by expanding the ‘Size’ category in the Attributes panel of the Backstage, and then change the width to 75 and the height to 50.

If you didn’t notice a change in size for the ship image in the scene, you most likely made this adjustment to the prototype actor, rather than the instance actor. Updating the prototype actor will result in all future instance actors created from this prototype to have that size, but all already existing instance actors will have to be changed manually, since the Self Attribute size is not affected by the prototype/instance lock.

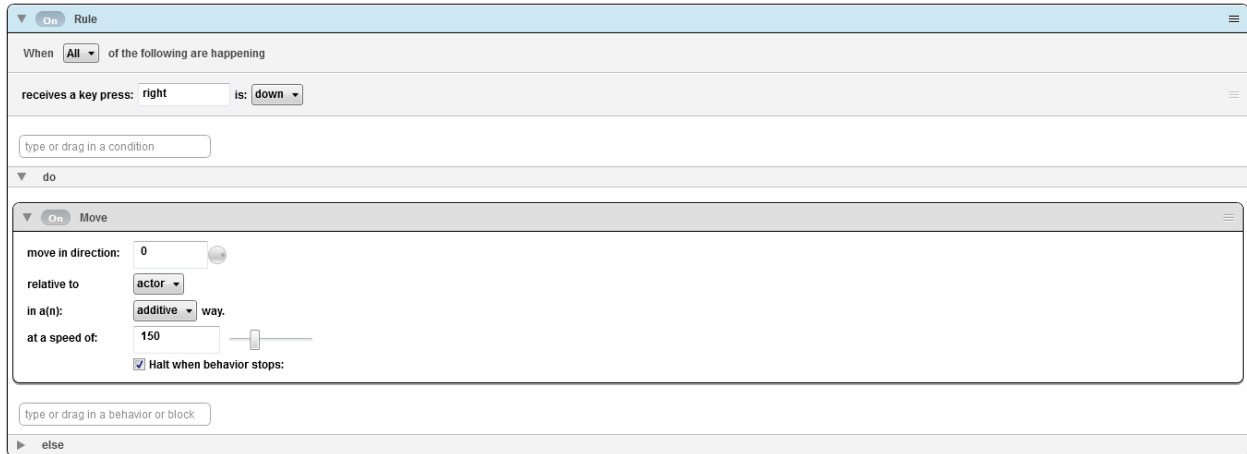


As for adding rules and behaviors to this prototype, first we need to decide on what we'd like it to do. Since this ship will be representing the player as the primary protagonist, we can be confident that we want the player to be able to control the movement of the ship and also the discharging of its weapon. In addition, let's introduce a danger element, so that the player's ship can potentially be destroyed if it becomes too damaged.

We'll focus on movement first, and for simplicity, we will create a rule that considers keyboard input, rather than accelerometer or touch screen input.

Tip: For some examples on how you might set up a movement user-interface for touch screen devices, check out the 'Official Cross-Platform Controller Template' in the GameSalad [Cookbook](#).

Be sure you're on the 'Player Ship(Prototype) tab and create a new Rule, using either dragging it from the 'Blocks' category in the 'Browser' panel or by clicking the '+' icon and selecting 'Rule' from the dropdown menu. After creating the new rule, drag a Key Behavior (found under Conditions) into the 'When' section and a Move behavior into the 'do' section. After a few quick adjustments, it should look like this:



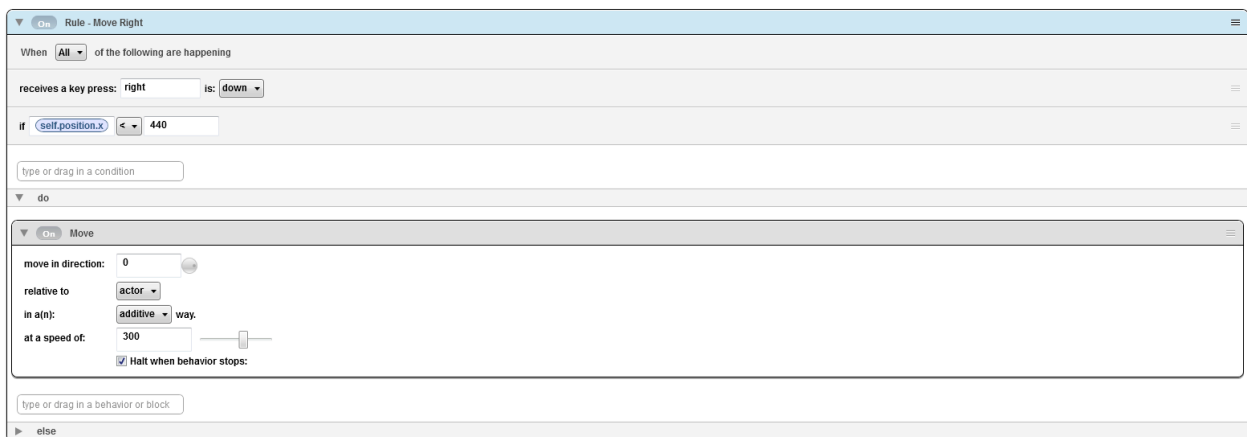
Tip: Next time you're creating a rule, try taking advantage of the auto-complete feature. For example, in the 'When' section, type "key" and press enter, and under 'do' type "move". This can save you lots of time!


Now whenever the right arrow on the keyboard is pressed, the 'Player Ship' actor will move to the right of its current position. Rather than manually create three more of these for the other directions, select the rule and use copy/paste to create three copies of it.

This will create an exact reproduction of the logic selected (and any other logic contained within it). Now just edit the condition key and movement direction for this new logic, and you're all set. Hit the green 'Play' button in the Stage panel and give it a test!

Notice that while you can move the ship around using the keyboard, it's also possible to fly it out of the viewable zone, where the player can no longer see it.

If you'll open up the prototype actor 'ship' (not 'Player Ship', which is the one we created), you can see how this problem was addressed in the original template:



Tip: If you're having a hard time seeing as much of the logic stack as you'd like to, try undocking the Backstage using the  icon on the Backstage title bar. To re-dock it, simply hit the button again.

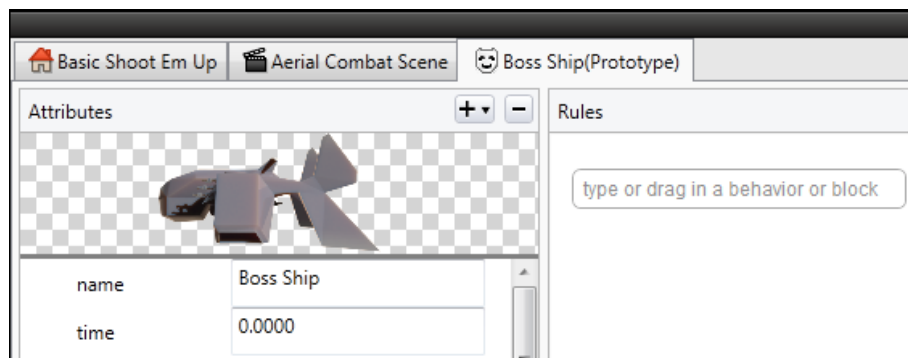
The original “ship” actor has an additional condition which must be true at the same time as the right arrow key is pressed for the Move Behavior to be triggered (notice the “When All of the following are happening” requirement at the top of the rule). Essentially, when taken in context of the other 3 Move rules, it means that actor “Ship” cannot move outside of the viewable part of the zone.

Feel free to make this rule supplement on your own actor if you like, but don't worry about it too much. This manual's goal is to provide you with the core concepts of Creator, so we won't be going too in-depth into best practices.

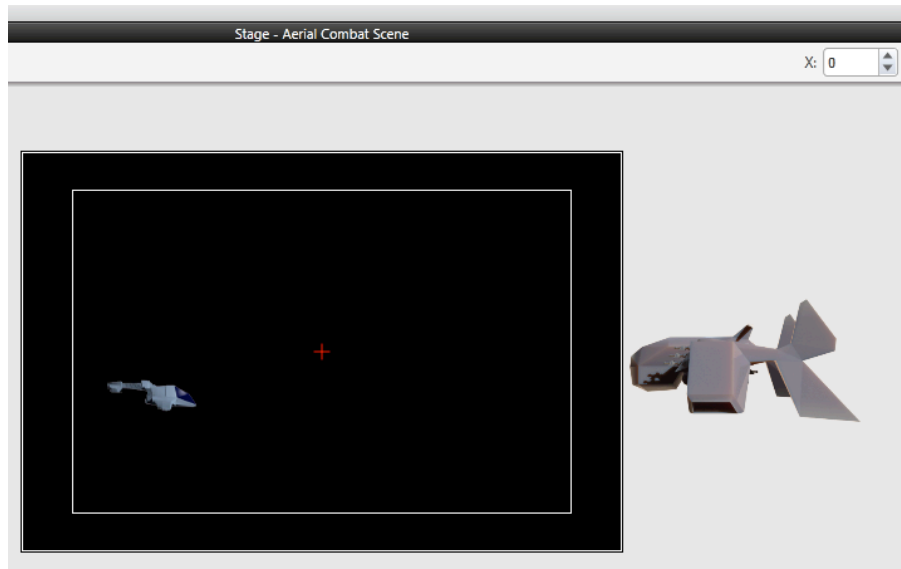
Tip: Notice that the rule we created is simply named “Rule”, while the template's rule is named “Rule – Move Right”. Double clicking the title bar of any rule or behavior allows you to rename it. Using an appropriately descriptive name allows you or your co-creators to immediately have an idea of what that portion of logic controls or affects, which can save a great deal of time when going over past work.

This is a good time to take a step back and review what's left to do. (Also, if you haven't already been doing so, be sure to save from time to time) Since this is only an example project, we won't be spending too much time on it, but some basic additions might include: a background, the ability to shoot, an enemy to shoot at (and to shoot back at you), a win condition, and a lose condition.

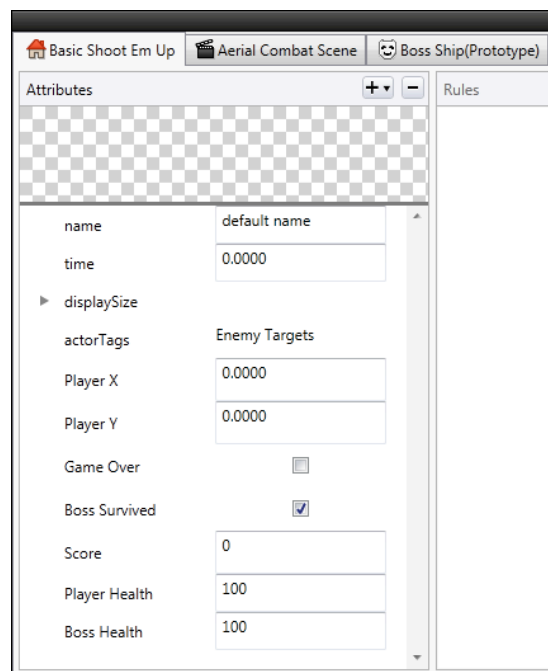
The next step on this path would be to create a new prototype actor using the Library panel and name it “Boss Ship”. Before dragging this actor into the scene, drag the image “boss” from the Images tab onto the prototype actor's image section (the grey and white checkered area), in the Attributes panel. In addition, change the size attribute of the prototype to 200 width by 100 height.



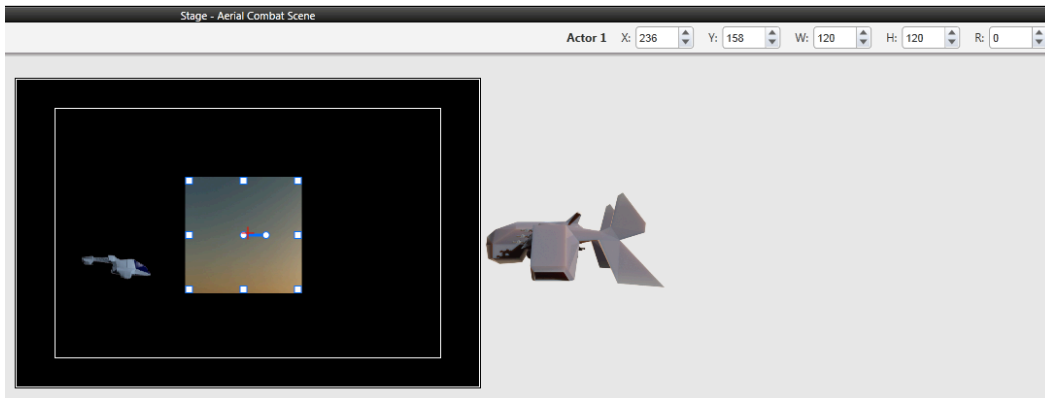
Now place it in the scene, and move the ‘Boss Ship’ actor so that it is just outside of the visible scene. This is so we can have it fly into the viewable area after the scene starts, for a more dramatic effect. Your stage for the ‘Aerial Combat Scene’ should now look something like this:



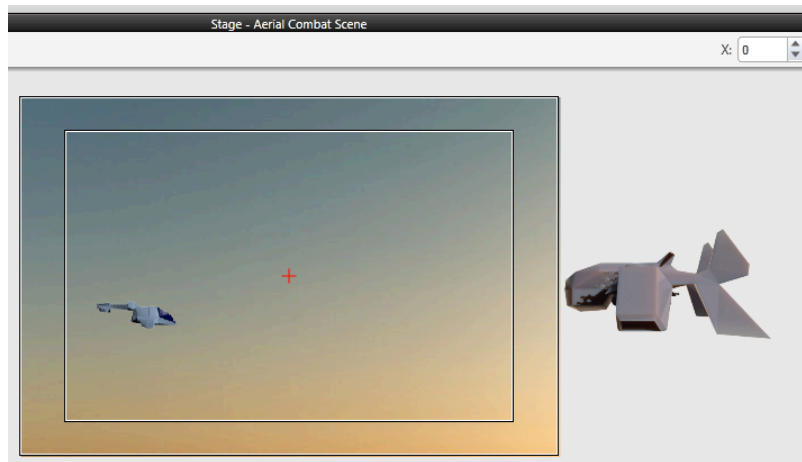
To give these two actors the capacity to keep track of their current health, create two new integer Game Attributes in the Attributes panel while the 'Basic Shoot Em Up' tab is selected, and name them "Player Health" and "Boss Health". Set their default value to 100.



Up next is our goal of adding a background. Drag the image titled "sunset" directly into the scene, which is actually a short cut for creating a new actor (which you should go ahead and rename "Background"). We'll also use another short cut by editing this new actor's size directly from stage, by using the options in the upper right-hand corner.



With the sunset actor selected, change the 'W:' (width) field to 480 and the 'H:' (height) field to 320. Unfortunately, now our "Background" actor is covering up our "Player Ship" actor. This is easy to fix, though, by simply right clicking on the "Background" actor in the Stage and select the 'Send to Back' option.



From here, we'll be using a few of the previously created actors for convenience, rather than having you recreate logic that already exists. However, we will be making a few tweaks to make it suite our needs. Open up the actor "missile" and adjust the Collide/Destroy Rule so that the actor will properly explode when it hits the player, by exchanging "ship" with "Player Ship". Make a similar change with the "laser" actor, by scrolling down to the bottom rule, and changing "actor with tag" with "actor of type", and then set the actor to "Boss Ship".

We have a few more pieces of logic to place in our "Player Ship" and "Boss Ship" actors, which can be seen in the below screenshots. First, let's go over "Player Ship": To summarize, by using the two Constrain Attribute behaviors, the actor "Player Ship" is constantly reporting its position to the game by way of two Game Attributes. These attributes are both 'real' attributes, and we didn't need to create them since they were already part of this particular template. Now other actors can use and react to these two Game Attributes' values, such as the heat seeking missiles the "Boss Ship" will be firing.

Tip: If an actor is suppose to be gone forever once off the visible portion of the scene (an enemy that was successfully dodged, such as a missile) then make sure to set specific behaviors to destroy these actors. Otherwise your game might begin to experience performance slow downs. In addition, actors that move too far beyond the boundaries of the scene itself will be automatically destroyed.

The image displays three screenshots of a game engine's behavior tree editor, showing different types of blocks and their configurations.

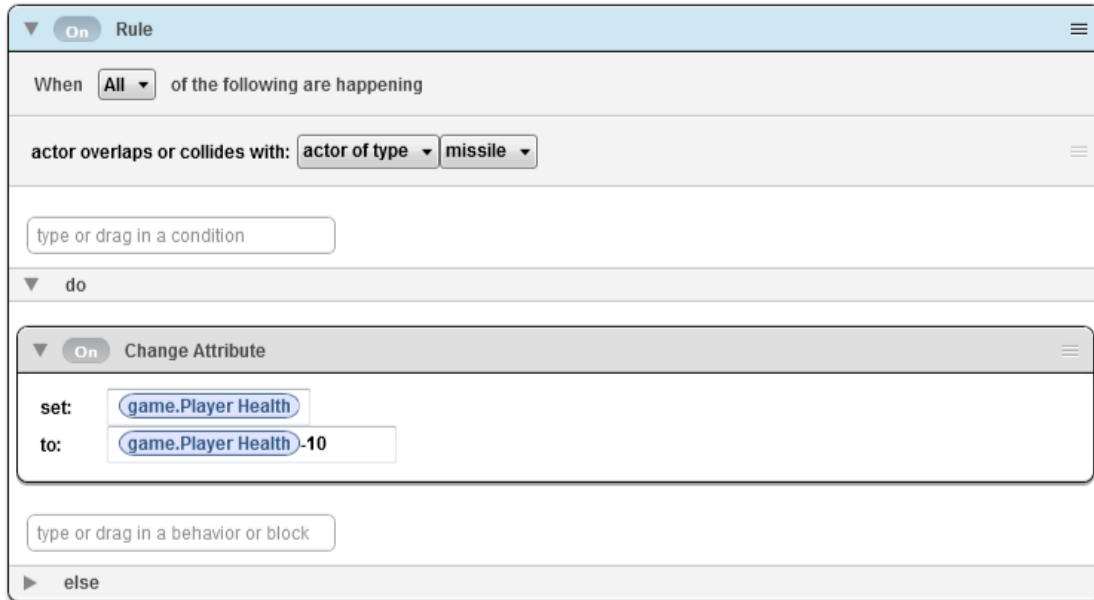
Block 1: Constrain Attribute
This block is titled "Constrain Attribute" and is currently "On". It has two fields: "constrain:" with the value "game.Player X" and "to:" with the value "self.position.x".

Block 2: Constrain Attribute
This block is also titled "Constrain Attribute" and is "On". It has two fields: "constrain:" with the value "game.Player Y" and "to:" with the value "self.position.y".

Block 3: Rule
This block is titled "Rule" and is "On". It has a "When" section with a dropdown menu set to "All". Below this, it specifies "receives a key press: space" and "is: down". There is a text input field below with the placeholder "type or drag in a condition". The "do" section contains a "Spawn Actor" block.

Block 4: Spawn Actor
This block is titled "Spawn Actor" and is "On". It has several fields: "spawn actor:" set to "laser", "in the direction:" set to "0", "relative to:" set to "actor", "from position:" set to "0", "relative to:" set to "actor", and "place:" set to "in front of actor". There is a text input field below with the placeholder "type or drag in a behavior or block".

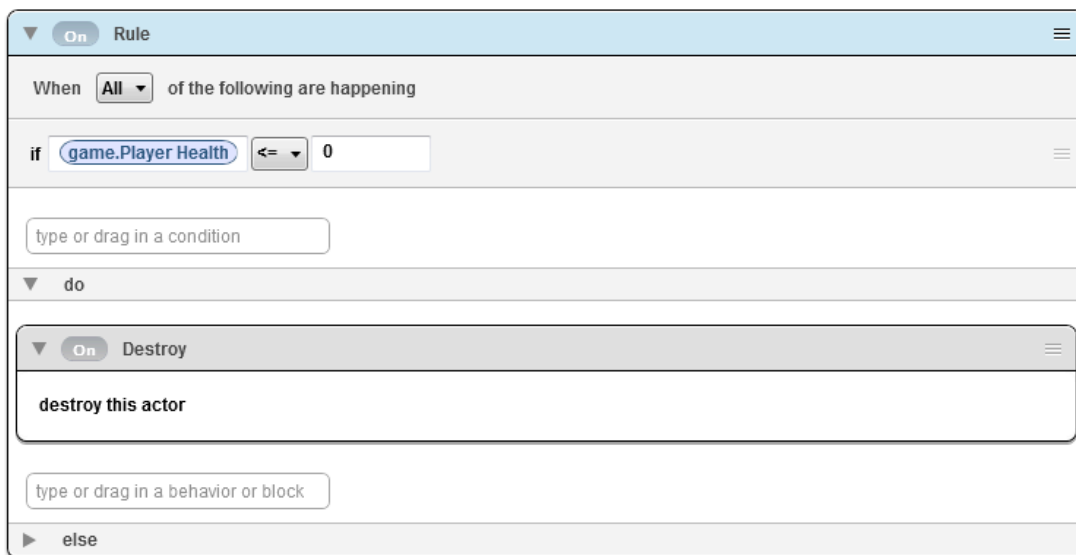
At the bottom of the "Rule" block, there is an "else" section with a right-pointing arrow.



Tip: While actors can reference their own Self Attributes, they cannot directly reference other actors' Self Attributes. To work around this, create a Game Attribute and have it be updated using a Change Attribute or Constrain Attribute Behavior by the reporting actor. Then have the referencing actor check and react to that newly created Game Attribute.

Additionally, we have a rule that states that pressing spacebar fires the player's laser by spawning the laser actor – simple enough!

One item I'd like to specifically call out is the use of the mathematical equations and functions, which opens up countless possibilities for game design. In this situation, I used a very straightforward equation to decrease the "Player Health" game attribute by 10 each time 'Player Ship' is struck by a missile. And of course, when the player's health runs out, their ship is destroyed.



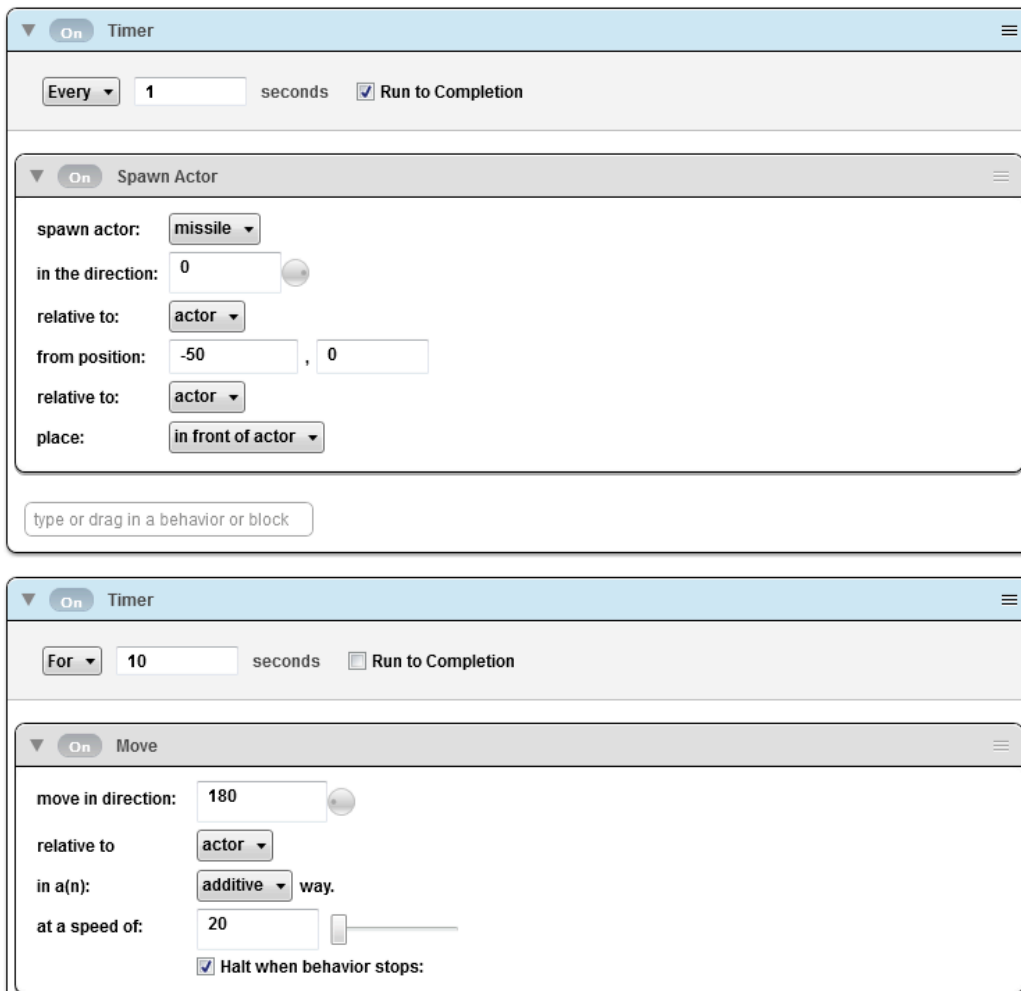
Now for the Boss Ship:

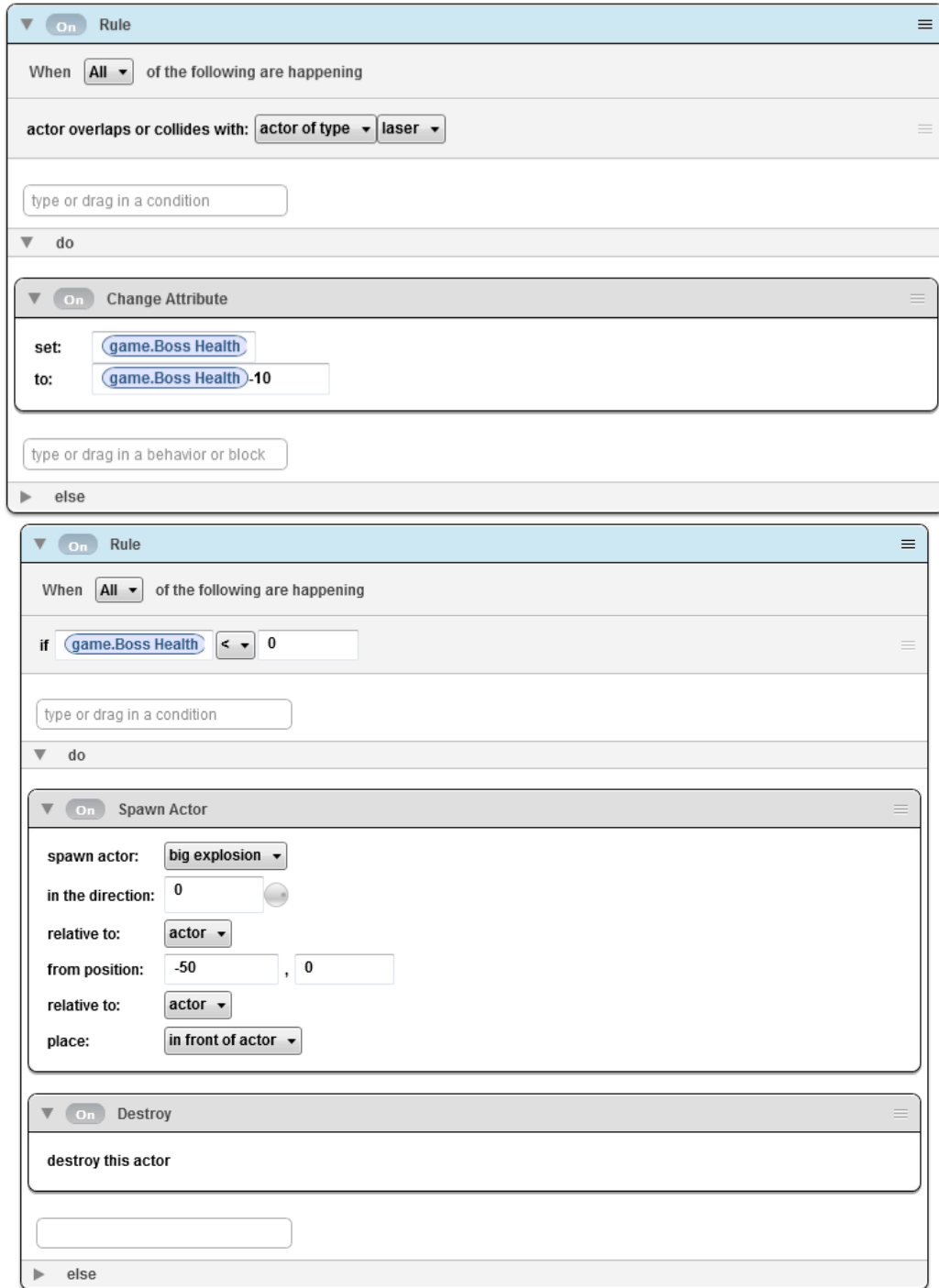
For our game, the “Boss Ship” acts independently of the player’s actions by firing a missile once a second. The Move behavior paired with the Timer behavior allows for the ‘Boss Ship’ to slowly enter the scene, before eventually coming to a full stop.

In the same way that “Player Ship” is damaged by missile collision, the ‘Boss Ship’ is damaged by laser collision, which will eventually cause it to be destroyed -- just after spawning the actor ‘big explosion’ (which has both an Animate behavior and a Play Sound behavior).

Tip: It is important to note that rules and behaviors fire off in a top-down order. This means that if you put two Change Scene Behaviors in the same Rule, the second Change Scene Behavior will never get a chance to execute, even if the Rule itself is valid and properly triggering.

If you like this effect on the boss, there’s also a ‘Small Explosion’ actor that you go back and add in Player Ship’s logic as a Spawn Actor behavior, just before the Player Ship is destroyed.





While certainly not a finished game, the above gives you an idea of some of the steps you would take when building out your scenes. Click the Play button in the Stage to see what we came up with. From here you could add depth and replayability by adding power-ups, various enemy types, an alternate ammo system, score tracking, additional scenes that scale in difficulty, and a great deal more.

Now that you have a better understanding of the fundamentals of the GameSalad Creator, you should try out some of your own ideas and experiment. See what you can come up with and remember to check out the other templates for inspiration.

At this point, the sky is the limit. Rules are capable of making any number of adjustments to your game. Animations can fire off, actors ordered to move around the scene, music or sounds played, additional actors spawned, the player's perspective can be shifted, or the value of any number of attributes can be changed. Then those attributes might cause other rules to trigger and so on, until – before you know it – you've made a game.

Thanks again for following along!

Tip: The remainder of this Creator Walkthrough goes further in-depth on information regarding the features and functionality of the GameSalad Creator. While we recommend that over time you read through this entire document, you should feel free to hop around to areas that specifically address what you are currently working on, rather than feeling that you must read it in a beginning-to-end fashion.

Scenes

Scenes are the building blocks of your game. They contain the objects of your game and provide an essential way to organize different sections of your game. For example, you may create a scene for the initial menu for the game, another for an individual game level, another to end the game, etc. You can use scenes to design and build your game in segments.

Scenes are containers for the actors in your game. You can split each scene into layers (similar to many design programs, such as Photoshop or Illustrator). Layers provide another level of (visual) organization for your game, and they allow you to group objects within your scene and arrange them in front of or behind other layers/objects. For example, one layer may contain your background, another may contain all your labels, and yet another may be for actors in your scene with which the player may interact.

Scene Attributes

The following attributes define each scene and are modifiable:

name – A descriptive way for you to refer to different scenes within your game.

time – The number of seconds a scene has been active. This attribute can be read and incorporated into your Rules/Behaviors, but not set.

size – The pixel dimensions (width and height) of the current scene.

wrap x – when enabled, actors that exit the left side of the scene will re-enter from the right side of the scene (and vice versa). When disabled, actors continue moving indefinitely off-screen unless they are explicitly destroyed (via behaviors you've set).

wrap y – similar to “wrap x” but in the up/down direction. when enabled, actors that exit the top of the scene will re-enter from the bottom of the scene (and vice versa). when disabled, actors continue moving indefinitely off -screen unless explicitly destroyed (via behaviors you've set).

gravity – the strength of gravity in the scene. The default value is 0. Using a value between 100 and 1000 will provide approximately “normal” gravity. We caution against using any values significantly above 10,000. Please note that gravity can be directed in both the X and Y directions; negative values will cause items to go in the opposite direction. Gravity affects all movable objects in the scene.

color – the background color of the scene, represented via red, green, blue, and alpha integer values between 0 and 1. You can edit any of these individual values, or you can select a color from a color picker.

camera – a compound attribute with the following sub-categories:

origin – the starting x and y position of the lower left corner of the camera relative to the scene.

size – this sets the width and height describing how much of the scene will be shown when the game is played. These values are set depending on the resolution selected in the Project Editor and cannot be modified manually. To adjust a scene size, click the “home” button, select the “Project Info” tab, and use the drop-down “Platform” menu. Selecting any of these options will automatically resize all of the scenes in the game to match that platform’s screen size.

tracking area – you can give actors within your scene a “Control Camera” behavior which ensures that the camera will follow the actor as they move through a scene. This tracking area sets the boundaries (width and height) for when to begin scrolling a scene (if possible) based on the position of an actor with the Control Camera behavior. The camera will snap to the actor with that behavior unless it would force the camera to move beyond the edge of a scene.

rotation – The rotation of the camera which changes based on auto-rotation.

autorotate – Rotates the scene to adapt to a player turning their device. For example, if a player turns their device upside down, you may want the game to autorotate to portrait (or landscape) upside-down to adjust to this new view. In contrast, if you are creating a maze or tilt game, you would not want the scene to rotate as the player tilts the device to navigate the ball (or whatever object) around the maze.

Attributes are the values (numeric or text) relating to an object that are easily and rapidly changed at any point. For instance, the positioning of the camera in a scene can be changed by altering the “camera origin” attributes, while the background color of the scene is alterable with the “color” attribute.

Several key behaviors affect which scene is currently active in your game. The foremost of these is the ‘Change Scene’ behavior.

Select any scene in your game from the drop-down menu in this Behavior. Generally, you’ll want this behavior in a rule stating some sort of precondition for changing the scene, such as an actor

reaching a goal, or achieving a certain score. When this behavior triggers, it'll change the focus of the game to the selected scene, and reset the scene you were just in. You can also use the 'Change Scene' behavior to return to a previous scene.

The Pause Game behavior functions similarly, but also has a few key differences. Instead of resetting the current scene, it simply pauses everything in it, and opens the selected scene.

Use the Unpause Game behavior in the scene you selected in the Pause Game behavior to return to the original scene. Typically, you'll want to place this behavior in a rule stating that a button (Unpause or Resume Game) has been pressed, or something similar.

Finally, we have the "Reset Scene" and "Reset Game" behaviors. The former will reset just the current scene and that scene's attributes, while the latter will reset all scenes and attributes in the game. It is also useful to be aware of the Camera controls. By clicking the camera image, we can adjust the camera zone. The camera zone is the movement leeway an actor has before causing the camera to move. In other words, suppose we have an actor that controls the camera's movement. Generally, as the actor moved through the scene, the camera would follow. But if the actor turned around and headed in the opposite direction, we would want it to move a small distance before the camera started to follow. That distance is the movement leeway, which is controlled with the camera zone. Use the handles to reduce or increase this margin; basically the actor controlling the camera will be able to move freely in the central black area before bumping into the camera zone and moving the camera along with the actor. To return to scene editing mode, simply click the arrow to the left of the camera image.

Actors

Actors can represent the character that the player is controlling or they can be the surrounding objects/characters that your player talks to, collides with, jumps over, or generally interacts with during gameplay.

Game designers often begin their process in Creator by creating the set of actors that they will need in their game across different scenes. These are known as 'prototypes' (or models) of the actual actors/characters instances that your players will interact with within your scenes. For example, in Pac-Man, you might create a 'ghost' prototype, specifying some visual attributes of the ghosts, and how the ghosts would behave in the game. Each ghost that actually appears in each scene/level of the game is a specific instance based on this ghost prototype (and would therefore inherit all the attributes and behaviors of the prototype). If you make a change to the ghost prototype, all locked ghost instances within the game would also be changed.

Instance actors are the building blocks of a scene. To create an instance actor, simply drag a prototype actor into a scene. Any changes you now make to this specific actor within the scene will only affect that instance actor, not the prototype itself.

General Attributes

name – a descriptive way for you to refer to an actor within your game.

time – the number of seconds an actor has been active or “alive” in the scene. This attribute can be read and incorporated into your rules and behaviors, but not changed.

position – the x and y position of the actor in the current scene. This property is relevant for instance actors within a scene, not prototype actors.

size – the pixel dimensions (width and height) of the current actor.

rotation – specifies the angle or rotation at which the actor appears initially in the scene. For example, if the value is 90, then the actor appears rotated 90 degrees counter-clockwise.

color – the background color of the actor, represented by red, green, blue, and alpha integer values from 0 to 1. You can edit any of these individual values, or you can select a color from a color picker.

Image – the image displayed for the actor (if any). This is not manually editable, but you can modify this by dragging an image from your artwork/sprites on to the actor instance within the stage.

tags – tags given to the actor.

Graphics-Related Attributes

visible – determines if an actor will be seen by the player. If checked, the actor will be seen. This attribute cannot change during gameplay. Unchecking this attribute will improve the performance of your game if the actor never needs to be seen. One reason you might have an actor who which doesn't need to be visible is if it only contains logic for playing sounds or music.

blending mode – determines how the actor's graphics will be drawn with respect to its background.

normal – overlays the actor over the background. This is the default blending mode. When in doubt, use this one.

opaque – copies the actor's pixels exactly, replacing the background and ignoring transparency. Opaque mode is the cheapest blending mode, so using it may improve the performance of your game, especially for large background images.

additive - adds the color value of each pixel of the image to the color value of the image behind it. This has an intense brightening effect, and is commonly used to draw emissive light sources (fire, sparks, explosions, lasers, and other things that are awesome)

screen – similar to additive. Instead of adding the actor and background colors, screen adds the actor color to the inverse of the background color. This results in a more realistic brightening effect, with less ‘overexposure’ of the image.

multiply – multiplies the color values of the actor’s pixels with the background. This tints or darkens the background image, and is useful for gradient overlays, tinted windows, shadow effects, or masking. Using transparency combined with multiply will have strange results, and should usually be avoided.

horizontal wrap/vertical wrap, which has three options:

stretch – the image will be stretched or compressed to fit within the boundaries of the actor.

fixed – the actor will show its image at the size/resolution of the file. The image will be centered to the actor and the actor can be made smaller or bigger than the image resolution without changing the look of the image.

tile – the image will be repeated if the actor is larger than the image resolution.

Motion-Related Attributes

linear velocity – the speed of an actor in a specific direction, specified in the X and Y directions.

angular velocity – the speed at which the actor rotates. Values greater than 0 cause the actor to rotate counter-clockwise. Negative values cause clockwise rotation.

max speed – the value of the maximum speed the actor can go if Apply Max Speed has been selected.

apply max speed – if checked, the actor will be limited to the speed set in Max Speed. Otherwise, the actor will continue to increase in speed.

Physics-Related Attributes (Non-Motion Physics)

density – the heaviness of the actor. A higher value will make the object harder to move by less dense actors. A value of 0 will make an object immovable but still affect other actors in the scene. Density can be set to any real positive number. (i.e. Density ≥ 0)

friction – specify how much the actor gets slowed down each time it contacts another actor. 0 is smooth; larger numbers make the actor slow down more quickly. Friction can be set to any real positive number. (i.e. Friction ≥ 0)

restitution – number that describes the bounciness of the actor. 0 is no bounciness. Restitution can be set to any real positive number between 0 and 2.

fixed rotation – specify whether or not the actor rotates when it collides with other actors in the scene

movable – this determines if the actor can be moved, including collision from another actor and gravity.

collision shape – an actor can have a rectangular or circular collision area. For a circular collision area, it is the largest circle that can fit entirely in the dimensions of the actor.

drag – this gives an actor linear drag. It will gradually slow down movement for an actor, if there are no other forces (movement behaviors) modifying the motion of the actor.

angular drag – this gradually slows the rotation of an actor, if there are no other forces (rotation Behaviors) modifying the motion of the actor.

New actor attributes may be created and removed using the (+) and (-) buttons of the Attributes panel. The newly created attribute will appear highlighted near the bottom of the attributes list. These attributes can be accessed from the 'Attributes' tab of the Backstage Browser, and be changed via the Change Attribute behavior unless otherwise noted.

As mentioned earlier, modifying a prototype actor affects all corresponding locked instances of this actor. You can customize specific instance actors by double-clicking an instance actor (that is, an actor placed in a scene). If you select an instance actor, the Rules editor will initially be locked so that you do not accidentally modify the logic of your instance actor to differ from its prototype. However, you can simply click on the lock icon to unlock the actor and allow editing. You can revert to the prototype's rules by pressing the Revert to Prototype button at the top of the Rules editor. If you do unlock an actor instance, any further changes to the prototype actor will not affect the specific instance.

Actors can be added to your scene in a couple of different ways. First, you can have your actors start in the scene. To do this, simply load up the desired scene in the scene and drag your actor to where you want it. Second, you could have your actor start "off-scene", but enter the scene at some point during gameplay. You'll notice that if your scene size is larger than the camera size (dictated by the platform you've chosen), there will be areas of your scene that are outside the scope of the camera. You can place actors here and have them move into the scene using a variety of movement behaviors, which are discussed later in this document. Finally, you can "spawn" actors using the Spawn Actor behavior. This will place the actor at a designated location and velocity in your scene.

Behaviors

Behaviors are actions that you can assign to actors to control how they interact, move, and change appearance. Behaviors are used to add logic to your game to control what happens when some event occurs (such as when an actor collides with an object), to make the actor take some action (such as changing its speed) or to change an actor's appearance, such as its size, image, color, or transparency.

There are four types of behaviors:

Blocks – Behaviors whose primary purpose is contain or consider other behaviors.

Conditions – Primarily inputs, such as key board presses, touches, collision detection, and mouse clicks.

Persistent – Behaviors that continue to act on an actor continuously (unless placed in a rule whose conditions are no longer true). These are indicated by a [B] next to the behavior name.

Actions – Behaviors that occur once, and only repeat themselves if placed in a rule whose conditions become false, and then true again. These are indicated by an [A] next to the behavior name.

There are currently 36 standard behaviors; they're listed below for your reference. You can also see each of these behaviors listed in the dictionary at cookbook.gamesalad.com/definitions, along with other terms.

Accelerate – Use Accelerate to specify the rate and direction of acceleration for an actor. If the drag attribute or gravity is not also applied to an actor, acceleration will be continuously applied, increasing the actor's speed until it reaches the maximum defined speed, if any. See also the "Accelerate Toward" Behavior.

Accelerate Toward – Use Accelerate Toward to specify the rate of acceleration and the targeted location of an actor. Use the expression editor to specify a static or moving target position. If the drag attribute or gravity is not also applied to an actor, acceleration will be continuously applied, increasing the actor's speed until it reaches the maximum defined speed, if any. The actor will continue accelerating past the target location along the existing trajectory unless slowed through other Behaviors or attributes. See also the "Accelerate" Behavior.

Animate – Drag and drop a sequence of images into this Behavior from your project library. Once added, images can be reordered, and additional images can be inserted. The controls in this Behavior allow you to specify the frame rate of the animation, up to 30 frames per second; whether or not the animation loops, stops at the last frame, or returns to the last image used before the animate Behavior started.

Change Attribute – This Behavior allows you to set, change, or increment a game, scene, or actor attribute. It can be used to change a numerical value, color, size, movement, acceleration, or any other value determined by an attribute.

Change Image – Specify an image to replace the current image on an actor. You can either drag the new image to this Behavior or specify an image from the drop-down menu.

Change Scene – This Behavior will stop the current scene and immediately move to the designated scene. It's best to place this Behavior inside a Rule that changes to the game credits scene, menu scene, or to a new game level after certain objectives are met.

Change Size – This Behavior changes an actor's size by a scale factor (use a negative number to shrink an actor). Note: the actor's original size will still determine its collision volume; to change this, use "change attribute" or "interpolate" instead. Use a timer to specify the amount of time that the transformation should take to occur.

Change Velocity – Specify the direction of movement for the actor at a constant designated speed. Once the direction is specified, other influences on movement will begin to affect the actor, such as drag, gravity, or other movement Behaviors.

Collide – This Behavior controls which actors or groups of actors will collide with each other. A group of actors may be created by using a tag in the project editor.

Constrain Attribute – This Behavior continuously updates the value of one attribute to match another attribute. This is particularly useful to keep two objects moving in sync, or to keep an actor tied to the movement of the mouse or touch.

Control Camera – Add this Behavior to an actor and the scene’s camera will scroll to follow. The tracking area for the camera can be changed in the Stage using Camera Edit mode. Only one instance actor per scene can have this Behavior; actors in non-scrollable layers cannot use the Control Camera Behavior.

Destroy – Immediately removes the actor from the scene.

Display Text – This Behavior will show the text entered in the box, and includes controls over the color, alignment, font, wrapping, and size of the text displayed. Wrapping will cause line breaks in order to keep all of the text inside the actor.

Group – This is an organizational Behavior, which allows you to group certain Behaviors and Rules together easily and clearly. It can also be created by selecting the “Create Group” button.

Interpolate – This Behavior allows you to change attributes from their existing value to a new value over a set period of time. Interpolate will use a constant rate of change over the designated time period. This Behavior can affect a rapid or gradual change in any game attribute, and cannot be obstructed or stopped by any other Behavior.

Load Attribute – Loads the value stored with a custom key name using the Save Attribute Behavior. A key is basically a storage location for a specific attribute. Use any key you want when saving an attribute, and then use the same key to load that same information later.

Move – Use to move in a particular direction relative to the actor or the scene at a specified velocity. Additive movement allows multiple move Behaviors to stack, or act on an actor simultaneously; Stacked movement causes only the most recent active movement Behavior to control the actor.

Move to – Use to move towards a specific X/Y coordinate relative to the actor or to the scene. This movement will stop upon arrival at the designated coordinates unless the controlling conditions are no longer valid and “run to completion” is not checked, in which case the movement Behavior will cease as soon as the controlling conditions are no longer valid.

Note – This Behavior allows the developer to record reference notes explaining a Rule, Behavior, group, or other aspect of the game. These will not be visible in or affect the operation of the game.

Particles – Spawns a designated number of particles from behind the actor. This Behavior includes options for color, size, lifetime, velocity, images, and more.

Pause Game – This Behavior will pause the current scene and display the scene selected in the Behavior over the current scene. Using the Unpause Game Behavior removes the scene and resumes the original scene.

Pause Music – This Behavior will pause the currently music track, if one is playing. Use the “play music” Behavior to resume the track.

Play Music – This Behavior causes the selected music file to start playing. Select “loop” to cause the selected music to begin again once it has played through to the end.

Play Sound – This Behavior causes the selected sound file to start playing. Select “loop” to cause the selected sound file to repeat each time it completes; select “run to completion” to prevent other Behaviors from interrupting the sound before it has played through to the end. “Positional Sound” and “Velocity Shift” will affect the volume and pitch of the sound as the actor controlling the sound effect moves through the scene.

Replicate – This creates duplicates of an actor without actually spawning additional actors into a scene, based on the value of an attribute. It is most commonly used to display the number of lives a player has left.

Reset Game – Resets the game and all the scenes in it. This will restore all attribute values to their original state, but will not delete keys saved using the “Save Attribute” Behavior.

Reset Scene – Resets the current scene and all the actors in it. If placed in the scene that appears during a pause, will not reset the underlying paused scene.

Rotate – Causes the actor to spin clockwise or counter-clockwise at the speed specified in the expression editor. The “rotate to angle” and “rotate to position” Behaviors perform similar, but unique tasks.

Rotate to Angle – Causes the actor to spin clockwise or counter-clockwise at the speed specified until it reaches a particular angle, at which point rotation will cease. Unchecking “stops on destination” will cause this Behavior to act similarly to the “rotate” Behavior.

Rotate to Position – Causes the actor to spin clockwise or counter-clockwise at the speed specified until it reaches the designated X/Y coordinate, at which point rotation will cease. Unchecking “stops on destination” will cause this Behavior to act similarly to the “rotate” Behavior. Use “Offset Angle” to rotate to a position a designated number of degrees from the specified X/Y coordinate.

Rule – Creates a condition or set of conditions to check before activating an enclosed Behavior. These conditions include player input (mouse clicks, touches, key presses) and attribute values. Rule also includes an “otherwise” section; Behaviors placed here will trigger whenever the conditions in the Rule are not true.

Save Attribute – Stores the value of an attribute with a custom key name. Any key name can be used; inputting the key name in the “Load Attribute” Behavior will yield the stored value. Saving a new attribute value with a previously used key name will result in overwriting any

existing saved data. Values stored using "Save Attribute" will remain accessible, even if the game or device is turned off.

Spawn Actor – Creates a new actor instance in the scene. Specify which actor to spawn and the directional and position of that actor relative to the scene or spawning actor. This allows any actor in a scene to spawn additional actors anywhere else in the same scene. Newly-spawned actors will immediately begin following any movement or other Behaviors associated with them.

Stop Music – This Behavior stops the current music track. Unlike the "pause music" Behavior, this Behavior resets the track, so that a "play music" Behavior acting afterwards will start the music track from the beginning.

Timer – This Behavior allows you to activate Behaviors or Rules at specified intervals. All timer values are in seconds. "After" triggers the Behavior or Rule once after the given time period; "every" triggers the Behavior repeatedly with a given delay between each trigger; "for" keeps a Behavior active for the duration of the time.

Unpause Game – If the 'Pause Game' Behavior has activated and opened the pause scene, this Behavior will remove the pause scene and resume the underlying paused scene.

Attributes

Attributes are one of the most powerful aspects of GameSalad. They are essentially value-holders or variables; they're designed to store numerical or text values to be used for different situations. They can easily be used for anything from the angle at which a cannonball is launched to displaying the name of a character in a game to recording a rapidly-changing score. Understanding how to use attributes properly is crucial to unlocking the full power of GameSalad.

Any time you need to record a number (or an angle, or a piece of text), creating a new attribute is probably the best way to do it. If you're struggling with this concept, focus on the numeric aspect of attributes. For instance, say you are building a racing game. Creating an attribute to record the speed of your racer gives you the flexibility to easily change that speed in a way that can be referenced in a variety of situations, such as accelerating, braking, and collisions. Similarly, another attribute might be used to record the amount of time it takes to go around the track, or your final score.

GameSalad comes with a large number of existing attributes in every game project, storing information about actors, scenes, and the game itself. This preset list of attributes controls certain base activities, such as scene size, actor motion, and game time. There are also options for creating custom attributes for any information you may wish to store as the developer. We'll touch on existing attributes briefly, and then discuss how to use custom attributes for anything in your game.

Attributes exist for your game, your scenes, and your actors. Game attributes can be accessed by all prototype actors and instance actors so that they can be seen in any scene. Scene attributes can only be accessed in a particular scene but all instance actors in that scene can individually be edited to view, use, and modify those scene attributes. Actor attributes can only be accessed by an instance actor, but can be initially defined and used in an prototype actor so that all instance of that prototype can have predefined behaviors.

There are six different types of attributes:

Boolean – these are true/false values. One example use for this could be the status of a button or door, where “true” would be open/on, and “false” would be closed/off. All Actors start with at least one boolean attribute by default under “physics” in their attribute list to toggle whether or not the actor is movable. Boolean attributes can be changed to true or false with the “change attribute” Behavior.

Text – letter and number values. These are primarily used in conjunction with the “display text” Behaviors to display scene titles, actor names, and much more.

Integer – whole number values, such as 15, 0, 42, and 5801. Integer is one of the more versatile attributes. A few of the many possible uses are storing the game’s score, specifying the number of units to spawn, or keeping track of how many reward objects are left in a scene.

Real – allows for decimal values, such as 3.582, 0.882, and -2.5. Obviously integers can work here as well. Real attributes allow developers to include fractional events, such as time ticking down (or up), pouring a volume of liquid, or hit point/damage records where fractions of a point can be dealt.

Angles – allows for values from 0 to 359, representing the degrees of a circle. You can use decimal values here as well within this range. Angle attributes are primarily used with objects that rotate or have some sort of angular movement, including launchers, such as the cannon in our “cannon physics” template.

Index – positive whole numbers, such as 0, 1, 58, 2804. Index attributes can fulfill many of the same roles as Integers, but have the interesting feature of being unable to store a negative number. For instance, an index attribute with a value of 4 that was told to subtract 6 would then have a value of 0, not -2. These will primarily be useful once GameSalad supports advanced features such as Arrays and Tables.

Existing attributes in your game can be viewed in the Attributes panel (located in the Backstage) after selecting the appropriate tab. These existing attributes record data about your game, scene, and actors, such as their size, location, color, and more. However, custom attributes contain data that is unique to your game - how fast your racecar is moving, or how many blocks remain in a puzzle. Simply click the “+” button at the top of the list of attributes, and rename as desired.

Because you’ll be accessing the value held by this attribute from actor behaviors, you may want to rename it something that makes sense to you and you’ll be able to remember later - such as “race car speed”, or “blocks remaining”. At times, you’ll want to record the location of an actor - you can do this through attributes as well, and these attributes might be named something like “PlayerX” and “PlayerY”, to record the X and Y coordinates of the player.

Tables

Creating a Table

Select the 'Game' tab in the Library panel and click the '+' button located beside the 'Tables' category to create a new table.

Editing a Table

Click on the table you just created in the 'Game' tab and you'll notice that a new tab is now brought up in the Backstage.

To the left of the words "Rows" and "Columns", there is a numerical value in an editable field. Click the box to change the value, and enter the number of Rows or Columns you'd like to have. You'll see each column in your table has a drop-down menu where you can select an attribute type. To change the type of column, select the drop down arrow and click on the type of attribute you'd like. If you start with one type of data and change it later, GameSalad will attempt to convert all of the cells in that column to the new attribute data type. To add information to a cell, make sure you've selected a column/attribute type, as well as set a number of rows you would like to utilize. Simply click on the cell and begin typing.

To Import a .csv file into GameSalad, navigate to the table you would like to import information to, and open it. Click the icon that has two arrows pointing in opposite directions, located on the right side of the window. Select the CSV you would like to import, and click "Open".

Select the type of information you're importing and click "Import".

Using Data in a Table

Navigate to the actor you would like to reference the data, and give it a behavior or rule that utilizes the expression editor. Click on the "e" to open the expression editor. Click the "Insert Function" drop down menu. Click the "TableCellValue" function Within the function `tableCellValue(table,row,col)` you can now enter the information accordingly. Navigate to the Game Attributes Browser and specify which of your tables to use in the first "table" coordinate. Specify the row and column coordinates. Please note that these can be modified and randomized by using additional functions and attribute references.

Additional Table Functions

There are two additional table functions that can be used to gather table information. "TableCol-Count" is used to get the maximum number of columns in the specified "table" field "TableRow-Count" is used to get the maximum number of rows in the specified "table" field.

These two functions are very helpful for pulling random data from the table, allowing the range of values to be highly customizable.

Publishing With GameSalad

Whether you're just putting the finishing touches on your latest amazing project and are ready to share your game with the world or you're still hard at work making it great and would like to ad-hoc test it on an Android device, you'll first need to publish your handiwork through the GameSalad Creator.

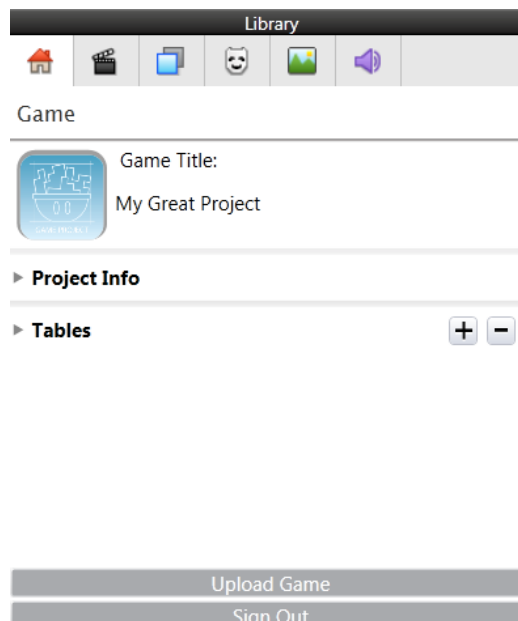
Tip: It's important to test publish your project prior to your scheduled due date. This gives you the opportunity to ad-hoc test your project and also gives you time to address any issues that might be preventing you from publishing, should you encounter any. Most App Stores require at least 2 weeks of time before you want to release your application, so plan accordingly.

For those of you who are familiar with the Mac version of Creator, please note that this process will be different in key areas, so we encourage you to give this section a careful read through.

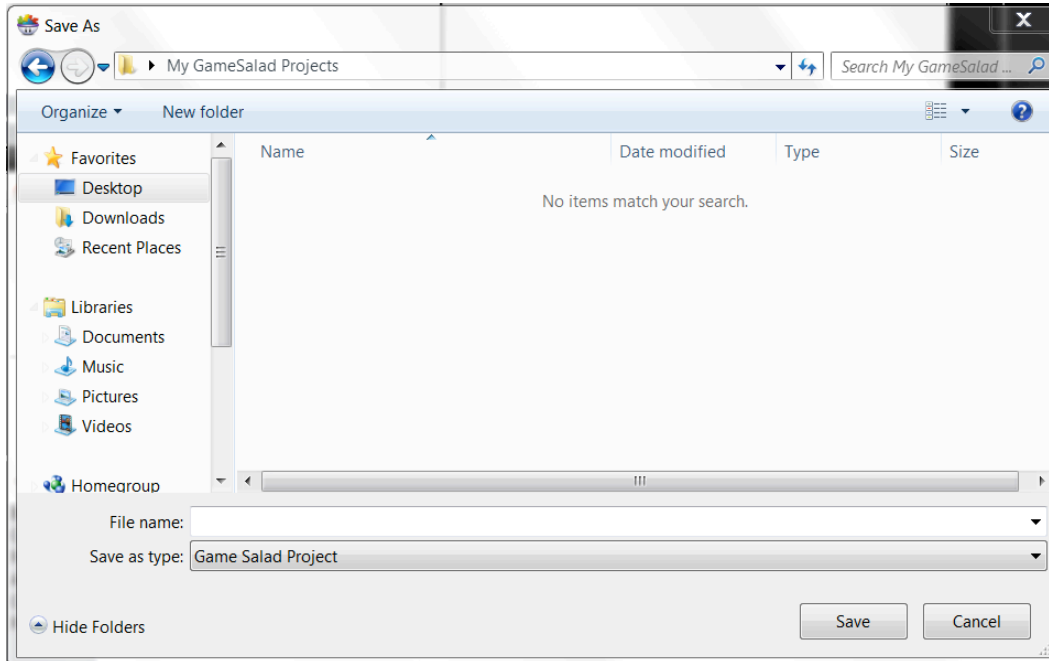
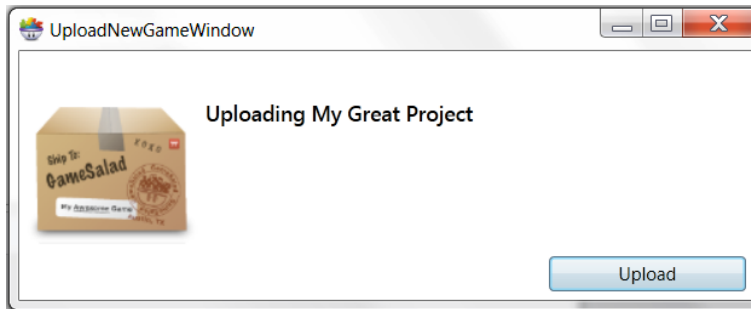
If you are one of the many new users who are working with GameSalad now that it's available on the PC – welcome! Below we'll be going step-by-step through the process of publishing your project for HTML5 through the GameSalad Arcade. In addition to HTML5 publishing, which is available to all users, Professional users also have the option to publish for Android devices for both testing and distribution purposes. To publish to iOS you will need to utilize Mac Creator because Apple requires the Mac platform for the signing process.

Once you're ready to publish (whether your game is finished or still a work-in-progress) the first step is to bring up the project file you've been hard-at-work on in the GameSalad Creator.

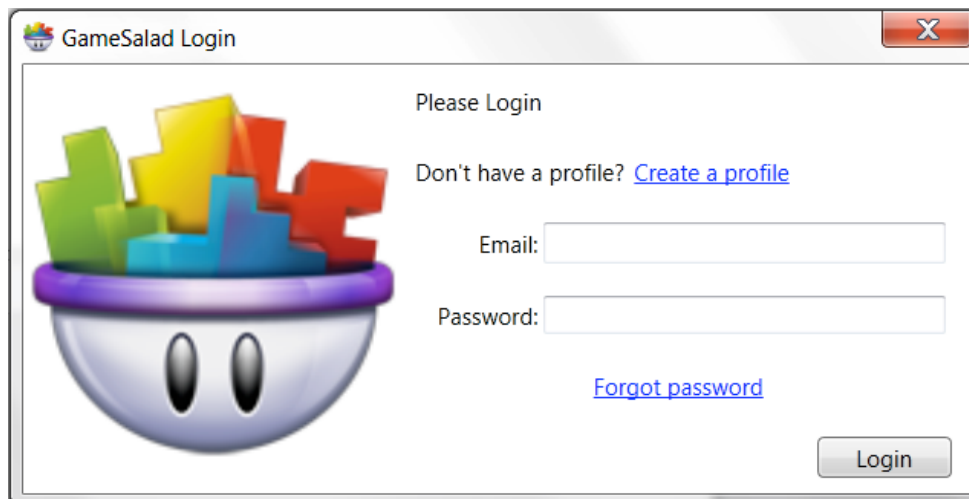
While on the Game tab in the Library panel of the Creator, you'll see the option to publish in the form of a 'Upload Game' button, just above the 'Sign In'/'Sign Out' button.



Clicking 'Upload Game' will prompt you to confirm the upload, after which you'll be asked where you would like to save a copy of your GameSalad Project file on your local machine. Choose a file name and hit 'Save' to continue on.

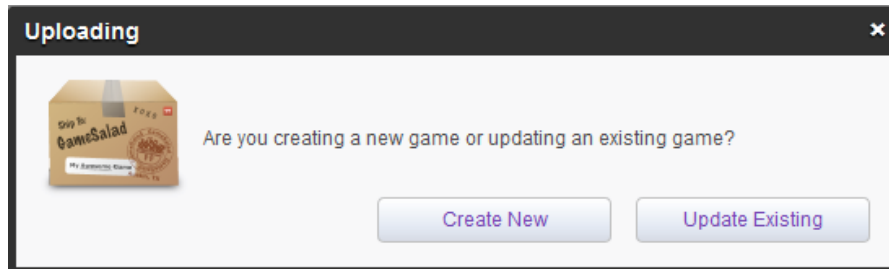


Note that if you were not already logged in to your account at time of upload attempt, you'll be asked to sign in which is a requirement for publishing. If you've forgotten your password – don't worry! Just click the 'Forgot password' option and a website will open walking you through the password reset process.



After saving your Project, your project will begin uploading to the GameSalad servers. Once this is complete, your default browser will bring up the second part of the publishing process, which is managed through the GameSalad website. You'll also be asked to sign into the website as well, if you weren't already.

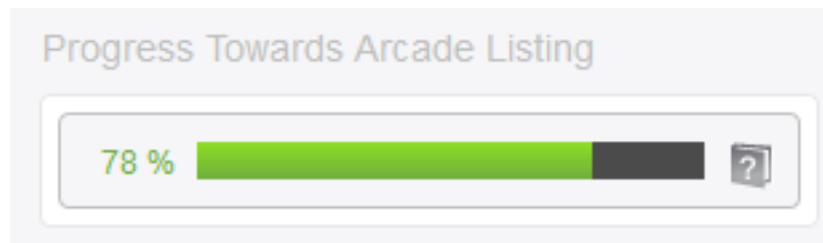
If this is your first time publishing with Windows GameSalad Creator, go ahead and select the option to create a new game. Later, you can use the 'Update Existing' option to swap out an older out-of-date version of your project file with a version that has the latest features, improvements, and bug fixes, rather than creating a new entry for each step forward your game takes. Keep in mind that updating an existing game will write over the existing file on the GameSalad server; so make sure you keep local copies of the various versions of your game!




Publishing for HTML5

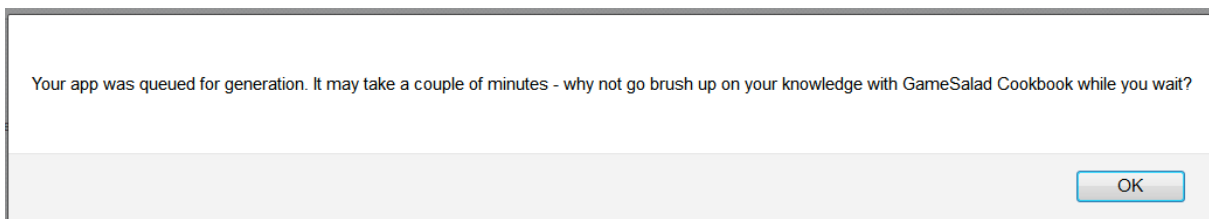
The General Info tab contains several fields, which allow you to provide important information about your project (title, description, instructions, tags, etc) as well as upload screenshots and an icon.

Providing these details and assets about this project will gradually fill up the progress bar in the upper right-hand corner of the window. Once this bar reaches 78%, you'll be able to list your game on the GameSalad Arcade!

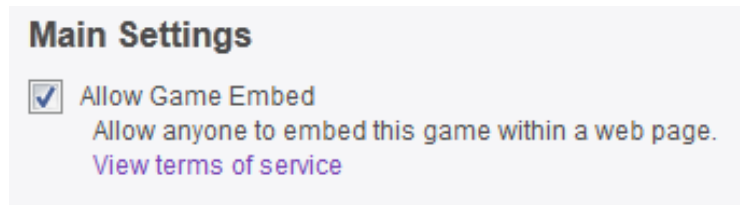


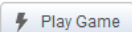
While we encourage users to fill out the entire form for presentation reasons, you actually only have to provide the following items to get your game included in the Arcade: game title, icon, and a screenshot.

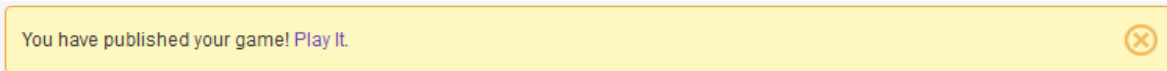
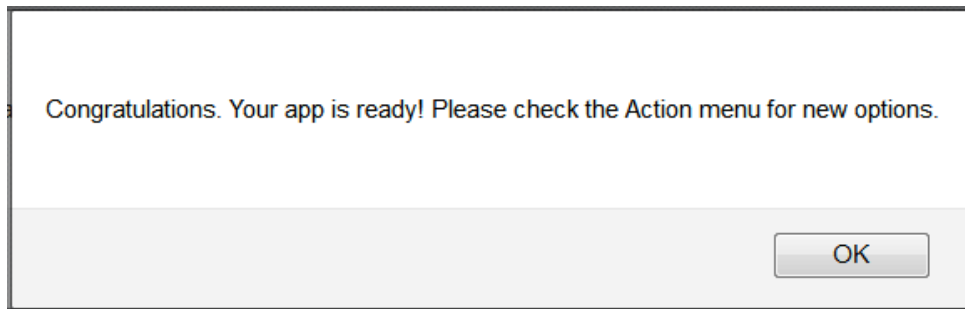
Once you've filled out the above items, click on the HTML5 tab under Platforms on the left side of the page. The 'Publishing' button on the right hand side should now be intractable, since you hit the 78% threshold: 



Congratulations! In a few minutes your game will be playable worldwide on the GameSalad Arcade. If you'd like to take it a step further and embed your game into a website or allow others to do so, you can enable 'Allow Game Embed' under Main Settings.



Once your Arcade game goes live, you'll receive both a pop-up notification and in-window notification that includes a link. In addition, the 'Play Game' button should now be clickable: 



Publishing for Android








Pro users can take Windows Creator even further, by publishing signed and unsigned .apks that can then be submitted to app distributors, allowing you to earn revenue off your hard work and talent.

Unlike HTML5 publishing, your Android publishes will not be made publicly available on the GameSalad Arcade, preserving exclusivity for you and any application distributors you choose to use.

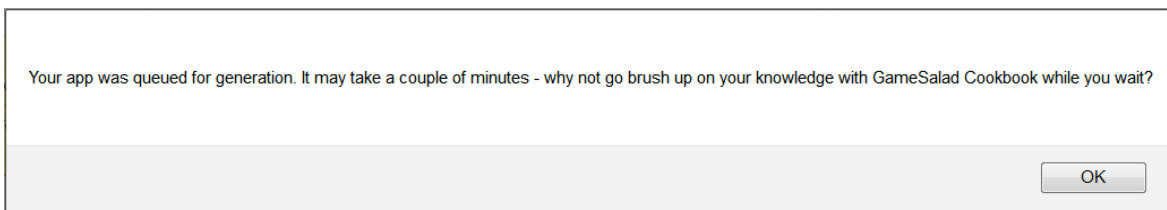
That said, publishing for Android is significantly more complicated than HTML5 publishing, due to a number of configuration and set-up requirements that are specific to the platform. Don't let that throw you off though, since this next section will provide you with the fundamentals for successfully generating your .apk file.

First, fill out the blank fields under the 'Main Settings' header using the formatting guidelines provided by the tool-tips which appear when you mouse over the book icons: 

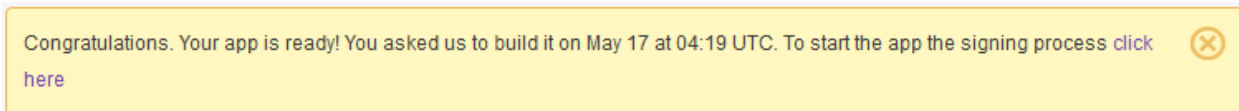
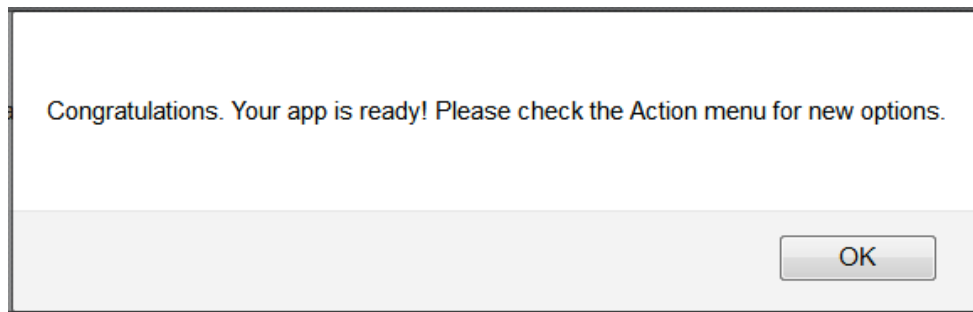
Main Settings

Android Package Name 	<input type="text" value="ex: com.mystudio.mygame"/>	
Display Name 	<input type="text" value="ex: My Game"/>	
Android Version Code 	<input type="text" value="ex: 1"/>	
Publicly Visible Version 	<input type="text" value="ex: 1.0"/>	

Once you've filled in these fields, you'll notice that 'Generate APP' has become selectable. Click this button to queue the generation of an unsigned .apk, which should become available within a couple minutes.



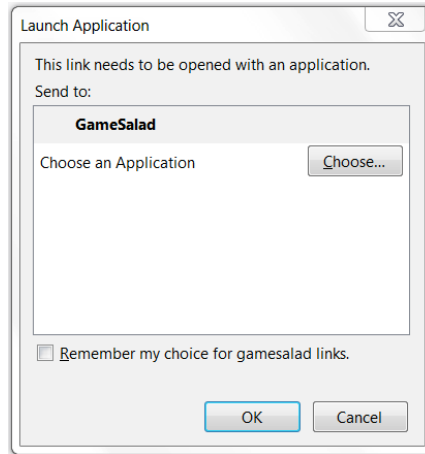
Once your unsigned .apk is ready you'll receive both a pop-up notification and in-window notification that includes a link to start the signing process. In addition, the 'Download' and 'Sign App' buttons should now be clickable.



At this point in the process you'll be able to use the 'Download' button to download the unsigned .apk. For the most part you'll need a signed .apk to do things like test your application on a device or submit it for third party app store distribution. However, this isn't true for all cases. For example, Amazon requires that you submit an unsigned .apk, as they prefer to sign the file on their end.

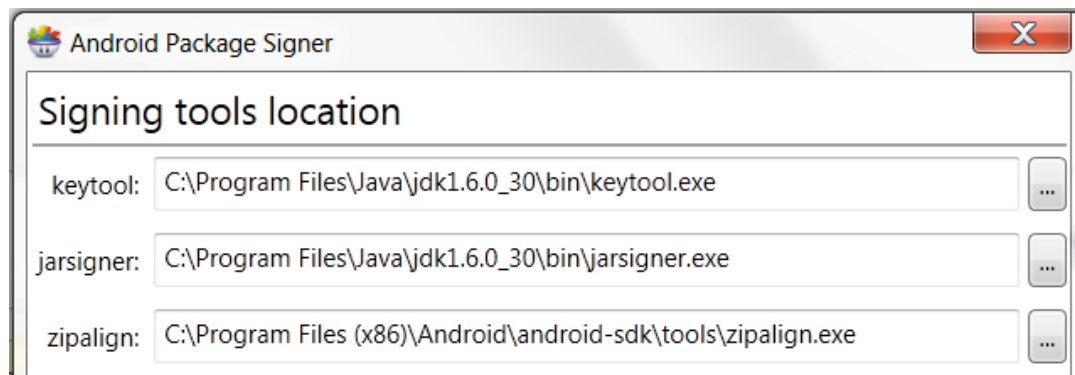
Signing an .apk is a more involved process than creating an unsigned .apk, so at this point this walk-through we recommend that you set up your PC environment using the steps outlined in the Cookbook entry Preparing Your PC for Android Development ([Part 1](#) and [Part 2](#)).

Once you've followed the steps outlined in the above link, click 'Sign App' which will ask to launch through the GameSalad Creator.



This will bring up the Android Package Signer, which is divided in to four parts: Signing tools location, Keystore, Key, and Package to sign.

For 'Signing tools location', use the browse icon next to each field to locate the appropriate .exe. While the specific location of each of these files may differ from computer to computer, you'll likely find these files in the following locations (after you've prepped your PC for Android development, of course).




Now, hit the 'New...!' button to begin creating your first PC Keystore. Choose an easy to find (and hard to forget) location for your Keystore.

Now simply fill out all the remaining fields! Your Keystore and Key passwords can match, but both must be at least 6 characters in length. The final section's (Package to sign) field should automatically be populated with the address of the file to be signed. Hit the 'Sign...!' button, and choose a safe spot to save your signed .apk.

Now you're ready to test your game on an Android device, prior to submitting the signed .apk to app distributors. For more information on how to go about this process, check out the Cookbook entry titled [Prepping Your Device](#).

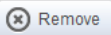
Download and Remove Options

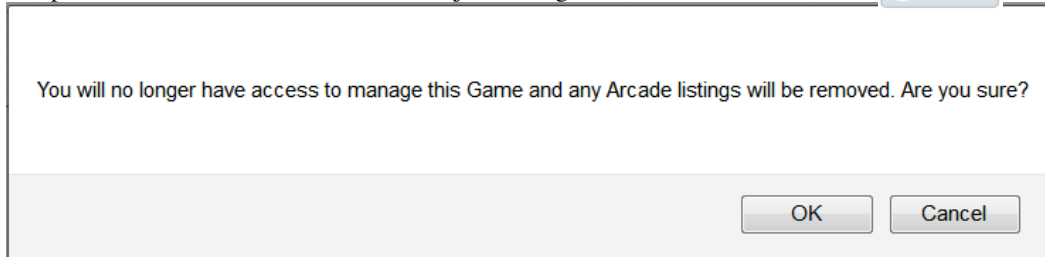
There are two other options on this page that are worth highlighting. The first one is the 'Download' button, located directly under the progress bar while in the General Info tab:  Download

If you own or have access to multiple PCs, you can use this feature to upload your project on one computer and then download a copy of the originating project file on to another. To access this page without having to go through the

first portion of the upload process, log in to your account on the website, then go to <http://publish.gamesalad.com/games>. From there, simply click the icon of the game you'd like to download, and then click the 'Download' button. Simple and handy!

Please note that the Download button located in the General Info tab is different from the Download button found in the Android Tab, which will provide a signed .apk to Pro users.

The second option is the 'Remove' button, located just the right of the 'Download' button: 



Clicking this will bring up a removal confirmation prompt. Clicking 'OK' will remove your project from the Arcade, which will in turn break any links (embedded or otherwise) that point towards your game. You will receive no further warnings or opportunities to change your mind after you confirm, so consider carefully and be sure to double-check that you an up-to-date copy of the project file backed-up locally!

Tips and Best Practices

- Save early and often! In addition, be sure to use some form of versioning (saving multiple files of your project) and to also keep copies of your project in separate storage devices (such as a flash drive or a second computer). This will allow you to be prepared against hardware failure, data corruption, or unintended alterations to your project. Should the worst come to pass and you find that you can no longer access your project, you may be able retrieve a version of it from the GameSalad servers if you've successfully published it at some point in the past. This recovery process is covered in more detail in the FAQ section of this manual.
- Keep notes and detailed documentation! While often overlooked, this can save you a lot of head scratching and confusion down the line, especially if your project deals with a large number complex interdependencies amongst actors and attributes.
- Don't get frustrated! Roadblocks happen to all of us from time to time. When confronted with a problem that doesn't appear to have a straightforward resolution, tackle it from multiple angles. You might also consider searching the forums to see if other GameSalad users have already found a solution to your issue. Should you decide to create a post with your question, please keep in mind that you're more likely to receive a helpful response if your post is in a positive tone and comprehensively explains the situation.

Frequently Asked Questions

Q: Can I publish to iOS from Windows Creator?

A: Apple currently requires app binaries to be published on a Mac, using developer certificates. At present, iOS projects must still be built through GameSalad Creator for Mac and then submitted

through your iTunes Connect Developer Account using your developer certificates. We're currently exploring alternatives to provide a solution for Windows users, and will have more information in the coming weeks and months.

Q: How do I transfer a project from Windows Creator to Mac Creator?

A: You can freely do this transition without having to worry about version numbers. All Windows projects should open up on a Mac. To transfer your project from Windows to Mac:

1. Open your project in Windows Creator
2. Select the 'export' option in the file menu
3. Take your exported project over to a Mac computer
4. Open the project in Mac Creator

Q: I'm still on Windows XP, do I have to update Windows?

A: GameSalad Creator for Windows is not currently supported on Windows XP. We understand that many PC users still actively use XP as their OS of choice. We are working to provide XP compatibility and hope to provide that supports in a future update.

Q: Do I own the rights to my project if I made it using the GameSalad Creator?

A: Yes! From the Terms of Service: “As between you and GameSalad, you retain ownership of all your copyright and/or other intellectual property rights applicable to, or embodied in, your submissions.” That said, please take the time to familiarize yourself with all your rights and obligations as defined by the Terms of Service. By visiting or using the GameSalad website, Creator, or associated services, you agreed to the Terms of Service as detailed at gamesalad.com/terms.

Q: What are the benefits of subscribing up for a GameSalad Professional Membership?

A: Professional developers have access to innovative methods of monetizing their apps, additional advanced Behaviors (such as Game Center Leaderboards and Open URL), are able to publish for Android, and can expect a faster response when reaching out to Customer Service. Full details of these benefits can be found at gamesalad.com/membership/pricing.

Q: How do I access Pro Behaviors, once I subscribe as a member?

A: After signing up for Pro membership, log out of and then back into GameSalad Creator. Pro Behaviors are located inside in the 'Behaviors' tab in the Backstage, under the 'Pro' section, just to the below of the 'Actions' and 'Custom' tabs. If these options aren't appearing, please wait a couple hours and try again. If at this point the Pro Behaviors still appear to be inaccessible, we recommend contacting GameSalad Customer Support for additional assistance at arcade.gamesalad.com/feedback. Please note that if you've opted to pay by e-check, that it may take up to several days for access to be granted while check clearance is pending.

Q: Why does my project run differently in Creator Preview Mode than on other devices?

A: Each environment has a unique hardware and software configuration – whether PC, Mac, iOS, or Android Device – which can affect how your project will perform. We recommend ad-hoc testing your project on all devices you intend to distribute for, since this is the most accurate form of testing.

Q: Can I use my own supplementary code in my project to add in new features or customize to customize existing features?

A: GameSalad’s philosophy is “Game development for everyone”, and so we’re focusing our own internal development efforts to provide features to users that don’t require coding. That said, we are always interested in hearing your feedback, ideas, and feature requests. These comments can and do factor into our scheduling process, and you can share your thoughts with us on our forums at forums.gamesalad.com/categories/new-features. For full details on what uses are permitted with GameSalad software, please refer to the Terms of Service at gamesalad.com/terms.

Q: I’m looking for art and sound assets for my project. Do you have any advice?

A: There are many different places (both physical locations and internet sites) to obtain assets for your project. The important thing is to have confidence that you have the legal right to include the assets into your project, such as permission from the copyright holder. Some of our users choose to create their own assets, while others commission work specifically for their project or purchase assets directly off the [GameSalad Marketplace](#).

Q: How can I recover a project file that I’ve previously published?

A: Locate the previously published project on our website at <http://publish.gamesalad.com/games> (requires a login), click the edit button, and use the ‘Download’ button while the ‘General Info’ tab is selected. Please note that this is only applicable to Windows Creator publishes at this time!

Troubleshooting

Issue: A Rule is successfully triggering once, but then never again.

Possible Resolution: For this situation, it helps to think as Rules as mousetraps. Once triggered, they need to be “told” by the Creator to ready themselves again. This happens as soon as the Rule’s conditions are no longer valid. For example, if you have a Rule that triggers when an integer attribute is 1, 3, or 5, the Rule will note that its conditions are no longer valid when that attribute is 2 or 4, and so it will ready itself to trigger again. However, should the attribute jump from 1 straight to 3 (skipping 2), as far as the Rule knows, it has already successfully triggered as intended, and will take no further action until readied again. We recommend adjusting your Rule’s conditions so that this situation cannot be encountered, or creating logic such that once the Rule triggers, it automatically readies itself.

Issue: I’m encountering an error when attempting to publish my project.

Possible Resolution: This can be caused by any one of several issues – from improper configurations on the publishing computer, to firewall/antivirus software interference, to publishing server congestion. We recommend that you contacting GameSalad Customer Support and provide as much detail as possible, including: The version of GameSalad Creator you are publishing with, the specific text of the error, and what platform you’re publishing for.

Issue: What are some ways that I might improve the performance of my project?

Possible Resolution: Performance is impacted by the hardware configuration of the device used, any other software simultaneously running alongside the project, the size of the project, and the logic configuration of the project. We recognize how important it is to our users to be able to create games that run as smoothly as possible, and are consistently working behind the scenes on further optimization improvements. In addition, we recommend that users streamline their logic whenever possible. This might mean:

- Reducing the number of actors active at one time

- Ensure you're not spawning new actors into the scene faster than you destroy existing actors
- Using fewer Timers and Constrain Attribute Behaviors
- Avoiding use of "Every X seconds" Timers that are set to fractions of a second
- Checking to see if there is any logic that might be unintentionally contradicting other logic

Further Assistance

Want to dig in and read up further? We're constantly adding additional resources for our users on the official GameSalad website, in the form of updates to the Cookbook (cookbook.gamesalad.com), as well as posts in the Salad Solutions section of the forums (forums.gamesalad.com/categories/Salad-Solutions)

Should you find yourself in need of technical assistance, please feel free to reach out to our helpful Customer Service Team, using the contact form at arcade.gamesalad.com/feedback.

Definitions

Accelerometer – Determines the angle and rate of movement of the device; useful for determining when a user rotates or tilts their device.

Accelerate (Behavior) – Specifies the speed and direction of acceleration for an actor. Actors will continue to accelerate unless drag is applied or another movement Behavior takes precedence.

Accelerate Toward (Behavior) – Allows user to specify the precise location an actor will accelerate toward.

Action – These are Behaviors that are not meant to fire continuously, such as movement, changing image, color, or size, or accepting keyboard input; they are best used when governed by a Rule.

Actor – All the items (both visible and invisible) in your game are actors; they are governed by Behaviors that control how they interact with both other actors as well as with people playing your game.

Actor Mode – This is the default mode for the Stage, and allows the actors to be placed, moved, rotated, and resized in the scene.

Actor Tag – Actor Tags are used to categorize actors; they can be added and removed through the Project Editor.

Alpha Color – Alpha controls the transparency of an actor, and can be set to any real number between 0 and 1, with 0 being completely transparent and 1 being completely opaque.

Animate (Behavior) – Displays a series of images in rapid succession to create animation.

Attribute – Attributes contain numeric or text values that govern various aspects of the game, scene, and actors.

abs (Function) – This provides the "absolute value" of a number. In other words, it will make a negative number into a positive number. For example, $\text{abs}(-5.23)=5.23$.

acos (Function) – This is the trigonometric arccosine (inverse cosine) function. Values for x should range from -1 to 1, with results from 180 to 0, respectively. Any other input values will result in 'nan'.

asin (Function) – This is the trigonometric arcsine (inverse sine) function. Values for x should range from -1 to 1 with results from -90 to 90, respectively. Any other input values will result in ‘nan’.

atan (Function) – This is the trigonometric arctangent (inverse tangent) function. Results will range from -90 to 90.
Behavior Library – A list of all available Behaviors which can be assigned to actors.

Behaviors – Behaviors are actions or states of being that apply to actors; they can change how actors move, look, sound, and act.

Camera Mode – An alternative mode while editing a scene, Camera mode allows the user to set the sensitivity of the camera’s movement.

Change Attribute (Behavior) – Allows user to set or change a game, scene, or actor attribute. For instance, users can create score-keeping systems, instructing the game to add points (or remove points) to a specific actor, or remove health/life from a player or actor.

Change Image (Behavior) – Change an actor’s image to a new image - useful for showing damage or other changes to an actor without having to create multiple actors.

Change Scene (Behavior) – Goes to a specific scene - useful for moving to the next level, a credits scene, or the game menu.

Change Size (Behavior) – Grows or shrinks an actor (use a negative number to shrink). Insert a timer container to control how long the growth or shrink should take.

Change Velocity (Behavior) – Specify movement changes relative to another actor or to the scene.

Collide (Behavior) – Use this Behavior in conjunction with a tag to control which actor or groups of actors the primary actor will bounce against.

Collision Shape – This option determines whether other objects will collide with this object as if it were round (or rounded), or square (or rectangular).

Constrain Attribute (Behavior) – Continuously updates an attribute - for instance, constraining the actor’s location to that of the mouse. Essentially ties two attributes together.

Control Camera (Behavior) – Allows users to cause the camera to follow an actor - keeping it in view.

ceil (Function) – The integer when you round up a value. For instance, $\text{ceil}(0.3095)=1$, $\text{ceil}(9.2850)=10$, and $\text{ceil}(3.5)= -3$.

cos (Function) – This is the trigonometric cosine function. $\text{cos}(0)=1$. For more information on sine and cosine, check out http://en.wikipedia.org/wiki/Trigonometric_functions.

Density – Density refers to the heaviness of the actor. A higher value will make the object harder to move by less dense actors. A value of 0 will make an object immovable but still affect other actors in the scene. Density can be set to any real positive number. (i.e. $\text{Density} \geq 0$)

Destroy (Behavior) – Removes the ACTOR from the scene - apply this Rule to objects that can be destroyed, like the blocks in Breakout or the bricks in Mario. Best used with a Rule - for instance, contact with the ball in Breakout.

Devices – The Devices pane (under the ‘Attributes’ tab in the Browser) allows you to reference various attributes relating to the device, including mouse, touch, accelerometer, screen, and audio.

Display Text (Behavior) – Allows users to change the color, size, font, and other elements of text displayed in-game. Change “alpha” to 0 (found under color in the attributes list) to make all parts of the actor invisible other than the text.

exp (Function) – The exponential function e^x , where e is approximately 2.71828182818. For more information, check out http://en.wikipedia.org/wiki/Exponential_function.

Fixed Rotation – Selecting this option will prevent the object from rotating when it collides with other actors in the scene. Leaving this box unchecked means that the actor will rotate normally when it collides with other actors.

Friction – Increasing this number will slow this object down more when it interacts with other objects. Set to 0 for no friction (and hence no slowing).

Function – Functions are various mathematical formulas available through the expression editor that allow you to have the Creator calculate sines, cosines, logs, and much more.

floor (Function) – The integer when you round down a value. For instance, $\text{floor}(1.5)= 1$, $\text{floor}(9.2850)= 9$, and $\text{floor}(-3.5)= -4$.

Gravity – Each scene can be set up with an X and Y gravity attribute. These are real numbers that affect all actors which are Movable.

Group (Behavior) – Creates a group container that holds a set of Rules or Behaviors.

GameSalad Creator – The best software in the world for making games with no coding!

Image Library – A repository of .png assets that have been imported into the GameSalad Creator.

Instance (actor) – An instance is a unique example of an actor, with altered Behaviors, attributes, or abilities from the prototype actor.

Interpolate (Behavior) – Allows you to cause an attribute (location, value) to go from A to B in a set amount of time - for instance, from 100 to 1 (a countdown) or from position X to position Y (for pre-programmed movement)

iOS – The operating system created by Apple, Inc. to run on the iPhone, iPad, and iPod Touch.

In (Function) – The natural logarithm of a value. The natural logarithm is the logarithm to the base e , where e is approximately 2.71828182818. For more information, check out http://en.wikipedia.org/wiki/Natural_logarithm.

Keyboard Input – Saves keyboard input text into a specified attribute.

Kiip – Kiip is a system to provide real-world rewards when players complete an achievement in your game. “Pro” account holders will have the option to integrate Kiip Rewards into their games, and will receive a revenue share when players accept a real-world reward offer.

Layers – Layers allow you to prevent actors from interacting with each other - this can allow you to create backgrounds, scores, and more without having the player’s actors bump into them.

Load Attribute (Behavior) – Loads the value stored by a custom key name from persistent storage. Allows users to change an attribute upon specific input or events.

Logic – A term which refers to the combination of Rules and Behaviors that jointly define how a project operates.

Log Debugging Statement (Behavior) – Logs a statement in the debugging window. Statements can be attribute values for error-checking or text entry to flag an event when your game is running for testing purposes.

log10 (Function) – The base 10 logarithm of a value. For instance, $\log_{10}(10)=1$, $\log_{10}(100)=2$. For more information, check out <http://en.wikipedia.org/wiki/Logarithm>.

Max Speed – This attribute controls the maximum speed an actor can reach through acceleration and gravity. Please note that some Behaviors, such as Change Velocity and Interpolate, will override Max Speed.

Movable – Allows you to specify that an actor is able to move (or not able to move) when interacting with other actors. (Check the box to allow it to move)

Move (Behavior) – Specifies movement in a particular direction relative to the actor or scene. Movement is perpetual unless stopped by some other Rule or object.

Move To (Behavior) – Specify movement towards a particular X&Y coordinate; upon arrival stops.

magnitude (Function) – Finds the length of a line segment drawn directly from the origin 0,0 to the given point. You can also include an offset to find the length between two points - for example, $\text{magnitude}(x-x',yy')$. Say you have one actor at 100,240 and another at 25,30. To find the distance between them, use $\text{magnitude}(25-100,30-240)$.

max (Function) – Returns the higher value of the two numbers or variable units. This can be very useful for determining if a new score is higher than an existing score, or for other similar comparisons. For example, $\text{max}(12,35)=35$.

min (Function) – Returns the smaller value of the two numbers or variable units. For example, $\text{min}(12,35)=12$.

Note (Behavior) – Allows users to write a note to self or other creators about a particular Behavior, Rule, actor, or group - useful or explaining why something is done a particular way.

Open URL (Pro Behaviors) – Only available to individuals with a Professional-level membership, this option allows the developer to specify a specific URL to open when a certain action takes place (we recommend a button being clicked or pressed). The URL will open in the user's default browser.

Orientation – This determines whether your game runs in “up and down” mode (portrait) or “side to side” mode (landscape) on iDevices (iPhone, iTouch, iPad).

Otherwise – An optional component of a Rule; Behaviors placed under this heading will trigger whenever the conditions of the Rule are not valid.

Particles – Particles are small objects that move out from the actor in a defined way. See “Particles (Behavior)” for more information.

Particles (Behavior) – Create an explosion! Radial or fountain of particles - can also set an image, color, lifetime, and other parameters.

Pause Music (Behavior) – Pauses a currently playing music file upon a particular event.

Platform – The devices or locations where your game will be able to run. Platforms that are currently available include iOS, Mac Desktop, Android, and HTML5.

Play Music (Behavior) – Triggers a music file to play - can play once or loop.

Play Sound (Behavior) – Triggers a sound file to play - can play once or loop.

Preview – Allows you to see how your game will look and run instantly! Can be done both in GameSalad and in a browser window. Use often to ensure that the Rules and actions you are giving your actors are working properly.

Project Size – The current memory needs of your project can be found in the bottom-right corner of the Game-Salad Creator. The recommended maximum size for GameSalad Arcade games is 20 MB.

Prototype (Actor) – A prototype actor is an actor that possesses all the overall governing Behaviors, but does not possess some of the specifics that an Instance Actor possess (such as spawning location).

Publish – Publishing uploads your game to GameSalad's servers, where it is turned into a binary that you can then submit directly to Apple, GameSalad Arcade, or an Android store.

Pause Game (Behavior) – This Behavior will pause the scene by freezing all activity in a given scene and opening another scene that you specify. Use this to open up in-game menus or simply pause the action. Use “unpause” to resume the scene.

Playhaven – PlayHaven is a creative new ad network that provides an interstitial app referral immediately before gameplay. “Pro” accounts will have the option to include the interstitial or not. “Pro” account holders will also receive a revenue share for completed referrals in their games. PlayHaven is also the first partner in GameSalad’s game referral system, included on all games published under a Basic account.

padInt (Function) – Displays an integer with the specified number of digits. For instance, padInt(32,5) will display 00032. However, padInt(38025,2) will display 38025. It will always display at least the minimum number of digits needed to retain the value of x.

padReal (Function) – Displays a floating point with padding and precision. For instance, padReal(9.1234,15,6) will display 9.1234 with at least 15 total digits (including the decimal) and at most 6 digits to the right of the decimal. - 00000009.123400.

pow (Function) – Returns the value of x to the power of y. For example, pow(2,3)=2*2*2=8.

precision (Function) – Displays a floating point number with the specified number of decimal places. For instance, prec(1234.234,2) will display 1234.23.

Replicate (Behavior) – Creates copies of an actor based on an attribute or integer. Useful for displaying the number of lives a player has left, having an item duplicate itself, and much more.

Reset Game (Behavior) – Resets the game and all scenes (best used with a Rule to specify WHEN or HOW this happens)

Reset Scene (Behavior) – Resets the current scene and all actors in it(best used with a Rule to specify WHEN or HOW this happens)

Resolution Independence – Allows your game to be displayed in low or high resolution, depending upon the capabilities of the device.

Restitution – Makes your actor bouncy - 0 is no bounciness; 2 is superball bouncy. Any real positive number from 0 to 2 can be used.

Rotate (Behavior) – Alone, this causes consistent clockwise or counter-clockwise rotation. Can also cause an item to only rotate when an event happens, such as pressing a keyboard key or collision with another actor.

Rotate to Angle (Behavior) – Causes rotation to an angle relative to another actor or to a scene.

Rotate to Position (Behavior) – Causes rotation to a specific XY coordinate on the screen or relative to actor.

Rule – Rules create a condition or set of conditions that, when met, cause actors to act in specific ways.

Rule (Behavior) – Creates a condition or set of conditions to check for player input or an attribute change.

random (Function) – Returns a random integer equal to or between the first integer and the second. For instance, random(1,5) could return any of the following values: 1,2,3,4,5.

Save Attribute (Behavior) – Save a particular value into persistent storage with a custom name. Allows users to save games (see “Load Attribute” for information on loading saved information).

Scene Attributes – These are attributes that specifically affect a scene, rather than an actor or the entire game. They include Name, Time, Size, Wrap X, Wrap Y, Gravity, Color, Camera, and Autorotate.

Show Ad (Pro Behaviors) – Shows an ad at designated location if one is available (requires additional configuration)

Spawn Actor (Behavior) – Creates a new actor instance in the scene - useful for projectiles, dropping items, etc.

Stop Music (Behavior) – Causes a music file to stop playing.

sin (Function) – The trigonometric sine function. This is similar to the cosine function, but is offset by one quarter of a wave cycle. For more information, see http://en.wikipedia.org/wiki/Trigonometric_functions. Tip: if you use the sine function and start your incrementing variable at 0, your actor's movement does not start at the middle point between the minimum and maximum points of the wave.

sqrt (Function) – Provides the square root of a value. Input values less than 0 will result in 'nan'.

Timer (Behavior) – Timer allows you to perform Behaviors or Rules at specified intervals. These intervals are defined as after a certain number of seconds, every couple of seconds, or for a certain number of seconds.

tableCellValue (Function) – Returns the value of a cell of a selected table at a certain row and column. Tables are numbered starting at 1. You can also use the row or column name as an input for this function. For example, `tableCellValue(game.Data,1,15)` will return the value in table, Data, at row 1 and column 15.

tableColCount (Function) – Returns the number of columns in the selected table.

tableRowCount (Function) – Returns the number of rows in the selected table.

tan (Function) – The trigonometric tangent function. For more information, check out http://en.wikipedia.org/wiki/Trigonometric_functions.

Unpause Game (Behavior) – This Behavior removes the pause screen and resumes the paused scene.

vectorToAngle (Function) – Finds the angle relative to the origin 0,0, given an X and Y coordinate. For instance, `vectorToAngle(100,200)=63.435`. You can also find the angle relative to an offset - for instance, `vectorToAngle(x-x',y-y')`. `VectorToAngle(100-200,150-250)` will find the angle between the points 100,150 and 200,250, or -135 degrees.