The ^Unofficial GameSalad® Textbook

By Michael Garofalo

The Unofficial GameSalad® Textbook

A guide to creating games and apps with GameSalad

Version 2.00 • December, 2012

Created by Photics, an Internet publishing business.

http://photics.com

Foreword

What do you want to be when you grow up? That's a tough question for a kid to answer, isn't it? Even as an adult, I'm still not sure. There are so many interesting things in life. Yet, one thing has stayed with me. From when I could barely reach the controls of an arcade, I knew that I enjoyed playing video games.

When I was about five or six, my mom took me to a music store. The walls were lined with various instruments. Would I learn to play the guitar or bang on the drums? Nope, I was mesmerized by the Pac-Man® arcade game. A rock star I would not be. Instead, my life followed a more technical path.

From video games, I learned about computers. From computers I learned about desktop publishing and the Internet. From the Internet, I learned about web development. All of that has lead me to game development. It's almost full circle. As the author of this book, I'm here to inspire you.

But unlike so many others who frown upon video games, who consider it a colossal waste of time, I am a connoisseur of this digital art form. Yes, I consider video games to be art. A tremendous amount work and creativity is required make a game. That's the point of this book. It is a comprehensive guide to game development with GameSalad. From conception to distribution, every aspect of making a game is considered. It's not an easy journey. But for those who have an affinity for game development, it can be incredibly satisfying and even entertaining.

This is more than just some mundane manual for miscellaneous software. It is an appreciation of life. We are not robots, nor we should try to absorb data like they do. When you are motivated to learn new things, you are more likely to understand the lessons. Video games are quite alluring. That's why I encourage an interest in them. It can lead you to great things in life, like a rewarding career or at least an entertaining hobby.

Through GameSalad, you can learn about math, physics, logic, marketing, sociology, law, economics, creative writing, foreign languages and practical job skills. Yet, this book is far from a classroom environment. It's all real world. Even if you don't make the next big game, the lessons that you can learn here are applicable to other professions.

This book is your guide through the virtual lands of GameSalad development. Will you find adventure and prosperity along the path of a game developer? I don't know for sure. Life is filled with wonders, but it is also filled with uncertainty. This book is designed to help you steer clear of pitfalls. It's also designed to help you with the progression of your game development skills. Ultimately, it's up to you to succeed. But perhaps through hard work and determination, you can learn new skills, make profitable games and even have some fun too.

Table of Contents

Section I - GameSalad Introduction 6

Chapter #1 - What Is GameSalad? 8

Chapter #2 - Getting Started 12

Chapter #3 - Game Ideas 16

Section II - Working With GameSalad 22

Chapter #4 - The Basics 24

Behaviors *31*

Display Text *41*

Behaviors Continued *45*

Professional Behaviors *62*

Back to the Basics *65*

Chapter #5 - Math 75

Chapter #6 - Physics 95

Section III -Building Blocks 102

Chapter #7 - Controls 104

Chapter #8 - Common Game Elements 120

Chapter #9 - Enemies 131

Chapter #10 - Setting the Scene 138

Section IV - Making Content 142

Chapter #11 - Graphics 144

Chapter #12 - Audio 151

Section V - Game Examples 156

Chapter #13 - Platformer 158

Chapter #14 - Space Shooter 164

Chapter #15 - Bouncing Balls 168

Chapter #16 - Puzzle Games 172

Section VI - Building A Better Game 176

Chapter #17 - Testing 178

Chapter #18 - Optimization 183

Chapter #19 - Fun 186

Section VII - Publishing Your Game 188

Chapter #20 - App Publishing 190

Chapter #21 - Mac Publishing 200

Chapter #22 - Web Publishing 202

Section VIII - Advanced Techniques 204

Chapter #23 - Pseudo 3D 206

Chapter #24 - Particles 210

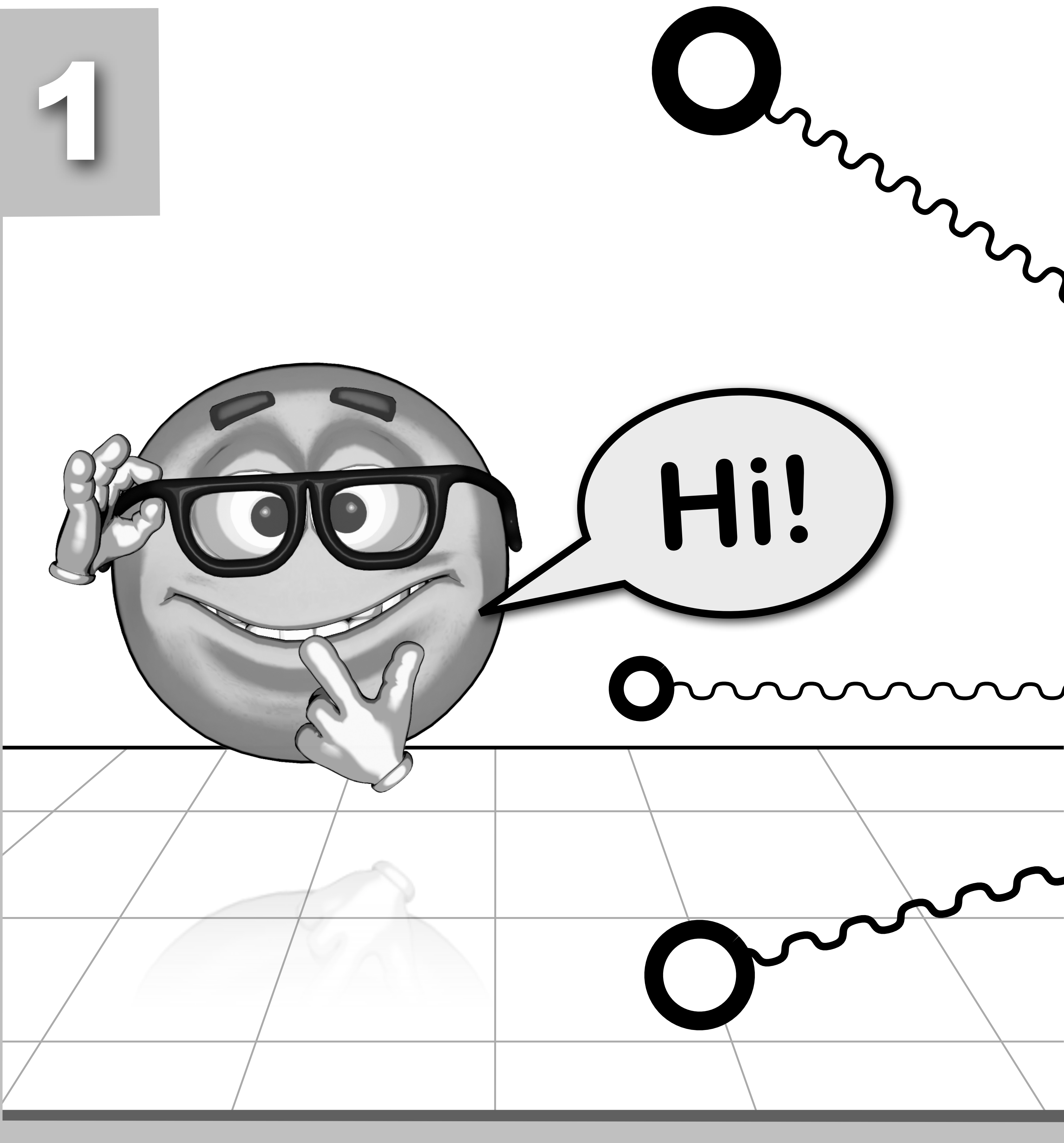
Chapter #25 - Limit Smashing 214

Section IX - Now What?! 218

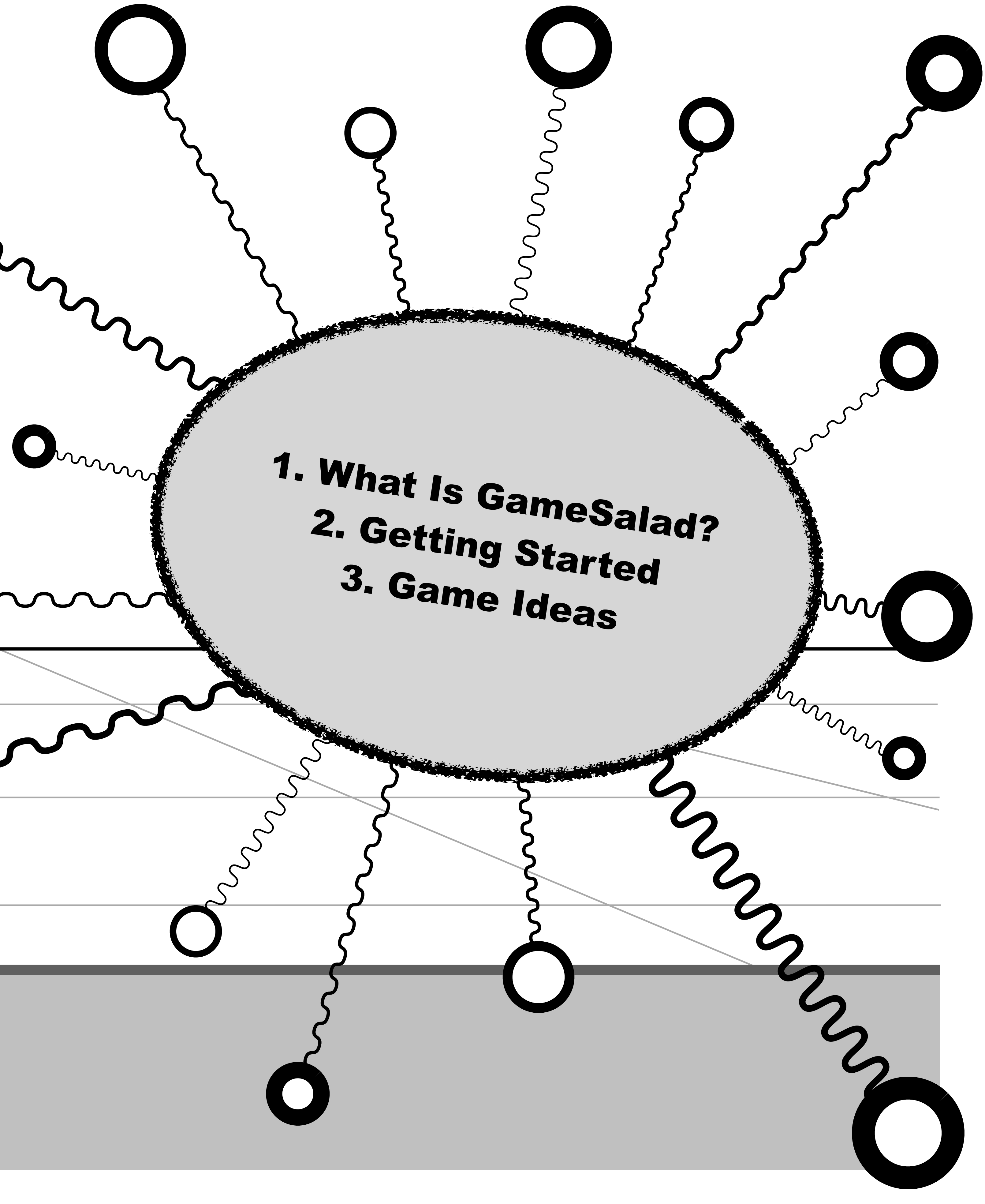
Chapter #26 - Promoting Your Games 220

Chapter #27 - The Future 224

1



Section I - GameSalad Introduction



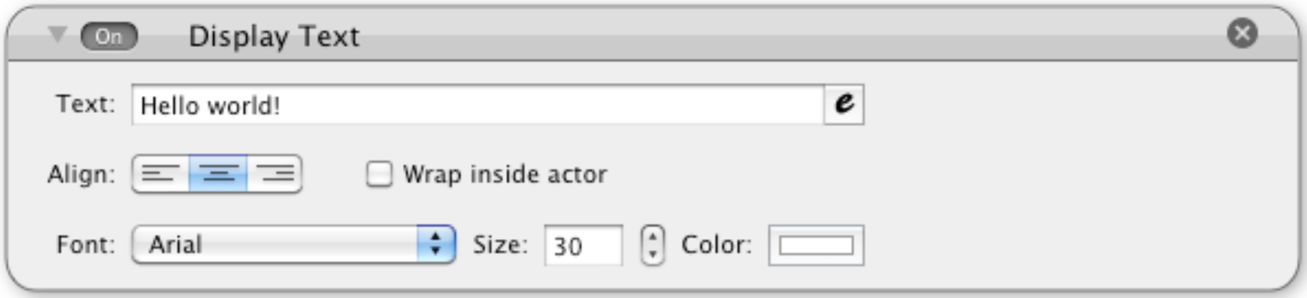
1. What Is GameSalad?
2. Getting Started
3. Game Ideas

Chapter #1 - What Is GameSalad?

Game creation for everyone™ ...that the GameSalad motto. It means that you don't have to use complicated programming languages in order to make video games. GameSalad is more visual. By using a drag-and-drop interface, game development becomes less tedious and more simplified. The creators of GameSalad set out with a mission to make game development more accessible and easier to understand. From the first few minutes of using GameSalad, you can see that the GameSalad team was successful.

If you require proof of this claim, simply try to complete this "Hello world" tutorial.

- Create a new project by launching GameSalad and clicking "Blank Project" in the "New" section.
- Click the "Scene" button, double-click the "Initial Scene", and then click the plus icon to create a new actor.
- Double-click the actor, and then drag the "Display Text" behavior into the "Drag your behavior here" area.
- If you want to be fancy, change the actor's alpha color to zero. That will hide the white box. It's in the attributes listing, under "Color".
- Click the back arrow to return to the Scene Editor - Initial Scene, and then drag the actor onto the scene.



Text: Hello world!

Align: centre

Wrap inside actor

Font: Arial Size: 30 Color:

That's it. You're done. GameSalad even typed the text for you, as "Hello world" is the default text for the "Display Text" behavior. Simply press the play button to see the "Hello World" text. From here, this sample program could be exported to the Mac Desktop, the iPhone / iPod Touch, the iPad, the web and more. The true programming language is hidden from the user, and essentially English becomes the programming language.

From that simple tutorial, a strength of GameSalad is highlighted. Its power comes from the graphical interface. This unique approach to game development gives GameSalad amazing speed and controllability. The software is simply easy to use. Before I was using GameSalad, I was using Xcode to create my iPhone apps. It was incredibly tedious and difficult for me. I hated having to bounce around from the different files in Xcode... AppDelegate.h, ViewController.m, info.plist or MainWindow.xib. What is that nonsense? Why should one have to endure such gibberish in order to make a good video game? With GameSalad, you don't have to struggle with Xcode. That is what makes GameSalad an excellent tool for beginners. Without the barrier of an arcane programming language, you simply get to master what really matters — making great games.

"2D"
Top-Down
<4:3 - CLASSIC>
Crosshair - for multi-direction fire
Radar
- your location
- Capital Ship
- Asteroids
Communication Pop-up (Hank/Philip)
Lives Remaining
Capital Ship
Asteroid
Capital Ship Info
HULL (Health)
Sonic Blast / Main Gun recharge indicators
Main Gun
"For Great Justice"

The previous image is a rough sketch of from one of my games. I had this concept for a game in my head, but I lacked the ability to make my vision a reality. For years, I tried various game development tools to complete this project. I was unsuccessful. It was just too difficult. Having to stare at lines and lines of code just sucked the joy out of the project. Things were much different with GameSalad. In about three weeks of working with the software, my game was ready for the app store.

With GameSalad, I was able to turn my game idea into a reality. If you're reading this book, you probably have lots of ideas for games. That's what separates people from computers. We have ideas. Computers mindlessly follow instructions. That's why I think GameSalad is a great tool. It's more about your ideas and your game content, rather than programming code.

Does that make GameSalad the magic bullet for game development? No, as there are limitations. Primarily, GameSalad is for the creation of two-dimensional games. While this book contains some tricks to recreate a few 3D effects, trying to pushing GameSalad beyond 2D will likely lead to frustration. GameSalad is better suited for projects that are similar to 8-bit and 16-bit arcade games… but with much better graphics. That's fine with me. While I like first-person shooters, and even racing games, playing a 3D game on a bus would likely give me motion sickness. After looking at the top selling games in the iTunes store — mostly 2D games — it seems that plenty of other people agree with me.

Another limitation of GameSalad is a lack of customization. Since the main goal of the software is to hide programming languages from the user, you can't go beyond what's already included in GameSalad. Even if you're familiar with coding, the software is not designed to accept modifications. That restriction creates quite the paradox. While GameSalad simplifies the process for making iPhone games, other tasks that should be simple are incredibly difficult or even impossible. I often find myself having to change my game just to workaround the limitations of GameSalad.

By better understanding the strengths and weaknesses of GameSalad, you can more accurately create a game plan. If you think that you're going to be creating a fancy 3D world, with fancy 3D characters, GameSalad is simply not ready for that. However, with modification of the game plan, you can still create a game that's fun.

TANK

With that in mind, I'm going to introduce you to someone. This is TANK, the main character for my online role-playing game. He's a bit of a square right now, but he should become quite the hero. Throughout this book, TANK is going to grow. He's the personification of GameSalad game development. At this stage, he's just an idea. Since I'm not satisfied with the current lot of MMORPGs (Massively Multiplayer Online Role-Playing Games) on the market, I decided to create my own RPG… with TANK as a main character.

But of course, I didn't start with a budget of $20,000,000 or a staff of 100+ people. This game started with a budget of about $100 and a staff of one — me! With GameSalad, I didn't need more than that. I just needed to simplify my plan a bit. Instead of a 3D MMOPRG, this game is a top-down RPG, like Gauntlet© (1985), The Legend of Zelda™ and so many other games from that era. Top-down means that the player's perspective is from above the main character.

What games will you make? I think it's important to have a goal. I'm not going to sugar-coat it for you. There are days where game development is just frustrating and tedious. Without something that drives you — an idea that you truly believe in — it might be tough to get through the low points. Even with the ease of GameSalad, the software doesn't make the game for you. The idea is more important than the tool. GameSalad is just that — a tool, like Adobe® Photoshop® or a hammer. It is up to you to create something wonderful with it.

Although, you have something that I didn't have when I was starting out — this book! It is filled with many different game examples that can be created with GameSalad.

Chapter #1 Summary

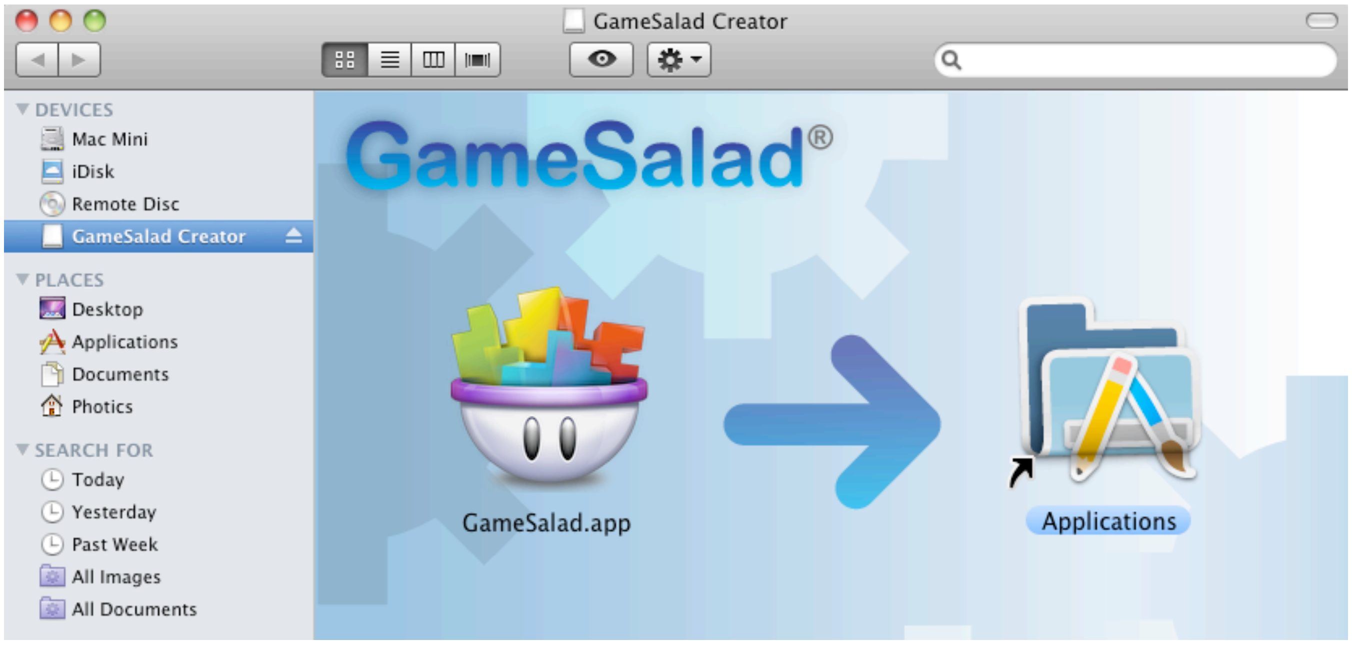
- GameSalad is a tool for creating games, primarily for the iPhone, iPad and iPod Touch. Web and Mac publishing is also possible.
- GameSalad uses a visual interface to hide programming code from the user. Essentially, English becomes the programming language.
- GameSalad is primarily for creating 2D games.
- While the user-friendly approach to game development will allow developers to quickly and easily create games, GameSalad doesn't accept outside code. This will likely frustrate advanced users.
- TANK is the mascot for this book. He will grow as more advanced topics are covered.

Chapter #2 - Getting Started

So OK, you've decided to take the plunge. You've carefully weighed the strengths and weaknesses of GameSalad and you've decided that the advantages far outweigh the disadvantages. You're ready to get to work. However, does your computer meet the software/ hardware requirements? Here are the recommended specifications…

Mac	Windows
Core 2 Duo CPU or better	Core 2 Duo CPU or better
Mac OS X 10.7 (Mac Lion) or higher	Windows Vista, 7 or 8
2 GB of RAM	2 GB of RAM

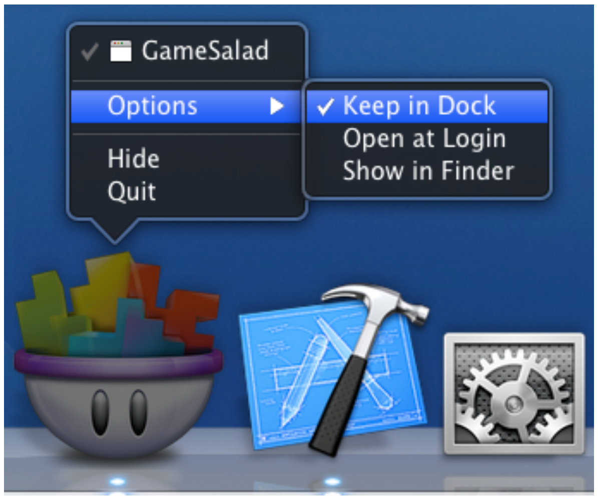
GameSalad originally started as a Mac only application. But with the addition of Windows support, the software has become more accessible. If GameSalad is your motivation for buying a Mac, you might want to consider the Mac Mini. It's the cheapest [new] Mac that you can get. That's what I use for GameSalad development. Even though it's low-end hardware, it runs GameSalad just fine. More RAM might not be a bad idea either. 2 GB is not a lot of memory. When I upgraded my Mac Mini to 4 GB, I noticed a significant boost in performance. If you're planning to make iPad games, a high-resolution monitor (1920x1280 or greater) can help too.



At the GameSalad.com website, you can download the GameSalad Creator. This is also where you can register for a GameSalad account. Once you've downloaded the GameSalad disk image, you can begin the installation process. Double-click the .dmg file and it will reveal some legal jargon. If you agree to the terms, a disk icon will appear. Double-click the icon to reveal a drag-and-drop window. The self-explanatory scene informs you of the next action. Simply drag the GameSalad icon onto the Application folder. This will install the software. That's all there is to it. Just open your application folder and launch the GameSalad app.

Note: Do you have enough hard drive space required for GameSalad? On a Mac, the application uses approximately 30 MB of space. That's fairly tiny for the awesome power that comes with GameSalad. Yet, your game projects can quickly eat up hard drive space. As comparison, my "Mobile Projects" folder is roughly 2 GB. I'm figuring that you'll need at least a few gigs of free space on your disk to install GameSalad and to have enough working space. This is not even counting the Xcode and/or Android SDK installs. Unless you're working on large and/or a lot of GameSalad projects, hard drive space probably won't be a problem. Yet, you might want to double-check your system before you install GameSalad.

Cool Tip



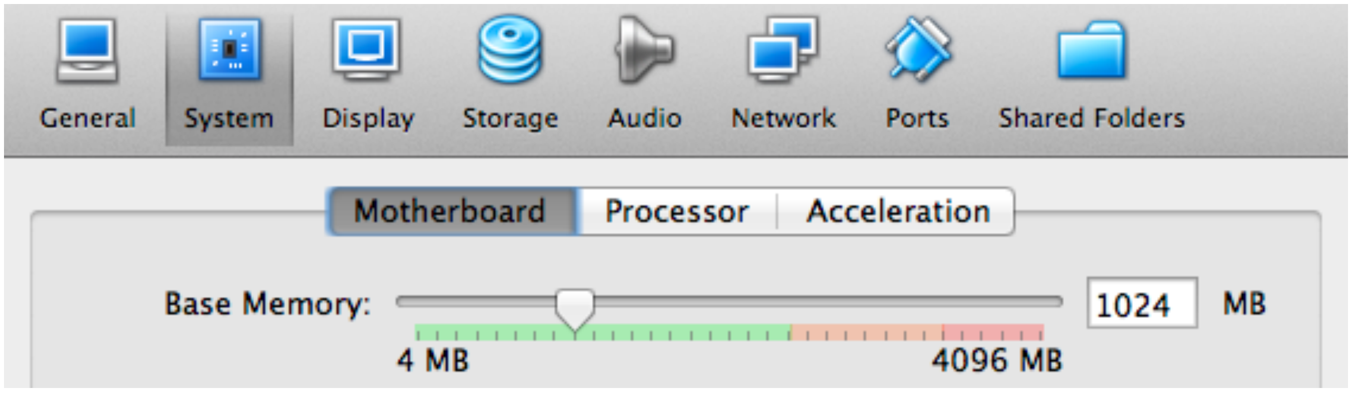
Are you having trouble finding GameSalad on your Mac or would you like to make it more accessible? There are quick ways to find and launch GameSalad. If you don't want to fumble around your application folder, type "GameSalad" in Spotlight. That's the magnifying glass at the top-right of the Mac OS desktop. Once GameSalad is launched, you can right-click (or command-click) the application in the dock, and then select "Keep in Dock" to make it stay there. While the method is slightly different for the Windows version of GameSalad, the same trick still applies. You can keep the GameSalad icon in your taskbar, allowing quick access to the application.

This book is primarily written from the Mac perspective. Yet, if you try both platforms, you'll likely find a lot of similarities between the Mac and Windows version of GameSalad. That raises an important question. Should you use a Mac or Windows computer? It's ultimately a matter of personal preference, your current equipment and your publishing targets for your apps. If you have a computer that already meets the hardware requirements, then that's an opportunity to save money. But even if you've chosen a side in the great Mac vs. PC debate, you might find it necessary to own both a Mac and a PC. The best testing is with native systems. While Google, Microsoft and Apple are competitors, you don't have to play favorites. If your primary goal is to make money, then try to see past the bias and look for opportunities.

What if you cannot afford to buy so much hardware, but you want to make apps for both Mac and Windows? That's where the Mac has a slight advantage. With Boot Camp, Windows can be installed on a Mac. Also, if you already have a powerful Mac, virtualization apps like VirtualBox can also be a solution.

- http://www.apple.com/support/bootcamp/
- https://www.virtualbox.org

Which is better — Boot Camp or Virtualization? Well, they both have issues. With Boot Camp, you can get the speed and performance of a real Windows computer. Yet, Boot Camp requires you to restart and boot from the Windows hard drive or disk partition. It can get tedious having to switch between operating systems. Virtualization is also no stranger to the mundane. While both the Mac and Windows can run simultaneously with virtualization, it can be a very slow experience. To speed up virtualization, having lots of RAM can help.



Base Memory: 1024 MB

The main problem with virtualization is that you have two (or more) mouths to feed. The above image is a screenshot of the VirtualBox System Settings. 1024 MB is only 1GB of RAM. If you walked into a retail store and saw a PC with such little RAM, would you want it?

These are things to consider while setting up your GameSalad workspace. The sad reality is that this is a risk. You could invest hundreds — or even thousands — of dollars in hardware, software and game assets, but not see a return on your investment. That doesn't even count the time invested — the most precious resource of all. So, is this something that you love? If you don't have the necessary equipment, then is this something that you can afford? These are tough questions for someone starting with GameSalad. While GameSalad makes it easier for you to create games, it also does the same thing for your competition. That can add a lot pressure for someone just starting with GameSalad.

Fortunately, GameSalad lets you start small. GameSalad is free to download.

- http://gamesalad.com/download

Then, if you're ready, you can upgrade to Pro. Since GameSalad Pro uses a subscription model, you might not want to waste hundreds of dollars on features that you're not even using yet. This gives you a chance to start working on your game before investing large sums of money.

The path to success can be difficult and uncertain, but that can also be what makes the journey exciting and rewarding.

Chapter #2 Summary

- Make sure that your Mac and/or PC meets the system requirements before installing GameSalad.
- GameSalad is free to download. While the free version is not as powerful as the Pro version, this gives you the ability to determine if GameSalad is the right software for you.
- To become a successful GameSalad developer, you might find yourself spending a lot of money on computer hardware and software.

Chapter #3 - Game Ideas

Before things start to get hectic, I suggest taking a moment out to plan out your game. What is it that you want to make with GameSalad? What are your goals? Are you here because you seek the iTunes bounty? The allure of loot, that is a common motivator for GameSalad development. It's how I got started. I wanted to make and sell iPhone games. Perhaps you have another goal. Maybe you are on a quest for knowledge. Maybe you want to make your website more fun. Maybe you don't even want to make games at all. GameSalad can be used for more than just games.

The point is this — have a goal and make a plan!

I've achieved some success on the iTunes App Store. Yet, that would not have happened if I didn't stay determined. There were many days where I felt like giving up. I often felt as if I was wasting my time. I could have made more money delivering pizzas, but that's not my dream job. Game and app development is what I want to do for a living, so I kept at it. The more I tried, the more my apps advanced.

Here's some general advice…

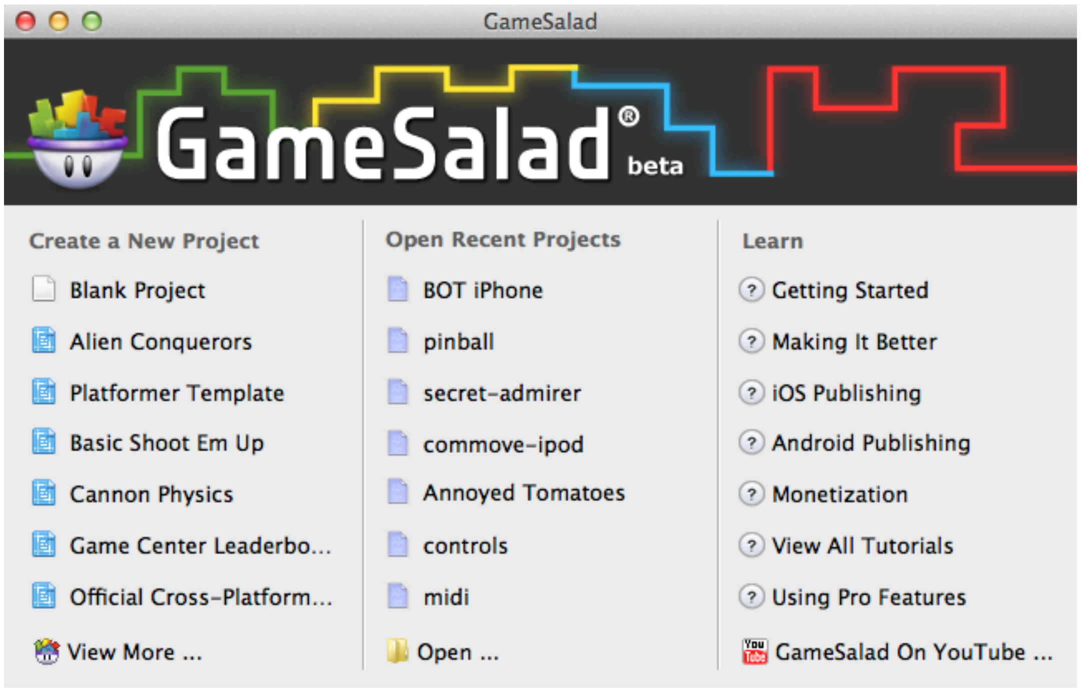
- **Take Breaks** - If you stare at your computer for too long, you'll get tired. Fatigue can lead to mistakes and frustration.
- **Learn** - When you make mistakes, or if something could have been done better, try to figure out what went wrong and make improvements.
- **Save** - Get into the habit of saving your work and making backups. Avoid the demoralizing hit of having to recreate work that you've already done. Once again, save and backup your work!
- **Know Your Strengths** - GameSalad can only do so much. You can only do so much. Have realistic expectations to avoid disappointment. Move in manageable steps towards your goals.
- **Adaptation** - This is essential for life, not just game development. Sometimes unexpected problems appear. Without the ability to recover from adversity, success will be harder to achieve.
- **Friends** - You don't have to be an army of one. If you know that you lack certain skills, team up with those who complement your abilities.

GameSalad has a "Jobs" forum where GameSalad developers can offer their services. This is a perfect place to form working relationships.

- **Have fun** - Don't forget that you're making games, not breaking bricks… unless you're making a game about breaking bricks.

As your game project grows, it becomes more difficult to make major changes. By knowing what you want to do beforehand, you can save yourself a lot of work. That's why it's a good idea to have a game plan. But even if you have no idea what to make, that's OK too. When you don't have a goal, then the goal is to find it. My first GameSalad game was decent, but my later projects were much better. GameSalad is a great way to practice game design. As you master the software you might start seeing possibilities that you never knew existed.

If inspiration hasn't struck you yet, you can look at the GameSalad samples. These beginner projects can give you an idea of what GameSalad can do. As you read about them, think about what you would want to do. What makes you happy? What games do you think others would want to play.



Alien Conquerors - With GameSalad, you can create one of your own shooters. It's a great introductory project, as it shows common GameSalad functions in action. It even throws in a little bit of advanced math on stage two. (The actor "Projectile2" uses Cosine for movement.) This template is essentially a clone of Space Invaders.

Platformer Template - If you've ever played Super Mario Bros.™, Mega Man™ or Sonic The Hedgehog™, you might be feeling the urge to create your own side-scrolling adventure. Games like this can be done with GameSalad, but it is an advanced project. In Chapter #13, platformers are described in greater detail. Trying a platformer as your first project is the equivalent of buying a brand new game and immediately setting the difficulty to ultra-hard. Almost every skill taught in this book can be applied to a platformer. I'm not against the idea. Just know what you're in for. The "Platformer Template" can serve as an introduction to physics, collision detection and motion. The trick to building a solid platformer is making all of the elements work in harmony.

In general, you might find a new appreciation for video games. Even creating the most basic of video games can be a challenge. As you climb the mountain of knowledge, the way you see the gaming world could change dramatically. Just don't let that swelled head stop you from being courteous to those around you. Many people might have no idea what you're talking about. "I updated the expression for the boss on level 9, making the artificial intelligence more responsive." Yeah, you might want to save talk like that for the GameSalad forums.

Basic Shoot Em Up - Like the Alien Conquerors template, the side-scrolling plane shooter is an easier project to work with than building a platformer. This shooter template teaches the basic principles of GameSalad game development. Shooting, spawning, character movement, collision detection and other GameSalad actions can be applied here. An important thing to remember is that the lessons in these templates are not mutually exclusive. You could make a platform shooter or a vertical and horizontal scrolling shooter.

Cannon Physics - This example shows off GameSalad's physics engine. "Canon Physics" is similar to Angry Birds. There is an important lesson that can be learned here. A relatively simple — but fun — game concept can yield tremendous success. As an independent developer, you probably don't have the money or the manpower to rival big game development companies. But with some imagination, you might be able to create a unique gaming experience.

Game Center Leaderboard Template - Game Center is Apple's social networking system for games. Throughout the Game Center app, you can see how your friends — and players throughout the world — are doing in games that support Game Center. If you want to add Game Center support to your own games, this template is a good place to start. In the "Professional Behaviors" section of Chapter #4, Game Center Leaderboards are covered in greater detail.

Official Cross-Platform Controller Template - Video games typically need controllers. This template is a great way to jumpstart your project, by learning how to create your own controllers - buttons, D-pads and thumbsticks. This topic is covered in Chapter #7 - Controls.

If you're ready to experiment with the templates, simply double-click the template, or click the template once and then press the "Open" button.

This is just the introduction stage. For the most part, you'll probably be using the "Blank Project" option. It can be difficult to start from a blank page. But as you master the concepts in this book, you might have far better ideas than the ones included with GameSalad. However, there are game genres that are far beyond the limits of the software. Here is a list of game types that would be really difficult — or even impossible — to recreate with GameSalad.

Mouse Hovering Games - If you're making a desktop only game, this isn't an issue. If you're making an iOS game, remember the limitations of the hardware. While the iPod can understand the equivalent of a mouse click, it doesn't recognize a finger hovering over the screen. This becomes an issue when creating cross-platform games. For more information on controls, see Chapter #8 - Common Game Elements.

Large Games - If your game has too many actors, or too many large images, performance will start to degrade. This becomes an issue when trying to support older iOS devices. Additionally, GameSalad games that are over 150 megabytes in size might not even be publishable. Because of limitations with HTML 5 apps, and over-the-air downloads to iPhones, you might want to keep your games under 20 MB. Chapter #18 contains information to optimize your game, but that can only go so far. When designing your game, hardware and software limitations should also be something to consider. This limitation is not necessarily a bad thing. A mobile device has a limited amount of space. If your game is too large, players with limited space on their phone might skip your app.

Word Games - While you might be able to craft a rudimentary Word Game, GameSalad isn't designed for something that uses and entire dictionary. With the addition of "Tables", there are some new options, but this is not an area where GameSalad excels.

Card Games - While you could probably do a card matching game, something like Poker or Solitaire would be insane to attempt with GameSalad. If you're in the mood to create a casino-like game, something like a slot machine is possible.

Games Requiring Outside The Box Features - GameSalad is designed for basic game development. It intentionally hides complex elements. That might not seem like a problem at first. You might hate to code. But once you want to do something beyond the scope of the included behaviors, you're out of luck. GameSalad projects are locked boxes. That means you can't do cool things like exporting to Xcode (for additional customization) or communicate with your own webserver/database.

It's difficult for me to expand the list of games that GameSalad cannot do. Most 2D games are within the realm of GameSalad. While the software does have limitations, some creative thinking can often lead to solutions. The other factor is that GameSalad can change. Something that's impossible today could be a non-issue tomorrow.

GameSalad Advantages	GameSalad Disadvantages
Easy to learn and use	Can't be customized
Fast game creation	Performance limits
Great community	No database support
Awesome at 2D games	Not a 3D game engine

Inspiration can arrive at any moment. When you really start getting into GameSalad game development, you might find yourself dreaming about it. Originality is a strength of independent game developers. I don't recommend trying to copy the ideas of others. Instead, start by increasing your game development abilities. Once you start to master the skills of the profession, your mind should awaken to new gaming possibilities.

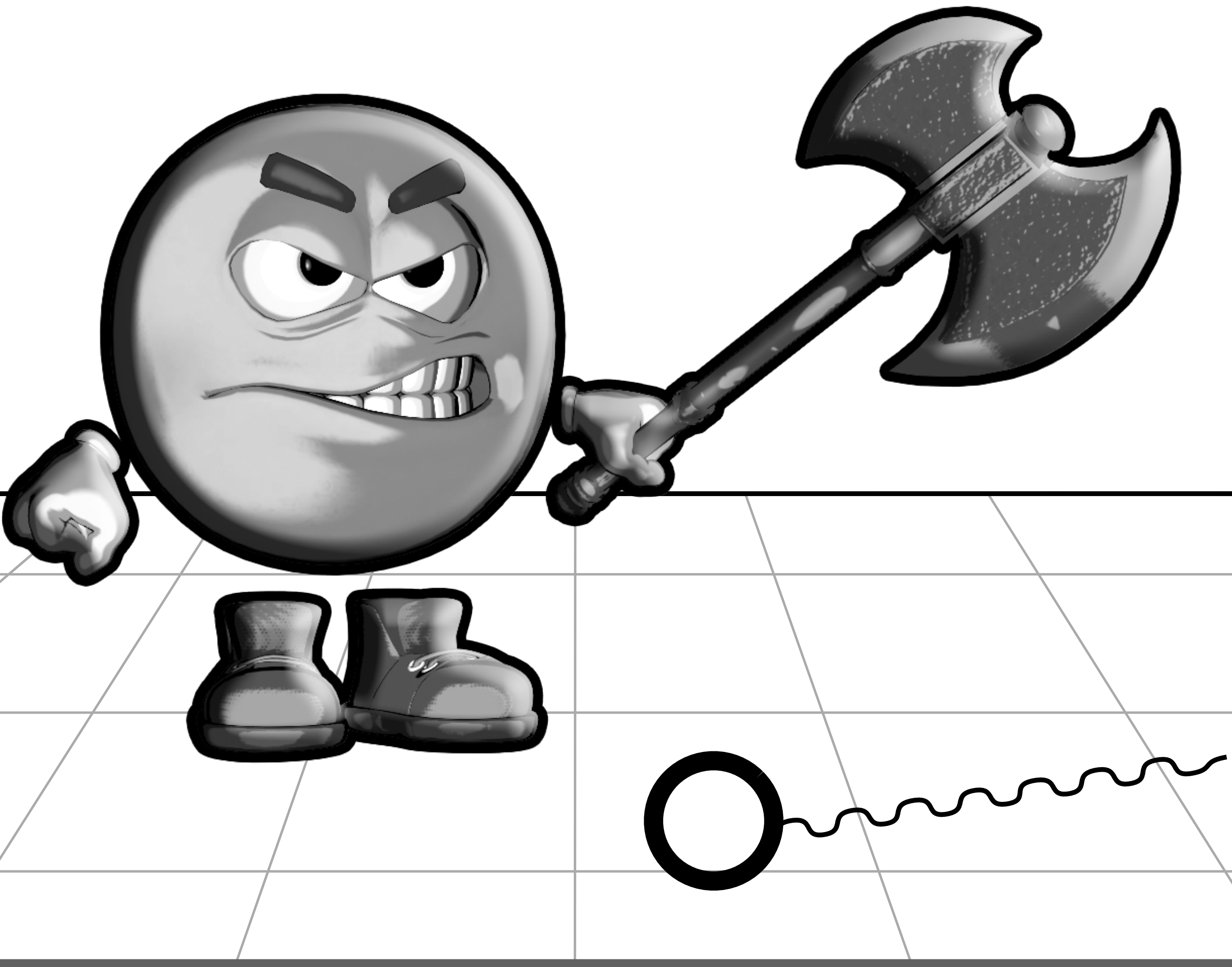
If you're in this for fame or fortune, remember that you'll have to meet the customer's needs. People like to be entertained. You might find something amusing, but will your audience feel the same? Knowing your target audience is also essential for developing game ideas. More on this topic can be found in Chapter #19 - Fun.

The point is to think beyond the technical. GameSalad has eliminated much of the mundane aspects of 2D game development. With the programming aspects of game development simplified, only your game's content will set you apart from other GameSalad developers. Furthermore, the iTunes App Store is already flooded with games. If you're just going to make another Asteroids clone, can you really expect success? There is opportunity in the iTunes App Store, but what is going to make you stand out? That's the importance of a good game concept. The projects mentioned in this book are only starting points. The objective is to lead you to the path of new ideas.

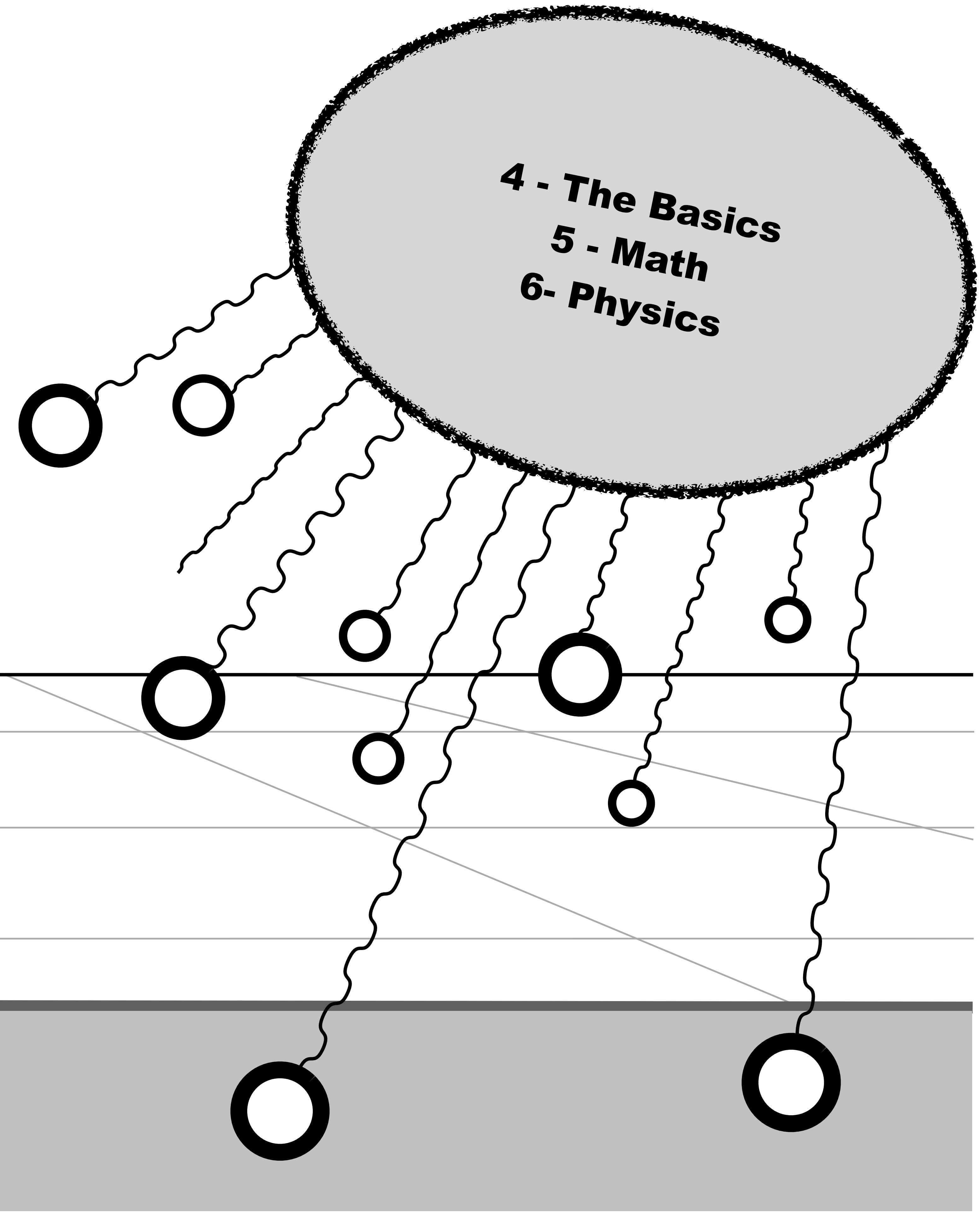
Chapter #3 Summary

- A good state of mind is important for creating games. While your games exist in the digital world, don't forget about the realities of your body. Remember to rest, learn and have fun.
- GameSalad includes some sample projects. They're designed to show you the basics of the software.
- The sample projects are only a fraction of what can be created with GameSalad. Open your mind to the possibilities.
- Know the limits. GameSalad can't make every game.
- Originality is the hallmark of independent development. Increase your chances of success by making your games unique.

2



Section II - Working With GameSalad



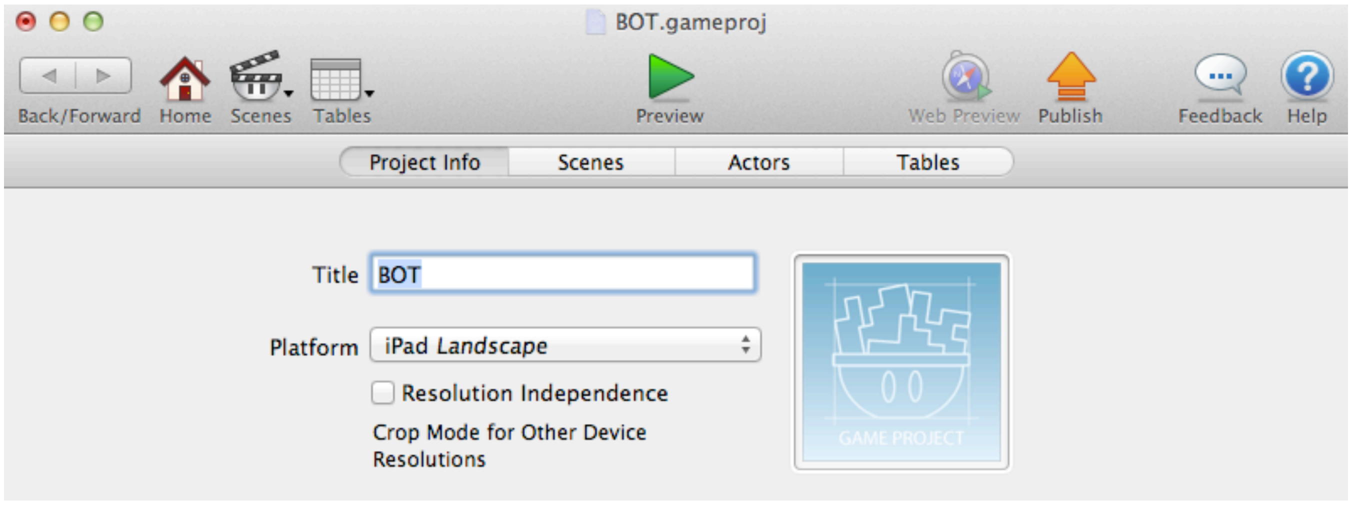
4 - The Basics

5 - Math

6- Physics

Chapter #4 - The Basics

With the formal introductions out of the way, now you're ready to get to work. This chapter is dedicated to an overview of the GameSalad application. After you launch GameSalad, and then start a blank project, you will be greeted with the "Project Info" screen. From this area you can enter descriptive information about your game. At the top of the window are buttons you can use to nagivate through your project.



Title BOT

Platform iPad Landscape

Resolution Independence

Crop Mode for Other Device Resolutions

Back/Forward - The first five buttons (at the top of the home screen) are related to navigation. The interface is similar to a web browser. By clicking the left and right icons, you can navigate through your project. If GameSalad cannot move in a direction, that arrow will be grayed out.

Home - If you want to return to the "Project Editor" area of GameSalad, click the "Home" icon. The "Home" screen provides macro-level management of your project.

Scenes - The "Scenes" icon provides for navigation to a specific scene in your project. Every GameSalad game must have at least one scene. But if you have more than one scene, this button can be quite useful. Press the "Scenes" icon once to list all your scenes. From this drop-down menu, pick the scene you want to switch to that scene. If you have more scenes in your project, than can be displayed by your monitor, the scene drop-down menu will add vertical scrolling arrows as necessary.

Tables - You can give your GameSalad projects the power of spreadsheets. This is a great way to store a lot of data, such as game dialogue, tile location, high scores or easier attribute management.

Preview - This button allows you to run your game. It activates the GameSalad "Player". When you are finished testing your game, you can click the "Back" arrow, or the use "Scene" button, to exit.

Web Preview - This launches the HTML 5 version of your GameSalad project. You can view the game within the GameSalad Creator. After activating the preview mode, the options to launch the project in a web browser (such as Firefox or Safari) becomes available.

Publish - When you feel that your game is ready, you can use this button to send your game to the GameSalad publishing server. A login is required. The options available here may depend on your GameSalad license. More on this topic is covered in Section VII - Publishing Your Game.

Feedback - Clicking this icon will send you to the GameSalad website. A login is required to use this feature. Once ready, you can send a bug report or make a feature request.

Help - By clicking the question mark icon, you can launch the GameSalad support website.

When starting a new project, you might notice a large gray box. This box is toward the right side of the "Project Info" window. (It's next to the "Title" and "Platform" fields.) By dragging a compatible image file onto this box, you can give your project an identifying look. It's also known as the primary image. If you're wondering why your "Recent Projects" have a generic images, it's probably because you haven't specified a primary image for your projects.

In addition to setting an icon, you can enter some textual information for your game. There are five main fields: Title, Platform, Description, Instructions and Tags. While starting out, only a title and platform choice is required. Unfortunately, the descriptive information does not carry over into iTunes. Although, I find it handy to keep all of these fields populated with useful information. When I am ready to publish my game to iTunes, I can simply cut-and-paste the text from these fields.

Platform - This part is tricky, as you'll have to determine the primary resolution for your game. That number depends on your target device. Are you making an iOS game, an Android game, a web game or a desktop game? Choose the option that most closely matches your project. You can change your mind later, but doing so can create a tremendous amount of work.

If you try to change the platform, a warning will be displayed. "Changing target resolution will change the visible area of existing scenes. Scenes smaller than the target display size will be resized.” That means if you’ve already created a few scenes, changing the platform size might ruin the display of those scenes. As a precaution, you might want to copy your game before changing the game size. That way, if you mess something up, then you can still go back to the copy.

“Enable Resolution Independence” is related to iPhone 4 publishing. If you want to create high resolution graphics for the iPhone 4, but you also want to have optimized graphics for older hardware, this option helps you do that. GameSalad will automatically take your higher resolution graphics and create lower resolution copies.

Note: - If you see a yellow triangle appear at the bottom-right of your GameSalad window, your project has likely exceeded the recommended file size. If you’re creating a game for HTML5 or iOS, GameSalad projects should be kept under 20MB. Otherwise, you might run into issues. What are those issues? Well, if your game is too large, you might not be able to upload it to the GameSalad server, the HTML5 version might not work at all and iTunes App Store sales could be lost.

Cool Tip

The poor man’s high-resolution monitor - Are you having trouble working on iPad (portrait) projects in GameSalad? Would you like more space to see your active behaviors? Setting your monitor to a higher resolution would resolve that issue, but what if you don’t have a monitor that supports 1920x1280 pixels? Here’s a trick that might help.

If you have a monitor that rotates, or if you don’t have a problem with turning your monitor on its side, you could give yourself more vertical space by rotating the screen 90°. In the Mac OS system preferences, select “Displays”. From the display option, you can select rotation. Selecting 90° or 270° will flip your resolution. (Some Macs will block this setting.) 1280x1024 becomes 1024x1280. That extra little space on the top and bottom, by sacrificing space on the sides, can actually make GameSalad more user friendly. The program tends to require more vertical space.

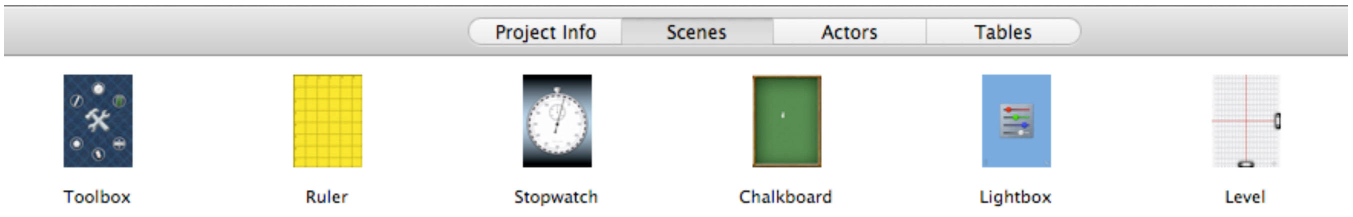
You can get even more space for GameSalad if you set the dock to auto-hide. In the Mac OS system preferences, select “Dock”. To auto hide the “Dock”, check the “Automatically hide and show the Dock” option.

If you’re testing iPad games in preview mode, hide the scene list to give you even more space. When combining all three space saving tricks, you should have enough space to see an unobstructed view of an iPad game in portrait mode.

Disclaimer - Remember, if your monitor isn't designed to rotate 90° or 270°, it may break. (This warning especially applies to laptop and iMac computers.) If the ventilation slots are hindered, overheating could occur. Also, I don't recommend that you bend your head 90° sideways. You can give yourself neck pain. This information is use at your own risk. This trick works best with monitors that are designed to rotate.

Also on this screen are tabs. Next to the "Project Info" button are three additional buttons

Scenes - When the project is launched, the "Project Info" tab is selected by default. Yet, the bulk of a typical GameSalad project involves the management of "Scenes" and "Actors". Initially, there's only one scene. More can be added with the plus icon. If you want to remove a scene, click a scene once and then press the minus icon at the bottom left of the Project Editor. (Backspace and the delete keys can also remove scenes.) By double-clicking a scene you can navigate to that scene.

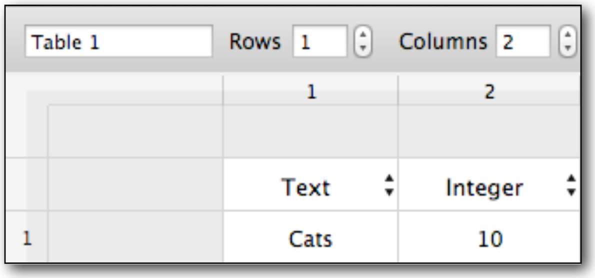


Actors - Much like in cinema or theatrics, actors perform in scenes. You can create and edit actors from the "Actor" tab. Click the "Actor" tab to activate it. Two columns should appear. The left most column is for managing tags. The larger right column is for editing actors. Double-click an actor to edit it. Much like with scenes, the plus and minus icons will add and delete actors. However, the "Actor" tab has two sets of plus and minus icons. The left pair is associated with the "Actor Tag" column. The right pair is associated with managing actors.

Unlike with scenes, actors can have "Tags". If you want to delete an actor from your project, while the "Actor" tab is active, the "All" tag must be selected before pressing the minus button. If the "All" tag is not selected, pressing the minus icon will removed the actor from the selected tag group

Tags - You can add "Tags" to an actor from the "Actor" tab of the "Project Editor". To create a new tag, click the plus icon at the bottom of the "Actor Tags" section. Actors can be clicked and dragged to be associated with a tag. First select the "All" tag, to see all of the actors in a project, and then click-and-drag the actor to the desired tag name. That should associate the actor with that tag group. To see all of the actors within a tag group, just click the tag name from the “Actor Tags” column. You could also create a new actor directly from a highlighted tag group. An actor can have more than one tag.

GameSalad tags have two main functions. When implementing rules or collision detection, tags allow an action to be associated with more than one actor. Tags can also be used to organize your actors.



	1	2
	Text	Integer
1	Cats	10

Tables - Initially, a GameSalad project will not contain any tables. To create one, click the “Tables” tab and then press the plus icon. Once a table is created, the data can be entered manually — similar to a spreadsheet program like Microsoft Excel. But if you already have a CSV (comma-separated) file, you can import it by pressing the “Import CSV” button. Rows and Columns can be added/removed with the corresponding up/down arrows, or by simply entering a new value. The top row is reserved for the “Attribute” type.

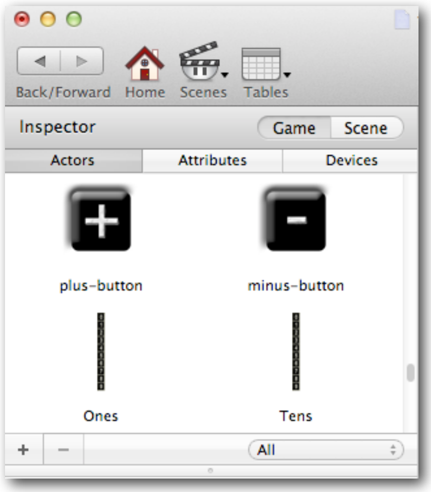
Cool Tip

Do you need more tables? That’s where the Windows version of GameSalad has the advantage. While the Mac version of GameSalad can only create 50 tables, the Windows version can create more tables.

With the fundamentals of the “Project Info”, “Scenes”, “Actor” and “Tables” tabs covered, the next topic is scene editing. Double-click a scene, from the “Scenes” tab, or use the “Scenes” button to activate the “Scene Editor”. This new view will contain the same tool bar, but three new interface elements will be activated - the “Inspector”, the “Media Panel” and the “Scene Preview”.

A lot of work can be done from the “Inspector” pane. You can use it to create actors, add actors to the scene, edit the attributes and parameters of the “Game” or the “Scene” and it’s right next to the “Media Panel”. Once you get used to the way GameSalad works, you might find yourself dragging-and-dropping content like a ninja.

So OK, how does it work? At the top there are two buttons. When the "Game" button is activated, you'll see three tabs - Actors, Attributes and Devices.



Actors - This tab is similar to the "Actor" tab on the "Project Editor" screen. The main difference is that you cannot edit tags. However, actors can be sorted by tags via the drop-down menu. (In the image to the left, the actors are being sorted by the "All" tag.)

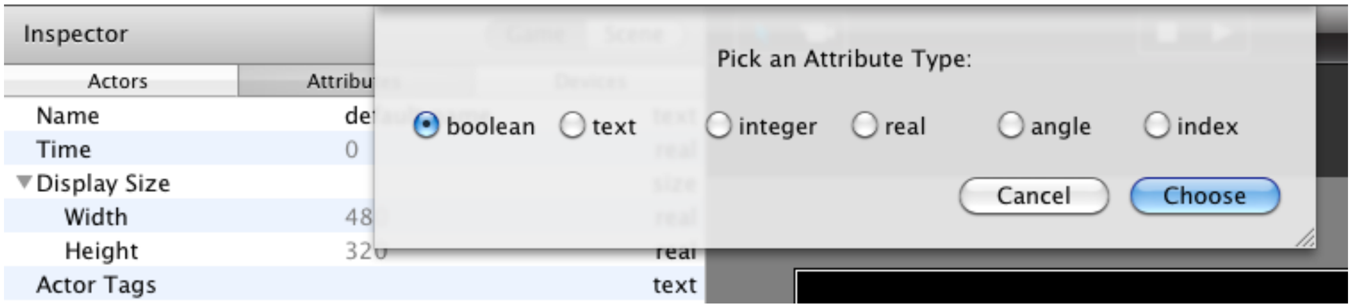
Devices - These are values specific to hardware, such as a mouse, a touch screen or an accelerometer. I generally do not alter device values from the "Device" tab. In other words, I typically ignore this tab. Usually, when I'm messing with the device settings, it's to control the values associated with audio volume. However, that can be accomplished with actors. This section is more for GameSalad, to monitor devices.

If you're paying attention, you might be saying to yourself, "Hey dude, you skipped one. Where's the explanation of the 'Attributes' tab?" That's right. I intentionally skipped the "Attributes" tab. I did this to call special attention to GameSalad attributes. The comprehension of this topic is critical to the understanding of GameSalad development.

Attributes are used by the GameSalad game engine to store data. That information can then be used by actors, the scene or the game itself. Some attributes are preexisting, like the ones in the "Device" tab, but you can also create your own attributes. You can create attributes for the three divisions of GameSalad - Game, Scene and Actor. Remembering the GameSalad hierarchy is important. An actor goes in a scene and a scene goes in a game. Attributes follow the hierarchy. Actor attributes can only be used by that actor. Scene attributes are used by actors in a specific scene or by the scene itself. Game attributes can be used by the whole game. All of that might seem confusing at first, so what does this mean for game development?

- If you want to keep track of a player's score, a game attribute would make sense.
- If you wanted to make the screen black for a single game level, and then white for another, a scene attributes could be used.
- If you wanted to give an actor hit-points, especially for an actor that is used more than once, an actor attribute is used.

You might find yourself making a lot of game attributes, so that's a good place to start. Back at the "Scene Preview" window, there's the "Inspector" pane. With the "Game" button selected, activate the "Attributes" tab. In this section, game attributes can be reviewed, modified and created. Click the plus icon to create a new attribute. A pop-up window should appear.



Pick an Attribute Type:

[✓ boolean] text integer real angle index

There are six choices for an Attribute Type.

Boolean - This is a true or false option. It is very helpful with game rules. For example, you could make a mute button. If you want the sound to play, set your custom attribute to "True". If you want the sound off, set the attribute to "False". The game rules could then act accordingly. When a "Boolean" attribute is listed in the "Attributes" tab of the "Inspector", it can be seen with a check box in the "Value" column. When that check box is checked, it means true. Unchecked means false.

Text - Most of the attributes are associated with numerical values. This attribute lets you enter pure text. There are some useful tricks that can be accomplished by changing text attributes. For example, if you want to add foreign language support, using text attributes can help with localization. Another example is when you want to combine text with numbers, when using the "Display Text" behavior. (Keep reading this chapter for details.)

Integer - This attribute is for whole numbers, both positive and negative. Examples: -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10. "Integer" will round a number. 1.5 becomes 2, -1.4 becomes 1 and 20.4 becomes 20.

Real - This attribute is for numbers with decimal places. These numbers can be positive and negative. Examples: 123.456, -1337, 0. This is the least restrictive attribute option for numbers.

Angle - This is an attribute for storing angles. It is a positive number. It's range is greater than or equal to zero, but less than 360. An angle can contain decimals. Numbers outside of this range will be converted. 360 is changed to 0 and -90 is changed to 270.

Index - This is similar to an “Integer” attribute, but only positive numbers are used. With an “Index” attribute, -10 becomes 0, 1.4 becomes 1 and 1.5 becomes 2.

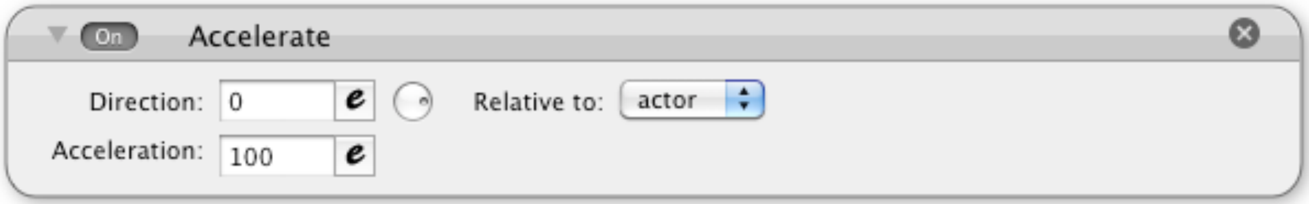
When creating attributes for numbers, why not just use “Real” attributes all the time? Often, that’s what I do. Even if you’re fanatical about optimization, only a tiny bit of memory is saved by using a Boolean instead of a Real attribute. Tiny is too large of a word. Next to nothing, that’s a more accurate description. If you’re only using a few attributes, the saving are trivial.

There’s also the issue of changing an attribute’s type. While renaming an attribute is a simple task, deleting an attribute will cause problems for all the actors associated with it. If I feel I’m going to need more options, I’ll pick “Real” rather than “Integer” or “Index”. Even a “Boolean” can be replaced with a “Real” attribute. Simply use 0 or 1 instead of false or true. (Oh, and if you want to rename your attribute, click on the attribute name once in order to select it, wait about a second, and then click it again.)

“Integer”, “Index” and “Angle” are useful if you want to limit the range of your numbers. The Integer type is great for rounding a value, should you not want to display decimal points. Although, some elements (such as color, sound or accelerometer values) have a range between zero and one. Without decimals, your GameSalad games can’t have nice things like volume sliders or accelerometer based controls. Also, GameSalad’s colors option — including alpha channels — use decimals. The point is to understand the main purpose for your attribute, and then to pick the best attribute type for it.

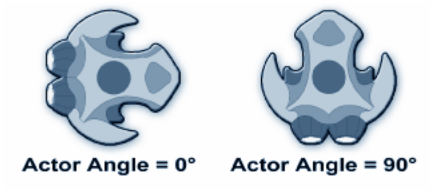
Behaviors

OK, now we’re at the good part. “Behaviors” are essential to GameSalad game development. There are two ways to add a behavior to an Actor. If you’re using the “Inspector” pane, you can drag a behavior from the “Media Panel” to the actor. I don’t recommend that approach for beginners. Instead, I suggest using the method from the “Hello world” tutorial. Double-click an actor and then drag a behavior to the open area on the right. The following is an explanation of the GameSalad behaviors.

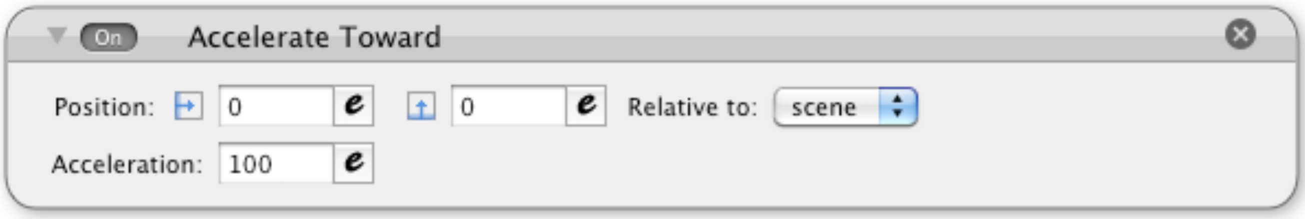


Direction: 0
Relative to: actor
Acceleration: 100

Accelerate - This is a way to move your actor. But much like a car trying to get on the highway, the speed increase is progressive. With the Accelerate option, the top-speed is not always immediate. There are three main settings for “Accelerate”. A higher “Acceleration” number increases the speed. The “Direction” field is for specifying an angle. You can click-and-move the circular icon to select an angle. You can specify the angle manually. Next is the “Relative to” drop-down menu. There are two selections. If you choose “scene”, the actor will move in a direction that’s relative to the screen. If you select “actor” the “Direction” angle will be added to the actor’s angle. The difference is tricky, but it usually has to do with game controls.



With some games, pressing left on a controller will always move the character to the left of the screen. The movement is constant. Some other games might use acceleration to imitate gravity. The direction of a game’s gravity is usually constant too. Making the acceleration relative to the scene, 270° or downward, works for gravity. However, there are scenarios where “scene” based acceleration doesn’t make sense. As an example, you might want to make a space shooting game. If you want the fighter to move forward — while also being able to turn — the movement needs to be relative to the direction of the fighter. Acceleration that is relative to the actor can be used to mimic a pair of thrusters. That can help create the illusion of a spacecraft propelling forward.

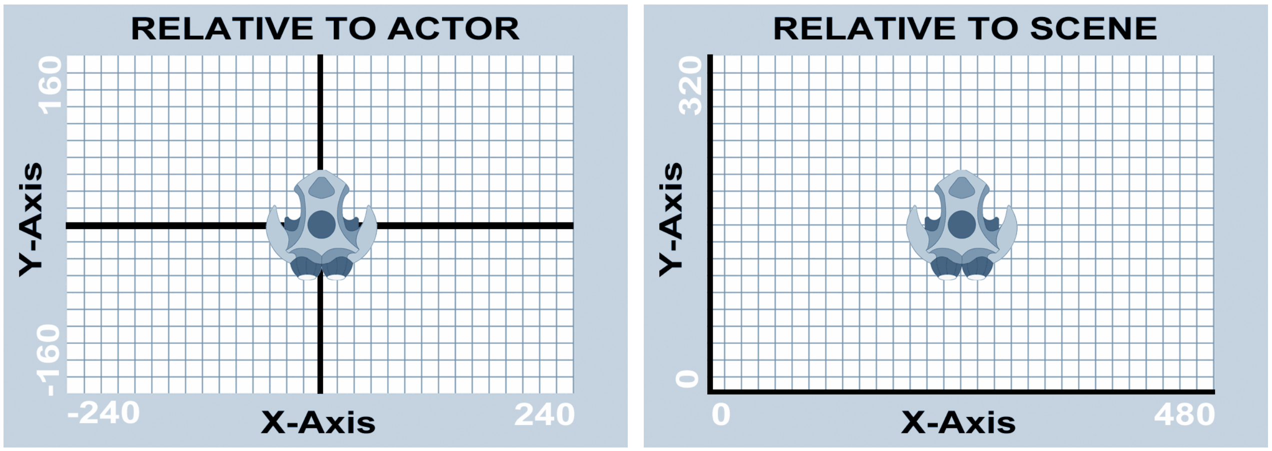


Accelerate Toward

Position: 0 0 Relative to: scene

Acceleration: 100

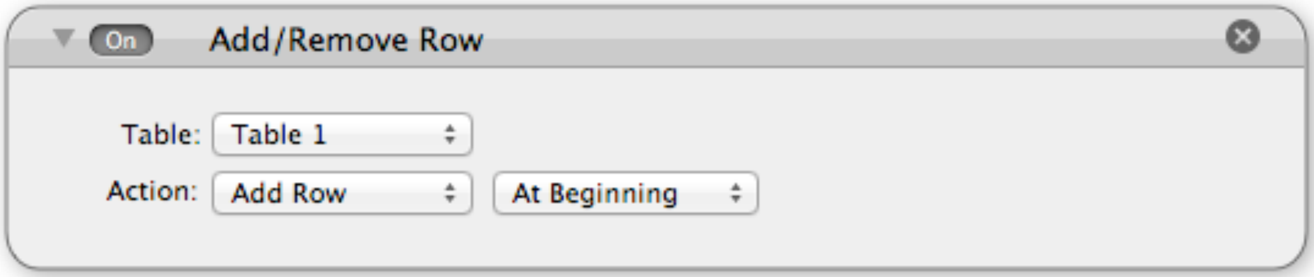
Accelerate Towards - Instead of specifying a direction, the “Accelerate Towards” option lets you specify a location. This functionality could be useful for creating artificial intelligence, simulating gravity or implementing click/touch based movement. The “Position” fields use pixels as measurement. If “scene” is selected for the “Relative to” setting, the origin is at the bottom-left of the scene. If the “actor” setting is selected, the origin is from the center of the actor. The diagrams below illustrates the difference.



Much like with the regular "Accelerate" option, the "Acceleration" field controls the speed. Higher numbers mean more speed.

You might have noticed a little "B" icon next to the acceleration related behaviors. That's because each is known as a "Persistent Behavior". They'll keep on going unless something stops them. That's a considerable issue with acceleration, as the speed could increase to the point of absurdity. There are two ways to slow down your actor. One, you could apply a maximum speed to your actor. This will prohibit your actor from accelerating beyond a specified speed limit. Another option is "Drag". This will slow down an actor. "Drag" mimics friction. By controlling these elements of speed, you can create a believable environment.

The acceleration behaviors are similar to the movement related behaviors. You might need to experiment with both sets, in order to determine which method is best for your actors. For more information on this topic, see Chapter 6 - Physics.



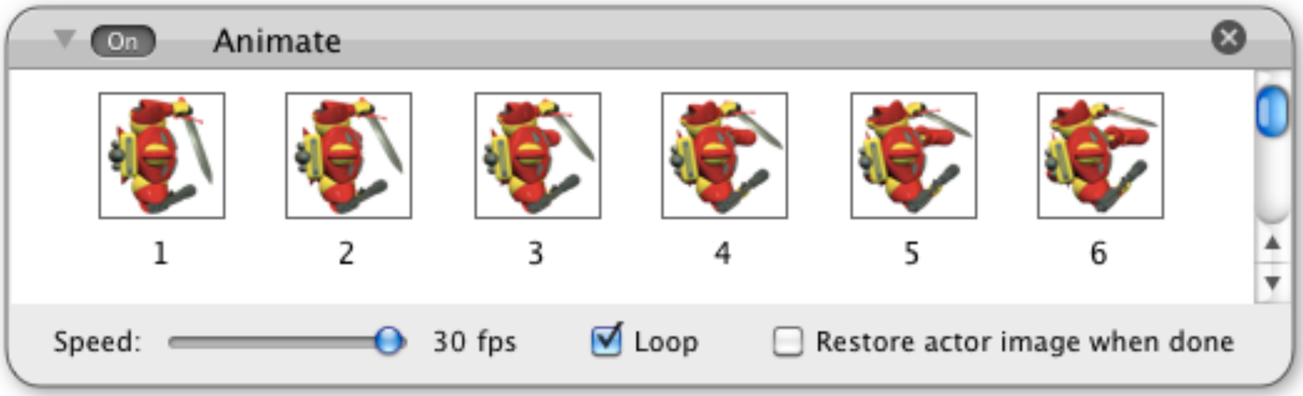
Add/Remove Row

Table: Table 1

Action: Add Row | At Beginning

Add/Remove Row - This is an "Action Behavior" that is related to "Tables". Much like an Excel spreadsheet, "Rows" can be inserted or removed. This is useful if you're creating your own high score lists. It could also be used for managing actor inventory. Perhaps your hero just picked up a new item. You could add a new row of data to keep track of the equipment. The "Table" option allows you to select which set of data that you want to work with. The "Action" option has three parts. The first gives you the option to "Add Row" or "Remove Row". The second option lets you select the location of the new row. You can choose "At Beginning", "At End" or "Index". If you choose "Index", then you can use an "Expression" to specify the row. Any data at or below the specified row will push downward. If you choose to insert data at row #2, the previous data at row #2 will move to row #3 and any subsequent data will also move down a row.

The "Add/Remove Row" behavior is only part of managing "Tables". Once a new row is created, the "Change Table Value" behavior is used to populate the rows with data.



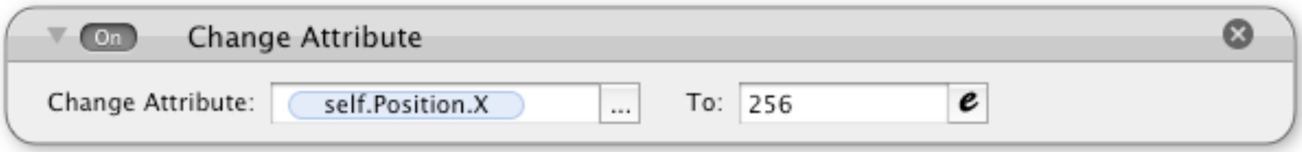
Speed: 30 fps [✓ Loop] Restore actor image when done

Animate - Much like a cartoon, GameSalad actors can have frames of animation. By drag-and-dropping images onto the "Animate" behavior, you can create an animated actor. This feature gives GameSalad plenty of options. Use it to create explosions, 3D animated characters or even movies. However, there are performance concerns associated with this option. If you have too many images in your project, or if the images are too large, your game could slow down and the loading times could increase dramatically. It depend on your project. Choosing your animation speed is a delicate balance between quality and performance. The slowest speed setting is 0 fps, which is pretty pointless for animations. The maximum speed is 30 fps.

The "Loop" check box enables the animation sequence to be repeated. If you want to create a running effect for your actor, looping the animated images is an effective way to do it. In addition to character animations, other possibilities are rocket thrusters, ocean water, environmental effects, flashing lights and more impressive projectiles. For more on this topic, see Chapter #11 - Graphics.

The "Restore actor image when done" check box can reset the actor to the original graphic after the animation has stopped.

To give an actor an image, click-and-drag an image to the "Drag Image Here" area. Images can be pulled from your Mac OS desktop or from the "Images" tab of the "Media Panel". More information about "Actor" settings, can be found in this chapter.

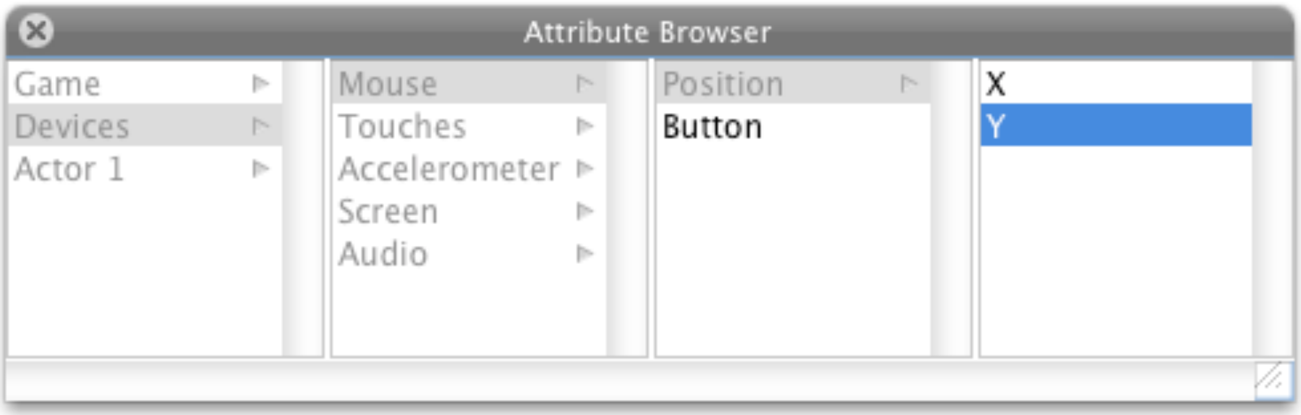


Change Attribute: self.Position.X ... To: 256

Change Attribute - If you're aiming to be a serious GameSalad game developer, you'll probably become quite acquainted with this behavior. "Change Attribute" is frequently used. In the left field, you select the attribute that you want to change. In the right field, you can set the new value. This behavior is useful for so many aspects of gaming — high scores, health, actor properties, scene values, game settings, etc.

In order to prevent issues, you might want to check the attribute's type before entering values in the "To" field. For example, putting text in a numerical attribute can cause text display problems — unless your goal is to display the number zero.

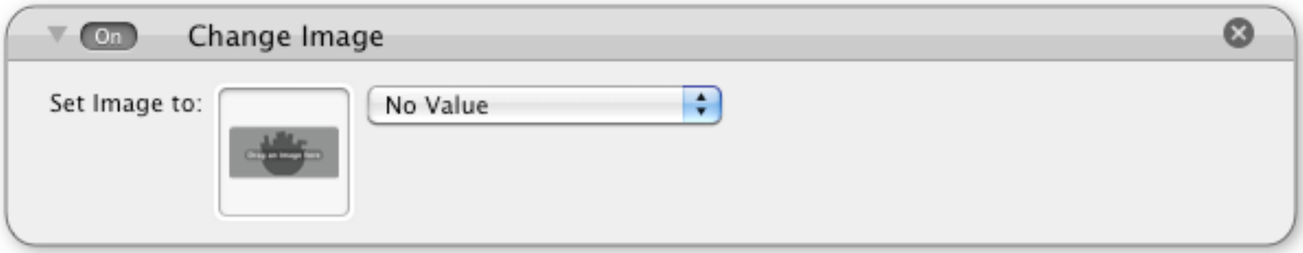
In the "Change Attribute" field, there is an ellipse on the right side. Clicking that icon will trigger the "Attribute Browser" pop-up window.



Generally, there are three main options to choose from. "Game" is for game specific attributes. The default game attributes are not really for use with the "Change Attribute" behavior, but your custom game attributes can handle it. The "Device" section is pretty much the same story. These attributes are more for reading than writing. But with the "Actor" section, you can have a lot of fun. In addition to custom attributes that you can create for your actors, you can also change an actor's size, color, location and rotation.

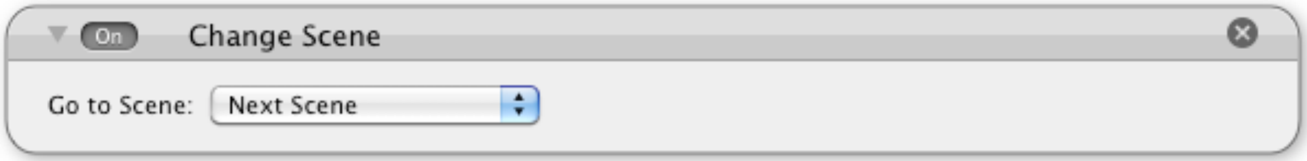
In the "Behaviors" section of the "Media Panel", the "Change Attribute" behavior has a small "A" icon next to its name. That stands for an "Action" behavior. The "Change Attribute" only happens once, unless recalled by a "Rule", a "Timer" or something of that nature. That makes the "Change Attribute" behavior different than the "Constrain Attribute" behavior. Changing an attribute is less intensive than constantly constraining it. However, your game design might require a constraint.

Now you might be wondering to yourself, "Hey dude, what's with that little 'e' icon?" That little button triggers the "Expression Editor" pop-up window. It grants access to mathematical power. More on this in Chapter #5 - Math.



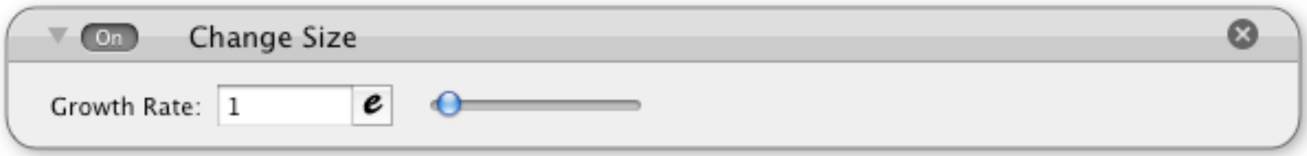
Set Image to: No Value

Change Image - This behavior allows you to change the image of an actor while the game is in action. Unlike with the "Animate" behavior, this is a change to a single image. There are many practical uses for this option. Ever play a racing game where you were driving at 188 miles per hour, and then slam head first into the wall? What happened to your car? With too many racing games — NOTHING — absolutely nothing happened to your car. Your car simply bounced off the wall and you kept on driving. But with the "Change Image" attribute, you can give your actor battle damage. Combine this behavior with rules or timers to create different looks for your actors.



Go to Scene: Next Scene

Change Scene - This behavior allows you to create levels for your game. When a player completes a task in one scene, you can trigger the "Change Scene" behavior to move the player to the next level. You could also use this behavior to create title screens, end game credits or other types of game states. The trick is to watch out for the loading times. If your scenes take too long to load, switching a scene could cause a dramatic break in the gaming experience. To avoid this issue, sometimes I limit my projects to one scene. There's a drawback to that scenario too. Cramming everything into one scene can cause problems with the performance of your game. It's a delicate balance. For more on this topic, see Chapter #18 - Optimization.



Growth Rate: 1

Change Size - This behavior allows you to increase the size of your actor. The larger the "Growth Rate" the faster the size increase. Setting this behavior to zero renders it useless. However, the rate can be negative. Setting the rate below zero will shrink an actor. However, the graphical slider does not go below zero. To decrease the size of an actor, manually enter a negative number.

I'm not a fan of the "Change Size" behavior. With the introduction of the "Interpolate", the older behavior is a bit obsolete. With "Interpolate", I have greater control of the size. I can specify the exact size I want the change to be and I can specify exactly how long I want the change in shape to be. With the "Change Size" behavior the increase in size is infinite. It will keep on growing until it is stopped by something else, like a timer or the removal of the actor. (A reduction in size will cease when either the width or height of the actor reaches one pixel.)

"Change Size" does have an advantage. To change size with "Interpolate", two behaviors are needed — one for width and one for height. With "Change Size" both width and height can grow simultaneously.

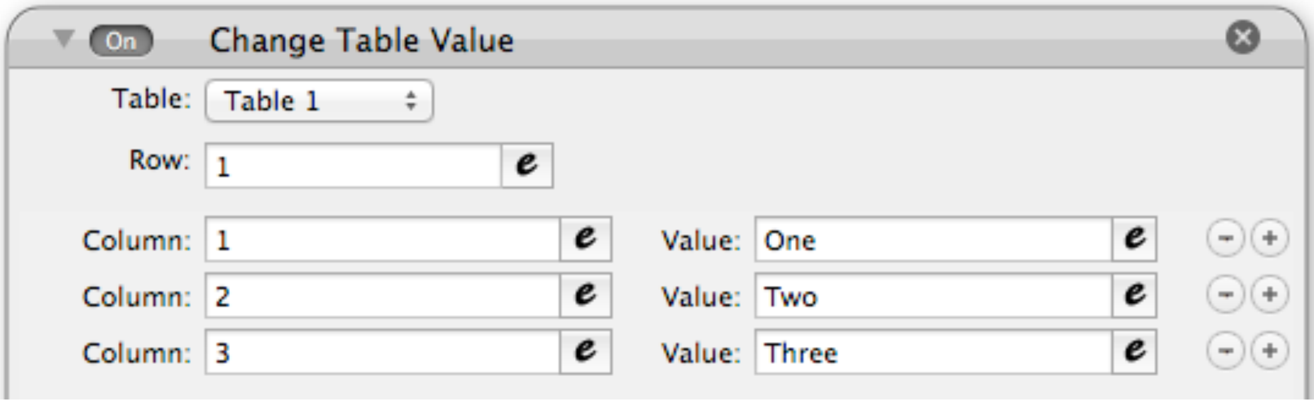


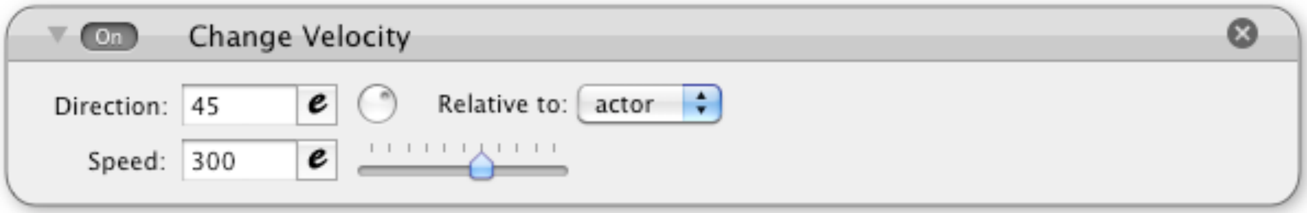
Table: Table 1

Row: 1

Column:	1	Value:	One
Column:	2	Value:	Two
Column:	3	Value:	Three

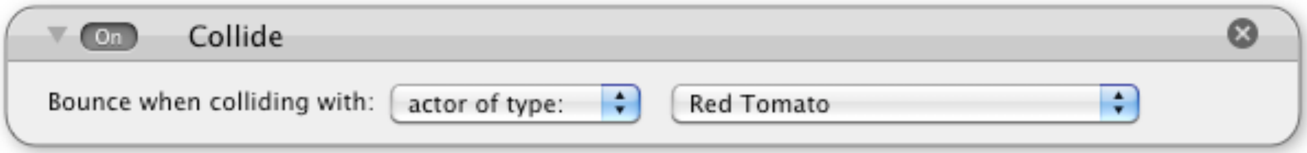
Change Table Value - Similar to "Change Attribute", the values in a "Table" can also be changed. The "Table" option allows you to select which set of data you want to work with. The "Row" field is for specifying the number of the row to work with. The "Column" field works with the "Value" field. The "Column" field is for specifying the "Column" number while the "Value" field is for entering the new data. If you want to work with multiple columns, you can click the plus icon to add new Column/Value fields.

There are important things to remember when entering new values. The "Change Table Value" behavior cannot create new rows or columns. To create a new Row, use the "Add/Remove Row" behavior. Columns cannot be created while the game is running. They need to be created beforehand. The "Attribute" type also needs to match. Placing "Text" in an "Integer" column could cause problems for your game.



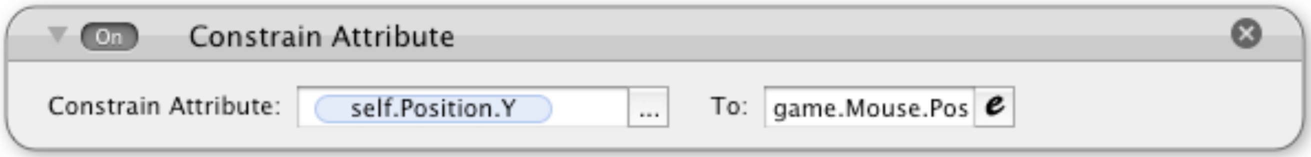
Direction: 45 Relative to: actor
Speed: 300

Change Velocity - This will move an actor in a direction, but it's a one-time action. Unlike the "Accelerate" or "Move" behaviors, "Change Velocity" is not constant. If the actor's "Drag" is greater than zero, eventually the actor's velocity will cease. Bumping into a wall or another actor can also change the velocity. This behavior is good for a one-time hit, like an explosion. It's also useful for when actors are first launched into an active scene, like a pinball hit by a plunger.



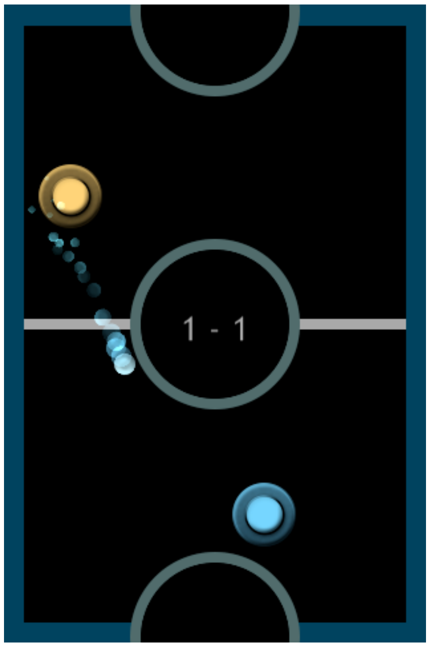
Bounce when colliding with: actor of type: Red Tomato

Collide - Now this is the kind of behavior that builds video games. If you want an actor to bounce when touching another actor, use the "Collide" behavior. The strength of the collision is determined by the actor's speed and the related "Physics" setting for the involved actors. (For more information, see Chapter #6 - Physics.) No collision will occur if actor has disabled movement. Collision detection could also be accomplished with the "Rule" and "Change Velocity" behaviors, but I find that the "Collide" is more consistent and it can uses the physics settings. In either scenario, tags are useful for managing collision detection. If you have a group of enemies, all with the same reaction to a specific collision, you can streamline your game logic with tags. Instead of creating individual collide behaviors for enemy X, enemy Y and enemy X, you can use the "actor with tag" option. Place all of the enemies in a single tag, and then set collisions to occur with that tag. If you need a specific collision for a single actor, use the "actor of type" option. The drop-down menu on the right will let you select from a list of actors if "actor of type" is selected, or from a list of tags if "actor with tag" is selected.


Constrain Attribute

Constrain Attribute: self.Position.Y To: game.Mouse.Pos

Constrain Attribute - This will constantly restrict an attribute to a specific value. The "Constrain Attribute" behavior is more processor intensive than the "Change Attribute" behavior. For optimizing your game, it's important to know when to use a constraint and when you shouldn't.



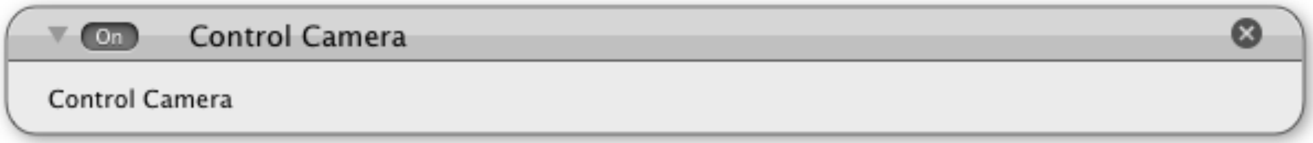
For example, I made an air hockey game. There was simply no way around it. For controlling the paddle, a "Constrain Attribute" behavior is necessary. When I touched the screen, on my half of the playing area, I wanted the paddle to follow my finger. The movement is constant, so the attributes "self.Position.X" and "self.Position.Y" needed constant updating. That's why "Persistent Behaviors" were used.

However, the player paddles should remain on the playing area, not on the walls or the other side of the field. At first, I created a "Rule" to change the paddle location. If the paddle was beyond a certain point, it's location would be changed back onto the playing area. Not only did this "Change Attribute" method create a poorly controlled paddle, I had to create rules for all of the boundaries. My game design didn't work, but I didn't want to add more "Constrain Attribute" behaviors. Five constraints for one paddle seemed a bit excessive to me. I had to rethink my game's design. Since I was already using a "Constrain Attribute" to control the paddle, I decided to improve the controls with an expression.

max(40,min(game.Mouse.Position.Y,216))

Look out! It's math, logic, formulas and stuff... run and hide... ah! Without getting too complicated — for now — the "min" and "max" functions are essentially being used as constraints. Instead of creating too many constraints, or ineffective rules, I optimized the game's logic to help out. This improved the game's performance.

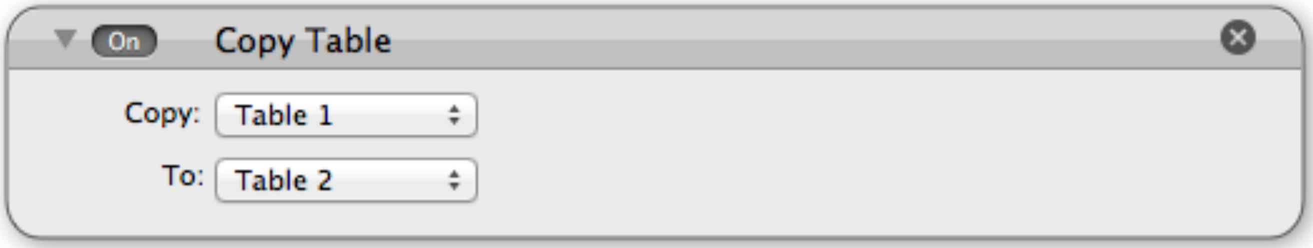
For more on mind melting matters like this, see Chapter #5 - Math. You could also check out the "Air Hockey Template". The point for the "Constrain Attribute" behavior is to be careful with how you use them. If you use too many, or if you use them improperly, it will seriously degrade the performance of your game. You might want to consider alternate game design ideas or the possibilities of the "Expression Editor".



Control Camera

Control Camera - If you're making a game that scrolls, you can use the "Control Camera" behavior to follow an actor. You can edit the sensitivity of the camera tracking from the "Scene Editor". This is a "Persistent Behavior" but it has a fail-safe — to keep your computer from having equivalent of a brain aneurysm. Basically, only one actor can control the camera at a time. If multiple actors are using this behavior, only the most recent use of this behavior will be acknowledged. This behavior will be ignored if it's called by an actor on a layer that is not set to scroll.

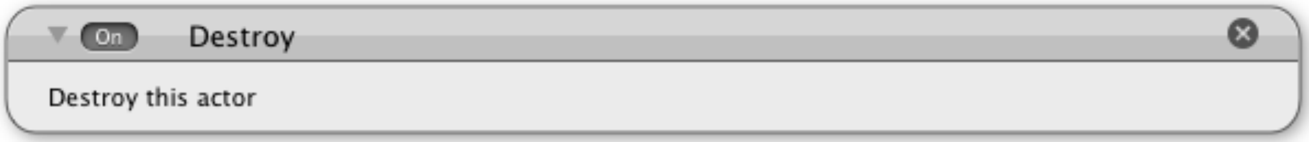
There are some creative uses for this behavior. If you're making a sports game, it's easy to make the camera follow the ball. However, you could make the camera follow a specific player. This approach could also be used for a real-time strategy game, for controlling units. The camera will cut to the new location of the actor. With the combination of quick cuts and scrolling camera movement, your playing area might seem larger and more dynamic. For even more unique camera movement, the "Control Camera" behavior could be made to follow an invisible actor.



Copy: Table 1

To: Table 2

Copy Table - Just as the name suggests, the “Copy Table” behavior will duplicates the contents of one “Table” to another. The “Copy” option allows you to select the original data set, while the “To” option selects the target. The target table is completely overwritten. Any extra rows or columns are removed.



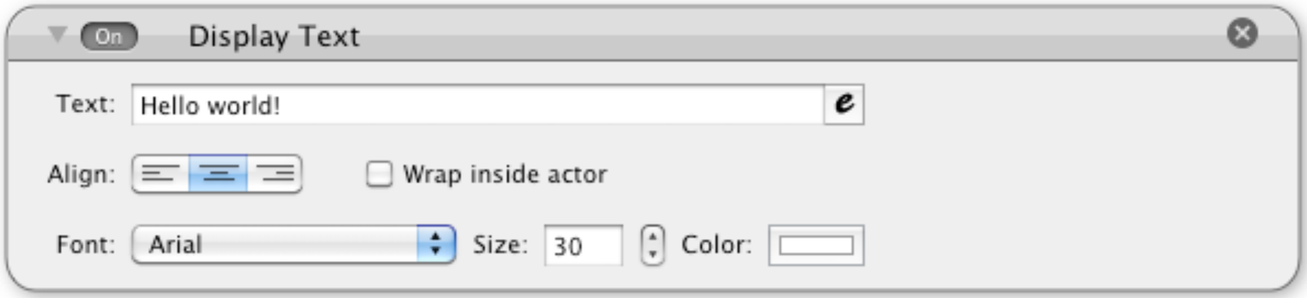
Destroy this actor

Destroy - When a game is in play, you can remove an actor from a scene with the “Destroy” behavior. This is often used to simulate an actor’s death. This behavior is usually not used by itself. Generally, it’s called by a “Timer” or a “Rule” behavior. For example, if an actor collides with a bullet, that actor is destroyed. There are alternatives to destroying an actor. It could be moved off-screen. It could also be made invisible, by setting the “Alpha” channel to zero.

Display Text

This behavior allows you to show text on top of an actor. However, there are some tricks needed in order to make this behavior more functional. For example, if you press the enter/return key in the “Text” field, it acts as if you’re done typing text. But if you’re not done typing text, and you wanted to add a new line instead, you can use the Alt+Enter key combination. That should start a new line.

The displayed text is vertically aligned to the center of an actor, but you can set the horizontal alignment with the "Align" setting. The three choices are left, center and right.



Text: Hello world!
Align: Wrap inside actor
Font: Arial Size: 30 Color:

If you have long paragraphs of text, the "Wrap" option will keep the text inside the actor’s box. Additional lines of text will automatically be created if the words are bigger than the width of the box. This can be useful for creating storybooks or other blocks of text.

To make the text look more attractive, there are several things you can do. The “Font” drop-down menu gives you the option to choose from different typefaces. The “Size” option lets you choose how big you want the text to be. The larger the size, the bigger the characters. The “Color” field lets you select a color for your text. The text will display beyond an actor’s size, so you don’t have to treat an actor like a bounding box for text. Also, if you don’t want the actor to be seen at all, you can set the actor’s “Alpha” channel to zero. An actor’s “Alpha” setting is different from making an actor invisible. If an actor’s “Visible” setting is unchecked, the actor’s text will not be shown.

The “Display Text” behavior also works with attributes. By using the “Expression Editor”, an attribute can be selected. The data contained in the attribute will be shown as text. However, there are even more tricks to know with this method. If you want to show more than one attribute, double dots are needed as separators.

```
game.Mouse.Position.X..game.Mouse.Position.Y
```

However, there’s a problem with this technique. If you try displaying the above expression in GameSalad, you’d probably see the two mouse positions shown as one confusing blob of characters. The above example has no spaces between the two sets of data. This is where the “Display Text” behavior gets tricky. While using expressions, regular spaces are no longer an option. Also, if you simply try to add regular text, it probably won’t display properly. So, how do you combine plain text with attribute data?

This technique is less intuitive than most things in GameSalad, but it’s still powerful feature. By using double quotes around regular text, and then separating that section with double dots, combinations of data can be displayed.

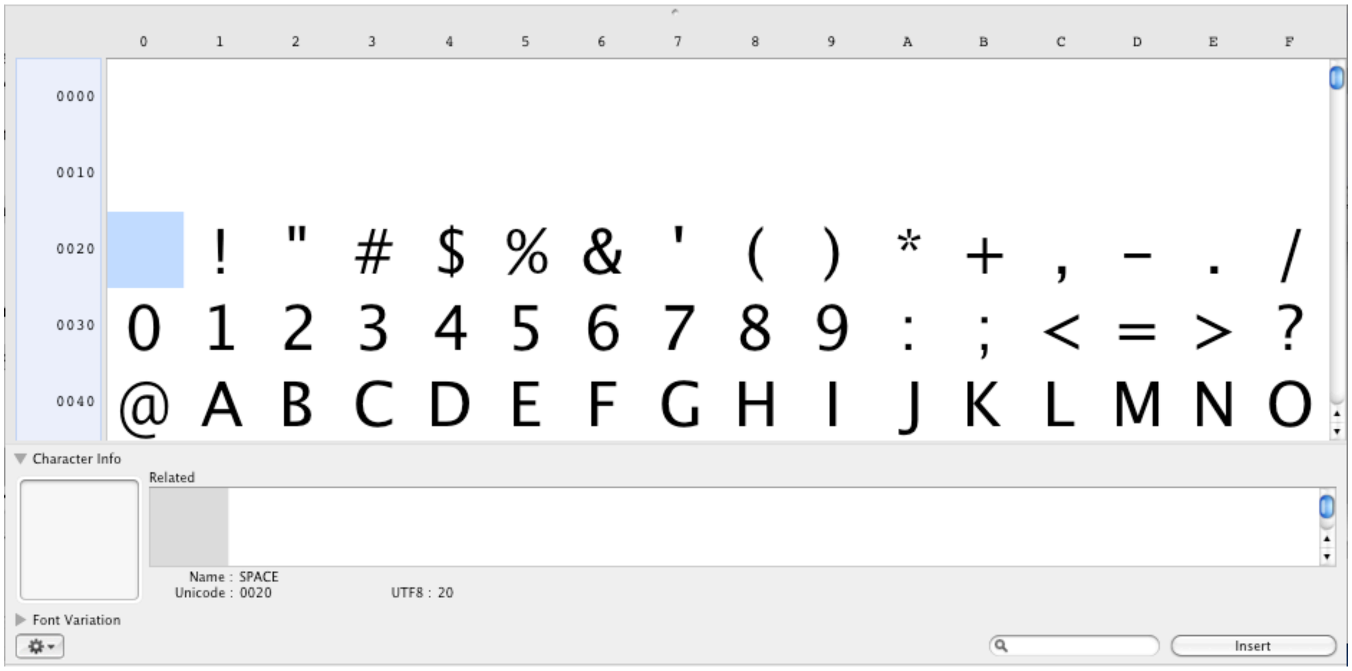
```
"X:"..game.Mouse.Position.X.."/".."Y:"..game.Mouse.Position.Y
```

If you try entering this example into GameSalad, the result is more readable than before. Yet, there are no spaces. If you try pressing the spacebar while using expressions, GameSalad simply stops you.

There are two ways to solve the spacing issue. By using a backslash, you can perform some pretty sneaky tricks with the “Display Text” behavior. For example, if you want a new line, just type \n in the “Text” field. (You could also use \r for a carriage return.) The following example shows how to add a space.

```
"X:\32"..game.Mouse.Position.X.."\32/\32".."Y:\32"..game.Mouse.Position.Y
```

By using escapes, \32 is the answer! Why 32? Why not 42, 0 or 1337?



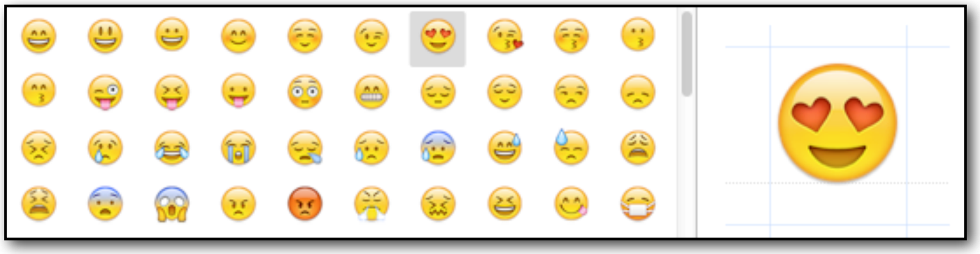
Name : SPACE
Unicode : 0020
UTF8 : 20

While running GameSalad, if you select "Edit" and then "Special Characters" from the menu bar, you can activate the "Characters" pop-up window. This is part of the Mac operating system. The image above is from Mac OS X 10.6 (Snow Leopard). In the grid, the "SPACE" character sits at spot number 32. By entering a character's position number into the "Text" field, it can be displayed. But unfortunately, it only works up to \126 — the tilde. Spot 127 doesn't seem to do anything and usage of \128 or above can crash GameSalad. Aside from a space, or a double quote, There's almost no sense in entering characters this way. With GameSalad, you only need to know how to escape a few characters.

Escape	Result
\"	Double Quote
\34	Double Quote
\32	Space
\n	New Line
\r	Carriage Return
\t	Tab

However, there is an easier way to add a space. If you press alt+space, you can make a space. That's much easier than using the \32 option.

For the most part, as long as your text is within the quotes, it should work fine. Although, you can run into cross-platform issues with special Characters. Emoji icons are a great example.



Just because a special character works on the Mac or even iOS, doesn't meant it will work on Windows, Android or the Web.

Placing multiple attributes in a single text field can create interesting possibilities. For example, it's an efficient way to use actors. Without the use of expressions, you might try to cheat. You might be tempted to use multiple actors, where only one actor would be sufficient. Too many actors in your GameSalad game will slow it down. You'd also have to waste time by aligning the two actors together to form sentences/paragraphs. By using text wisely, you can create cool effects and optimize your game. What kind of cool effects? If you're creating an RPG, you could make the dialogue more interactive with attributes. "Hey, Human, what are you doing?" Attributes could be used to display a character's race. Instead of having to create a new line of text for every race in the game, you could simply create the line once.

"Hey,\32"..game.Race..",\32what\32are\32you\32doing?"

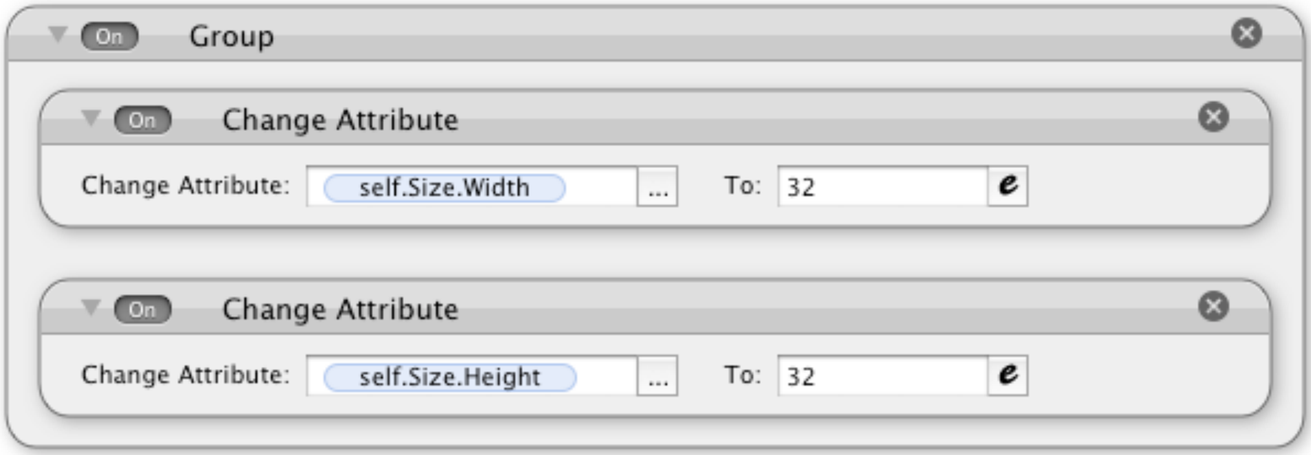
or

game.Text-1a..game.Race..game.Text-1b

Either method will work. For games with a lot of text, the second method might be more manageable. By keeping all of your text in separate attributes, the text is more readable and quicker to edit. You don't have to fumble with the backslash key. Essentially, the attributes list can become like a database for all of your dialogue.

Even better, by using tables to manage your game's text, multilingual translation becomes easier to manage. Each column could be a new language and each row could contain a different word or phrase of game text. An example of this can be found in the "Multilingual" template.

Behaviors Continued


Group

Change Attribute

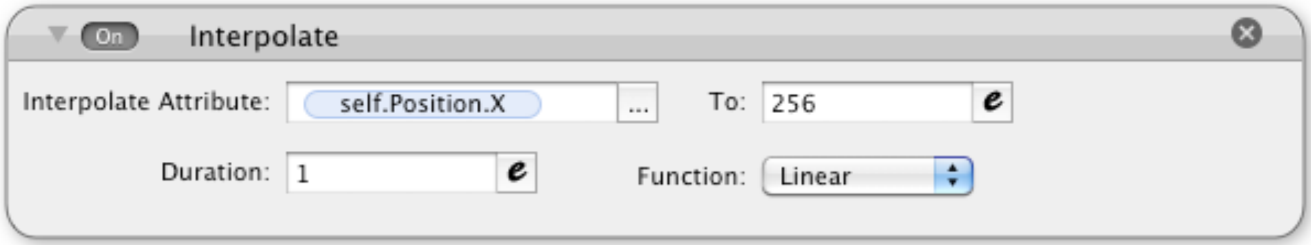
Change Attribute: self.Size.Width To: 32

Change Attribute

Change Attribute: self.Size.Height To: 32

Group - This behavior is for tidying up your game logic. The use of this behavior is optional, but you could “Group” to pair similar behaviors. For example, to change an actor’s size, you need to set the “Width” and the “Height” values.

The two “Change Attribute” behaviors can be dragged into the “Group” behavior. The container expands to hold both of the behaviors. Then, when you’re done, you can click the gray arrow to collapse the group. The contents will be hidden until revealed again.


Interpolate

Interpolate Attribute: self.Position.X To: 256

Duration: 1 Function: Linear

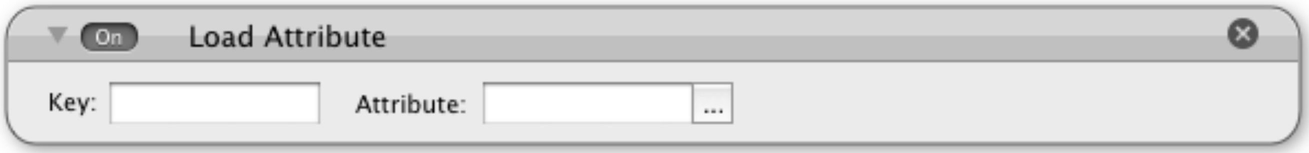
Interpolate - Use this behavior to transition from one attribute value to another. Unlike “Change Attribute”, “Interpolate” increases or decrease a numerical value. It is not intended for text. The “Interpolate Attribute” field is for the attribute that’s being changed. The “To” field is for the new value. The “Duration” field is for the length of time (in seconds).

The “Interpolate” behavior has an advanced design, which makes it fairly powerful. It can be used to move an actor, even if an actor’s “movable” option has been deselected. With the “Duration” feature — a built-in timer — this behavior can accomplish tasks that would normally require multiple behaviors. By changing an actor’s “Alpha” channel, this method can also be used to create screen fades, such as fade-to-black or fade-to-white.

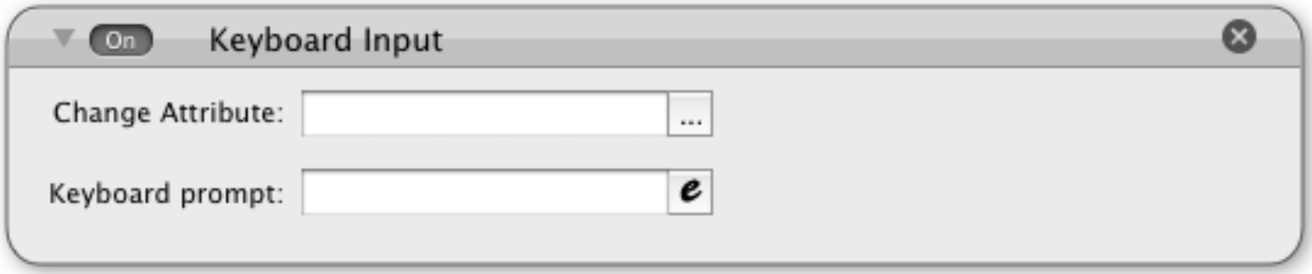
The “Interpolate” behavior also has an option to control how the interpolation transpires. The “Function” drop-down menu is for choosing the interpolation path.

- **Linear** - provides a constant rate of change.
- **Ease in** - causes the initial rate of interpolation to increase slowly.
- **Ease out** - causes the final rate of interpolation to decrease slowly.
- **Ease in/out** - causes the initial rate of interpolation to increase slowly and the final rate of interpolation to decrease slowly.

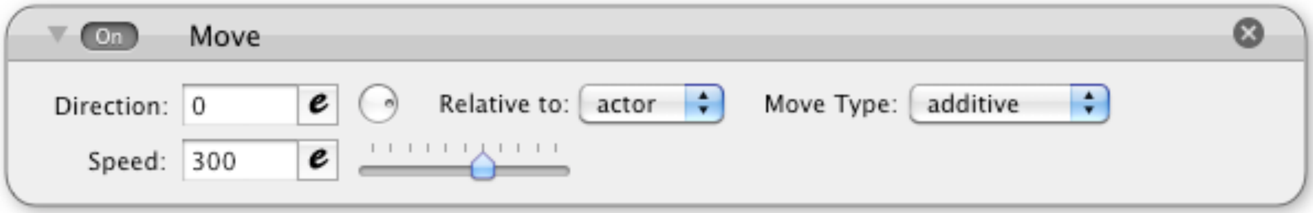
You could use “Ease in” to mimic something like gravity, as the initial speed is slower than the ending speed. You could use ”Ease out” for smoother animation or to mimic friction, like a car rolling to a stop. Ease in/out” could also be used for smoother looking animation, like a patrolling enemy, a pendulum or an actor that moves from one spot to another. The gradual increase and/or decrease can create a more realistic effect.



Load Attribute - With this behavior, saved attributes can be reloaded into the game. The “Load Attribute” is used with “Save Attribute”. Together they can save a player’s progress, keep track of high scores or restore game settings. The “Key” field is for the name of the saved data. The “Attribute” field is for choosing an attribute to associate with the saved data.

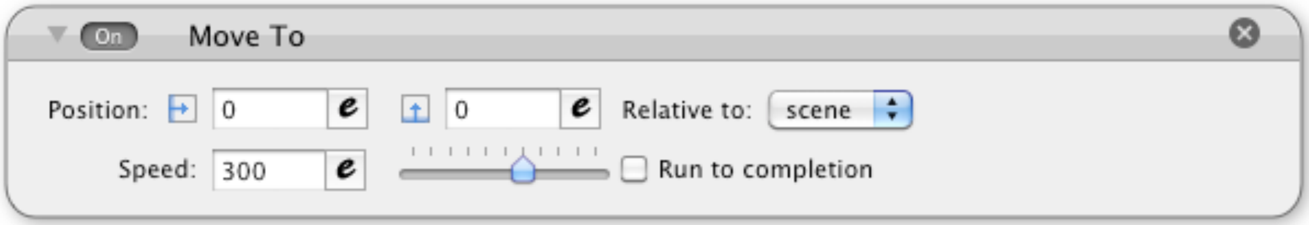


Keyboard Input - If the iOS devices don’t have a physical keyboard, how can you let players enter information? With the “Keyboard Input” behavior, you can activate the iOS keyboard. This behavior also works for creating Mac and HTML 5 apps. There are only two options with this behavior. “Change Attribute” lets you select a target attribute. That’s where the entered data will be saved. The “Keyboard prompt” field gives the onscreen keyboard a title. This feature is useful for letting players customize their gaming experience — like naming their character. It could also be handy for creating in-game quizzes or a level-select password system.



Direction: 0 Relative to: actor Move Type: additive
Speed: 300

Move - This behavior is a constant stream of speed. It’s like cruise control for actors. Unless some other force acts upon the affected actor, the “Move” behavior will move the actor at a constant velocity. Like with the “Accelerate” behavior, the options are similar. Use the “Direction” field to set the angle of movement. “Relative to” is a toggle between “actor” or “scene” based movement. “Speed” controls how fast the actor will move.

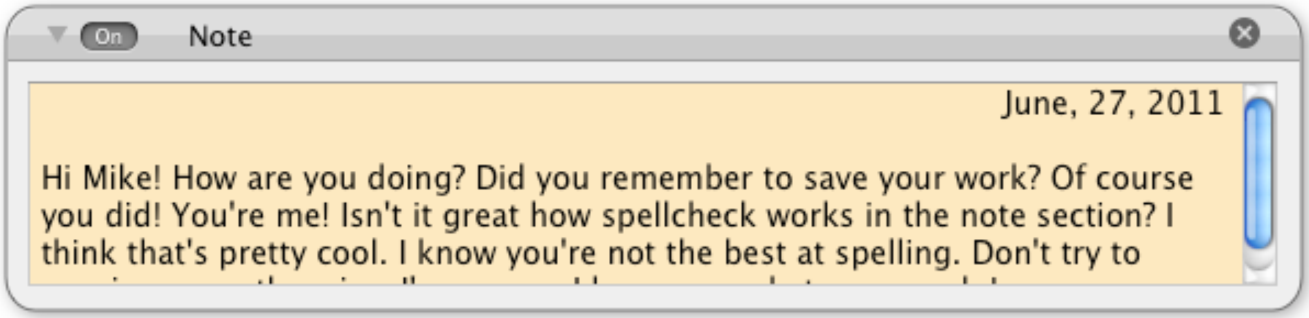


Position: 0 0 Relative to: scene
Speed: 300 Run to completion

Move To - This behavior is to set a target for an actor. When activated, the “Move To” behavior will cause an actor to speed toward a specific location. Once at the location, the movement stops. This is known as a “Completing Behavior”. However, the target location is not necessarily static. If attributes are used in the “Position” settings, those attributes could be changed before the movement stops. If the target is changed before the movement is stopped, the actor will change direction and follow the new coordinates. If the actor reaches the destination before the “Position” values have changed, movement stops and further target changes are ignored. If the “Move To” behavior is recalled by a “Rule” or a “Timer”, the movement can start again.

A guided missile is a good analogy. A heat seeking missile will follow its target until it explodes. When it reaches its destination - boom! That’s similar to what happens with this behavior. Once the target is reached, the movement stops. (If you want the actor to explode, you can do that too.) “Move To” and “Accelerate To” are excellent tools for creating artificial intelligence. However, unlike the "Accelerate To" behavior, the "Move To" effect will stop once the target is reached.

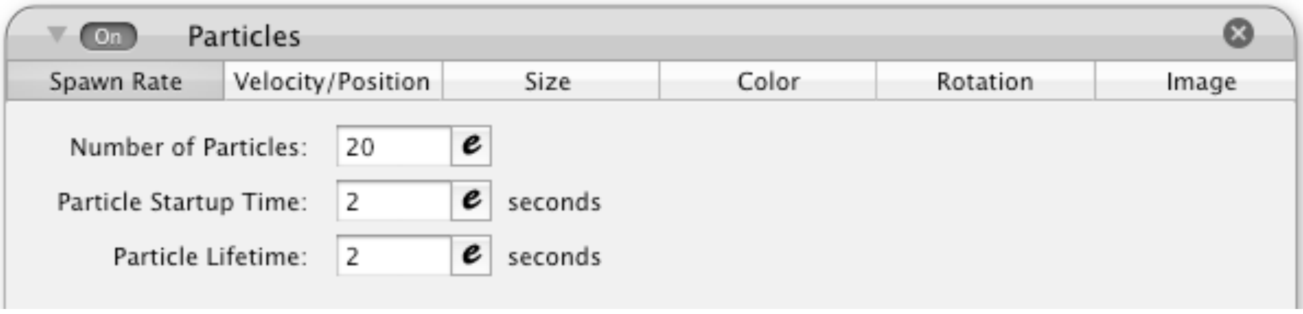
"Run to completion" will continue this behavior, even if a "Rule" related condition has ceased. In other words, if you place "Move To" in a "Rule", use the "Run to completion" option when you want to finish the movement. Otherwise, the movement will stop when the rules are no longer met.



June, 27, 2011

Hi Mike! How are you doing? Did you remember to save your work? Of course you did! You're me! Isn't it great how spellcheck works in the note section? I think that's pretty cool. I know you're not the best at spelling. Don't try to

Note - It's a bit funny to me. GameSalad is so powerful that I sometimes forget about my completed tasks. Since I usually work alone, I rarely leave notes for myself. Instead, I'll go back and try to remember what I've done before. You don't have to operate so inefficiently. You can leave messages for yourself in GameSalad. The "Note" behavior is especially useful if you're working in a team or creating tutorials for others. (Note: I listed this as a behavior, but the more technical term is "Text Content Container".)

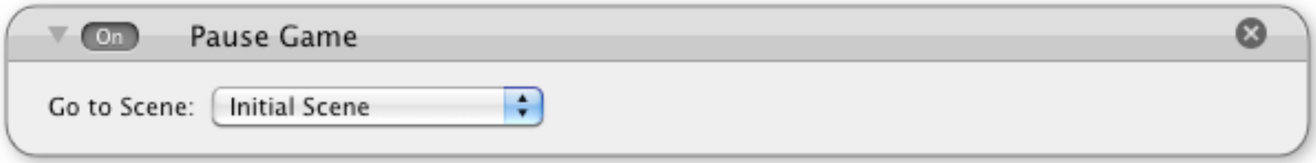


Number of Particles: 20

Particle Startup Time: 2 seconds

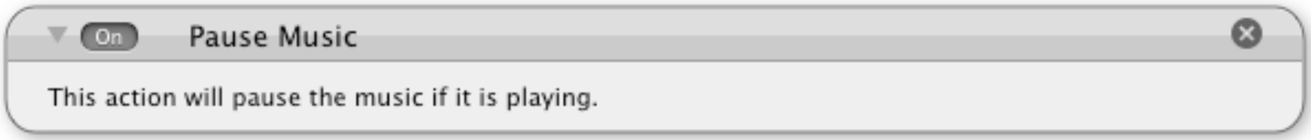
Particle Lifetime: 2 seconds

Particles - This behavior is just too awesome to be explained here. For more information, see Chapter #24 - Particles.



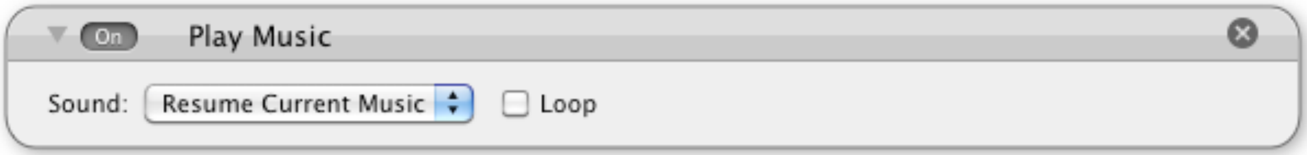
Go to Scene: Initial Scene

Pause Game - This behavior will stop the physics of the current scene and overlay the selected scene on-top of the layer. This is great for creating in-game menus, where the player can change game options, take a break from the action or restart the game. There is a danger using this option. While it freezes the physics, it doesn't stop the music or the sounds. Depending on your game, you might need to use the "Pause Music" behaviors too. Also, watch out for loops. Using a pause within a pause can make your game highly unstable. The overlayed scene should be light — something that loads quickly, without a lot of actors, behaviors or large images. The "Unpause Game" behavior can be used to resume the game.



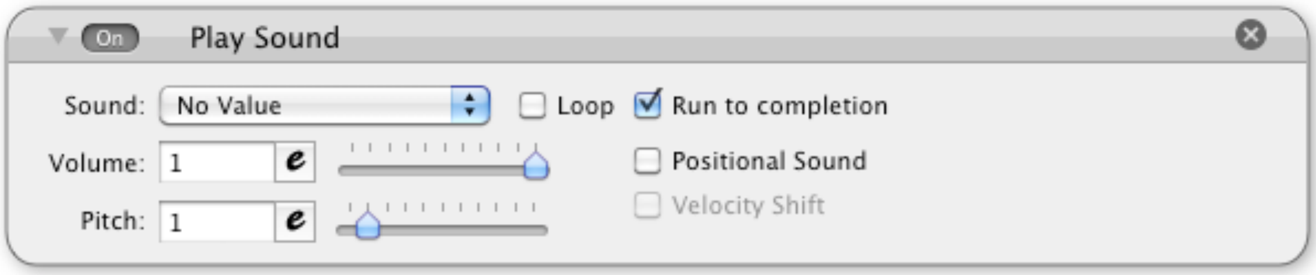
This action will pause the music if it is playing.

Pause Music - If you want to stop the background music, you can use the "Pause Music" behavior. This works well with a button, giving the player the ability to toggle music. This behavior works in conjunction with the "Play Music" behavior.



Sound: Resume Current Music ☐ Loop

Play Music - There are two audio types in GameSalad games - "Music" and "Sound". If you want to put background music in your game, the "Play Music" behavior can help you do it. The "Sound" drop-down menu lists the available music files in your GameSalad project. It does not list files that have been formatted for sound. To continue playing music that was previously paused, select the "Resume Current Music" option. The "Loop" check box is for repeating the music file. Once the end of the song is reached, "Loop" will start the song again from the beginning. This is a good option to enable if your sound files are designed for it.



Sound: No Value ☐ Loop ✓ Run to completion
Volume: 1 ☐ Positional Sound
Pitch: 1 ☐ Velocity Shift

Play Sound - If you want to add sound effects to your game, you can use the "Play Sound" behavior. This is an "Action" behavior, which is often combined with a "Rule" or a "Timer". The "Sound" drop-down menu lists the sound files in your project. Only sound files — not music files — are listed. The "Loop" option will continuously restart the sound file after it has finished playing.

The "Run to completion" option will allow the sound to keep playing, even if a "Rule" condition is no longer met or the actor has been deleted. This option is useful for actors that have died, as the sound will keep playing even though the actor was destroyed. However, there are issues with after-death actor sounds. For example, if "Loop" and "Run to completion" are checked, the after-death sound will be stopped immediately.

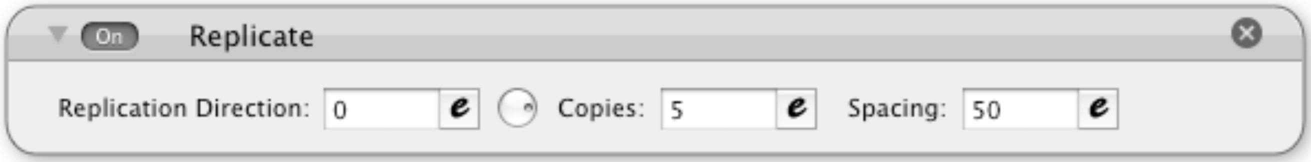
The "Volume" option allows you to set how loud the sound effect can be. The numbers range from 0 to 1. For 50% volume, enter .5 into the field. The slider can also be used to raise or lower the "Volume" number. The range does not go below zero or above one.

"Positional Sound" is similar to volume, but it's based on actor location. If the actor is on the left side of the screen, this option will make the left speaker sound louder than the right speaker. If the actor is on the right side of the screen, the right speaker is louder than the left speaker. It depends on how far the actor is away from the center. This technique works well with non-scrolling scenes.

"Pitch" can speed up or slow down the playback rate of a sound file. The higher the number, the faster and more squeaky your sounds will become. This technique is useful for using a single sound file in multiple situations. For example, a small explosion might have a high pitch rate. A larger explosion might have a lower pitch rate.

"Velocity Shift" takes advantage of "Pitch" modification. Depending on your actor's speed and direction, this effect will modify the sound accordingly. This technique can be used to create a better sense of speed, like race cars or environmental effects. When "Velocity Shift" is used on a colliding actor, the sound is quite profound.

For more information on Pause Music, Play Music and Play Sound, see Chapter #12 - Audio.



Replication Direction: 0 Copies: 5 Spacing: 50

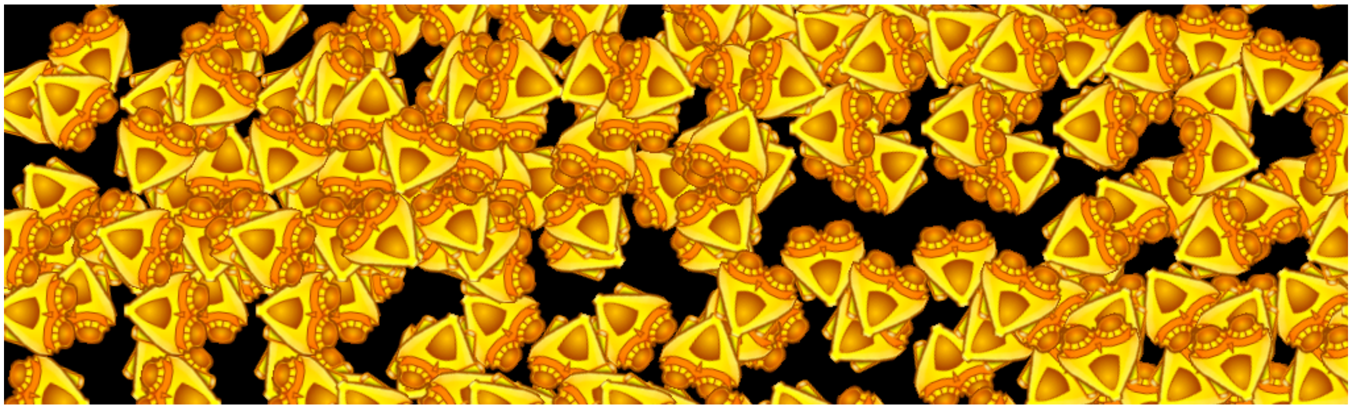
Replicate - This behavior will duplicate the main actor. These clones will exist in a location that's relative to the main actor. The "Replicate" behavior is a way to visually displaying the number of player lives remaining.

The "Replication Direction" setting specifies the angle of alignment line. A setting of 90° would align the clones directly above the actor. More commonly, a direction of 0° would align the clones directly to the right of the actor. The angle is "Relative To" the scene.

The "Copies" field is where you can specify the number of clones to create. This number can be zero, in order to show that there are no more lives remaining. If the number is less than or equal to zero, the main actor will be invisible — but not destroyed.

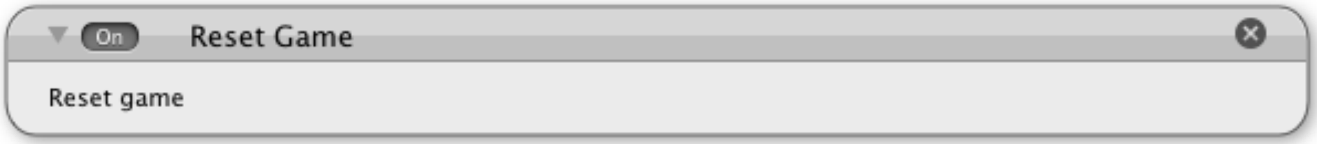
The "Spacing" field determines the amount of pixels between center points. That's different than the spacing between the graphics. To prevent overlap, the radii of the actors also has to be considered.

I didn't seriously consider this feature until I worked on this book. My game design philosophy is this — you only get one life in real life, so that's how it should be in my games. I didn't have a use for this "Behavior", so I tried to create some interested effects with it. If you're not careful with the "Replicate" behavior, things might get a little crazy...



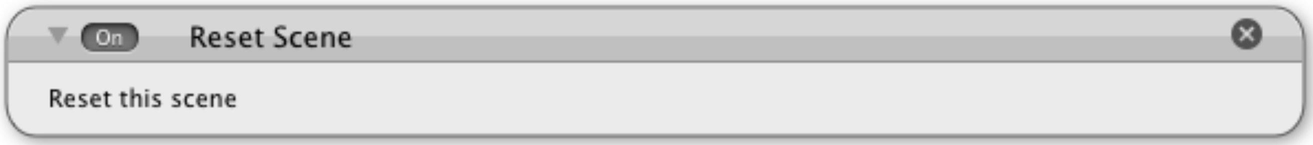
Apparently, spawning actors that use the "Replicate" behavior can quickly populate a screen. Instead of spawning one actor, it was like spawning five. It depends on the "Copies" value. I was a bit confused as to what was going on, but I did notice some interesting things. Only the lead actor will have collision detection. The clones will not. Also, when the "Spawn Actor" behavior is used, only the lead actor will create a new actor. The clones will not.

However, "Particles" will work on the main actor and the clones. Instead of having particles shoot out from a single point, "Replicate" can create a series of "Particle" generators. This could create interesting effects.



Reset game

Reset Game - This behavior acts very much like the reset button on video game consoles. When the "Reset Game" behavior is used, the game is restarted. In doing so, all of the unsaved data is lost. Actors, attributes, scenes are all restored to the initial game state. There is an exception. The game.Time attribute will not be reset. This behavior is useful when the game is over. After a certain amount of time, or triggered by a button, the "Reset Game" behavior could be called — returning the player to the title screen.

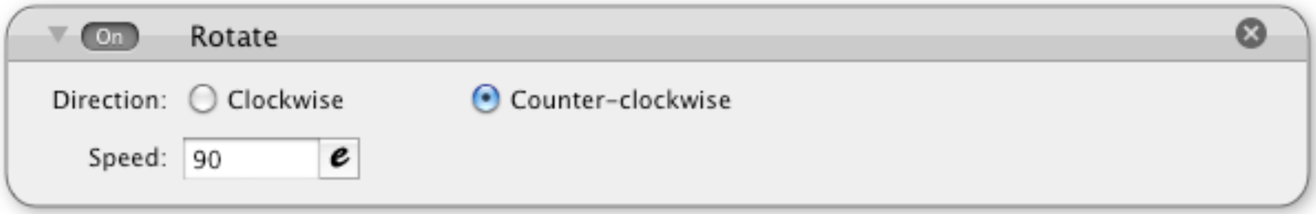


Reset this scene

Reset Scene - This behavior is less dramatic than the "Reset Game" behavior. Instead of resetting the entire game, only the scene's actors and attributes are reset. While using this method is more precise, greater care may be required.

If you're like me, you don't like to create scene attributes. It's easier to create game attributes instead. I like to have the option of accessing data across multiple scenes. But if only the scene is reset, and not the game attributes associated with that scene, it could cause problems with the replay of that scene.

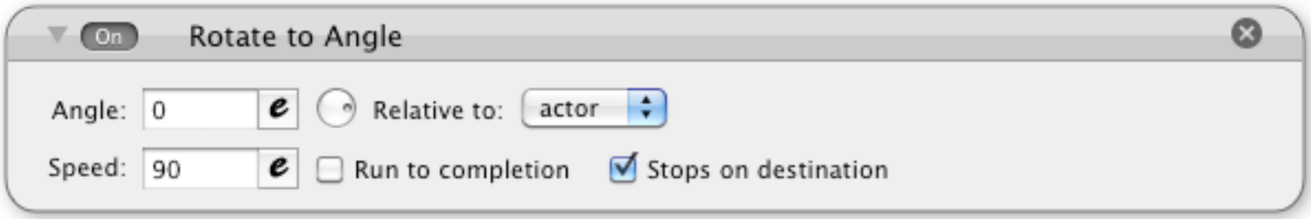
If it's such a hassle, why use the "Reset Scene" behavior? Reloading the whole game could annoy the user. If the player wants to retry a level that they've just played, the "Reset Scene" keeps the player right where they want to be. Although, reloading the whole scene could take much longer than just resetting a few actors and attributes. Building a better user experience for your players could help with sales. That means keeping the loading times to a minimum.



Direction: Clockwise [✓ Counter-clockwise]
Speed: 90

Rotate - This behavior is used to spin a character on its center point. The radio buttons enable you to choose between "Clockwise" and "Counter-clockwise" rotation. The "Speed" field can increase or decrease the rate of rotation. The "Speed" value can be negative, which can negate the clockwise / counter-clockwise options.

The "Rotate" behavior is a persistent, meaning your actor can rotate indefinitely. If that is not the desired effect, you can use a "Rule" to limit when the behavior takes effect.



Angle: 0 Relative to: actor
Speed: 90 Run to completion [✓ Stops on destination]

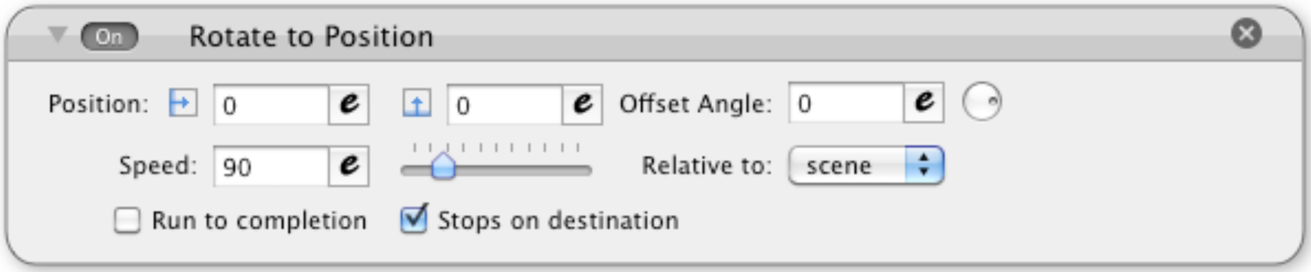
Rotate to Angle - This behavior is for rotating an actor to a specific angle. Instead of simply changing an actor's "Rotation" attribute with the "Change Attribute" behavior, the "Rotate to Angle" is more visual and more relevant for creating in-game physics. This behavior is a

"Completing Behavior". Once the "Rotate to Angle" behavior reaches its destination, as specified in the "Angle" field, the rotation stops.

The "Relative to" drop-down menu lets you choose between "actor" and "scene". If "scene" is selected, the angle will be relative to the scene. If "actor" is selected, the target angle will be the sum of the "Angle" value and the current actor rotation value.

The "Run to completion" option will continue the actor's rotation, regardless if a containing "Rule" is no longer active.

As for the "Stops on destination" option, sometimes it matters and sometimes it doesn't. Since "Rotate to Angle" is a "Completing Behavior", it will typically stop after reaching the target angle. However, if you deselect "Stops on destination" and you use an attribute that changes in value, the "Rotate to Angle" behavior can keep rotating towards the new value.


On Rotate to Position
Position: 0 0 Offset Angle: 0
Speed: 90 Relative to: scene
☐ Run to completion ☑ Stops on destination

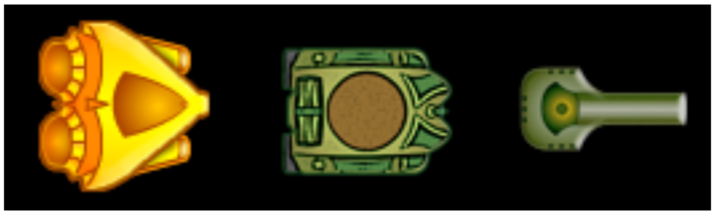
Rotate to Position - This behavior will cause the actor to turn and face a point on the screen. This could also be accomplished with the "Rotate to Angle" and an "Expression", but the "Rotate to Position" is easier to use. Enter the X and Y values for your target. You could even use attributes, such as another actor's location, the mouse position or a touch location.

The "Offset Angle" can change adjust the direction that the actor will face. For example, a value of 180 would make the actor look away from the target point. The "Relative to" option applies to the "Offset Angle".

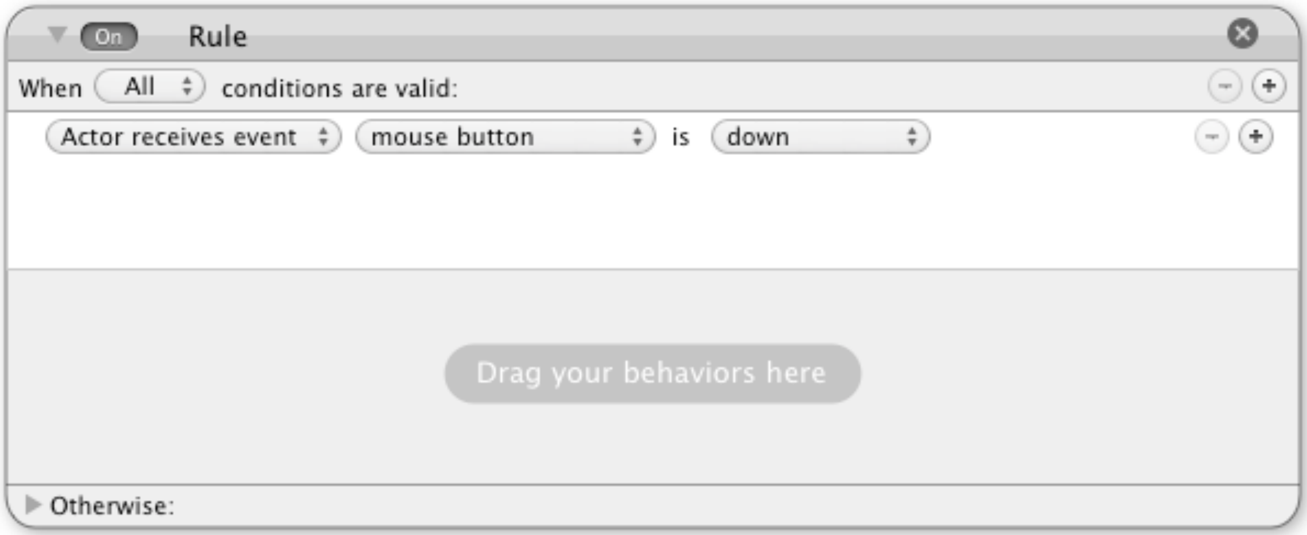
The "Speed" field allows you to control how fast the rotation occurs.

If the "Rotate to Position" behavior is used within a "Rule" container, the "Run to completion" option will allow the rotation to continue — even if the conditions satisfying the rule have ceased. If you want continuous tracking of a moving target, deselect the "Stop on destination" option.

The "Rotate to Position" behavior is phenomenal for simulating artificial intelligence. An enemy is more daunting if its constantly staring at you. This behavior also works well with touch-based controls. If you want an actor to move to a point, it's better when the actor faces that direction. By using the touch location (or mouse coordinates) as your "Position" variables, your actor will turn and face the target location.



For rotational effects to look natural, it's important to align your graphics. Your actor should probably be facing towards the 0° mark. In the image above, the three actors have images that face towards the right. This helps when rotating an actor toward an angle or a position. The nose of the spacecraft, or the point of a cannon, is like a line on a protractor. Once aligned, it becomes easier to make actors face moving targets. By using a "Rotate to Position" behavior on a tank turret, the gun can turn to face a touch point or a mouse position.



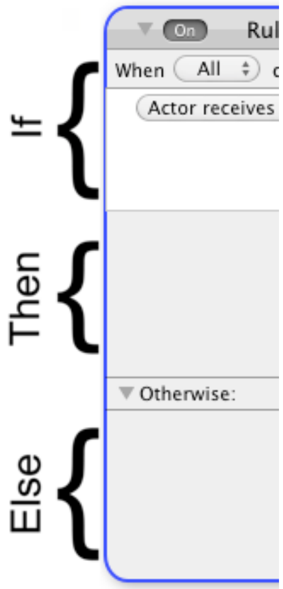
Rule

When All conditions are valid:

Actor receives event mouse button is down

Rule - This behavior is essentially a classic method of programming - If/Then/Else. When the conditions are met, the behaviors inside the main part "Rule" container will be activated. But if the conditions are not met, the behaviors in the "Otherwise" area are activated.

Similar to a "Group" or a "Timer", the "Rule" behavior is a "Behavior Container". This is signified by the little "G" icon. Other behaviors, including other rules, can go inside the "Rule" behavior.



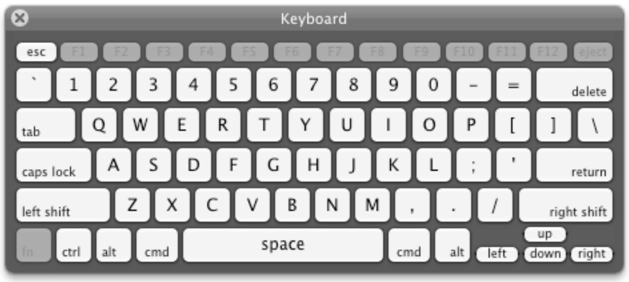
A "Rule" is useful for things like controls, collision detection or attribute related actions. For example, if an actor is touched, it could be moved or destroyed. If you wanted to destroy the actor, the "Destroy" behavior would be dragged to the "Drag your behaviors here" area. That's the "Then" area of the If/Then/Else combo.

To trigger this event, the top part of the "Rule" behavior needs information about the conditions. The first drop-down menu let you decide if "All" or "Any" of the conditions are required. You can add or remove conditions with the plus and minus icons.

The next set of options help you to determine a condition required. The "Actor Receives Event" option is mostly related to hardware, while "Attribute" is related to game data. Depending on what you select for the first drop-down menu, the latter choices can change. If "Actor receives event" is selected, your second choices are "mouse button", "overlaps or collides", "key", "mouse position", "auto-rotation" and "touch".

Mouse Button - This option is related to the state of the mouse button. (For a two-button mouse, the left-click button is one monitored.) The two options associated with "Mouse Button" are "up" and "down". This doesn't seem too complicated of a condition. It's also the default rule. If the mouse button down, do something. For game design, that alone is not very useful. However, this is an excellent starting point. It teaches two important lessons about rules. 1) Sometimes multiple conditions are required to get the effect you want. 2) The "Mouse Button" "down" (or "up") condition can also be the equivalent of a touch on an iOS device. Note: the "up" condition is not registered by default. The mouse button has to be pressed and then released before the "up" condition is recognized.

Overlaps or Collides - With the "Collide" behavior, you could make actors bounce around like balls. However, that might not be the reaction you want for overlaps or collisions. Instead, you may want an actor to explode, change color, change animation or do something besides bounce. With the "Rule" behavior and the "overlaps or collides" condition, you can be more specific with actor interactions. As with the "Collide" behavior, "Tags" can be important. You can detect for collisions based on an individual actor or actors with a specific tag. Also, with the "Otherwise" portion of the "Rule" behavior, you can make actions occur when specific actors are not touching.


Keyboard
esc F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 eject
` 1 2 3 4 5 6 7 8 9 0 - = delete
tab Q W E R T Y U I O P [ ] \
caps lock A S D F G H J K L ; ' return
left shift Z X C V B N M , . / right shift
fn ctrl alt cmd space cmd alt left up down right

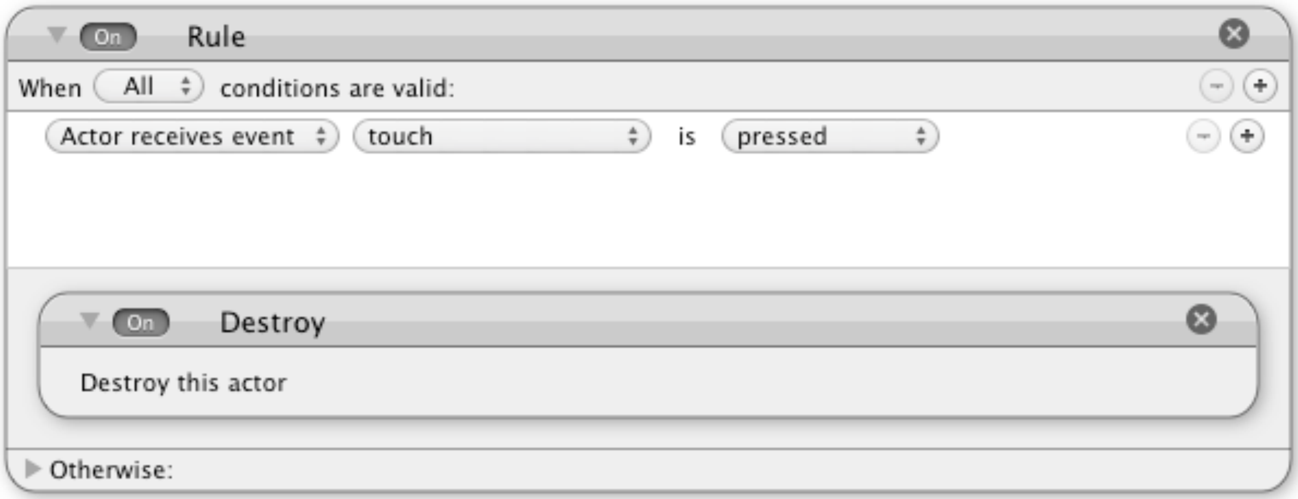
Key - This is one of the more awesome features within GameSalad. I remember how ridiculously hard it was to create game controls with other game development software. But with GameSalad, it's little more than clicking a virtual keyboard. With the "key" option selected, press the "Keyboard" button to launch the pop-up window. Pressing a virtual button will populate the "Keyboard" field. This is great for selecting an arrow key or a specific shift key. (It will distinguish between left and right.) While only one key should be assigned to the "Keyboard" field, you can use the "Any" option to make the "Rule" behavior sensitive to more than one key. For example, one condition could be for the "W" key and another could be for the up arrow key. This will allow you to add two traditional methods of keyboard based controls — at the same time — arrow keys and WASD keys.

Mouse Position - If the mouse pointer is "inside" or "outside" an actor, that status could be used as a condition for a rule. However, there are more options for the mouse. GameSalad uses a grid to organize the game scene. The on-screen mouse pointer can be tracked by its location on the grid. The values are in X (horizontal) and Y (vertical). The X and Y values can be used as game conditions. You could create conditions based on the location of the mouse position. For example, if you were making an air hockey game, you might want to restrict a player to the bottom-half of the screen. If game.Mouse.Position.Y is less than 240, then "Constrain" self.Position.Y to game.Mouse.Position.Y. Rules are a big part of GameSalad. That's why it's important to understand the "Rule" behavior and the concept of If/Then/Else.

Auto-Rotate - This is an iOS related option. If the orientation is in transition, the "auto-rotate" condition is "Started". If the rotation has ceased, the status is "Finished". This option is useful for creating effects while the screen is in the process of auto-rotating.

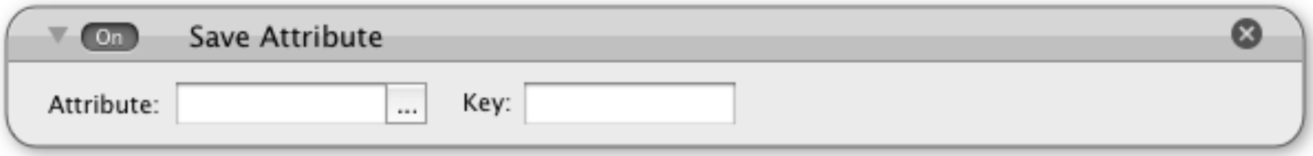
Touch - This is an iOS related option. However, a mouse click also counts as a touch. If the "pressed" option is selected, a touch must first occurs on the actor for the condition to be satisfied. This is similar to the "inside" option, except that the latter doesn't care where the touch originates. If the screen was touched "outside" the actor, the touch won't count as "pressed" if a finger drags the touch "inside" the actor. The "touch" condition can be used to create buttons and interactive elements that require the user to press the screen.

If you wanted to destroy an actor when it was pressed, you could select "touch" and is "pressed" for the conditions.



In the rule example, touching the actor will delete it. Go ahead, you can try putting this example in GameSalad for yourself. Drag the actor onto the scene, play the project and then touch the actor to destroy it. You murder, you actor killer, with your fingers of death. Actually, this is poor example. With the actor destroyed, you can’t play with it. In order to experiment with different touches, you’d need to constantly reset your game sample. For more educational value, and to reduce tedium, you can use a “Display Text” or “Rotate” behavior for when the actor is touched — instead of “Destroy”.

Practicing with “Rules” is a great way to learn about GameSalad. Throughout this book there are examples of this behavior in action.



Save Attribute - This is the companion to the “Load Attribute” behavior. Together they allow attribute data to be reused when a “Game”, “Scene” or “Actor” is reset. The first field is for choosing the “Attribute” data that you want to save. This record requires a name. You can enter a name in the “Key” field. The name is up to you, but it should be something descriptive and easy to remember. That “Key” needs to be entered in a “Load Attribute” behavior for the “Attribute” data to be reloaded.

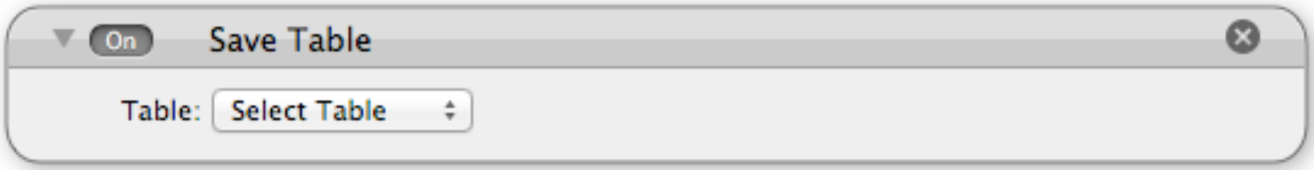
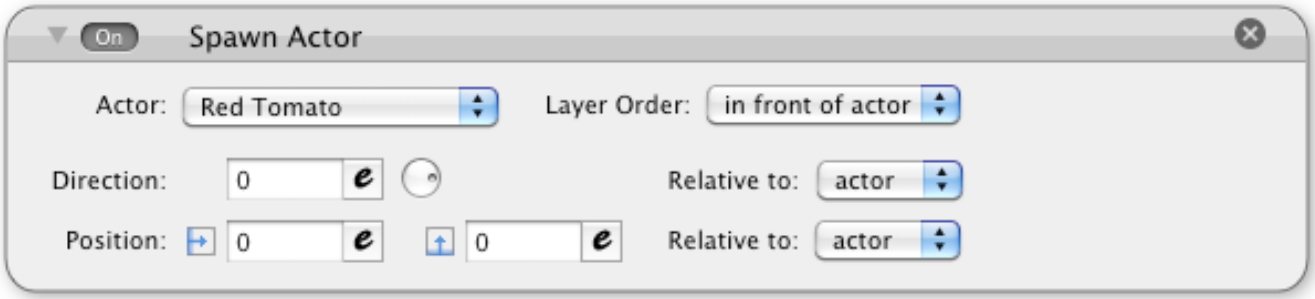


Table: Select Table

Save Table - This is a surprisingly dangerous behavior. Once you save a table, that's it. The previous data is overwritten — even if you reset the game. So, if a player wants to start from the beginning of the game, how can they do that? The trick is to copy tables. Treat one table as your source. The copy of the table is for working. That way, if you ever need to revert to the original data set, you can just recopy the source table. When you're ready to save a table, this behavior is easy to use. Simply select your table from the "Table" option menu. This behavior is best used in a "Rule", when a player is at a safe point to save their progress.



Actor: Red Tomato
Layer Order: in front of actor
Direction: 0
Relative to: actor
Position: 0 0
Relative to: actor

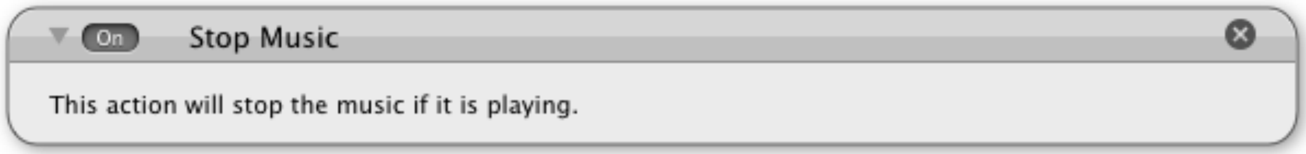
Spawn Actor - This behavior will add a new actor into an active scene. It is useful for creating enemies, projectiles, alert messages and plenty of other game elements. There are two basic ingredients in a GameSalad game... "Scenes" and "Actors". So obviously, the usefulness of this behavior is huge. Yet, use it with caution. If you try to spawn too many actors at once, it may cause performance issues.

The "Actor" drop-down menu is populated with a list of the actors in your project. The "Spawn Actor" behavior can only add actors that already exist in your project. However, these actors don't need to be on the screen already. That's what makes this feature handy. By combining "Spawn Actor" with a "Rule", you could fire a bullet by pressing the spacebar.

That's where the other options enter the picture. If you want your spaceship actor to fire a bullet, you then have to decide where that bullet should enter the screen. The "Layer Order" determines if the bullet should be "in front of actor" (on top of the spaceship), "in back of actor" (underneath the spaceship), "front of layer" (on top of all of the actor in the same layer) or "back of layer" (underneath all of the actors in the same layer).

The “Direction” is important, as you probably don’t want a bullet to shoot towards the right when your spaceship is facing towards the left. Frequently, the “Expression Editor” is used to align the current actor with the actor being spawned. The attribute self.Rotation is common for spawning actors. That will give the new actor the same angle as the spawning actor. In such situations, the angle would be “Relative to” the “scene”.

Of course, your spaceship doesn’t have to be about violence. You might want to spawn a cool rocket thruster actor. While this is normally done with “Particles”, you could use an actor to accomplish a similar — perhaps better — effect. Using actors instead of particles might make sense if you wanted your rocket thrusters to burn enemies. (Particles don’t use collision detection.) Ah, more violence! The point is that you have to enter the proper “Direction” and “Position” information into the “Spawn Actor” behavior. For many applications, this behavior is pretty straightforward. Yet, for rotating actors, it can get complicated rather quickly. For more details, read about rocket thrusters in Chapter #8 - Common Game Elements.

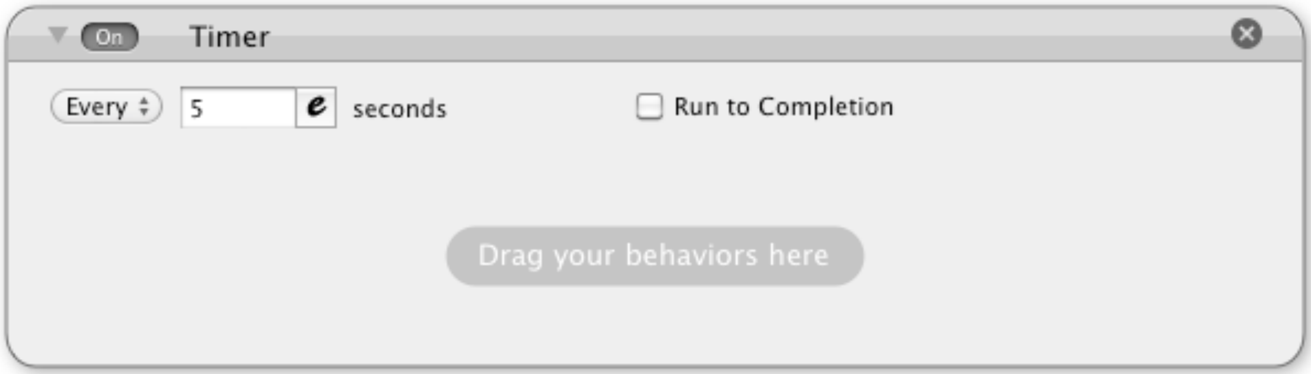


This action will stop the music if it is playing.

Stop Music - This behavior is similar to the “Pause Music” behavior. But instead of pausing the music, the “Stop Music” behavior stops the music. So… err… what’s the difference? After testing the differences between these two behaviors, I think it’s probably better to use the “Pause Music” behavior instead of the “Stop Music” behavior . When you pause the music, you can use the “Resume Current Music” option. (That’s located in the “Play Music” behavior.) If you stop the music, there’s nothing to resume.

I tried to think of scenarios where one would want to stop the music, instead of pausing it, but I couldn’t think of any. If you want silence, pause the music. If you want to restart the song from the beginning, simply use the “Play Music” behavior and select the song that should be played. The only thing I could think of is performance. If you’re sure that you will not be resuming the music, then it might slightly better to use the “Stop Music” behavior. Otherwise, if no memory issues are present, I recommend using “Pause Music” instead of “Stop Music”.

GameSalad will not play two music files at the same time. Simply playing a new music file will stop the previous music file from playing.



Timer - This is a “Behavior Container”. When an actor is in an active scene, the behaviors inside that actor will run immediately. However, there are two ways to reserve a behavior for later use. The first is with a “Rule”. The behaviors inside the container will not fire until the conditions have been met. The “Timer” behavior works in a similar manner, except that the conditions are in relation to time.

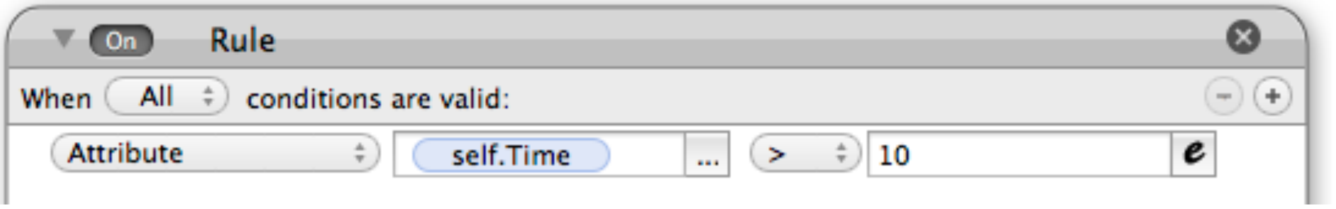
There are three main options for the “Timer” behavior.

- **Every** - This option is a behavior repeater. Whatever behaviors are in the container, they will be activated at specific intervals. The “seconds” field is where you can specify how often the behavior should be repeated. However, the activation is only for a split second. “Persistent Behaviors” will not work well with the “Every” option. Instead, this setting is better for “Action” based behaviors.
- **After** - This option will trigger the contained behaviors after a specific amount of time has passed. After that point, the enclosed behaviors will function normally.
- **For** - This option will start behaviors immediately, but then cease after a certain amount of time has passed.

If the “Timer” behavior is contained by another “Behavior Container”, the “Run to Completion” option functions like an override. It will allow the “Timer” behavior to complete it’s function. However, this option only works for the “After” and “For” option. “Run to Completion” does not work with the “Every” option. Instead, many behaviors have their own “Run to Completion” option. When applicable, such a setting could be used inside a “Timer”.

“Timers” are ideal for using behaviors at specific points in your game. For example, if you fired a real missile, it would only fly so far. After a certain time, it would run out of fuel and crash back down to the ground. You could simulate that with a “Timer” and an “Accelerate” behavior. "For" 10 seconds, accelerate the missile upwards. "After" 10 seconds, "Rotate" the missile downwards.

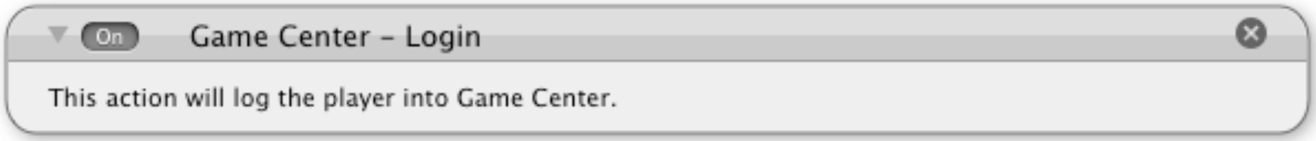
Although, it's not always necessary to use a "Timer". Every actors has their own internal clock. It's the self.Time attribute. As soon as the actor is created, the timer starts.

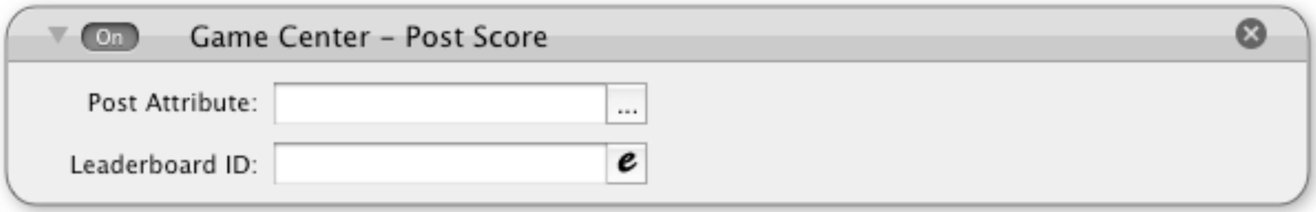

On Rule
When All conditions are valid:
Attribute self.Time ... > 10

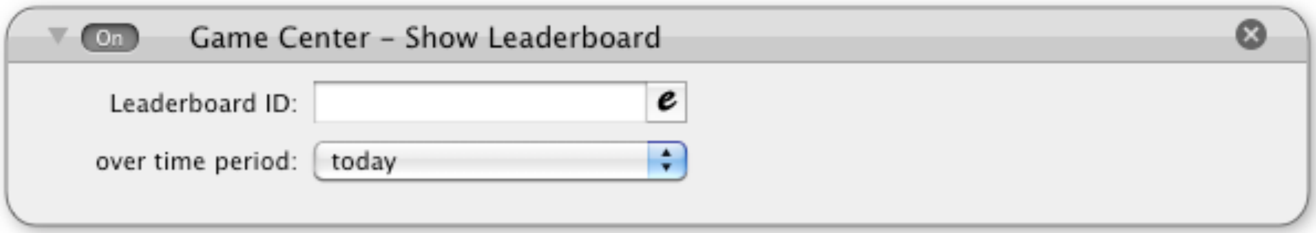
The above example is alternate way for destroying projectile actors. If a bullet is fired, it can be destroyed soon after it is launched. A "Rule" can be used to monitor an actor's self.Time value.

Professional Behaviors

"Look at you all fancy, spending all that money on GameSalad game development." If you upgraded your GameSalad account to "Professional", you'll gain access to additional behaviors.


On Game Center – Login
This action will log the player into Game Center.


On Game Center – Post Score
Post Attribute: ...
Leaderboard ID:


On Game Center – Show Leaderboard
Leaderboard ID:
over time period: today

Game Center - Adding a Game Center Leaderboard to your game can be a little tricky. It doesn't just involve GameSalad. It also involves iTunes Connect. First you'll have to log into

Apple’s system and create a Game Center Leaderboard. (If you don’t already have a listing for your app, you’ll have to create one.) The key thing to remember is the ID — a unique name for your leaderboard. As an example, I used com.photics.bot.gold for ranking players in “BOT” on the “Gold” leaderboard. With that setup, I could start working in GameSalad.

But even with the proper ID for your leaderboard, there are some important things to decide. Do you want the players to automatically login? What if their iOS device doesn’t support Game Center? If so, you’ve just given them an annoying alert screen to dismiss — every time the app is launched. So, you might want to add a Game Center login button to your game. You also might want to set the “Minimum Supported iOS Version” to 4.1 - the first version of iOS with Game Center. (That can be accomplished in the “Advanced Platform Settings” in the publishing area.) Limiting your app to iOS 4.1 or higher will reduce the amount of potential customers. That’s why a Game Center toggle is a good solution. Let the player decide if they want to enable or disable Game Center features.

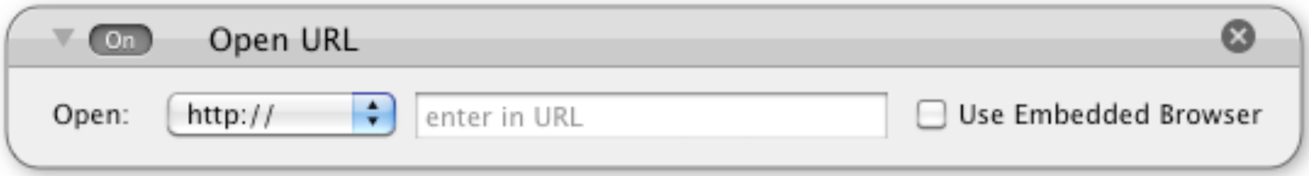
You could also add a “Post Score” button. By using just the "Post Score" behavior, you can combine the login and the submit score into one behavior. If the “Post Score” behavior is used when the player is not logged in, it will popup a login alert. This is not such an elegant method of sending scores to Game Center. To create a more seamless experience, first login the player and then submit the score.

Warning - Don’t try to launch all your Game Center behaviors at once, or they might misfire. Instead, try using buttons and/or spread out the Game Center behaviors throughout the game. A login button at the title screen makes sense, and then add “Post Score” buttons where appropriate. If you want to submit two or more scores, there should be some time between the activation of the behaviors. You can have up to 25 leaderboards per game.

Game Center achievements work in a similar fashion to leaderboards. Throughout your game, you can set goals for players to pursue. This can increase the replay value of your game.

Obviously this is something that should be tested. And unfortunately, that requires some extra effort. Game Center needs to be tested in sandbox mode. To activate this, you will need an ad-hoc version of your game and your iOS device should not be jailbroken. If you can get that working, it’s a better testing environment for Game Center features.

To encourage competition, you might want to add a “Show Leaderboard” button to your game. This will let players access the top scores from within your app.



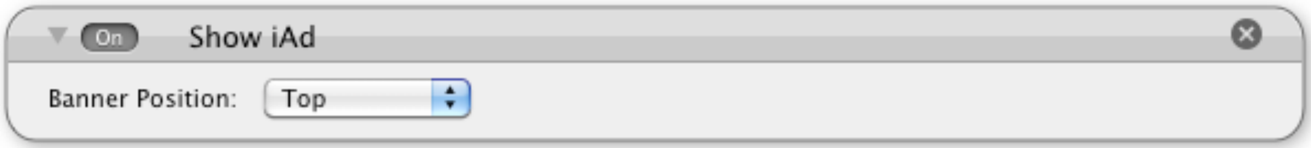
Open URL - If you want to open a web page, you can simply cut and paste the link into the URL field. This behavior is typically used as part of a rule — turning an actor into a hyperlink. However, it's not limited to just web addresses. It can also be used for email addresses.

If you're serious about customer service, you can use the "Open URL" behavior for providing help to your customers. However, you're a game developer. Think beyond that. You could create amazing game experiences with this option. For example, you could create a mystery game. In this adventure, the main character finds a computer. The character clicks a button on the screen to contact a secret character. The player actually has to sent an email to receive a solution to the puzzle. You could setup an email auto responder to give out a password or clue — bringing the real world into the game world.

Do you want to throw a contest? Maybe you want to give a prize to the first person that beats your game. How can you verify such a feat? You could hide an email address in the game. When a player clears the game, they are given access to a button — a way to contact you. It's not 100% immune to cheating, but it could be a nice touch for honest players. To help prevent cheating, a challenge question could be used — to verify that your game was actually played.

If your website has a contact form, you could also place that into your game. That's what the "Use Embedded Browser" option is about. Instead of launching a new application, your players can stay inside your game.

Another option for the "Open URL" behavior is to encourage reviews. By linking to your app listing, you can motivate players to rate your game. Don't be too annoying about it though, as that might result in bad reviews.



Banner Position: Top

Show iAd - Apple has implemented their own advertisement system for apps. What's really nice about iAds is that some truly posh and powerful companies are participating. These aren't your typical website advertisements. No, these are like apps within an apps. A player can click an iAd and still stay inside your game. The banner ad expands to reveal additional information, while keeping your app open.

But the real reason to use iAds — money! It's an additional revenue stream. Will you get rich with iAds in your games? Just like a paid app, your game has to be really good to be incredibly successful. Otherwise, you will likely be crushed by the competition. When using iAds, think of how that changes your game design. You don't want to be too annoying, but you don't want to be too timid either. With iAds, you're supposed to make money when an advertisement is shown or when it is clicked.

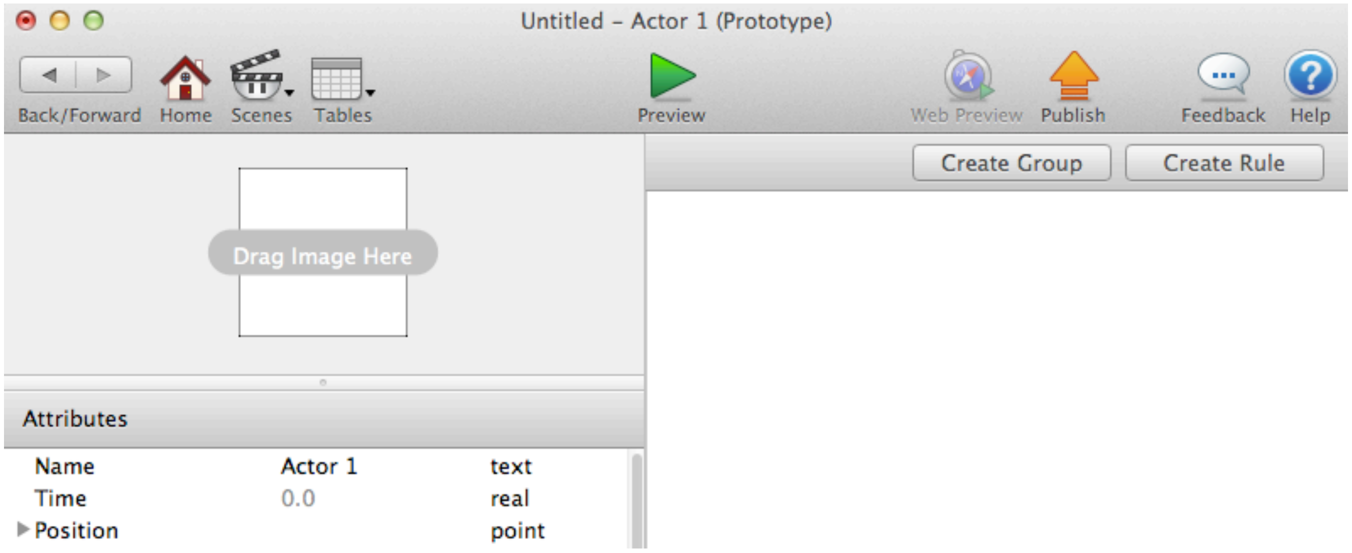
To use this behavior, remember to leave space in your game for the advertisements to appear. Although, it can't be a blank space, as an iAd might not appear. If Apple is low on advertisements to display, an ad might not show up at all. If you don't account for the missing space, your app may be rejected. If your game is in portrait mode, the space is 320x50 pixels. If your game is in landscape mode, the space is 480x32 pixels. By using the Banner Position option you can specify if the advertisements should appear at the top or the bottom of your app. The GameSalad team did a really nice job with the implementation of iAds. Simply place the "Show iAd" behavior into any active actor. That's it. Your game is ready to show iAds. GameSalad even takes care of orientation support. As the phone is rotated, the iAd will adapt to the new scene orientation.

There are some important thing to remember. While your game is ready for iAds, you still have to enable it in the iTunes Connect. After you create your app listing — but before your app is approved — you can enable support for iAds. Don't use this behavior more than once at a time. Also, don't go clicking your apps fraudulently. If Apple suspects cheating, they probably won't pay you.

While these behaviors may not look like much, they can help make your app more profitable. For more information, see Chapter #26 - Promoting Your Games.

Back to the Basics

The behaviors are your programming arsenal. These weapons are fired by your actor army. With a better understanding of behaviors, we can take a closer look at actors. Double-clicking an actor will bring up the "Actor Editor". The left side of this screen is similar to the "Scene Editor", as it contains the "Media Panel". From the first time you open the "Actor Editor", you can see how GameSalad works. You drag and drop game elements to make things happen.



Name	Actor 1	text
Time	0.0	real
▶ Position		point

By dragging an image from your desktop, onto the "Drag Image Here" box, you can associate an image with an actor. The image will also be stored in the "Images" tab, so that other actors can access the file too.

There are plenty of cases where you do not need an image. For example, the "Display Text" behavior may work better with just text and no images or color at all.

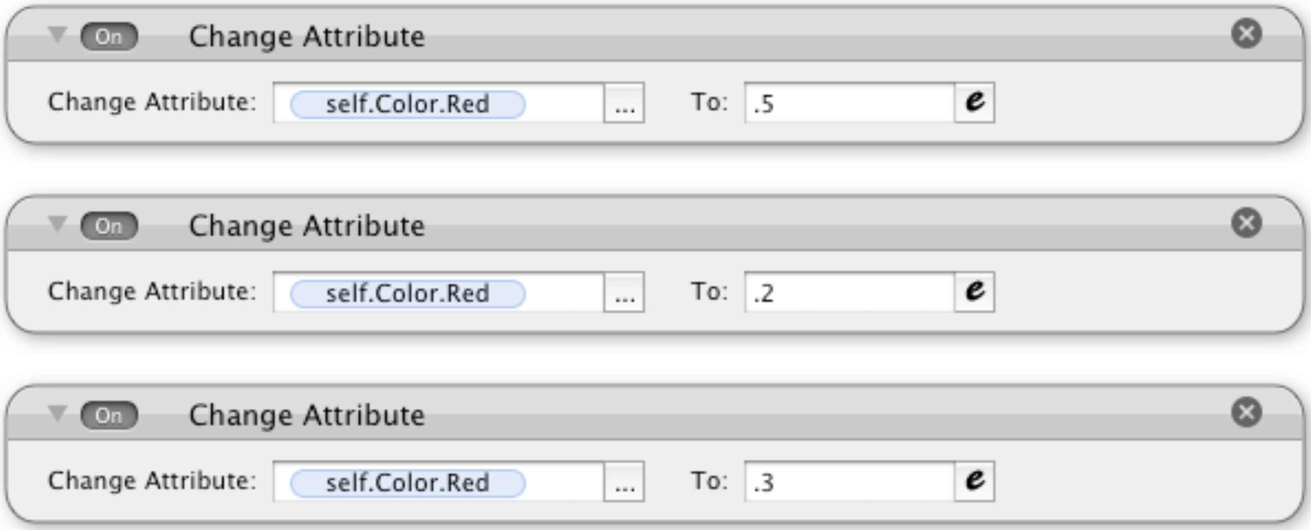
If you decide that your actor needs an image, GameSalad accepts many different formats. You can import bmp, eps, gif, jpg, psd and png. There's only one format that matters though. That format is png. Regardless of what format GameSalad accepts, it's only going to convert it to png. For greater control over your files, I recommend making the conversions yourself. GameSalad accepts more file formats than the ones listed here, but none of that really matters. The only graphic format you need for GameSalad actors is png. For more information on this topic, see Chapter #11 - Graphics.

Actors start with their own default set of "Attributes". The "Behaviors" used in actors frequently modify this data. If you want an actor to "Move", the "Position" attributes are changed. If you want an actor to "Change Size", the "Size" attributes are changed. If you want an actor to "Rotate", the "Rotation" attribute is changed.

"Color" is also import with an actor. GameSalad uses the RGBA color space. That stands for Red, Green, Blue and Alpha. An actor starts out as a white box, but the RGB colors can be modified to create other colors. By modifying an actor's "Alpha" channel, an actor can be made transparent or even invisible. More information about "Color" and "Graphics" settings can be found in Chapter #11. "Motion" and "Physics" are covered in Chapter #6.

With that information, you should be ready for the right side of the “Actor Editor” screen. There are two buttons on the right side. The “Create Group” and “Create Rule” buttons are just quick ways to create attributes. Dragging “Behaviors” from the “Media Panel” has the same basic effect. Although, the buttons do have a handy feature, You can highlight already placed “Behaviors” by clicking on them. (To select multiple “Behaviors”, hold the shift key.) Once you have highlighted the behaviors, you can press the “Create Group” or “Create Rule” to put the selected “Behaviors” in a new container of your choosing.

Before you start to drag-and-drop like a ninja, there’s something you should know. This is important. Behaviors fire in order. The “Behaviors” closest to the top fire first. So, here’s a quick quiz to test your reading comprehension skills.



Change Attribute

Change Attribute: self.Color.Red To: .5

Change Attribute

Change Attribute: self.Color.Red To: .2

Change Attribute

Change Attribute: self.Color.Red To: .3

The above image is a screenshot from the GameSalad “Actor Editor” screen. If this actor was placed on an active “Scene”, what would be the final value for the self.Color.Red attribute?

A) .5 B) .2 C) .3 D) 1.0 E) 0.0

I don’t want your eyes to accidentally glance over the solution to this problem. So, here’s another cool tip!

Cool Tip

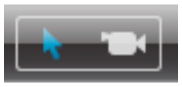
Quick Copy & Paste - If you’ve made modifications to a “Behavior” that you’d like to copy, you can use the “Copy” and “Paste” options in the “Edit” section of the menu bar. But if you want a faster approach from within an actor — or actors in other GameSalad projects — you can click and drag an existing “Behavior” (or multiple “Behaviors”) while holding the alt key. A green plus icon will appear on your mouse pointer. Keep holding the alt key while you move the transparent image of your “Behavior”. When you reach the new location, let go of the mouse button. A copy is created in the new location and then you can let go of the alt key.

The answer was “C”. The “Change Attribute” behavior is not cumulative. With each new occurrence, the value is changed. GameSalad reads an actor’s “Behaviors” like the pages in this book. The order is from top to bottom. That’s an important concept for working with GameSalad. That’s how you do your programming.

Every “Behavior” has an “X” icon at the top-right. That’s how you can delete “Behaviors”. That doesn’t seem too difficult. It’s certainly not as dangerous as deleting “Attributes”. If you make a mistake, and you delete the wrong “Behavior”, you can simply bring it back with the “Undo” command from the “Edit” section of the menu bar. Although, if you hit the “Preview” button, the option to undo mistakes is lost.

This is almost it for the basics of GameSalad, but a closer look at the “Scene Editor” is needed first. This window is not just a place to drop “Actors”. It’s also an excellent way to test and edit your games — LIVE!

There are five icons, in three groups, at the top of the main area.



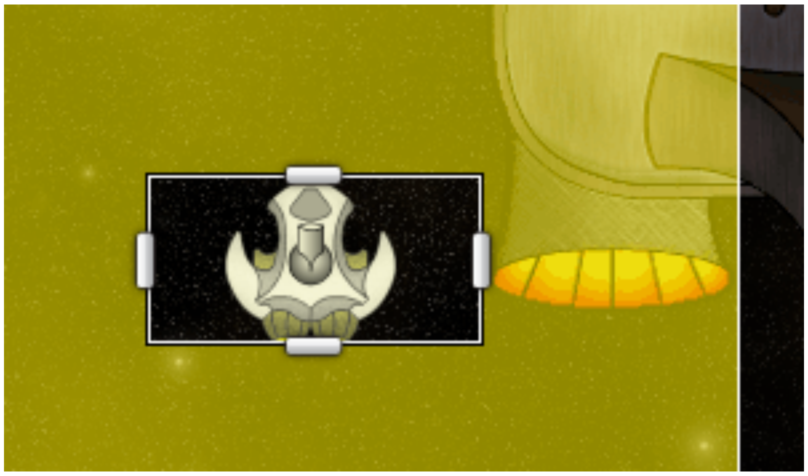
There’s a set of icons towards the left side of the “Scene Editor”. These icons are for altering the “Camera” settings. This controls the player’s perspective — and there can only be one camera in the game. The arrow icon is for moving actors and the camera icon is for moving the camera. If your game area is not larger than the screen size of your target device, the camera icon will pretty much do nothing… NOTHING! So, go out there and set a larger scene size!

From the “Inspector” pane, click the “Scene” button. From there, select the “Attributes” tab. As described earlier, there is a difference between “Scene” attributes and “Game” attributes. The “Scene” attributes are restricted to the current scene. For example, if you change the “Size” here, it does not change the size of another scene. Once you have the “Attributes” tab highlighted, you should be able to find the values for “Size”. Click the dark arrow down to show the contents. If you picked “iPhone Landscape” as your target platform, the default

“Width” should be 480 and the default “Height” should be 320. The numbers are pixel counts. The original iPhone, in landscape mode, has 480 pixels horizontally and 320 pixels vertically.

That’s a great size, if you don’t want your game to scroll. If you want your actors to explore and discover, then your playing area needs to be larger. For example, if you were building a platformer, your actor might run towards the right of the screen. You might create a “Width” of 8000 and a “Height” of 600. When you change these values, you should be able to see it happen in the “Scene”. If you watched very carefully, you might have noticed a white box. It didn’t move. Instead, the scene expanded to the right and the top of the box. That box is the representation of the camera’s view. With the camera icon selected, you can move that box around the scene.

In many cases, you might not need to position the camera. Just use the “Control Camera” behavior on an actor. The camera will follow the actor that most recently used the “Control Camera” behavior.



The camera box is actually two boxes. It’s a box within a box. The inside box is for creating some tolerance. To have the camera move in a less jostling manner, expand the “Tracking Area” to create some leeway. By default, the camera sensitivity is actually pretty loose. If you’re using camera to follow an actor, you might want to experiment with different sizes for the “Tracking Area”. If the camera is set to constantly follow an actor, the quick camera movements might give the player an uneasy feeling. If the setting is too loose, the game might feel slow and less exciting.

You could think of the camera as a square donut. Your actor stays inside the hole. The donut is the camera. When the actor pushes up against the inside wall of the donut, the donut moves. OK, maybe that’s not the best of analogies, but it does make me hungry! If the donut analogy doesn't help you to visualize what's going on, the next set of "Scene Editor" icons can show you exactly what's going on.

There's a big and green icon in the main tool bar of GameSalad. The mighty "Preview" button hogs all the glory. Yet, there's another way to test your GameSalad game. By pressing the play icon, you can edit your GameSalad games in a real-time setting. When you're done previewing your game in this mode, you can use the stop icon. That will return the "Scene Editor" to normal mode. The stop icon will reset the scene. If you want to stop the action, but not reset the scene, you could use the pause icon.

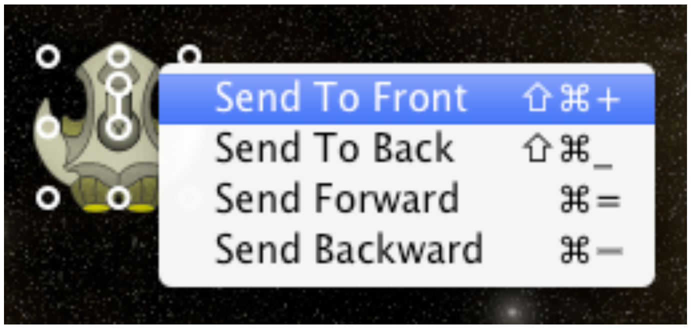
Use of this feature is an excellent way to visualize actions in your game. This is especially true with the camera. You can watch the camera follow a moving actor, or you can even adjust the size of the "Tracking Area", while the game is in play.

If you don't like the location of an actor, you can move it — LIVE! Personally, I don't use this feature a lot. Yet, the live preview mode might help you to visualize your game and to better position your actors.

Why wouldn't I use such a cool feature? There are some limitations. Primarily, it's hard to control an actor in this mode. If an actor is selected, pressing an arrow key will move the selected actor. This is especially confusing when your game's controls are mapped to the arrow keys. The same problem applies for touch games. If you try clicking the playing area with the mouse, you might select the background while trying to move your actor. This mode is also terrible for screenshots, as the camera outline taints the view.

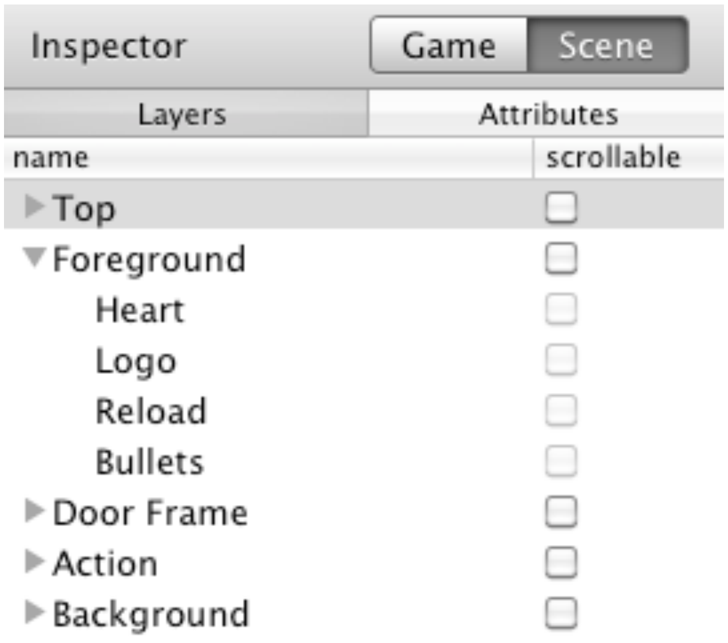
To help work around this issue, you could map additional key setting to your controls, such as the WASD keys for an alternative to the arrow keys.

The icon on the right side is for toggling the "Show Initial State" option. If you are using the live editing mode, the "Show Initial State" option will display transparent images for actors that moved or that are no longer in the scene. This is useful for knowing where to click. That is how you can select and edit an actor in live editing mode. GameSalad allows you to reposition, rotate and resize actors while the live editing mode is activated. (Clicking the moving actor in live editing mode is not the same as clicking the original actor position.) There is an alternative to the "Show Initial State" option. By highlighting the "Layers" tab during live editing, you can click the names of the actors in the tab. That will also highlight the actors in the scene window.



When you click an actor that has been placed in a scene, a box-like image should appear. The image contains 10 circles. Each of these little circles is a click points. Depending upon which one you use, an actor can be moved, scaled or rotated. If you click-and-drag the center circle, or anywhere on the actor that's not covered by a circle, you can change the actor's position. If you click an outer circle, either on the corners or the edges, you can change the actor's size. (Hold the shift key to scale the height and width of the actor proportionally.) If you click the circle that's connected by a line to middle circle, you can change the actor's angle. (While rotating an actor, holding the shift key will rotate the actor at 45° intervals.)

If you want more precise measurements, or your actor is too small to grab properly, you can double click the actor to edit it directly. You can enter the "Position", "Size" and "Rotation" values manually. If you're having trouble clicking on an actor, you can select the actor from the "Layers" tab.



name	scrollable
▶ Top	☐
▼ Foreground	☐
Heart	☐
Logo	☐
Reload	☐
Bullets	☐
▶ Door Frame	☐
▶ Action	☐
▶ Background	☐

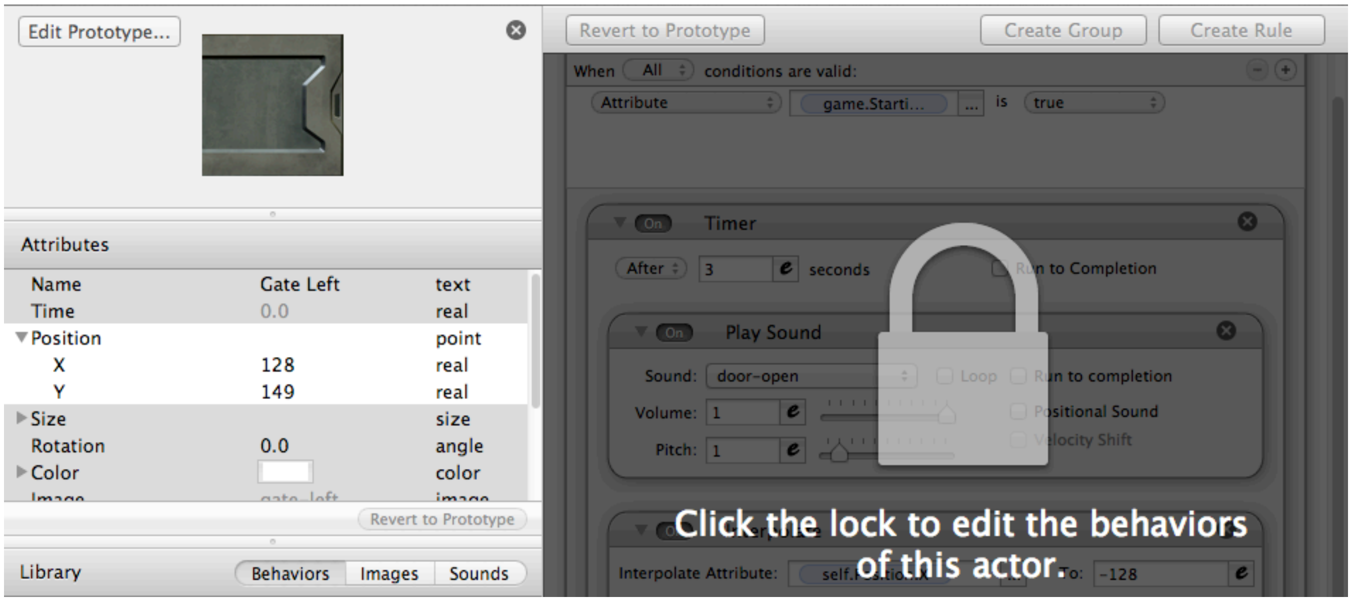
Like a deck of playing cards, actors can be stacked on top of each other. With the "Layers" option, you can control the order. By accessing the "Inspector" pane, press the "Scene" button. Once that has been selected, you should be able to see the "Layers" tab. Each actor is displayed in order. The actors on the top of the pile are near the top of the list. By clicking and dragging a name, you can arrange the order of your actors. You can also arrange actors (within a single layer) from the scene itself. Right-click or control+click an actor to bring up a menu. There are four options in this menu.

- **Send To Front** - This will send the actor to the top of the layer in which it is contained. It will not move above its current layer.
- **Send To Back** - This will send the actor to the bottom of the layer in which it is contained. It will not move below its current layer.
- **Send Forward** - This will move the actor up one space, but not above its current layer.
- **Send Backwards** - This will move the actor down one space, but not below its current layer.

The "scrollable" check box is for creating a HUD (Heads-Up Display). If the check box is not checked, the scene will not scroll. That's perfect for things like controllers, scoring displays, health bars and interface elements. Their position stays constant, while the main actor (the one controlling the camera) moves around the playing area. Only a layer can use the "scrollable" option. If you do not want an actor to scroll, it should be placed in a non-scrollable layer. Like with "Actors" and "Scenes", you can make a new layer with the plus icon.

If you want to delete a layer, the actors must be cleared out of the layer first. When all of the actors have been deleted, or moved to another layer, you can delete the empty layer. You can also rename a layer. Click the name once. It should be highlighted with a gray color. Click it again. This time it should be highlighted in blue, or whatever "Highlight Color" is specified in you Mac OS "Appearance" settings. Wait about a second and then click the name again. That sequence should enable layer renaming.

When you edit an actor in a scene, the changes you make are only on that actor — not the actor's "Prototype". An actor on a scene can have unique features. But in doing so, the link to the "Prototype" actor is weakened. If you double-click an actor on a scene, you should see a giant lock.



Attributes

Name	Gate Left	text
Time	0.0	real
▼Position		point
X	128	real
Y	149	real
▶Size		size
Rotation	0.0	angle
▶Color		color
Image	gate_left	image

When All conditions are valid:
Attribute game.Starti... is true

Timer
After 3 seconds Run to Completion

Play Sound
Sound: door-open Loop Run to completion
Volume: 1 Positional Sound
Pitch: 1 Velocity Shift

Interpolate Attribute: self.Position.X To: -128

Click the lock to make individual changes. The “Prototype” will not be affected by these changes. So, why make an actor unique? Why not just use actors that are completely linked to the “Prototype” actor? One good reason has to do with the “Attribute Browser”. The “Current Scene” attributes won’t appear in the “Attribute Browser”, unless an actor on the scene is being edited. If you want to access things like “Gravity”, “Camera” settings or a scene’s background color, you might need to edit the actor directly. While many of the scene attributes cannot be changed during game play, there are plenty that can. Even if the data cannot be edited, that data can still be read.

COMMON MISTAKES!

Attributes will not work if they’re just typed into a behavior. Typing scene.Time is not the same as selecting and entering scene.Time from the “Attribute Browser”. Additionally, GameSalad doesn’t permit cutting-and-pasting of scene attributes into “Prototype” actors.

Overall, the GameSalad interface is pretty intuitive. Before you start building your first GameSalad game, you might want to play with the different “Attributes”, “Behaviors” and GameSalad options. By watching how things run, you might be inspired and your understanding of GameSalad should improve. That’s the beauty of GameSalad. With traditional game development methods, a wall of intimidating text would separate you from design. GameSalad lowers the wall.

This chapter was written with two goals — present a detailed overview of the GameSalad software, while also being organized as useful reference guide. Even if you don't understand how all of the "Behaviors" work right now, that's OK. The idea is that you can revisit this section later. Through a combination of practice and learning, you can become a GameSalad master.

Chapter #4 Summary

- GameSalad has an interface that is similar to a web browser.
- The bulk of GameSalad game development is related to the creation of "Actors" and "Scenes". Both can be created from the "Project Editor" screen.
- "Attributes" store and record data. You can create six different types - "Boolean", "Text", "Integer", "Real", "Angle" and "Index".
- There are three areas for the creation of "Attributes" - "Game", "Scene" and "Actor". That order is an important part of working with GameSalad. "Actors" go in "Scenes" and "Scenes" are part of the "Game".
- By adding "Behaviors" to your "Actors", you can make things happen in your game.
- "Actors" in a "Scene" are stacked. The order is controlled by "Layers".
- When an "Actor" is placed on a scene, it becomes a copy that is based on the "Prototype". When you edit an "Actor" directly on a "Scene", the related links between the "Actor" and the "Prototype" are severed.

Chapter #5 - Math

Back in high school, I was becoming increasingly bored with Math class. With each year, the lessons seemed to become more and more abstract. Looking back, the reason is obvious. The teachers were paid to teach academics. If you don't know what that word means, look it up. The discovery of its true meaning was a bit of a shock to me. It basically means hypothetical information — topics that are not immediately applicable to the real world.

That seemed pretty useless. How was cosine, sine and tangent supposed to help me find a job? I remember enormous pressure to determine what I wanted to be in life. Mathematician was not an option. As math class grew more and more alien, I became more and more frustrated. One day, the teacher noticed that I found her class incredibly dull. I spent most of the time drawing on my notebook. She decided to point out my lack of interest in calculus as a matter for the whole class. Since it was many years ago, I don't remember the exact dialogue. I do remember one thing vividly. I remember asking, "When am I going to use this stuff?"

The teacher was unable to answer.

That was her one moment to inspire me, to show me the true value of math. She failed and I failed the class. It was the only class that I ever failed in my entire life, but that's usually what happens when you stop going. Not only is it hard to learn advanced mathematics, it must be hard subject to teach. That's why it's not uncommon to see math teachers drudging through the topic like robots.

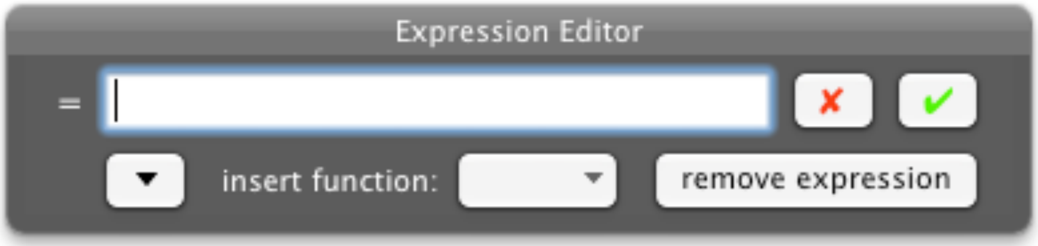
Now, decades later, it is my turn to inspire you. I must show you the value of math. Will I fail? No! It's because I have something that my math teacher did not. I have GameSalad.

You like money, right?! Calculus, trigonometry and arithmetic are tools. You can use them in your quest for the treasures of the iTunes App Store and other markets for your games. Even if you're altruistic, even if the pursuit of coin is not what motivates you, mathematics is often at the core of a good video game.

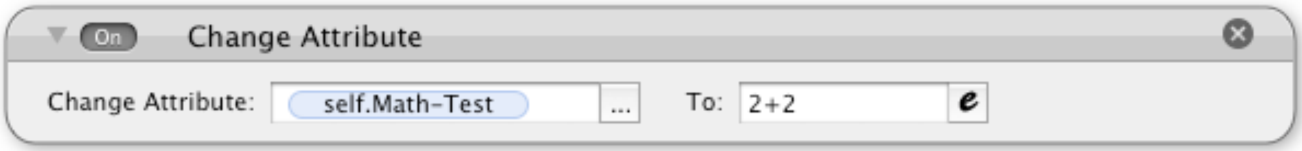
Games… that was the one word that the teacher could have used. "Dude, you like games right? If you want to make them, then you'll need this information!" Suddenly, trigonometry and calculus are not so dull, not when they're bringing computer games to life. Artificial intelligence, scoring and touch controls are all examples of math in video games.

Fearing such a challenging topic, other writers might have buried this topic deep in the book. Not here! I put it at chapter #5. Why would I do such a thing? It's because you can save yourself a lot of headache and you can build better games with this information. A strong foundation is important. Math, logic and physics, they're like the steel and concrete for a good video game

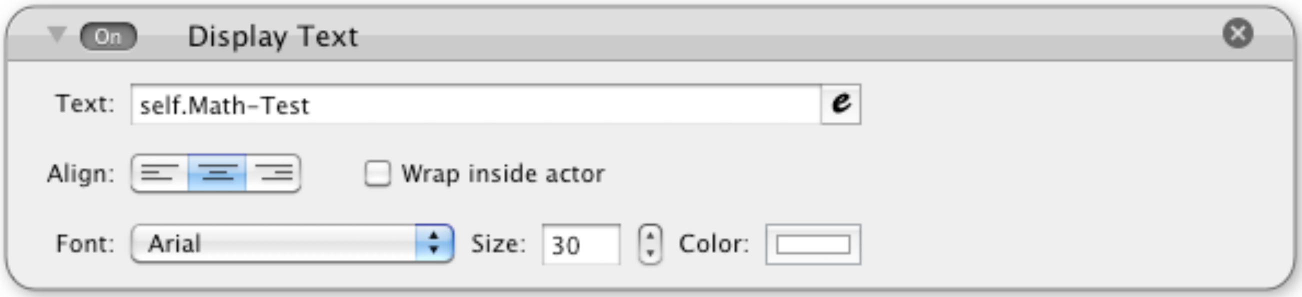
So OK, enough hype. Instead, take a closer look at your nemesis… the "Expression Editor".



There's a bit of a love / hate relationship with this box. It lets you add some amazing things to your game, but it can also cause tremendous problems. Similar to traditional programming methods, if you mess up just one character, your entire game could crash. So, we're going to start off with something nice and easy… 2+2.



Change Attribute: self.Math-Test To: 2+2



Text: self.Math-Test
Align: center
Wrap inside actor: ☐
Font: Arial Size: 30 Color:

The "Expression Editor" accepts basic mathematical functions. Addition, subtraction, multiplication and division. The above example shows addition. If you enter "2+2" in the "Expression Editor", the result is four. The self.Math-Test is an "Attribute". It has been set to "Real", as we're working with numbers — not text! The "Change Attribute" behavior is going to take the solution of 2+2 and place that as the value of self.Math-Test.

The "Display Text" behavior will show the results. (A new actor and displayed text is white by default, so you might need to adjust the color settings.) The behaviors are run in order — top first. First the self.Math-Test attribute is changed to the result of 2+2 and then that data is displayed on the screen as text.

If you run through this little math example, you can see that the answer is four. If you can accomplish this, you have proof-of-concept. GameSalad actors can be like little calculators floating through your game.

Like buttons on your calculator, you can enter special math functions in the "Expression Editor". Specific characters will triggers those functions.

Function	**Purpose**
+	Addition
-	Subtraction
/	Division
*	Multiplication
^	Exponents
%	Modulo

The first four are basic math. The symbols are similar to ones on the Mac OS calculator. The only difference is with division. A slash is used instead of the traditional division sign. If you try using the division sign in the "Expression Editor", like 4÷2, it will not function properly. So, use a slash for division.

If you want to use exponentiation, you can use the ^ character. That's shift+6, also know as a caret, a circumflex accent, or the little arrow thingy. Basically, it's an easier way to write a superscript. 4^2 is how you would write 4^2 (pronounced: four squared) in GameSalad.

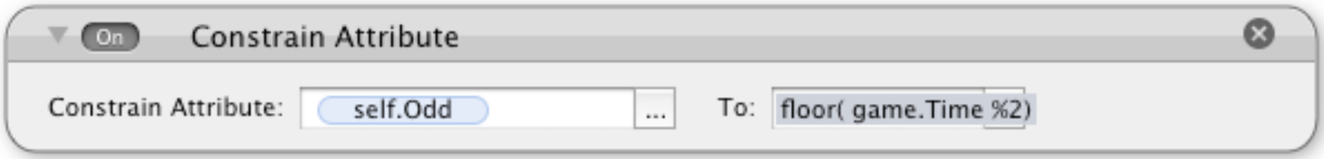
4^2 is the same as 4*4
4^3 is the same as 4*4*4

Is math class coming back to you?! Oh no… night sweats… the terror… the agony! Stay calm friend, it's going to get worse. If you wanted the square root of a number, you could use the Exponent function, but use a factor of .5 instead of a whole number. 9^.5 = 3. The square root of nine equals three.

So, before I go further, why should you care about this stuff? Obviously addition, subtraction, multiplication and division are useful, but what about square roots and exponents? Well, if you wanted to calculate the distance between two actors, you could use the Pythagorean Theorem. Considering that the entire game exists on a grid, and that there are right-angles all over the place, $A^2 + B^2 = C^2$ has significant purpose in GameSalad. But for now, it's back to the basics.

If you've endured a typical high school math class, you probably remember that parentheses play an important role. Whatever is inside the parentheses is calculated first. It is no different with GameSalad. It's also very important to keep track of each open and close parenthesis. Otherwise, your game could crash or you could cause problems with your expressions.

With the percent sign, you can call the "Modulo" function. It will deliver the remainder after division. For example, 23 divided by 10 is 2.3. If you used the "Modulo" function, the result would be 3. It would be entered into GameSalad as 23%10. The larger number goes first. If the second number is larger, then "Modulo" just returns the first number. So, in GameSalad land, 10%23 = 10. So, what good does that server? Well, for starters, it's a good way to keep track of even and odd things.



Constrain Attribute

Constrain Attribute: self.Odd To: floor(game.Time %2)

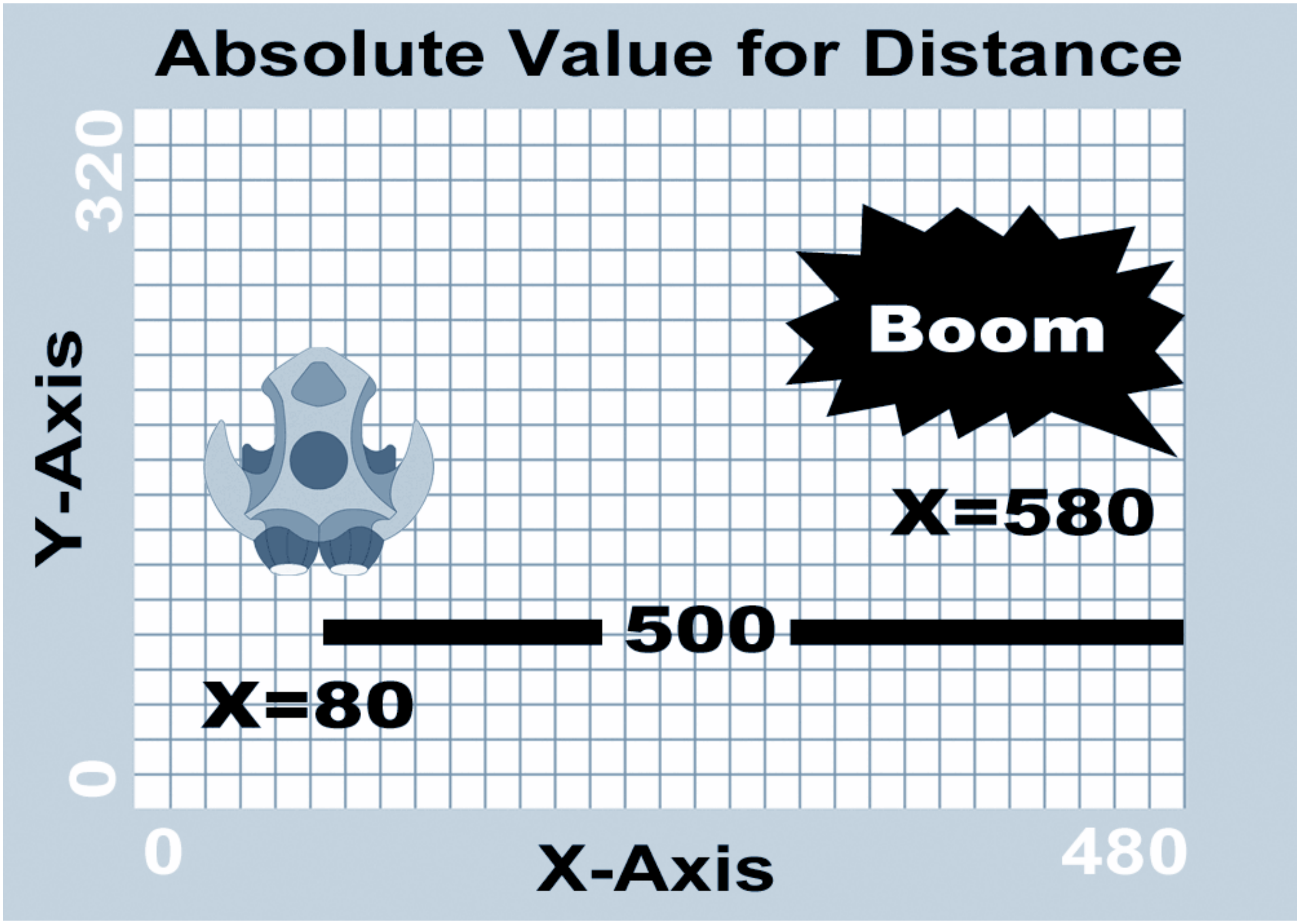
In the above example, the self.Odd attribute is a "Boolean". Instead of typing "False" or "True", You can use the number 0 for false and the number 1 for true. (Any number other than 0 is true.) The expression entered in the "To" field basically returns a 0 when the seconds are even and a 1 when the seconds are odd. The example is just a proof of concept. You don't have to use game.Time. It could be part of a coin game or a dice toss. It doesn't have to be limited to even or odd either. The result for "Modulo" can be a decimal. If the second number is negative, the result will also be negative.

I jumped a little bit ahead with the previous example. Even and odd are phrases for integers. The game.Time attribute uses decimals. I used the "floor" function to strip out the decimals. In addition to the basic functions, like multiplication and addition, GameSalad also includes some advanced functions.

You can type these functions manually, or you can use the "insert function" drop-down menu. Whichever way you prefer, it's still important to keep track of the parentheses.

abs - This will return the "Absolute Value" of a number. In layman's terms, a negative number would change to positive. For example, -17.5 would change to 17.5. This function is useful in situations where negative numbers would not work. The game's volume cannot be negative. An actor's RGBA colors cannot be negative.

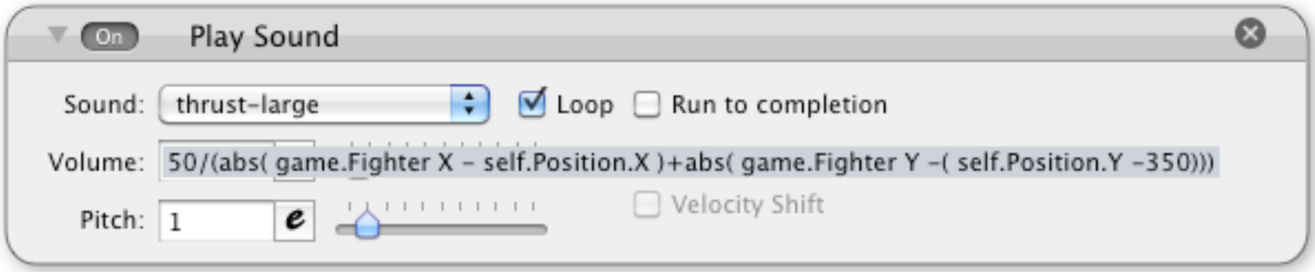
If you wanted to create environmental sound effects, such as an actor walking towards a waterfall, the rushing water sound should get louder as the main character walks closer to the waterfall. The volume of the sound could be controlled by the distance of the two actors. If the waterfall's X position is 1000, and the actor's X position is 500, the difference could be turned into a volume setting. 1000 minus 500 = 500. The problem happens when the actor walks past the waterfall. Instead of 1000 minus 500, we'd get 1000 minus 1500. That's -500. When converting that number into volume, a negative number would cause problems. The sound effects can't play at -5 volume, but the absolute value of -5 would work just fine.



The above image is to help you visualize this. Your fighter pilot might be able to hear the off-screen explosion, but unable to see it. The further away the explosion, the quieter the sound should be.

When you use the "abs" function, it doesn't matter if you subtract 580 from 80 or 80 from 580. The absolute value is still the same 500 pixels. With calculations like this, you can create many different types of effects. Not only can you control the sound of the explosion, but you could use a "Rule" to create the force of the explosion. Lighting effects could also apply. If the explosion was replaced by a fire, the actor could become brighter as it neared the flames. These are all examples of math bringing life to your game. Sharing data between GameSalad actors is tricky. But when you do, mathematical functions can make those interactions more interesting.

The "abs" is an important, so here's a real world example.

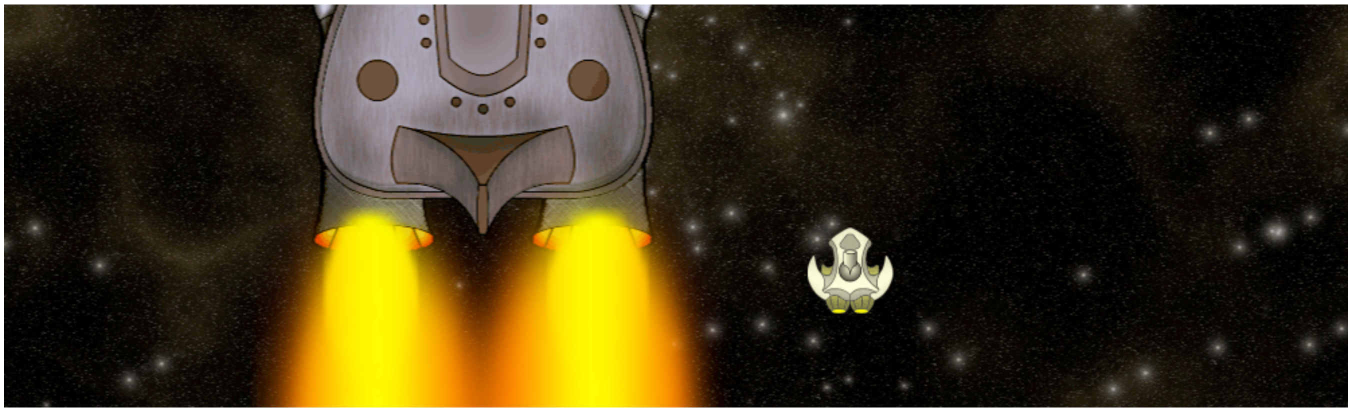


Sound: thrust-large ☑ Loop ☐ Run to completion
Volume: 50/(abs(game.Fighter X – self.Position.X)+abs(game.Fighter Y –(self.Position.Y –350)))
Pitch: 1 ☐ Velocity Shift

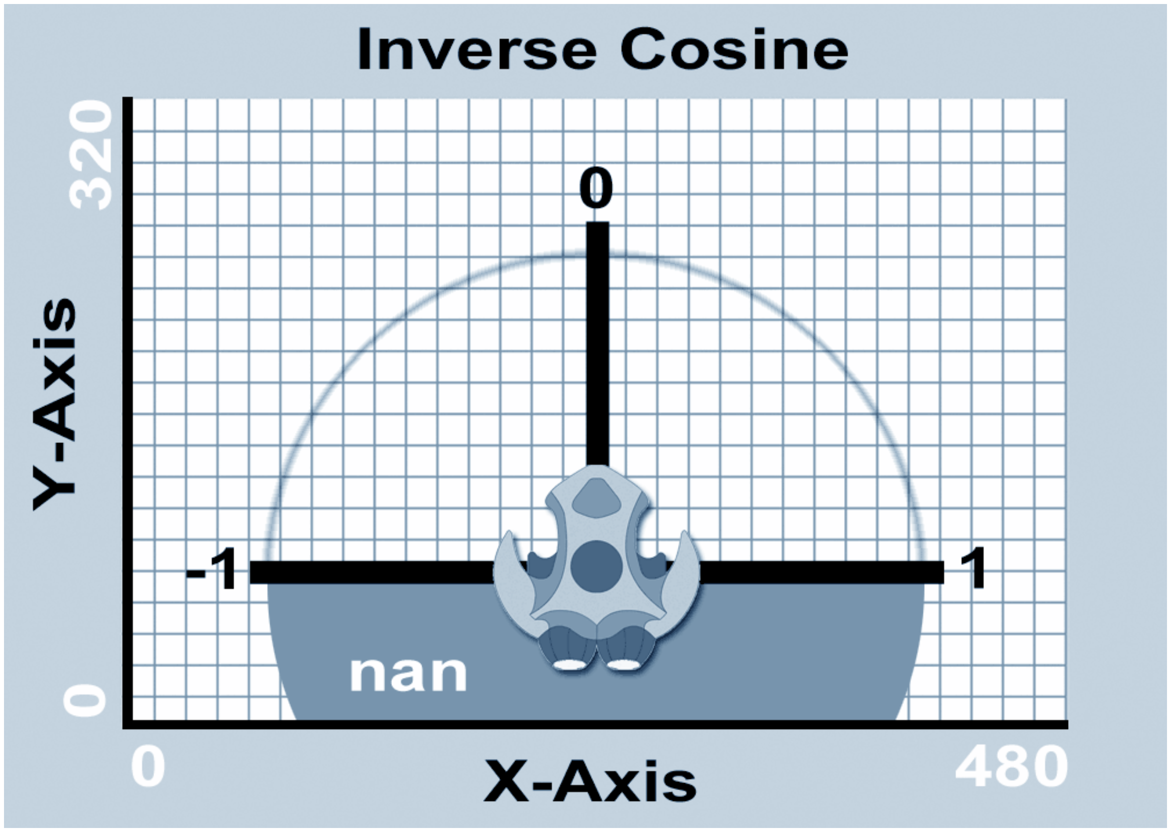
That's a "Behavior" from my very first GameSalad game. It was called Photics: Revisions - Course Correction. In that game, you're supposed to protect a big ship. And even though sound doesn't travel through space, it would be a pretty boring game without sound effects. So, the closer you get to the ship's thrusters, the louder the sound gets. To accomplish this, I used the following expression for "Volume".

50/(abs(game.Fighter X-self.Posistion.X)+abs(game.Fighter Y-(self.Position.Y-350)))

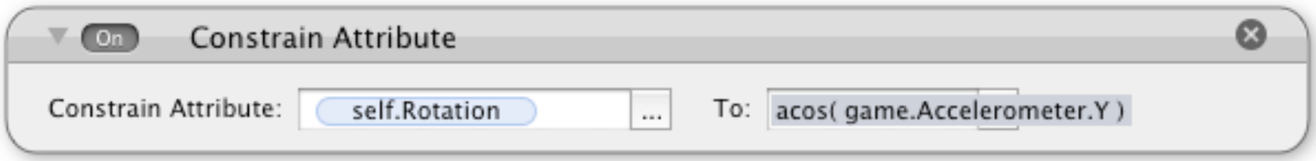
I'm adding the absolute value of the X and Y distances between the two actors. (I subtracted 350 from self.Position.Y because the thruster sound should come from the bottom of the actor, not the middle.) As for the number 50, I admit that this was a crude attempt. The distance has to be converted to a number between 0 and 1, as that's how "Volume" works in GameSalad. Yet, the volume has to get lower as the fighter flies away. So, I decided to treat the expression as a fraction. The distance becomes the denominator and the numerator is just an arbitrary number. I kept plugging in different numbers until I found something that worked. Remember, I'm someone who failed math class. I found it dull and boring. But with GameSalad, math is something that I can visualize. I can see it working. It has purpose. It has entertainment value. I'm learning by playing. Fun is not often associated with math class, but that's what happens with GameSalad. So, if I can come up with a complicated looking formula like that, maybe you can too.



acos - Ah, it's my arch enemy… the "arccosine", also known as the "Inverse Cosine". Actually, we're buddies now, as "acos" is a good way to convert "Accelerometer" data into angles. The input range for "acos" is -1 to 1. (That's similar to the "Accelerometer" data.) It will return an angle from 180 to 0. Anything outside that range will not be processed by GameSalad. Instead, the invalid result is simply listed as "nan".



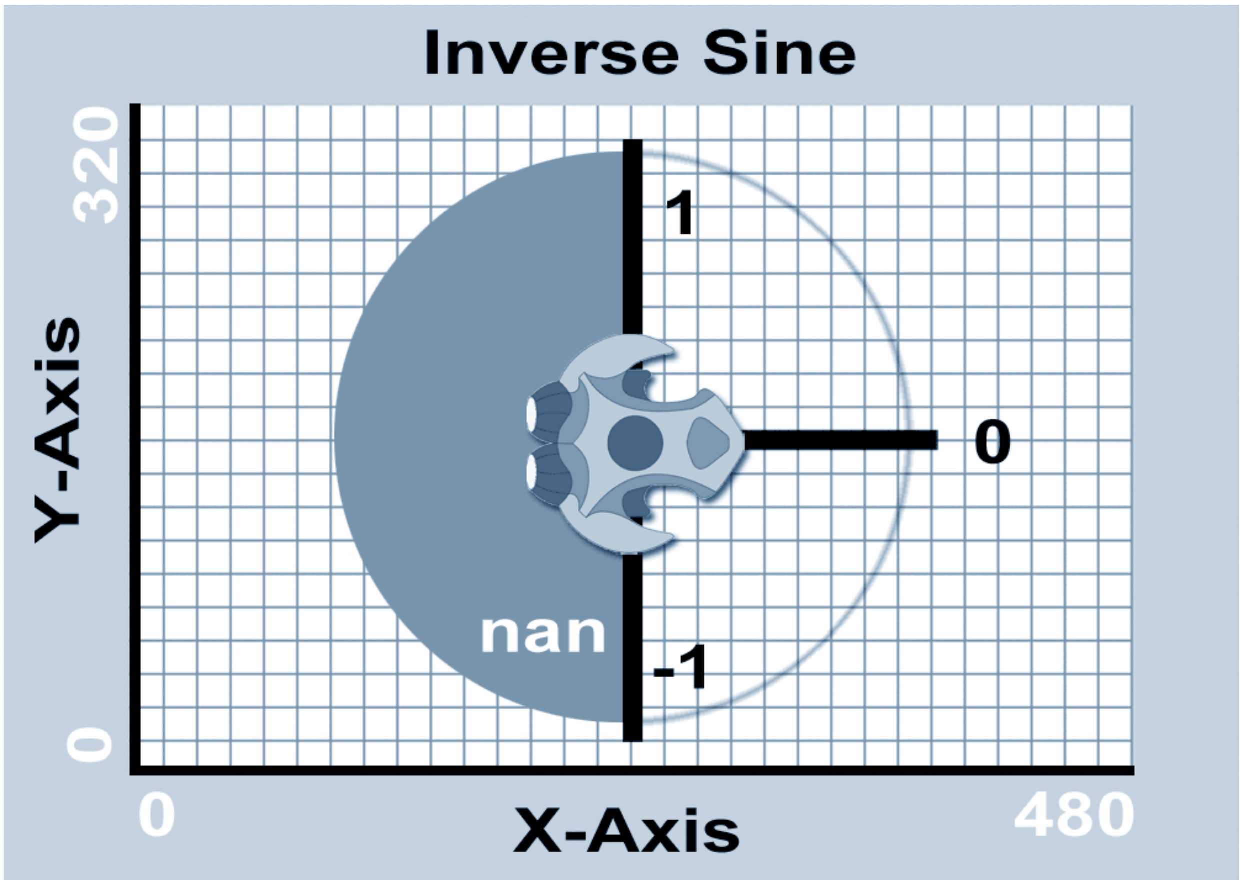
If you already have an iOS device ready for testing, you can visualize the difference for yourself. Use the "Constrain Attribute" behavior to constrain the self.Rotation attribute to acos(game.Accelerometer.Y).



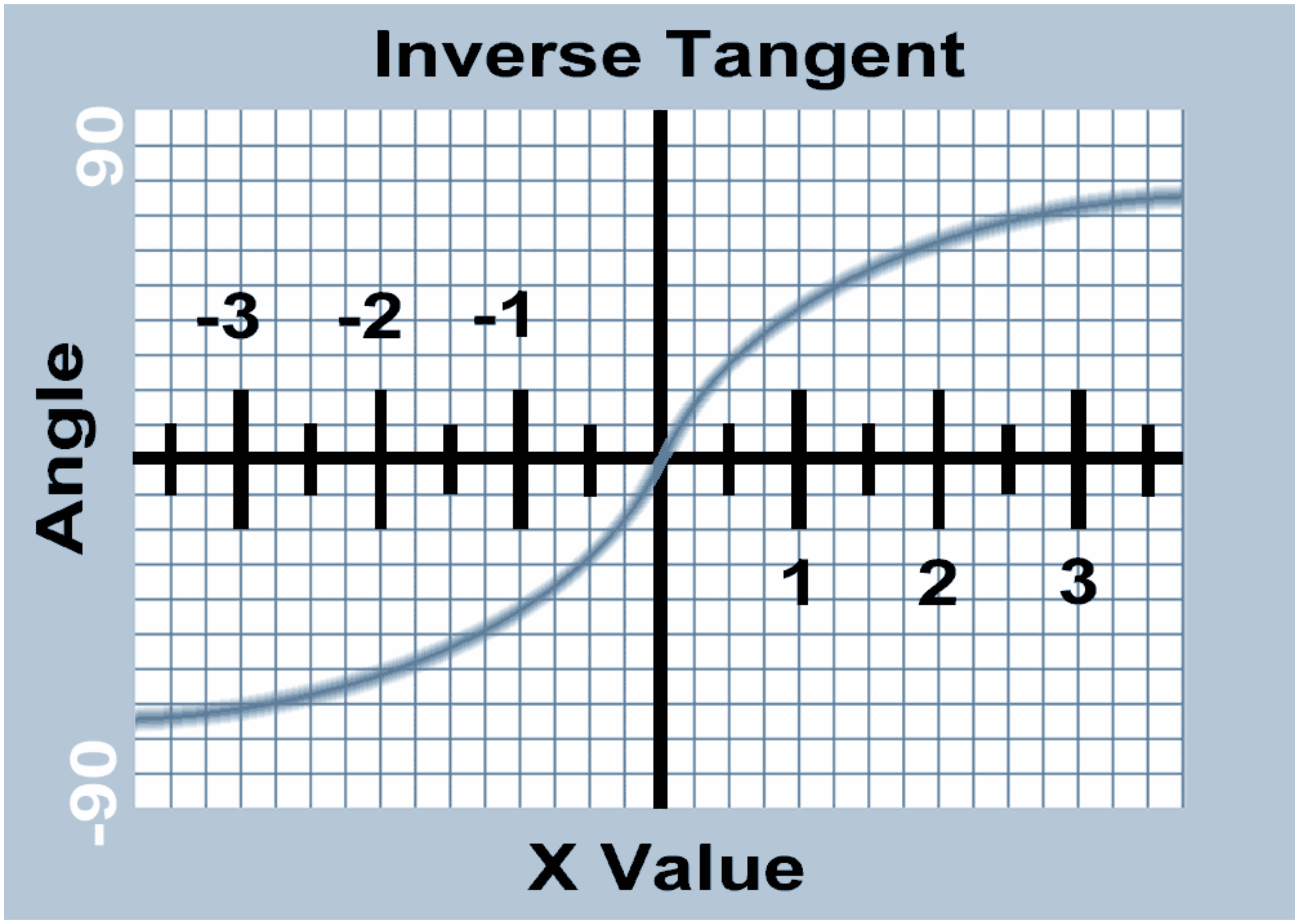
Constrain Attribute: self.Rotation To: acos(game.Accelerometer.Y)

There are two important things to notice from this test. One, the game might crash if you turn your testing device too far. That's the "nan" value messing up the self.Rotation attribute. Also, you might notice that your actor is rotating in the opposite direction of the testing device. If you're looking to build accelerometer based controls, that might not be the effect you're looking for. There are ways to fix these issues with other functions.

asin - The "arcsine" or "Inverse Sine" is another trigonometry function. In GameSalad, the "asin" function will accept an input of -1 to 1. The result will be from -90° to 90°. If the input value is outside of the accepted range, the result will be "nan".



atan - At initial glance, the -90° to 90° range of “arctangent” (or “Inverse Tangent”) seems similar to the “Inverse Sine”. Yet, the path is quite different. For instance, there’s no limit on the input value. Plus, the input value has to be pretty high before the angle will be rounded up to 90° — it has to be well over 9000!



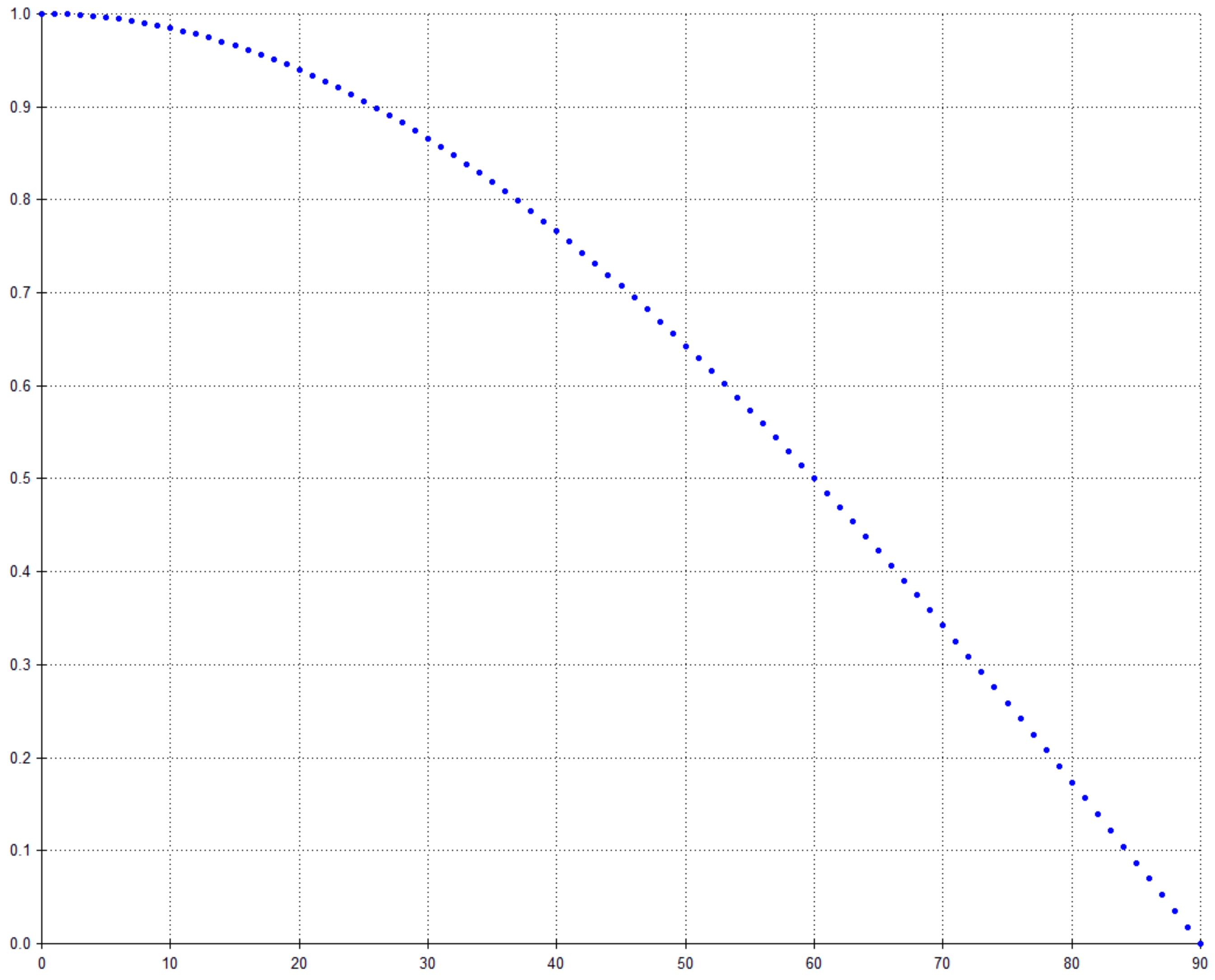
You can convert the -90° to 90° to 180° to 0° by adding 90 to your result. However, this is not ideal for accelerometer based controls, as it’s fast in the center and slow on the edges. You might be able to use it for other techniques. The diminishing rate of change, from the value of zero, might be useful to create things like black holes, bungee cords and pendulums. For actors that behave differently, based on their distance from a certain point, the “atan” function could be useful.

ceil - This function will round up a decimal to the nearest integer. For example, 1.6 would become 2, -1.4 would become -1, and 5.4 would become 6. When I see this function, I think of the word “Ceiling”. When I want to look at the ceiling, I usually look up.

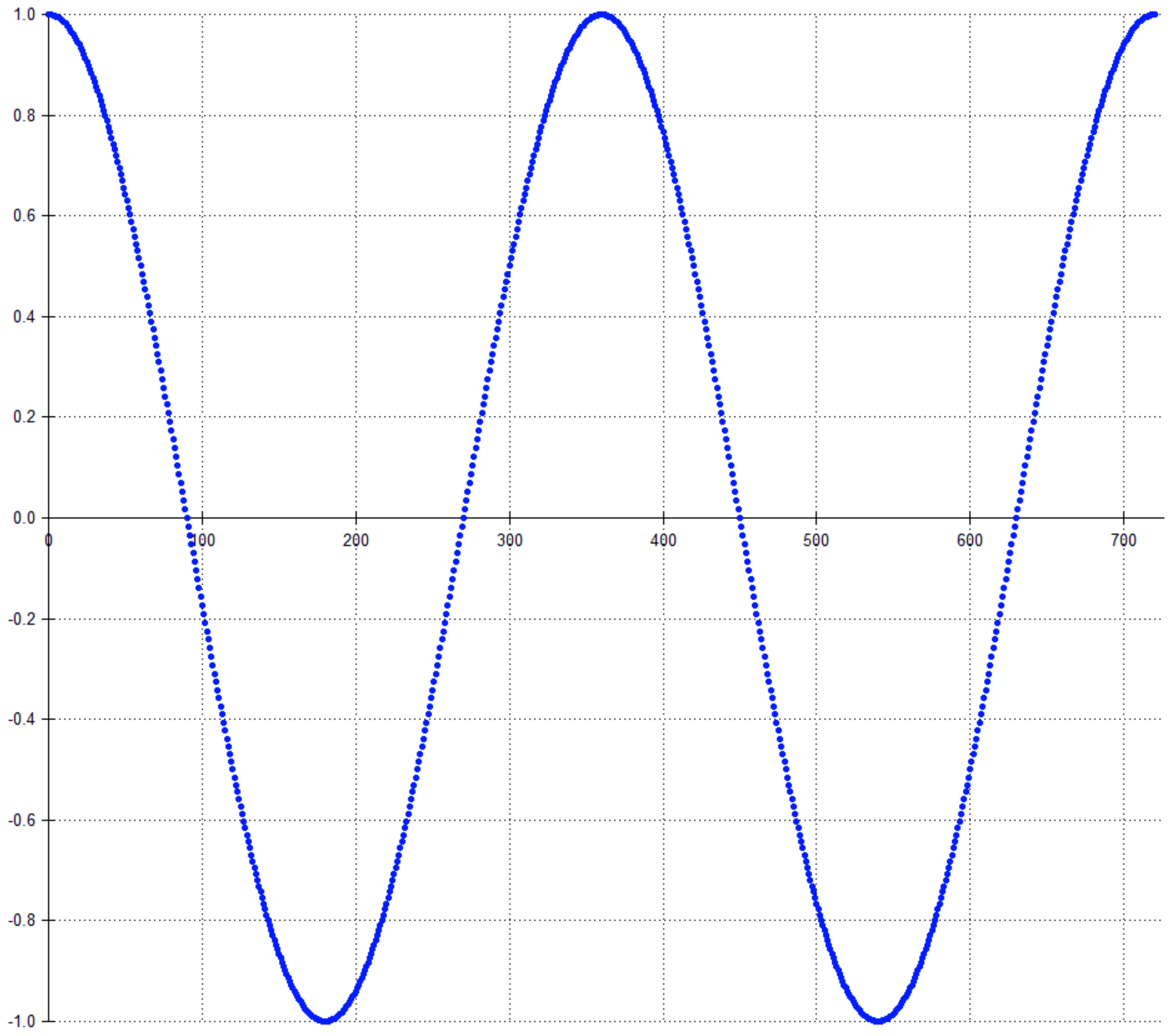
cos - This is the "Cosine" function. It's huge in video games, as it can make objects move in a wave pattern. It can also be combined with the "Sine" function, to make objects move in a circle. When used with the "Absolute Value" function, "Cosine" can make actors bounce like a ball. You can try constraining an actor's self.Posistion.Y value to the following equation…

abs(cos(game.Time*125))*125+50

When you select "cos" from the "insert function" menu, it adds cos(angle) to your expression. You can put any number inside the parentheses, but then it's simply treated as an angle. The "Cosine" function will then return a value with a range of -1 to 1.



The above chart shows the result for the first 90 degrees, from cos(0) to cos(90). The numbers "ease in" towards their decline. If you've tried making an actor bounce with the "cos" function, that's why the ball slows down as it reaches its arc.



The above chart shows two full "Cosine" cycles. When the value for X ranges from 0 to 360, the value for cos(X) will equal 1 when X=0, X=360 and X=720.

If you would like to better visualize this repeating cycle, you can constrain an actor's Y location to the following expression…

cos(game.Time*50)*100+160

The game.Time attribute gives the actor motion. When a GameSalad game is launched, the game's clock is always running. The 160 value puts the actor in the middle of the screen. (This assuming that the orientation is landscape.) The value of 50 controls the speed of movement.

A higher number would make the actor move faster. The value of 100 controls the range of movement. A higher number would make the actor travel further.

exp - This is the "Exponential" function. Use this function if you want some ridiculously large (or ridiculously tiny) numbers. The "Exponential" function can get you there in a hurry.

X	Result	X	Result
0	1	1	2.71828
-1	0.367879	2	7.38906
-2	0.135335	3	20.0855
-3	0.0497871	4	54.5981
-4	0.0183156	5	148.413
-5	0.00673795	6	403.429
-6	0.00247875	7	1096.63
-7	0.000911882	8	2980.96
-8	0.000335463	9	8103.08
-9	0.0012341	10	22026.5

Why would you use something like this? Basically, it's for growing or shrinking a value. At initial glance, it looks like a cool basis for an RPG experience chart. TANK is already part of the level grind and he hasn't even been added to the game yet. To gain a level, the experience point requirement could be something like…

ceil((exp(self.Level)+100)

Of course, to math buffs, "e" is of greater significance than level grinding. It's an irrational number that is up there with pi. When you use "exp", it's similar to the use of e^X. The value of "e" is approximately 2.71828, as shown by exp(1).

floor - When you think of a regular floor, you probably think of something down towards the ground. The "floor" function will round down a decimal to the nearest integer. For example, 1.6 would become 1, -1.4 would become -1, and 5.4 would become 5. As shown in the even and odd example, we were checking to see if the seconds were even or odd. If a number was even, there would be no remainder when the seconds were divided by 2. If the number was odd, the remainder would be 1. However, the game.Time attribute stores the data as a decimal. The “Modulo” function would include that information, skewing the even / odd results. The “floor” function strips out the decimal part.

floor(game.Time)%2

If the game.Time was 5.55, the floor value of that would simply be 5. The “Modulo” portion of the expression returns the number 1. If that value was sent to a “Boolean” attribute, it would be interpreted as “True”.

ln - This is the “Natural Logarithm” function. It is related to the “exp” function. If you had the value of 22026.5, and you wanted to know how many steps it took to get there, the “ln” function can give you the answer. The “Natural Logarithm” function will not work with a number less than, or equal to, zero.

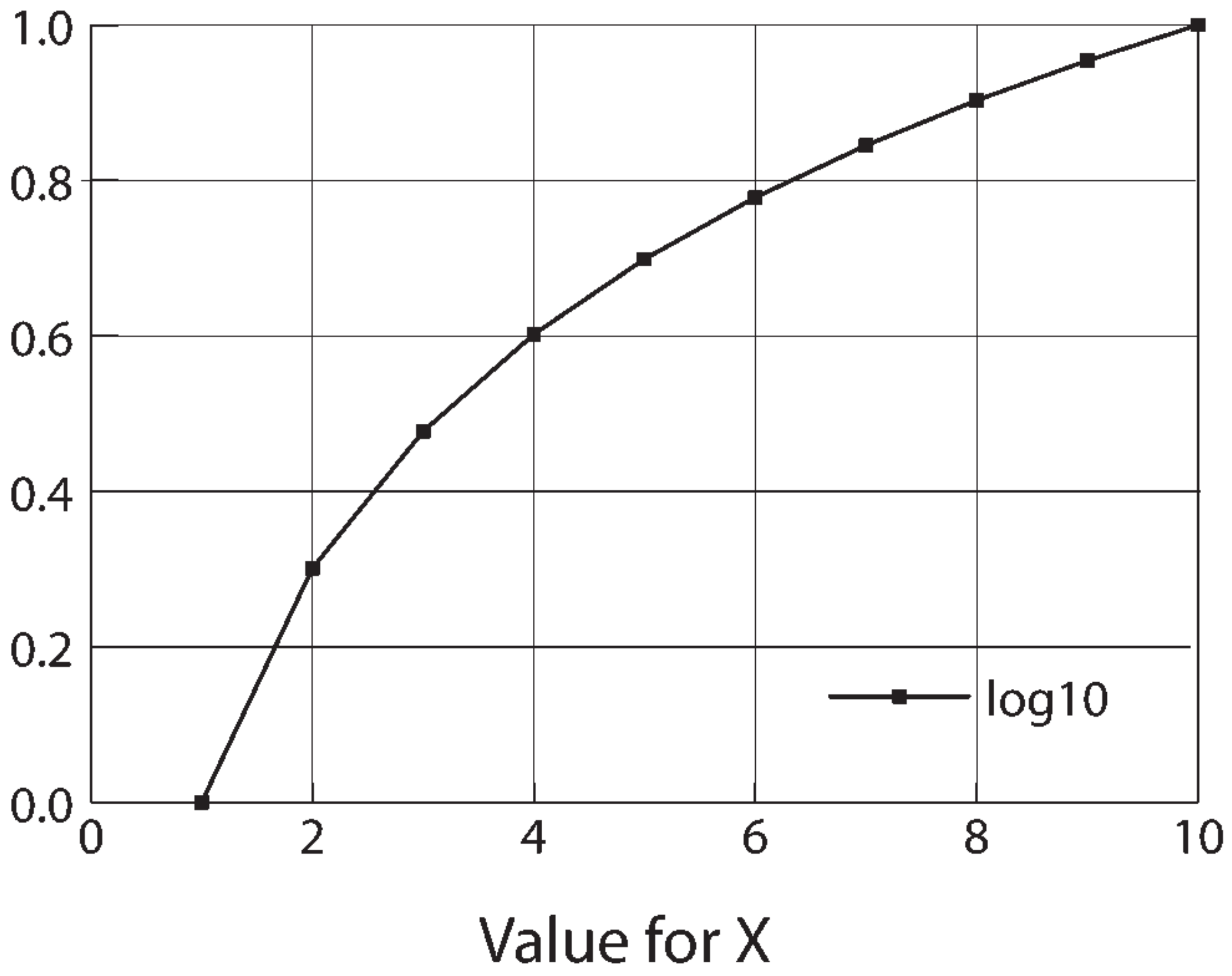
log10 - This is “Base 10” logarithm. It accepts values that are greater than zero. Entering a value lower than zero will return a “nan” error. Entering a value of 0 will return a “-inf” error. This function is a quick way to see how many digits a value has.

X	Result		X	Result
1	0		10	1
0.1	-1		100	2
0.01	-2		1000	3
0.001	-3		10000	4
0.0001	-4		100000	5
0.00001	-5		1000000	6
0.000001	-6		10000000	7
0.0000001	-7		100000000	8
0.00000001	-8		1000000000	9
0.000000001	-9		10000000000	10

So naturally, I wondered what would happen if I didn't use a multiple of 10. I figured that 500 should give a result of 2.5, because it's halfway between 100 and 1000. The "log10" function doesn't work that way. That's because the "log10" function is not linear.

log10(500) = 2.69897

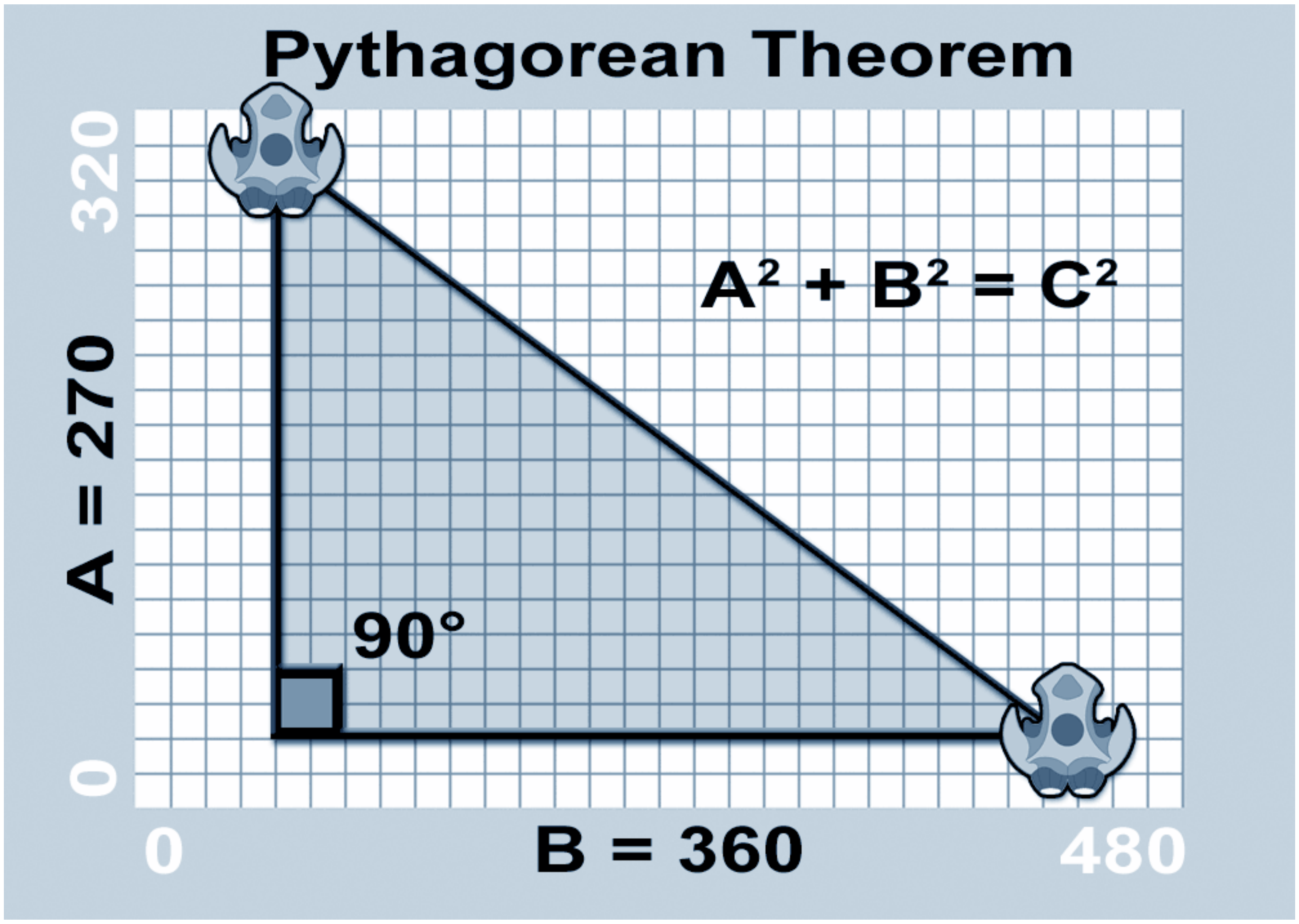
If you used the "floor" function, you could strip out the decimal numbers and get a solid integer. In the chart to the left, you can see how the "log10" function "eases out".



This chart could be extended, but it's basically the same cycle…

- log10(5) = 0.69897
- log10(50) = 1.69897
- log10(5000) = 3.69897

magnitude - This is the "Pythagorean Theorem" in action. $A^2 + B^2 = C^2$. By entering the X distance and the Y distance between two points, the "Magnitude" function can calculate the diagonal distance between two actors.



Here's how it works. Every GameSalad actor has an X and Y location. In the above example, one fighter is at location 60, 305. (The X value is first.) The X value is 60 and the Y value is 305. The second fighter is at location 420, 35. The X value is 420 and the Y value is 35. The difference between the two X locations is 360. The difference between the two Y locations is 270. Now the "Pythagorean Theorem" can be used to figure out the length of the triangle's third side, which is also the distance between the two actors. The answer is…

magnitude(360,270)

The "Magnitude" function makes it easier. You can also use attributes inside the "Magnitude" function. By sharing actor data, through the use of game or scene attributes, you can calculate the distances between actors with the magnitude function. When you insert the "Magnitude" function into the “Expression Editor”, it will have a space for an X and Y value. A common use of the “Magnitude” function is like the following expression…

magnitude((game.X-Target-self.Posistion.X),(game.Y-Target-self.Posistion.Y))

That expression will take the absolute value of X and Y, and then apply the “Pythagorean Theorem”. The result will be the distance between the two points. This technique is use for many advanced game elements. If you wanted to add a touch based controller to your game, the distance of the controller (from the center point) could be used to determine speed. If you wanted more realistic explosions in your game, distance could be used to determine the force of the explosion. That same idea applies to sound effects. You could make supporting actors louder, based on how close they are to the main actors.

max - This function will return the larger of two numbers. For example, max(1,5) would be 5. This function is useful if you do not want a value below a certain point.

min - This function will return the smaller of two numbers. For example, min(1,5) would be 1. This function is useful if you do not want a value above a certain point.

You can combine the “min” and “max” functions to keep a value within a specific range. For example, you could prevent a player’s energy from falling below zero or going above 10.

“Change Attribute” game.Player.Energy to:
min(10,max(0,game.Player.Energy))

The above “Behavior” would take the current game.Player.Energy and then run it through the expression. First, the “max” value would be calculated. If game.Player.Energy was 11, it would be selected over 0. That’s because 11 is greater than 0. However, 10 is less than 11, so the “min” function would choose 10 over 11. The final result would be 10 for the player’s energy.

mod - This is a newer GameSalad function, but the effect is not new. It’s the “Modulo” function. Instead of using a percent sign, you could use mod(x,y). 10%4 is the same as mod(10,4). This is a matter of personal preference. However, seeing the letters “mod” might make your expressions more readable for other developers. The percentage sign could be confused with a different type of mathematics.

pow - This is another way to calculate exponents. The pow(2,3) is the same as 2x2x2. Take the number two, and times it by itself three times. The result is eight.

random - This allows you to generate a random integer. The range of the random number is determined by the values within the parentheses. For example, a coin toss would be random(1,2) and a six-sided die would be random(1,6). If you wanted a pair of dice, you could use random(1,6) twice. You could separate the results with two attributes - like game.Dice1 and game.Dice2 - or could add up the result immediately.

(random(1,6))+(random(1,6))

For the serious dice rollers, there is something important to know. Using random(1,6) twice it's not the same as random(2,12). With a pair of dice, 2 is the lowest possible outcome and 12 is the highest possible outcome. Yet, the odds with a 2-12 range are different from a 1-6 + 1-6 range. If you used random(1,12) as your expression, there's only one chance to roll a 7. If you used random(1,6) twice, there are six chances to roll a 7. They are 1+6, 2+5, 3+4, 4+3, 5+2 and 6+1.

sin - The "Sine" function is similar to the "Cosine" function, as "Sine" can also make objects move in a wave pattern. The output range is -1 to 1. You can visualize how these two functions work together.

"Constrain" self.Position.X to:
(cos(game.Time*88)*99)+240

"Constrain" self.Position.Y to:
(sin(game.Time*88)*99)+160

The above "Behaviors" — with their associated expressions — will cause an actor to move in a circular pattern around the center of the screen. The "Orbital" template shows this in action. It doesn't have to be circular though. By changing the numbers 88 or 99 in one behavior, but not the other, the path can be dramatically different.

sqrt - This function is used for calculating the "Square Root" of a number. If that number is less than zero, the result will be "nan". Using sqrt(X) is an alternative to the X^.5 expression. However, the latter will still process negative numbers. The "abs" function might be of use with the "sqrt" function, as the "abs" function forces positive numbers.

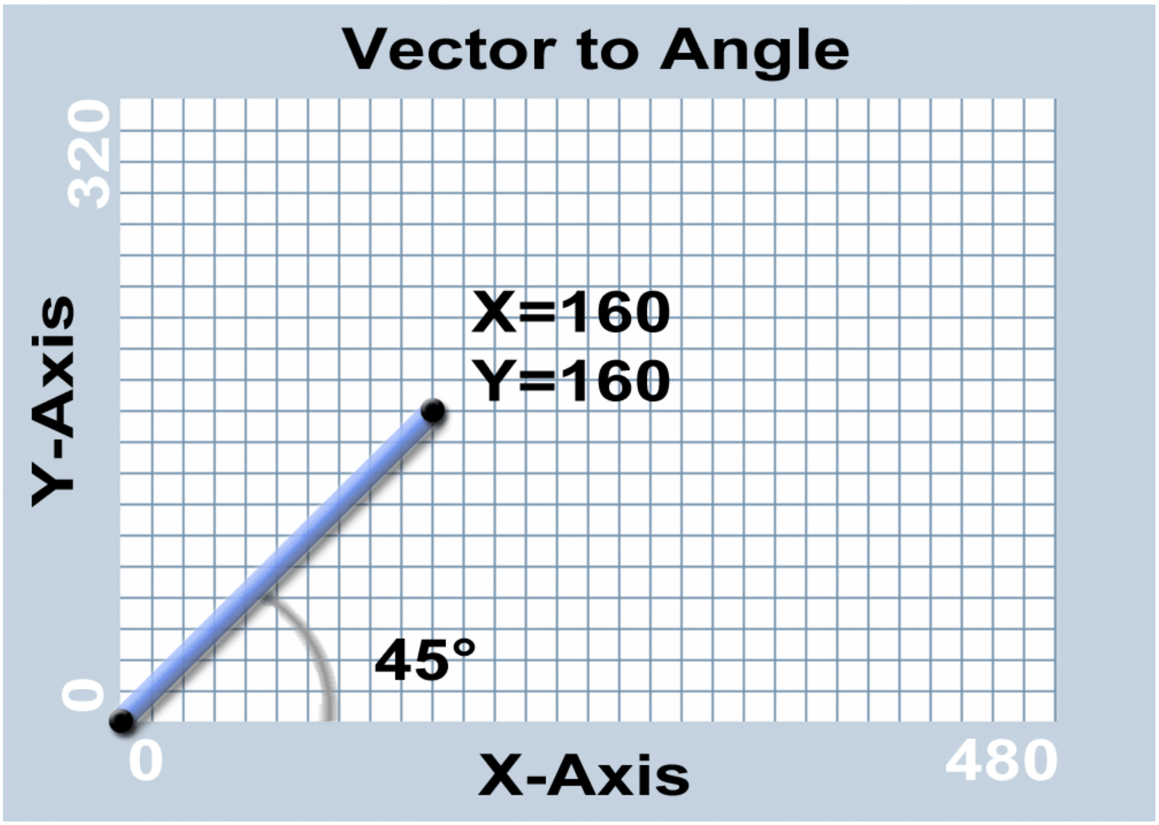
sqrt(abs(x))

Why would you need the "Square Root" of a negative number? Does such a thing even exist? If you multiply a negative number by a another negative number, you'll get a positive number.

For example $-5^2 = 25$. The “Square Root” of 25 is 5. You can’t get back to -5 by normal means. You could pretend that the “Square Root” of -25 is -5, which is why you might use the X^.5 expression. But why use “Square Root” at all?! If you want a larger actor to move in proportion to a smaller actor, like with a mini-map or radar, the “sqrt” function might be useful. But since the “Pythagorean Theorem” is built into the “Magnitude” function, one of the major uses for the “sqrt” function has been mitigated. However, if you have a solid understanding of trigonometry, you could enter your own expressions — that where the “sqrt” function would be useful.

tan - “Tangent” is another function related to trigonometry. The “Sine”, “Cosine” and “Tangent” functions are useful in finding the size of unknown angles in right triangles. Yet, the next function makes that task a lot easier.

vectorToAngle - By entering the value for X, and a value for Y, the “vectorToAngle” function will return an angle. That angle is from the point of origin to the X and Y coordinates.

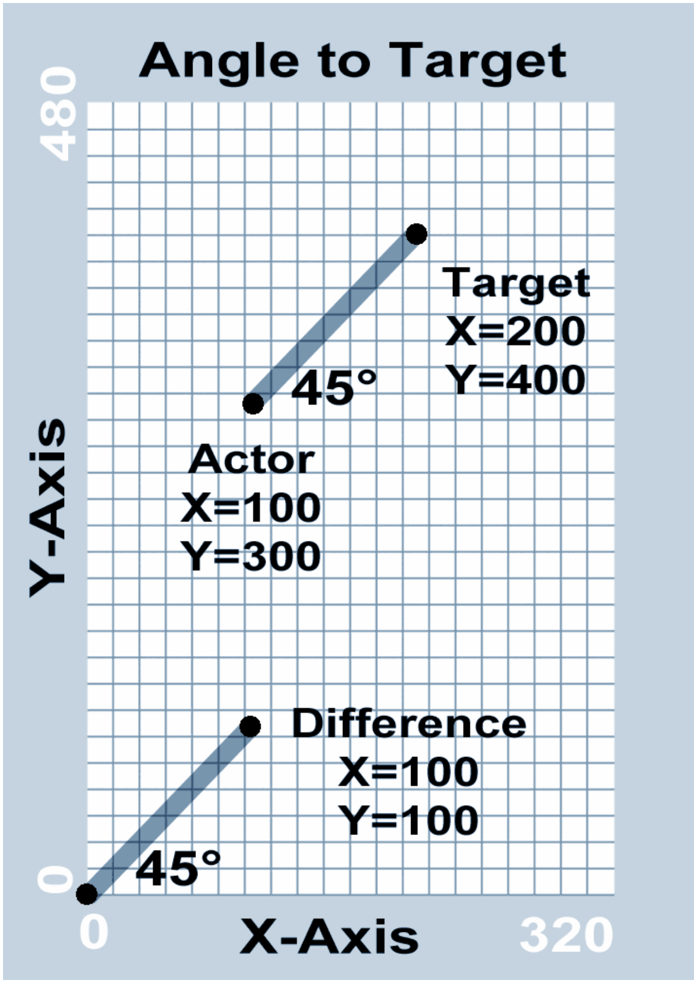


In the above graph, the X and Y values are both 160. By using the “vectorToAngle” function, your GameSalad application can calculate the slope of the line, from origin to 160, 160. The result is 45°.

Now that's great, if you wanted to know the angle from origin. But it's more likely that you'll want to know the angle between two actors. You can still use the "vectorToAngle" function to accomplish that task.

vectorToAngle(game.X-Target-self.Position.X, game.Y-Target-self.Position.Y)

The above expression will give the angle between the actor and its target, in relation to the scene. It's important to remember the order. Otherwise, you'll get different results.



In the graph above, the actor's location is 100, 200 and the target is 200, 400. Using the above expression, the values need to be subtracted. 200-100=100 and 400-300=100.

When comparing the original line of distance, with the line between the origin and the difference point, they are parallel to each other.

The understanding of this concept is important to GameSalad game development. For example, if you wanted to shoot at the target, you'd want your bullets to face the enemy. You could use the “vectorToAngle” function as part of your “Spawn Actor” behavior. To determine the proper angle for your projectiles, your target could be a mouse position or a touch point. This function is also useful for on-screen controls. However, if you need an actor to continuously face a certain point, the “Rotate to Position” behavior might be easier for you to use.

Chapter #5 Summary

- While GameSalad doesn’t require programming knowledge, a strong understanding of math and logic can be helpful.
- GameSalad accepts many different mathematical functions. They can create impressive effects for your games.
- This chapter is a tough one. If you don’t understand all the concepts right now, maybe review this chapter after you have more experience with GameSalad.

Chapter #6 - Physics

GameSalad has an amazing physics engines. That one of the first things that I found so endearing about the software. Within a few minutes, I could have actors bouncing into each other. Having used other game development software before, I was pleased to see things were different with GameSalad.

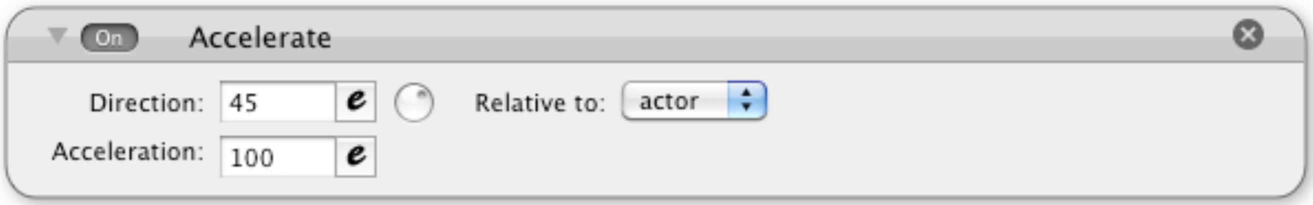
However, while GameSalad is an impressive tool, don't treat it like the real world. That's not to highlight the limitations of the software, but the limitations within your own mind. You've lived in the real world your whole life. You expect certain things to happen a certain way. GameSalad doesn't always work that way. Often, the unexpected will happen. Actors won't behave the way you want them to. That's what this chapter is about. It's to review GameSalad's Physics engine, how to take advantage of it and how to build a better video game.

From the moment you create a new actor, physics properties are assigned to it. You can access this information from an actor's "Attributes" settings. The main categories are "Motion" and "Physics". There are four main settings for "Motion".

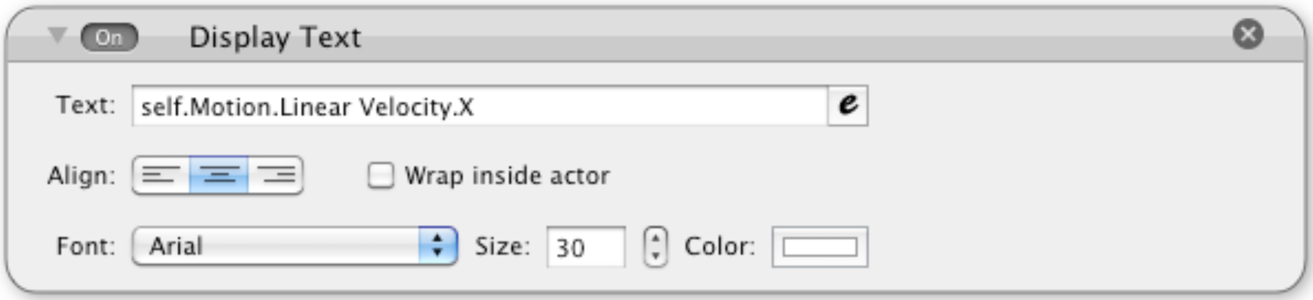
Linear Velocity - This is the speed of an actor, along the X and Y axes. The unit of measurement is pixels per second. If an actor has a self.Motion.Linear Velocity.X value of 200, and its self.Position.X value is currently 150, the X location for the actor will be 350 one second from now — unless some outside force acts upon it. "Linear Velocity" values can be positive or negative. When X & Y is combined, it gives an actor 360° of movement. If an actor has self.Motion.Linear Velocity.X value of 200, and a self.Motion.Linear Velocity.Y value of 200, the direction is 45°. It's traveling along both axes at the same rate.

Angular Velocity - This is the speed and direction of the actor's rotation speed. If the value is negative, the actor spins clockwise. If the value is positive, the actor spins counter-clockwise. The larger the value, the faster the spin.

If you need help visualizing this, you can try some experiments. Create an actor and give it a "Move" behavior. Then, give the actor a "Display Text" behavior. Enter either (or both) of the "Linear Velocity" attributes in the text field, via the "Expression Editor". By changing the angle of the "Move" behavior, you can monitor the change in linear attributes. You could also try this with the "Accelerate" behavior.



Direction: 45 Relative to: actor
Acceleration: 100



Text: self.Motion.Linear Velocity.X
Align: Wrap inside actor
Font: Arial Size: 30 Color:

"Linear Velocity" doesn't have to be constant. Use of the "Accelerate" behavior can quickly increase or decrease an actor's speed. It can also change direction. To expand the experiment, add a few more actors to the scene and make them bounce with the "Collide" behavior. This will allow you to watch the self.Motion.Angular Velocity attribute change value. Use the "Display Text" to show the value changes after collision.

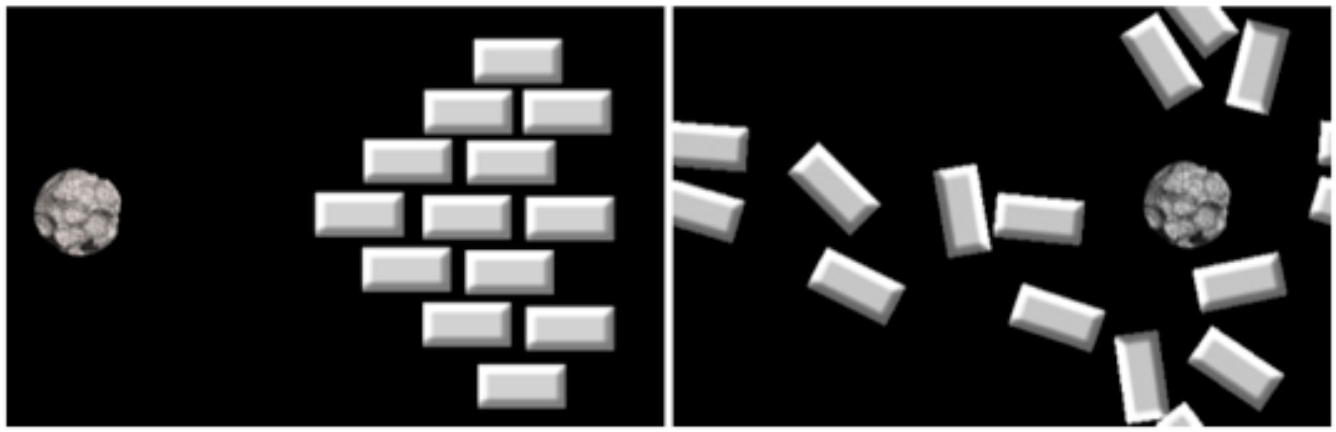
These attributes control an actor's directional and rotational movement. Speed and direction are key concepts in physics and in game development. The above experiments may help you to better understand "Motion" in GameSalad.

Max Speed - This is a limiter. With the "Accelerate" behavior, an actor could increase speed indefinitely. Not only would that make a game hard to play, it would make the game hard to see. If an actor moves too fast, it will just flicker across the screen. That's the hardware, struggling to draw frames of animation. To prevent this problem, it's usually a good idea to limit an actor's "Max Speed".

Apply Max Speed - Even though a value is set for "Max Speed", the limiter doesn't work unless the "Apply Max Speed" option is enabled. There's something extra tricky about that too. You can't apply a maximum speed after the game is play. However, you can modify the "Max Speed" attribute. If you had an actor running on the ground, and then running through water, you could change the "Max Speed" attribute to simulate the different densities of water and air.

A feature like that is awesome for role-playing games or real-time strategy games — when terrain matters. If you like to see it for yourself, The "Water Resistance" template can show the effect in real time. The trick is using an "overlaps or collides" rule. When the actor is overlapping a water actor, the Max Speed can be changed to a lower number. Otherwise, the Max Speed can be changed to the normal value.

Density - As shown with the previous example, the actor moves slowly through water. That's mimicking density, because air is not as dense as water. GameSalad is not for calculating collision between millions of water molecules, but the "Density" attribute allows for collision management on a more macro level. A bowling alley is a good way to think of it. If you threw a bowling ball at a bowling pin, the light pins simply ricochets off the heavier bowling ball. If you want your actors to behave the same way, the bowling ball actor would have a much higher value for "Density". Something like 1000 for the bowling ball, and 10 for the bowling pins, could create a believable effect.



Those bricks don't stand a chance, not when the asteroid has 100 times more density. The value for "Density" must be greater than or equal to zero. That still leaves plenty of room for experimenting. Testing different "Density" values is a good way to learn about this attribute. It can also be fun!

Friction - This setting has two functions. When an actor collides with another actor, it can bounce with the "Collide" behavior. But with the "Friction" setting, actors can also rub against each other. That contact can cause actors to rotate or slowdown. The value for "Friction" has to be greater than or equal to zero. The default setting for an actor is three. If either of the colliding actors has a value of zero for "Friction", no related effects will occur for either actor.

This setting is useful for platformers. With "Gravity", you can tether an actor to the ground. (You can apply X & Y values for gravity in the "Scene" attributes.) Now, when an actor moves across the ground, it can slide or be stopped immediately. If you want to create an ice like surface, give the ground actor a low number for “Friction”. If you want the actor to run across sand, give a high number for “Friction”. However, if a rock is moving up or down a hill, the “Friction” setting won’t cause it to rotate. The spinning, related to the friction setting, only occurs upon the initial impact.

Restitution - This setting controls the bounciness of an actor. The range is from 0 to 2. It’s like a multiplier. The “Restitution” on both actors determines force of the impact. If you want no bounce to occur, set the “Restitution” on both actors to zero. If you want the impact to cause some loss of energy, set the “Restitution” on both actors to less than one. If you want there to be an increase of energy on impact, set the “Restitution” on one of the actors to a value greater than one. A “Max Speed” setting could be very useful here, as a high “Restitution” setting could cause the actors to bounce out-of-control.

- **No Restitution (0)** - Bowling balls, anvils, bricks, walls, floors
- **Low Restitution (0.1-.05)** - Baseballs, grenades, vehicles
- **Medium Restitution (.06-1.0)** - Rubber balls, weak springs
- **High Restitution (1.1-2.0)** - Trampolines, strong springs

Like a compact car hitting a train, “Density” is also a factor in “Restitution”. Lighter (or less dense) actors will barely move heavier actors, regardless of “Restitution” settings.

Fixed Rotation - The spin on an actor can determine how it collides with other actors. If you don’t want your actor to spin when it collides with other actors, you can simply check the “Fixed Rotation” option. Even though this option is selected, you can still change the rotation an actor with “Behaviors”.

Movable - If you do not want an actor to move at all, you can deselect this option. Doing so can help increase the performance of your game, but the actor will lose many of the physics and motion related options, such as angular and linear velocity. This setting cannot be changed while the game is in play. However, there are still ways to move an unmovable actor. You could use the “Interpolate” behavior to change an actor’s “Rotation” or “Position” attributes. Also, collision and overlap detection can still occur with other actors, if they have the “Movable” option enabled. Generally, things like walls and floors are usually not set to move. If you deselect “Movable”, the “Fixed Rotation” option becomes irrelevant. For more on game optimization, see Chapter 18.

Collision Shape - Currently, there are two collision shapes for an actor, “Rectangle” and “Circle”. An actor’s inherent shape is always a “Rectangle”. It has a length and width, the opposite sides are parallel and all of the angles are 90°. However, you can place images inside an actor that have transparency. For example, you could make a circular asteroid.

If you’re trying to evade an asteroid, you wouldn’t want to be killed by an invisible square. By choosing “Circle” for the “Collision Shape”, the collision detection appears to be more natural. The trick is to align your circular graphic with the actor’s shape. Oval shaped images will not work well, as the circular “Collision Shape” will only expand to the smallest length — either width or height. For optimal alignment, the diameter of your circular image should be the same horizontally, vertically and diagonally.

Drag - Perpetual motion doesn’t seem to exist in the physical world. Eventually, outside forces will slow objects down. Even in space, planets fall out of orbit and stars fall into black holes. This is especially true on planet Earth, where there are plenty of things to slow you down. Either a car racing along the highway, or a plane flying through the air, wind resistance slows vehicles down. If you want to add such effects to your game, you could try using the “Drag” attribute. By default, the value is zero. It cannot be set below that point, but it can be increased. The higher the number, the more the actor will be slowed down. If you set the value too high, the actor may be unable to move.

“Drag” is similar to “Friction”, and they can work together, but “Drag” doesn’t require collision or overlapping. That means you don’t have to worry about aligning actors, ensuring that they touch perfectly. Instead, you can have an overall speed dampener on demand.

But unfortunately, things are not always perfect in GameSalad land. An update can change the way things work — most of the times for the better, sometimes for the worse. As of 0.10.1 Beta, “Drag” seems a little broken. While “Drag” will slowdown a moving actor, it doesn’t seem to completely stop the actor. The “Breaking” template illustrates this issue. In the example, pressing the “B” button to slowdown the Tank. With the “Drag” method of breaking, the Tank doesn’t always stop. Instead, it just twitches in place. For more reliable breaking, I included an alternative method. It involves “Interpolating” the “Max Speed” to zero. This technique is much more effective. It’s also an important overall lesson for GameSalad — adaptability. There can be more than one way to accomplish a task, but which is the best one for the situation? A lot of wacky problems can block your path to success. But with creative problem solving, you might be able to think of great solutions.

Angular Drag - If you don’t want your actor to spin indefinitely, you can use “Angular Drag” to slow down an actor’s rate of rotation. An actor’s spin will change the way it collides with other actors. If you let your actors spin out-of-control, you could see some wacky physics in your games.

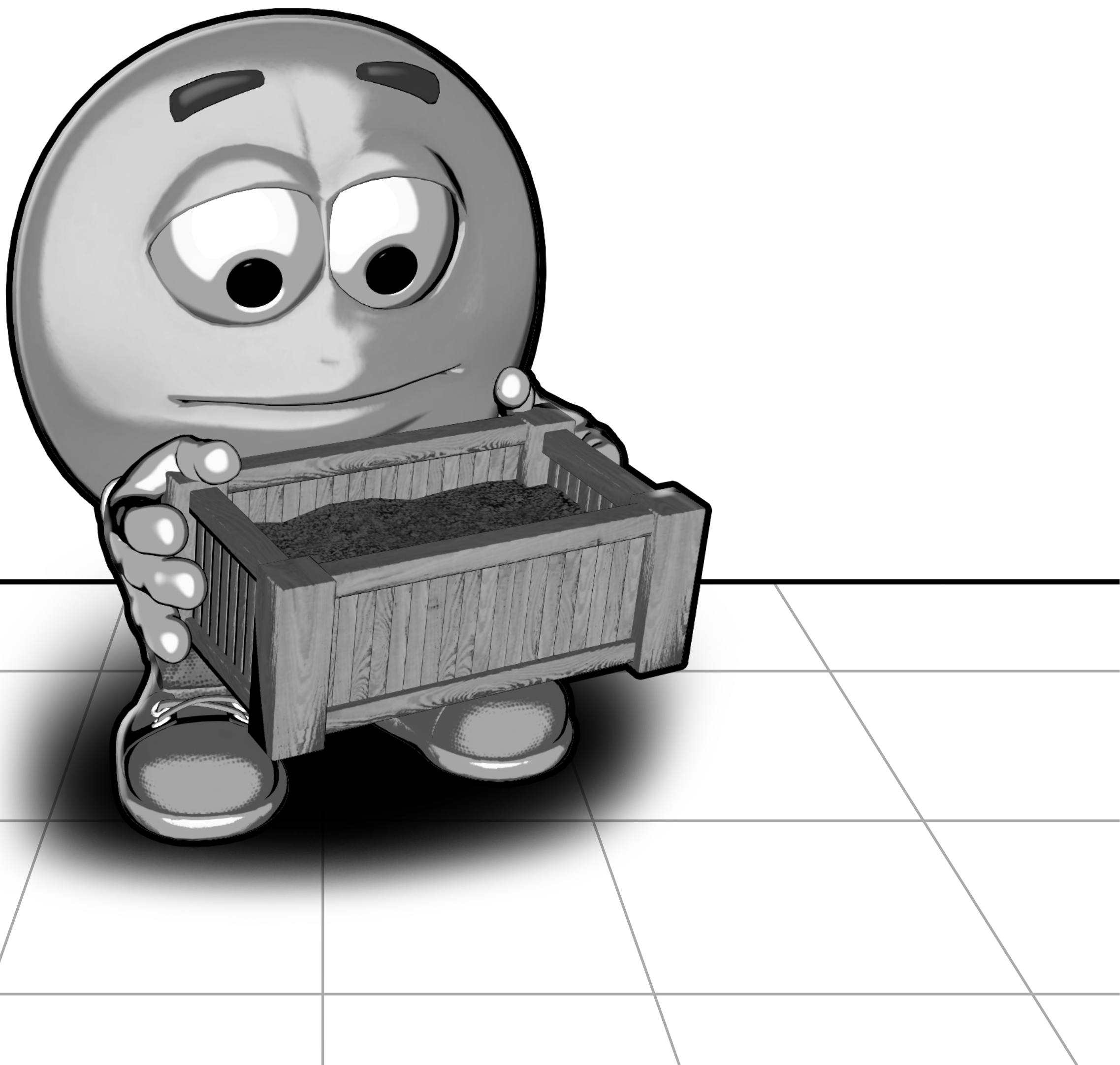
Creating a believable world is a good design goal. Proper use of GameSalad's "Physics" options can help accomplish that objective. However, there are some important balances to remember. The real world can be an excellent source of inspiration, but GameSalad cannot recreate everything in it. Plus, customers are often looking for new experiences. Your game doesn't have to behave like reality.

That's why practice is important. By playing around with the different "Physics" settings, you can learn about the strengths and weaknesses of the software. You can also gain firsthand experience on what's fun and what's unsettling. The knowledge gained through GameSalad experimentation can help you build better games.

Chapter #6 Summary

- Every actor in GameSalad has "Motion" and "Physics" settings. By changing and controlling these "Attributes", you can create a virtual world.
- The "Physics" engine simplifies what happens in real life. What works in reality may not work in your GameSalad game.
- If you are not careful with the "Attribute" values for your actors, your game could be quite unsettling to the players. Use "Max Speed" to keep your actors from moving around too quickly.
- By disabling the "Movable" option in actors that do not need it, you can dramatically improve the performance of your game.
- Experimenting can be helpful. If you want to see what works — chuck some actors in a scene, give them some "Behaviors", mess around with the "Physics" settings, and discover what GameSalad can do.

3

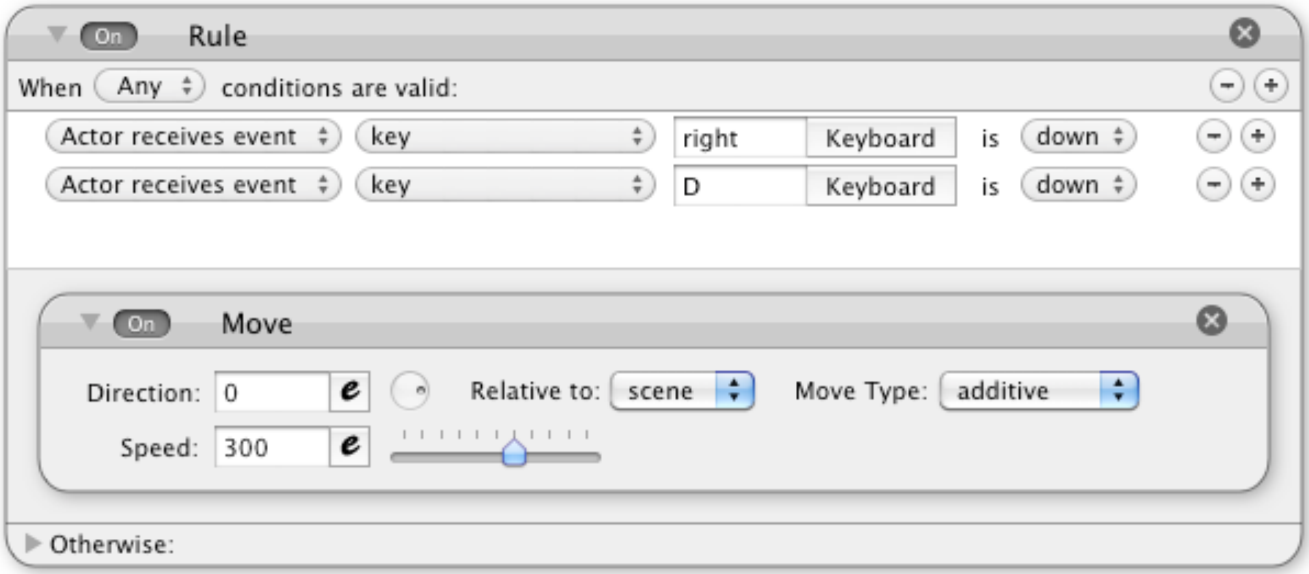


Section III - Building Blocks

7 - Controls
8 - Common Game Elements
9 - Enemies
10 - Setting The Scene

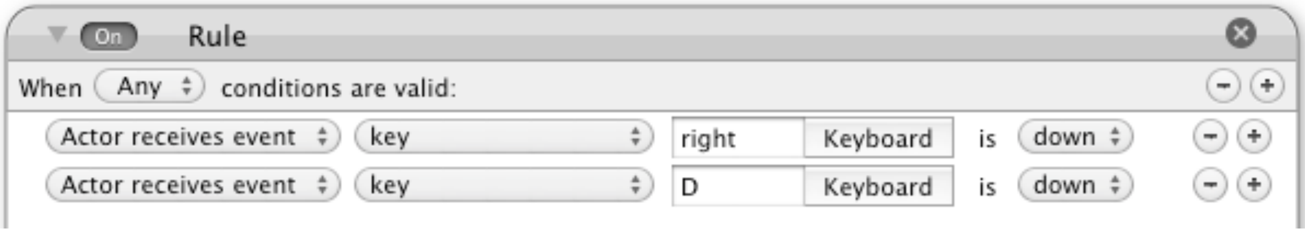
Chapter #7 - Controls

Wow, does your brain hurt? Those last three chapters were tough. But if you managed to understand the concepts in the last three chapters, the rest of the book should be a lot easier for you. A good reason why, this chapter has a lot of pictures. Here's one now…


Rule
When Any conditions are valid:
Actor receives event | key | right | Keyboard | is | down
Actor receives event | key | D | Keyboard | is | down
Move
Direction: 0
Relative to: scene
Move Type: additive
Speed: 300
Otherwise:

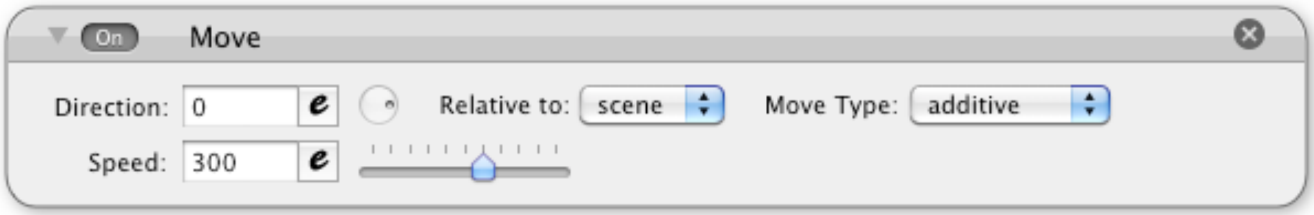
Most games revolve around the movement of the main character. That's what this chapter is about — how to move your main character around the scene. In the image above, that's the most basic of GameSalad controls. Simply create an "Actor", create a "Rule", create the "Conditions" and then create the "Behavior".

When the right arrow, or the "D" key is pressed, the actor will move to the right. The "Direction" is determined by the "Move" behavior. From here, it should be easy to deduce how to create an 8-way controller. But just incase you can't figure it out, here's a closer look.


Rule
When Any conditions are valid:
Actor receives event | key | right | Keyboard | is | down
Actor receives event | key | D | Keyboard | is | down

The "Rule" has two conditions. If the right-arrow key is pressed, or if the "D" key is pressed, the conditions are met. The "Rule" is set to "Any". That accepts either method of control. For a single-player game, it's nice to give players options. Maybe a player doesn't want to use the arrow keys. Maybe the player wants to use the WASD keys instead. It's not that much extra work for you, so you can create a user-friendly game with a little more effort.

The "Move" behavior determines how the actor will move. You set that with the "Direction" value. 0° will move the character to the right. It's up to you to determine the "Relative" setting. You could make it relative to "Actor" or "Scene". For this first experiment, it doesn't matter. The actor is not going to spin. If you have a rotating actor, then you can decide which method makes more sense for your game.

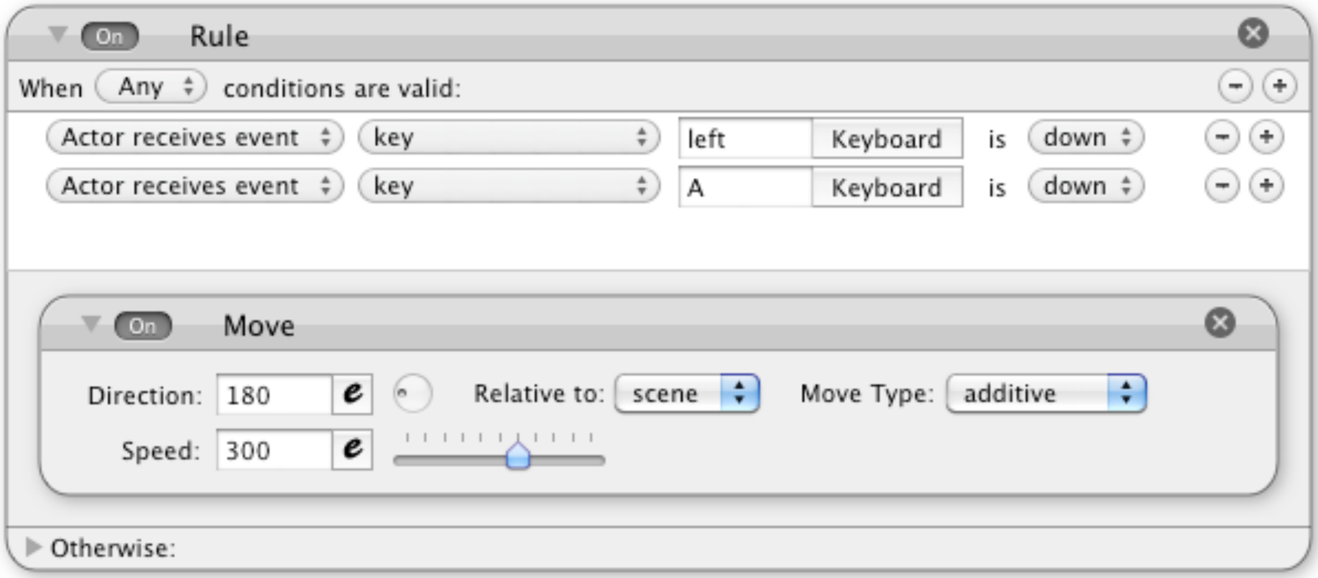


Direction: 0
Relative to: scene
Move Type: additive
Speed: 300

For the other directions, you can simply move the circle icon in the direction that you want the actor to move. The following chart gives the exact directional value for common "Key" and "Move" assignments.

Key	Direction
up or "W"	90°
down or "S"	270°
left or "A"	180°
right or "D"	0°

With this method, you can create something like a "Joystick" for your GameSalad games. The directional movement comes from the combination of two adjacent keys. For example, up and right would move the actor at a direction of 45°.



Rule

When Any conditions are valid:

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

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

Move

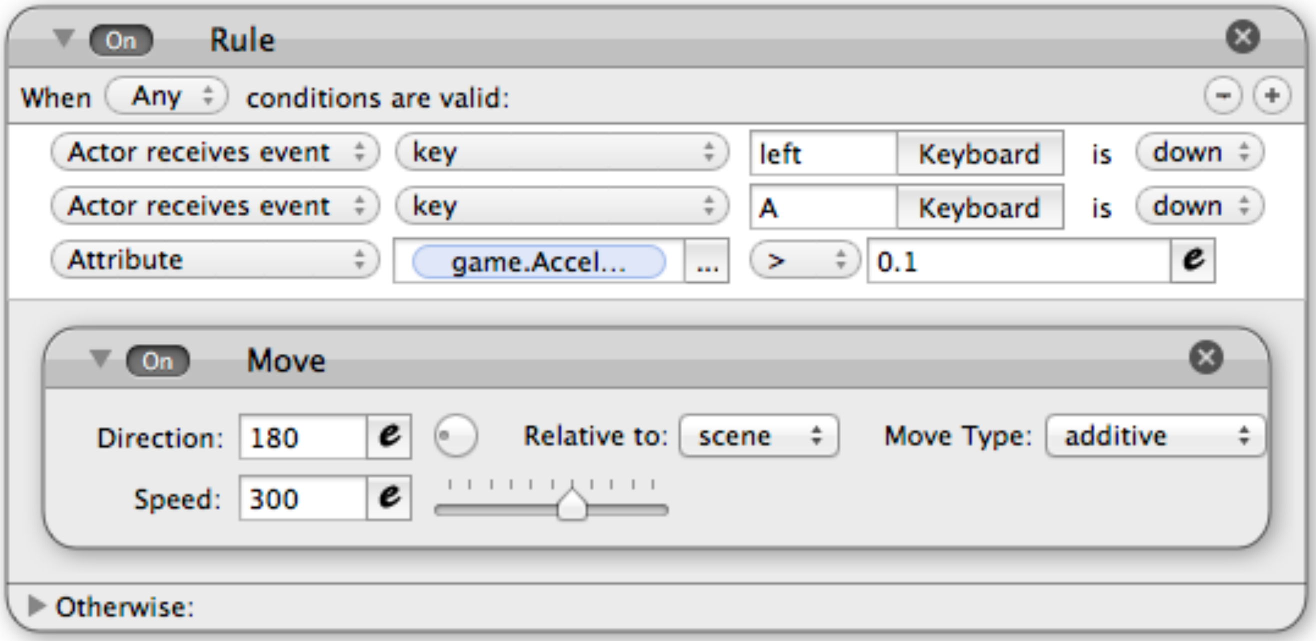
Direction: 180 | Relative to: scene | Move Type: additive

Speed: 300

Otherwise:

The above image shows how to move the actor towards the left. For the next control experiment, only the left and right controls are used. GameSalad is primarily used for making iOS games. That's great, except that the iPhone, iPad and iPod Touch don't come with keyboards. You'll need to be a little bit more creative to create controllers for those platforms. Fortunately, those iOS devices have an accelerometer.

By tilting an iPhone (or similar device) in a certain direction, the hardware detects a change in the direction of gravity. This can be used as a control mechanism for your games.



Rule

When Any conditions are valid:

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

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

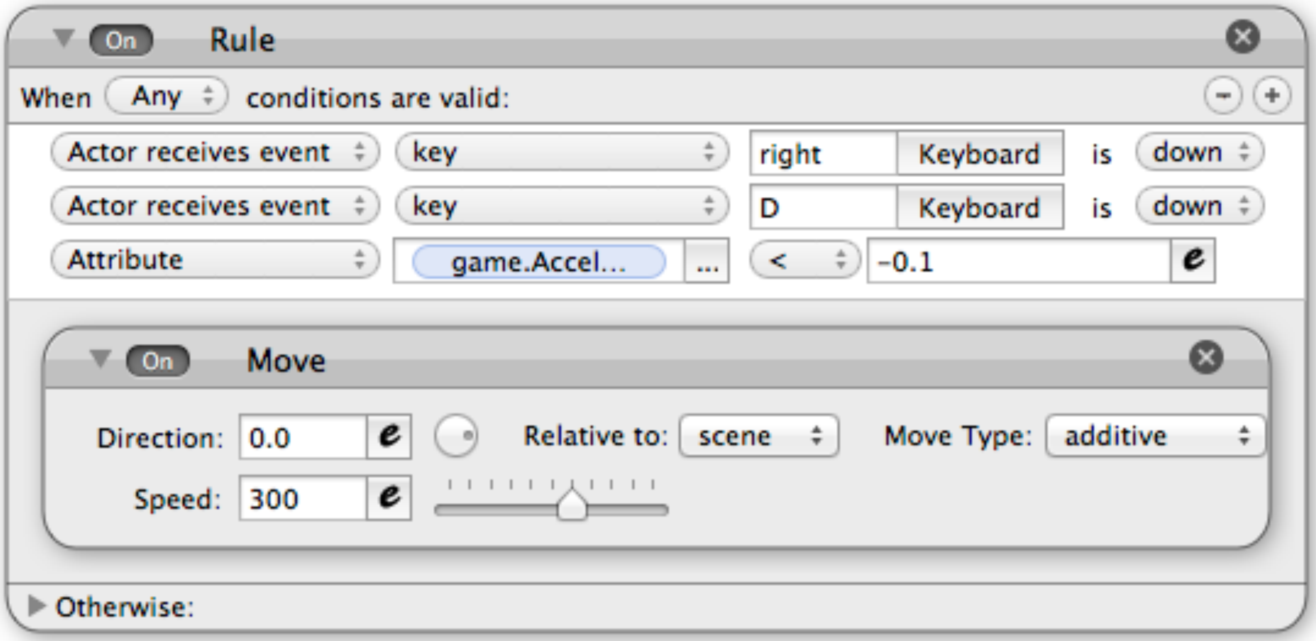
Attribute | game.Accel... | > | 0.1

Move

Direction: 180 | Relative to: scene | Move Type: additive

Speed: 300

Otherwise:


On Rule

When Any conditions are valid:

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

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

Attribute | game.Accel... | < | -0.1

On Move

Direction: 0.0 Relative to: scene Move Type: additive

Speed: 300

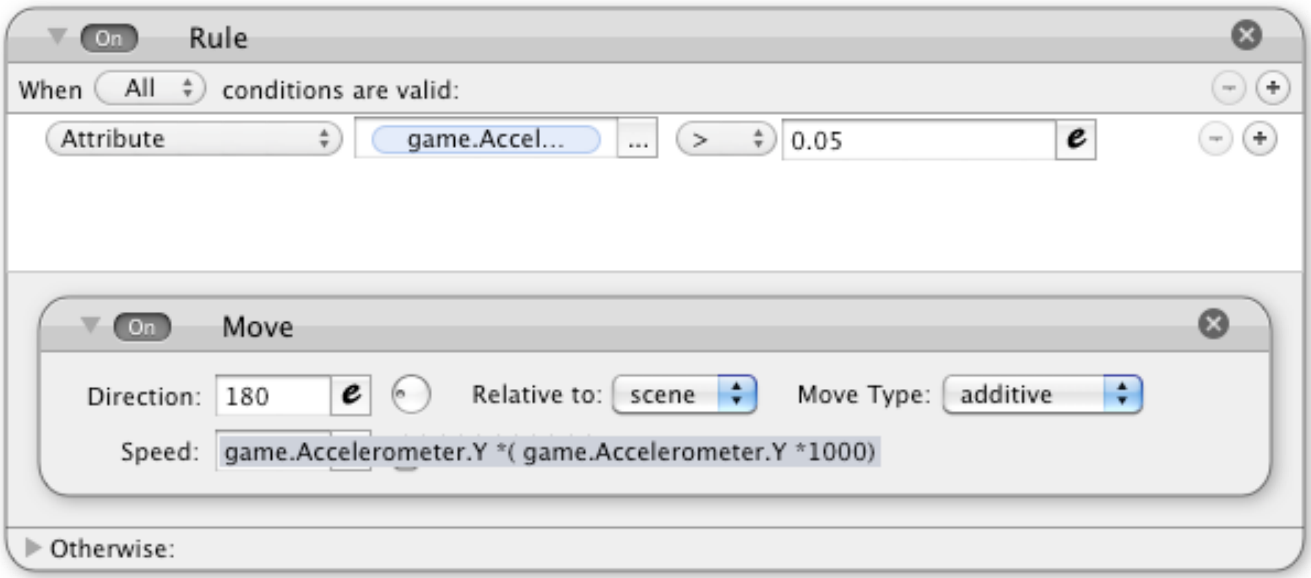
Otherwise:

The two previous images are an example of accelerometer based controls. What's nice about this setup is that it will work with a keyboard or with an accelerometer. This setup assumes that the iPhone's "Home" button is on the right side. The game.Accelerometer.Y attribute has a range of -1 to 1. When the value for Y is greater than 0.1 it moves to the left. When it's less than -0.1 it moves to the right.

You might notice a gap, between -0.1 and 0.1. That's because the accelerometer is a bit jittery. If you tried greater than zero (or less than zero) as your divider, the actor would shake left and right when the iPhone was held parallel to the ground. It's tough working with the accelerometer. That's why I created another formula. The Y value is a decimal. I used that as a multiplier to modify the speed.

game.Accelerometer.Y*(game.Accelerometer.Y*1000)

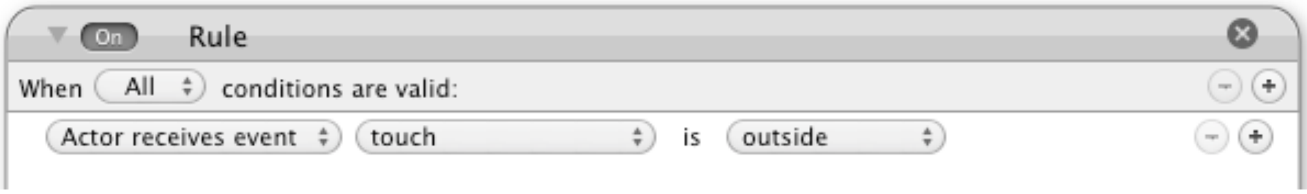
Analog controllers are popular with video games. If you press lightly towards the right on an analog controller, the character moves slowly towards the right. If you press heavily towards the right, the character moves quickly towards the right. With accelerometer controls, and some creative math, you can use the iPhone's accelerometer like an analog controller. By gradually increasing the speed, you can make the movement less jittery. Although, there are drawbacks. If you're modifying the "Speed" variable with an accelerometer attribute (as shown in the previous example) it will not work on platforms without an accelerometer.


Rule
When All conditions are valid:
Attribute game.Accel... > 0.05
Move
Direction: 180 Relative to: scene Move Type: additive
Speed: game.Accelerometer.Y *(game.Accelerometer.Y *1000)
Otherwise:

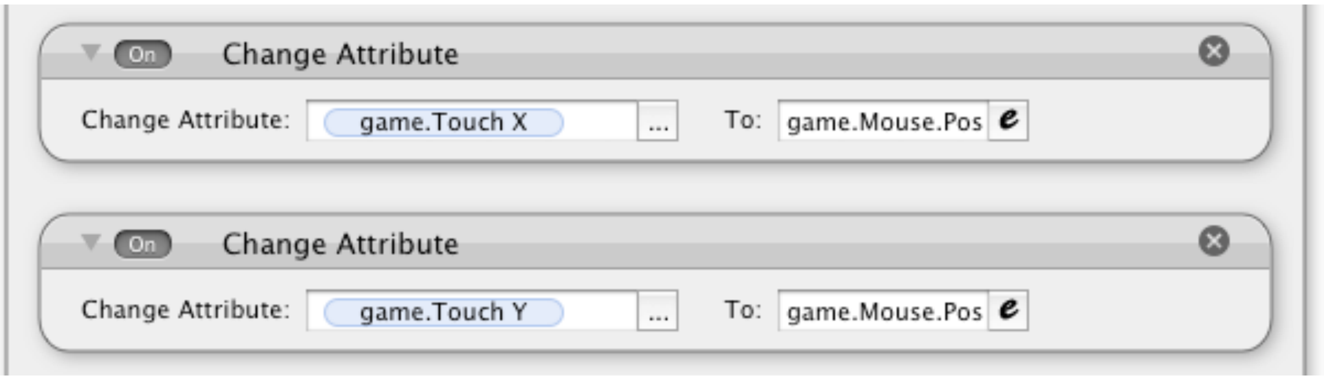
Also, the player really has to move around to play the game, constantly twisting and turning the iPhone. When I design games, I try to imagine the people that will play it. When I imagine this control method in action, I think of a business man. He's in a suit and he's on his way to work. If he's sitting on a train, surrounded by dozens of people, is he really going to want his hands, wrists, arms and elbows flinging around? It could be a funny sight, but it's probably best to give your players more options.

Without a physical keyboard or buttons on iOS devices, what other choices do you have? Lately, I've been building games that can be played with a single touch. Here's one of my favorite methods of control...

First, create a "Rule". Set it to "When All Conditions are valid" and when "Actor receives event". You have some leeway with the next part. A mouse-click is pretty much the same as a touch on an iOS device, so you can select when the "mouse button" is "down" or when a "touch" is "pressed". To create less jittery movement, I selected when a "touch" is "outside". That means if the actor is touched, the "Rule" won't be satisfied. But if a touch happens anywhere else on the screen, that's OK.

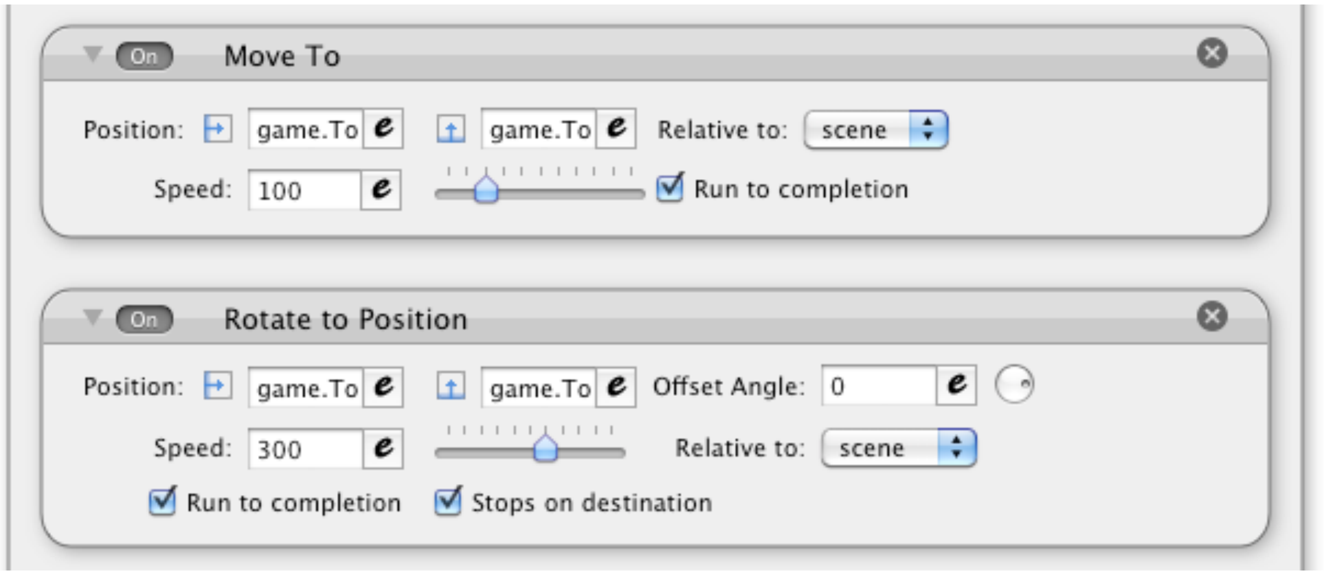

Rule
When All conditions are valid:
Actor receives event touch is outside

The funny thing about "mouse position" or "touch" points, they can change. If you click and hold the mouse, and then drag it around the screen, the "mouse position" will change. In this scenario, I don't want the actor to constantly follow a moving mouse cursor or changing touch point. So instead, the initial value is to be recorded. I created two game attributes, game.Touch X and game.Touch Y. That's the first "Then" part of the "Rule".


On Change Attribute
Change Attribute: game.Touch X To: game.Mouse.Pos
On Change Attribute
Change Attribute: game.Touch Y To: game.Mouse.Pos

In a background actor, I created a "Rule" to record touches. The game.Mouse.Position.X and game.Mouse.Position.Y values are being stored by the game.Touch X and game.Touch Y attributes. Managing touch points can be incredibly complicated on an iOS device. There are 5 on the iPhone and iPod touch. There are 11 on the iPad. That's why I try to limit my game's design to a single touch point. It makes my games easier to manage and compatible with desktop computers. Even though mouse positions are being recorded, that's the same as recording the first touch on an iOS device.

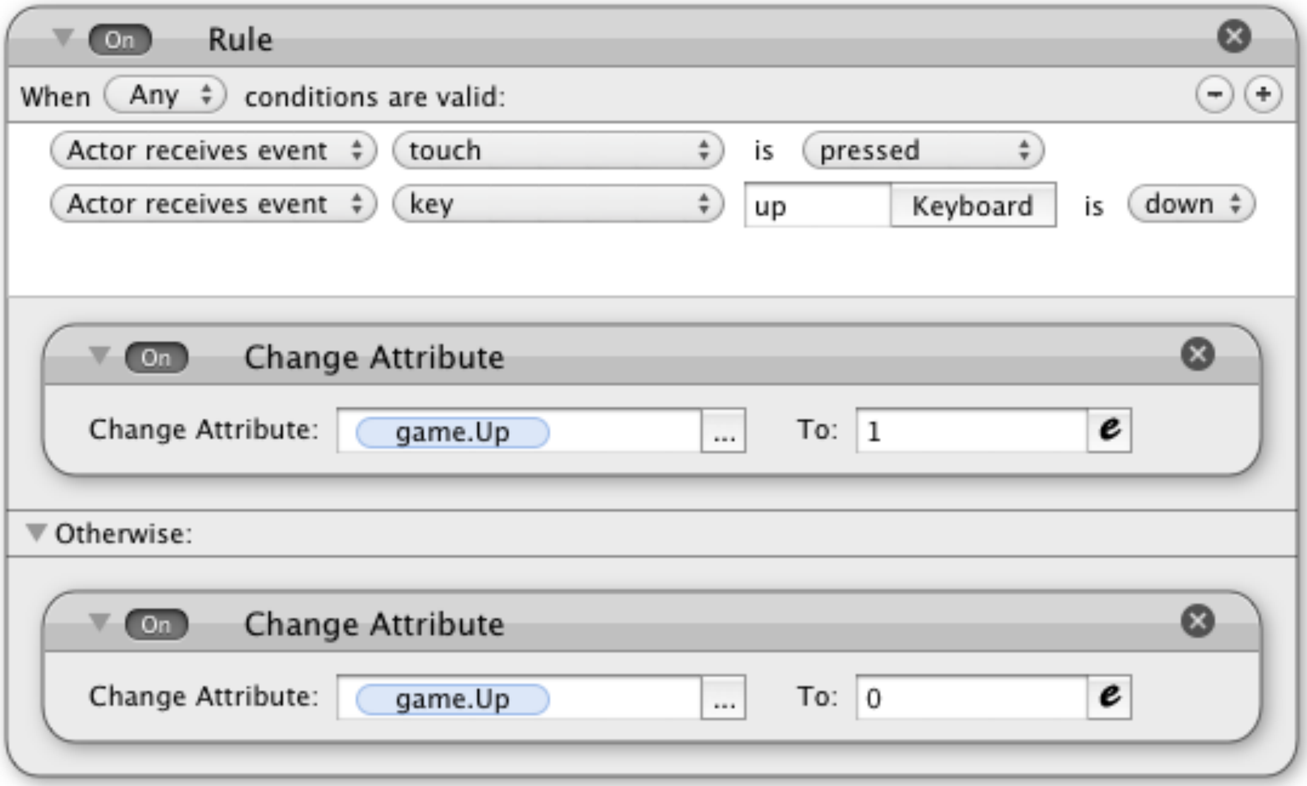
Now here's the awesome part…


On Move To
Position: game.To game.To Relative to: scene
Speed: 100 Run to completion ✓
On Rotate to Position
Position: game.To game.To Offset Angle: 0
Speed: 300 Relative to: scene
✓ Run to completion ✓ Stops on destination

With the "Move To" behavior, the actor will move to game.Touch X and game.Touch Y. With the "Rotate to Position" behavior, the actor will turn and face the direction it is moving towards. This method of control is the basis for my RPG. TANK now has the ability to move around the screen. It's a full 360° range of motion and works on desktops or iOS devices.

The above example works for a top-down game. Instead of having to draw different frames of animation for each direction of movement, I only need to create an image from a bird's eye view. GameSalad automatically rotates the actor towards the direction of movement. Here's how to test it for yourself. Drag an image onto your actor, one that faces toward the 0° direction, and then watch how your actor moves around the screen.

Obviously, the accelerometer or single touches won't cover every type of game. Yet, the iPhone doesn't have any extra buttons for controls. What are you going to do? You can make your buttons.


On Rule

When Any conditions are valid:

Actor receives event touch is pressed

Actor receives event key up Keyboard is down

On Change Attribute

Change Attribute: game.Up To: 1

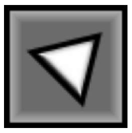
Otherwise:

On Change Attribute

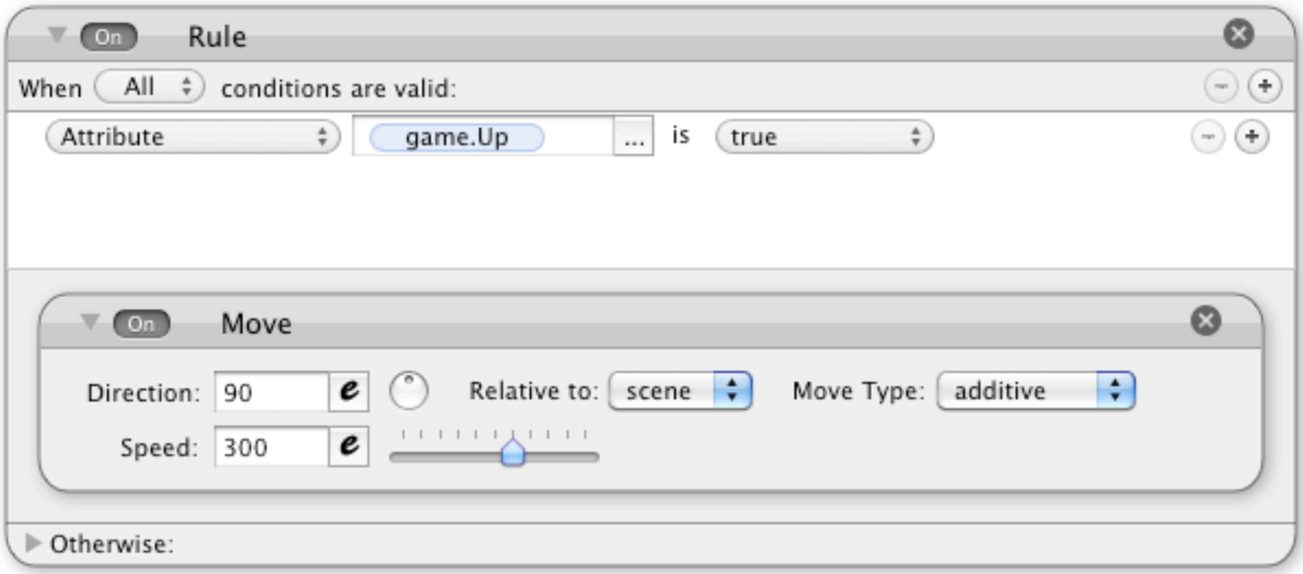
Change Attribute: game.Up To: 0

The above image shows how to turn an actor into a button. First, the game.Up attribute was created. It's a "Boolean" attribute, as the up button only has two states — pressed or not pressed. Next, the "Rule" has two conditions. If the actor is touched, the condition has been met. If the up arrow has been press, that will work too. The latter is more for testing. You don't need to include a "key" for this example. But if you do include more than one way condition for upward motion, then "Any" should be selected in the rule. If "All" was selected in this example, the button would need to be touched and up on the keyboard would need to be pressed. "Any" means that either input method is acceptable — touch or keyboard.

If the condition has been met, then the game.Up attribute will be changed to "true". If the conditions have not been met, the game.Up attribute will be changed to false. (In the above example, I was too busy to type "true" or false, so I just used the numbers 1 and 0.) You could duplicate this process for each of the buttons you need to create. You only need one graphic. Just create an arrow that is pointing at a 45° angle. Place that image on the up actor, and then rotate the actor 45° into a diamond shape. That process will create an up arrow, as shown in the left image. The other actors (left, right and down) can be created in a similar fashion. Rotate the other actors to their proper angles and then align the actors together to form a "Touch Pad" controller. For a less obtrusive look, you can set the controller to a transparent color. Just adjust the "Alpha" channel for each actor.


The image to the left is a look at the controller image — in it's raw form. That one graphic could be reused three additional times to create a four-way controller. It's important to think in such a manner to optimize the performance of your game. Also, this speeds up your development time. Instead of having create four separate graphics, only one graphic is needed.

Now that the buttons have been created, another actor needs to monitor (and react to) changes in the movement related attributes.

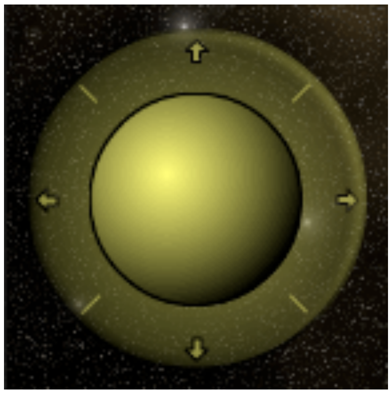


In the above example, the actor will move up when the game.Up attribute is true. This combination of "Rules" and game "Attributes" enables the actors to share data. It's like they're communicating with each other. That's how you can get your actors to interact with other actors. In addition to directional buttons, you could create other types of buttons. If you want to make a fire button or a jump button, the process is essentially the same. When one actor is touched, it can change an attribute. Another actor monitoring that attribute can react accordingly.

This method could also be used to make title screens and a user interface. Each button on the title screen, or user interface, could be related to a specific action. If you want to skip the scene, pause the music, change a game setting, buttons can help make it happen.

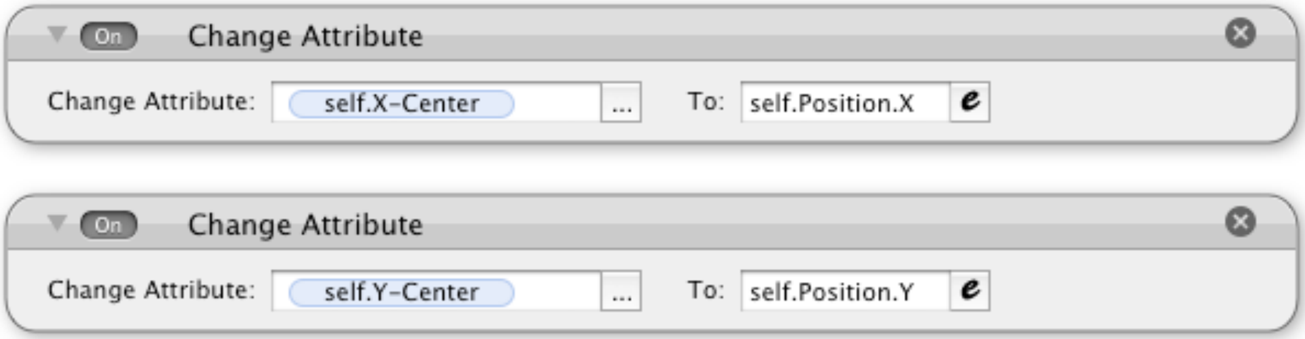
As an example, clicking on a flag could work like clicking a button. By pressing the corresponding flag, the game's language settings could be changed. Separate actors aren't always necessary though. By using additional rules, an event can be triggered if the touch is greater than / less than a certain location.

You can take your on-screen controls one step further, but it is significantly more difficult. You could create a "Thumbstick". It is excellent method for an analogue controller that's on-screen. Yet, it's far more involved to create and it requires additional processing power.



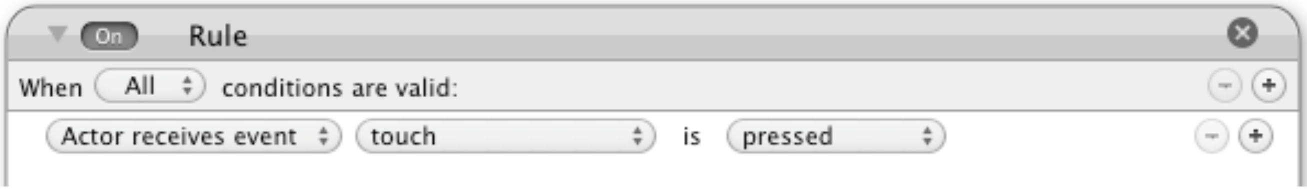
In the image to the left, there are three actors shown — the background, the controller's base and the controller. If the controller is touched, the controller follows the movement of the touch. Through "vectorToAngle" and "Magnitude", we can calculate the angle and the distance from the center of the controller's base. Those values can then be sent to an actor, informing it of which direction to move and how fast it should move. To simply things, here's how to do it with just one actor.

First, grab the actor's initial X and Y positions. Since the controller's center point is probably not going to be the same as the scene's origin (X=0,Y=0) point, an offset is needed. Otherwise, "vectorToAngle" and "Magnitude" will not function properly. To accomplish this, create two actor attributes, one for the initial X location and another for the initial Y location.

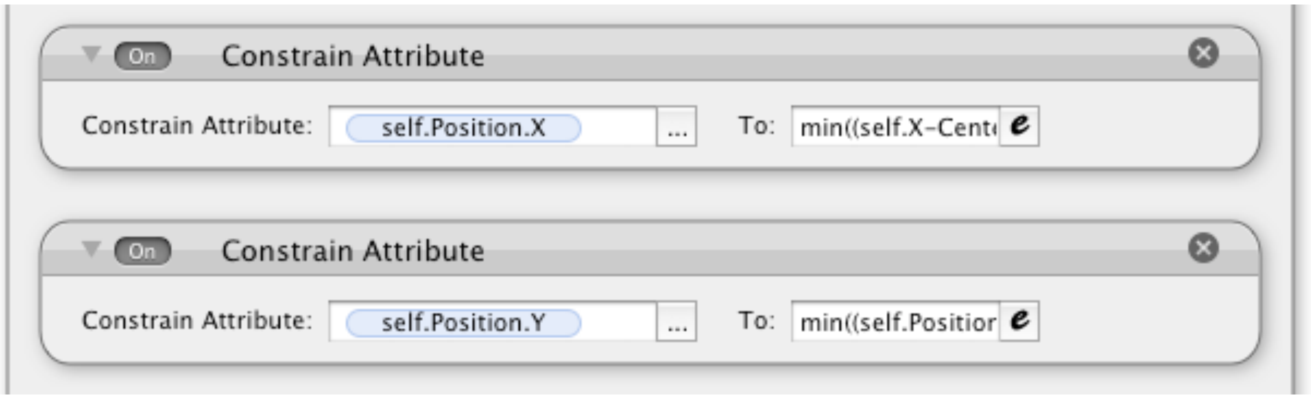

Change Attribute
Change Attribute: self.X-Center To: self.Position.X

Change Attribute
Change Attribute: self.Y-Center To: self.Position.Y

In this example, the two actor "Attributes" are called self.X-Center and self.Y-Center. With the initial location of the controller recorded, the next set of "Behaviors" can function properly. The advantage of using the actor's initial X and Y positions, instead of manually entering a number, allows the repositioning of the controller. When creating your game, it helps to keep things dynamic. Otherwise, if you decide to change your game's layout, you might run into tedium with editing the behaviors.


Rule
When All conditions are valid:
Actor receives event touch is pressed

The next "Behavior" is a "Rule". The thumbstick should not function unless it's touched. If it is touched, then we want the controller to move. That's the task for the next two "Behaviors".

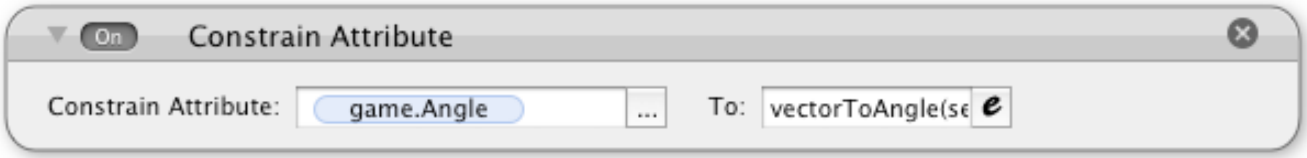

Constrain Attribute
Constrain Attribute: self.Position.X To: min((self.X-Cent

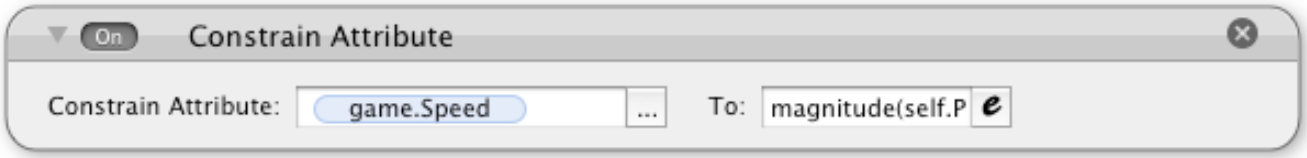
Constrain Attribute
Constrain Attribute: self.Position.Y To: min((self.Positior

If the actor is touched, it should move to the mouse location. Yet, we don't want the controller flying all over the screen. It should be confined to a specific location. That's what the next two expressions can accomplish.

min((self.X-Center+30),max(game.Mouse.Position.X,(self.X-Center-30)))
min((self.Y-Center+30),max(game.Mouse.Position.Y,(self.Y-Center-30)))

By using the "min" and max" functions, the controller stays inside a specific range. That's what the number "30" is about. It creates an invisible square, one that the controller cannot move beyond.

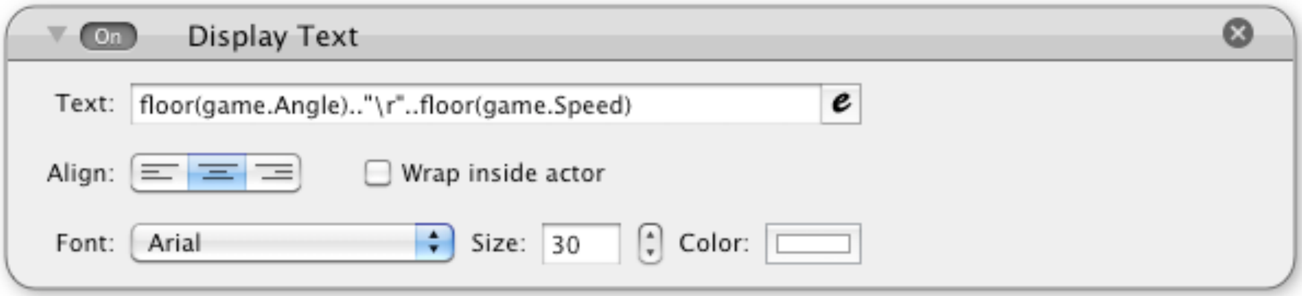

Constrain Attribute
Constrain Attribute: game.Angle To: vectorToAngle(se


Constrain Attribute
Constrain Attribute: game.Speed To: magnitude(self.P

With the controller able to move around, the data can be recorded. Here are the two formulas to calculate the controller's angle and distance from the center point.

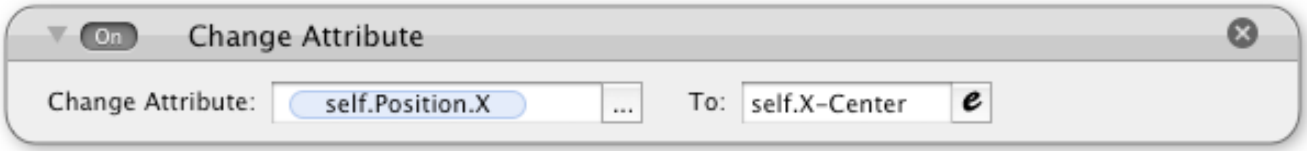
vectorToAngle(self.Position.X-game.X-Center,self.Position.Y-game.Y-Center)
magnitude(self.Position.X-game.X-Center,self.Position.Y-game.Y-Center)

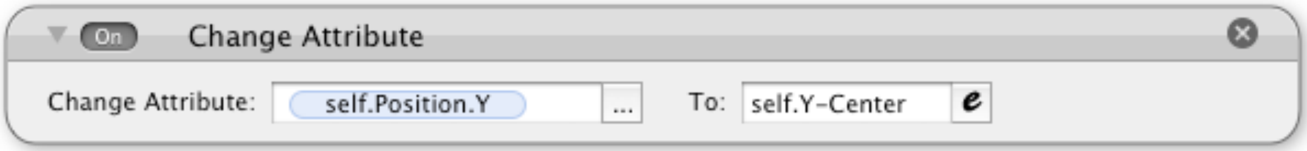
If you like to see what's going on with your controller, you can use "Display Text" and the following formula to visualize both the angle and the speed values.


Display Text
Text: floor(game.Angle).."\r"..floor(game.Speed)
Align: center
Wrap inside actor: unchecked
Font: Arial Size: 30 Color:

floor(game.Angle)..“\r”..floor(game.Speed)

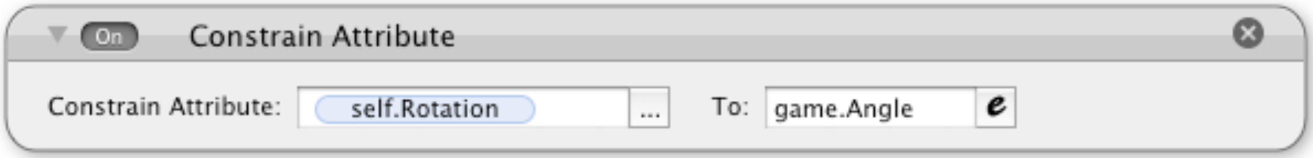
When you let go of the controller, you might want it to snap back to the middle. To create that effect, you can simply use the “Otherwise” area of the “Rule” to reset the actor.

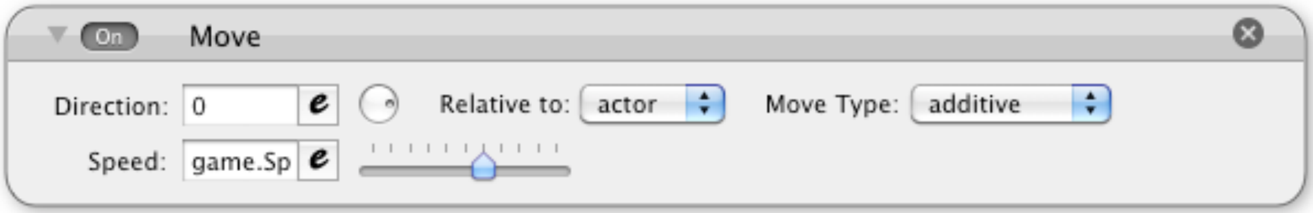

Change Attribute
Change Attribute: self.Position.X To: self.X-Center


Change Attribute
Change Attribute: self.Position.Y To: self.Y-Center

We're done right?! Wrong, this actually creates a slew of problems. For one, the speed values are significantly higher if the actor is moving diagonally. That's because the limiters on the controller only work horizontally or vertically, not diagonally. You can fix that with "min" and "max" functions on your "magnitude" value.

Also, the actor needs to read the values. Fortunately, that's not too hard at all.

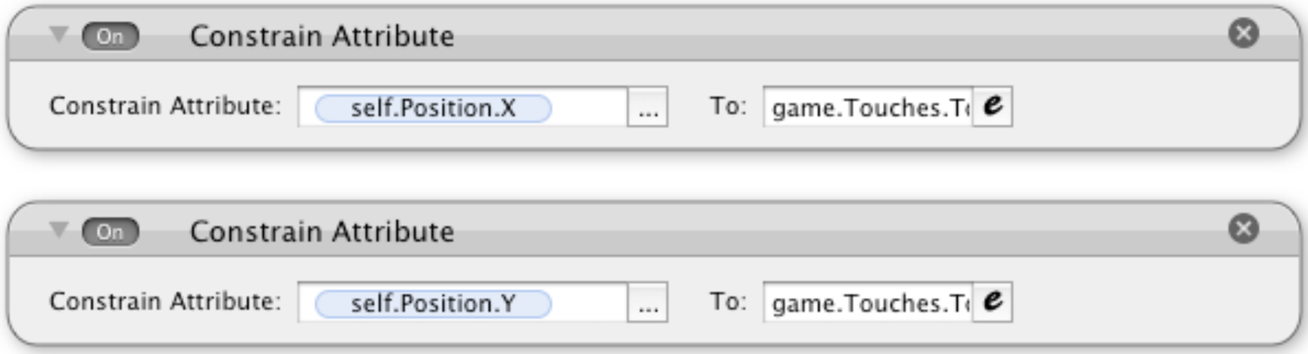

Constrain Attribute
Constrain Attribute: self.Rotation To: game.Angle


Move
Direction: 0 Relative to: actor Move Type: additive
Speed: game.Sp

Like with other examples, the actor's image should face towards the 0° direction. That way, when the self.Rotation value is changed (by the game.Angle value) it will appear that the actor is flying in the direction it is moving. The movement is controlled with a "Move" behavior. The direction should be 0° and it should be "Relative to" the "actor". In this example, the actor only moves in the direction it's facing. The last part is controlling the "Speed" of the movement. That's where you can use the game.Speed value. If the actor isn't moving fast enough, you can use a multiplier to speed things up.

Another problem, the actor can keep moving even though the controller is no longer being pressed. To handle this issue, you could create another game "Attribute" for controller activity. A "Boolean" should do the trick. If the controller is touched, then the value is "true". If the controller is not touched, then the value is "false" and nothing happens. A "Rule" stop movement if the value is "false".

There... we have perfect on-screen controls now, right? Wrong! Unless you're building a game where no additional controllers or buttons are needed, there will still be issues. For example, if you had a fire button, your controller wouldn't work if the fire button was touched first. The mouse position values are confined to the first touch point. To handle this issue, you need to keep track of the touch points. You can do this with invisible actors, one for each touch point — all five (iPhone) or 11 (iPad) of them. By constraining a sensor actor to the location of each touch point, overlaps with controllers can be detected.


Constrain Attribute

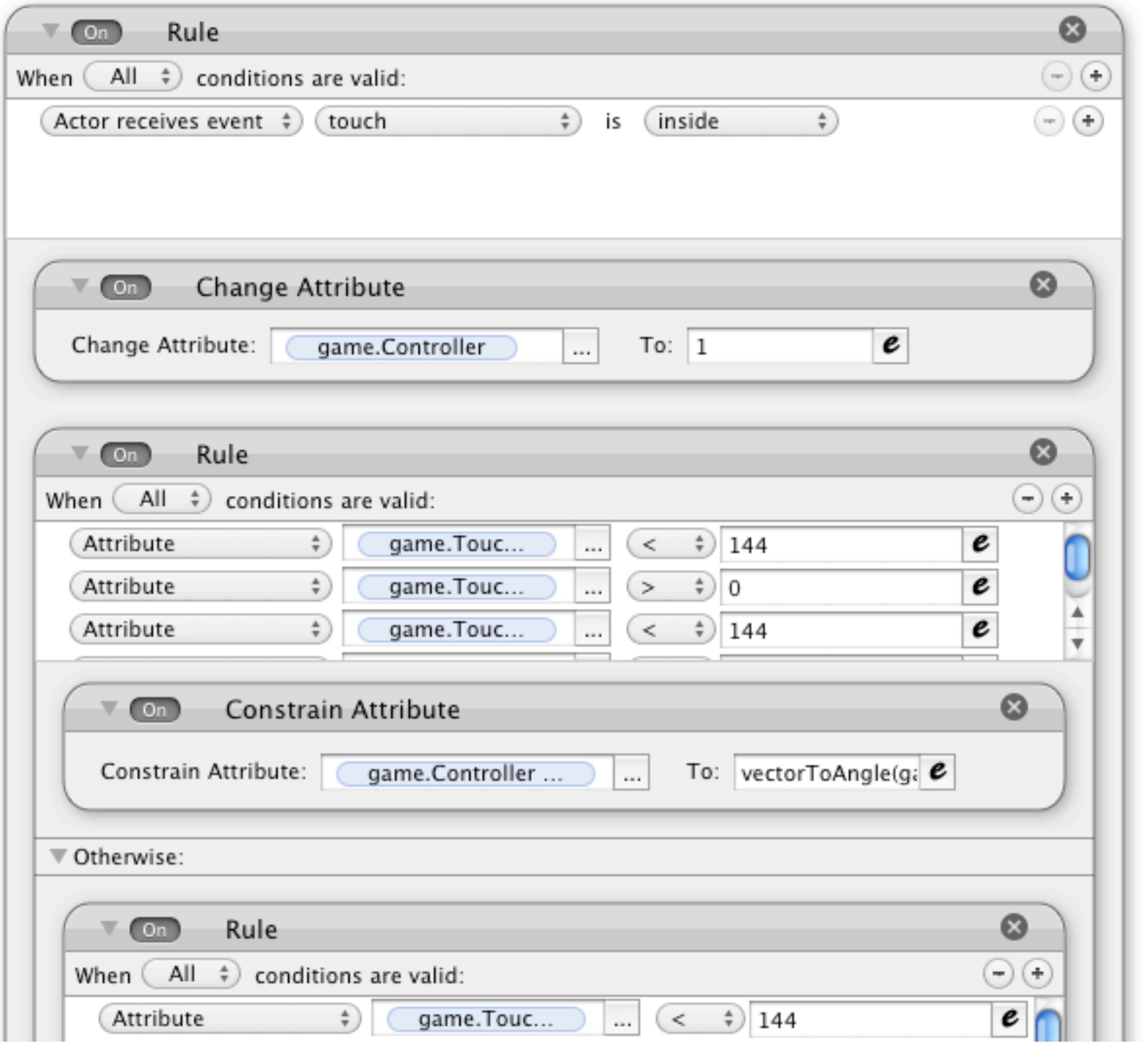
Constrain Attribute: self.Position.X To: game.Touches.T

Constrain Attribute

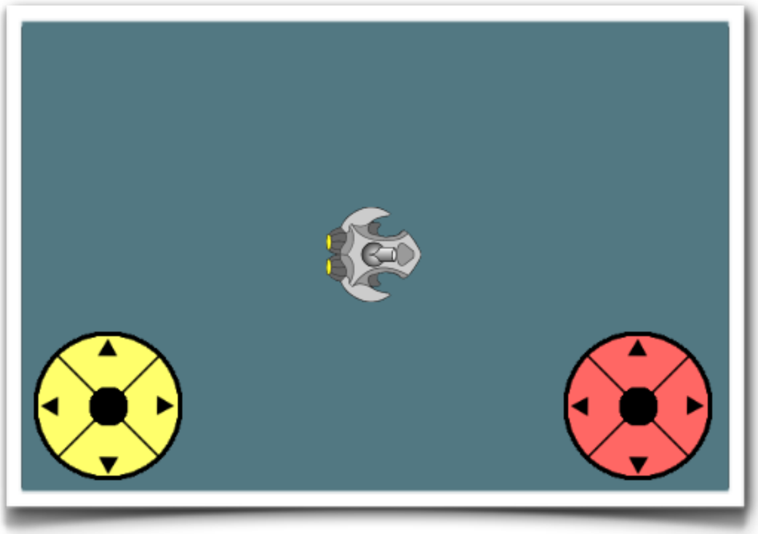
Constrain Attribute: self.Position.Y To: game.Touches.T

The above "Behaviors" constrain an actor's position to a specific touch point. For example, an actor named "Touch Sensor 1" would have it's position tied to the game.Touches.Touch 1.X and game.Touches.Touch 1.Y locations. The thumbstick would then have to react to the specific touch point that it is in contact with. If "Touch Sensor 2" is overlapping the thumbstick, then its location should be used for the thumbstick's expressions. Otherwise, the unused sensors should be sent off the screen. That adds a lot of extra "Behaviors" to your game. It could dramatically hurt performance. Plus, your game's desktop compatibility is lost.

Supporting multiple touch points adds a lot of extra work to your project and it can significantly decrease performance of your game. My own experimentation with dual controllers lead to disappointing results. After a while, the controllers would get stuck. But if you simply need to support touch points in your game, the "Official Cross-Platform controller Template" in the GameSalad launch window is a good place to start. The template shows how to manage multiple touch points or to simply create a D-Pad controller.


Rule
When All conditions are valid:
Actor receives event touch is inside
Change Attribute
Change Attribute: game.Controller ... To: 1
Rule
When All conditions are valid:
Attribute game.Touc... ... < 144
Attribute game.Touc... ... > 0
Attribute game.Touc... ... < 144
Constrain Attribute
Constrain Attribute: game.Controller ... ... To: vectorToAngle(g
Otherwise:
Rule
When All conditions are valid:
Attribute game.Touc... ... < 144

But to make things easier and more efficient for my RPG, I created a different way to manage the controls. Basically, by using rules, I'm detecting if a touch is within a square. If the touch is within the square - and the thumbstick is touched — then control the character. With the “Otherwise” portion of the rule, other touchpoints are detected. So if touch point 1 doesn’t meet the criteria, check touch point 2 or 3 and so on. This allows a single actor to be used as a virtual controller. Particles, based on the position of the active touch, could be used to create a moving thumbstick effect. With lots of touchpoints to track, your rules might start looking like the pyramids in Egypt. That doesn’t bother me. I like this method better, as my thumbsticks stopped jamming.



If the description of the unofficial dual-stick controls is difficult for you to visualize, the "Controls" template uses this method. The fighter can be moved with the left thumbstick and the turret is controlled with the right thumbstick. The "Rules" do most of the work, eliminating the need for five invisible sensor actors.

If you take the hard road, you can gain new control options with multi-touch. For example, you could have two invisible actors that work with multi-touch. The distance between those two actors could control the camera's zoom. If the actors are pinched closer together, zoom out. If the actors are pulled together, zoom in — just like the Safari web browser in iOS. You can gauge the distance between the two actors with the "magnitude" function. That variable could be used to change the camera's size. With those same two actors, you could also rotate the camera. With the "vectorToAngle" function, you can use the angle between the two invisible actors to modify the camera's "Rotation" value.

But again, you may not need something that fancy. You could simply create a zoom button and dramatically optimize your game. As for rotating the camera, I messed around with that and I got a bad case of vertigo. The screen spun around little too fast for me and it made me feel terrible. You might want to be careful with camera rotation!

It's probably a good idea to think of your game's controls well in advance. Even with the best graphics and sound effects in the world, if players can't play your game they probably won't like it.

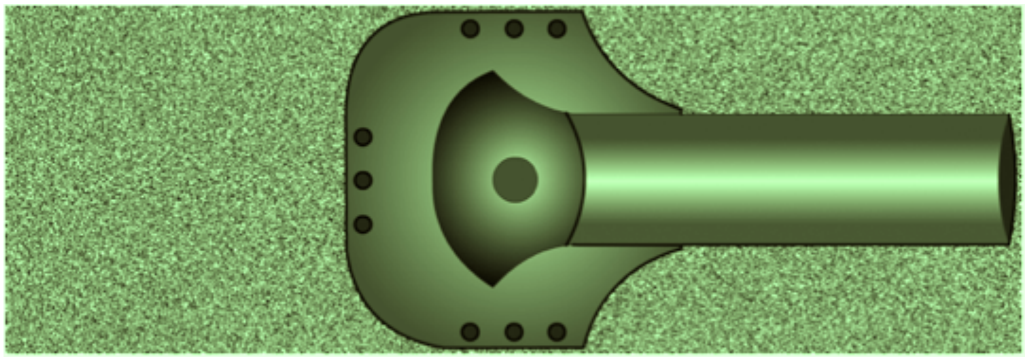
Chapter #7 Summary

- A “Rule” could be set to accept “Any” condition, allowing you to map a single “Move” behavior to multiple keys and/or conditions.
- The accelerometer, on an iOS device, can be used as a controller.
- Using a “touch” condition on a “Rule” can turn an actor into a button.
- Use “Attributes” to share data between actors.
- An 8-way on-screen control pad can be created from four actors and a single image.
- A mouse click is treated as the first touch point on an iOS device. Restricting your game to a single touch point allows it to be compatible with the iPhone, iPod Touch, iPad and desktop computers.
- The “magnitude” and “vectorToAngle” functions can be very useful with joystick based controls.
- An iOS game can have multiple touch points. Some on-screen control methods simply require multi-touch support, such as dual controllers and pinch gestures.

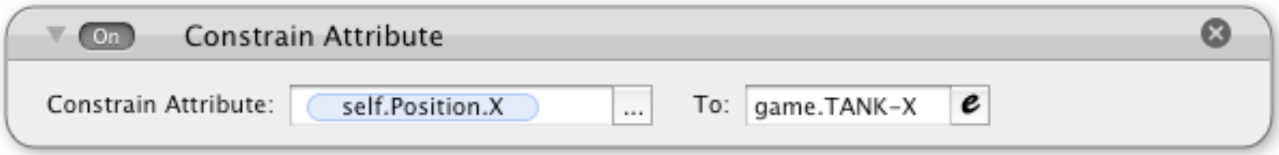
Chapter #8 - Common Game Elements

Now that you can create a main character for your game, and move it about the screen, it should be able to do stuff. That's what this chapter is about, common elements found in video games.

Shooting - This is a highly common activity in a video game, so that can be the first example. With GameSalad, it's actually pretty simple. You can use the "Spawn Actor" behavior to place a bullet actor in the screen. Then, that actor does the rest of the work. So, I'm going to add a little bit extra to this example. The following is how to build a tank turret.

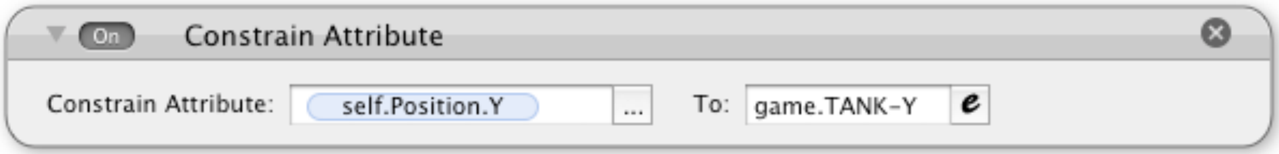


Obviously, you'll need a tank turret graphic. The above image is what I created for one of my games. There are key features to notice. One, the TV-like static is not part of the final image. I put that noise in there on purpose. The transparent areas of your image matter. Do you see that big circle in the middle? That dot represents the center point. The actor is going to be a rectangle, with a large empty space on the left side. That way, when the actor rotates, it spins on the center point. Also, the turret points toward the 0° direction. Those are two important concepts to consider when creating your images. If the image is going to spin, which direction is it going to face and where is the center point? In the above example, those two issues have already been addressed. That makes it easier to implement the following steps.



Constrain Attribute

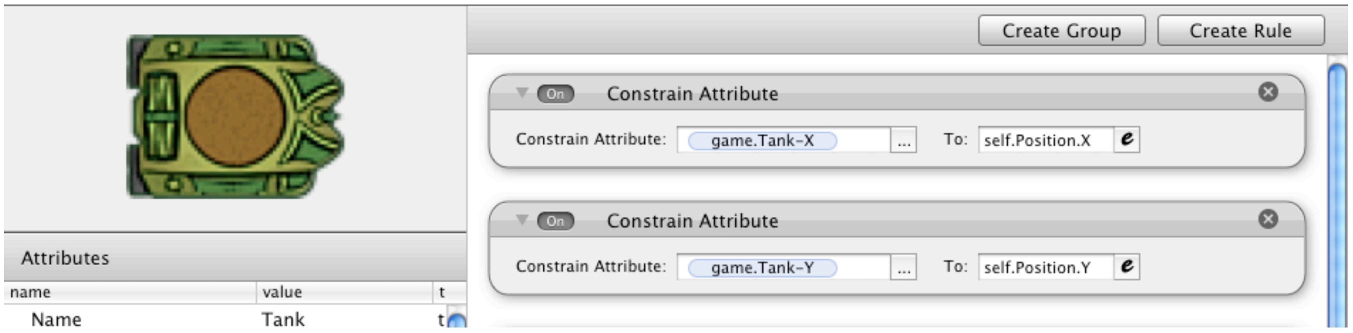
Constrain Attribute: self.Position.X To: game.TANK-X



Constrain Attribute

Constrain Attribute: self.Position.Y To: game.TANK-Y

Using two constrain "Behaviors" the turret is digitally welded to the tank. The X & Y values of the turret actor will be the same as the X & Y values for the tank actor. That's why it was important to have a clearly defined center point. It makes it easier to align the actors.

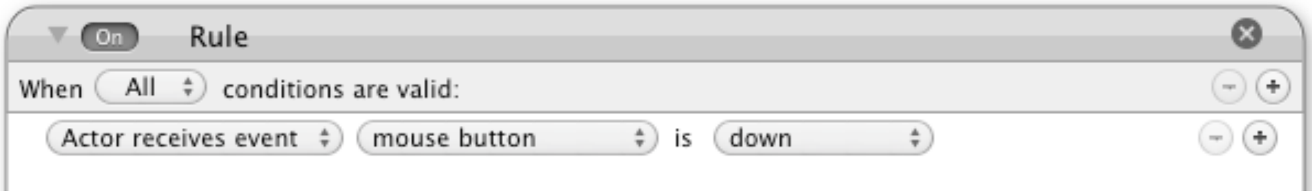


Constrain Attribute
Constrain Attribute: game.Tank-X To: self.Position.X

Constrain Attribute
Constrain Attribute: game.Tank-Y To: self.Position.Y

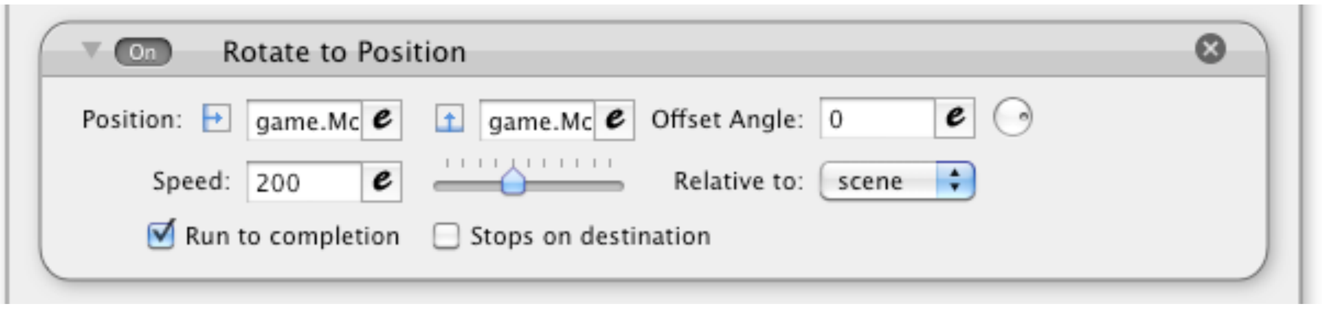
Name Tank

The tank actor needs to constrain the game.Tank-X and game.Tank-Y "Attributes" to its location. The tank actor feeds information to the turret actor. That's how the turret knows where it should be. The next steps are for the turret.



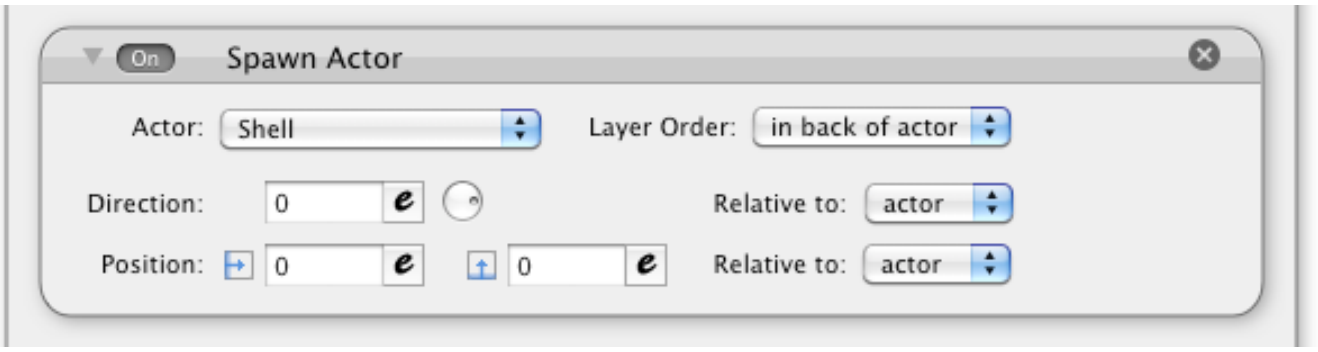
Rule
When All conditions are valid:
Actor receives event mouse button is down

When the mouse is down, the turret is supposed to shoot. You could change this "Rule" to match your own control scheme. For example, if you had a shoot button, it could change a custom "Attribute" to "true" when the button is pressed. If the value of game.Shooting is "true" then run the turret's "Behaviors".



Rotate to Position
Position: game.Mc game.Mc Offset Angle: 0
Speed: 200 Relative to: scene
☑ Run to completion ☐ Stops on destination

This part makes the turret spin. You could use your own method. This one uses the mouse location for targeting.

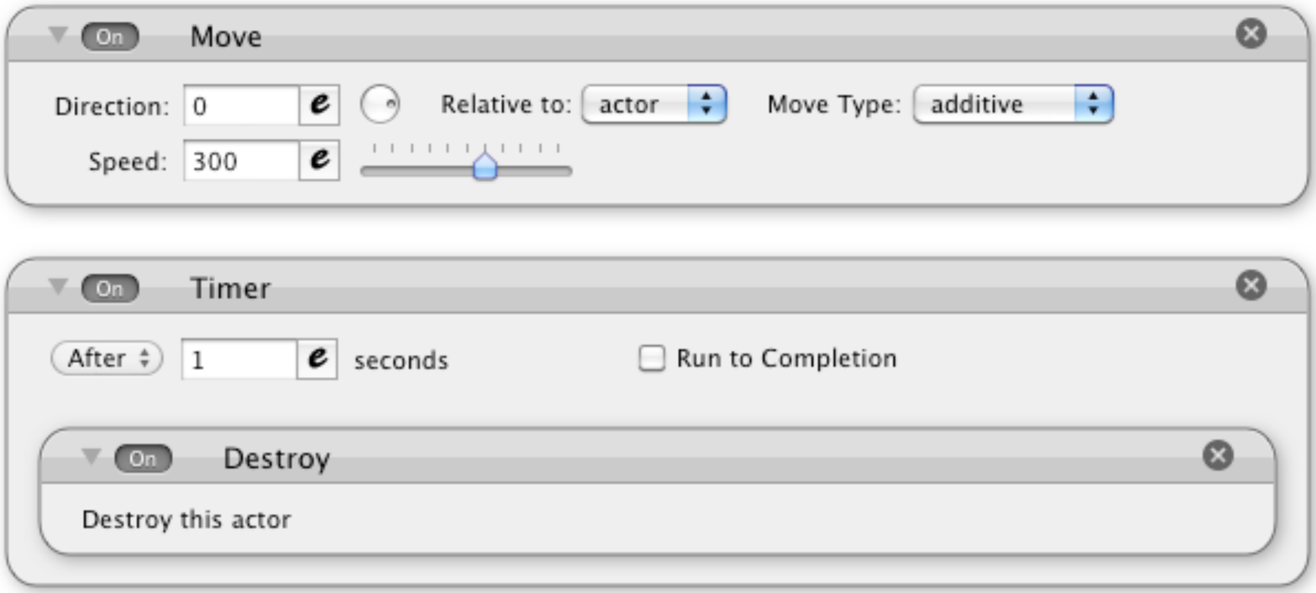

Spawn Actor

Actor: Shell
Layer Order: in back of actor
Direction: 0
Relative to: actor
Position: 0 0
Relative to: actor

Now the fun begins. This is how the bullets are made. When the mouse is pressed, a bullet actor is spawn. At initial glance, it might look like nothing special is happening here, but there are two important things to notice. First, the "Layer Order" is set to "in back of actor". That spawns the bullet actor underneath the turret. That order creates a more realistic effect. Otherwise, the bullet will appear above the turret. By spawning the bullet actor below the turret, it seems to be traveling through the turret. To help maintain this effect, the bullet actor should not be larger than the width of the turret.

The other important thing to notice is the "Relative to" setting. For both occurrences, it was set to "actor". That's because the bullet should spawn from the turret, not anywhere else on the screen. And when the projectile flies out of the turret, the angle of the bullet should match the turret.

The bullet's job is pretty simple. It's just supposed to fly straight. It can do this with a "Move" behavior.


Move

Direction: 0
Relative to: actor
Move Type: additive
Speed: 300

Timer

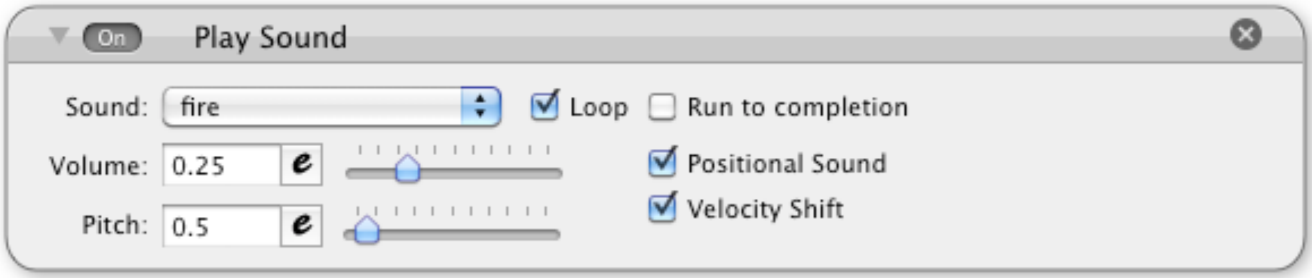
After 1 seconds
Run to Completion

Destroy

Destroy this actor

The timer is to ensure that the bullet doesn't fly around forever. Off-screen actors can still be in the game, eating up precious CPU power. By destroying a bullet soon after it is spawned, you can help maintain the performance of your game and create a more believable bullet.

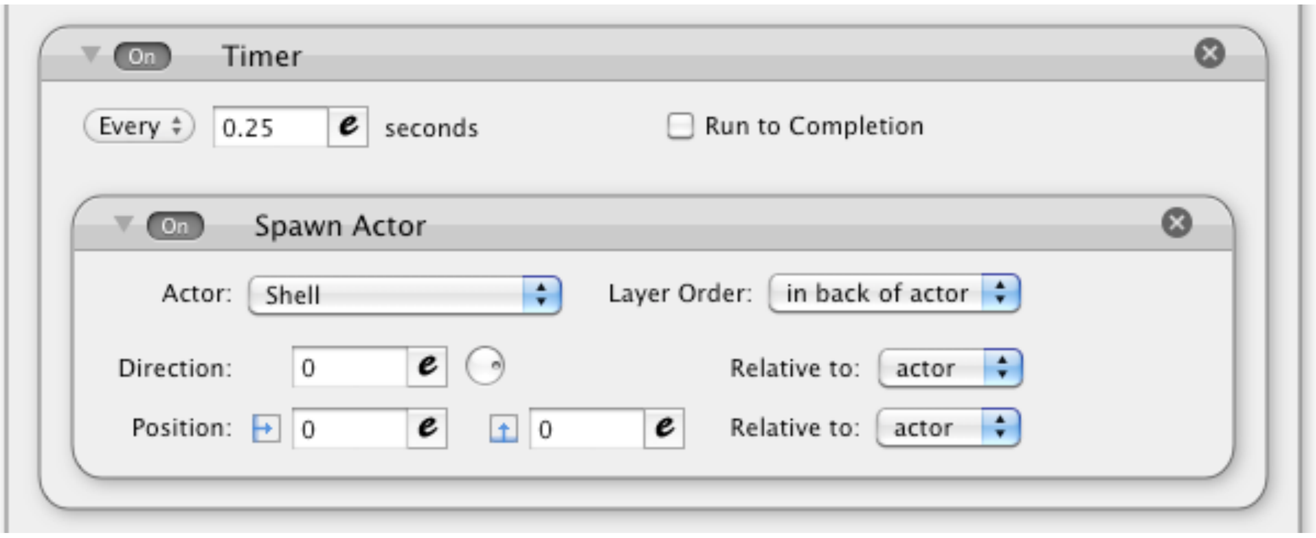
There are other things you can do to make your bullets more believable. You could also destroy your bullets when they overlap with another actor. You can use a "Rule" to make that happen. You could also add sound effects.



Sound: fire
[✓] Loop
[ ] Run to completion
Volume: 0.25
Pitch: 0.5
[✓] Positional Sound
[✓] Velocity Shift

I like to keep my sound effects on the bullets themselves. That allows me to use the inherent audio effects of the "Play Sound" behavior. To prevent problems, I keep "Run to completion" unchecked. The bullet actor is going to be destroyed. In this scenario, I don't need to keep playing the sound effect if the actor is no longer in the game.

If you really want to keep your game streamlined, only allow one bullet to be fired at a time. Then, you don't have to spawn and delete the bullet. You only have to change its position or hide it off-screen. I know that's a more optimized way of managing a game, but it's simply not as fun. If you're going to put a gun in a game, you might as well let the player shoot.



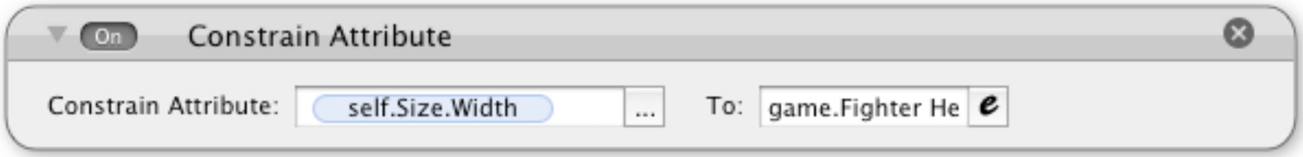
Every 0.25 seconds
[ ] Run to Completion

Actor: Shell
Layer Order: in back of actor
Direction: 0
Relative to: actor
Position: 0 0
Relative to: actor

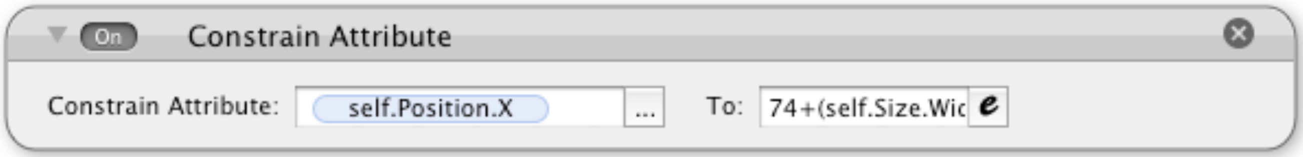
You can add a timer to the turret actor's firing rule. That will enable auto-fire while the mouse button is held. With this setting, the player doesn't have to click like crazy. Although, a bullet will be spawned with each new press. Players can still click like crazy if they want. It's good to give them options. However, be careful with this setting. If too many bullets are spawned, your game can slow down dramatically.

That can be a lot to remember. If you need a more visual guide for building shooting games with GameSalad, the "Shooting" template might be able to help. It features the Tank and Turret system. You might not be a fan of the controls. But if you've made it this far in the book, you should be able to make your own customizations.

Health Bar - If you're going to have your character shoot, other characters might be shooting back. So, if you want your hero to be more than a one-hit wonder, you can add a health bar to your game.



Constrain Attribute: self.Size.Width To: game.Fighter He

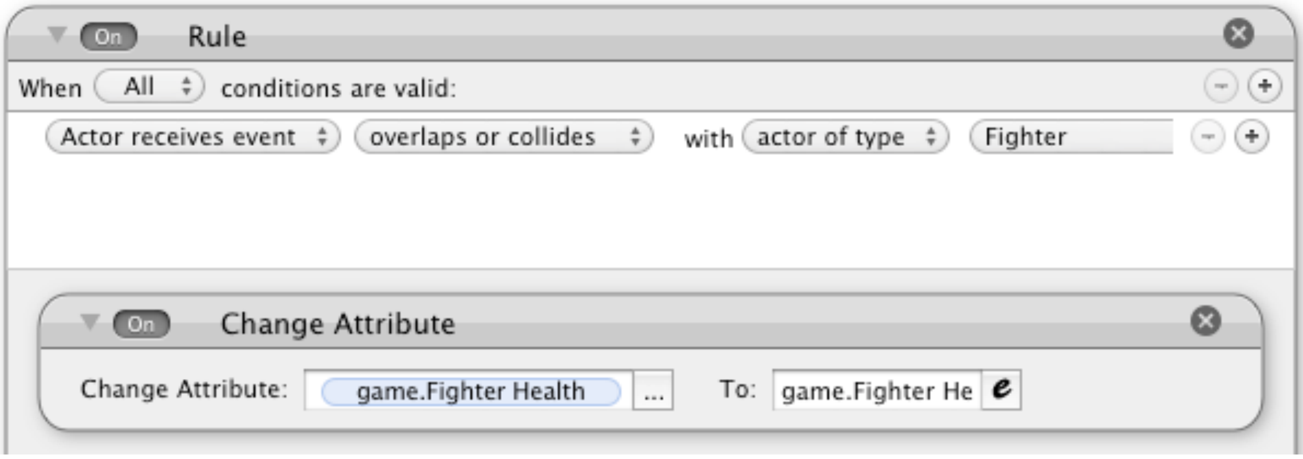


Constrain Attribute: self.Position.X To: 74+(self.Size.Wic

The above image shows the basics of a health bar. The value for game.Fighter Health is controlling the size of the health bar. 100 health equals a 100 pixel wide bar. Instead of 1:1, you could modify the ratio with math. The position of the bar is being constrained too. Generally, health bars fill from left to right. To recreate that effect, the health bar has to change its location. As the bar expands and contracts, the following formula keeps the actor in the correct position.

74+(self.Size.Width/2)

The value of 74 can be changed to match your game. That number is the starting point of the health bar, which is also the left side of the health bar. The center point is half the actor's width. If the actor was 50 pixels wide, its center location should be +25 pixels to the right of the base. The actor can be a simple red box. If you want something more creative, you might want to remember that the actor has to stretch.


On Rule

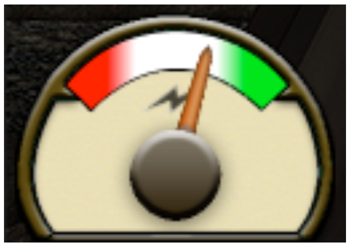
When All conditions are valid:

Actor receives event overlaps or collides with actor of type Fighter

On Change Attribute

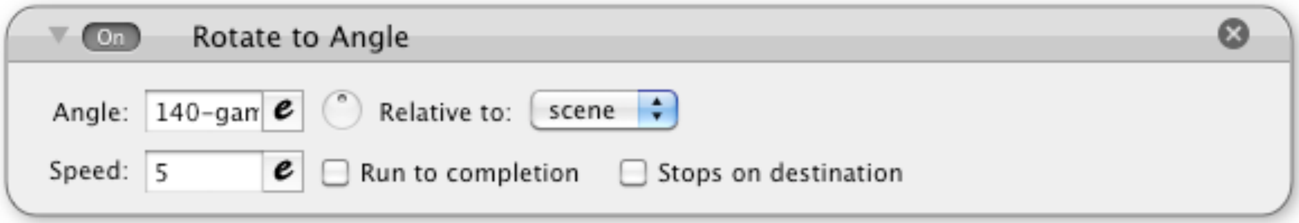
Change Attribute: game.Fighter Health ... To: game.Fighter He

Then, when harmful actors collide with your main actor, points can be deducted from the health bar. Another "Rule" could be created to detect for death. If the health bar value is less than or equal to zero, the main actor could be destroyed. To prevent your health bar from getting too large, or too small, you can use the "min" and "max" functions to restrict the range of the custom health "Attribute".



You could also change the health bar to an energy gauge. Instead of growing and shrinking bar, a needle rotates to show the power level. The image to the left shows three actors — the background, the gauge and the needle. Only the needle needs to move for this part. Like the tank turret, it's important to properly align the center point. Otherwise, the needle won't rotate properly.

With the health bar, you needed two constraints. One to controls the center point and another to control the width. But with a gauge, you only need to manage the "Rotation". The center point stays the same.


On Rotate to Angle

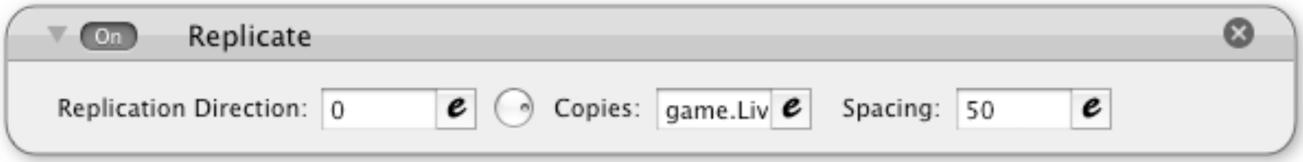
Angle: 140-gam Relative to: scene

Speed: 5 Run to completion Stops on destination

The "Rotate to Angle" behavior can be used to display the current energy level. The trick is to remember that an angle's value increases in a counter-clockwise direction, while a typical gauge usually increases in a clockwise direction. To resolve this matter, just subtract the game's energy level from the angle of the starting point.

In the example shown above, the gauge is on empty when the needle is at 140°. The needle has a 100° range. So, if the gauge is full, it rotates to a 40° angle. Half energy would put the gauge at a 90° angle.

Lives - Many classic video games didn't have things called health bars or energy gauges. If you got hit, you died. What they did have were extra guys or lives. If you want to recreate this classic method, you can use the "Replicate" behavior.

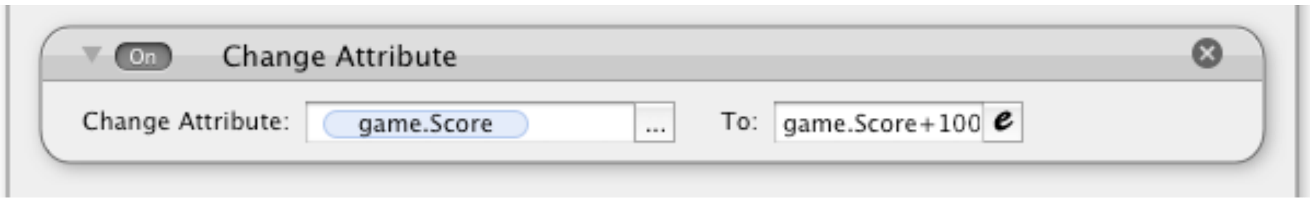


Replication Direction: 0
Copies: game.Liv
Spacing: 50

With the game.Lives value, the number of player lives left can be displayed. This was already discussed in chapter 5, but here's what's new. You could also use the "Replicate" behavior as something like a health bar. Instead of having to use constraints, you can simply make a little graphic and have that be replicated for each bar of health. That style is similar to the health bars in Mega Man or The Legend of Zelda.

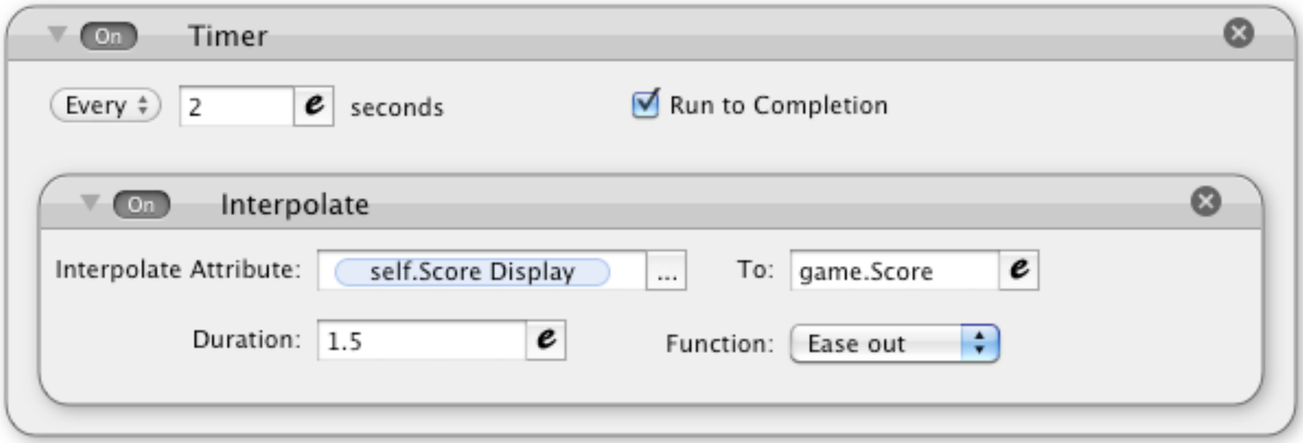
Power Ups - In this chapter, it was shown how to make actors hurt each other. With an overlap or a collision actors could destroy each other or they could deduct points from a health bar. Modify the process to help your actors. You can create 1up actors, health boosters and power ups. For example, if the main character overlaps with a bonus life, that bonus actor would be destroyed. But instead of causing harm to the main actor, the custom "Attribute" for player lives could be increased by one.

Scoring - Also similar to health bars, you can keep score. When a player shoots an enemy, collects a bonus item or completes a level, you could increase the player's score. This is typically accomplished with the "Change Attribute" behavior.

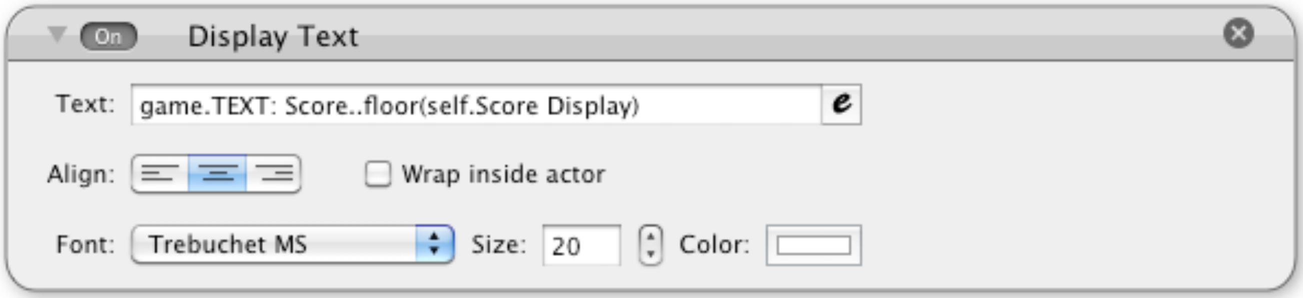


Change Attribute: game.Score
To: game.Score+100

Whenever a condition has been met, points can be added to a custom attribute. In the above example, 100 points is added to the game.Score value. The result is a little plain, so here's a fancy way to display a player's score.

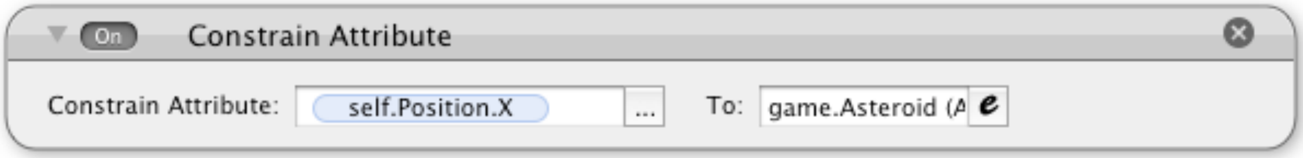

Timer
Every 2 seconds ☑ Run to Completion
Interpolate
Interpolate Attribute: self.Score Display To: game.Score
Duration: 1.5 Function: Ease out

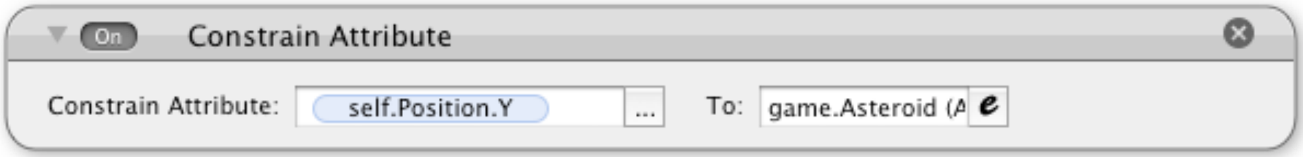
Every two seconds, the self.Score Display value will count up to the game.Score value. (A timer is used to keep the display from getting stuck.) Instead of an instant change, the score races upwards. It looks more dramatic. To show the self.Score Display value, you can use the "Display Text" behavior. Although, you might want to use the "floor" function to prevent the display of decimal numbers.


Display Text
Text: game.TEXT: Score..floor(self.Score Display)
Align: center ☐ Wrap inside actor
Font: Trebuchet MS Size: 20 Color:

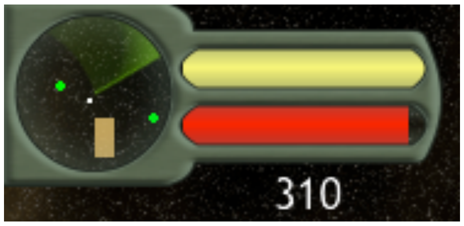
The actual game.Score data is not being increased by the "Interpolate" behavior. Use of the self.Score Display attribute, instead of modifying the game.Score attribute directly, helps prevent corruption of the data.

Radar / Mini-Map - You've got your actor on the screen. It's flying around, shooting around and collecting power-ups. What is there to do next? Go look for trouble — what else is a hero going to do? A radar system or mini-map can make the hunt easier. If your scene area is larger than your screen size, you can display off-screen action with a radar or mini-map. This is accomplished by constraining smaller actors in proportion to their larger counterparts.


Constrain Attribute
Constrain Attribute: self.Position.X To: game.Asteroid (A


Constrain Attribute
Constrain Attribute: self.Position.Y To: game.Asteroid (A

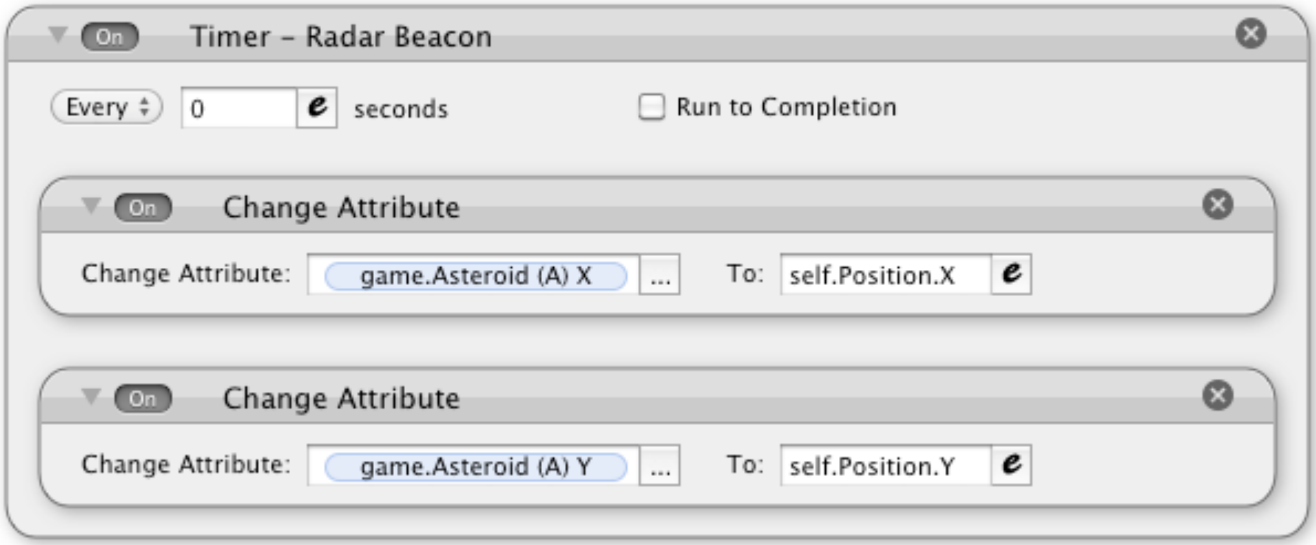
game.Asteroid (A) X/32+8
game.Asteroid (A) Y/32+412



In the example to the left, a radar beacon is constrained in proportion to the location of an asteroid. I made the radar 1/32 the size of the game screen. The full scene size is 2048 x 2048 pixels. The radar box is 64 x 64 pixels. By dividing the asteroid's location by 32, the location on the radar is revealed. The +8 and +412 values align the radar blips with the location of the radar box on the screen. In the example, the radar box is at the top-left corner. The X offset is 8 pixels and the Y offset is 412 pixels.

Adding a radar or mini-map to your game can be extremely tough on your game's performance. For each actor you want to track, another radar dot is needed. With each new dot, you're adding four constraints. One X and Y combo to record the target's location and another X and Y combo to move the radar blip. You're also adding additional actors to your game.

I figured out a way to reduce the number of constraints and actors used by a radar system, but it probably won't work for all situations. A "Timer" can be near constant if the "Every" delay is set to 0 seconds. Yet, if copies of actors are fighting to change the same attributes, you can get a flickering effect. A single radar blip can jump to multiple locations. In this example, one blip is assigned to a large asteroid. If another copy of that asteroid appears in the scene, the radar blip flickers between both positions. I used this approach because it improved performance and it fit the theme of my game. In other science fiction stories, asteroid fields usually mess with electronics. My game's radar didn't have to work perfectly. I turned a glitch into something of a feature. It's possible to expand and improve on this concept. The general idea is this — instead of having several constant radar blips, maybe have just a few flickering radar blips.


On Timer – Radar Beacon

Every 0 seconds ☐ Run to Completion

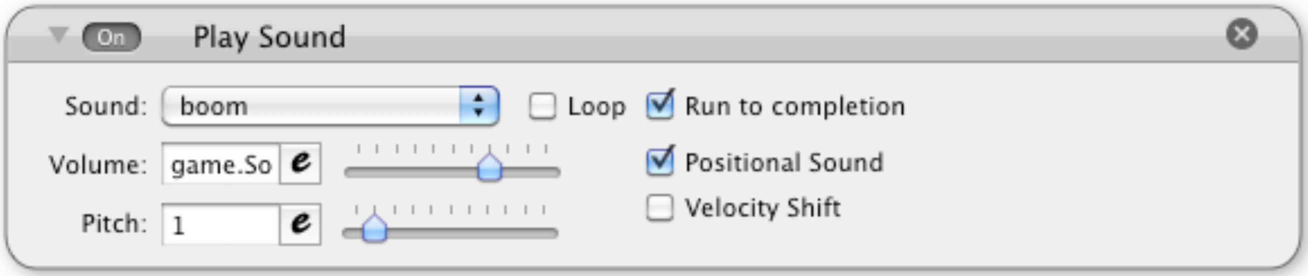
On Change Attribute

Change Attribute: game.Asteroid (A) X To: self.Position.X

On Change Attribute

Change Attribute: game.Asteroid (A) Y To: self.Position.Y

Options - Players like to have options. Some players might want the sound effects to be louder, others might want the difficulty to be easier. Can you make everyone happy? If your game has enough options, maybe you can come close. Similar to the other examples in this chapter, "Attributes" play a key part in game options. For example, you can create a volume settings that can apply to all of your sound effects.


On Play Sound

Sound: boom ☐ Loop ☑ Run to completion

Volume: game.So ☑ Positional Sound

Pitch: 1 ☐ Velocity Shift

Place a custom "Attribute" in "Volume" field of your "Play Sound" behavior. In the example above, <u>game.Volume</u> is used. Place the same "Attribute" in every one of your "Play Sound" behaviors. Then, a menu screen would be needed for changing the sound. One actor can be a button to increase the <u>game.Volume</u> value. Another actor can be a button to decrease the <u>game.Volume</u> value. The same concept can be expanded to other game settings. If you want to make a game harder, you could make the enemies do more damage. Then, after the player has changed the game options, use the "Save" and "Load" behaviors to reuse the preferences for the next game.

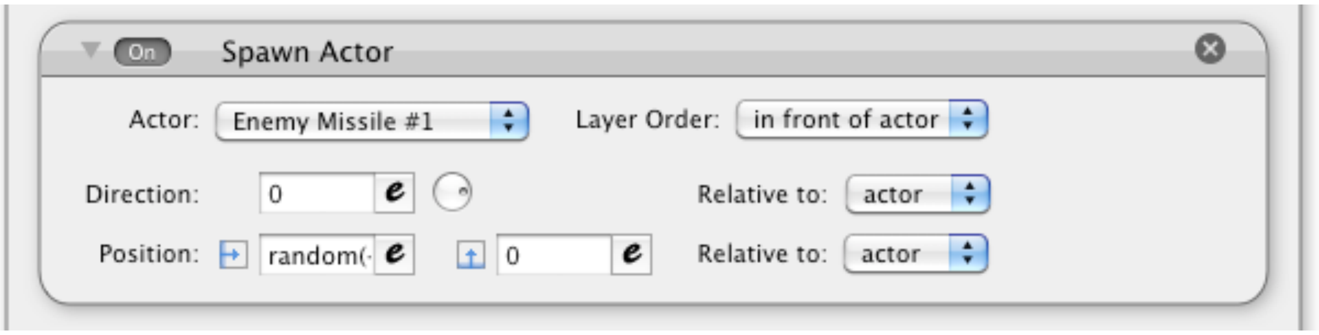
Chapter #8 Summary

- Many common game elements can be accomplished with actor-to-actor communication. Use a combination of “Attributes” and “Behaviors”, to add action to your game.
- Would an energy gauge work better than a health bar? Should your game have a mini-map? As you create your game, you might want to be mindful of optimization. By simplifying your game’s design, you can improve your game’s performance.
- Even though GameSalad can be used to make a game quickly and easily, you can still add professional touches to your game. You can add an option screen with volume control and difficulty settings.

Chapter #9 - Enemies

Not all games have to involve conflict, but this chapter is for the ones that do. The better the enemy, the more heroic the hero. Remember your role in this epic battle. Your goal is to make the player feel good. That is how you can become a hero in the real world. If your players want to be a good guys, then your enemies should be designed as such. The bad guys shouldn't be too hard. If a player can't achieve success, they will give up on your game in frustration. Yet, the challenge shouldn't be too easy either. The players will abandon your game out of boredom. The trick is to find the right balance. Your enemies should be tough enough to provide challenge, but easy enough to cater to the egos of your players.

In creating your enemies, you might want to decide how they will enter the scene. You could accomplish this in two ways. You could pre-populate a scene with enemies or you could use enemy spawners. There are benefits and drawbacks with both methods.



Spawn Actor

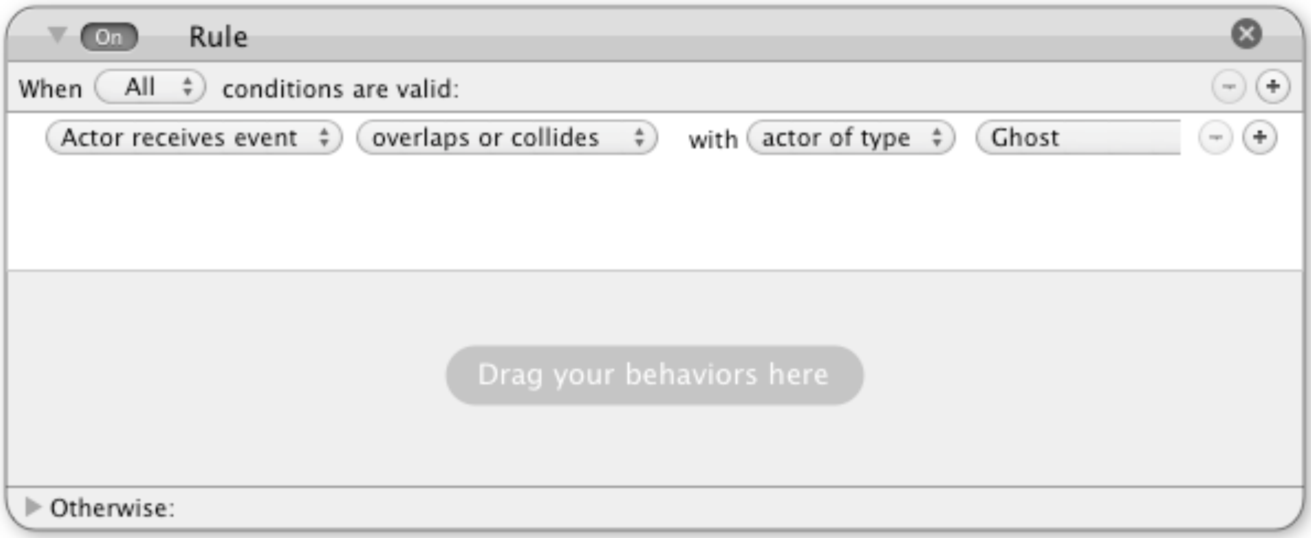
Actor: Enemy Missile #1 Layer Order: in front of actor

Direction: 0 Relative to: actor

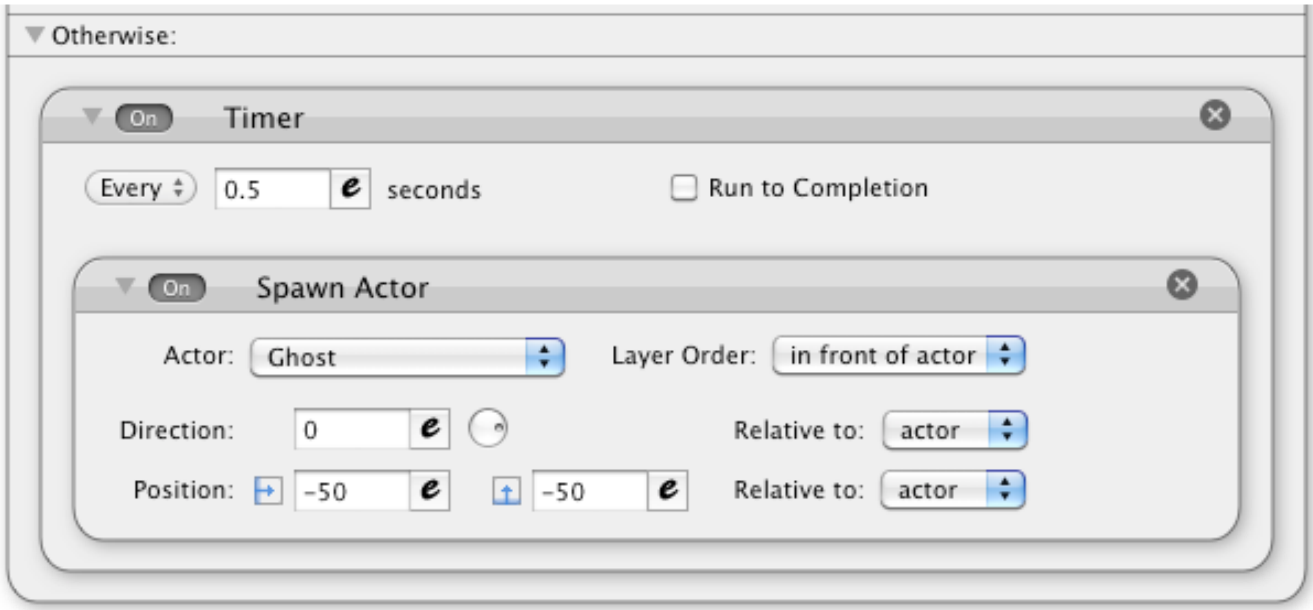
Position: random(0 Relative to: actor

Spawning - If you create enemy spawners, you can add enemies to your scene whenever you need them. You could also increase the spawn rate, to make the game harder with each new level. By using the "Timer" behavior, you could add a new enemy to the scene every few seconds. You could also make monster generators, like the ones found in Gauntlet.

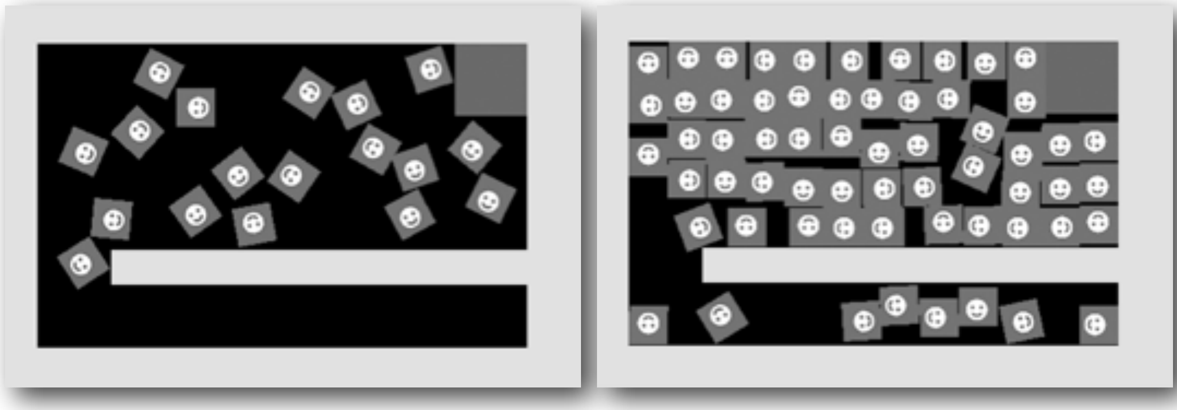
How do you stop a monster generator? If you leave a spawner unchecked, it will constantly add enemies to your scene. That's consistent with vintage arcade games. But if you want something more subtile, you could place your "Spawn Actor" sequence inside a "Rule". If the conditions are met, spawn an actor. However, there is another trick to using a "Rule". You could reserve spawning activity for when an a condition is not met.


Rule
When All conditions are valid:
Actor receives event | overlaps or collides | with | actor of type | Ghost

There it is! It's the "Rule" for monster generators. "But… err… Mike, I don't see anything. The container is empty!" Open the "Otherwise" portion of the "Rule" and look again.


Otherwise:
Timer
Every 0.5 seconds
Run to Completion
Spawn Actor
Actor: Ghost
Layer Order: in front of actor
Direction: 0
Relative to: actor
Position: -50 -50
Relative to: actor

If the generator is touching a ghost, the scene is probably too crowded, so it shouldn't spawn another ghost. However, if a ghost is NOT touching the generator, there's room for another ghost. Not is the key word. With collisions and overlaps, you might want something to occur when actors are not touching each other. With the "Otherwise" portion of the "Rule" behavior, you can make that happen. When the "overlap or collides" is NOT happening, the ghosts will be spawn at a rate of two per second. (I sped things up for the next two images.)



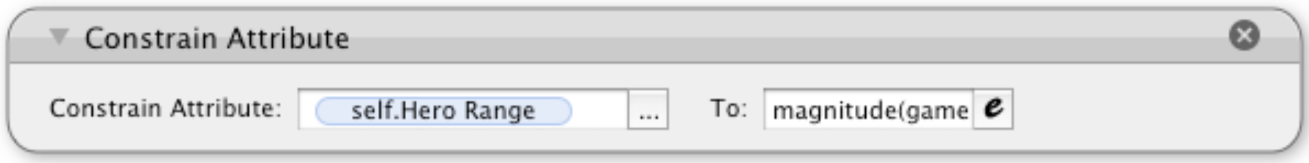
In the above example, the square ghosts are just bouncing around like balls. The generator chucks the ghosts into the scene, as shown in the above-left image. But then, when things get too cramped, the generator stops. In the above-right image, the generator has stopped spawning ghosts.

Trying to find the source of evil, in a virtual world, could be an exciting adventure. By using spawners, your game can have a more sandbox feel. You can spawn enemies randomly or according to certain conditions. That dynamic action can increase the replay value of your game, as the enemy encounters are somewhat unique with each new game.

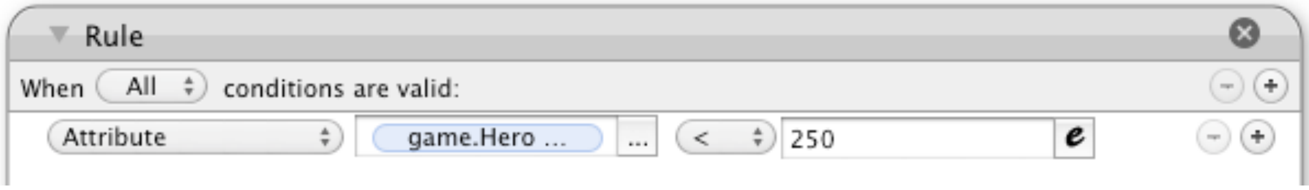
In addition to those cool effects, spawning makes it's easier to manage your game. In the right image, there are 61 enemies. (I counted each one.) That's just from a single generator. Adding that many enemies manually would be incredibly tedious. If your game has a large amount of enemies, you'll probably be using spawners. However, the drawback is quite significant. Constantly spawning and destroying actors is a processor intensive task. If you try to spawn too many actors at once, your game's performance can suffer.

To avoid that performance issue, you might want pre-populate your scenes with enemies. That's so boring. So, here's a trick to spice things up. It's called "Aggro". It's a common term in online role-playing games. Usually, off in the distance, a monster will be wandering about. It's just walking around, doing it's own thing. Then suddenly, you realize that you're too close. The monster sees you. It's mood is now aggressive. It charges at you. It wants to kill you!

You can make GameSalad actors behave in a similar way. By using "Attributes" to share location, and using the "Magnitude" function to calculate distance, you can create an aggro bubble around your enemies. If the main actor steps inside the bubble of aggro, the monster can then change its action.


Constrain Attribute

Constrain Attribute: self.Hero Range To: magnitude(game


Rule

When All conditions are valid:

Attribute game.Hero ... < 250

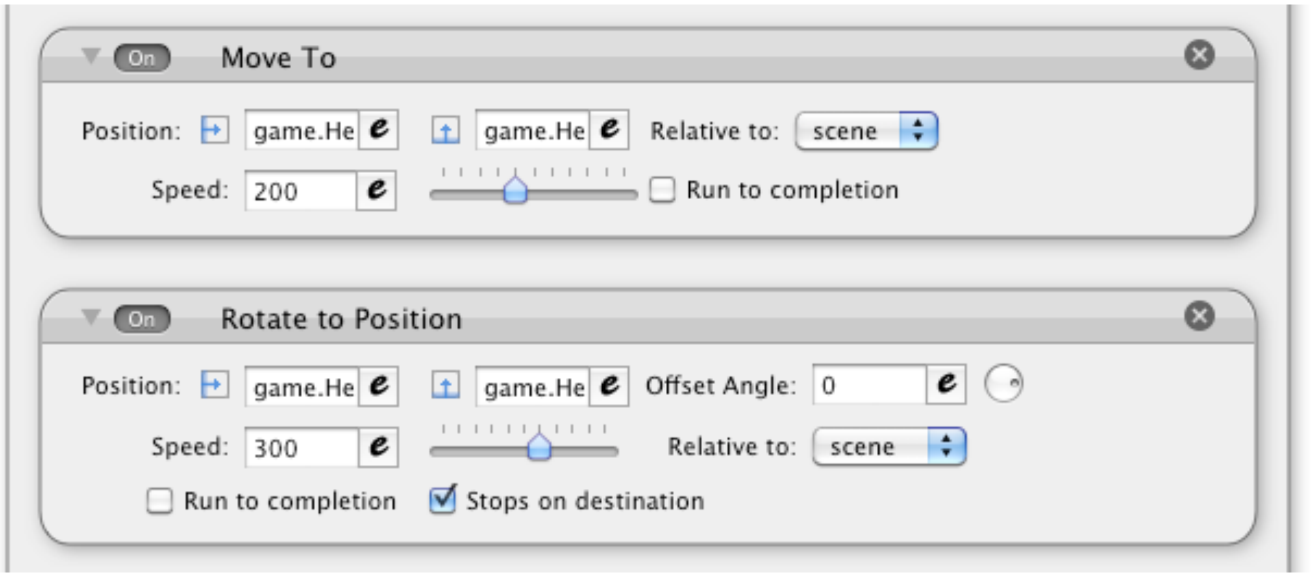
In the above example, the enemy has a custom "Attribute" of self.Hero Range. That's used to track the space between the hero and the enemy. This is accomplished with the "magnitude" function and a "Constrain Attribute" behavior. If that value falls below a certain point, the "Rule" goes into action. The "Expression" used to calculate the distance between the two actors is probably familiar to you by now.

magnitude(game.Hero-X-self.Position.X, game.Hero-Y-self.Position.Y)

That "Expression" essentially creates an invisible circle around the enemy. Whenever the hero crosses that circle, the condition for the "Rule" has been met.

Artificial Intelligence — We're now entering a spooky area of game design, making it appear as if the enemies can think for themselves. For decades, I've slain countless video game foes. Yet, when I created my first "thinking" GameSalad actor, it was almost as if I fell through reality. The process was so easy, so simple, but it made me question my very existence. In the above example, what's really going on? The enemy is reacting to a condition. If the hero is too close, do something. Is that not how humans react? If a bee is too close, shoo it away. I started thinking philosophically. What makes me different from the actor? I started thinking about free will and self-awareness. It was pretty amazing stuff. That's why I enjoy GameSalad development. It can open your mind.

So, back to our enemy. It's just part of a program in a machine. It doesn't think. It doesn't choose. It just follows instructions. Your goal is to make it appear that enemy is thinking. You can do that by giving it good instructions. The first part has been accomplished. The enemy has detected the hero. Now, the enemy should do something. In most video games, a hostile enemy typically run towards the main character. That's pretty easy to do with GameSalad. Since the enemy already has access to the hero's location, they just need to be plugged into the "Move To" and a "Rotate to Position" behaviors. When the enemy turns aggressive, it will turn to face the hero and the enemy will move toward the hero. It was actually a little creepy to me when I first tested it out. That's how I knew I was creating a fine enemy.


Move To
Position: game.He game.He Relative to: scene
Speed: 200 Run to completion

Rotate to Position
Position: game.He game.He Offset Angle: 0
Speed: 300 Relative to: scene
Run to completion [✓ Stops on destination]

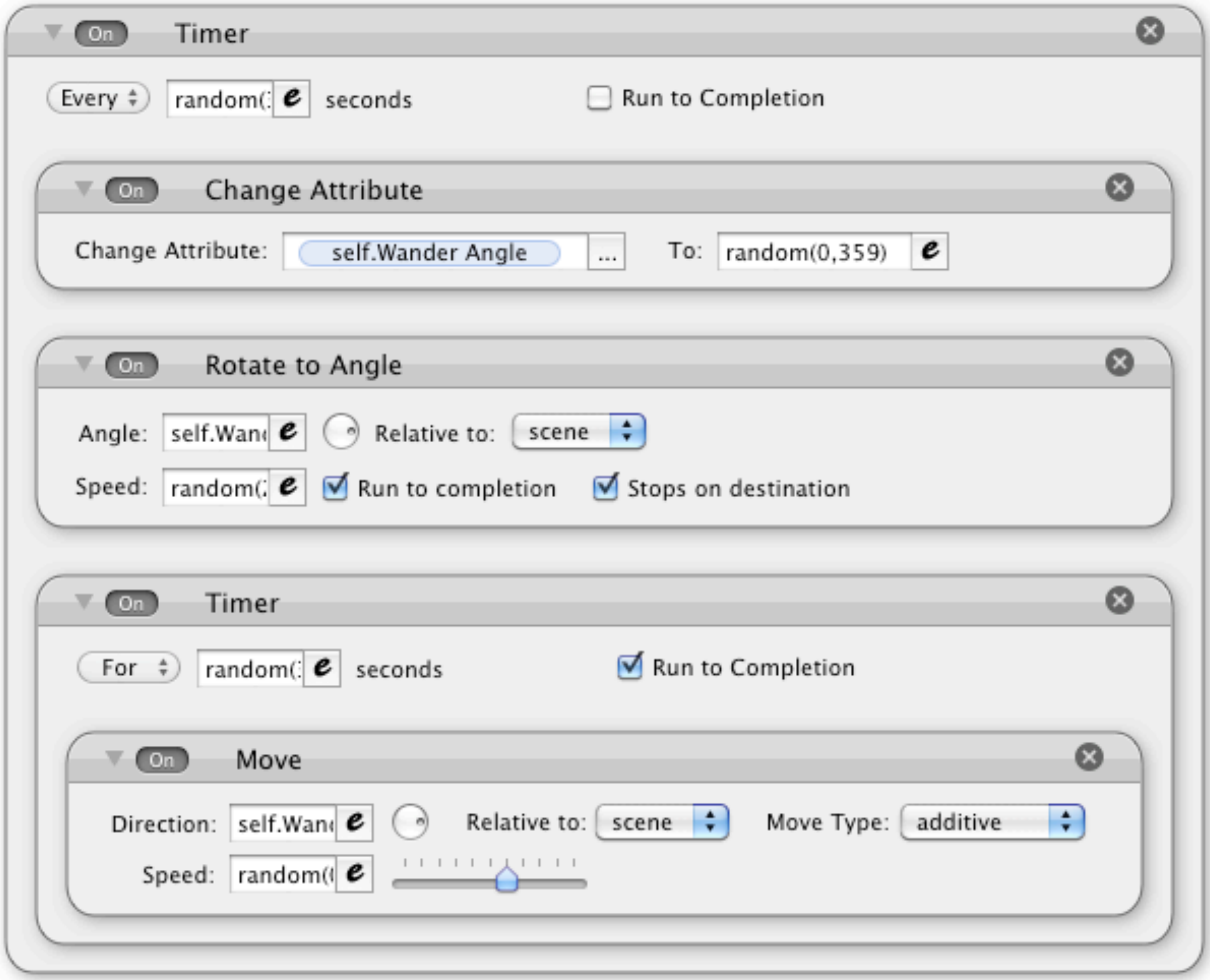
The above image shows the two "Behaviors" for this example. They would go inside the "Rule". Of course, this setup does create some problems. First, you wouldn't want your monsters to running into walls. It won't know to run around an obstacle, so that's something to consider when designing your levels.

Another issue is stopping the aggressive monster. Do you want your enemies to chase your actors forever or should the hero be able to escape? With some additional work, you can create a way to solve that issue. In the above example, the enemy is aggressive when the hero is less than 250 pixels away. If the hero runs 500 pixels away from the aggressive enemy, the aggression is lost and the enemy returns to its starting position. Creating that buffer zone creates more believable reactions. If I'm spotted by a charging rhino in Africa, it shouldn't chase me all the way to Times Square in New York City.

You might want to add an idle sequence to your enemies. You can use the "Otherwise" section of your aggro "Rule" to create non-aggressive tasks for your enemies, such as pacing back and forth or sleeping. You can easily make an enemy pace around an area with the "Move To" behavior, a "Timer" and the "random" function. Every few seconds, the actor could move to a random location from its current location. The trick is to record the start position. If the enemy wanders too far in one direction, then send it back towards the starting point. Once again, this is concept builds on the "magnitude" function.

If you go crazy with “magnitude”, you might find yourself with a lot of constrains in your game. That’s bad for performance. So, there’s another way to create reaction points. Instead, you could make traps. If the hero crosses an invisible actor, it could triggers a sequence of awful events. Perhaps you want arrows to shoot out of walls or something harmful like that. That is one possibility. Another possibility is simply an alarm. If the alarm is triggered, nearby enemies could be alerted to the presence of the hero. It’s all built on the same basic combination of GameSalad tools — “Attributes”, “Behaviors” and “Functions”.

The following is a wandering sequence for an overhead game. It uses the “Random” function to pick a direction. The range is 0-359. Then, the character will turn at a random speed towards that angle and travel at a random speed in that direction. That’s a lot of randomness, but that’s what creates a more believable idle sequence. The following image shows what the setup looks like.


Timer
Every random(: seconds ☐ Run to Completion

Change Attribute
Change Attribute: self.Wander Angle To: random(0,359)

Rotate to Angle
Angle: self.Wan Relative to: scene
Speed: random(: ☑ Run to completion ☑ Stops on destination

Timer
For random(: seconds ☑ Run to Completion

Move
Direction: self.Wan Relative to: scene Move Type: additive
Speed: random(

The self.Wander Angle attribute records the random angle. The “Rotate to Angle” and “Move” behaviors use that angle for direction. There are five different “random” functions. Together, they set the pace and direction for the idle movements. If players can detect the pattern, the enemy becomes less of a character and more of a program. The above example is not the definitive guide for all idle actors in GameSalad games. Each character should behave differently. It’s fun to just drop in numbers randomly, but that’s not what this is about. It’s almost like painting a picture. By picking certain colors the mood and the meaning of the picture can change. It’s the same with creating video games, but your palette is vast.

That’s where it can be difficult to be an independent game developer. If you’re a great illustrator, but your logic and programming skills are weak, it can hurt your game. Every little detail can matter in building a better game. That’s especially true with creating enemies. Every aspect of your enemy can reenforce or break that character’s theme. If you’re building a monster, does it look like a monster, does it move like a monster, does it sound like a monster and does it react like a monster?

From this point, we’re moving beyond the technical. Now, it’s more about mastery of the trade. It’s more about games as art. With GameSalad, almost anybody can learn how to make an enemy move around the screen. What’s going to separate you from the crowd is knowing when and why to do it. Enemies can be more than just boxes you shoot at. They can be characters in a story.

Chapter #9 Summary

- Enemies can be pre-populated in a scene or they can be spawned into an active scene. Either method has advantages and disadvantages.
- The “magnitude” function is quite handy for building artificial intelligence. It can be used to create reactions when two actor are within, or beyond, a certain range.
- The “Move To” and “Rotate to Position” behaviors can be used to make an enemy chasing the main character.
- You might be noticing some repetition in the last few chapters. That’s because the same basic principles apply. Once you understand how GameSalad “Attributes”, “Behaviors” and “Functions” work, the rest is more about your ability to be creative.

Chapter #10 - Setting the Scene

If you've made it through all of the chapters so far, you should have a strong understand of how GameSalad works. You can start building your gaming environment. Yet, that black box can be an intimidating foe. It seemingly mocks you with its emptiness. Perhaps a clearer understanding of the settings can help. This chapter takes a closer look at "Scenes".

First, you might want to determine the size of your game. You might not know what you want to put in the black box. But if you know the general size of your game, that's step in the right direction. It can be changed later, but that could cause extra work. Even if you selected iPhone Portrait or iPad Landscape for your game, that's still not the same as determining a size for your "Scene".

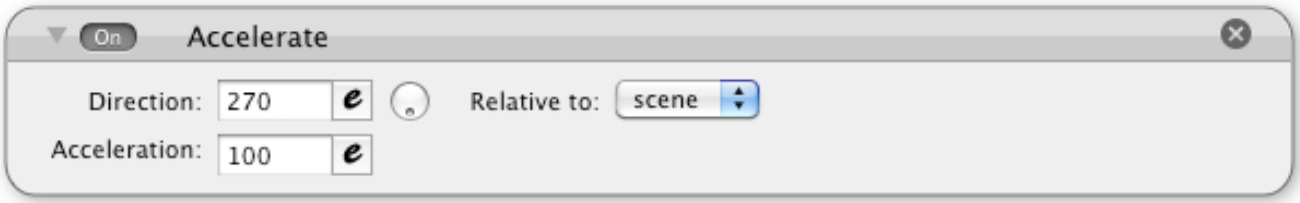
Before you add any actors to your game, you can make some pretty big decisions about its design. Do you want to build a game that scrolls? If so, you're going to need a bigger box. The size of the screen doesn't have to be equal to the size of your playing area. Be careful, as this is a danger area. If your goals are too lofty, and your game is simply too big, the performance can suffer.

"How big can I make a scene?" That's a common question. The answer — larger than you probably should! (For comparison, my first GameSalad game was 2048 x 2048 pixels. That's not really a big game, but my next four projects were all smaller.) Just to be sure, I played around with the "Size" settings. I expanded the "Width" of a "Scene" to 10,000,000 pixels. Surprisingly, the test worked on my iPod Touch. Yet, I knew that this wasn't a good game size. For one, the "Width" value was abbreviated as 1e+07. If the number has to be truncated, that's a pretty clear warning sign. Also, if I had to fill that area with actors, my game would probably crash from lack of memory. But even if I kept the game streamlined, such a large space could be pretty boring. With an actor moving at 100 pixels per second, it could take over a day to travel from one side to the other. Not only is performance a valid concern, so is entertainment. Will your game be fun? Size matters, so think, build and test carefully. It can be devastating to work on a game for hours, days and weeks, only to realize that it won't run on many iOS devices.

Another thing you'll have to decide for your "Scene" is wrapping. If an actor reaches the edge of the playing field, should it keep going or should it appear on the other side? If you want the latter, set "Wrap X" and/or "Wrap Y" to true. If you don't enable wrapping, an actor can travel beyond the "Size" of the "Scene". But if an actor travels 500 pixels beyond the "Width" or "Height" limit, the actor will be destroyed.

The silly looking image (split in two) shows what happens when "Wrap X" is enabled... disturbing! The smiley face has been cut in half. That's why you might want to combine wrapping with a larger playing area. By creating some leeway, actors don't immediately appear on the other side. It's similar to what happens in Pac-Man, while going through the side door.

Gravity - Right below the "Wrap" settings, are the settings for "Gravity". This is a tough decision. Should you include gravity in your game? You could avoid it. Instead, you could simulate "Gravity" with the "Accelerate" behavior.



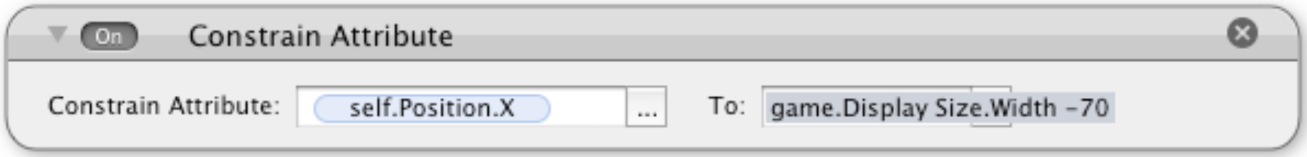
Direction: 270 Relative to: scene
Acceleration: 100

If you only have a few actors that need "Gravity", it might be easier and more efficient to use the "Accelerate" behavior. However, if you have a lot of actors in your game, it's quicker to use the "Scene" setting. Actors that are not set to "Movable" do not use the "Gravity" setting. For things like walls and floors, that can be OK. Unless you're building destructible environments, you probably wouldn't want those types of actors to move around.

Color - You don't have to fall for a beginner mistake. You can change the background "Color" of a "Scene". If you want a white background for your game, you don't have to make a big white actor. That's wasting precious processing power. To change the "Color", go to the "Inspector" pane, click the "Scene" button and then look for the "Color" attribute. You can also the <u>scene.Color</u> attribute to change the "Color" in-game. Creating a background can be tricky. Your first reaction might be to throw a big graphic in the background. I did that and hindered my game's performance. It wasn't until later that I learn what to look for. I removed the background images from my game and the frames per second jumped by 15-20%. For more information on this matter, see Chapter #18 - Optimization.

Autorotate - This is related to iOS publishing. You can turn an iOS device to different orientations. If you're developing for the iPhone, iPad or iPod Touch, the easiest way to handle this issue is to click both check boxes for your target orientation. If you're building a "Portrait" game, select "Portrait" and "Portrait Upside Down". If you're building a "Landscape" game, select "Landscape Left" and "Landscape Right". (Left and right refers to the position of the "Home" button.) That makes it nice and easy to meet Apple's autorotation requirements.

Depending on your game's design, four-way orientation can work with a scrolling game. The problem is the HUD – that's usually doesn't scroll. To fix this problem, you can grab the "Width" and "Height" values of the screen. With that information, you can postilion your interface elements in relation to those values. In the example below, the joystick is being constrained to 70 pixels less than the game.Display Size.Width value.


Constrain Attribute

Constrain Attribute: self.Position.X To: game.Display Size.Width -70

Cool Tip

How many scenes do you need? - During your development you might start to wonder about the size of your game. Is it big enough? That could be a dangerous train of thought. Sure, a game should be big enough to satisfy the players, but that's not the only issue. Every scene change is a place where the game has to load.

Gone are the days of the Commodore 64 floppy disks. Back then, it could take minutes for a game to load. Today, players can get grumpy if they have to wait mere seconds. Sometimes the loading sequences can be short. But even if it's 2-3 seconds, that can add frustration to the game... especially if they have to do it over... and over... and over again.

While building "BOT", I boasted that the game had 146,800,640 pixels of exploration on Retina Displays. For six months, I tried to craft a huge game. But once the game launched, it was disappointing to see the bad reviews. By looking at the Game Center scores, I could tell that the majority of the players didn't get very far. Roughly 7% managed to get to the final stage of the game. What good is content if the players don't see it?

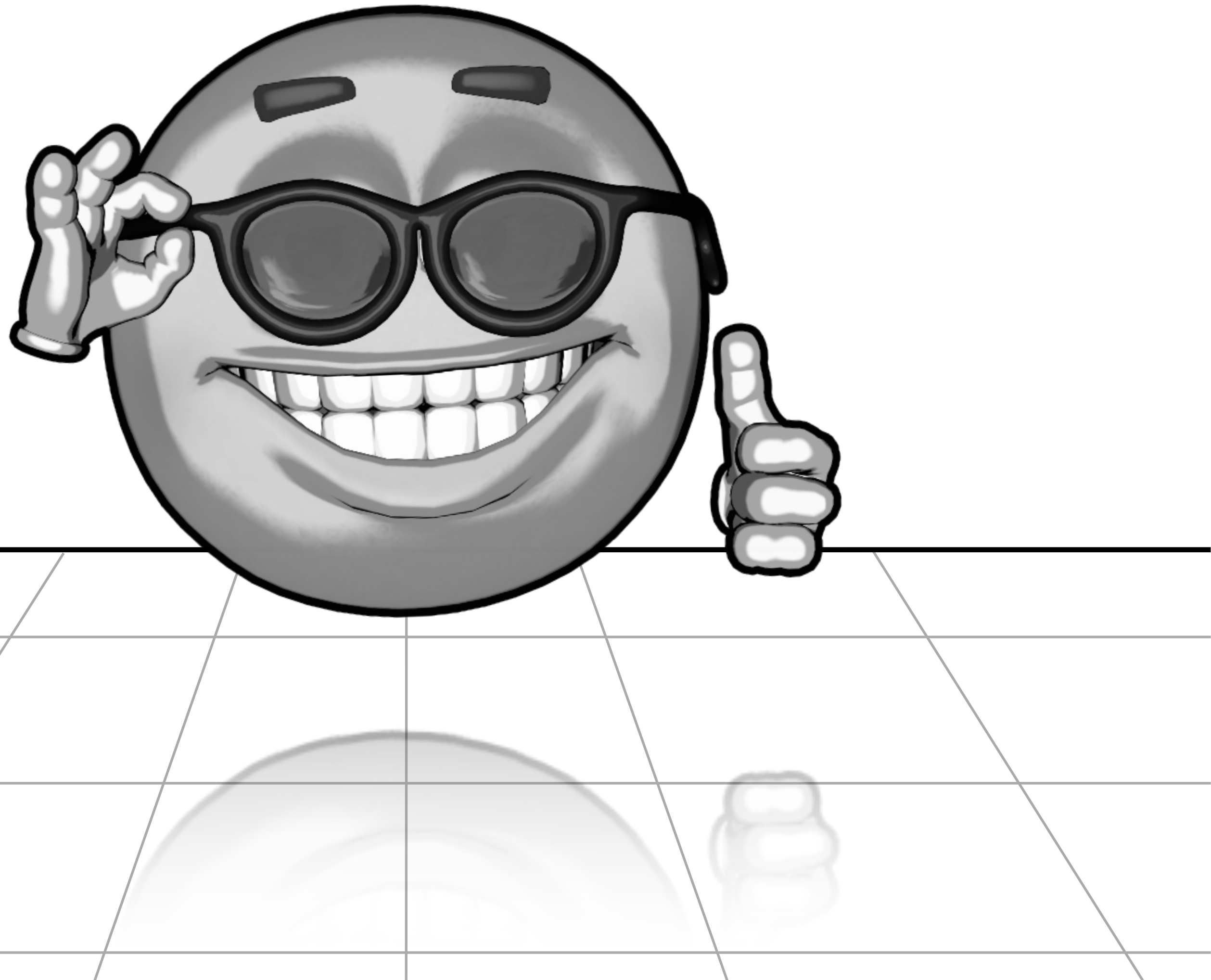
On the flipside, I created a puzzle game that was featured by Apple. In that app, all of the action was crammed into one scene — and this was before GameSalad added tables.

Before you get lofty goals stuck in your head, you might want to really look at your scene design. Not just look... REALLY LOOK at your scenes! Is your game fun? Are players really going to enjoy themselves? By cutting down the loading time, that's a good step. Even greater than that, which would you rather experience — 10 fun levels or 100 mediocre levels? The former can be memorable. The latter could be torturous.

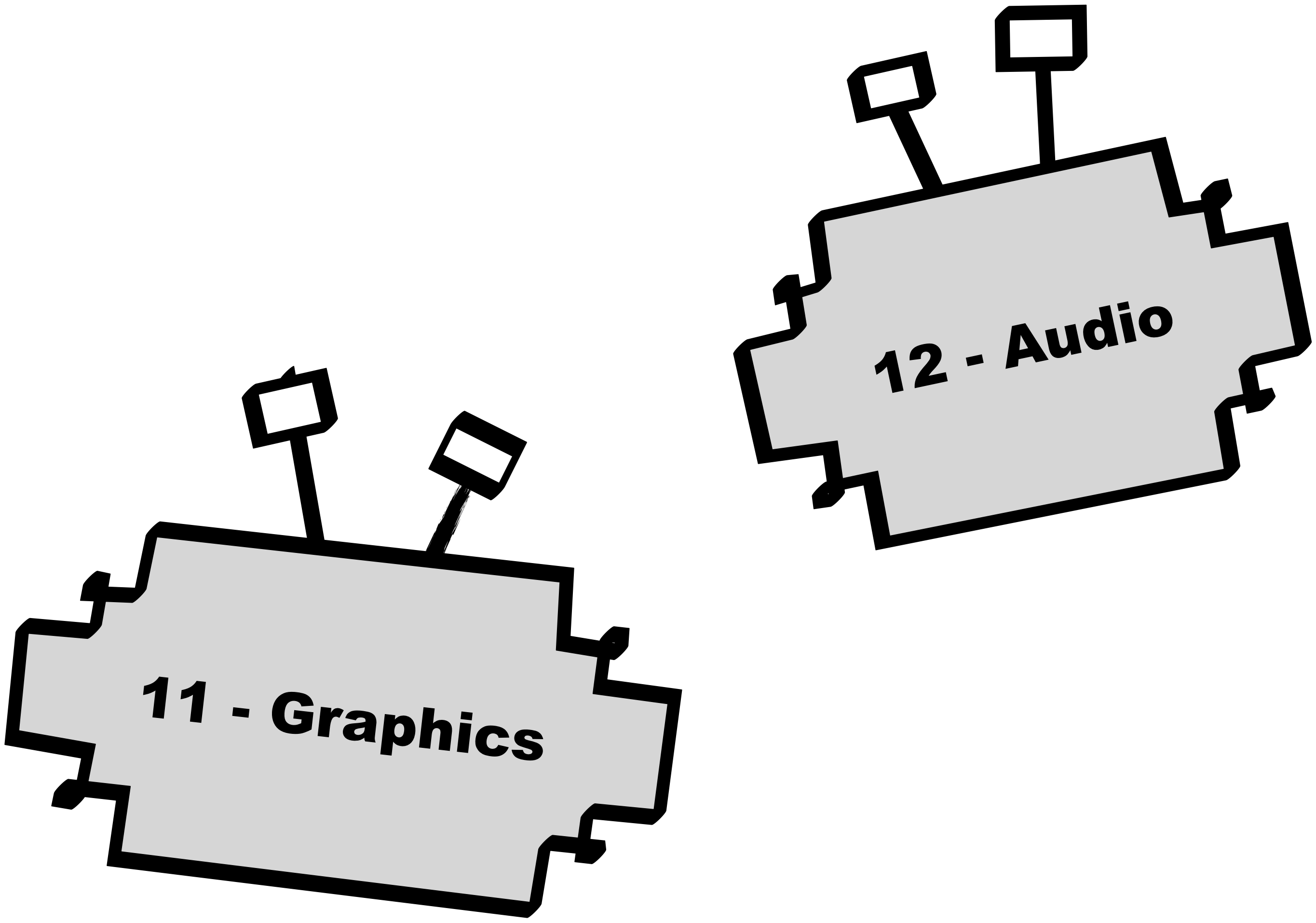
Chapter #10 Summary

- A "Scene" can be huge, but that could cause problems with performance and playability.
- Setting a scene to "Wrap" will cause actors leaving one side of the "Scene" to enter on the opposite side.
- The "Accelerate" behavior can be used to simulate gravity.
- The background color of a "Scene" can be changed.

4



Section IV - Making Content


11 - Graphics
12 - Audio

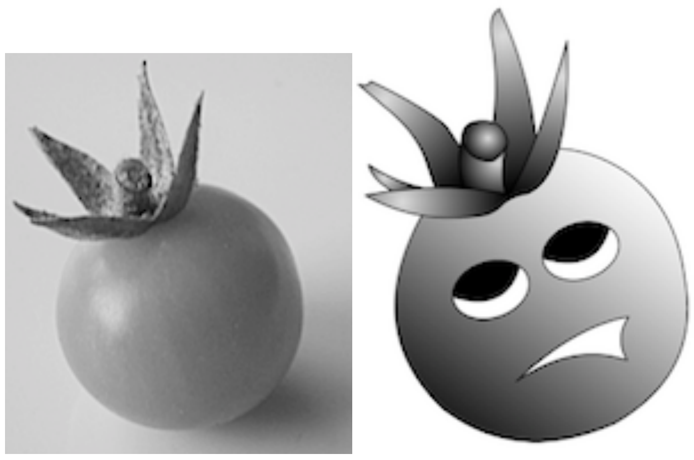
Chapter #11 - Graphics

With GameSalad, the need for traditional programming is thrown out the window. However, the need to be creative, that's still important. Creating quality artwork is absolutely critical for your GameSalad games. When a consumer is considering the purchase of your game, they'll probably be looking at your game's graphics most of all. That means your artwork needs to look great… but what if you can't draw? That's what this chapter is about.

It all starts with a tomato. That's right… a tomato seed to be precise. I planted some cherry tomatoes seed in tiny little pots, with a little bit of soil. The seeds were so little. It seemed almost impossible that something so small would grow at all. Yet, I did what I was supposed to. I gave it water and I gave it sunlight. Then, one day, tiny little leaves began to emerge from the dirt.

Even with signs of hope, there were many days that I thought my tomatoes would die. Some days it was too cold, other days it was too hot. On some days I'd forget to water my plants.On other days, it would rain a lot. Yet, by keeping up with the plants, they kept growing. By the middle of summer, they were about five feet tall. That's how your GameSalad game development can grow. There are things that seem impossible. There are days where the randomness of life seems against your favor. But if you keep at it, keeping doing what you're supposed to be doing, you can probably find success. I'm far from a farmer, but I can grow tomatoes. If you're far from an artist, you can still make artwork for your game.

What does that have to do with making games? It's the start of the creative process. Inspiration can come from almost anywhere. It can strike at any moment. One seemingly unrelated hobby crept into another. I was looking at this tomato and I thought it would be a cool character for a video game. So, I dropped it into Adobe® Flash®. I think it's great software for tracing images.



That's right — trace — if you can do that, you can create artwork. The result is shown above. That's one of the early versions of a character from Annoyed Tomatoes. I'm not sure why Angry Birds became so popular. That's why I was thinking about creating a parody. The final version of my game became something else entirely, but it all started with a cherry tomato. I don't even like to eat raw tomatoes.

Some of the artwork in my games originates just from drawing lines and curves in Flash. I then take that raw graphic and mess around with it in Photoshop. That's because the final format for a GameSalad graphic is png. Both Flash and Photoshop can export to png.

GameSalad automatically converts your images to png. There are a lot of formats you can drop into GameSalad, but they'll only end up as png. For greater control, make that your target format. If you start with a jpg or gif, which use lossy compression, you're wasting quality. Stick with png. I went a little crazy trying to figure that out. The reason is actually quite logical. GameSalad — and png images — both use the RGBA color space. That allows for transparency and some interesting effects. If you want to create graphics for GameSalad, you're probably going to need a graphics editor that can export to png. Don't worry! Even if you're poor, there are plenty of programs that can create png files. Here's a list of some free ones…

- Inkscape - http://inkscape.org
- GIMP (The GNU Image Manipulation Program) - http://gimp.org
- DAZ Studio™ - http://daz3d.com

I'm a big fan of Pixelmator. It's not free, but it is far cheaper than Photoshop.

Explaining how those programs work could fill another book. So, I'm just going to quickly highlight DAZ Studio. It's a 3D graphics and animation program and it's not actually free. It's tell-ware and they want you to subscribe to their newsletter. You're supposed to tell two people about the software. It's also a limited version of Daz Studio Advanced. Even with those limitations, the software is still awesome. I realize that I might not have impressed you with my sad tomato, that's why I included a rendering Aiko shown above. The Aiko 3D model is freely available from the DAZ 3D website. There are some restrictions about the use. For example, you can't embed the 3D file in a program and distribute that program. But with

GameSalad, that's not a problem. It only uses 2D images. They're a perfect match. At the DAZ 3D website, you can get free and inexpensive artwork for your game. If you look carefully, you'll see that this book is actually loaded with artwork from that website.

Maybe you don't understand how revolutionary this combination can be. With GameSalad, you don't need to program. With Daz Studio, you don't need to draw. (Victoria, shown above, is here to make that... point.) You don't need to be an artist or programmer to make games. What else is there?! With the knowledge in this book to put it all together, there are few roadblocks left in making your own games. It's truly an exciting time to be a game developer!

Anyway, back to the basics. There's more to know about graphic files in GameSalad. Once you have your graphic placed on an actor, there are a lot of things you can do to it. For one, you can change the color of the graphic. That's where the RGBA color space is pretty handy. By changing the "Color" attributes of an actor, you can change the tint of an image or you can make it transparent. However, you can manage color and transparency settings from a graphics program. Why would you want GameSalad to mess with your image?

Ever play a fighting game where two (or more) of the fighters look pretty much the same, but only their colors are different? That's a game optimization technique. It saves memory. By reusing the same artwork, and simply changing the color, a different colored character can be created. If the effect is obvious, it looks cheesy. People will notice the similarity and think your lazy. Today's gamers aren't too sympathetic about saving memory. If you're going to use this technique, you'll probably want to disguise it.

Instead of creating clones, color can also be used to show damage. Changing an actor's color is a quick and easy effect. You could use the "Interpolate" or "Change Attribute" behaviors to create some in-game effects. You could also set the "Alpha" channel to 0 and make the actor disappear.

Although, you might want to be careful with invisibility. If you're creating a fade effect, or something like that, using the "Alpha" channel makes sense. But if you are using an actor like a sensor, or maybe you're building some other type of hidden actor, it's probably better to deselect the “Visible” option in the “Graphics” settings. By disabling visibility, you can save processing power. There are two major drawbacks of this option. One, you can’t change this setting in-game. (This is similar to the “Movable” option, in the “Physics” settings — by optimizing and actor in this manner, you limit its options.) Two, if an actor is not visible, the “Display Text” behavior will not work properly.

Blending Mode - This is another feature of the RGBA color space. If you’ve used Photoshop layer before, this is probably familiar to you. When actors are on top of each other, you can set their colors to blend together. For example, if you had a glass of cranberry juice, objects behind the glass would appear reddish. Your actors can behave the same way. You could create an underwater scene by overlaying your scene with a bluish and transparent actor.

Normal Mode - This is the default setting. It will allow actors to blend their colors together, straight through to the background. However, transparency is needed. If the actor has no transparency, either through an image or “Color” settings, any pixels underneath will be covered.

Opaque - This mode removes transparency from an actor and its associated image. Instead, the actor fades to black as the “Alpha” value decreases.

Additive - This mode is similar to “Normal Mode” except that it brightens colors as actors overlap. For example, if one actor was pure red, another actor was pure green and another actor was pure blue, the resulting color would be white. Their values are added together. If one actor was 50% red and other actor was 50% red, the result would be 100% red.

Screen - This creates a tint that usually lightens the colors beneath the actor. It can be useful for brightening up graphics.

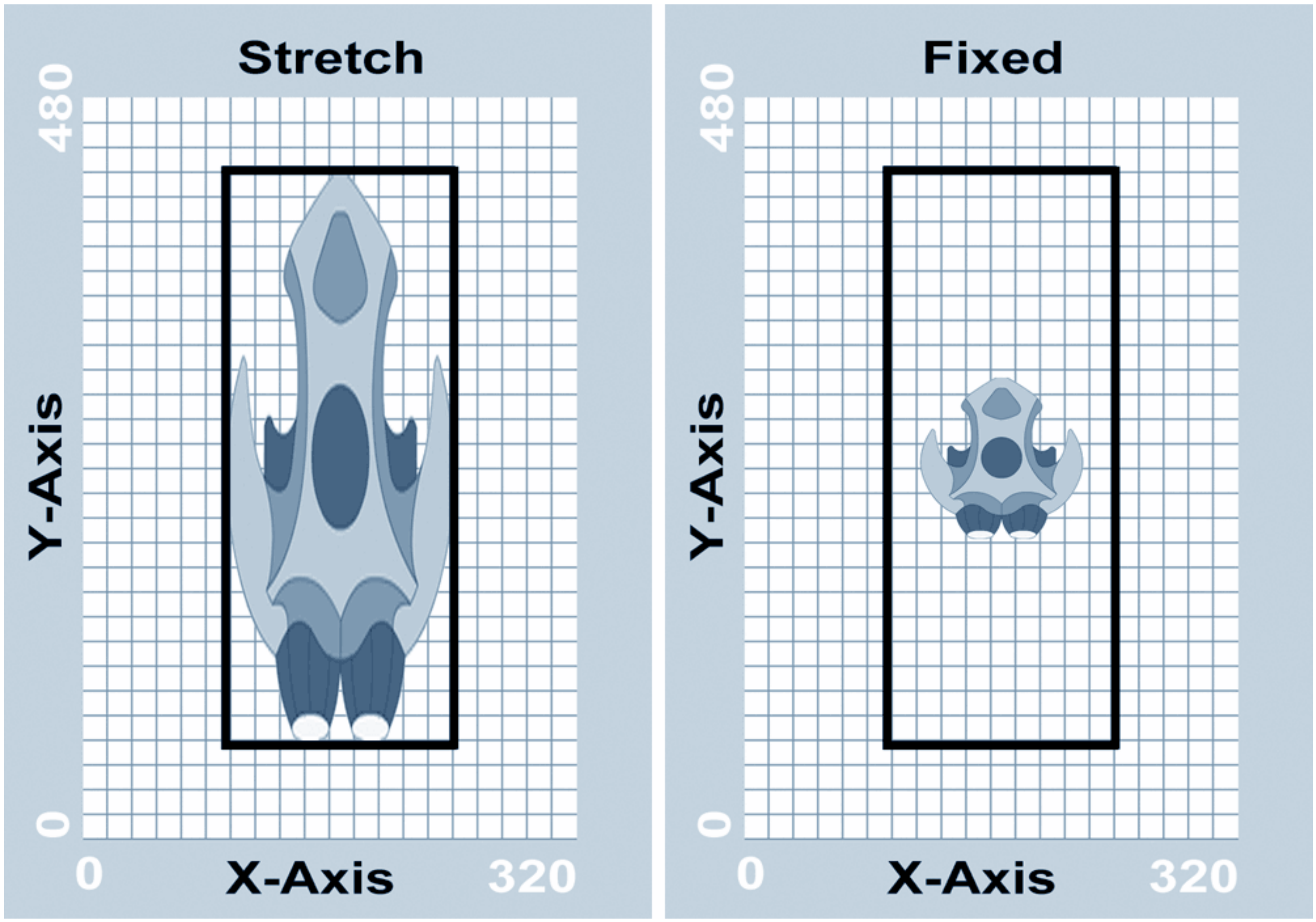
Multiply - This creates a tint that usually darkens the colors beneath the actor. It can be useful for darkening images.

I usually use “Normal mode”. When I want to create special effects — like fire — I usually use the “Additive” setting. I normally do not use the other settings. That shouldn’t discourage you though. If you’re playing with an actor that overlaps another actor, these additional settings could be something to play with. Even if you don’t get the effect you’re looking for, maybe it inspire you to create a similar effects elsewhere.

But in general, don’t use settings whimsically. When beginners first get Photoshop, the result is usually an explosions of drop shadows and lens flares. The same is with GameSalad. If you don't have to use a "Blending Mode", then why worry about it? While creating your game, it's important to be objective. As you add new content and new effects, there's an important question to ask yourself. "Does this make my game better?"

If a game element doesn't really improve your project, then you'll probably want to get rid of the offending element. People are generally slow to praise, but quick to complain. By creating your games with purpose, and eliminating unnecessary elements, your project becomes less of a target for complaints.

Wrap - When you drop an image onto an actor, the image is displayed within the actor's bounding box. By default, the image will "Stretch" to fill the size of the actor. This is probably the best idea if you're working with collision detection. However, you can modify the "Wrap" settings to achieve different effects.



Another wrap method is "Fixed", which is actually no effect at all. As shown above, the image will simply sit in the middle of the actor. This is often problematic, as the collision area can be greater than the image. That combination could create unrealistic collision effects. If the bounding box is smaller than the image, the graphic is clipped. In most scenarios, “Stretch” is a better setting. If you want the graphic to be actual size, simply match the size of the actor with the size of the image.

Tile - This effect is similar bathroom tiles. An image is repeated until the entire area is covered. It’s a great effect for creating bricks, grassy fields, hardwood floors, starry skies and other large areas that can have graphic patterns applied to them. Basically, it’s an optimization technique. From what I’ve seen of GameSalad game development, it’s one that’s largely ignored. One of the quickest ways to kill the performance of your game is to give it a giant background. But with the “Tile” setting, you can use smaller graphics files. That can save memory and improve performance.

Below is an example of a horizontally tiled image. The “Vertical Wrap” setting could be “Fixed” or “Stretched”, as the dirt image is only being repeated horizontally.

There are downsides to tiles. It’s hard to make a good “Tile” file. Even if you have a great “Tile” graphic, it’s usually not as impressive as a full graphics file. But if it’s good enough for Super Mario Bros., it could be good enough for your game.

One more important note about tiles, they start from the bottom-left corner of an actor, so that might be something to consider when creating your actors and tiles.

Tile Width / Height - If you select “Tile” for your “Wrap” mode, then there’s a new graphics option. The size of the tile can be specified. Building on the bathroom floor analogy, do you want little tiles or large tiles? With this setting, you can control how often the tile is used inside an actor. A 64x64 pixel image inside of a 320x320 actor, could be scaled down to a 32x32 pixel tile.That would be 10 tiles across and 10 tiles down, for a total of 100 tiles inside the actor.

The size of the tile can be changed while the game is in play. This technique can be used for scaling tile images. Although, .tile Width/Height attributes use integers, so the tile zoom effect is not so smooth. That means the value of the tile width and height is rounded. 14.4 pixels would be treated as 14 pixels.

Flip - What if you want your side-scrolling character to run to the left? Without image flipping, you’d need two sets of images — one of the character facing toward the right and another facing left. That's why this option is huge space saver. For fighting games and platformers an image can simply be reversed horizontally or vertically. This option can be changed while the game is in play.

Preload Art - By default, this option is enabled. This setting informs GameSalad that you want the graphic file to be loaded at the start. If you want to wait, preferring to load the actor image when the actor first appears on the screen, then uncheck this option.

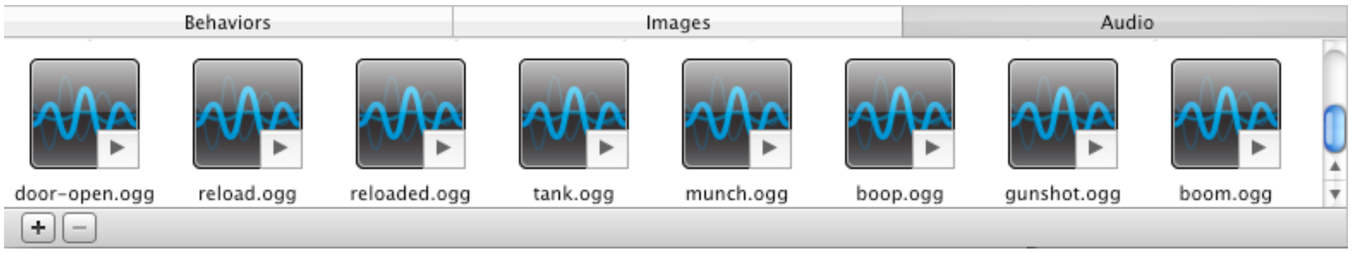
If you're having trouble creating graphics for your projects, there is an alternative. You could hire a freelancer to create the graphics for you. There's a "Jobs" forum on the GameSalad website. There are also other sites online where you can hire a freelancer. GameSalad added the Marketplace to make it easy to get content for your games. In the "Images" tab of the GameSalad Creator library, there's the "Purchase Images…" button. You can license content for your games directly from GameSalad.

Chapter #11 Summary

- Even if your artistic abilities are limited to tracing, you still might be able to create original artwork. If not, the Internet is loaded with free graphics programs and plenty of places to legally acquire artwork.
- All images imported into GameSalad are converted to png files, regardless of the original format. To preserve quality and image control, it's a good idea to create your artwork in png format.
- GameSalad uses the RGBA color space. That allows for transparency, color tinting and blending effects.
- To save memory, and increase performance, you can "Tile" an image instead of using a much larger graphic.

Chapter #12 - Audio

With GameSalad, you can drag-and-drop sound and music files into your project. They will appear in the "Audio" tab of the "Media Panel". Like with images, you have to be careful about the size of your audio files. Otherwise, it may hinder the performance of your game.



What if you don't have any sound effects or audio files? You can make some or buy some. The first thing you might want to do is to grab some software for audio editing. If you're creating a game for the iPod Touch or iPhone, you'll probably need to downsample (reduce in quality) your audio files.

I recommend Audacity for audio editing. It is open source and cross-platform software. That means you can use a Windows PC or a Mac OS computer to create and edit audio files.

- Audacity - http://audacity.sourceforge.net

It is loaded with ways to add effects to your audio files, it can easily resample audio files and it can be used for audio editing. It's also create for recording voice overs and other sound samples. As an example, I used audacity for recording an explosion. I put a microphone next to a gas stove. I turned on the gas and then lit a match.

- BOOM -

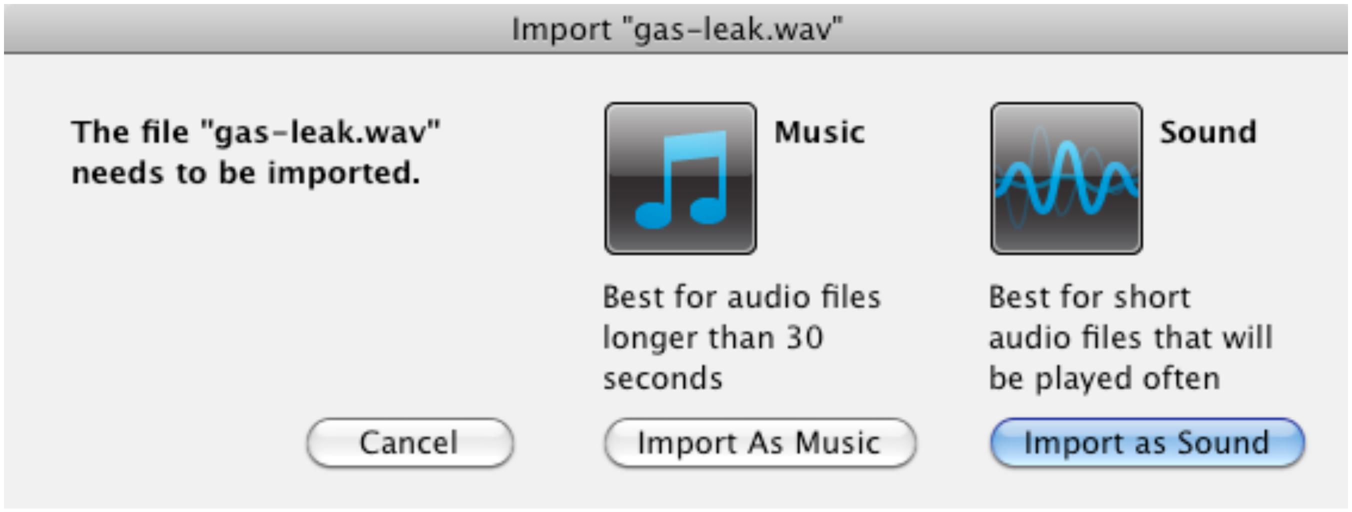
Obviously I lived. The final sound effect was used in several of my games. Perhaps your sound recording techniques should be less dangerous. Simply tapping some pots or wooden sticks can yield interesting results. You can record some decent sounds effects just by roaming around with a laptop, a copy of Audacity and a decent microphone. As you experiment with different types of material, you might want to be mindful of your friends, family and neighbors. A project like this can get loud. I recommend reducing the background as much as possible. That will help with recording clean sound effects.

Once you have your raw audio track, you'll probably want to downsample it to 22050 Hz and 16-bit. Anything higher might be overkill. Players may not be using headphones, so they may not even be able to hear the difference. That higher quality audio is taking up system resources that could be better used on other game elements.

There's also tough decision to make. Should your audio be in stereo? If you think that the majority of your iPhone, iPod Touch or iPad players will not be using headphones, you might be wasting system resources. With my games, I didn't seem to care. I spent a lot of money on the audio files and I wanted the players to hear the music in stereo — even if I was the only player who would ever hear it. You can get attached to your game's content, so it can be difficult to remain objective. Performance versus quality is a constant struggle in developing games for mobile devices.

One thing to be sure of, if you're recording sound effects with a mono microphone, the sound files do not need to be saved as stereo sound — it wasn't recorded as stereo sound in the first place. Such a sound file can be flattened to a mono sound file.

Audacity exports to wav format. Such files can then be dragged into the "Audio" tab of the "Media Panel". When you do that, the following screen should appear…


Import "gas-leak.wav"

The file "gas-leak.wav" needs to be imported.

Music

Best for audio files longer than 30 seconds

Sound

Best for short audio files that will be played often

If you select "Import As Music", the audio file will be converted to the m4a format and it will only be available to the "Play Music" behavior. If you select "Import as Sound", the audio file will be converted to the ogg format and it will only be available to the "Play Sound" behavior.

There is another format for "Sound" files. You could add caf files to GameSalad. Supposedly, this format improves performance, but I haven't really noticed a difference. If you would like to experiment for yourself, Audacity can export to the caf format, but you'll need the 1.3 version. When you export the file, select the following options…

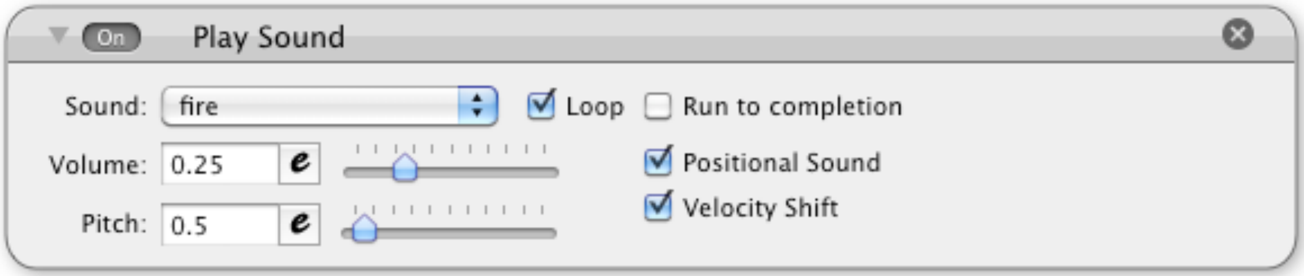
- Other uncompressed files
- CAF (Apple Core Audio File)
- Signed 8 bit PCM

8-bit files are significantly smaller. If you're trying to recreate a traditional video game, with that authentic beeping sound, then maybe caf can work for you.

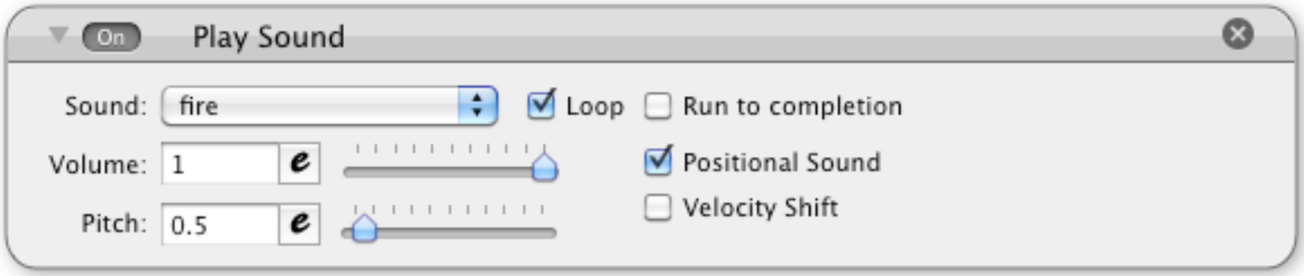
Although, if you're interested in cross-platform compatibility, caf is a bad format choice. By adding images and sounds through the Windows version of GameSalad, the accepted media file formats are listed...

Media Format	Type
.png .jpg .jpeg .gif .tiff .wdp .bmp	Image file (PNG recommended)
.ogg .wav .m4a	Audio file

When making audio files, Record and save your files so that they're nice and loud. It shouldn't be too loud that you get distortion, but certainly loud enough to be heard. You can't make a file louder in GameSalad. You can only play it at 100% "Volume" or less. That's why it's important to start high. If a sound file is too loud, you (or the player) can throttle the volume back.

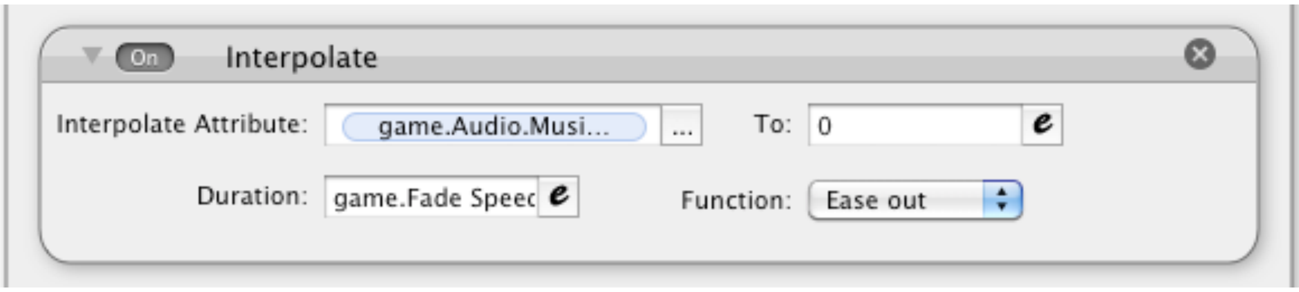


Sound: fire
Loop [✓]
Run to completion [ ]
Volume: 0.25
Positional Sound [✓]
Velocity Shift [✓]
Pitch: 0.5

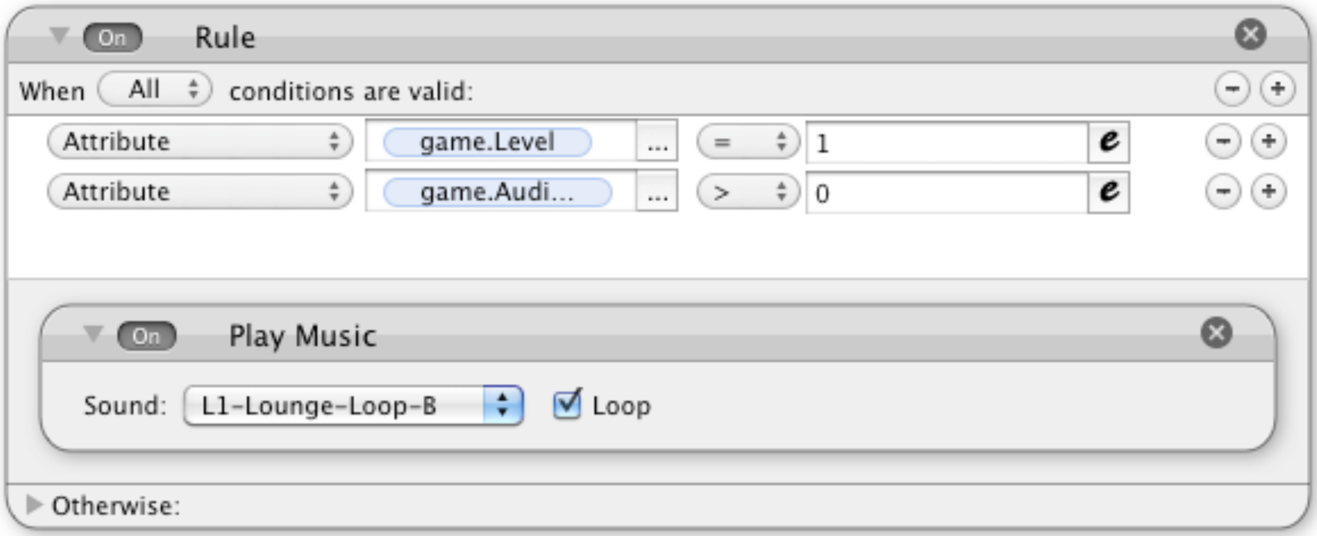


Sound: fire
Loop [✓]
Run to completion [ ]
Volume: 1
Positional Sound [✓]
Velocity Shift [ ]
Pitch: 0.5

Even though these are just audio files, you can still create some tricks to liven them up. In the above example, I'm using two "Play Sound" behaviors to create sound effects for a fireball. If you look carefully, you can see the difference. Both are using "Positional Sound" because it's a fireball that bounces around the screen. I enjoy creating the swooshing sound, from the left ear to the right ear. I think it adds some depth to the game. I also like to use "Velocity Shift" option, as it's an easy way to add bouncing sound effects. However, I don't want that sound to be so harsh. I lowered that "Behavior" to 25% "Volume".


Interpolate
Interpolate Attribute: game.Audio.Musi... To: 0
Duration: game.Fade Speec Function: Ease out

Here's another audio trick. The "Interpolate" behavior can be used to fade the music volume. Such a technique could be applied to dynamic battle music. When the hero is in battle, a sound file could be faded in. By changing the music to something more dramatic, you can make combat more interesting. Then, when the enemies are defeated, the combat music could fade out and the regular music could fade back in. Two "Play Music" behaviors cannot be used at once. However, a short music loop can be played with the "Play Sound" behavior. This is an important use of music. It helps to create the mood. By careful use of the "Volume" and "Pitch" settings, you can help enforce the theme of your game.


Rule
When All conditions are valid:
Attribute game.Level = 1
Attribute game.Audi... > 0
Play Music
Sound: L1-Lounge-Loop-B ✓ Loop
Otherwise:

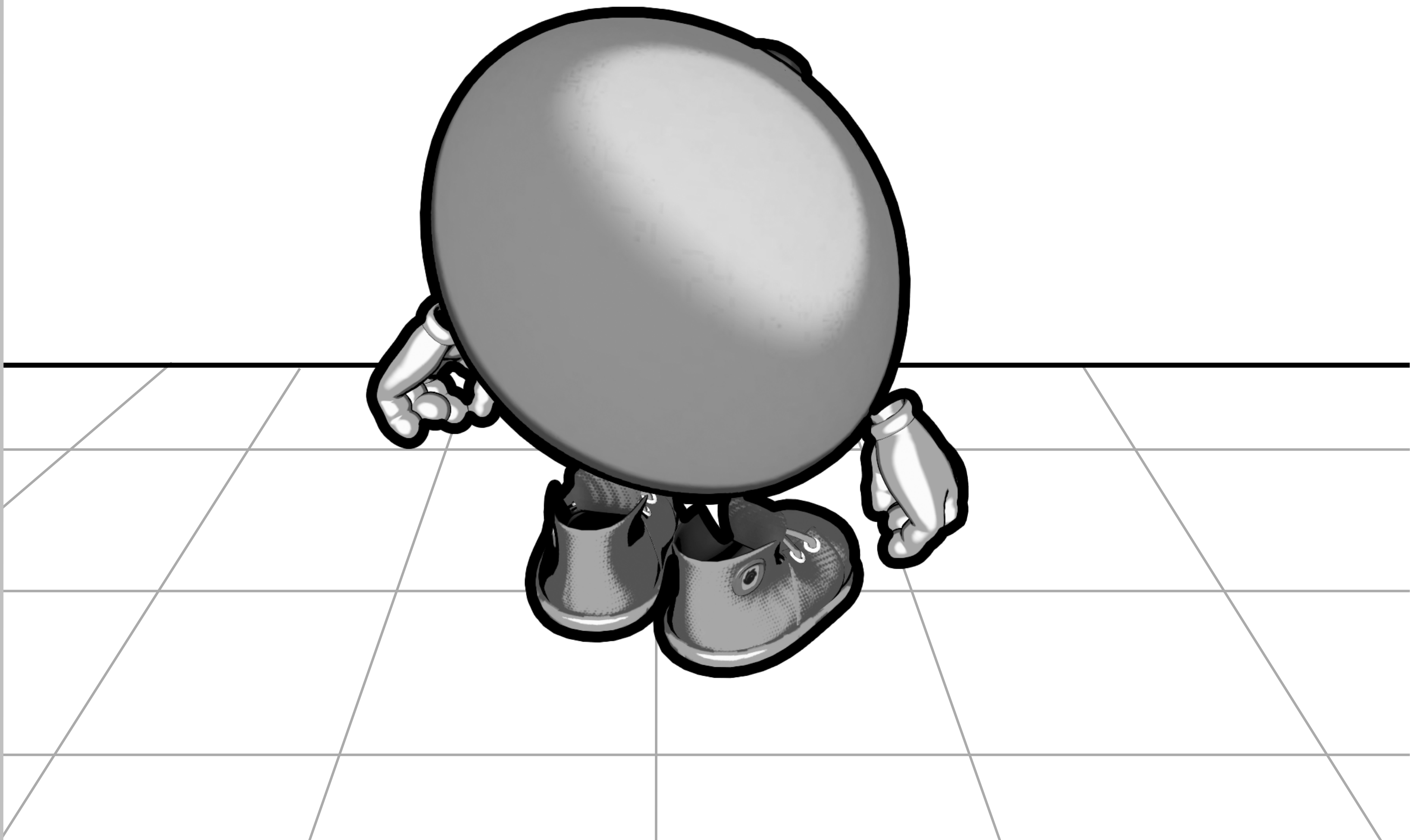
In the above example, the background music is only played during level one. The other condition is that the game.Audio.Music Volume setting has to be greater than zero. There's no need to play music that can't be heard.

Managing audio files in GameSalad is not as complicated as managing image files. There are simply less options for audio files. However, this is still an important piece of game design. A great sounding game can make the experience more memorable and more fun.

Chapter #12 Summary

- GameSalad has separated the audio library into two parts — Music and Sound.
- Music files are saved in the m4a format.
- Sound files are saved in the ogg format
- Audacity is a great companion for GameSalad, allowing you to create, and modify audio files.
- Just as image files should be optimized, audio files should be optimized too. Doing so can increase the performance and reduce the file size of your game.

5

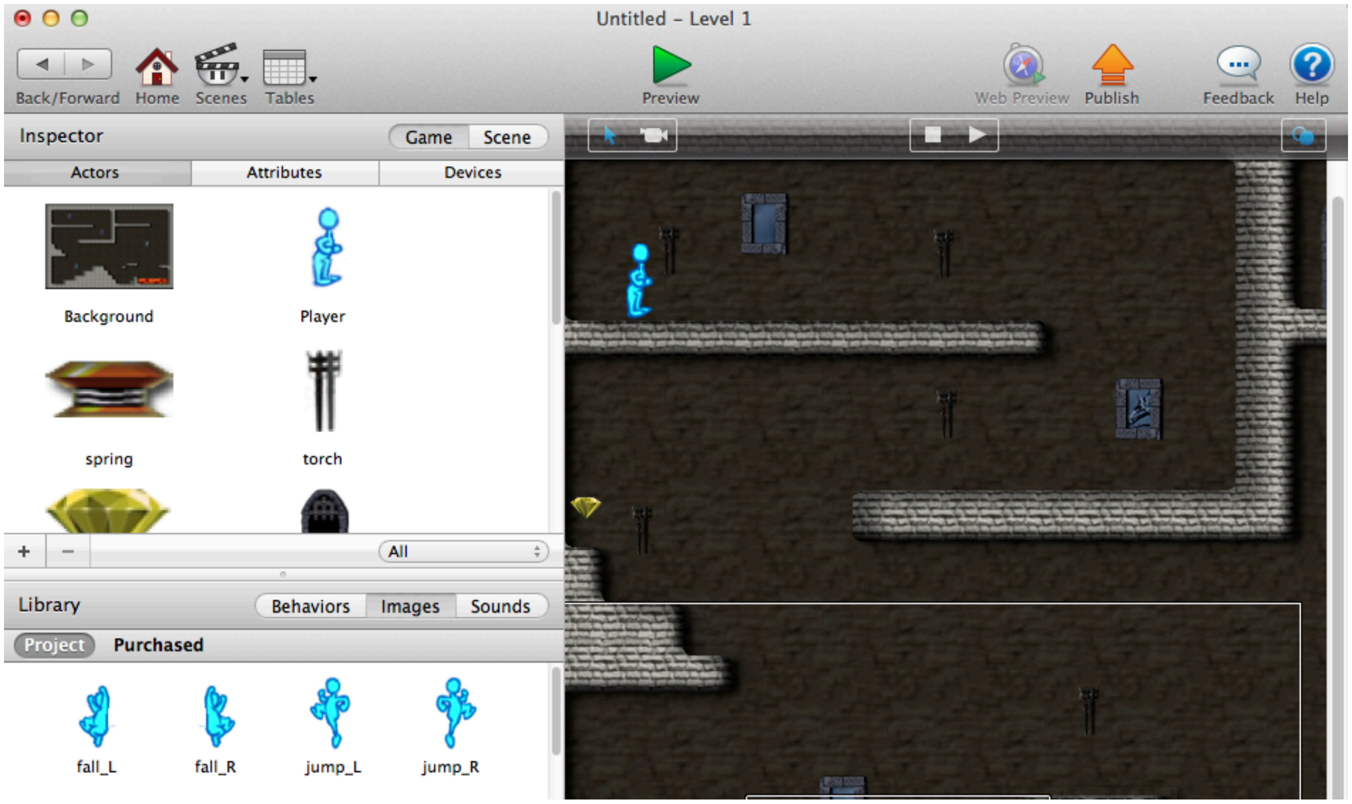


Section V - Game Examples

13 - Platformer
14 - Space Shooter
15 - Bouncing Balls
16 - Puzzle Games

Chapter #13 - Platformer

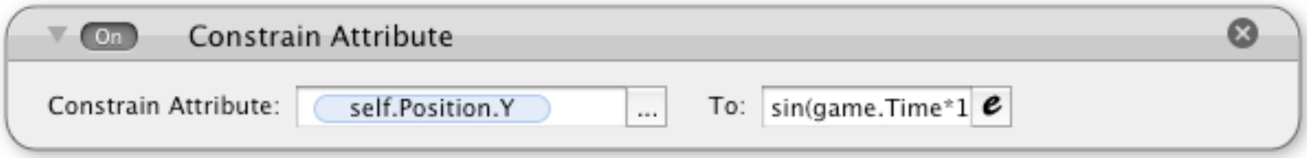
If you're a child of the 8-bit gaming era, you might be dreaming about your own "Platformer". It's a significant project. This chapter aims to highlight the pieces needed for creating a game similar to Super Mario Bros. or Mega Man. You also might want to review the "Platformer Template" before creating your own game. The default template shows how to achieve many of the basic concepts of this genre.



Platforms - Since this is a platformer, I figure we could start from the ground up. First, set your scene size. How big should it be? That's really up to you. The objective is to give the player a sense of exploration without creating a game that performs poorly. Once your scene is set up, you can start adding actors into it. I like starting with the ground because I don't want my main character to fall through the world.

There are two main types of platforms that you can create — moving and unmovable. When creating the ground actors of your platformer, they should probably be unmovable. You can use a series of ground actors, with some spaces between them, to create gaps. That way, your character has to jump over them. With the "Movable" option unchecked, the main actor cannot push them away. While you're modifying the "Physics" section, you might want to modify the

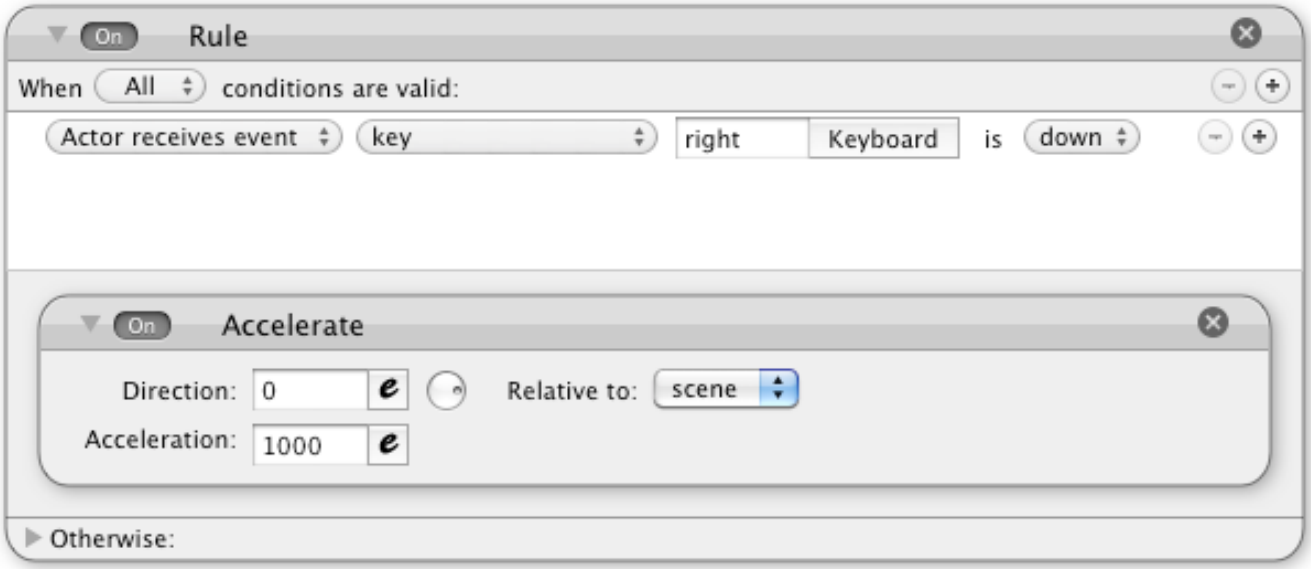
"Restitution" setting. You don't want concrete or dirt floors to be bouncy. You also might want to "Tile" your ground graphics to save memory and increase performance. For platforms that move, they can still have their "Physics" set to unmovable. Just use the "Interpolation" behavior to create the effects you're looking for. Also, you could use "Functions" and constraints to make your platforms move in a circular or linear pattern.


Constrain Attribute

Constrain Attribute: self.Position.Y To: sin(game.Time*1

Look out! It's trigonometry!

Moving - With a general playing field established, now you can add your main character to start testing it. Obviously, your character will need to move. The "Accelerate" behavior seems to work best, but you can try the "Move" behavior to see if it works for your game. You'll probably have to play with the settings until you get good combination for character movement. Don't forget about the "Drag" and "Gravity" settings. Otherwise, your character won't stop. You'll also have to determine the level of "Friction" on your actors.


Rule

When All conditions are valid:

Actor receives event key right Keyboard is down

Accelerate

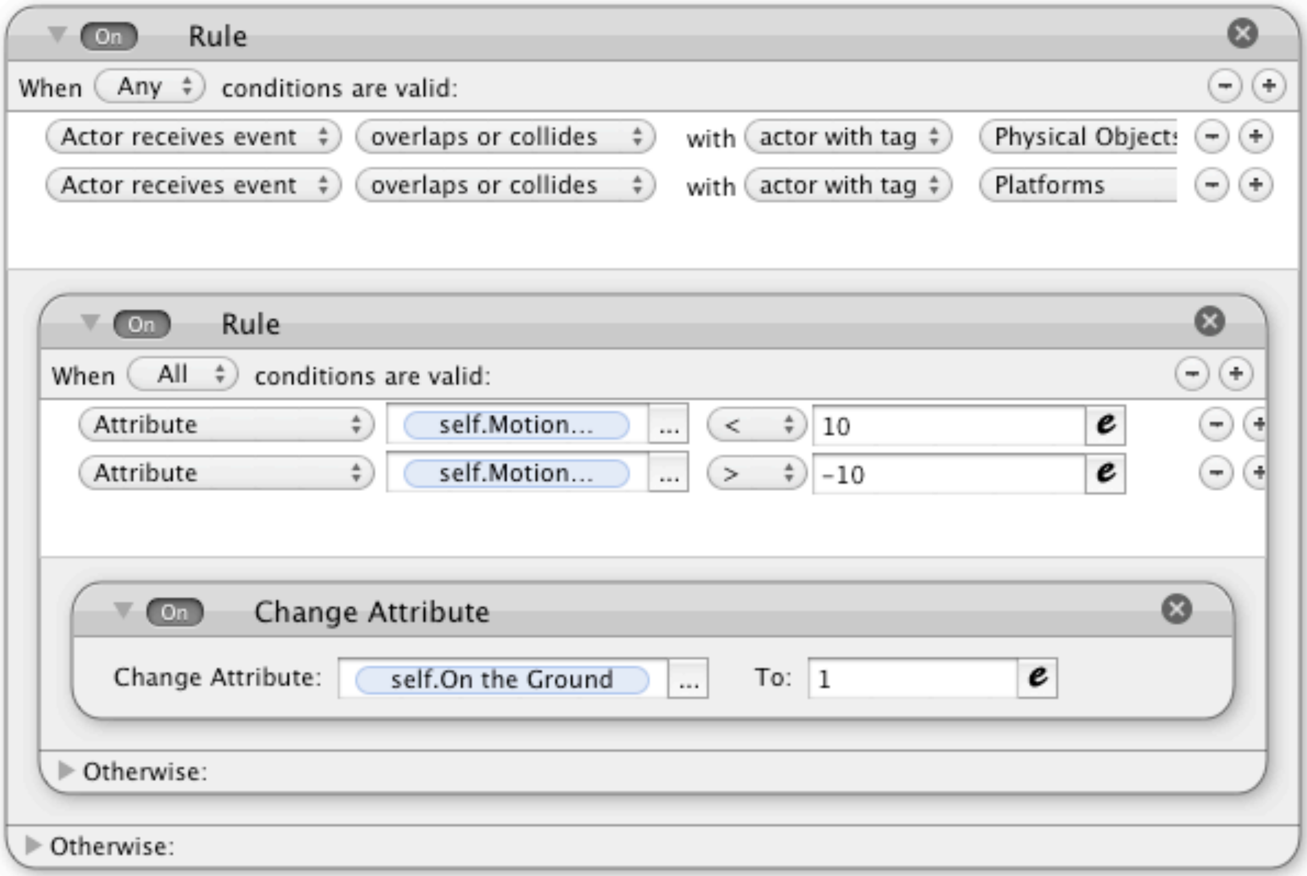
Direction: 0 Relative to: scene

Acceleration: 1000

Otherwise:

A big problem with moving is that it has to happen with the ground. Maybe your character can move slightly while flying through the air, but running and jumping should only occur from the ground. That means your character will have to detect if it's on the ground or not.

The tutorial included with GameSalad shows how to accomplish this task. In the tutorial, two conditions are monitored to determine if the actor is safely on the ground. Whenever the actor touches a ground actor, and when self.Motion.Linear Velocity.Y is close to zero, then the game considers the actor to be touching the ground.



Rule

When Any conditions are valid:

Actor receives event | overlaps or collides | with | actor with tag | Physical Objects

Actor receives event | overlaps or collides | with | actor with tag | Platforms

Rule

When All conditions are valid:

Attribute | self.Motion... | < | 10

Attribute | self.Motion... | > | -10

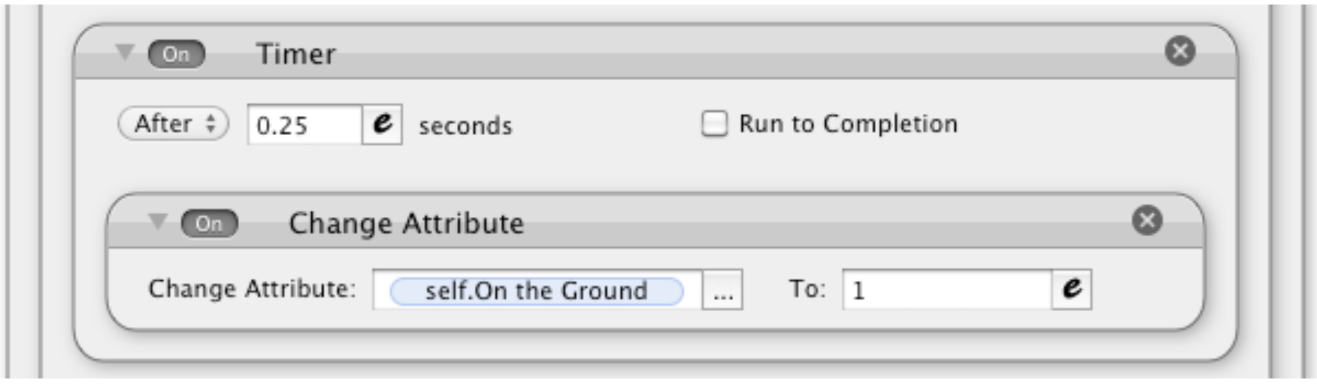
Change Attribute

Change Attribute: self.On the Ground To: 1

Otherwise:

Otherwise:

There's a problem with that approach. The actor can still jump while floating in the air. When the actor reaches the height of a jump, the Y velocity value is near zero. At that point, the actor only needs to be touching the side of a platform to jump again.



Timer

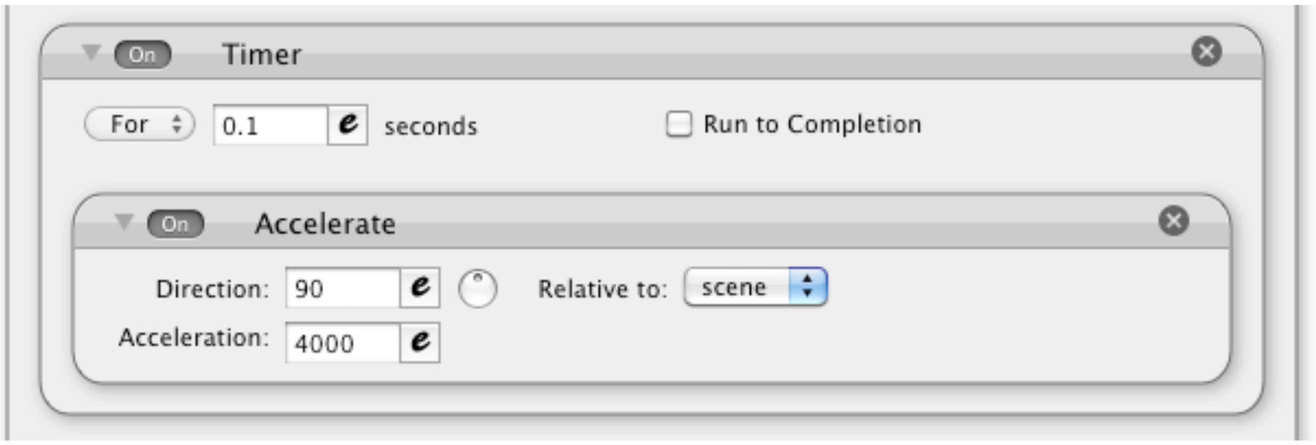
After 0.25 seconds ☐ Run to Completion

Change Attribute

Change Attribute: self.On the Ground To: 1

I fixed that issue by throwing a 0.1 second “Timer” on the “Change Attribute” behavior. With the “Run to completion” option unchecked, that tells the actor it needs to maintain near zero Y velocity before jumping again. This is a crude, but fairly effective fix. It’s not the only way to resolve this issue. The point is that you’ll probably run into issues with jumping. This is your opportunity to test and try new things. By experimenting with different “Behaviors”, you can learn more about the software and become a better game developer.

Jumping - The tutorial shows a pretty straightforward way to handle jumping. When the spacebar is pressed, and if the actor is considered to be on the ground, then “Accelerate” upwards for a split second.

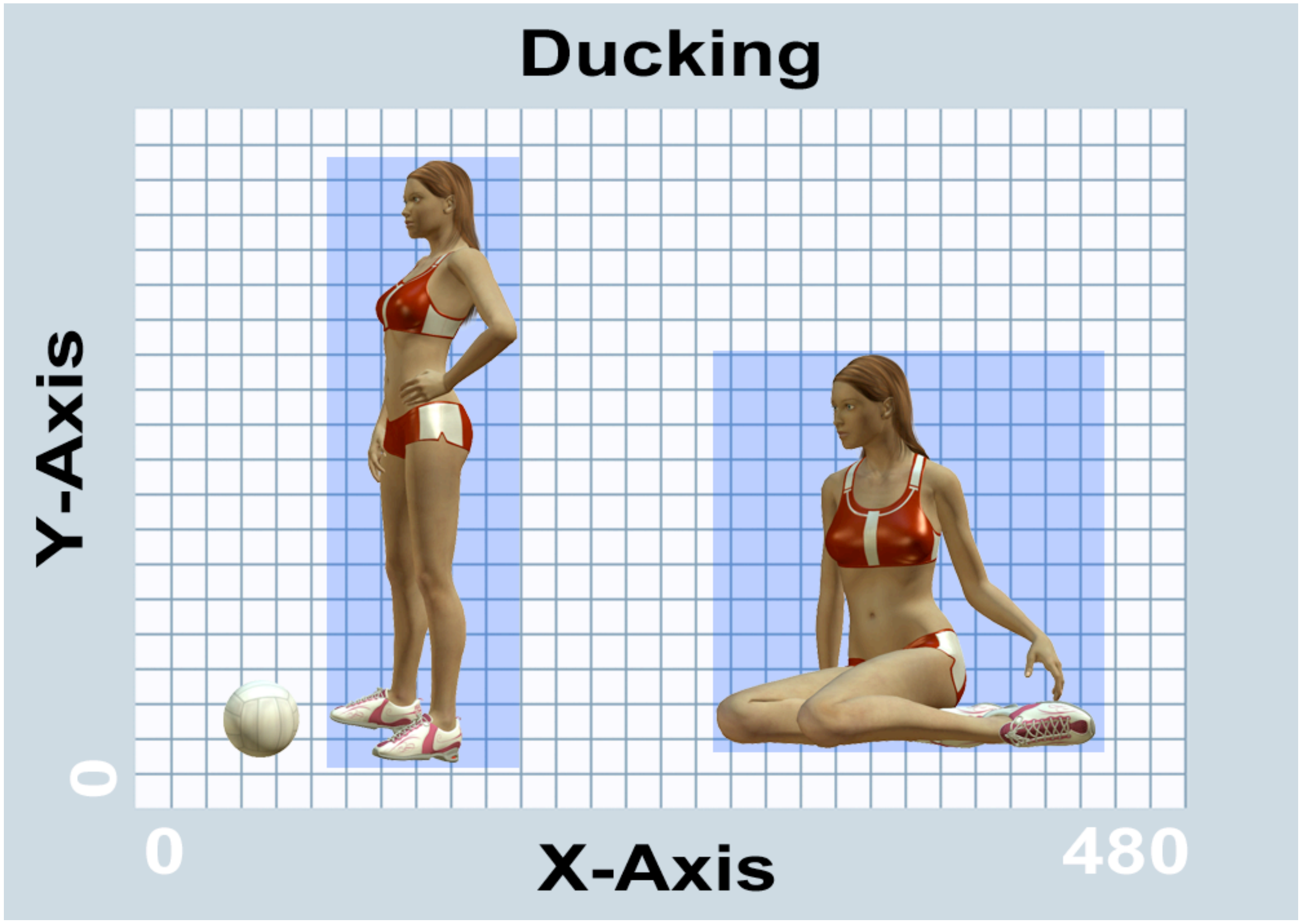

Timer
For 0.1 seconds
☐ Run to Completion
Accelerate
Direction: 90
Relative to: scene
Acceleration: 4000

The upwards motion has to fight “Gravity”. In the example shown above, I bumped up the “Acceleration” value to 4000. I wanted the actor to jump higher. It’s a delicate balance between the two values. If “Gravity” is too strong, the actor won’t jump very high — if at all. If the “Acceleration” value is too high, then the character will jump unrealistically.

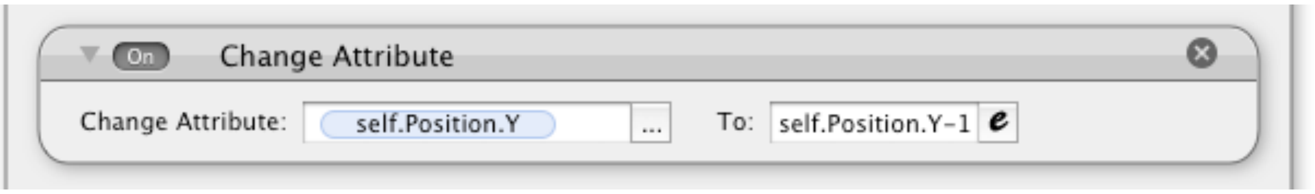
You can play with these values to get the settings you want. Also, you can change these settings in the game. If an actor receives an item that boosts jumping power, the “Acceleration” value can be increased. If you’re creating a fantasy game, the “Gravity” value can be flipped or altered. For example, you could make a moon level with higher jumping. Each “Scene” can have its own “Gravity” settings or you could change the value while the game is active.

The “Jumping” template is not only an example of vertical jumping, but it also shows how to add “Wall Jumping”. In many platformers, the hero can slide down the side of a wall and then kick/jump off it. By using three invisible actors — to the left, bottom and right of the main actor — the main actor can jump accordingly.

Ducking - The trick to making an actor duck or crouch is to change two characteristics of the actor at the same time — the "Image" and the "Size". With a "Rule" you can make an actor duck when the down arrow is pressed. Then when released, return to normal. When the down arrow is down, use the "Change Image" behavior. You could also use the "Animate" behavior, but that might cause issues with collision detection. The main idea is to change the graphic to match the action. You'll also want to change the shape of the actor to match the new image. In the example shown here, the actor's bounding box tries to match the size of the actor. The gray area shows where collision can occur. Your images should closely match the collision area.



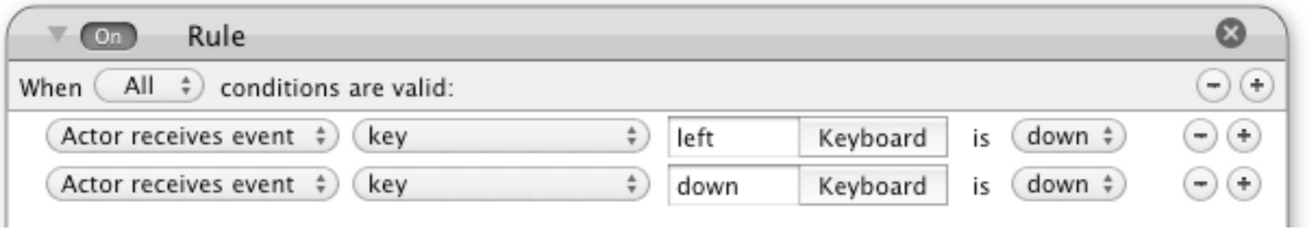
So that's it right? It's a piece of cake! Actually, no. There are several issues with ducking that need to be resolved. The first issue is that the actor will duck awkwardly. That's because GameSalad actors have a center point. When the actor's size is changed, it grows or shrinks towards the middle. If you use the "Change Attribute" to instantly reduce the actor's size, the actor is going to float over the ground and then drop down to the ground with "Gravity".


Change Attribute

Change Attribute: self.Position.Y To: self.Position.Y-1

For a more natural effect, change the self.Position.Y value to align the new actor size with the ground. You probably won't need an offset for when the actor is standing up, as collision detection should instantly put the actor above the ground.

Another issue with ducking is movement. Your character shouldn't be able to move at full speed while crouching. You can set up some "Rules" to your movement "Behaviors" to balance out that issue. The below example simply prevents left acceleration while the down arrow is pressed.


Rule

When All conditions are valid:

Actor receives event key left Keyboard is down

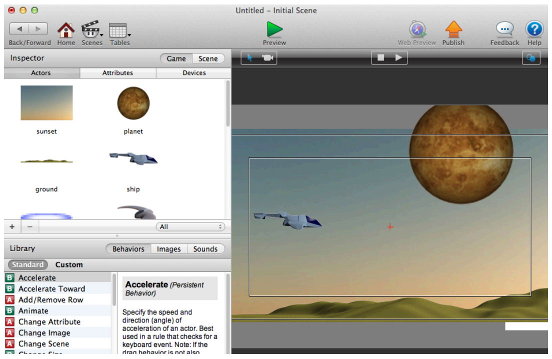
Actor receives event key down Keyboard is down

Coins - This is fairly straightforward, but also a key part of a platformer game. The hero doesn't just save the damsel in distress, he also collects loot. You can add coins and power-ups to your games by creating specific actors for that purpose. Then, when the hero "overlaps or collides" with the bonus item, the appropriate "Attributes" can be changed. You could increase the player's score or create bonus items.

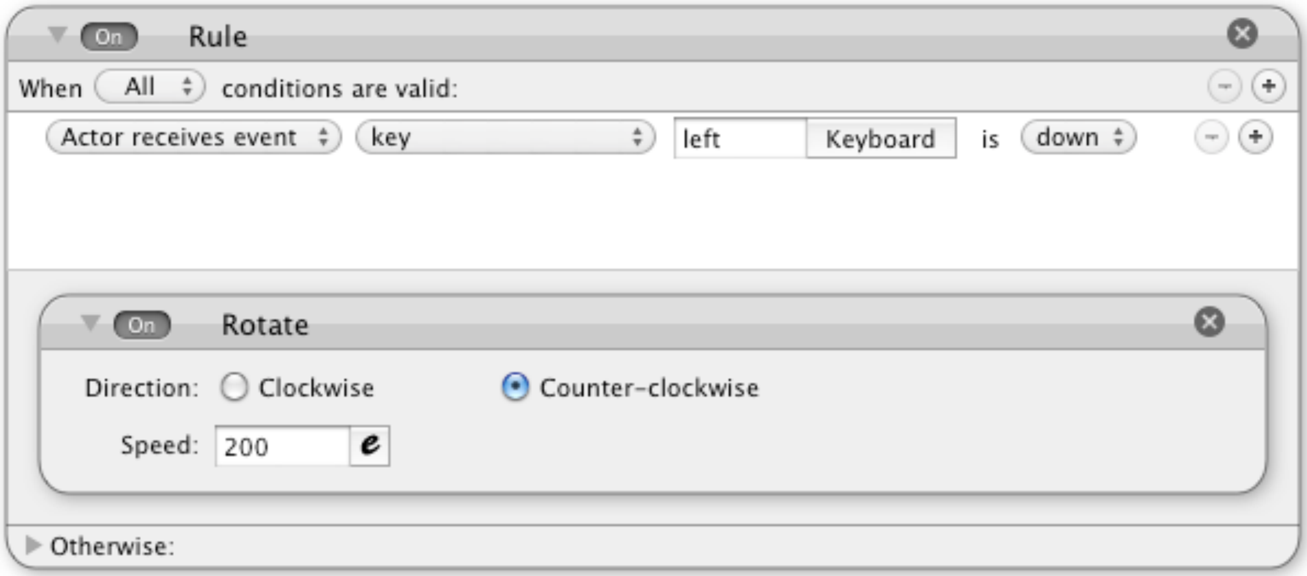
Goal - What is going to end the level? What enemies will the hero fight? Is there a boss at the end of the level? These are questions for the design stage of your platformer. Each level could be a separate "scene" in your GameSalad project. Then, when the level is completed, the actor can move to the next level. That's easily accomplished with the "Change Scene" behavior. However, for something less instantaneous, you could add actors to create some sort of end-level sequence. Show the player's score, things accomplished, a fireworks display or something encouraging. It's a good idea to make the player feel like they've accomplished something. That makes the game more entertaining.

Chapter #14 - Space Shooter

There are two main types of Space Shooters that can be created with GameSalad. If you look at the “Space Rocks” tutorial, you can see an example of a top-down shooter. If you look at the “Basic Shoot Em Up” template, you can see an example of a side-scroller. You don’t have to choose. You can make a game that has both side-scrolling and top-down levels.



The ironic thing about the example for a side-scrolling shooter is that it doesn’t actually scroll. Instead, enemies are spawned by the actor on the bottom right. Do you see that white box? It’s just outside the screen. With tricks like that, you can make an endless level. Enemies are just chucked into the scene. The hero shoots at them. This process can continue until a certain requirement has been met or the actor is destroyed. The controls for a side-scroller are straightforward. It’s up, down, left, right and shoot. This is an excellent project for beginners as it’s easy to manage and build. Things can get a little confusing if you’re creating a top-down shooter.



On Rule

When All conditions are valid:

Actor receives event key left Keyboard is down

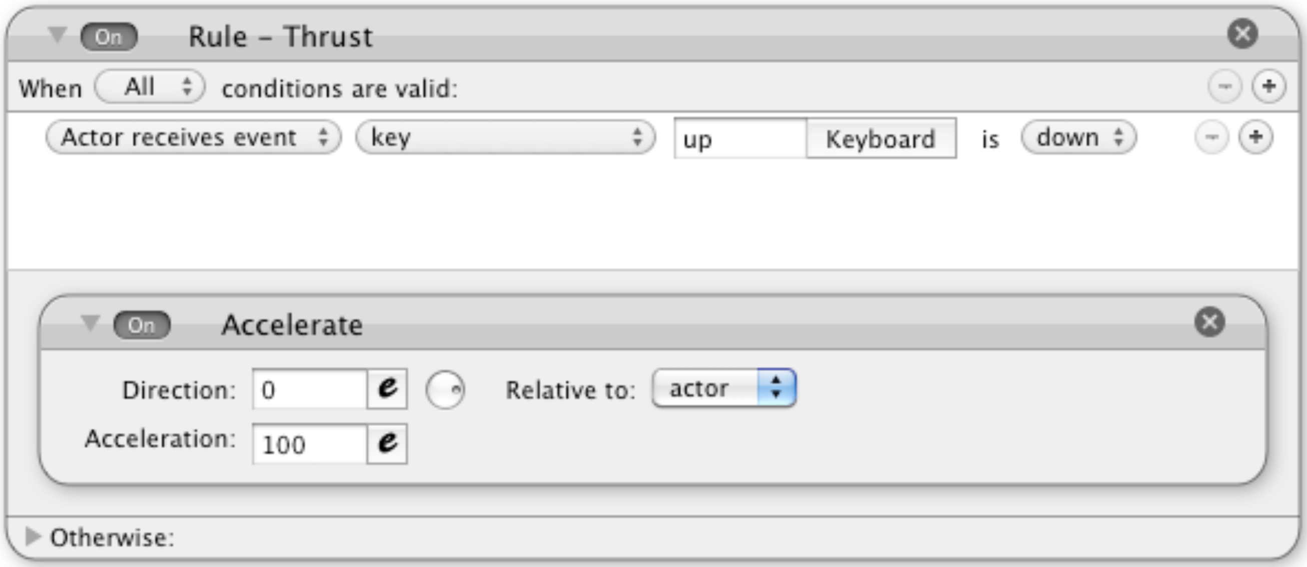
On Rotate

Direction: Clockwise [✓ Counter-clockwise]

Speed: 200

Otherwise:

Instead of simply moving in a direction, a top-down shooter can have a rotating actor. In the "Space Rocks" tutorial, the actor rotates when the left or right keys are pressed.


Rule – Thrust

When All conditions are valid:

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

Accelerate

Direction: 0 | Relative to: actor

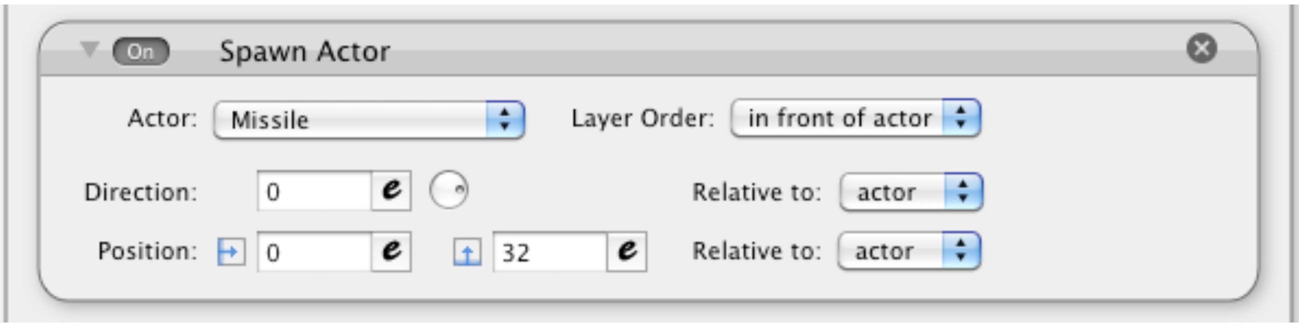
Acceleration: 100

Otherwise:

The above image shows how to create thrust with the “Accelerate” Behavior. By mixing thrust with rotation, the actor is free to move in a 360° range. When the up key is pressed, the actor can “Accelerate” in a “Direction” that’s “Relative to” the “actor”. Once again, it’s important to align the image with the 0° mark. Otherwise, the thrust might not match the direction that the actor is facing.

This control scheme mimics rocket thrust. You might notice that the "Space Rocks” example doesn’t actually include an active thruster effect. That’s because some advanced techniques are required. For more information about this effect, see Chapter #24 - Particles.

Shooting - In both tutorials, shooting is handled the same way. An actor is spawned when the spacebar is pressed. That bullet actor has a “Behavior” to create forward movement. That actor is also set to “Destroy” itself after a certain amount of time has passed. The enemy actors in the game are set to “Destroy” themselves if they “overlap or collide” with the bullet actors. The sound can be played from the main actor or the bullet itself. This shooting example has the same general idea as the one covered in Chapter #8. However there is a slight difference with the “Space Rocks” example.


Spawn Actor

Actor: Missile

Layer Order: in front of actor

Direction: 0

Relative to: actor

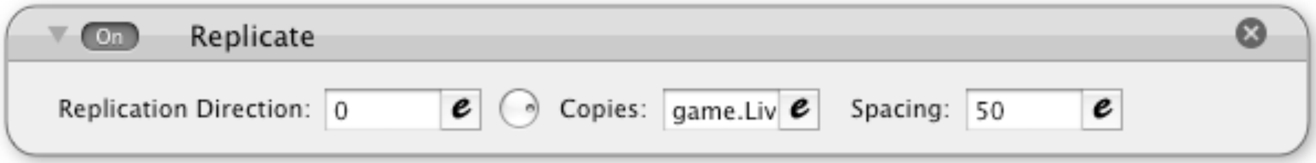
Position: 0 32

Relative to: actor

There’s a Y offset of 32 pixels in the “Spawn Actor” behavior. The missiles are actually being spawned “in front of actor”. In Chapter #8, the projectiles were spawned below the actor and from the center of the actor. Either method is acceptable. The general ideal is to create a more realistic look for the projectiles. It should appear as if the missiles come from a launcher.

Debris - It’s not enough just to make an actor disappear. Something spectacular should happen. In the “Space Rocks” example, the larger asteroids break up and become smaller asteroids. That’s not what’s actually happening. Just before the larger actor is destroyed, it creates two smaller actors. (Note: The “Destroy” behavior should be put at the bottom of the container. Otherwise, the “Behaviors” below the “Destroy” behavior may not activate.”) By using this method, you can create an array of effects.

For example, you can use the “Animate” behavior to create an explosion effect. This explosion could have collision detection. The explosion could harm the main actor. You could also spawn bonus items randomly, such as weapon power ups or score bonuses. If you wanted to be fancy, you could show the enemy pilots ejecting from their ruined aircraft.

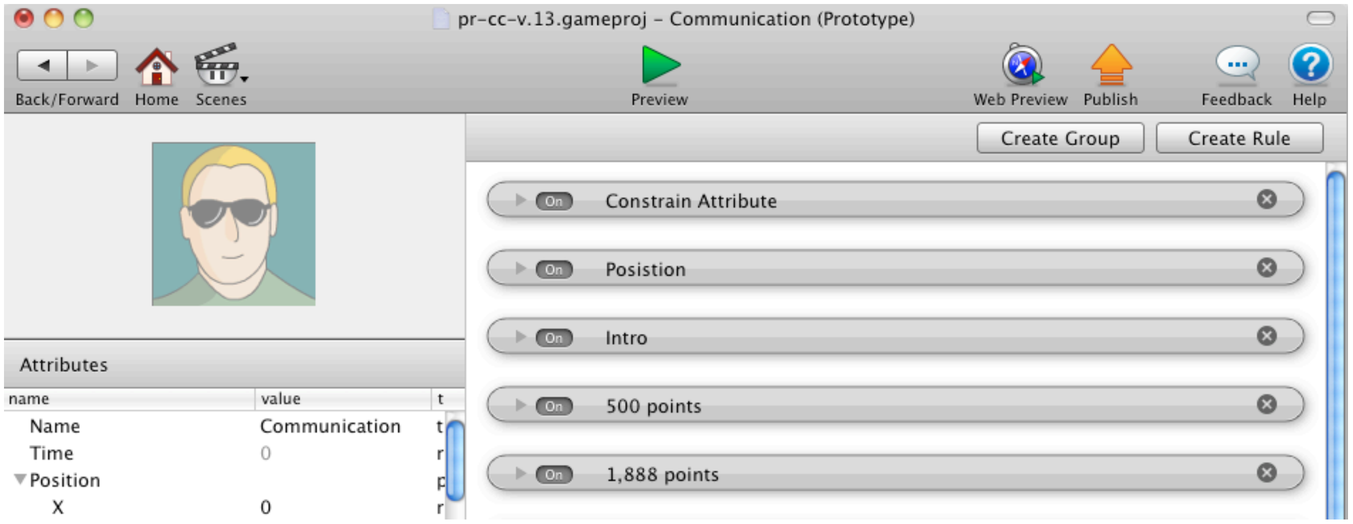

Replicate

Replication Direction: 0

Copies: game.Liv

Spacing: 50

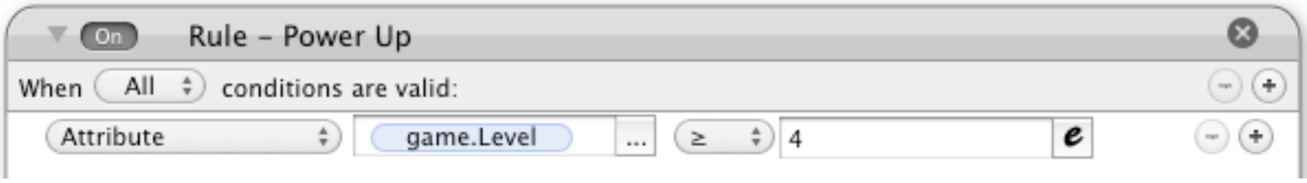
Lives - The “Space Rocks” example shows the “Replicate” behavior in action. A separate actor is created, specifically for displaying the number of lives remaining.



On Constrain Attribute
On Posistion
On Intro
On 500 points
On 1,888 points

name	value	t
Name	Communication	t
Time	0	r
▼Position		p
X	0	r

Another thing to remember is to add excitement to your game. Players have been shooting at sprites for more than three decades. What's going to give your shooter more value than the countless games that came before it? You can spice things up with a story. You can make things more interesting by actually communicating with the player. At key points of the game, the main characters could say something interesting. Encouragement is a big part of gaming. It's important to remember to make the player feel happy, and part of the game.



On Rule – Power Up

When All conditions are valid:

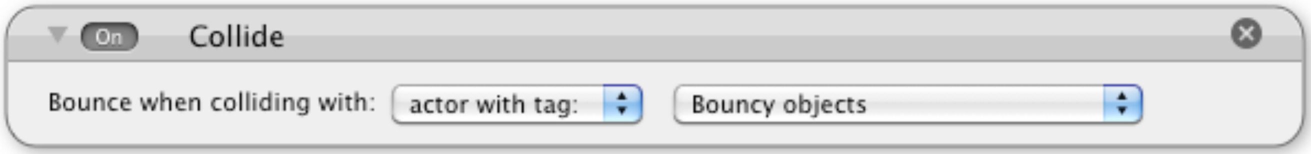
Attribute game.Level ≥ 4

Power Ups - You can create upgrades to your weapons in different ways. They could be acquired from slain enemies or you could create a weapons shop. To create a power up, create the conditions with a "Rule" behavior and modify your actors accordingly. Adding power-ups to your game is another way to break up the monotony of a space shooter. It can also give the players a good reason to keep playing.

Chapter #15 - Bouncing Balls

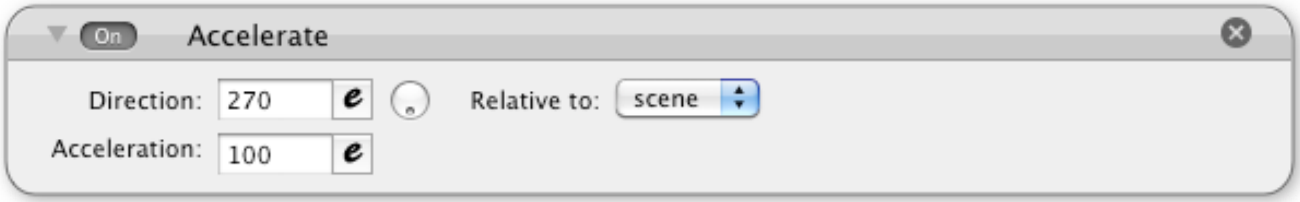
I remember the days of the bouncing square. So many different games were based on this simple concept. Oh look, it's a tank game. It's a tennis game. It's an old west shoot out. It's a brick smasher… no, it's just a bouncing square. The hardware was limited. The format was new. It created a glut of bouncing block games. About three decades later, things have improved. We can make better circles, but the glut has returned. The iTunes Store is loaded with clones. Should you really bother with this genre of games? If you think you're going to wow the iTunes App Store with your modified version of the "Crazy Ball Wall Breaker" tutorial, you might want to think again. The basic economics of supply and demand is in action.

However, this is a great tutorial. If you're going to learn the trade, it's a good idea to start at the beginning. To create a ball with GameSalad is ridiculously easy.



Bounce when colliding with: actor with tag: Bouncy objects

With the "Collide" behavior, you can make actors bounce off each other. You can select objects by actor name or tags. For managing collisions between multiple actors, tags might make it easier. That's basically it, but there are some things that you might want to add to make the effect more impressive. One, you'll need to add energy to your scene. The ball needs to move. The easiest way to hand that problem is to add "Gravity" to your scene. I usually go with an alternate method for "Gravity". I use the "Accelerate" behavior to simulate the same effect.

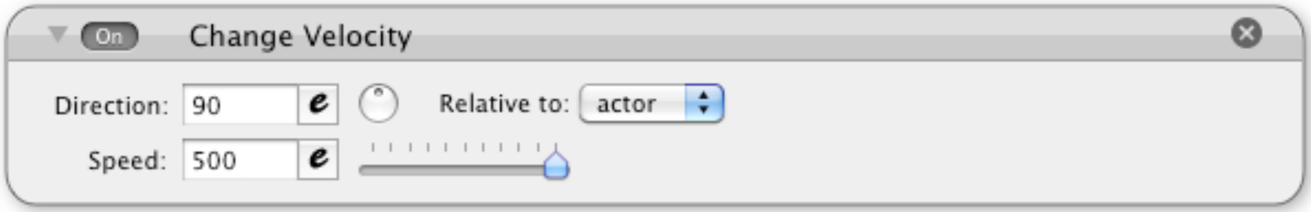


Direction: 270
Relative to: scene
Acceleration: 100

By setting the "Accelerate" behavior "Relative to" the "scene" and a 270° "Direction", the ball moves downwards. If you're creating a pinball game, the ball is probably the only actor that needs "Gravity". Regardless of which method you choose, you'll probably want to deselect the

"Movable" option on your walls. This will make your game more efficient and it will keep the walls stationary.

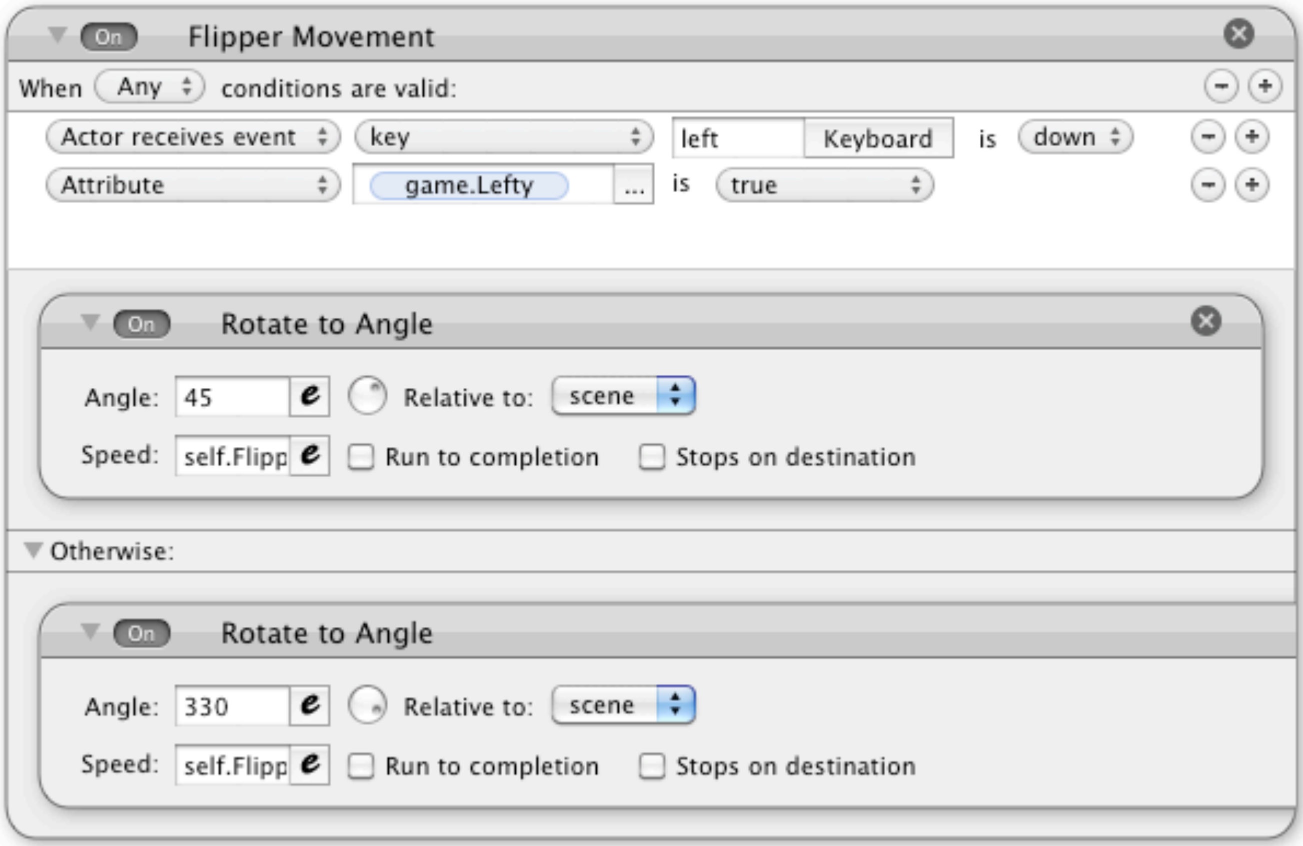
The game still needs more energy. For example, the ball should have some sort of initial movement. Maybe the pinball was launched into the scene by the plunger or maybe a paddle knocked the ball upwards. This can be achieved with the "Change Velocity" behavior.


Change Velocity
Direction: 90
Relative to: actor
Speed: 500

The above example gives the ball an initial "Speed" of 500. The 90° angle is opposite of the force that opposes it. This will cause the ball to move upwards, but "Gravity" will eventually take over. The "Change Velocity" behavior is not constant. It is an "Action" behavior. The upward movement is lost to the constant effect of the "Accelerate" behavior. But that's OK, it's supposed to do that. The ball can bounce upwards again when it hits the ground.

That's why you might want to tweak your "Physics" and "Motion" settings. What should be the ball's top speed? How bouncy should the ball be? Another huge question, should the ball spin? By enabling "Fixed Rotation", you can get different ball movement. It depends on your game, but I usually let the ball spin. I think that the movement seems more natural that way. If you're building a ball, you'll probably want to set the "Collision Shape" to "Circle". It's important that your graphics align with the "Collision Shape", or the bouncing movement might seem unrealistic to the player. The same is true for the walls and obstacles in your game.

Even though the bounce originates from the "Collide" behavior, there are many other settings to manage. Testing is important, to see if your game feels natural and fun to play.


On Flipper Movement

When Any conditions are valid:

Actor receives event key left Keyboard is down

Attribute game.Lefty ... is true

On Rotate to Angle

Angle: 45 Relative to: scene

Speed: self.Flipp ☐ Run to completion ☐ Stops on destination

Otherwise:

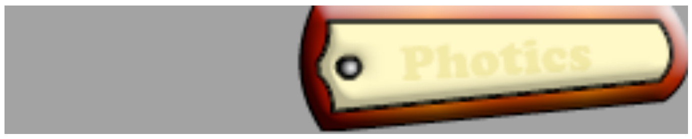
On Rotate to Angle

Angle: 330 Relative to: scene

Speed: self.Flipp ☐ Run to completion ☐ Stops on destination

If you want to create a pinball flipper, there are two main tasks to complete. First, the flipper needs movement control, just like a regular actor. It starts the same way - with a "Rule". When a "key" is pressed, or another condition is met, then activate the "Behaviors" in the container. "Otherwise", active the "Behaviors" in the other container. By using the "Rotate to Angle" behavior, you can make an actor behave like a flipper. Just enter the appropriate values for the "Angle" fields.

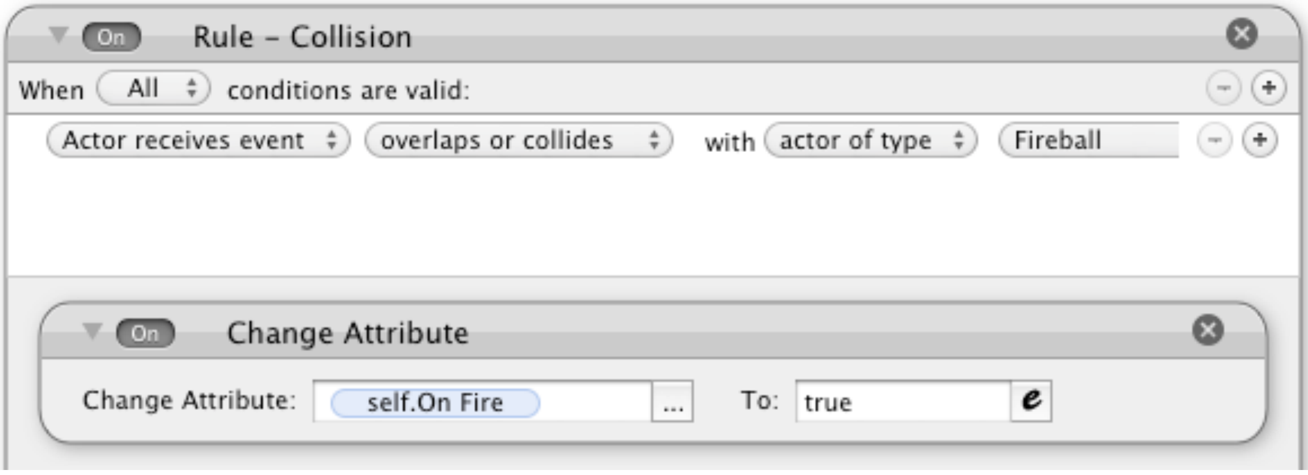
The second part is a bit tricky. Since GameSalad only provides two shapes for collision detection, the creation of a pinball flipper requires a bit of creativity.



The above image is the left flipper from a pinball game. In order to show the areas of transparency, a gray background was added to the image. This is similar to the Tank Turret from Chapter #8. The flipper is mostly on the right side of the image. That's because the actor rotates along the center point. The big dot in the middle represents the center point. If you're going to create a pinball game with GameSalad, your level design has to compensate for the extended collision area. Otherwise, the ball will bounce with the invisible parts of the flippers.

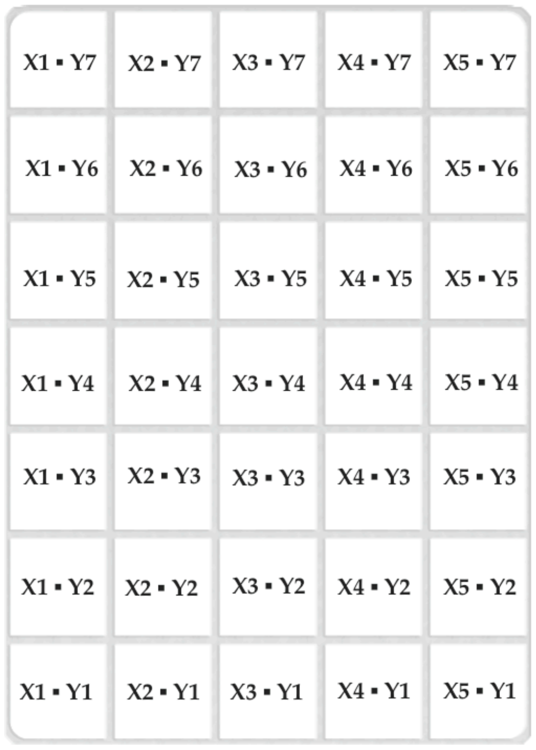
You also might want to double-up on your "Collide" behaviors. During development, I couldn't figure out why the ball would go past the flippers. To resolve this issue, I added copies of the "Collide" behavior to the ball and the flippers — one for each actor. Also, on the flippers, I put the "Collide" behavior right at the top of the list. Their main function is to "Collide" with the ball. I had to make sure that's what happened. I also capped the maximum speed of the ball to something more manageable.

You can also make your environment destructible. That can make the game more fun. Obviously that's the core of the brick smashing game, but there are creative ways to remove actors from the scene. In both of my ball bouncing games, there are actors that can be destroyed. But since the main actor is a fireball, I don't just destroy the other actors. Sometimes I set them on fire first. They burn away and then they're destroyed. Afterwards, a bonus item can be displayed or the number of points added to the score.


On
Rule – Collision
When All conditions are valid:
Actor receives event overlaps or collides with actor of type Fireball
On
Change Attribute
Change Attribute: self.On Fire ... To: true e

The trick is to get the core functionality just right. Then, you can test your game. If you have some processing power left over, you can add special effects to your game.

Chapter #16 - Puzzle Games

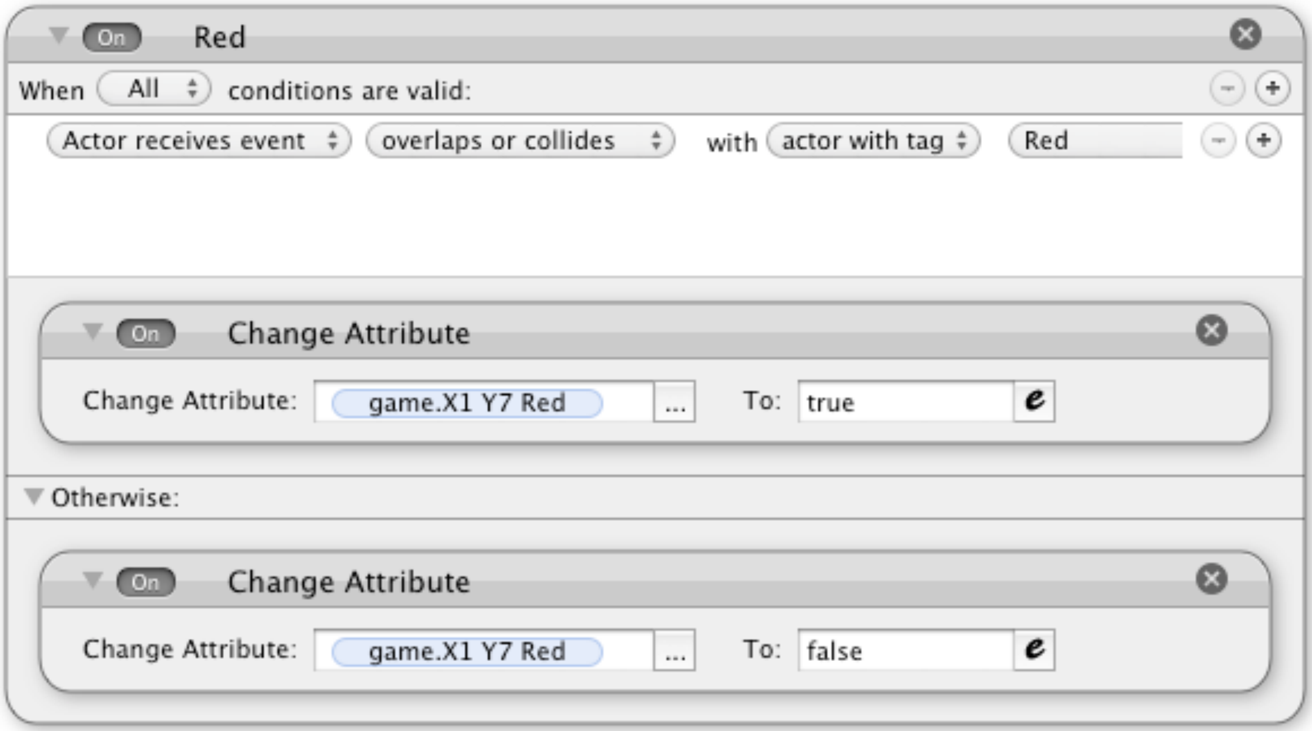

X1 ▪ Y7 X2 ▪ Y7 X3 ▪ Y7 X4 ▪ Y7 X5 ▪ Y7
X1 ▪ Y6 X2 ▪ Y6 X3 ▪ Y6 X4 ▪ Y6 X5 ▪ Y6
X1 ▪ Y5 X2 ▪ Y5 X3 ▪ Y5 X4 ▪ Y5 X5 ▪ Y5
X1 ▪ Y4 X2 ▪ Y4 X3 ▪ Y4 X4 ▪ Y4 X5 ▪ Y4
X1 ▪ Y3 X2 ▪ Y3 X3 ▪ Y3 X4 ▪ Y3 X5 ▪ Y3
X1 ▪ Y2 X2 ▪ Y2 X3 ▪ Y2 X4 ▪ Y2 X5 ▪ Y2
X1 ▪ Y1 X2 ▪ Y1 X3 ▪ Y1 X4 ▪ Y1 X5 ▪ Y1

I remember the process involved for creating my first puzzle game with GameSalad. I sat in my chair, looking out the window or the walls of my office. I wasn't really staring at anything important, but my mind was focusing on a new game design. I wanted to create a puzzle game, but it seemed almost impossible to make it with GameSalad. Actors can communicate with each other through "Attributes" and "Behaviors". Yet, I wasn't sure how to do this on a large scale. If too many "Rules" and other "Behaviors" are added to an actor, the game will slow down and performance will suffer.

So, I just kept thinking. I did this over the course of days, perhaps weeks. When I slept, I would dream about game design. When I awoke, I tested different possibilities to see if it would work. There always seemed to be some sort of limitation. Creating a puzzle game with GameSalad was proving to be a bigger enigma than the game itself. Then, I realize a solution — the grid!

In each square of the grid I placed an invisible sensor actor. That way, I could quickly identify the type of an actor in a certain spot. This process created a tremendous amount of work for me, as I had to create "Attributes" for each of the squares. I also had to track the different types of actors that could enter that square.

In GameSalad development, you might find yourself with a difficult dilemma. Is your idea going to create more work for you, but create a better experience for the player? When faced with that decision, you should always opt to do the extra work. It's easy write the message, but it's a very high standard to adhere to. Maybe you have limited time. Maybe you're getting bored with your project and you just want to finish already. But as experience has shown me, it's probably better to put the extra work into your projects. The reviewers on the App Store seem more likely to point out your mistakes than your successes. That's why it's important to get as close to perfection as possible. If you take too many short cuts, the players will probably notice — and that's when sales start to suffer.



On Red

When All conditions are valid:

Actor receives event overlaps or collides with actor with tag Red

On Change Attribute

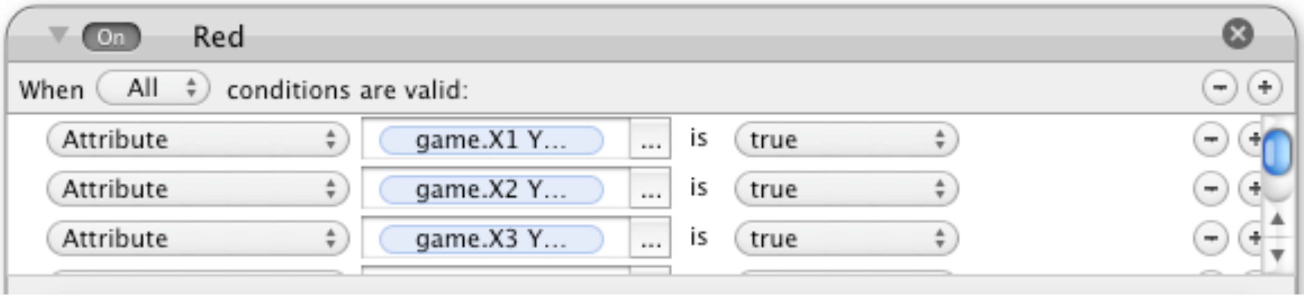
Change Attribute: game.X1 Y7 Red ... To: true

Otherwise:

On Change Attribute

Change Attribute: game.X1 Y7 Red ... To: false

Ultimately, I made a game called Commove. The objective of the game is to make five in a row (horizontally) of the same color. To check for matches, sensor actors are being used. In the image above, I'm changing the value for game.X1 Y7 Red to true when it touches a red actor. The sensor is essentially a combination of the "Rule" and a "Change Attribute" behaviors. This combo records overlaps or collisions. With another actor, I check for colors matches in that row.



On Red

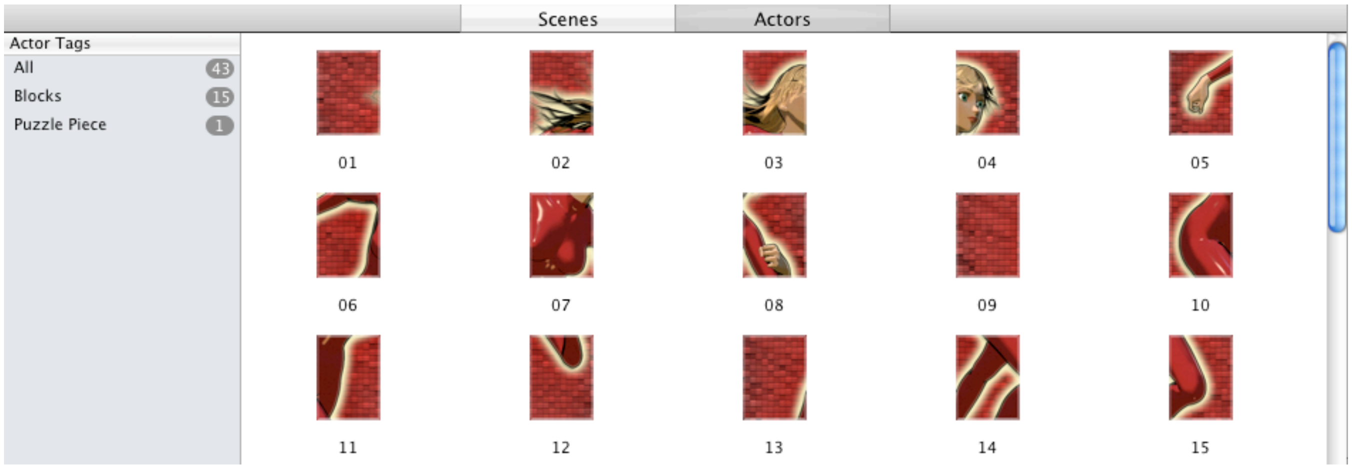
When All conditions are valid:

Attribute game.X1 Y... ... is true

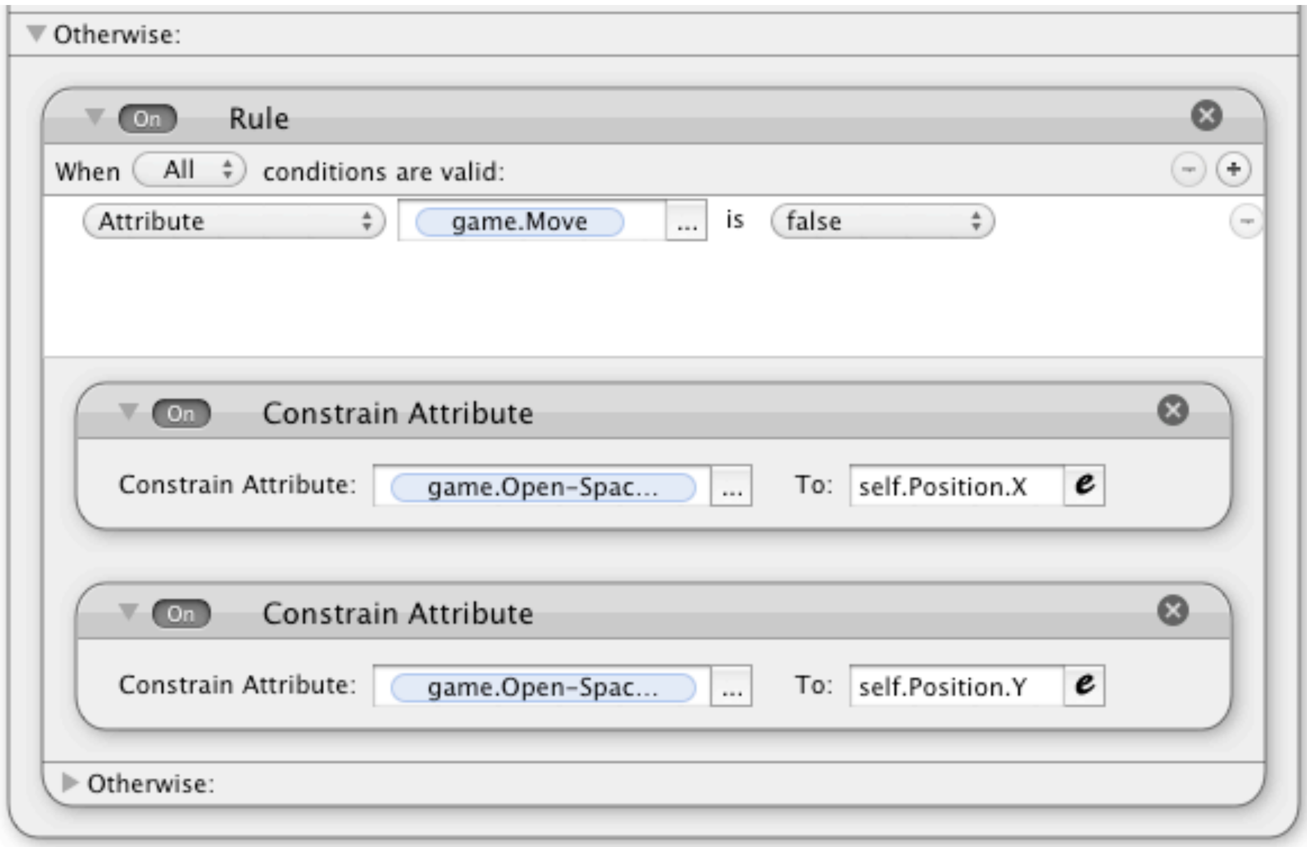
Attribute game.X2 Y... ... is true

Attribute game.X3 Y... ... is true

With a "Rule" behavior, all of the Y7 "Attributes" are being checked by a single actor. A "Rule" is assigned to each color. If "All" of the values for that row are red, then the condition has been satisfied. When that happens another actor is spawned. The new actor destroys all of the actors in that row. The row is complete and bonus points are awarded. This sensor approach can be modified to accommodate different types of games.



A slide puzzle can also be created by using sensor actors, as shown by the squares above. In this example, the sensors serve two purposes. The first is to control movement.



Otherwise:

Rule

When All conditions are valid:

Attribute game.Move is false

Constrain Attribute

Constrain Attribute: game.Open-Spac... To: self.Position.X

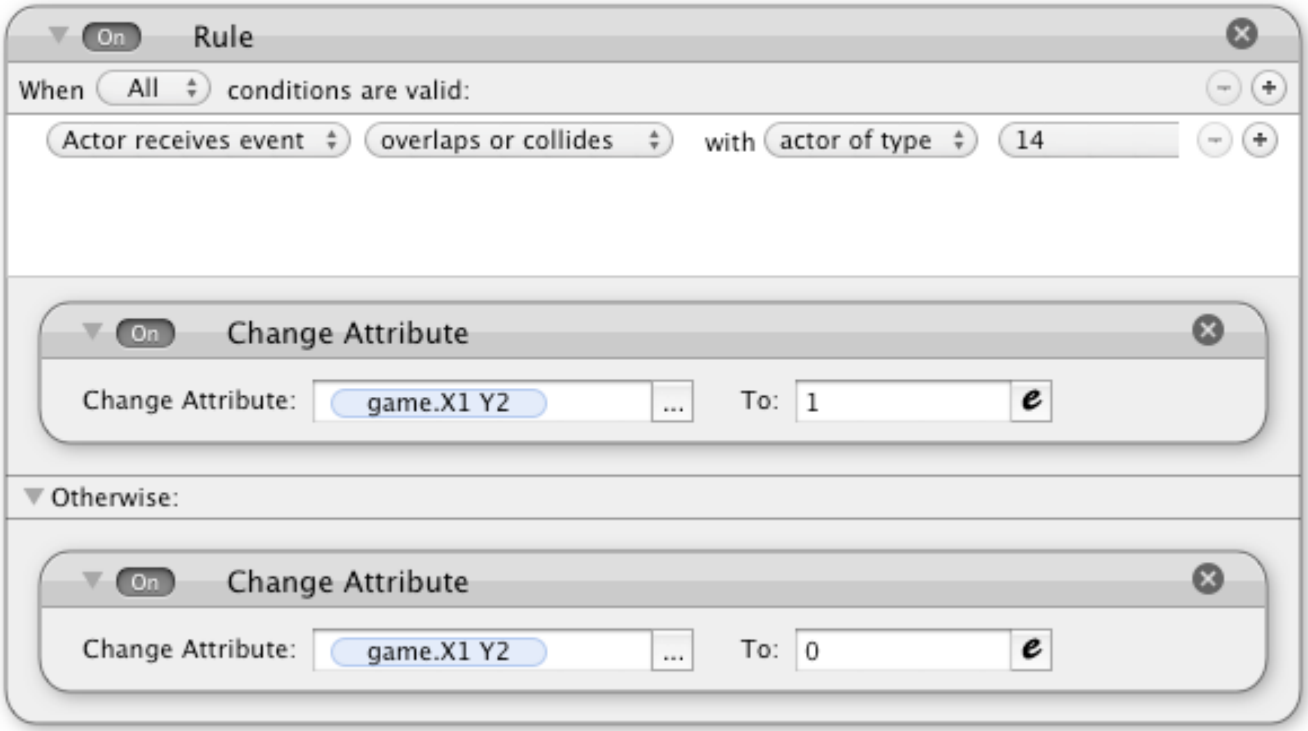
Constrain Attribute

Constrain Attribute: game.Open-Spac... To: self.Position.Y

Otherwise:

When a sensor is NOT touched by a puzzle piece. The sensor declares that spot as open. This is accomplished with the “Otherwise” portion of a “Rule” behavior. The puzzle pieces that are close enough to the open spot are then permitted to move to the open area.

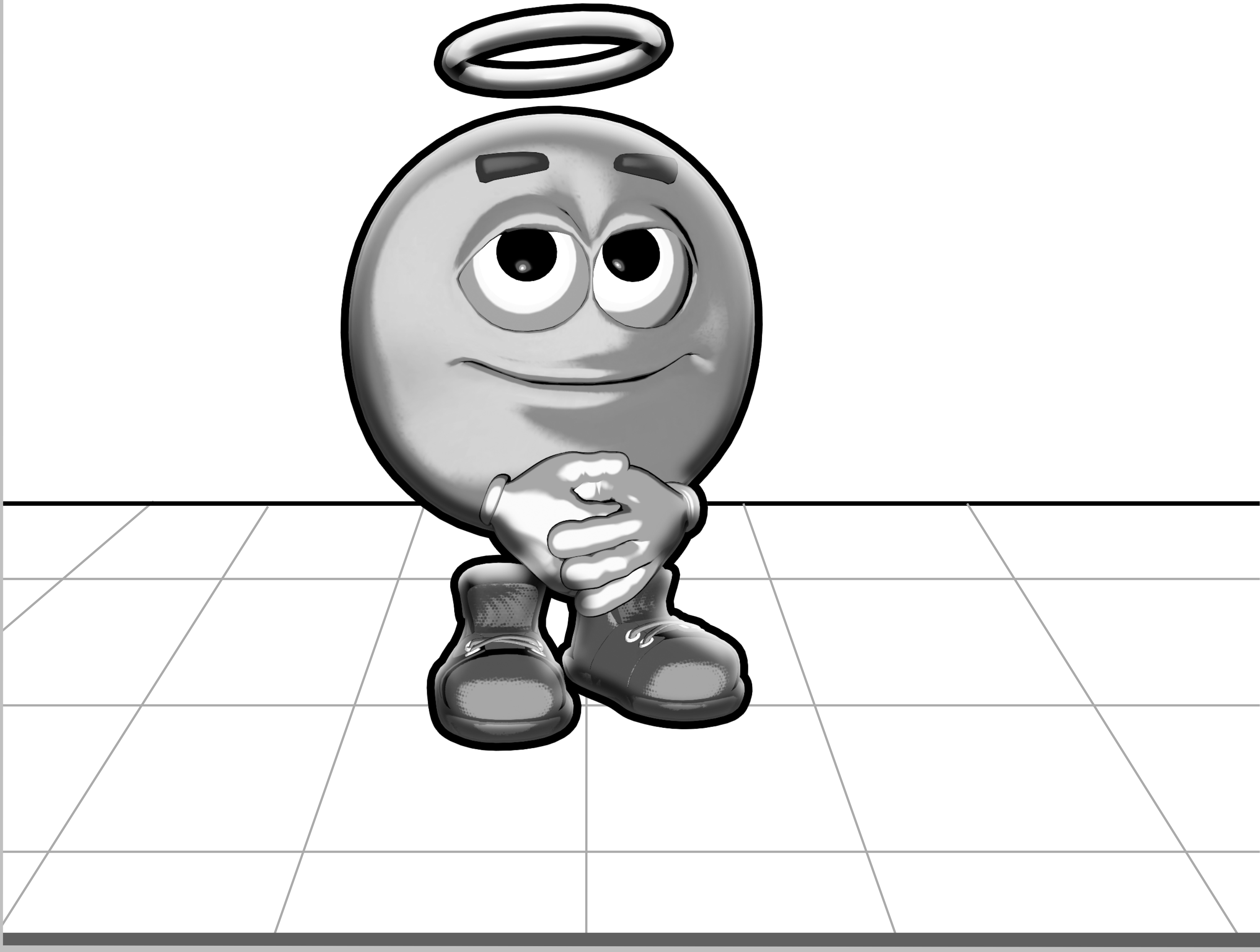
The sensor has another job. It checks to see if the game is completed. A piece is considered to be in its proper position when it touches the corresponding sensor. In this example, the fourth piece of the puzzle belongs in the X4 Y4 space.



Another actor checks to see if all the sensors "Attributes" are listed as "true". If they are, the game is complete. The tedious part is creating all of the actors and the game logic. But once you understand the general idea, sensors can be used to expand your GameSalad design possibilities.

Be careful though. If there too many actors on the scene, your game can slow down dramatically. That's why it's important to set your sensor actors as invisible and unmovable. Performance is especially concerning if you reverse the process. Instead of moving actors onto sensor actors, you could spawn sensors into the game.

6



Section VI - Building A Better Game

17 - Testing

18 - Optimization

19 - Fun

Chapter #17 - Testing

As you reach the end of a project, your energy level could be pretty low. You're tired of looking at the same thing over and over again. As a result, game testing could suffer. Little mistakes can dramatically hurt sales and permanently ruin the reputation of your game. Once those 4 and 5 star ratings become 1, 2 and 3 star ratings, it's very difficult for your app to climb back up. Even if you correct the mistakes with your app, it might already be too late. One of the keys to success on the iTunes App Store is to get it right the first time. The tiniest of errors can cripple your game's success. This chapter takes a closer look at the testing process, with the aim of bringing your GameSalad games closer to perfection.

Don't rush - If you're reading this, you're probably an independent developer. That means you're the boss. You set the deadlines. That means you have to stop yourself from getting too tired or just plain lazy. A good strategy might be to set specific design goals. From the beginning, decide what you want your game to do. Then, document those goals. Make a check list. If your game doesn't meet your own requirements, then why launch your game?

It doesn't have to be anything elaborate, but you should have a general outline for your game. Also, a realistic time-frame is important too. If you're going to spend a month developing your game, then you might want to set aside a few days for testing. It's part of the game development process. Each piece is important.

Don't wait until your game is in review to test your game. That process can easily take a week. If Apple finds something wrong with your game, they're going to bounce your game to the back of the line. You'll have to fix your game and then start the waiting all over again. Even if you catch the mistake before Apple does, you'll have to pull your app down and then rejoin the review line at the beginning.

Don't wait until the end - Testing is not something that should occur when your project is done. It should be a constant part of the development process. One of the worst experiences I had with GameSalad was a mysterious drop in performance. One day, my game was running great. The next day, I lost about 25% of my game's performance. I didn't know why this happened. I was constantly hitting the "Preview" button to ensure that my game worked, but I wasn't keeping track of additions to my project. When I tested my game with an iPod touch, I was shocked and annoyed to see such a dramatic drop in frame rates. I didn't know why that happened, so I had to randomly cut back on every aspect of my game. I didn't know if it was related to sound, graphics, game logic or too many actors. If I had done a better job of testing my game as I created it, I would have known exactly what happened. (If you're wondering what the problem was, I learned that large images are rough on GameSalad.)

Be consistent - In the middle of developing your game, you might not want to make any major changes. Otherwise, you could throw off your testing numbers. You might want to hold off on upgrades for your testing device or the GameSalad software, while your game development is in progress. If you feel that an upgrade of GameSalad is important to your game's progress, make a copy of your project first. Then, while working from the copy, test every aspect of your game. Create a new set of benchmarks. Otherwise, it could be more difficult for you to backtrack issues.

Test orientations - When testing with an iOS device, you might want to check all of the orientations for every scene. It's an easy mistake to make. If you checked the wrong "Orientation" box in the "Scene Editor" your game could have a serious problem. You could see that your game is working in "Portrait" mode, but then you turn the iOS device sideways. The game could rotate to "Landscape" mode and not return to "Portrait" mode. Your gaming area will be cut off.

Assume nothing - Something that worked yesterday might not work today. Just because you tested an area of your game before, something seeming unrelated might have caused an issue. Obviously you don't want to waste time, but there should still be a final check. Maybe you missed something that you thought you fixed. Maybe that copy of an actor is slightly different from the original. You do lose time by checking your work. That is a precious resource. Yet, you could lose a lot more time by not checking. Plus, you're making a game. If you're not having fun playing your game, then maybe you should question why you're making it.

That perhaps is the ultimate test. If you can play your game for the thousandth time, but still enjoy it, you might be making a winner.

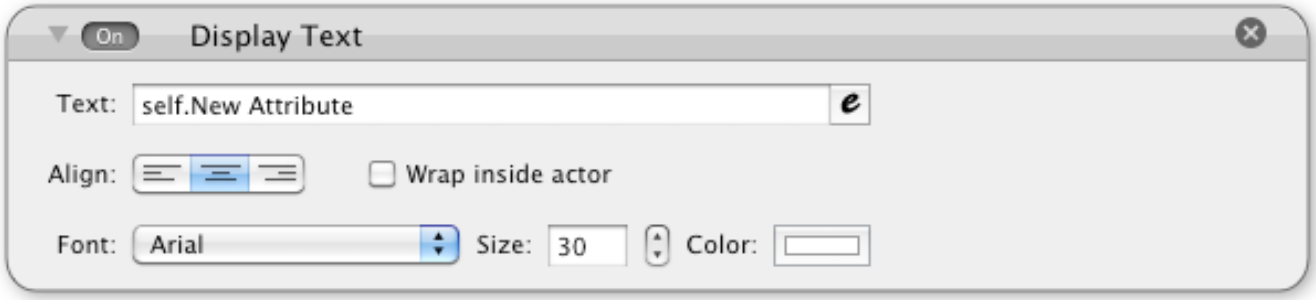
Let it idle - When I get a brand new computer, I like to leave it on overnight. I give it the burn-in test. If the computer can make it through the night without overheating or any other calamities, the computer is probably OK. Games are similar. Don't just reset your game. Look for memory leaks. Change "Scenes" and see if your game can still play. If you test your game in short playing sessions, you're testing under optimal conditions. Imagine some player trying to get a really high score. If your game crashes, that player is probably going to be very angry. If the player is vengeful, you'll likely end up with a bad rating on the iTunes App Store. As you near the competition of your project, you might want to give your game a really long test. You might want to see what happens if you play your game for a really long time.

It's not just about bugs - An important thing to remember is that bugs aren't the only issue with GameSalad games. You're also testing for enjoyment. Is the game fun? If you're finding yourself hating to test the game, is it because you're bored or is it because the game is boring? While testing, you're looking for anything that hinders the player's experience. That could be enemies that are too easy or levels that are too long.

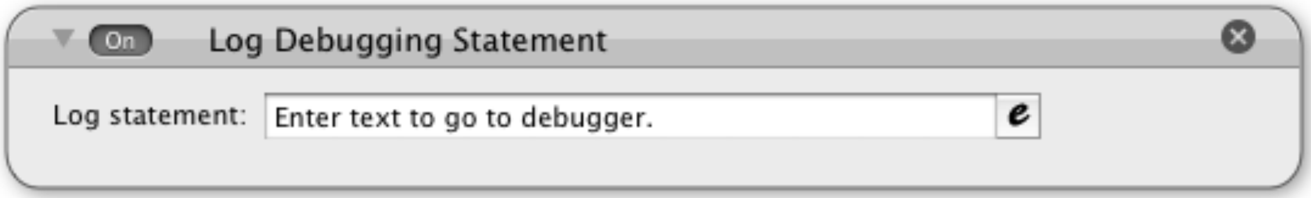
Share - There's a stigma surrounding video games. All to often gaming is viewed as anti-social behavior. Game development can make you seem like even more of a recluse. If that prejudice is accurate, you might not have any friends around to test this game. But even if you're a hermit, and you haven't seen another living person for the last 10 years, you might want to consider hiring game testers. Sharing your game with others is an absolutely critical part of testing. When you can see the expressions on the faces of your testers, you gain tremendous insight towards the success of your game. Are they having fun? Are they frustrated? What areas are they having the most trouble with? If you're the only one who plays your game before it launches, that's a bit of a risk.

Your friends might have no interest in testing your game. But if that's the case, are they truly your friends? Even non-gamers should be curious about what you've been doing for the last month. I have trouble believing people when they say, "I don't play video games." It has been my experience that if you press them on the issue, you can often expose a game playing past. Obviously you don't want any unwilling testers. But if your game can excite the skeptical player, then your game is probably on the right track.

The poor man's debug - While GameSalad gives you some live edit features, it can be tough to see what's going on with "Attributes". That's why I suggest the "poor man's debug" method. Simply use the "Display Text" behavior to show the values that you're interest in.


Display Text
Text: self.New Attribute
Align: centre
Wrap inside actor: off
Font: Arial Size: 30 Color:

You can create an on-screen display that can be toggled by a hidden key sequence. Sometimes I put hidden tricks into my games too, which can help me with testing. I'm not sure if anyone has ever found my hidden developer cheats. That might be OK if players do uncover the game cheats. Some of the most fond memories I have from old NES games are of cheats and tricks. It might not be a bad idea to bury a hidden cheat in your game.



Log statement: Enter text to go to debugger.

GameSalad also has a debugger. By using the "Log Debugging Statement" behavior, you can send attribute data to the log. Pressing Command+D in the GameSalad Creator should bring up the log window. However, this behavior is not persistent. That's why this behavior is more effective within a "Rule". Much like the developer cheat idea, you could add a condition that grabs the debugging behavior.

To emphasize how important testing can be to your project, here is a partial checklist for use with your game testing.

GRAPHICS

- All images look sharp
- All graphic files are optimized to reduce their size
- All image sizes are powers of 2
- All actors graphics closely match their collision shape
- Proper blending modes are used for all actors

AUDIO

- All audio files sound clear and have strong volume
- All audio files are optimized to reduce their size
- Destroyed actors are not causing sound loop issues

BEHAVIORS

- All expressions have proper use of parentheses
- Behaviors are ordered in actors by their priority
- No actor has an excessive amount of behaviors

PERFORMANCE

- Frame rate is consistently above 30 FPS on the supported devices
- iOS memory usage isn't too high
- All invisible actors have their "Visible" option unchecked
- All unmovable actors have their "Movable" option unchecked

SCENES

- Each scene can be loaded properly
- Orientation modes are properly set to match the scene
- Wrapping option is properly set for each scene
- The scene size, for every scene, is consistent with the with the project parameters
- Unused actors are eliminated from the scene

PROJECT MANAGEMENT

- Unused actors, scenes and assets are removed from the project
- The correct platform is targeted

Chapter #18 - Optimization

If you've been carefully reading this book, you might have noticed optimization techniques scattered throughout the chapters. But since performance is a huge issue with GameSalad development, I believe it is necessary to highlight this matter. This chapter contains information on optimizing your game. They're neatly organized in one location for quick reference.

Power Hogs - There are certain "Behaviors" that are more taxing on the hardware than others. The following is a list of "Behaviors" that you should use sparingly.

Animate - This an issue because it usually involves a lot of images. In general, it's important to optimize your graphics. That's even more important with animation.

Constrain Attribute - This one is another big offender. If you have one or two constraints, it's usually not a problem. But with too many, your game can dramatically slow down. This is a "Persistent Behavior". The game is constantly checking the value of the "Attribute". Alternatives could be "Change Attribute" or the "min" and "max" functions.

Particles - This can quickly cripple the performance of the game. If you add too many "Particles", or if large images are in use, slowdown is highly likely. To compensate for this, don't use that many "Particles" or create an option screen where the user to control the amount of "Particles" on the screen. Image optimization is huge for this "Behavior". It's also possible to have "Particles" without images. For an example of this, see the "Space Background" tutorial in Chapter #24.

Rule - By itself, this is not so much of an issue. But if you load up an actor with too many rules, the performance will suffer. That's why I like my tank turret approach. I can assign shooting tasks to the turret and movement tasks to the tank. Even if this doesn't result in a dramatic performance increase, it does make editing the game a lot easier. I find that the GameSalad editor hangs while opening "Behavior" heavy actors.

Spawn Actor - The more actors you spawn at once, the greater risk of poor performance. To compensate for this issue, you can recycle actors. Instead of destroying an actor, you can simply move it off-screen and then reuse it later.

Timer - Just like a "Rule", this is a behavior container. Too many timers, can hurt performance.

Actors - In general, you want to keep your game lean. If you have too many actors, or too much game logic, the performance can drop. Yet, some game designs simply require additional actors. You can resolve that issue by optimizing each actor.

Visible - If an actor doesn't need to be seen, disable the "Visible" option in the actor's "Graphics" settings. Setting an actor's "Alpha" channel to zero is not the same thing. This is useful for sensor actors.

Movable - If an actor doesn't need to move, disable the "Movable" option in the "Physics" settings. When actors touch, only one actor needs to have this option enabled for the "Physics" related settings to work.

Graphics - Do you really need an image? There is much that can be done with a colored actor box. If you do need an image, is it the right size? Oversized images can hurt performance. Additionally, you can reduce the size of images by reducing the color palette. An Indexed PNG file can be limited to 256 colors. While iOS devices will still use the same amount of RAM, the file size may be dramatically reduced. This speeds up loading times and reduces the size of the app. With less colors, the quality of the graphics can suffer. But if you're trying to keep your app under 20 megabytes, reducing color palettes might be something to consider. Players might skip your app if the file size is too large.

Powers of 2 - It's usually a great idea to keep your image dimensions to the following pixel sizes: 2, 4, 8, 16, 32, 64, 128, 256, 512 and 1024. If you read the GameSalad forums, that suggestion is thrown around quite frequently. Not convinced that it was such a major issue, I decided to test it out for myself. I created three nearly identical images. One was 512x512, another was 513x512 and the largest was 513x513. I then checked the memory usage.

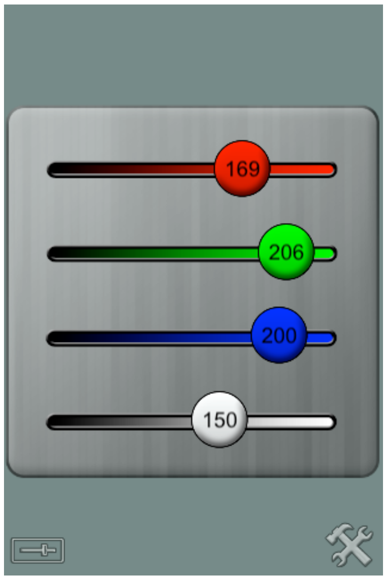
Image Size	Memory Use
512x512	512 KB
513x512	1.0 MB
513x513	2.0 MB

By adding one pixel row to the width of the image, and one pixel row to the height of the image, the memory required to run this image on my iPod Touch was increased — FOUR TIMES! With only one more pixel on the width of the image, the memory requirements doubled. Stay inside the containers. Use "Powers of 2" for your graphic sizes. If you're not optimizing your images to conform with this standard, you're simply throwing away memory. If your memory requirements are too high, the game will crash.

You can mix and match the "Powers of 2". A 64x128 image is optimized, as both of those numbers are "Powers of 2". Smaller images are better for performance. Your images cannot be more than 1024x1024. But realistically, images that large can cripple performance. Even with smaller sized images, this can still be an issue. Every little bit of wasted memory adds up. This is especially true with the "Animate" and "Particle" behaviors. Both of those features use a lot of images. With each image that is not optimized, you're wasting precious space. Ignore this optimization technique at your own peril!

Chapter #19 - Fun

What makes a game fun? The answer to that question is very subjective. What might be interesting to me could be boring for someone else. Yet, as a game developer, one of your design goals should be to make a fun game. This chapter covers some basic ideas to add enjoyment to your GameSalad games



Make it personal - People have their own preferences. Let players add their own style to your game. By adding customizations to your game, players can form a stronger attachment to your game. There are some simple ways to do this and some very difficult ways to do this. An easy example, you could add a color tint to an actor or your user interface. The user can adjust the color of the light by sliding one of the four buttons. Each button represents an RGBA channel. This could be applied to an RPG character creator system. A female player could make their character pink. Maybe she hates stereotypes and she could make her character blue. By giving players options, you're catering to their needs to feel special. That's a big part of gaming. Another way to customize the experience is by letting the player name their character. The "Keyboard Input" behavior makes it easy to do that. You could also create your own on-screen keyboard if you wanted to give your game a fancy look.

Intuitive Controls - If your game has bad controls, it can make the entire game bad. Getting the character movement perfect is critical to your game's success. If the music or sound effects are bad, a player can mute your game and turn on the radio. As for graphics, that didn't stop the Wii™ from going up against more powerful consoles. It's the controls. The player needs to feel as if the game character is an extension of their own existence. I remember playing Super Mario 64 for the first time. I didn't just rush into the adventure. I stayed outside the castle for a long time, jumping around like a happy idiot. Up until that point, I wasn't very excited about most 3D console games. But unfortunately, that excitement led to disappointment. I expected a similar experience from the Legend of Zelda™: Ocarina of Time™, but it just wasn't as fun. While many people may think favorably of that game, I hated that I couldn't jump freely. I also hated that stupid owl. I didn't need every little detail explained to me. I couldn't even skip the annoying dialogue. Little things like that can frustrate the players.

Part of your battle is managing player expectations. You might find yourself getting bad ratings for features you simply cannot add to your GameSalad game. It's a little irrational, but that's what you're up against — high expectations. You'll need to hype up your game to motivate players to try it. But if you build up too much hype, the players could disappointed by a product that didn't meet expectations. It's another one of those delicate balancing acts you'll have to perform as an independent game developer.

Audio - Even though players can play with the sound off, is that really what you want the players to do? No, it's probably better to have something like Mega Man 2. Decades later, it still hits me on occasion. Maybe I'm in my car, at my office or even in the shower, but out of nowhere it strikes — I find myself singing some of those old songs. Will players remember your songs? If not, maybe you need better music. The same goes for sound effects. The Pac-Man death sound, the Super Mario power up, those are memorable and enjoyable sound effects. If you're having trouble finding sound effects or background music, the Internet a great place to look. Just search for sites that license the assets you need. It can cost some money to acquire professional grade music, but it can dramatically improve the quality of your game.

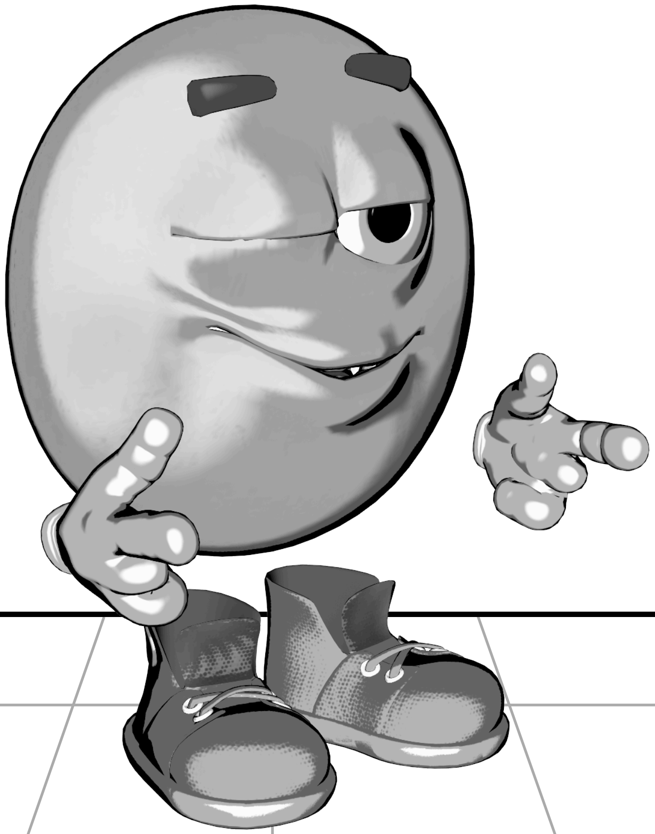
Graphics - The same goes for graphics. Many of the iTunes App Store success stories have cutesy graphics. While that seems to be helpful, an adorable look is not necessarily needed for fun. The point is to give your game a distinguished look. Make your game professional and memorable. Again, you might have to spend some money to make that happen. But if you believe in your game project, it might be worth it.

Challenge - Make your game somewhat difficult, but not impossible. Also, reward your player as they accomplish difficult tasks. Maybe reveal a bit more of the story, add an animation sequence or something cool like that. Adding different difficulty levels could be a good idea too. I see a lot of babble online about "hardcore" gaming. Some players simply hate easy games. They want more value for their dollar. Yet, there are casual players that can frustrate easily. Difficulty settings can help you manage this issue. Adding multiple endings is a nice touch too. That's more work, but it can make your game more fun to play.

Bonuses - If you're making a shooting game, add lots of different guns. If you're making a puzzle, add different play modes. Add some Easter-eggs to your game. Give something for players to talk about. With Super Mario Bros., I remember the minus world and the infinite lives trick. Perhaps an extreme challenge in your game could start some buzz on the Internet. Maybe players from all over the world will try to figure out the secret.

What do you like about video games? What do others like about video games? Think about that and see how that knowledge can be applied to your game.

7

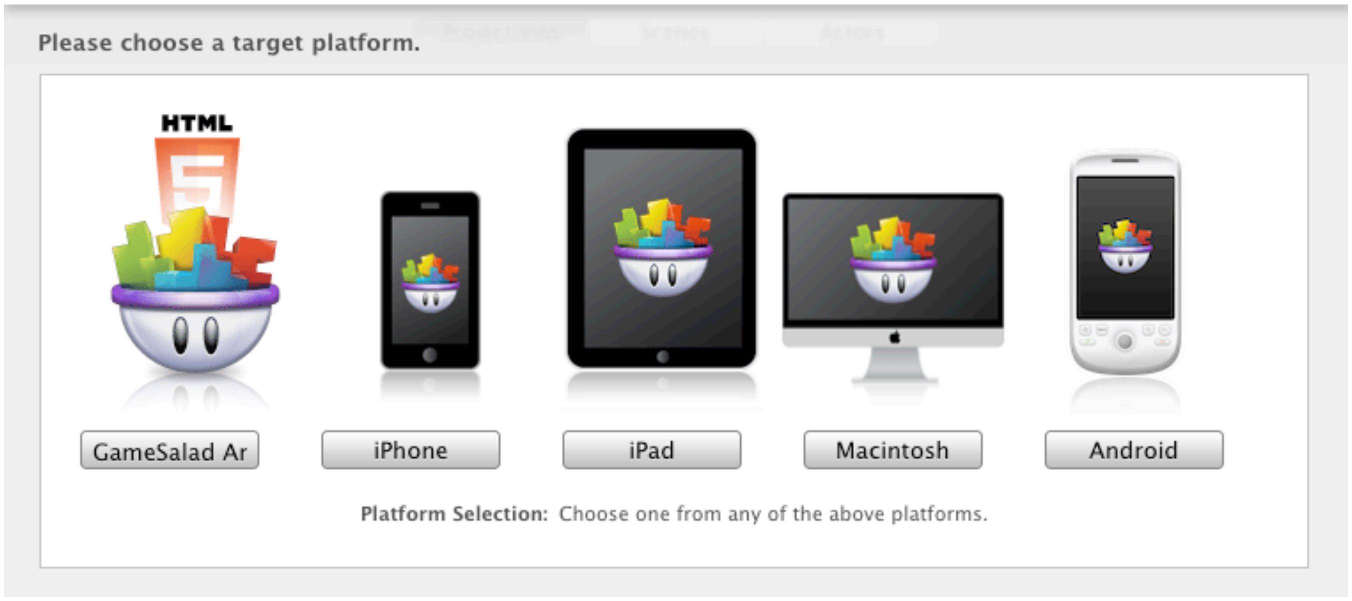


Section VII - Publishing Your Game

20 - App Publishing
21 - Mac Publishing
22 - Web Publishing

Chapter #20 - App Publishing

Those last few chapters were fairly easy, but this is where it gets tough. Getting your game on iTunes is typically a slow and frustrating process. It can suck out much of the joy in making GameSalad games. Yet, that's where the money and fame is. That's where this chapter is heading. The following is a tutorial on publishing your game on to the iTunes App Store.


Please choose a target platform.
HTML
GameSalad Ar
iPhone
iPad
Macintosh
Android
Platform Selection: Choose one from any of the above platforms.

When you click the "Publish" button a window should appear. Depending on your project's orientation, it lets you choose between five options — GameSalad.com, iPhone, iPad, Mac and Android. For the beginning of this chapter, the focus is on the iPhone and the iPad. You'll probably want to check that your project matches your platform choice. Otherwise, it could result in Apple rejecting your binary.

In general, you may see a lot of rejection with iTunes App Store publishing. Apple is extremely picky. One of my apps got rejected for having the words "for iPad" in the title. Yet, after realizing that so many apps also had the words "for iPad" in the title, I tried again. I used parentheses and added (for iPad) to my title. The app was approved. Apple is quite fickle too.

After you make a selection, a new window will appear. Select "Create New Game" if this is the first time uploading the project. Otherwise, select the game you're changing and click "Update". Your project is saved on the GameSalad.com website. If you're having problems with uploading your game, it might be a login issue. I suggest quitting GameSalad and then logging in before publishing your game. With a refreshed connection I tend to get less problems.

The next window will give you the option to enter additional information and settings for your app.

Overview - This is a general description about your game. It's more for GameSalad.com use than App publishing. Anything entered on this screen will have to be reentered into iTunes Connect. However, much of it should be pre-populated. When you first created your game, you could enter a title, description and tags. That information is reused here. From the Overview screen, you can also add information about the categories and icon for the app.

Platform - From this section you will need a "Provisioning Profile". You can click the "Get Provisioning Profile" button for additional information.

Creating my first "Code Sign" was confusing and frustrating. Maybe these instructions will make it easier for you. Also, this part should probably be competed before you start publishing your app. To create your code, follow the following instructions.

- Log into the "iPhone Dev Center" website.
- Click the "iPhone Provisioning Portal" link.
- From the vertical menu on the left, select "App IDs".
- Click the "New App ID" button.
- Enter something descriptive in the "Description" field, like the game's name.
- If this is your first app, use the "Generate New" option for the "Bundle Seed ID" setting.
- Enter a "Bundle Identifier", such as the reverse order of your domain name. I usually use "com.photics._____" for my games. The blank is filled in with a word for the game.
- Press the "Submit" button to send the data to Apple.
- From the vertical menu on the left, select "Provisioning" link and then click the "Distribution" tab.
- Click the "New Profile" button.
- Select "App Store" for the "Distribution Method".
- Enter a "Profile Name", which is usually similar to the game's name.

- Select the "App ID" that you just created and then press the "Submit" button.
- When on the "Distribution" tab again, your "Provisioning Profile" might be listed as "Pending". Just a refresh of the page usually changes the "Status" to "Active". When it's ready, you can "Download" the file.
- Drag the file onto the Xcode application. Wait until the file is processed. Now you should have a "Provisioning Profile" ready for GameSalad.

That wasn't so terrible, right?! Aren't you glad that you bought this book? Apple's official "Developer Guide" might also assist in navigating through Apple's walled garden. The guide can be accessed from the main iTunes Connect page. This is some of the least enjoyable aspects about creating games with GameSalad. But after your fourth, fifth and sixth game, the process should get easier and easier. You'll need a new "Code Sign" for each game that you create. Otherwise, if your code is not unique, your game won't upload. It's a tedious process, but necessary for publishing your apps on the App Store. If you did this step correctly, you should see a listing for your app in the GameSalad "Provisioning Profile" drop-down menu.

The "Display Name" is what will appear underneath the icon when it is on an iOS device. There's not a lot of room for too many characters. To keep your game's title looking professional, pick something short or possibly abbreviate your name. Otherwise, the iOS device will automatically truncate your name is too long. It's not a pretty site, all jumbled up letters, the horror!

The Version Number is important, because Apple checks this information. If you're creating an update to your game, it's important to specify the proper "Version Number". Otherwise, the default of "1.0" should be fine.

With the "Advanced Options" from the "Platform" tab, you can check the supported orientations. Check the modes that are supported in your game. This can be tricky if only certain scenes support multiple orientations. Pick the combination that most closely matches your game. I usually check both of the modes for the orientation that my game supports, such as "Landscape Left" and "Landscape Right".

You can also decide if ARM7 processor should be required. If your game tends to slow down on older devices, this option blocks devices that don't have the newer processor. If your game lags, you will likely get more negative reviews. With the ARM7 option, you block iOS devices without the newer processor. Of course, if you do that, you're eliminating potential customers.

Before you restrict your game to ARM7, you might want to check if your game can be optimized.

Apple wants everything to look shiny, with rounded edges. So, you can check the “Enable Glossy App Icon” option. That will give your square icon Apple’s standard look. If you prefer to create the effect manually, then you can disable this option.

Video - From this tab you can specify a link for a YouTube video. Since this is not related to iPhone publishing, I just skip this step. Although, it is a good idea to make a video for your game. It could encourage customers and app reviewers to try your game.

Screenshots - From this tab you can specify a screenshot for your game. You can have up to five screenshots and the order can be rearranged. With at least one screenshot listed, you can go to the next step.

Review - When you’re ready, you can “Publish” your project. Your project is sent to the GameSalad for publishing, and then a file is sent back to you. Zip the file by right-clicking the app and then selecting “Compress” from the menu.

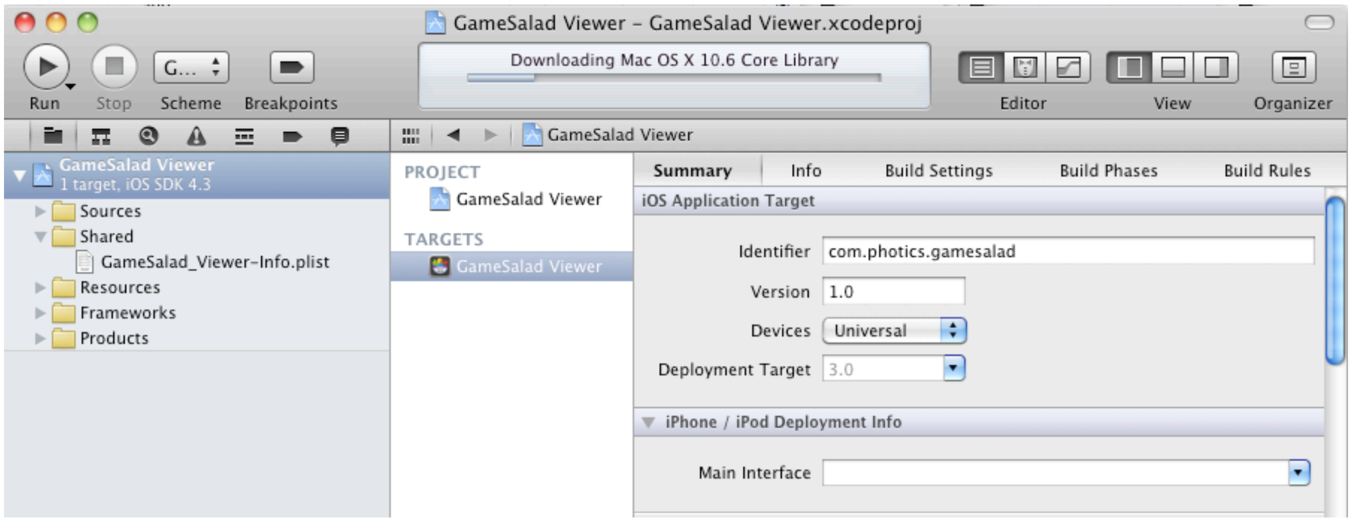
But before you actually publish your iOS game, you might want to test it on an actual device. This process is similar to the “Code Sign” process. Go to the "iPhone Provisioning Portal" and click the “Devices” link. Once on that page, you can click the “Add Devices” button. It will ask for a “Device Name” and “Device ID (40 hex characters)”. For the name, that’s something of your choosing. For the ID, you might have to hit iTunes.

Connect your iOS device to your Mac. Launch iTunes and locate your hardware. Click on the name of your device to access the “Summary” tab. Now here’s where it gets sneaky. You have to click the “Serial Number” to access the “Identifier (UDID)” information. This is not a visible button. The text itself has to be clicked. With that information, you can register your device.

Next, access the “App IDs” page. A new one will be needed for GameSalad. Many developers use their domain name, like “com.photics.gamesalad”. Whatever you pick for your name, that information will be needed at a later step.

Return to the “Provisioning” page, but stay on the “Development” tab. Create a “New Profile” for your GameSalad testing. If things are progressing smoothly, you should see your iOS device listed on the page. Enter a “Profile Name”. I put “GameSalad”. I also checked my name in the “Certificates” area. I selected my GameSalad “App ID” and I checked the box for my iPod Touch.

Are you tired yet? There's still more to do. Now you should be able to grab this file. Drop it onto the Xcode application like before. But this time, there's some work to be done in Xcode. Don't worry. It shouldn't be too scary. It's not any real programming, but it is a bit more technical than what's normally involved with GameSalad. You'll need the latest "GameSalad Viewer". It's available at the official GameSalad website.



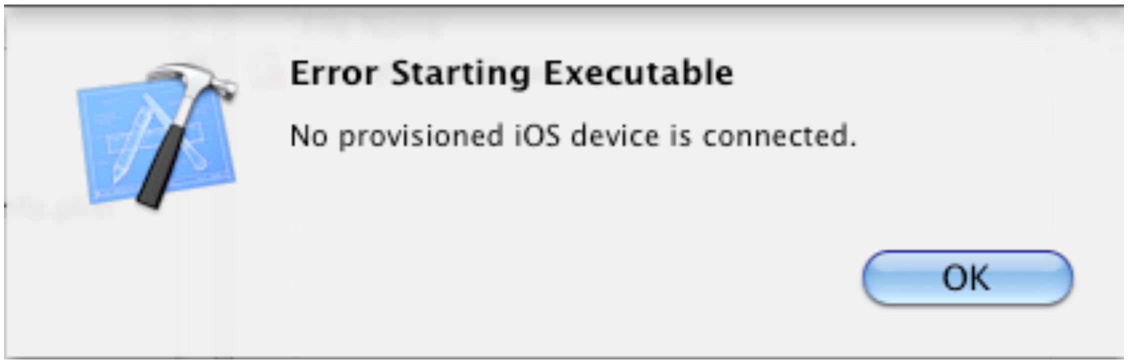
iOS Application Target

Identifier com.photics.gamesalad
Version 1.0
Devices Universal
Deployment Target 3.0

iPhone / iPod Deployment Info

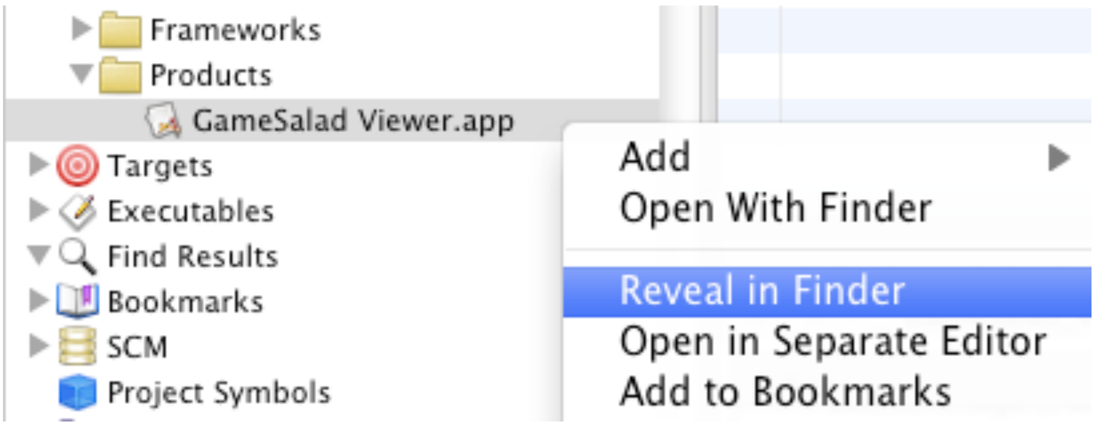
Main Interface

Open up the enclosed Xcode project and then make one change. The "Identifier" field should match the setting you entered on the "App ID" page in the "iPhone Provisioning Portal". With those settings changed, simply click the "Build and Run" button. That should launch the "GameSalad Viewer". It should work that way, but you might get the following error...


Error Starting Executable

No provisioned iOS device is connected.

If that happens to you, there is a fix. Simply find the app on your harddrive and drag it onto iTunes. Then, just synch your device. When I do that, I can run the viewer normally.



For a quick way to find the app file, just open the "Products" folder and right-click the app. By selecting "Reveal in Finder", the app file should appear. This is only a sample of Xcode action. I've made some Xcode apps and it was tough. Yet, you can use this software to make your own apps. Xcode is much harder to learn than GameSalad, but you'll have more control over your app. I don't write this book to discourage use of Xcode. I think it's awesome software. This app was made with Xcode. But unfortunately, it's difficult software for me to use... really, REALLY difficult. I think of GameSalad as a stepping stone. It introduces you to what's possible. From there, you can decide if you want to learn more.

Once the software is on your testing device, you can start testing your GameSalad projects directly on your iOS hardware. Just launch the GameSalad software, make sure your iOS device is on the same network as the Mac running GameSalad, launch the viewer software on your iOS device and then click the preview icon that appears. Your GameSalad project should load and then run on your iOS device. It's quite a task, but it should be quite satisfying once it works.

GameSalad Viewer allows you to preview and test your GameSalad game on an iOS device.

To get GameSalad Creator for your Mac, visit http://www.gamesalad.com

Once your GameSalad project has been tested and exported to a compressed binary, you're ready to log into iTunes Connect and create a listing for your app. You'll need at least one screenshot, at the appropriate resolution, and one 512x512 game icon. That larger icon should match the icon embedded in your project. Get your assets in order to get your game past Apple's wall.

The rest is digging through the fields. Make a good description for your app, think of good keywords, set up your hyperlinks, list a contact email and fill in the other requirements. Be very certain about your information, as there are many thing you cannot change after your app is published. If you need to make a restricted change, you can try releasing a new version of your app. When everything is set, use Apple's Application Loader to upload your game. For more information on making your apps more successful, see Chapter #26 - Promoting Your Games. It contains marketing information that's relevant to App Store publishing.

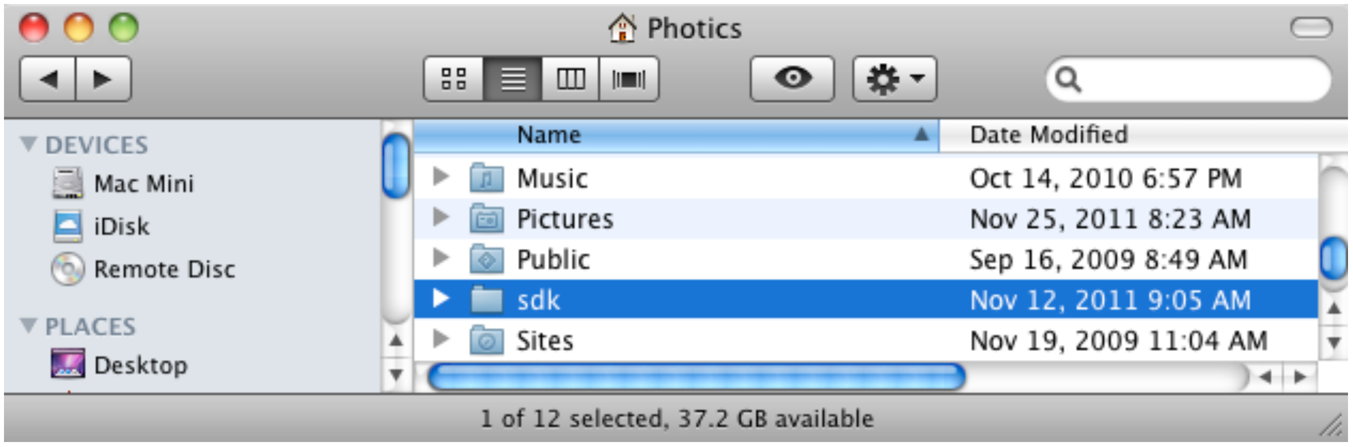
Ad Hoc - Before you actually send your game to Apple, you might want to do some serious testing first. That means the creation of an Ad Hoc version of your app. The process is also similar to creating an actual app. The main difference is selecting "Development" instead of "Distribution". You'll need the UDID code for your iOS device and it will have to be registered in the iOS Provisioning Portal. You can get the UDID code for you iOS device in iTunes. Just click your device, go to the summary page and then click the words "Serial Number".

iTunes Connect Mobile - And while you're on Apple's developer website, you might want to pick up the iTunes Connect Mobile App. It is primarily for viewing financial reports on your iOS device, but it does something even better — notifications. When your app changes status, you can receive an alert. This is great for getting ready for the launch of your app. Once you know your app is approved, you can get ready for a marketing blitz. I find that this is more productive than constantly refreshing the iTunes Connect page. Apple usually sends an email when an app is "Ready For Sale" too.

Apple is not the only game in town. Once your app is on the iTunes App Store, you might want to publish it on the Android Market. The process is similar, perhaps easier. Just like registering as an Apple Developer, Google expects the same from their Android developers. Yet, the process is far less grueling and it's less expensive too. I originally started as an Android developer back in 2008. It wasn't $99 a year. $25 was good enough for Google. Not per year… it was just $25.

There was no review process either. I could upload a game whenever it was ready — even beta versions. If a player found a technical problem, a quick update could be uploaded.

So, if that sounds like fun to you, developer.android.com is a site you might want to visit. The first step is to download the Android SDK — for Mac OS X. By simply renaming the downloaded folder "sdk" and moving it to my user folder, I was almost ready to publish. (On my Mac, it's "Photics". The name is specific to the Mac user — but the folder should have an icon of a house on it.)

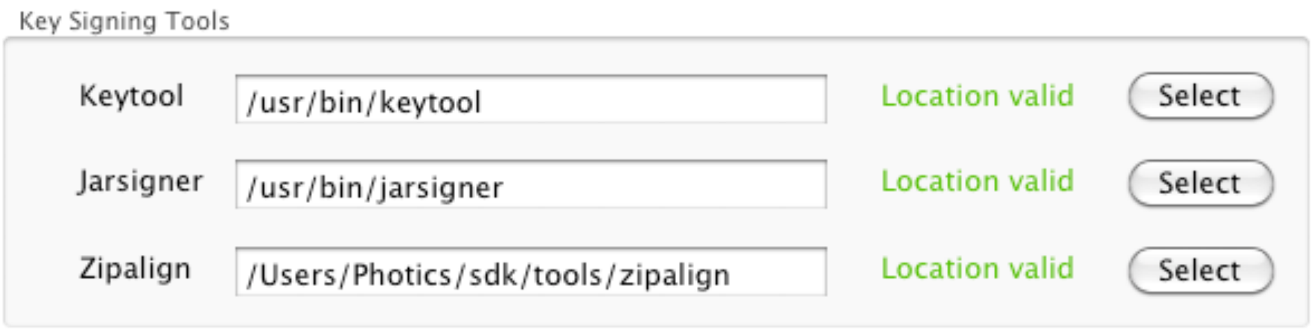


Name	Date Modified
Music	Oct 14, 2010 6:57 PM
Pictures	Nov 25, 2011 8:23 AM
Public	Sep 16, 2009 8:49 AM
sdk	Nov 12, 2011 9:05 AM
Sites	Nov 19, 2009 11:04 AM

1 of 12 selected, 37.2 GB available

If you want to check for updates and additions to the Android SDK, you can run the "Android" Unix Executable File in the "Tools" folder of the SDK. That will launch the "Android SDK Manager". This is more for future use, as I didn't update the SDK. A clean installation was enough.

With the SDK installed, GameSalad has almost all of the necessary components for Android publishing. You can check this by trying to publish to Android. First, press "Publish" and select "Android" on the "Target Platform". In the "GameSalad Publishing Manager" — on the "Platform" tab — is an option for "Android Settings". By pressing the "Configure" button, location validity can be checked.


Key Signing Tools

Keytool	/usr/bin/keytool	Location valid	Select
Jarsigner	/usr/bin/jarsigner	Location valid	Select
Zipalign	/Users/Photics/sdk/tools/zipalign	Location valid	Select

If all of the locations are valid, then there's only one missing component — a "Keystore". This is similar to creating certificates for Apple. The difference with Android is that you can get lazy. You don't need a "Keystore" for every app. Although, if you're into security, you might want to consider it. The point is that you'll need at least one. If you don't have a keystore, clicking the "Generate Keystore" button on the "Android Configuration" screen should take you to a Google webpage, containing information on creating a "Keystore".

Once you have all the software in place, the rest of the configuration settings can be entered. The ”Android Package Name” is typically in the reverse domain name style — such as com.photics.name. If your game uses accelerometer data, then that corresponding option should be checked. The “Touchscreen” option depends on how many touch points are being used at one time. To make your Android game more compatible, you might want to limit the amount of simultaneous touches. As an example, if your game only uses one touch point, then select “Default” as the option for “Touchscreen”.

The rest of the settings in Android publishing are pretty straightforward and/or similar to iOS publishing. You might think that Android publishing is far easier than iOS. That might be true, except for one important piece — testing. The big problem with Android is that there are so many different Android devices. That’s where the “Android Viewer” (check the GameSalad website for the latest download) can help.

The best method for installing the “Android Viewer” may vary depending upon your Android Device and personal preferences. I use a Samsung Galaxy Player for testing. It’s like the iPod touch of the Android world. Installing the viewer is as easy as copying a file via USB and launching it with the "My Apps" application.

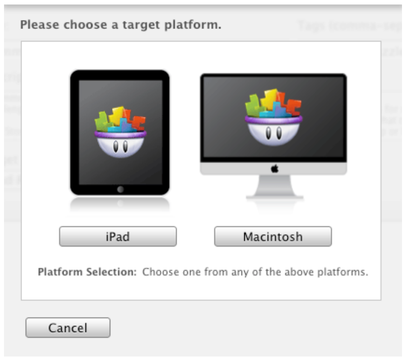
Once your app is tested and ready to go, there are many publishing options for Android Apps. Google’s Android Market isn’t the only game in town. And since apps can be installed directly on Android devices, there’s even the option of selling apps directly to customers.

Chapter #20 Summary

- Using GameSalad is only part of publishing an iOS app. Apple's iPhone Dev Center website is also involved.
- To publish your project, you'll also need properly sized icons and screenshots. This artwork should match your project. Otherwise, your app could be rejected.
- The GameSalad Viewer is used for testing your app on iOS hardware.
- The iTunes Connect website is where you submit your app for review. The app listing should contain accurate and detailed information about your project.
- Creating an Ad Hoc version of your app is a more involved, but more complete, way to test your app.
- Apple has official documentation on iOS publishing. It could be a handy resource to have.
- Once your app is ready for iOS publishing, you might want to publish your app for Android too. The process is quite similar.

Chapter #21 - Mac Publishing

If you don't have a preference as to which orientation your game should support, I suggest choosing landscape. That will make it much easier to port your game to the Mac. If you already have an iPad game in landscape mode, there's hardly any extra work. That's what this chapter is about — getting your GameSalad game on the Mac. One of the strengths of GameSalad is that a single project can be sent to multiple platforms.

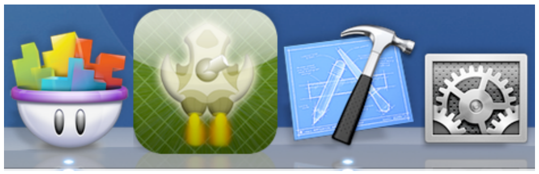

Please choose a target platform.

Platform Selection: Choose one from any of the above platforms.

I like Mac publishing. It's fun. I press the "Publish" button and that's pretty much it. The process is very similar to iPhone publishing. Yet, there are two tough decisions to make. The first question — What size should your game be? Unlike iOS publishing, there isn't a clear standard. Do you set the "Target Platform and Orientation" setting to "Macbook", "720 HD" or "iPad"? If your game is not running in "fullscreen" mode, you could also use one of the iPhone options. I think 1280x800 is starting to emerge as the standard. That's the minimum size for screenshots on the Mac App Store.

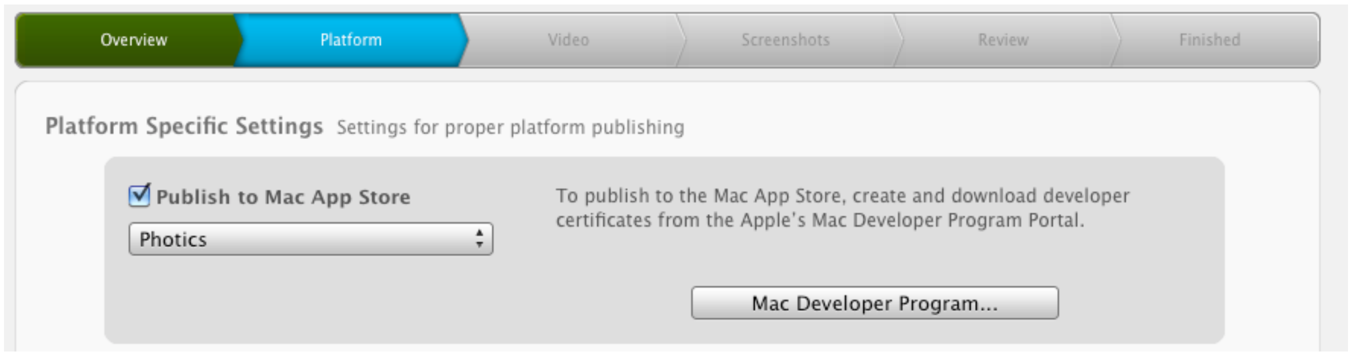
The second question — Should you register for the Mac App Store? Unlike iOS devices, the Mac Desktop is not restricted. You can install third-party applications without approval from Apple. However, the Mac App Store is a fairly popular and convenient way to distribute your apps. If you want to be a registered developer on the Mac App Store there's a separate fee from the iPhone developer registration. If you're struggling to make money on the iTunes App Store, do you really want to drop another $99 on the Mac App Store? It could be a tough question.

The Mac Desktop has more power. So, if you are struggling to optimize your game for iOS devices, your game might be more successful on the Mac Desktop. That's very important to consider when designing Mac games. How will your game take advantage of the system? Playing with a mouse and keyboard can be very different than a touch screen and an accelerometer.



While publishing your game, there's the option to include a 512x512 icon. An import thing to remember is that the icon should be scalable. Even though the icon is 512x512, it's rarely shown at that size. It typically sits on the desktop, at a fraction of the original size. The other big publishing setting is in the

“Advanced Options”. You can have your game ”Start in Fullscreen” mode. If “No” is selected, the game will launch in a window. “Fullscreen” mode can still be activated if command+F is pressed or “Fullscreen” is selected from the menu. This feature will cause the game to cover the entire screen. If the game does not match the screen ratio, black bars will be added to the sides. Those black bars prevents distortion of your graphics. If you want your game to automatically start in Fullscreen mode, then select the “Yes” option.



Platform Specific Settings Settings for proper platform publishing

☑ Publish to Mac App Store

Photics

To publish to the Mac App Store, create and download developer certificates from the Apple's Mac Developer Program Portal.

In the Platform tab of the GameSalad Publishing Manager, there’s the option to select, “Publish to Mac App Store”. That’s where the developer registration is needed. By clicking the “Mac Developer Program”, registered developers can get started on creating the necessary certificates. You'll need three certificates - Mac Developer Application, Mac Developer Installer and the WWDR certificate. I had lots of trouble downloading that last one. I had to grab it from the iOS side of Apple’s developer site.

Even if you have no desire to distribute Mac versions of your games, it’s a great way to capture raw video footage. With “Fullscreen” mode, you can use screen-capture software to grab high-res footage. This also resolves the issue of having to crop your video. If you record footage directly from the GameSalad previewer, that footage will need to be cropped. A “Fullscreen” Mac version of your game makes it easier to create a promotional video.

GameSalad’s main function seems to be support for iOS devices. The Mac is more of a fringe benefit to using GameSalad. However, there is still some potential here. You could create free versions of your game, in order to encourage the purchase the deluxe or full version of your game. A free Mac game could also be used to promote iOS versions of your game. The Mac App Store is also off to a strong start. Pixelmator — a graphics program for the Mac — grossed $1,000,000 in just 20 days. The decision to transition the software in a Mac App Store exclusive worked out for the makers of Pixelmator. Will supporting the Mac App Store work out for you?

Chapter #22 - Web Publishing

GameSalad is not just about creating games. It's also a gaming community. The creators of GameSalad are in the process of building up a game site. That's where Web Publishing enters the picture. Similar to Flash, you can embed your games on a web page. The main difference is that GameSalad uses HTML 5. That means your games can be played on lots of different platforms, but without the need to download a special plugin. Only an HTML 5 compatible browser is necessary.

Web publishing is another fringe benefit to using GameSalad. However, there are some issues in this area. One of the major issues is that the technology is bleeding-edge. Older web browsers simply do not support HTML 5. Newer browsers might have some compatibility issues. Another issue is that GameSalad does not offer much flexibility with web publishing. For example, Web Publishing has restricted size formats. Currently, you can only use 480x320 for your games. This is yet another reason to consider landscape for your game's default orientation.

Arch Fiery was created at 480x320 pixels, but it still wasn't suited for GameSalad publishing. With lots of actors, particles, parallax scrolling and looping audio, the performance just wasn't there. Some game features didn't work in HTML5. Other features worked, but ran slowly. If you're serious about publishing a good web game, you might want to be minimalistic in your design.

Another issue is that the Web Publishing feature puts your game on the GameSalad.com website. The software doesn't export a file for you. Instead, your game is put on the GameSalad.com website. This is an important concept to understand. If you enable game embedding, others can put your game on their website. They can put banner ads around your game. They could be making money from your game — while you make nothing.

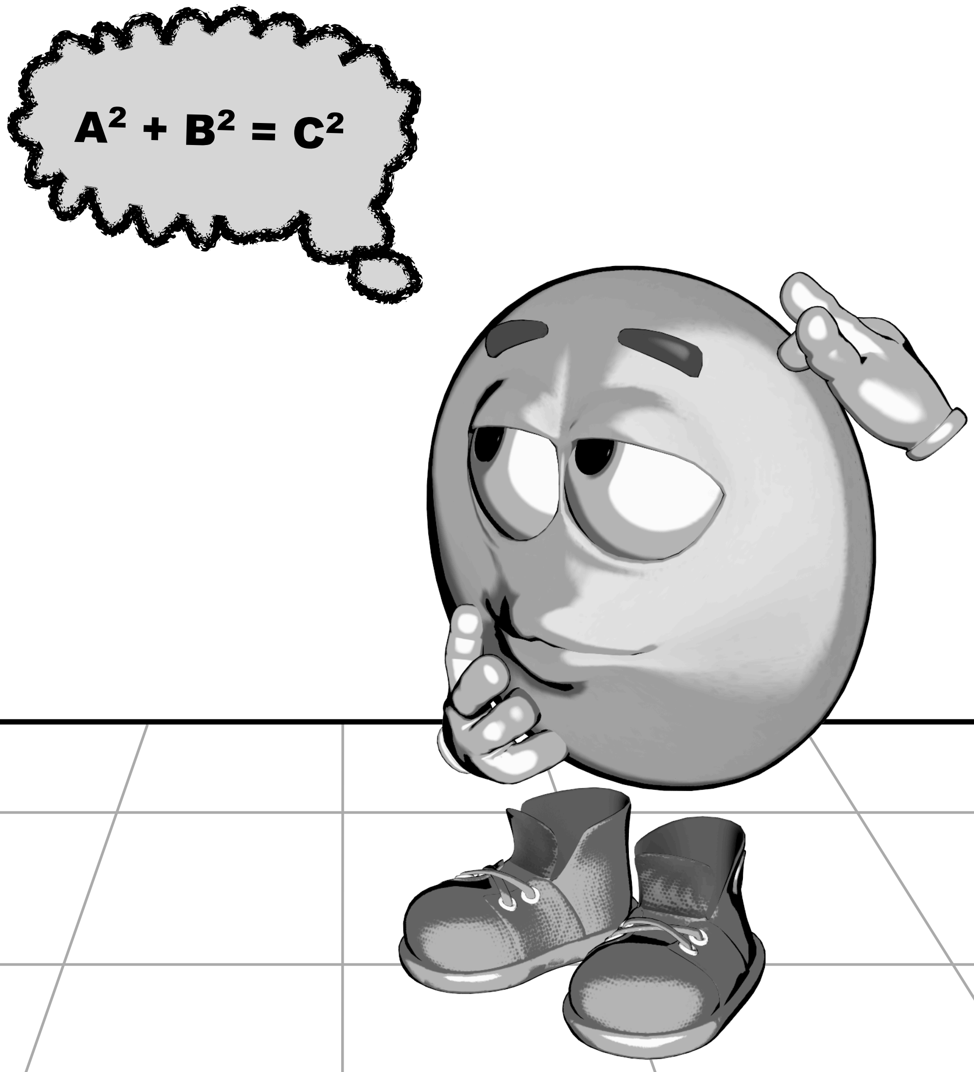
So, why do it? Why use the web publishing feature if it's not very robust? Well, its good marketing. Sharing a GameSalad game is a good way to make a reputation for yourself. You can earn credibility with your GameSalad peers by tossing a good game up on the GameSalad website. If you're popular in the community, the more likely people will be willing to help you. This is also an excellent way to test your game. With iOS publishing, your game's first shot is basically its best shot. If you mess up your launch window, it's pretty much over for your game. It's incredibly difficult for a buried game to make a successful comeback. By sharing your project with the GameSalad community, you can get feedback on your game. It's a good test run before publishing to the App Store.

Web publishing is also a way to get introduced to GameSalad. Before you spend any money on the software, you can complete a trial run. From conception to production, you can experience what it's like to make your own games. If you feel tired or frustrated, this might not be something to pursue. But if you feel invigorated, excited and proud, you're probably on the right track. Learning how others feel about your work is important too.

However, be careful. Others could rip off your work. Once your game is online, it can be reverse engineered. With your assets exposed to the entire world, unsavory individuals might incorporate your best ideas into their game.

A good idea might be to create a limited version of your game, with links to the full iOS or Mac version of your game.

8



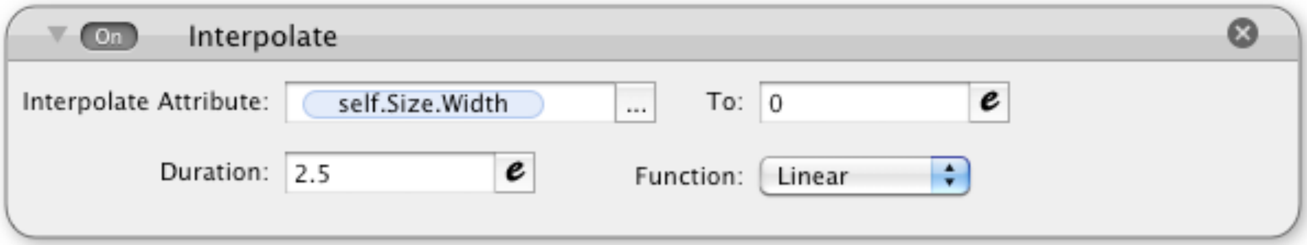
Section VIII - Advanced Techniques

23 - Psuedo 3D
24 - Particles
25 - Limit Smashing

Chapter #23 - Pseudo 3D

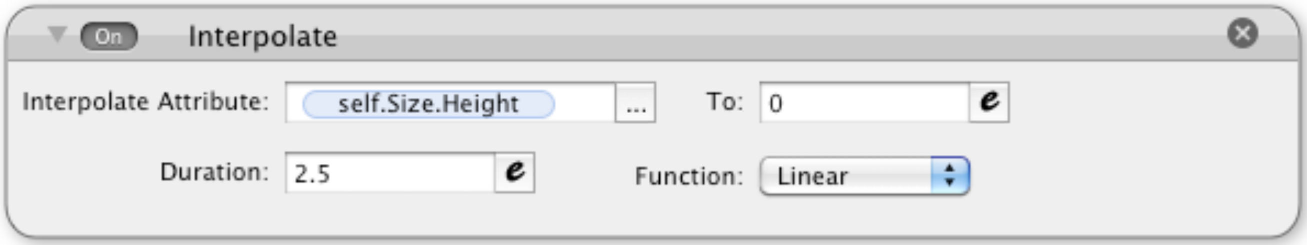
GameSalad is not really for creating 3D games. However, you can fake some cool 3D effects with the software. That's what this chapter is about.

Fake Z Axis - If X is width and Y is height, then Z is depth. GameSalad actors don't support the Z axis, or do they? When something is closer to you, it appears larger. That's an easy effect to mimic. You can just change the size of an actor. With the "Growth Rate" behavior, you can make actors zoom in and out. You could also use "Change Attribute" or "Constrain Attribute" on the actor's width and height values.


Interpolate

Interpolate Attribute: self.Size.Width To: 0

Duration: 2.5 Function: Linear


Interpolate

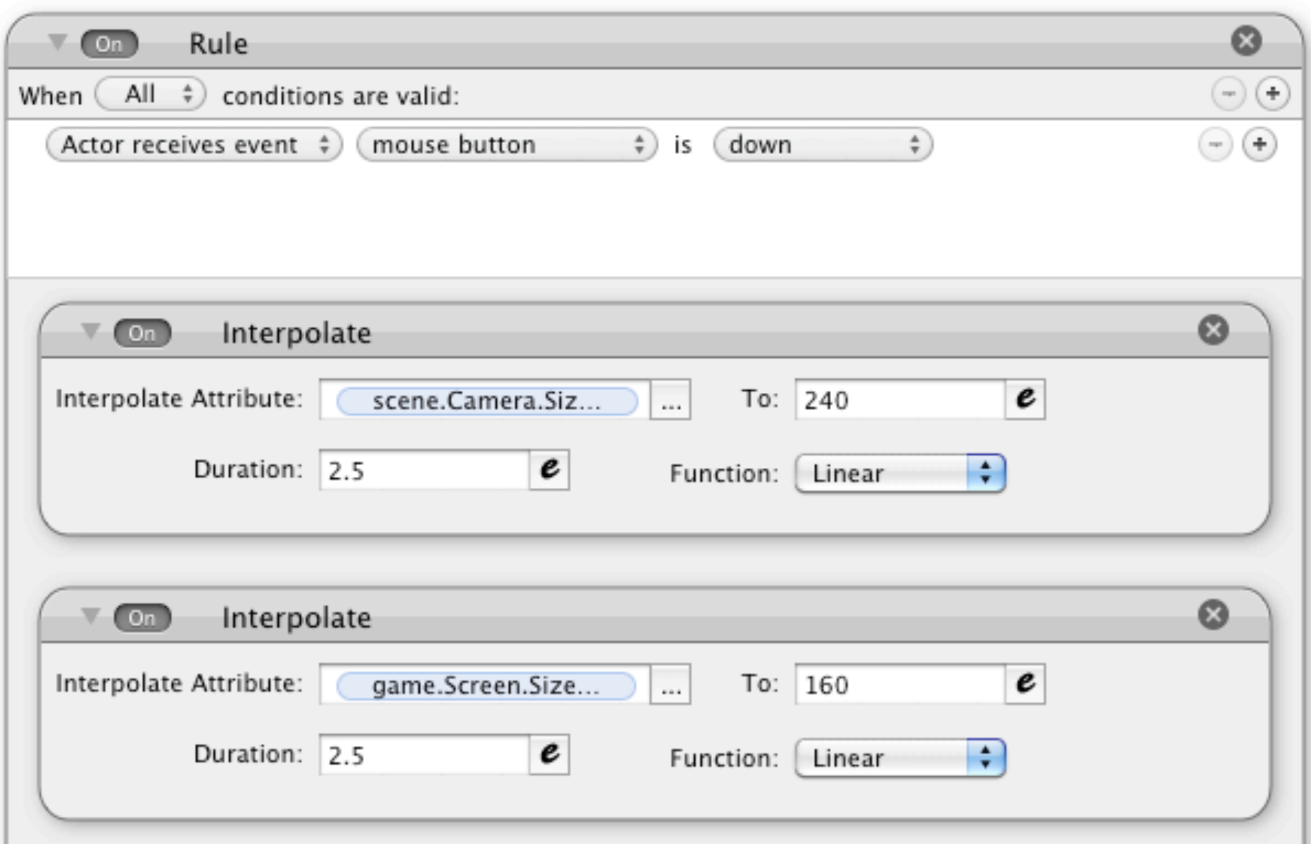
Interpolate Attribute: self.Size.Height To: 0

Duration: 2.5 Function: Linear

With two "Interpolate" behaviors, projectiles can be scaled. As they get smaller, it helps create the illusion of traveling into the distance. That's because they're shrinking. This technique could be adapted to create other 3D effects.

There is a danger in this technique. The collision detection might not work properly. If you notice your bullets going through actors, or if overlaps are not being detected, you can fix this problem by using a quick "Timer". If you change the shape every few split seconds, GameSalad has a chance to catch up and process the changes in shape. The animation won't be as smooth, but that's less noticeable than missed collisions.

You could also change the size of the camera. Instead of scaling a few actors, you can scale the entire screen by changing the camera size. I've been pondering the possibility of a sniper game — with a scope that zooms in and out — but zooming the camera is tricky. The problem is offsetting the HUD. If you're going to try a similar technique, there's a tough decision to make. Do you use camera zoom or do you just scale the actors?

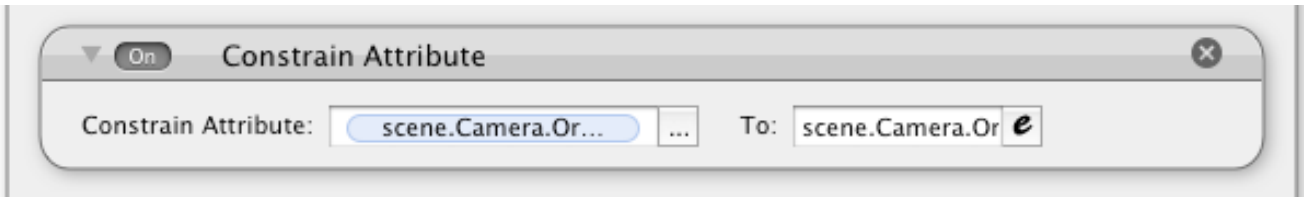

Rule
When All conditions are valid:
Actor receives event mouse button is down

Interpolate
Interpolate Attribute: scene.Camera.Siz... To: 240
Duration: 2.5 Function: Linear

Interpolate
Interpolate Attribute: game.Screen.Size... To: 160
Duration: 2.5 Function: Linear

If you decide to zoom the camera, the above image shows how to accomplish that effect. Start with a "Rule" behavior. When the "mouse button" is "down", the two "Interpolate" behaviors can change the camera's size — one for the width and one for the height. They are scene behaviors, so you'll probably need to edit an actor directly on the scene. To double the size of the scene, I cut the camera size in half.

I find that camera movement is somewhat inconsistent with this technique. The "Control Camera" behavior doesn't work as smoothly. To compensate for this issue, the camera can be controlled manually.

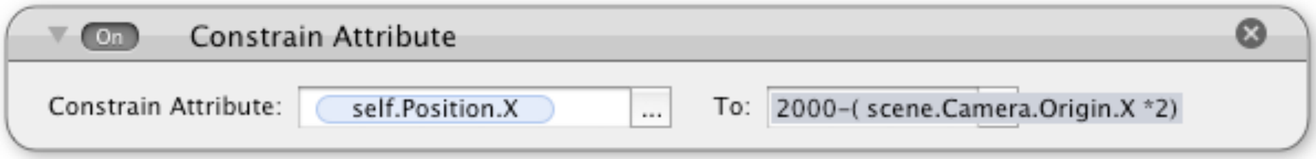

Constrain Attribute
Constrain Attribute: scene.Camera.Or... To: scene.Camera.Or

In the example, the scene.Camera.Origin.X and scene.Camera.Origin.Y attributes are constrained to the X and Y mouse positions. However, an offset is needed. For a 480x320 sized game, I subtracted 240 from the X mouse location and 160 from the Y mouse location. When the button or touch is released, the "Otherwise" portion of the "Rule" pulls the camera back to its original size and location. That's all there really is to it. It's a great effect, very smooth and lots of room for improvement.

Parallax Scrolling - Back in the days of 16-bit gaming, I used to stare in amazement. Different background layers were moving at different speeds. It was a common effect for platformers and side-scrolling games. This effect created such a sense of depth that was amazing to me. The far background scrolled slower than the near background. Objects on top of the main character moved faster. Two decades later, shouldn't the same effect be in your games? Ah, but it's tricky. This is a common question on the GameSalad forums, so I knew I had to answer it here.

Naturally, placing your background actors on different "Layers" is a good start. A separate layer for each level of parallax scrolling is not required, but it can help keep things organized. By simply putting them in the scene they're assigned to a certain order. If you've made it this far in the book, knowing how "Layers" work isn't the problem.

The problem is making the background move with the main actor, but not when the camera has stopped scrolling. That's the trick. Your main layer doesn't move. The camera does the moving — which creates a scrolling effect. But the near and far background, they should move relative to the camera's position.


Constrain Attribute

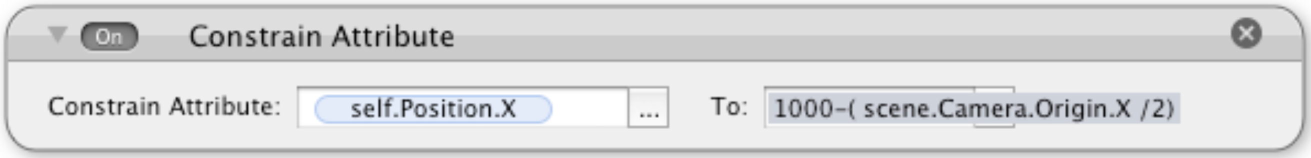
Constrain Attribute: self.Position.X To: 2000-(scene.Camera.Origin.X *2)

The above "Constrain Attribute" is an example of how to move the far background. It moves slower than the main actor and it should be underneath the main playing area. The 1000 number is an arbitrary starting point. That value depends on your game. The real action is with the scene.Camera.Origin.X attribute. It is divided by 2 to get a slower effect. You could divide by more or less to create different scrolling speeds. The value from inside the parentheses is subtracted from the arbitrary starting position.

That's it. There's the secret formula. You can create multiple background layers by giving them different dividers. An even further background could use 3 or 4 as a divider. It depends on your game. And for an extremely far background, simply constrain it to the camera's origin. (Like with the sniper zoom tutorial, you might need an offset. The camera's origin is at the bottom left, not at the center.)

The trick is to map your scrolling to the camera, not to the main character or controls. Otherwise, your main actor might bump into something and stop… but the backgrounds will keep on going.

If you want to add some trees in the foreground, or other scenic items to put between the player's vision and the main character, you can slightly reverse the process.



Constrain Attribute: self.Position.X To: 1000-(scene.Camera.Origin.X /2)

By creating a multiplier, the foreground actor can move faster than the scrolling camera.

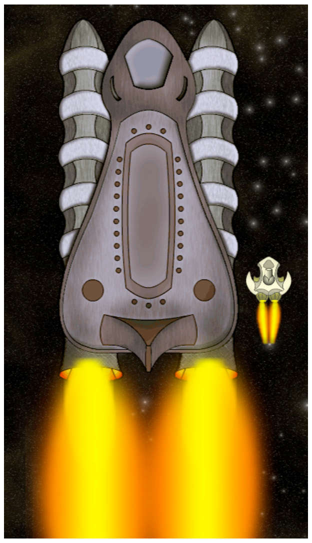
2000-(scene.Camera.Origin.X*2)

The problem is that this can become quite a processor intensive task. If you use lots of large images, your game could lag. To help prevent that issue, you could look for ways to tile your backgrounds and foregrounds. For example, fog or animated rain could be an excellent foreground effect. For the background, you could create tiles for sand, dirt, rolling hills or other types of patterns.

Animate - Even the best 3D game engines are still performing the same basic function as GameSalad. That's displaying a 2D image to a screen. Mac, iPad, iPhone, this book, it doesn't matter. The image is flat. So, if you need a specific 3D effect, maybe the "Animate" behavior can help you out. A series of animated images can create a 3D effect.

While GameSalad is primarily a 2D engine, it can be pushed a little further. That little extra flair could be what separates your game from the thousands of other games. I don't recommend sacrificing performance or playability for pseudo 3D effects. However, if you are mindful of optimization, you can pull off some neat tricks. If you think it's impossible, you might want to look back at classic arcade games. The developers of those games did so much with so little.

Chapter #24 - Particles

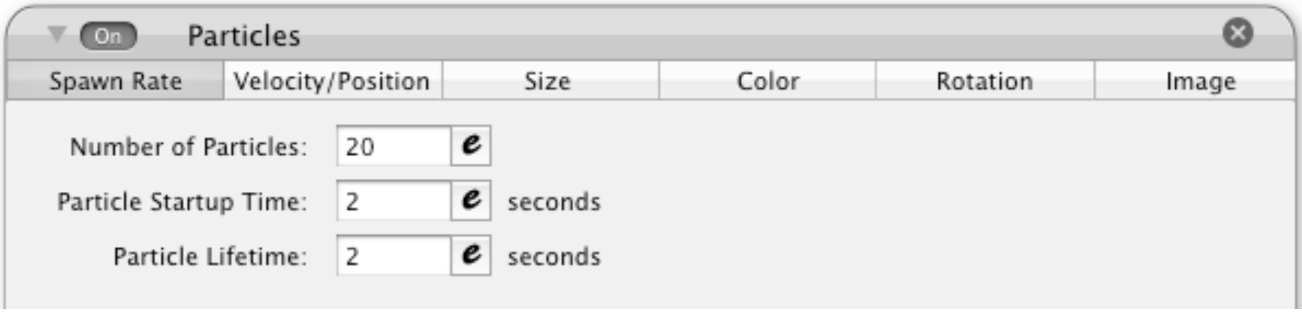


I remember playing EverQuest® on my computer. It was pretty amazing. I was in this vast world that was filled with danger and adventure. I also remember how the pursuit of lofty loot was a major driving force in the game. People would actually pay real money for virtual items.

What made those items so special? I remember questing for days, in order to get a fiery sword. It wasn't really that special of a weapon. The only truly remarkable thing about it was the fire effect. The blade was engulfed in flame. The weapon was more than some tool to slay large rats with. It was a status symbol. By wielding such a prestigious weapon, I had gained status with my peers.

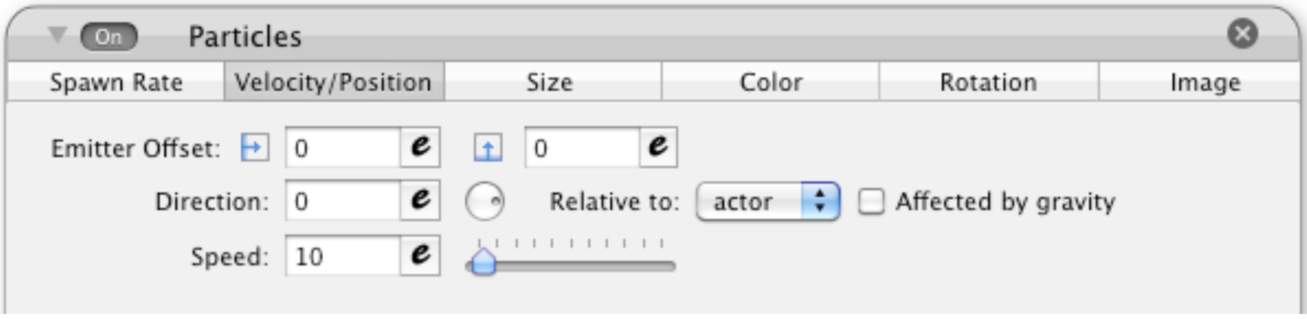
I remember that EverQuest graphics were a bit low in polygon counts. The characters and the landscapes weren't that impressive to me. But particle effects, EverQuest got that right. Seeing the screen aglow with dazzling color made the game more entertaining. It's the same with GameSalad. The "Particles" behavior puts a little extra oomph in the game. It can make your games prettier.

I remember the joy of adding a thrust effect to my first GameSalad spacecraft. There was something about that glowing energy, it made my game feel more professional and more alive. That's what this chapter is about — adding particle effects to your game. This is one of the most powerful GameSalad features, which is why I'm calling attention to it. This chapter is dedicated to a single "Behavior". That's because it's one of my favorites.



Number of Particles: 20
Particle Startup Time: 2 seconds
Particle Lifetime: 2 seconds

If you don't think that the "Particles" behavior is special, then how do you explain the six tabs? It's got power. With one actor, you can flood the scene with hundreds of little images. With the default setting, only 20 are used. That because you can seriously slow down your game if the "Number of Particles" setting is too high. The "Particle Startup Time" is how long it should take until all of the "Particles" are shown. Their life is fleeting. With the "Particle Lifetime" option, you can decide how long it takes before the "Particles" are destroyed. It's not unusual to tweak these settings a lot, trying to find balance is between performance and special effects.

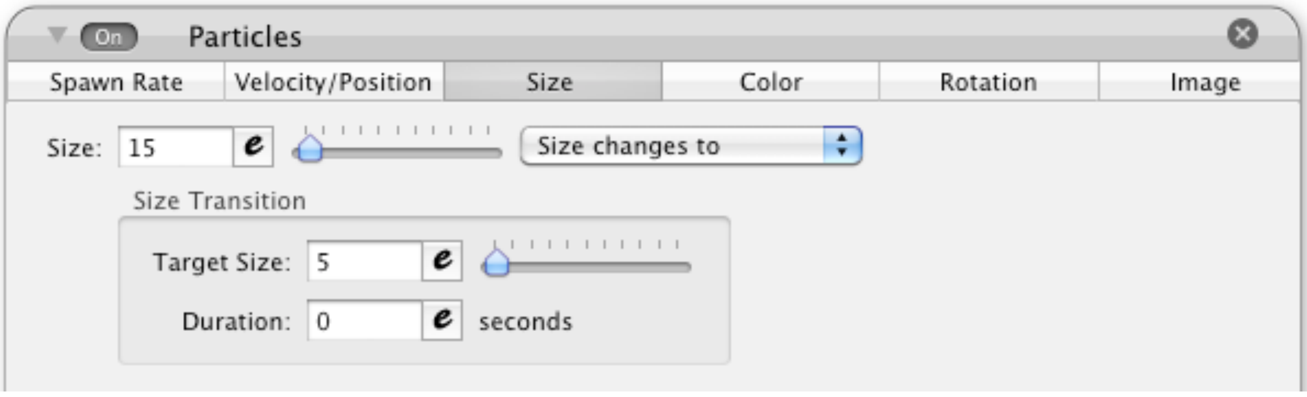


Emitter Offset: 0 0
Direction: 0 Relative to: actor ☐ Affected by gravity
Speed: 10

"Particles" don't have to just sit there. You can make them move with the "Velocity/Position" settings. First, you can set the "Emitter Offset" values. This is the X and Y positions from the center of the actor. By entering positive or negative numbers, you can determine where the particle should appear. This gives you a great opportunity to be creative. By adding functions, you can create unique effects.

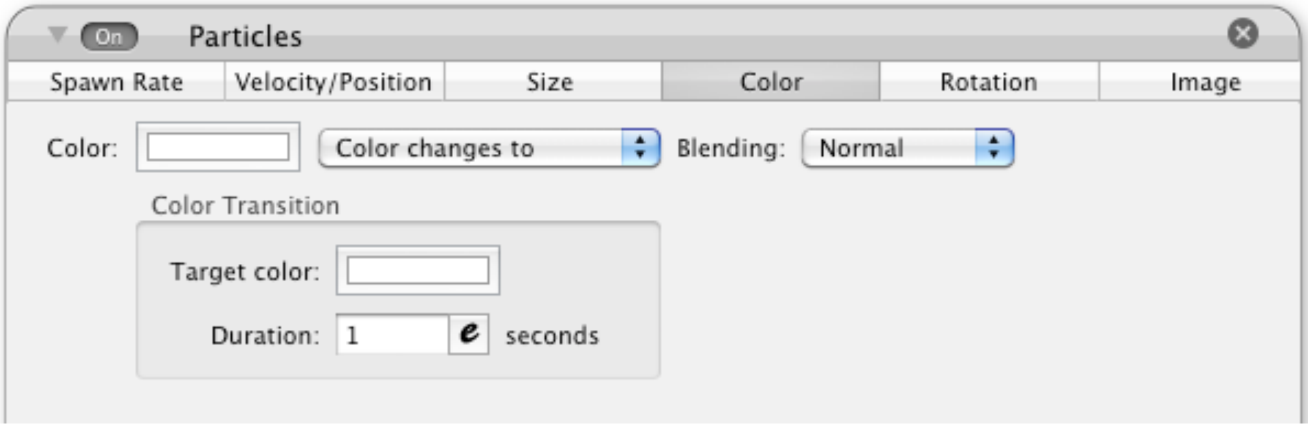
sin(game.Time*120)*88
cos(game.Time*120)*88

With "sin" and "cos" functions, the particles are spawned in a circular pattern. You could add the "random" function to create even more variety. For my space fighter, I used both the self.Rotation value with the "sin" and "cos" functions. The thrust should only appear from the back of the spacecraft. With two expressions for the X & Y offsets, the thrust effect appears more natural.



Size: 15 Size changes to
Size Transition
Target Size: 5
Duration: 0 seconds

With the “Size” tab, you can set the size of the “Particles”. However, there is only one value. That’s because “Particles” are squares. Both the height and width are the same. With the “Size Transition” option, you can change the size of a particle. The “Duration” option controls how long it will take for a particle to change shape. That value should be shorter than the “Particle Lifetime” value. Otherwise, the transition will not complete.



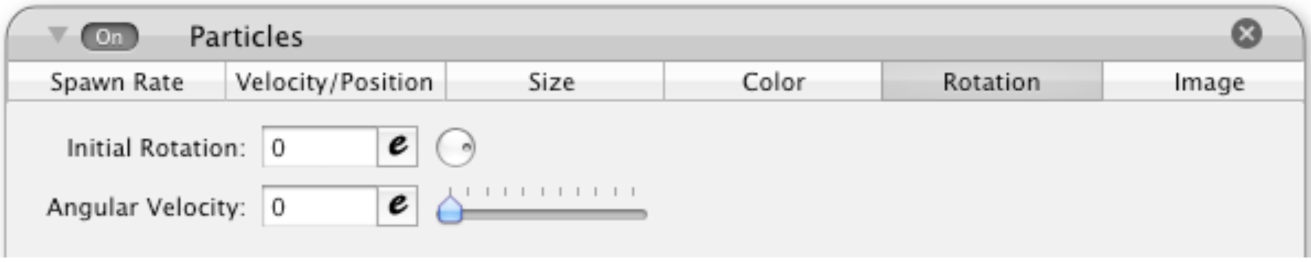
Color: Color changes to Blending: Normal

Color Transition

Target color:

Duration: 1 seconds

You can use the “Color” options to linearly change the color of the “Particles”, or you could just pick a solid color. This setting will tint an image. You can also choose a “Blending” mode. I find that “Additive” is great for simulating fire.



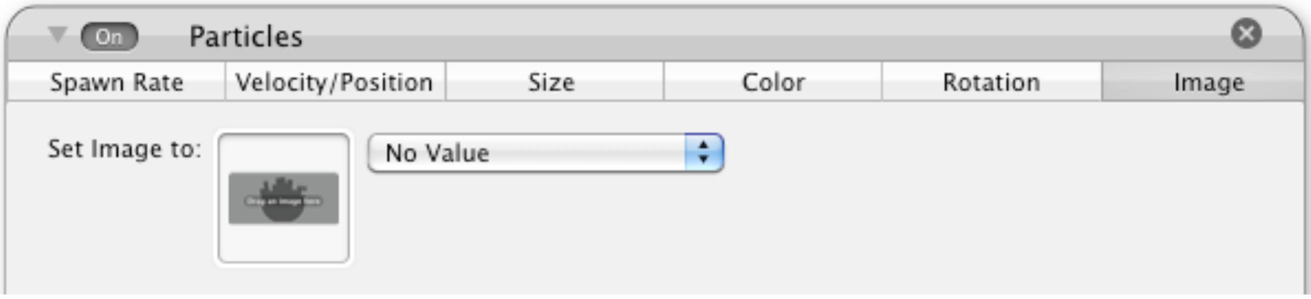
Initial Rotation: 0

Angular Velocity: 0

The “Rotation” tab controls how the “Particles” spin. The “Initial Rotation” value is the angle of the particle when it first appears. If you want it to be aligned with the actor, you could put self.Rotation as the value for this setting. Other common settings for this value are zero and random(0, 359). Zero doesn’t change the angle of the particle, while the “random” option places the particle with an arbitrary direction.

The “Angular Velocity” setting determines the spin of the “Particles”. This can be negative or positive number. If the value is positive, the spin is counter-clockwise. If the value is negative, the spin is clockwise. If this value is assigned to an “Attribute”, and that value is changed, the new value will not change the “Angular Velocity” of an existing particle. For variety, you can assign a “random” function to this field. Each new actor will move with a random spin.

A good way to test your GameSalad skills, and to see how particles work, is to create a project where you can control the different settings — live. Just by playing around with the different values, you might get great ideas for your game.



Set Image to: No Value

With the "Image" section, you can choose the graphic for your "Particles". Existing images in your project can be accessed via the drop-down menu, or you can simply drag-and-drop a graphics file onto the image box. Optimization is huge. With so many copies of the graphics file displayed on the scene at once, the memory and processor requirements can be intense.

Realizing this, I wondered if I needed a graphic at all. No, I didn't! I created a nice space background with simple white squares. With three different copies of the "Particles" behavior, I created layers of stars. The smaller stars moved slowly. The bigger stars rotated and moved quickly. Using the same general idea, I was also able to create a 3D effect. The stars spun out from the center, as if the main character was flying through space. Using the "Color" options, the stars would get brighter as they got closer to the actor. Using the "Size" option, the stars would get bigger too.

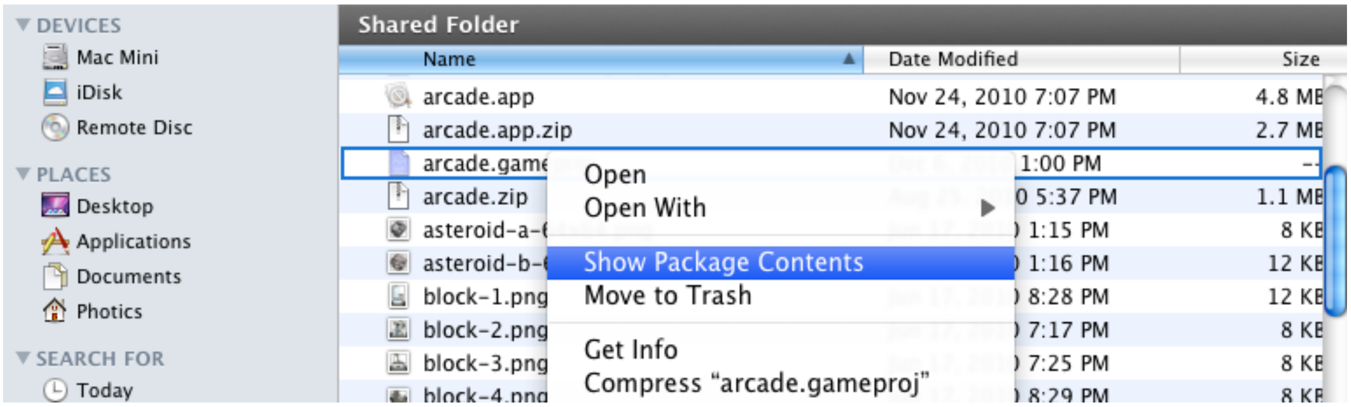
This technique worked wonderfully for my classic-style games. I was able to put about 100 stars on the screen without any significant slowdown. The difference in performance is huge. Large graphics for background can kill your game's performance. This is especially true if you're building a scrolling game. A combination of tiles and clever use of "Particles" can help conserve on power. The "Starfield" template has two examples of "Particles" without images.

That concept might seem quite shocking to some GameSalad developers — using "Particles" to optimize your game. But if you're careful about the size of your graphics, you can create some amazing effects and not kill your game's performance.

Chapter #25 - Limit Smashing

With GameSalad, I often think something is impossible to do with the software. But with some creative thinking, a lot of those barriers are smashed. This chapter is about smashing those GameSalad limits. What was once thought of as impossible, is now possible.

I can't sort my attributes - That's not entirely true. You can sort them, but you'll have to do it manually. You can do some editing to your GameSalad project by accessing the raw files. You can do this by locating the project file, right-clicking the file and then selecting "Show Package Contents".



Name	Date Modified	Size
arcade.app	Nov 24, 2010 7:07 PM	4.8 MB
arcade.app.zip	Nov 24, 2010 7:07 PM	2.7 MB
arcade.game	1:00 PM	--
arcade.zip	5:37 PM	1.1 MB
asteroid-a-	1:15 PM	8 KB
asteroid-b-	1:16 PM	12 KB
block-1.png	8:28 PM	12 KB
block-2.png	7:17 PM	8 KB
block-3.png	7:25 PM	8 KB
block-4.png	8:29 PM	8 KB

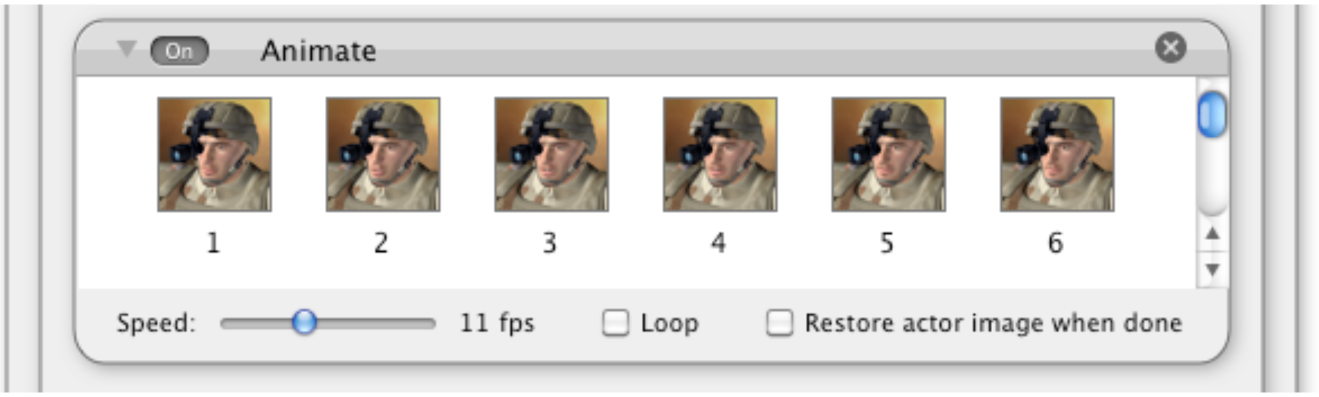
Once inside, you can edit the object.xml file. It might look scary, but it's just xml code. There's nothing terribly challenging in there. You do have to be careful though. You might want to backup your project before venturing into this area. Although, if you are brave enough to mess around in here, you can access your files directly.

As an example, the icon for your project is in the screenshots folder. It's the primary.png file. I usually don't mess around too much in here. Although, it's great as a backup. When you make a copy of your GameSalad project, you're making copies of all the assets in that project. It's like the poor man's version control.

I can't move an actor out of a layer while the game is in play - True, you can't get an actor out of a layer while the game is in play. What you can do is quickly delete that actor and then have that actor spawned in the scene you want. If your game is well optimized, the player probably won't notice the difference. You might need to create spawner actors in your target layers. With an "Attribute" and some rules, you can toggle spawning and deleting.

However, to make the effect truly seamless, you might have to carry "Attribute" values over to the new actor. Why go through the hassle? This workaround can create lots of cool effects. If you're creating a platformer, an actor can jump behind a box or fence. In general, this problem can be mitigated by limiting the number of layers — one for the HUD and one for the action. Yet, the destroy/spawn technique shows that there are creative ways around GameSalad limitations.

I can't import video files - Even though GameSalad doesn't accept video files, what is that really? It's pretty much just a bunch of still pictures and an audio track. You can mimic that with GameSalad. Just use the "Animate" behavior to show your frames and then play an audio file with that actor. It's not the most optimized video player in the world, but it can get the job done.

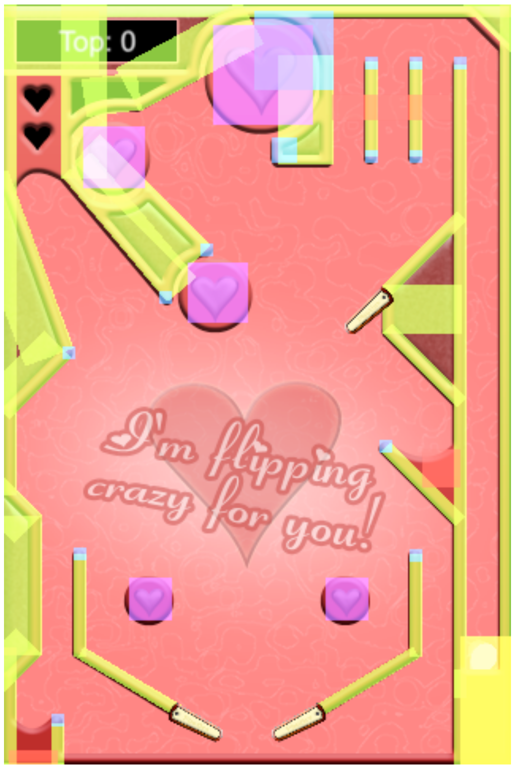


Speed: 11 fps ☐ Loop ☐ Restore actor image when done

You might want to create a non-standard size for your video. A 320x480 video is not an efficient size for GameSalad. But if you crop your images to 256x480, you can save a lot of memory. A square video (256x256) could save even more memory. This is a way to add some quick cut-scenes to your game. You could even mix it up a bit, by adding video elements to your game. If your character access a computer terminal or a TV in the game, the screen could flash a video.

I can't create custom collision shapes - So, you want something more than a circle and a square. Actually, you could do a lot with those two shapes. With "tags" and invisible actors, you can flood your game with all sorts of collision shapes. In my first GameSalad pinball game, I outlined the edges of my playing area with rectangle shapes and some circles. This allowed me to create a unique shape for my game.

If you would like to see an example of using rectangles and circles to build complex collision shapes, the "Secret Admirer" template uses this technique. The third mini-game is pinball.



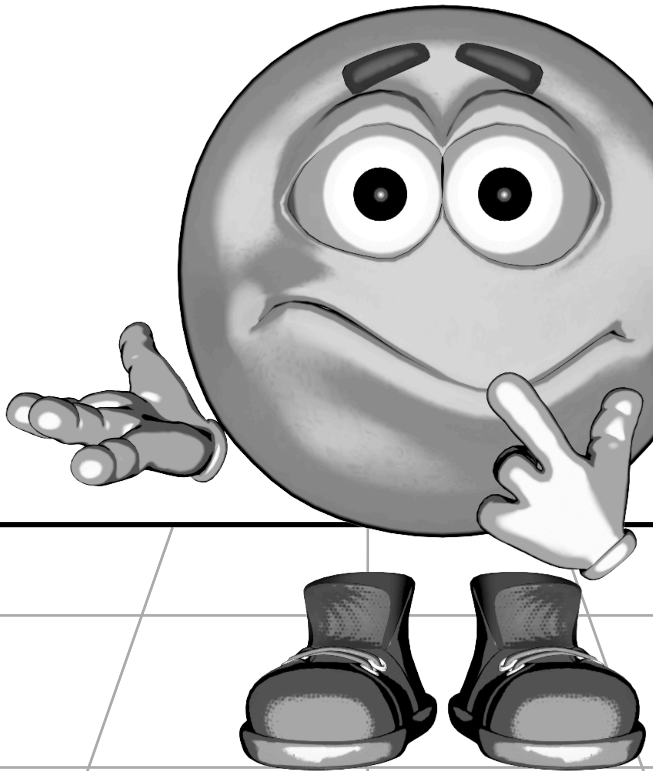
The image to the left shows the game with visible hit boxes. If you don't think this is enough power, perhaps you might have missed the popular fighting games from the early 90's. When a character throws a punch an aggressive hit box is assigned to the fist and part of the arm. The enemy has a few target hit boxes. It could be one for the head, another for the torso and a third for the legs. If the punch hit box overlaps a target hit box, that counts as a successful punch. There are a lot of games you can make with the basic concept of placing, constraining or spawning hit boxes.

If you look closely at the image, you can see different colored shapes. The cyan and purple boxes are actually circles. This helps to create rounded edges. Since the hitboxes don't move or collide with each other, those actors can overlap. To show the overlapping, the hitboxes were set to transparent. This is one of the annoying parts. Once development is over, the hitboxes cannot just be alpha zero. They have to have their "Graphics" set to "Invisible". Otherwise, the performance hit could be too great. That means each collision shape needs to have its visibility manually disabled.

As your understanding of GameSalad advances, you might find yourself struggling with the limits of the software. Some of the workarounds can be quite tedious and time consuming. That's when you might be ready for the next level of game development.

A lot of people choose GameSalad because they "hate to code". If that sounds like you, then you might want to better understand why. Do you hate to code because you don't like it or is it because you don't understand it? If it's the latter, perhaps revisit some programming challenges after you've mastered GameSalad. Coding might not seem as confusing to you. Does that mean you should learn to be a programmer? If that's not what you enjoy, that's understandable. However, there is a warning with GameSalad. If you stay within the limits of software, you and/or your games could have trouble moving forward. There are lots things that GameSalad simply doesn't do.

9



Section IX - Now What?!

26
Promoting Your Games

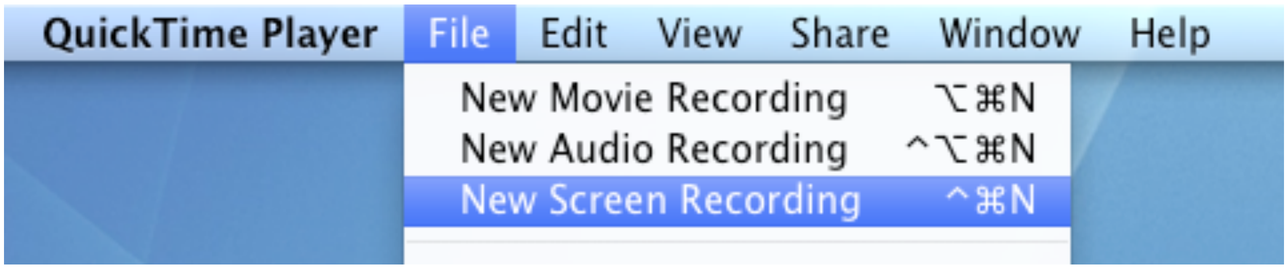
27
The Future

Chapter #26 - Promoting Your Games

Even after you turn your great idea into a game, it takes more to succeed. Your hard work on your game is not enough. Your great game concept is not enough. In order for your game to succeed, you have to get the word out there. You have to convince the masses to download and play your game. That's the focus of this chapter — promoting your game.

Make a video - Just before your game launches, you should have a video of your game ready. They are called "video" games after all, so screenshots just aren't enough. By creating a video for your game, you're showing potential customers that you're a hard worker and that your game is worthy. Although, that assumes your video is of decent quality. It is probably better not to have a video rather than to distribute a poor one.

Just like many other areas of GameSalad development, you can hire someone to create a video for you. However, you might want to try it yourself. If you've already got a new Mac with Snow Leopard, you already have many of the tools required to make a video.



By launching the QuickTime® Player, you can make a "New Screen Recording". You can record footage from the Mac desktop version of your game. You could also grab footage from the GameSalad "Preview" mode. However, this is video only. If you want to include audio too, you'll need a male-to-male audio cable. That's to send the sound from the headphones to the microphone. If you don't want to mess with that, you could simply lay down an audio track for your game.

But in order to edit audio and video content, you'll need editing software. Since I was trying to save money, I used iMovie®. The software was included with my Mac Mini. I decided to try it out. I was quite impressed. There are some great special effects included with the software, including transitions and templates.

If you don’t mind spending a little money, there’s a neat Mac App called Sound Stage. It easily snags video footage from the iOS Simulator. I used it to create the promo for BOT. If you’re interested in seeing how TANK turned out, he’s in that game.

If you’re going to send your app off for review, it could help to have a promotional video. By creating a solid marketing package, you’re sending a message that you’re a professional.

Pricing - This is a tough issue with selling your app. The gamers in iOS land are spoiled. They have thousands of games to choose from. Why should they pick yours? If you price your app too high, they probably won’t. What’s too high? Apparently, anything more than tier 1 (which is 99¢ for the United States) might be too much. The higher you price your app, the harder you’ll have to market it and the better your game will have to be.

A lot of developers just default to tier 1. It’s just too hard to fight against the surge of cheap apps. However, there's a possibility to be sneaky. If your app is featured, meaning that Apple likes it and calls attention to your app, you could raise the price. Personally, I consider this type of price switching to be unethical. I believe that all of my customers should be treated fairly. I don’t think it’s fair that someone should pay $2 for my game on Wednesday, knowing that I’m going to charge $1 for it on Thursday. I set the tier for my apps and I rarely budge.

This ideology can hinder marketing efforts, as being creative with your pricing can help drive sales. Featured apps are more likely to be purchased than apps that are not. In fact, out of all the marketing efforts that I tried, being featured was the most effective at generating sales. Apple decides what apps they want to feature. That’s why it’s important to build a great game and price it competitively. Launching the game without technical issues can also help a lot.

Even if you contact dozens of review websites, make promotional videos, send out all 50 promotional codes and you could tell all your friends about your app, it still might not make a difference. Your app could still perform poorly. Even if you build a good app, it simply might be overlooked. It’s a mix of luck, skill and good timing.

Surge - The release day is huge. When you see that your iOS app is “In Review”, be on high alert. Be ready to control the release date setting. You can check the status of your app in the iTunes Connect website. Once your app is set to launch, get ready to build a surge. You have a 24 hour window to generate as many sales and downloads as possible.

You could try picking a launch date that’s in the future, or you could try setting your app to launch the next day that it’s approved. The trick is to get listed on the top of the new releases listing. Even if you try your best, a weird Apple glitch could thwart your efforts. It's very frustrating to wake up on launch day and see your app not listed properly.

That's why the iTunes App Store might not be the best place to start. Launching first on Android has advantages. The first is that there's no lengthy review process. You can launch your game, get immediate feedback from players and then quickly resolve those issues. You could launch several versions of your game on Android before Apple even gets done reviewing your first version. With the knowledge gained from Android, you could have a more successful iTunes App Store launch.

Promo codes - Don't just give these away to anyone. I don't even recommend giving them away to review sites. At least, not at first. Since there are so many developers out there, hungry for attention, you might be wasting your code. I hate sending out codes, only to realize that the code has gone unused. Instead, maybe contact the review websites early, see if they're interested in an AdHoc version of your game. Additionally, promo codes are not as powerful as they once were. Apple has disabled reviews from promo codes. That means someone actually has to buy the app to review it. Only give away promo codes to those that may help your cause. Not only is it important to give away a code, it's important to track that activity. Which websites are actually reviewing your game? If they don't review your game, but they try to sell you an advertising package instead, then you can boycott them from future game launches. Without proper tracking, you could be repeating your mistakes.

Sometimes review sites want something special, like exclusive information or an interview. Just bombing them with a promo codes isn't a good idea. Try to see their needs. They want to build and maintain a popular website. You want a popular game. How can you make it mutually beneficial, without being illegal or immoral? Again, ethics are a concern. In desperation to receive fame and fortune, you may be tempted to fake reviews or cross moral boundaries. I recommend against it. Stay strong. I think it is better to win with hard work, determination and skill.

Invest - Here's an interesting thing that I noticed about my GameSalad budget. I'm willing to spend money on graphics and audio content, but I haven't spent anything on marketing. That realization is a bit shocking to me, especially considering that I used to work in Marketing. I know how the game works. Big companies spend millions of dollars in trying to reach such a tiny percentage. For example, if you paid for the creation and distribution of 10,000 fliers, but only 100 people responded to those fliers, was it worth it to send that many fliers? That depends on how much money was invested and how much money was made. If it was $1000 to create and distribute the fliers, but each new customer spent $100, that's $9,000 of new revenue. But unfortunately, that strategy doesn't work for 99¢ apps in the app store. If you're making less than a dollar per app, that doesn't leave a lot of room for marketing costs. You might want to spend some money on marketing, but it will have to be creative and inexpensive. Otherwise, you could lose more money than you make.

Make More Money - By joining Apple's affiliate program, you might be able to get back some more money. As an example, if I link to my apps from Photics.com — and someone successfully buys my app after clicking the link — I get a 5% bonus. So instead of giving Apple 30%, the number is reduced to 25%. That extra 5% could be redirected towards future marketing efforts, like a launch day surge for my next app.

Build A Following - It's going to be really difficult to hit a home run on your first swing — especially if you're starting out as an unknown. To combat this takes patience, planning and time. Just to get your name out there, you might want to launch some games for free. That can be a tough decision to make, especially if you've just spend lots of money on software and hardware. Building a following can also an investment.

That's where GameSalad Pro is helpful. It gives you options in building a following. Open URL could be used to link to your website or social networks. iAds could be used to offset the costs of releasing free apps. But most importantly, GameSalad Pro gives you access to different markets... Mac, Windows, Android, iOS. Those are some big spaces to make a lot of new friends.

Chapter #26 Summary

- Creating a promotional video can help with the marketing of your game.
- The iTunes App Store is very competitive market, with ridiculously low pricing. That can limit your marketing and pricing options.
- The launch day of your app is critical to the success of your app. If you get it wrong, your game could be buried by the competition.
- Be careful with promo codes. You only have a limited amount with each update. If you give them to the wrong people, the codes could be left unused.
- The professional behaviors can be used to help with marketing.

Chapter #27 - The Future

This book is just a guide. It's up to you to build your dream game and then to make it successful. Even after reading this book, even after launching your first game, the learning process can continue. This is the end of the book, but this could be only the beginning of your game development career. These are my parting words to you. It's my advice for the future.

Keep learning - I thought I knew everything there was to know about GameSalad. But as I wrote this book, I found plenty of undiscovered knowledge. Even if you do know everything about GameSalad today, it can change tomorrow. There might be new features in the software. Apple will likely release new gadgets. Keep your mind open and feed it more information. That will help you to stay competitive. Knowledge can take your skills to new levels.

That doesn't just go for GameSalad. This software can introduce you to many different professions. Maybe you'll find something that you like more than GameSalad. Maybe you'll decide to learn Xcode, maybe you'll become a 3D animator or maybe your interest in game creation and computers could lead to something spectacular.

Have fun - The best reason to make GameSalad games is because you're passionate about it. If you're not having fun, then why do it? There are better ways to make money and there are better ways to be famous. GameSalad is a great hobby, but it's incredibly difficult to turn that hobby into a full-time job. Oh sure, you might be putting in the hours of a full-time job, but collecting a full-time salary is tough. There are some GameSalad success stories, so it is possible.

Protect your stuff - Keep a close watch on your games. You might want to search for your games on the Internet. If you see that your games are being stolen, you might want to consider your legal options and/or send take-down notices. As your games get popular, you get the rewards and the perils that come with success.

Track your success - Each game you publish can be a little lesson. It's a good idea to learn what worked. Learn from your mistakes and try not to repeat them. Also, try not to get mad at constructive criticism. Feedback is good. It helps you to learn. While bad feedback is unpleasant, I think that it's usually better than no feedback at all.

Once again, remember to have fun. You're making games!

Thanks for reading,
- Michael Garofalo