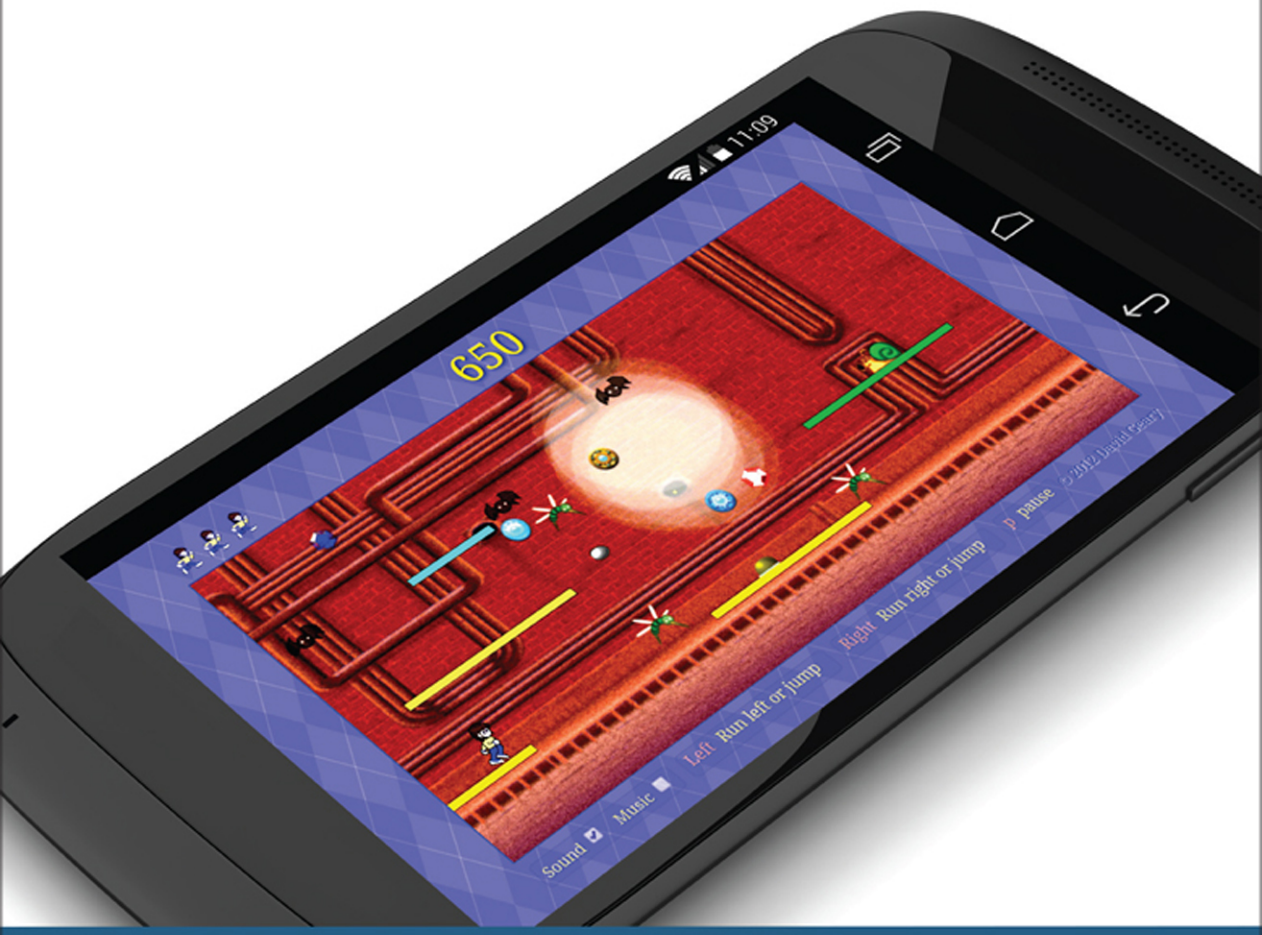
CORE

HTML5

2D Game Programming



David Geary

Core HTML5
2D Game Programming

This page intentionally left blank

Core HTML5 2D Game Programming

David Geary

Pearson

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Library of Congress Cataloging-in-Publication Data

Geary, David M. (David Mark), 1957- author.
Core HTML5 2D game programming / David Geary.
pages cm
Includes index.
ISBN 978-0-13-356424-2 (pbk. : alk. paper) — ISBN 0-13-356424-X (pbk. : alk. paper)
1. HTML (Document markup language) 2. Computer games—Programming. 3. Computer animation. I. Title.
QA76.76.H94G43 2015
006.7'4—dc23

2014014836

ISBN-13: 978-0-13-356424-2
ISBN-10: 0-13-356424-X
Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana.
First printing, July 2014

Contents

Preface

This book is for experienced JavaScript developers who want to implement 2D games with HTML5. In this book, I chronicle the development of a sophisticated side-scroller platform video game, named Snail Bait, from scratch. I do not use any third-party graphics or game frameworks, so that you can learn to implement everything from smooth animations and exploding sprites to developer backdoors and in-game metrics, entirely on your own. If you do use a game framework, this book provides valuable insights into how they work.

Because it's meant for instructional purposes, Snail Bait has only a single level, but in all other respects it's a full-fledged, arcade-style game. Snail Bait simultaneously manipulates dozens of animated objects, known as *sprites*, on top of a scrolling background and simultaneously plays multiple sound effects layered over the game's soundtrack. The sprites run, jump, fly, sparkle, bounce, pace, explode, collide, shoot, land on platforms, and fall through the bottom of the game.

Snail Bait also implements many other features, such as a time system that can slow the game's overall time or speed it up; an animated loading screen; special effects, such as shaking the game when the main character loses a life; and particle systems that simulate smoke and fire. Snail Bait pauses the game when the game's window loses focus; and when the window regains focus, Snail Bait resumes with an animated countdown to give the user time to regain the controls.

Although it doesn't use game or graphics frameworks, Snail Bait uses Node.js and socket.io to send in-game metrics to a server, and to store and retrieve high scores, which the game displays with a heads-up display. Snail Bait shows a warning when the game runs too slowly, and if you type CTRL-d as the game runs, Snail Bait reveals a developer backdoor that gives you special powers, such as modifying the flow of time or displaying sprite collision rectangles, among other things.

Snail Bait detects when it runs on a mobile device and reconfigures itself by installing touch event handlers and resizing the game to fit snugly on the mobile device's screen.

In this book I show you how to implement all of Snail Bait's features step by step, so that you can implement similar features in your own games.

A Brief History of This Book

In 2010, I downloaded the graphics and sound from a popular open source Android game named *Replica Island*, and used them to implement a primitive version of Snail Bait on Android.

At that time, I became interested in HTML5 Canvas and I started working on my previous book, *Core HTML5 Canvas*. As I wrote the Canvas book, I continued to work on Snail Bait, converting it from Android's Java to the browser's JavaScript and the HTML5 `canvas` element. By the time that book was finished in 2012, I had a still primitive, but close to feature-complete, version of the game.

Later in 2012, I started writing a 10-article series for IBM developerWorks on game programming, based on Snail Bait. Over the course of the next ten months, I continued to work on the game as I wrote the articles. (See "Online Resources" below for a link to those articles.)

By summer 2013, Snail Bait had matured a great deal, so I put together a presentation covering Snail Bait's development and traveled to Sebastopol, California to shoot a 15-hour O'Reilly video titled "HTML5 2D Game Development." In some respects that video is the film version of this book. Although the video wasn't released until September, it was one of the top 10 bestselling O'Reilly videos for 2013. (The "Online Resources" below has a link to that video.)

When I returned home from Sebastopol in July 2013, I started writing this book full time. I started with the ten articles from the IBM developerWorks series, rewrote them as book chapters, and ultimately added ten more chapters. As I was writing, I constantly iterated over Snail Bait's code to make it as readable as possible.

In December 2013, with Chapters 1–19 written, I decided to add a final chapter on using the techniques in the book to implement a simpler video game. That game is Bodega's Revenge, and it's the subject of Chapter 20.

How to Use This Book

This book's premise is simple: It shows you how to implement a sophisticated video game so that you can implement one of your own.

There are several ways you can use this book. First, I've gone to great lengths to make it as skim-friendly as possible. The book contains lots of screenshots, code listings, and diagrams.

I make liberal use of Notes, Tips, Cautions, and Best Practices. Encapsulating those topics in callouts streamlines the book's main discussion, and since each Note, Tip, Caution, and Best Practice has a title (excluding callouts with a single line), you can decide at a glance whether those ancillary topics are pertinent to your situation. In general, the book's main discussion shows you how things work, whereas the callouts delve into why things work as they do. If you're in a hurry, you can quickly get to the bottom of how things work by sticking to the main discussion, skimming the callouts to make sure you're not missing anything important.

Chapters 1–19 of the book chronicle the development of Snail Bait, starting with a version of the game that simply displays graphics and ending with a full-featured HTML5 video game. Chapter 20 is the Epilogue, which uses much of what the book covered in the previous 19 chapters to implement a second video game.

If you plan to read the book, as opposed to using it solely as reference, you will most likely want to start reading at either Chapter 1 or Chapter 20. If you start at the beginning, Chapter 20 will be a recap and review of what you learned previously, in addition to providing new insights such as using polar coordinates and rotating coordinate systems.

If you start reading at Chapter 20, perhaps even just skimming the chapter, you can get an idea for what lies behind in the previous 19 chapters. If you start at Chapter 20, don't expect to understand a lot of what you read in that chapter the first time around.

I assume that many readers will want to use this book as a reference, so I've included references to section headings at the start of each chapter, in addition to a short discussion at the beginning of each chapter about what the chapter entails. That will help you locate topics. I've also included many step-by-step instructions on how to implement features so that you can follow those steps to implement similar features of your own.

The Book's Exercises

Passively reading a book won't turn anyone into a game programmer. You've got to get down in the trenches and sling some code to really learn how to implement games. To that end, each chapter in this book concludes with a set of exercises.

To perform the exercises, download the final version of Snail Bait and modify that code. In some cases, the exercises will instruct you to modify code for a

chapter-specific version of the game. See the next section for more information about chapter-specific versions of Snail Bait.

Source Code and Chapter-specific Versions of Snail Bait

This book comes with the source to two video games. See "Online Resources" below for URLs to the games and their source code.

You will undoubtedly find it beneficial to refer to Snail Bait's source code as you read this book. You will find it more beneficial, however, to refer to the version of the game that corresponds to the chapter you are reading. For example, in the first chapter we implement a nascent version of Snail Bait that simply draws the background and the game's main character. That version of the game bears little resemblance to the final version, so referring to the final version of the game is of little use at that point. Instead, you can access the version of Snail Bait corresponding to the end of Chapter 1 at corehtml5games.com/book/code/ch01. URLs for each of the book's chapters follow the format `corehtml5games.com/book/code/ch??`, where *??* represents two digits corresponding to chapter numbers from 01 to 20, excluding Chapter 2.

As mentioned above, exercises at the end of each chapter correspond to the final version of Snail Bait, unless otherwise stated.

Prerequisites

No one would think of taking a creative writing class in a language they couldn't speak or write. Likewise, you must know JavaScript to implement sophisticated games with HTML5. JavaScript is a nonnegotiable prerequisite for this book.

Nearly all the code listings in this book are JavaScript, but you still need to know your way around HTML and CSS. You should also be familiar with computer graphics and have a good grasp of basic mathematics.

Your Game

Finally, let's talk about why we're here. I assume you're reading this book because you want to implement a game of your own.

The chapters of this book discuss individual aspects of game programming, such as implementing sprites or detecting collisions. Although they pertain to Snail Bait, you will be able to easily translate those aspects to your own game.

The order of the chapters, however, is also significant because it shows you how to implement a game from start to finish. In the beginning of the book, we gather raw materials, set up our development environment, and then start development by drawing the game's basic graphics. Subsequent chapters add animation, sprites, sprite behaviors, and so on. If you're starting a game from scratch, you may want to follow that same outline, so you can alternate between reading about features and implementing them on your own.

Before you get started coding in earnest, you should take the time to set up your development environment and become as familiar as you can with the browser's developer tools. You should also make sure you shorten your development cycle as discussed at the end of Chapter 2. The time you initially spend preparing will make you more productive later on.

Finally, thank you for buying this book. I can't wait to see the games you create!

David Geary
Fort Collins, Colorado
2014

Online Resources

Core HTML5 2D Game Programming's companion website: corehtml5games.com

Play Snail Bait: corehtml5games.com/snailbait

Play Bodega's Revenge: corehtml5games.com/bodegas-revenge

Download Snail Bait: corehtml5games.com/book/downloads/snailbait

Download Bodega's Revenge: corehtml5games.com/book/downloads/bodegas-revenge

David's "HTML5 2D Game Development" video from O'Reilly: shop.oreilly.com/product/0636920030737.do.

David's "HTML5 2D Game Development" series on IBM developerWorks: www.ibm.com/developerworks/java/library/j-html5-game1/index.html

A video of David speaking about HTML5 game programming at the Atlanta HTML5 Users Group in 2013: youtube.com/watch?v=S256vAqGY6c

Core HTML5 Canvas at http://amzn.to/1jfuf0C. Take a deep dive into Canvas with David's book.

This page intentionally left blank

Acknowledgments

I am fortunate to have a great editor—the only editor I've had in nearly twenty years of writing books—who is always receptive to my ideas for my next book and who guides my books from conception to completion. This book was no different. Greg Doench helped shepherd this book through the process from an idea to a finished book.

I'm also fortunate to have a wonderful copyeditor, Mary Lou Nohr. She has copyedited every one of my previous books, and she graciously agreed to smooth out my rough edges once again.

This is the second book that I've done with Alina Kirsanova, who's a wizardess at taking my PDFs and making them look super. Once again, Julie Nahil oversaw the production of the book and kept everything on track as we headed to the printer.

For every book I write, I select reviewers who I think will make the book much better than I ever could have alone. For this book, I had four excellent reviewers: Jim O'Hara, Timothy Harrington, Simon Sarris, and Willam Malone. Gintas Sanders also gave me permission to use his coins in Snail Bait and gave me some great critiques of the game.

When I shot the "HTML5 2D Game Development" video for O'Reilly, I taught a class in front of a live audience. One of the audience members asked great questions and came up with several insights. Jim O'Hara was one of my most conscientious reviewers and, as he did in class, provided lots of great questions and insights.

My editor, Greg Doench, put me in touch with Tim Harrington, who is a Senior Academic Applications Analyst at Devry University with a background in game development. Like Jim, Tim came up with lots of insights that made me rethink how I presented material.

I wanted to find a graphics expert for this book who knew a lot about game programming, and I found one. Simon Sarris, who, much to my delight, is not only both of those things, but is also an excellent writer. He made this book better in several different ways.

Finally, I was fortunate to have William Malone review this book. William is a professional game developer who's implemented games for *Sesame Street* (see Cookie Kart Racing at http://bit.ly/1nlSY3N). William made a tremendous difference in this book by pointing out many subtleties that would've escaped me, especially concerning mobile devices.

About the Author

David is the author of *Core HTML5 Canvas* and coauthor of *Core JavaServer Faces*. David has written several other bestselling books on client- and server-side Java, including one of the bestselling Java books of all time, *Graphic Java*.

This page intentionally left blank

CHAPTER 1

Introduction

Topics in This Chapter

The great thing about software development is that you can make nearly anything you can imagine come to life on screen. Unencumbered by physical constraints that hamper engineers in other disciplines, software developers have long used graphics APIs and UI toolkits to implement creative and compelling applications. Arguably, the most creative genre of software development is game programming; few endeavors are more rewarding from a creative standpoint than making the vision you have for a game become a reality.

The great thing about game programming is that it's never been more accessible. With the advent of open source graphics, sound, and music, you no longer need to be an artist and a musician to implement games. And the development environments built into modern browsers are not only free, they contain all the tools you need to create the most sophisticated games. You need only supply

programming prowess, a good understanding of basic math (mostly trigonometry), and a little physics.

In this book we implement two full-fledged HTML5 video games so that you can learn how to create one of your own. Here are some of the things you will learn to do:

- Use the browser's development tools to implement sophisticated games
- Create smooth, flicker-free animations
- Scroll backgrounds and use parallax to create a 3D effect
- Implement graphical objects, known as *sprites*, that you can draw and manipulate in a canvas
- Detect collisions between sprites
- Animate sprites to make them explode
- Implement a time system that controls the rate at which time flows through your game
- Use nonlinear motion to create realistic jumping
- Simulate gravity
- Pause and freeze your game
- Warn players when your game runs slowly
- Display scoreboards, controls, and high scores
- Create a developer's backdoor with special features
- Implement particle systems to simulate natural phenomenon, such as smoke and fire
- Store high scores and in-game metrics on a server with Node.js and socket.io
- Configure games to run on mobile devices

NOTE: HTML5 technologies used in Snail Bait

This book discusses the implementation of an HTML5 video game, named Snail Bait, using the following HTML5 APIs, the most predominant of which is the Canvas 2D API:

- Canvas 2D API
- Timing Control for Script-based Animations
- Audio
- CSS3 Transitions

In this book we develop Snail Bait entirely from scratch, without any third-party game frameworks, so you can learn how to implement all the common aspects of a video game from the ground up. That knowledge will be invaluable whether you implement a game by using a framework or not.

The book's epilogue discusses the implementation of a second video game—Bodega's Revenge—that shows how to combine the concepts discussed in the book to implement a simpler video game.

NOTE: Play Snail Bait and Bodega's Revenge online

To get the most out of this book, you should play Snail Bait and Bodega's Revenge so you're familiar with the games. You can play Snail Bait online at corehtml5games.com/snailbait, and you can find Bodega's Revenge at corehtml5games.com/bodegas-revenge.

NOTE: Particle systems

A particle system uses many small particles that combine to simulate natural phenomena that do not have well-defined boundaries and edges. Snail Bait implements a particle system to simulate smoke, as you can see in Figure 1.1. We discuss particle systems in detail in Chapter 16.

1.1 Snail Bait

Snail Bait is a classic platform game. The game's main character, known as the runner, runs along and jumps between floating platforms that move horizontally. The runner's ultimate goal is to land on a gold button that paces back and forth on top of a pulsating platform at the end of the game. That button is guarded by two bees and a bomb-shooting snail. The runner, pulsating platform, gold button, bees, bomb, and snail are all shown in Figure 1.1.

The player controls the game with the keyboard:

- *d* or ← turns the runner to the left and scrolls the background from left to right.
- *k* or → turns the runner to the right and scrolls the background from right to left.
- *j* makes the runner jump.
- *p* pauses the game.

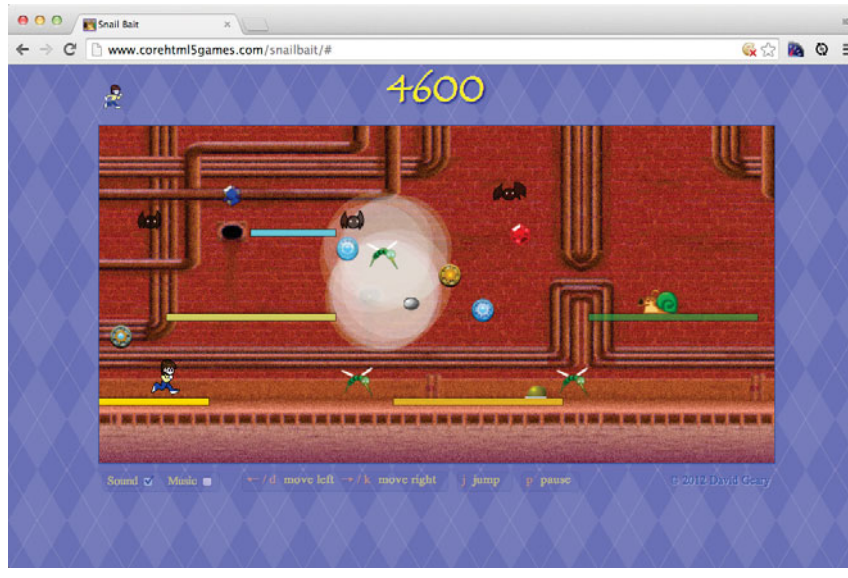


Figure 1.1 Snail Bait

When the game begins, the player has three lives. Icons representing the number of remaining lives are displayed above and to the left of the game's canvas, as you can see in Figure 1.1. In the runner's quest to make it to the end of the level, she must avoid bad guys—bees and bats—while trying to capture valuable items such as coins, rubies, and sapphires. If the runner collides with bad guys, she blows up, the player loses a life, and the runner goes back to the beginning of the level. When she collides with valuable items, the valuable item disappears, the score increases, and the game plays a pleasant sound effect.

The snail periodically shoots snail bombs (the gray ball shown near the center of Figure 1.1). The bombs, like bees and bats, blow up the runner when they hit her.

The game ends in one of two ways: the player loses all three lives, or the player lands on the gold button. If the player lands on the gold button, the player wins the game and Snail Bait shows the animation depicted in Figure 1.2.

Snail Bait maintains high scores on a server. If the player beats the existing high score, Snail Bait lets the player enter their name with a heads-up display (HUD), as shown in Figure 1.3.

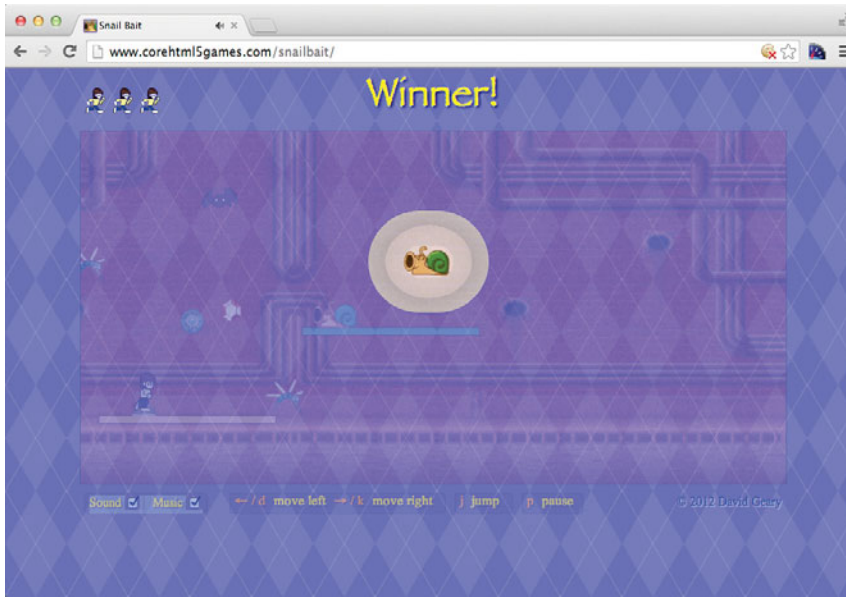


Figure 1.2 Snail Bait's winning animation

Figure 1.3 Snail Bait's high scores

If the player doesn't win the game or beat the existing high score, Snail Bait displays game credits, as shown in Figure 1.4.

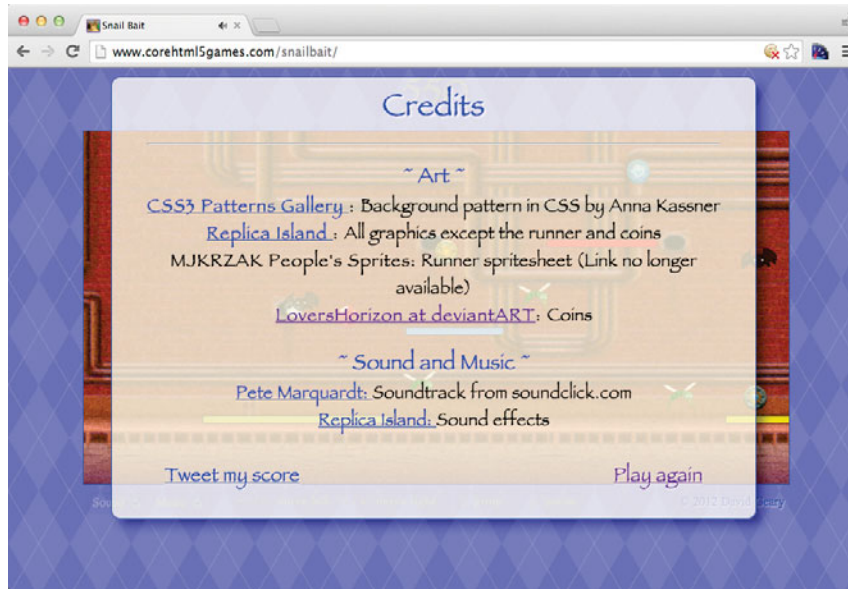


Figure 1.4 Snail Bait's credits

With the exception of the runner, everything in Snail Bait scrolls continuously in the horizontal direction. That scrolling further categorizes Snail Bait as a *side-scroller* platform game. However, that's not the only motion in the game, which leads us to sprites and their behaviors.

NOTE: Platform video games

Donkey Kong, Mario Bros., Sonic the Hedgehog, and Braid are all well-known, best-selling games where players navigate 2D platforms, a genre known as platformers. At one time, platformers represented up to one-third of all video game sales. Today, their market share is drastically lower, but there are still many successful platform games.

CAUTION: Snail Bait performance

Hardware acceleration for Canvas makes a huge difference in performance and has been implemented by most browsers since the middle of 2012. Should you run Snail Bait in a browser that does not have hardware-accelerated Canvas, performance will be terrible and the game probably won't work correctly. When you play the game, make sure your browser has hardware-accelerated Canvas. Here is a list of browser versions that have hardware-accelerated Canvas:

- Chrome 13
- Firefox 4
- Internet Explorer 9
- Opera 11
- Safari 5

WASD?

By convention, computer games often use the w, a, s, and d keys to control play. That convention evolved primarily because it lets right-handed players use the mouse and keyboard simultaneously. It also leaves the right hand free to press the spacebar or modifier keys such as CTRL or ALT. Snail Bait doesn't use WASD because it doesn't receive input from the mouse or modifier keys. But you can easily modify the game's code to use any combination of keys.

1.1.1 Sprites: The Cast of Characters

With the exception of the background, everything in Snail Bait is a sprite. A sprite is a visual representation of an object in a game that you draw on the game's canvas. Sprites are not a part of the HTML5 Canvas API, but they are simple to implement. Following are the game's sprites:

- Platforms (inanimate objects)
- Runner (main character)
- Buttons (good)
- Coins (good)

- Rubies and sapphires (good)
- Bees and bats (bad)
- Snail (bad)
- Snail bombs (bad)

Besides scrolling horizontally, nearly all the game's sprites move independently of one another. For example, rubies and sapphires bounce up and down at varying rates of speed, and the buttons and the snail pace back and forth along the length of the platform on which they reside.

That independent motion is one of many sprite behaviors. Sprites can have other behaviors that have nothing to do with motion; for example, besides bouncing up and down, the rubies and sapphires sparkle.

Each sprite has an array of behaviors. A behavior is just a JavaScript object with an `execute()` method. Every animation frame, the game iterates over all its visible sprites and, for each sprite, iterates over the sprite's behaviors, invoking each behavior's `execute()` method and passing the method a reference to the sprite in question. In that method, behaviors manipulate their associated sprite according to game conditions. For example, when you press j to make the runner jump, the runner's jump behavior subsequently moves the runner through the jump sequence, one animation frame at a time.

Table 1.1 lists the game's sprites and their respective behaviors.

Table 1.1 Snail Bait sprites

Sprites	Behaviors
Platforms	Pulsate (only one platform)
Runner	Run; jump; fall; collide with other sprites; explode
Bees and bats	Explode; flap their wings
Buttons	Pace; collapse; make bad guys explode
Coins, rubies, and sapphires	Sparkle; bounce up and down
Snail	Pace; shoot bombs
Snail bombs	Move from right to left; collide with runner

Behaviors are simple JavaScript objects, as illustrated by Example 1.1, which shows how Snail Bait instantiates the runner sprite.

Example 1.1 Creating sprites

```
runBehavior = {  // Just a JavaScript object with an execute method

   execute: function (sprite, // Sprite associated with the behavior
                      now,    // The current game time
                      fps,    // The current frame rate
                      context, // The context for the game's canvas
                      lastAnimationFrameTime) { // Time of last frame

      // Update the sprite's attributes, based on the current time
      // (now), frame rate (fps), and the time at which Snail Bait
      // drew the last animation frame (lastAnimationFrameTime),
      // to make it look like the runner is running.

      // The canvas context is provided as a convenience for things
      // like hit detection, but it should not be used for drawing
      // because that's the responsibility of the sprite's artist.

      // Method implementation omitted. See Section 7.3 on p. 187
      // for a discussion of this behavior.
   }

};

var runner = new Sprite('runner',              // name
                        runnerArtist,          // artist
                        [ runBehavior, ... ]); // behaviors
```

Snail Bait defines a `runBehavior` object, which it passes—in an array with other behaviors—to the runner sprite's constructor, along with the sprite's type (`runner`) and its artist (`runnerArtist`). For every animation frame in which the runner is visible, the game invokes the `runBehavior` object's `execute()` method. That `execute()` method makes it appear as though the runner is running by advancing through the set of images that depict the runner in various run poses.

NOTE: Replica Island

The idea for sprite behaviors, which are an example of the Strategy design pattern, comes from Replica Island, a popular open source (Apache 2 license) Android platform game. Additionally, most of Snail Bait's graphics are from Replica Island. You can find out more about Replica Island at replicaisland.net, and you can read about the Strategy design pattern at http://en.wikipedia.org/wiki/Strategy_design_pattern.

NOTE: Sprite artists

Besides encapsulating behaviors in separate objects—which makes it easy to add and remove behaviors at runtime—sprites also delegate how they are drawn to another JavaScript object, known as a sprite artist. That makes it possible to plug in a different artist at runtime.

NOTE: Freely available resources

Most game developers need help with graphics, sound effects, and music. Fortunately, an abundance of assets are freely available under various licensing arrangements. Snail Bait uses the following:

- Graphics and sound effects from Replica Island
- Soundtrack from soundclick.com
- Coins from LoversHorizon at deviantART

See Chapter 2 for more information on obtaining game resources and setting up a development environment.

1.2 HTML5 Game Development Best Practices

We discuss game development best practices throughout this book, starting here with seven that are specific to HTML5.

1. Pause the game when the window loses focus.
2. Implement a countdown when the window regains focus.
3. Use CSS for user interface (UI) effects.
4. Detect and react to slowly running games.
5. Incorporate social features.
6. Put all the game's images in a single sprite sheet.
7. Store high scores and realtime in-game metrics on a server.

We examine the preceding best practices in detail later in the book; for now, a quick look at each of them introduces more of Snail Bait's features.

1.2.1 Pause the Game When the Window Loses Focus

If an HTML5 game is running in a browser and you change focus to another tab or browser window, most browsers severely clamp the frame rate at which the

game's animation runs so as to save resources such as CPU and battery power; after all, why waste resources on a window or tab that's not visible?

Frame-rate clamping wreaks havoc with most collision detection algorithms because those algorithms check for collisions every time the game draws an animation frame; if it takes too long between animation frames, sprites can move past one another without detection. To avoid collision detection meltdowns resulting from frame-rate clamping, *you must automatically pause the game when the window loses focus*.

When Snail Bait pauses the game, it displays a toast to let the player know the game is paused, as shown in Figure 1.5.

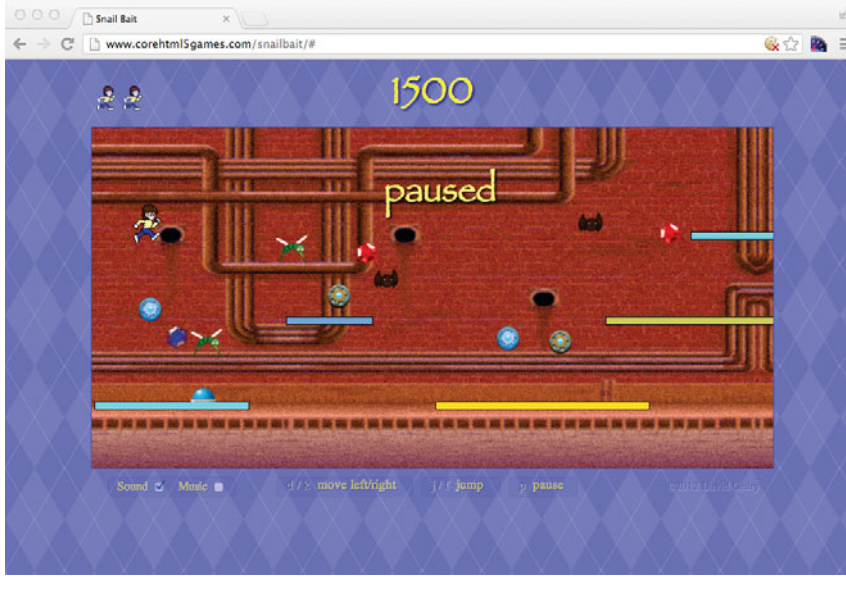


Figure 1.5 Snail Bait paused

NOTE: Pausing is more than stopping the game

When a paused game resumes, everything must be in exactly the same state as it was when the game was paused; for example, in Figure 1.5, when play resumes, the runner must continue her jump from exactly where she was when the game was paused.

In addition to pausing and unpausing the game, therefore, you must also freeze and thaw the game to ensure a smooth transition when the game resumes. We discuss pausing and freezing the game in more detail in Chapter 4.

NOTE: Toasts

A toast—as in raising a glass to one's health—is information that a game displays to a player for a short time. A toast can be simple text, as in Figure 1.5, or it can represent a more traditional dialog box, as in Figure 1.8 on p. 14.

1.2.2 Implement a Countdown When the Window Regains Focus

When your window regains focus, you should give the player a few seconds to prepare for the game to restart. Snail Bait uses a three-second countdown when the window regains focus, as shown in Figure 1.6.

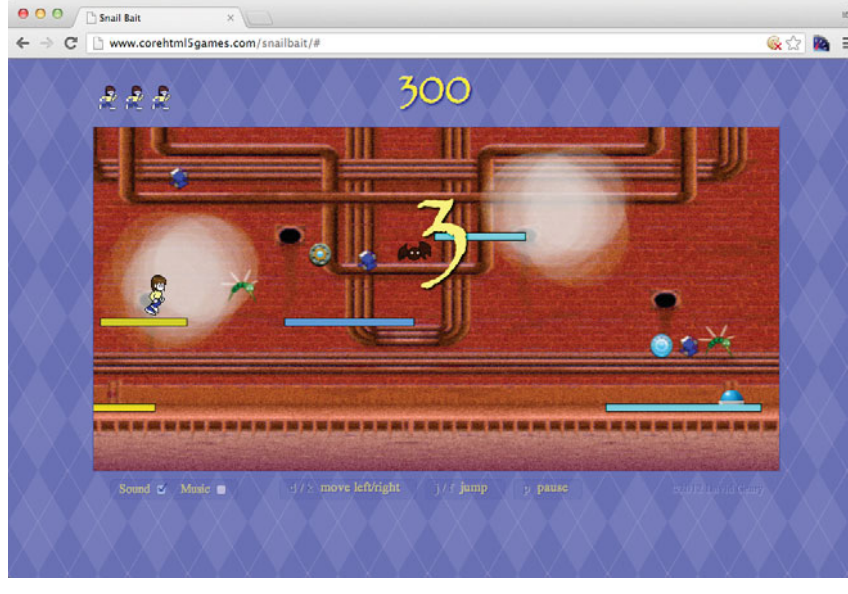


Figure 1.6 Snail Bait's countdown after the window regains focus

1.2.3 Use CSS for UI Effects

Figure 1.7 shows a screenshot taken a short time after the game loads.

Note especially two things about Figure 1.7. First, a toast containing simple instructions is visible. That toast fades in when the game loads, and after five seconds, it fades out.

Second, when the game starts, the checkboxes (for sound and music) and instructions (telling which keystrokes perform which functions) below the game's canvas

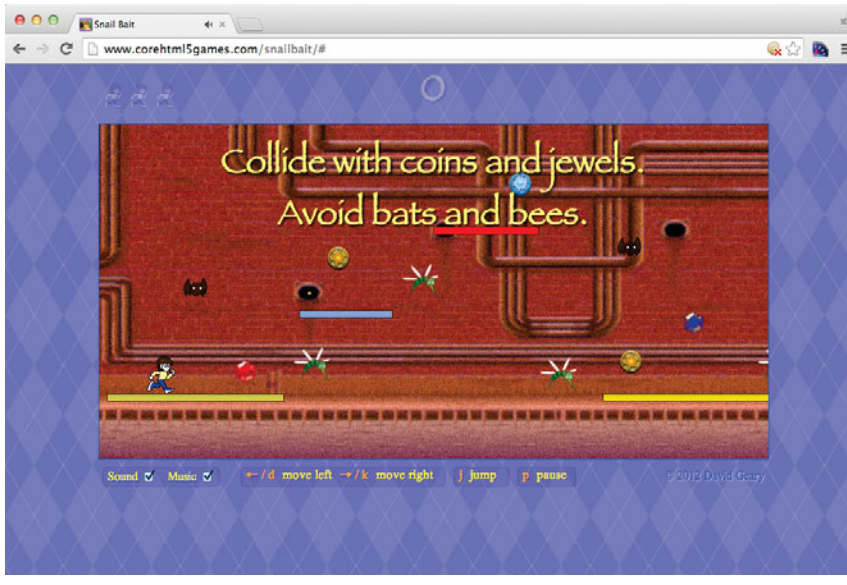


Figure 1.7 Snail Bait's toasts

are fully opaque, whereas the lives indicators and scoreboard at the top of the game are partially transparent, as shown in Figure 1.7. As the game's instructions toast fades, that transparency reverses; the lives indicator and scoreboard become fully opaque, while the checkboxes and instructions become nearly transparent, as they are in Figure 1.6.

Snail Bait dims elements and fades toasts with CSS3 transitions.

NOTE: Focus on what's currently important

When Snail Bait starts, the instructions below the game's canvas are fully opaque, whereas the lives indicator and score above the game's canvas are partially transparent. Shortly thereafter, they switch opacities; the elements above the canvas become fully opaque and the elements below become partially transparent.

Snail Bait goes to all that trouble to focus attention on what's currently important. Initially, players should pay attention to the instructions below the game's canvas; once the game is underway, players will be more focused on their score and how many lives are remaining.

1.2.4 Detect and React to Slowly Running Games

Unlike console games, which run in a tightly controlled environment, HTML5 games run in a highly variable, unpredictable, and chaotic one. Players can do things directly that significantly affect system performance, for example, running YouTube videos in another browser tab or window. Other performance killers, such as system backup software running in the background unbeknown to game players, can easily make an HTML5 game run so slowly that it becomes unplayable. And there's always the possibility that your players will use a browser that can't keep up.

As an HTML5 game developer, you must monitor frame rate and react when it dips below an unplayable threshold. When Snail Bait detects that an average of the last 10 frame rates falls below 40 frames per second (fps), it displays the running slowly toast shown in Figure 1.8.

Figure 1.8 Snail Bait's running slowly toast

1.2.5 Incorporate Social Features

Many modern games incorporate social aspects, such as posting scores on Twitter or Facebook. When a Snail Bait player clicks on the Tweet my score link that appears at the end of the game (see Figure 1.4 on p. 6), Snail Bait creates a tweet announcing the score in a separate browser tab, as shown in Figure 1.9.

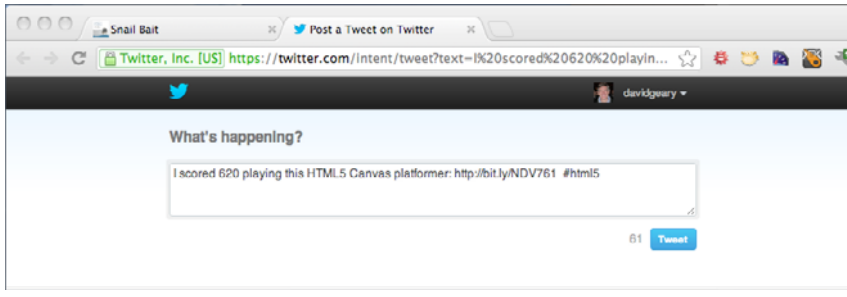


Figure 1.9 Snail Bait's Twitter integration

1.2.6 Put All the Game's Images in a Single Sprite Sheet

You can do several things to make your HTML5 game (or any HTML5 application) load more quickly, but the single most effective thing is to decrease the number of HTTP requests you make to the server. One way to do that is to put all your game's images in a single image, known as a sprite sheet. Figure 1.10 shows Snail Bait's sprite sheet.

Figure 1.10 Snail Bait's sprite sheet (the gray background is transparent)

When Snail Bait draws the game's sprites, it copies rectangles from the sprite sheet into the canvas.

NOTE: Sprite sheets on mobile devices

Some mobile devices place limits on the size of image files, so if your sprite sheet is too large, you may have to split it into multiple files. Your game will load more slowly as a result, but that's better than not loading at all.

1.2.7 Store High Scores and Send Realtime, In-game Metrics to the Server

Most games interact with a server for a variety of reasons. Snail Bait stores high scores on a server in addition to sending game metrics during gameplay. Snail Bait does not use any third-party graphics frameworks; however, it does use two JavaScript frameworks—Node.js and socket.io—to communicate between the player's computer and a server. See Chapter 19 for more details.

1.3 Special Features

Snail Bait has three noteworthy features that add polish to the game and make playtesting more productive:

- Developer backdoor
- Time system
- Particle systems

Snail Bait reveals the developer backdoor, shown in **Figure 1.11**, when you press CTRL-d. With the backdoor visible, you can control the rate at which time flows through the game, making it easy to run the game in slow motion to see how game events such as collision detection take place. Conversely, you can run the game faster than normal to determine the best pace for the game.

You can turn collision rectangles on for a better look at exactly how collisions occur; if the smoking holes obscure your view, you can turn the smoke off by deselecting the Smoke checkbox. You can also fine-tune the threshold at which Snail Bait displays the game's running slowly warning, shown in **Figure 1.8**, or you can turn it off entirely, which lets you playtest slow frame rates without Snail Bait intervening at all.

When you playtest a particular section of the game, you can avoid playing through the preceding sections every time you test: In addition to the controls at the top of the game's canvas, the developer backdoor displays a ruler at the bottom of the canvas that shows how far the background has scrolled horizontally in pixels.

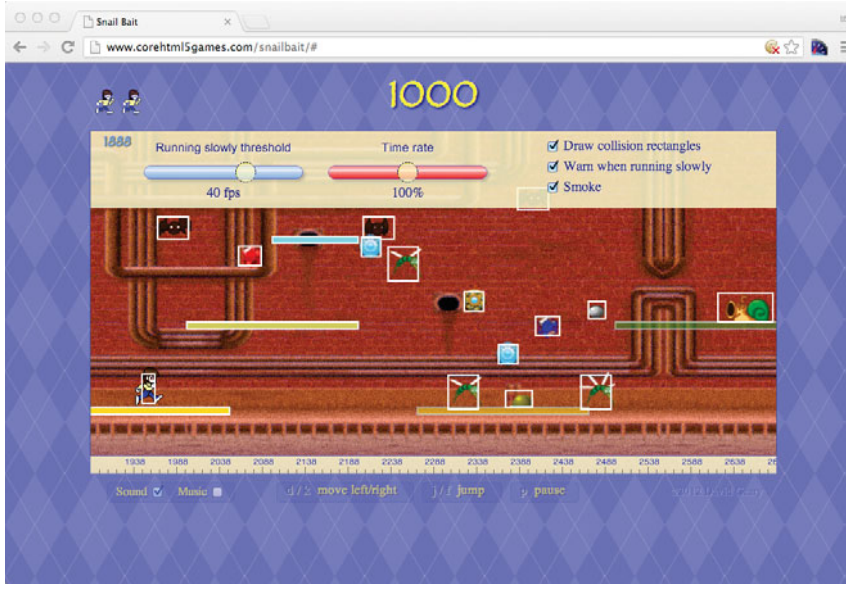


Figure 1.11 Snail Bait's developer backdoor

You use those values to restart the game at a particular horizontal location, thereby avoiding the preceding sections of the game. For convenience, when the developer backdoor is visible you can also simply drag the game, including the background and all the sprites, horizontally to reposition the runner.

The developer backdoor lets you control the rate at which time flows through the game by virtue of Snail Bait's time system. Everything that happens in Snail Bait depends on the current game time, which is the elapsed time since the game started; for example, when the runner begins a jump, the game records the current game time, and thereafter moves the runner through the jump sequence frame by frame, depending on how much time has elapsed since the runner began the jump.

By representing the current game time as the real time, which is Snail Bait's default mode, the game runs at its intended rate. However, Snail Bait's time system can *mis*represent the current game time as something other than the real time; for example, the time system can consistently report that the current game time is half of the actual time, causing the game to run at half speed.

Besides letting you control the rate at which time flows through the game, Snail Bait's time system is also the source of special effects. When the runner collides with a bad guy and explodes, Snail Bait slows time to a crawl while transitioning

to the next life. Once the transition is complete, Snail Bait returns time to normal, indicating that it's time to resume play.

Finally, Snail Bait uses two particle systems to create the illusion of smoke and fire in the background. In Chapter 16, we take a close look at those particle systems so you can create similar effects of your own.

Now that you have a high-level understanding of the game, let's take a look at some code.

NOTE: Snail Bait's code statistics (lines of code)

- JavaScript: 5,230
- CSS: 690
- HTML: 350

NOTE: A closer look at Snail Bait's code

- snailbait.js: 3,740
- Supporting JavaScript code: 1,500
- Initializing data for sprites: 500
- Creating sprites: 400
- Sprite behavior implementations: 730
- Event handling: 300
- User interface: 225
- Sound: 130

1.4 Snail Bait's HTML and CSS

Snail Bait is implemented with HTML, CSS, and JavaScript, the majority of which is JavaScript. In fact, the rest of this book is primarily concerned with JavaScript, with only occasional forays into HTML and CSS.

Figure 1.12 shows the HTML elements, outlined in white, and their corresponding CSS for the top half of the game proper.

Everything in Snail Bait takes place in the arena, which is an HTML DIV element. The arena's `margin` attribute is `0, auto`, which means the browser centers the arena and everything inside it horizontally, as shown in Figure 1.13.

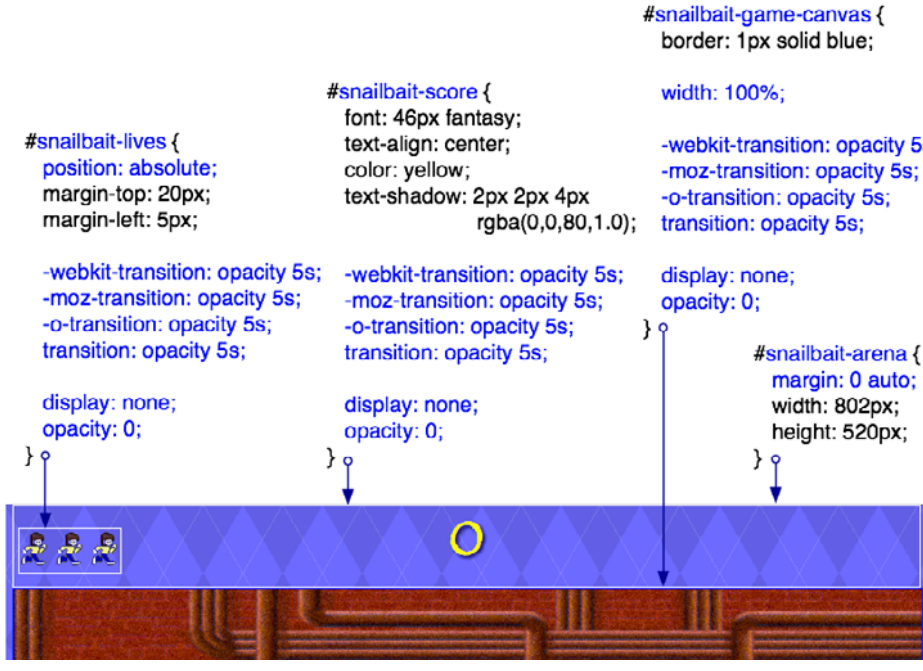


Figure 1.12 Snail Bait's CSS for the top half of the game

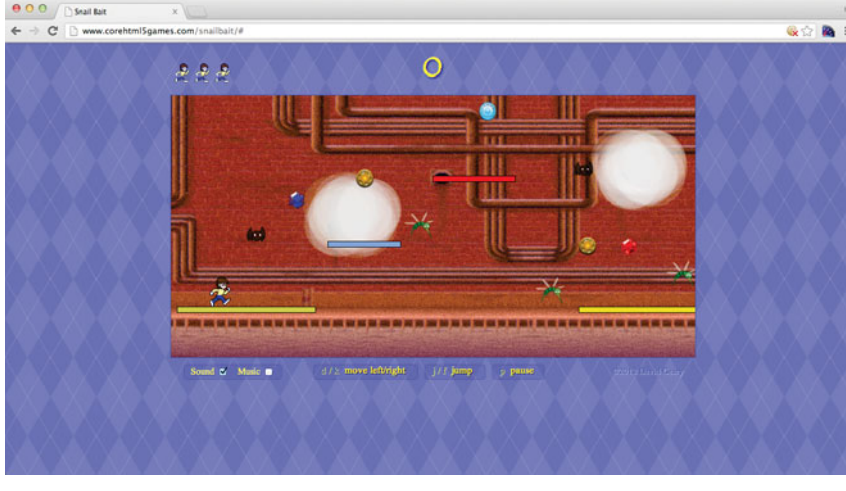


Figure 1.13 Snail Bait stays centered horizontally in the window

When Snail Bait loads resources, it displays the animation shown in Figure 1.14. During that animation, none of the game's elements are visible, which is why all the elements in Figure 1.12 have their `display` attribute set to `none` (with the exception of `snailbait-arena`, which has no visible characteristics of its own).

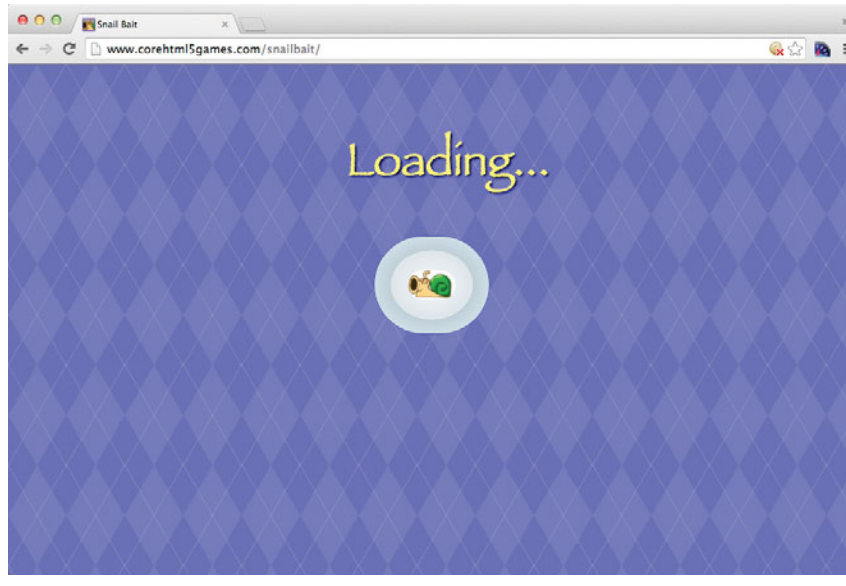


Figure 1.14 Snail Bait at startup

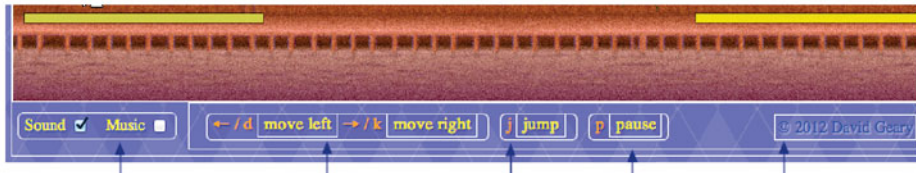
After the game loads resources, it fades in the game's elements by setting their `display` attribute to `block` and subsequently setting their `opacity` to `1.0` (fully opaque). Elements that have a transition associated with their `opacity` property, like `snailbait-lives`, `snailbait-score`, and `snailbait-game-canvas`, transition into view over a specified period of time.

The `snailbait-lives` element has an `absolute` position; otherwise, with its default position of `static`, it will expand to fit the width of its enclosing DIV, forcing the score beneath it.

The game canvas, which is an HTML5 `canvas` element, is where all the game's action takes place; it's the only element in Figure 1.12 that's not a DIV.

Figure 1.15 shows the HTML elements in the lower half of the game.

Like the lives and score elements in the upper half of the game, the browser does not display the elements at the bottom during the game's loading animation, so those elements are initially invisible and have an opacity transition of five seconds,




```
#snailbait-sound-and-music {
  position: absolute;
  margin-left: 5px;
  margin-top: 9px;
  padding: 5px;
  padding-top: 2px;
  padding-bottom: 2px;
  color: yellow;
  font-size: 0.9em;
  text-shadow: 1px 1px 1px
    rgba(0,0,0,0.5);
  background: rgba(0,0,0,0.1);
  border-radius: 5px;
  border: thin solid
    rgba(0,0,0,0.20);

  -webkit-transition: opacity 5s;
  -moz-transition: opacity 5s;
  -o-transition: opacity 5s;
  transition: opacity 5s;

  display: none;
  opacity: 0;
}
```

```
.snailbait-keys {
  color: orange;
  text-shadow: 1px 1px 1px
    rgba(0,0,0,0.5);
  background: rgba(0,0,0,0.1);
  border: thin solid
    rgba(0,0,0,0.20);
  border-radius: 5px;
  margin-left: 10px;
  padding-right: 5px;
  padding-left: 5px;
  padding-bottom: 5px;

  display: inline;
}
```

```
#snailbait-copyright {
  float: right;
  margin-right: 2px;
  margin-top: -2.3em;
  padding: 2px;
  color: blue;
  text-shadow: 1px 1px 1px
    rgba(255,255,255,0.5);
  font-size: 0.85em;

  -webkit-transition: opacity 5s;
  -moz-transition: opacity 5s;
  -o-transition: opacity 5s;
  transition: opacity 5s;

  display: none;
  opacity: 0;
}
```

```
.snailbait-explanation {
  color: yellow;
  text-shadow: 1px 1px 1px
    rgba(0,0,0,0.5);
  margin-top: 5px;
  padding-right: 5px;
  padding-left: 5px;
  padding-bottom: 2px;

  display: inline;
}
```

```
#snailbait-instructions {
  margin-left: 155px;
  padding-left: 5px;
  padding-top: 10px;
  height: 2em;

  color: yellow;
  font-size: 0.95em;

  -webkit-transition: opacity 5s;
  -moz-transition: opacity 5s;
  -o-transition: opacity 5s;
  transition: opacity 5s;

  display: none;
  opacity: 0;
}
```

Figure 1.15 Snail Bait's CSS for the bottom of the game

which Snail Bait uses to fade them and all their contained elements in along with the score and lives elements at the beginning of the game.

The `snailbait-sound-and-music` element, like the `snailbait-lives` element, has an `absolute` position to prevent its width from expanding. The `snailbait-keys` and `snailbait-explanation` DIVs have `display` attributes of `inline` so they appear horizontally inline with the other elements in their enclosing DIV, instead of being stacked vertically.

Example 1.2 lists Snail Bait's HTML proper, omitting a considerable amount of HTML for things like the running slowly warning and developer backdoor.

Example 1.2 index.html (excerpt)

```
<!DOCTYPE html>

   <!--
       Basic HTML elements for Snail Bait. Elements for things such
       as sounds, credits, toasts, developer backdoor, etc. are
       omitted for brevity.
   -->

<html>
   <!-- Head.....................................................-->

   <head>
      <title>Snail Bait</title>
      ...

     <link rel='stylesheet' href='snailbait.css'>
   </head>

   <!-- Body.....................................................-->

   <body>
      <!-- Arena.....................................................-->

      <div id='snailbait-arena'>
         ...

         <!-- Lives indicator.....................................-->

         <div id='snailbait-lives'>
            <img id='snailbait-life-icon-left'
                 src='images/runner-small.png'/>

            <img id='snailbait-life-icon-middle'
                 src='images/runner-small.png'/>

            <img id='snailbait-life-icon-right'
                 src='images/runner-small.png'/>
         </div>
```

```
<!-- Score ..................................................-->

<div id='snailbait-score'>0</div>
...

<!-- The game canvas.........................................-->

<canvas id='snailbait-game-canvas' width='800' height='400'>
   Your browser does not support HTML5 Canvas.
</canvas>
...

<!-- Sound and music.........................................-->

<div id='snailbait-sound-and-music'>
   <div id='snailbait-sound-checkbox-div'
     class='snailbait-checkbox-div'>

      Sound <input id='snailbait-sound-checkbox'
                type='checkbox' checked/>

   </div>

   <div class='snailbait-checkbox-div'>
      Music <input id='snailbait-music-checkbox'
                type='checkbox' checked/>
   </div>
</div>

<!-- Instructions............................................-->

<div id='snailbait-instructions'>
   <div class='snailbait-keys'>
      &larr; / d
      <div class='snailbait-explanation'>move left</div>
      &rarr; / k
      <div class='snailbait-explanation'>move right</div>
   </div>

   <div class='snailbait-keys'>
      j <div class='snailbait-explanation'>jump</div>
   </div>

   <div class='snailbait-keys'>
      p <div class='snailbait-explanation'>pause</div>
   </div>
</div>

<div id='snailbait-mobile-instructions'>
```

(Continues)

Example 1.2 *(Continued)*

```
            <div class='snailbait-keys'>
               Left
               <div class='snailbait-explanation'>
                  Run left or jump
               </div>
            </div>

            <div class='snailbait-keys'>
               Right
               <div class='snailbait-explanation'>
                  Run right or jump
               </div>
            </div>
         </div>

         <!-- Copyright.....................................................-->

         <div id='snailbait-copyright'> &copy; 2012 David Geary</div>
      </div>

      <!-- JavaScript.....................................................-->

      <!-- Other script tags for the game's other JavaScript files are
           omitted for brevity. The final version of the game puts all
           the game's JavaScript into a single file. See Chapter 19
           for more details about how Snail Bait is deployed. -->

      <script src='snailbait.js'></script>
   </body>
</html>
```

The canvas element is where all the action takes place. The canvas comes with a 2D context with a powerful API for implementing 2D games, among other things, as you will see in Section 3.1, "Draw Graphics and Images with the HTML5 canvas Element," on p. 64. The text inside the canvas element is fallback text that the browser displays only if it does not support HTML5 canvas element.

One final note about the game's HTML and CSS: Notice that the width and height of the canvas is set with canvas element attributes in the preceding listing. Those attributes *pertain to both the size of the* `canvas` *element and the size of the drawing surface contained within that element*.

On the other hand, using CSS to set the width and height of the canvas element *sets only the size of the element*. The drawing surface remains at its default width and height of 300 × 150 pixels, respectively. That means you will have a mismatch between the canvas element size and the size of its drawing surface when you

set the element's size to something other than the default 300 × 150 pixels, and in that case *the browser scales the drawing surface to fit the element*. Most of the time that effect is unwanted, so it's a good idea to set the size of the `canvas` element with its `width` and `height` attributes, and not with CSS.

At this point, you've already seen the end of the Snail Bait story. Now let's go back to the beginning.

Draw into a small canvas and let CSS scale it?

Some games purposely draw into a small canvas and use CSS to scale the canvas to a playable size. That way, the canvas is not manipulating as many pixels, and so increases performance. You will take a performance hit for scaling the canvas, of course, but scaling with CSS is typically hardware accelerated, so the cost of the scaling can be minimal. Today, however, nearly all the latest versions of modern browsers come equipped with hardware-accelerated Canvas, so it's just as fast to draw into a full-sized canvas in the first place.

NOTE: Namespacing HTML elements and CSS classes

To avoid naming collisions with other HTML elements, Snail Bait starts each HTML element and CSS classname with `snailbait-`.

1.5 Snail Bait's Humble Beginning

Figure 1.16 shows Snail Bait's initial set of files. Throughout this book we add many more files, but for now all we need is an HTML file to define the structure of the game's HTML elements, a CSS file to define the visual properties for those elements, a JavaScript file for the game's logic, and two images, one for the background and another for the runner.

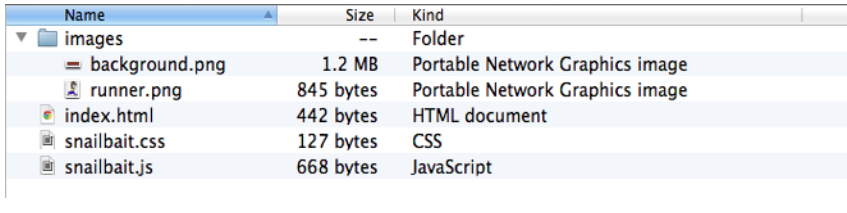


Figure 1.16 Snail Bait's initial files

Figure 1.17 shows the starting point for the game, which simply draws the background and the runner. To start, the runner is not a sprite; instead, the game draws her directly.

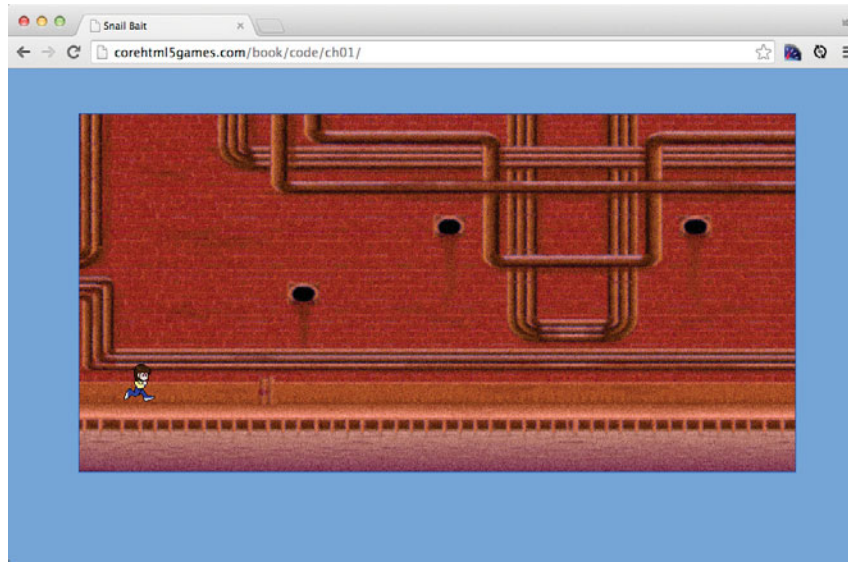


Figure 1.17 Drawing the background and runner

Example 1.3 lists the starting point for the game's HTML, which is just a distilled version of the HTML in Example 1.2.

Example 1.3 The starting point for Snail Bait's HTML

```
<!DOCTYPE html>
<html>
   <head>
      <title>Snail Bait</title>
      <link rel='stylesheet' href='snailbait.css'/>
   </head>

   <body>
      <div id='snailbait-arena'>
         <canvas id='snailbait-game-canvas' width='800' height='400'>
            Your browser does not support HTML5 Canvas.
         </canvas>
      </div>
```

```
      <!-- JavaScript.....................................................-->

      <script src='snailbait.js'></script>
   </body>
</html>
```

Initially, the arena contains only the game's canvas, which is 800 pixels wide by 400 pixels high and has a thin blue border. **Example 1.4** shows the starting point for Snail Bait's CSS.

Example 1.4 The starting point for Snail Bait's CSS

```
body {
   background: cornflowerblue;
}

#snailbait-arena {
   margin: 0 auto;
   margin-top: 50px;
   width: 800px;
   height: 400px;
}

#snailbait-game-canvas {
   border: 1.5px solid blue;
}
```

Example 1.5 shows the starting point for Snail Bait's JavaScript.

Example 1.5 The starting point for Snail Bait's JavaScript

```
var canvas = document.getElementById('snailbait-game-canvas'),
   context = canvas.getContext('2d'),

   background  = new Image(),
   runnerImage = new Image();

function initializeImages() {
   background.src = 'images/background.png';
   runnerImage.src = 'images/runner.png';

   background.onload = function (e) {
      startGame();
   };
}
```

(Continues)

Example 1.5 *(Continued)*

```
function startGame() {
   draw();
}

function draw() {
   drawBackground();
   drawRunner();
}

function drawBackground() {
   context.drawImage(background, 0, 0);
}

function drawRunner() {
   context.drawImage(runnerImage, 50, 280);
}

// Launch game.....................................................

initializeImages();
```

The preceding JavaScript accesses the `canvas` element and subsequently obtains a reference to the canvas's 2D context. The code then draws the background and runner by using the three-argument variant of `drawImage()` to draw images at a particular location in the canvas.

The game starts when the background image loads. For now, starting the game entails simply drawing the background and the runner.

1.6 The Use of JavaScript in This Book

Proficiency in JavaScript is an absolute prerequisite for this book, as discussed in the Preface. JavaScript, however, is a flexible and dynamic language, so there are many ways to use it. The purpose of this section is to show you how this book uses JavaScript; the intent is not to teach you anything at all about the language. To get the most out of this book, you must already know everything that you are about to read, or preferably skim, in this section.

This book defines several JavaScript objects that in more traditional languages such as C++ or Java would be implemented with classes. Those objects range from the games themselves (Snail Bait and Bodega's Revenge) to objects they contain, such as sprites and sprite behaviors. JavaScript objects are defined with a constructor function and a prototype, as shown in **Example 1.6**, a severely truncated listing of the `SnailBait` object.

Example 1.6 Defining JavaScript objects

```
var SnailBait = function () {
   // Constants and variables are declared here

   this.LEFT = 1;
   ...
};

SnailBait.prototype = {
   // Methods are defined here

   draw: function(now) { // The draw method takes a single parameter
      ...
   },
   ...
};
```

JavaScript objects are instantiated in this book with JavaScript's new operator, as shown in **Example 1.7**.

Example 1.7 Creating JavaScript objects

```
SnailBait.prototype = {
   ...

   createSnailSprites: function () {
      var snail,
          snailArtist = new SpriteSheetArtist(this.spritesheet,
                                              this.snailCells);

      for (var i = 0; i < this.snailData.length; ++i) {
         snail = new Sprite('snail',
                            snailArtist,

                            [
                              this.paceBehavior,
                              this.snailShootBehavior,

                              new CycleBehavior(
                                     300,  // 300ms per image
                                     5000) // 1.5 seconds interlude
                            ]);

         snail.width  = this.SNAIL_CELLS_WIDTH;
         snail.height = this.SNAIL_CELLS_HEIGHT;
```

(Continues)

Example 1.7 *(Continued)*

```
            snail.velocityX = snailBait.SNAIL_PACE_VELOCITY;

            this.snails.push(snail);
        }
    },
    ...
};
```

The `createSnailSprites()` function, which we refer to as a method because it resides in an object, creates a sprite sheet artist, a sprite, and an instance of `CycleBehavior`. That cycle behavior resides in an array of behaviors that `createSnailSprites()` passes to the `Sprite` constructor.

This book also defines objects using JSON (JavaScript Object Notation), as shown in **Example 1.8**.

Example 1.8 Defining JavaScript objects with JSON

```
var SnailBait = function () {
    ...

    // A single object with three properties

    this.fallingWhistleSound = {
        position: 0.03, // seconds
        duration: 1464, // milliseconds
        volume: 0.1
    };

    // An array containing three objects, each of which has two properties

    this.audioChannels = [
        { playing: false, audio: null, },
        { playing: false, audio: null, },
        { playing: false, audio: null, }
    ];
    ...
};
```

Finally, the JavaScript code in this book adheres closely to the subset of JavaScript discussed in Douglas Crockford's book *JavaScript: The Good Parts*. The code in this book also follows the coding conventions discussed in that book.

NOTE: The use of ellipses in this book

Most of the code listings in this book omit irrelevant sections of code. Those irrelevant sections are identified with ellipses (…) so that you can distinguish partial from full listings.

1.7 Conclusion

Snail Bait is an HTML5 platform game implemented with the `canvas` element's 2D API. As you'll see throughout the rest of this book, that API provides a powerful and intuitive set of functions that let you implement nearly any 2D game you can imagine.

In this chapter, we looked at Snail Bait from a high level to get a feel for its features and to understand some of the best practices it implements. Although you can get a good grasp of its gameplay from reading this chapter, you will have a much better understanding of the game if you play it, which you can do at corehtml5games.com.

At the end of this chapter, we looked at a starting point for Snail Bait that simply draws the background and the runner. Before we build on that starting point and begin coding in earnest, however, we'll take a brief detour in the next chapter to become familiar with the browser development environment and to see how to access freely available graphics, sound, and music. If you're already up to speed on HTML5 development in general and you know how to access open source assets online, feel free to skip ahead to Chapter 3.

1.8 Exercises

1. Use a different image for the background.
2. Draw the runner at different locations in the canvas.
3. Draw the background at different locations in the canvas.
4. In the `draw()` function, draw the runner first and then the background.
5. Remove the `width` and `height` attributes from the `snailbait-game-canvas` element in index.html and add `width` and `height` properties—with the same values of `800px` and `400px`, respectively—to the `snailbait-game-canvas` element in the CSS file. When you restart the game, does it look the same as before? Can you explain the result?

This page intentionally left blank

CHAPTER 2

Raw Materials and Development Environment

Topics in This Chapter

In this book we build a game. Like all builders, we must gather raw materials and become competent with our tools before we begin. For most games, the following raw materials are standard fare:

- Graphics
- Sound effects
- Music

The following, which add some polish to your HTML5 game, are optional:

- A favicon
- A webpage background
- An animated GIF

Favicons are small images that browsers display either in the address bar or in a tab. Webpage backgrounds can be images or, as is the case with Snail Bait, they can be drawn with CSS. Snail Bait also displays an animated GIF while it loads its resources.

Fortunately, all the necessary materials, from game graphics to animated GIFs, are readily available. Not only that, but you can easily find high-quality graphics, sound effects, and music on the Internet under permissive open source licenses, such as Creative Commons.

The following developer tools will help us turn the preceding materials into a compelling video game:

- Text editor
- Console
- Debugger
- Profiler
- Timelines

Additionally, game developers must have:

- An image editor
- A sound editor

Browser development environments, which typically contain all the preceding development tools and more, are also free. And you can also use high quality, freely available image and sound editors, such as GIMP and Audacity.

This chapter briefly describes the Chrome developer tools and shows you how to access freely available graphics, sound effects, and music on the Internet. You will also see how to edit sounds and images and how to create animated GIFs, favicons, and CSS backgrounds.

NOTE: Game development wasn't always this accessible

Before the advent of open source resources and freely available development environments, game development was much more difficult. Developers had to pay steep prices for development environments in addition to typically paying artists and musicians to create graphics, sound, and music for their games.

NOTE: Money-making games use open source resources

Nowadays there are many types of open source licenses, and some of them, like Creative Comments, pretty much let you do anything you want with open source resources, as long as you attribute works to the original artists. In fact, quite a few for-sale games are based entirely on open source graphics.

2.1 Use Developer Tools

You will undoubtedly use developer tools as you implement HTML5 games, and your familiarity with those tools will help determine how quickly and easily you can implement a game.

As this book was written, all major browser vendors—Chrome, Safari, Firefox, Opera, and Internet Explorer—provided powerful developer tools for free. Although the specifics of those tools vary, the fundamentals are similar. Developers log information to the console, debug with a debugger, locate performance bottlenecks with a profiler, and monitor events with timelines.

A comprehensive discussion of browser developer tools is beyond the scope of this book; however, this section exemplifies the use of Chrome's developer tools to implement games.

CAUTION: Chrome is a moving target

Over the course of the development of this book, Chrome's look-and-feel changed considerably. Some of the screenshots you see in this book may not correspond exactly to the current version of Chrome, but the functionality should be the same.

2.1.1 The Console

Video games are predicated upon time. Games tirelessly create one animation frame after another to create the illusion of motion. When a game draws an animation frame, it uses the elapsed time since the last animation frame to determine where to move its graphical objects, known as sprites.

If you set a breakpoint in the debugger at a line of code that's called for every animation frame, you will greatly increase the amount of time between frames, no matter how quickly you click the debugger's resume button. More importantly, values that depend on the amount of time between animation frames, such as the frame rate itself, will be enormously out of whack, as **Figure 2.1** depicts in the top screenshot, which shows a nonsensical frame rate of less than one frame per second in the debugger.

On the other hand, logging to the console does not cause the game's code to stop running, so you can monitor values such as frame rate, as illustrated in the bottom screenshot in Figure 2.1.

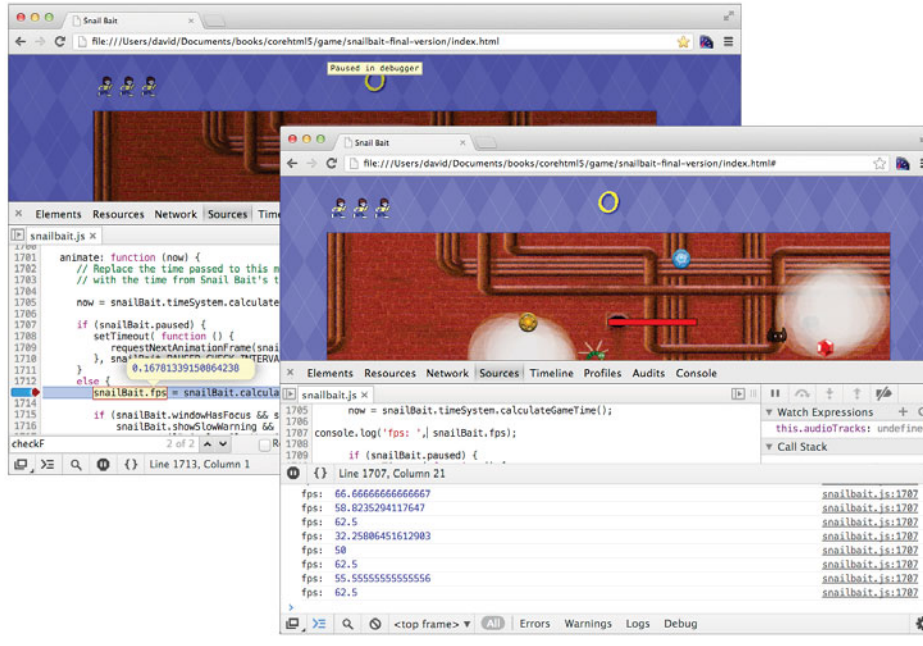


Figure 2.1 Debugger vs. console

In the bottom screenshot in Figure 2.1, Snail Bait uses the `console.log()` method to display the game's frame rate. Chrome's console comes with several other methods that, like `console.log()`, log messages to the console: `debug()`, `error()`, `info()`, and `warn()`. Those methods are identical to `log()`, but the browser categorizes them so you can filter out particular types of messages. Example 2.1 shows how Snail Bait uses the `error()` and `warn()` methods.

The JavaScript in Example 2.1 shows two of Snail Bait's sound methods, which we discuss in much greater detail in Chapter 14. It's an error if we cannot move to a particular position in a sound file (known as seeking) because by the time `seekAudio()` is invoked, all sounds are loaded and we should therefore be able to seek anytime we want.

On the other hand, if by some chance Snail Bait tries to simultaneously play more sounds than it can support, no audio channels will be available and Snail Bait won't be able to play the sound. However, not being able to play a sound when multiple sounds are already playing is not an error, so Snail Bait emits a warning to the console instead.

Example 2.1 Using the console to report warnings and errors

```
SnailBait.prototype = { // An object containing the game's methods
   ...

   seekAudio: function (sound, audio) {
      try {
         audio.pause();
         audio.currentTime = sound.position;
      }
      catch (e) {
         console.error('Cannot seek audio');
      }
   },

   playSound: function (sound) {
      var channel,
          audio;

      if (this.soundOn) {
         channel = this.getFirstAvailableAudioChannel();

         if (!channel) {
            if (console) {
               console.warn('All audio channels are busy. ' +
                            'Cannot play sound');
            }
         }
         else {
            ...
         }
      }
   },
   ...
};
```

Browsers graphically depict errors and warnings with red and yellow icons, respectively, and they also group them under error and warning categories so that you can view one or the other. The bottom screenshot in **Figure 2.1** shows buttons in the browser's status bar for filtering log messages by error, warning, log, or debug.

Chrome's console is full featured, as Table 2.1 illustrates.

Table 2.1 The Chrome Console API

Method	Description
`assert(expression, errormsg)`	If `expression` is `false`, the browser appends `errormsg` to "Assertion failed: " and shows the resulting string in the console as an error.
`clear()`	Erases all content from the console.
`count(label)`	Appends the number of times `count()` has been called (at that location in the code, and with the same label) to the label and displays the resulting string; for example, `console.count('Drawing')` results in output such as `Drawing:` *X*, where *X* is the number of times that line of code has executed with the label `Drawing`.
`debug(object [, object,...])`	The same as `log()` except messages are grouped under debug messages.
`dir(object)`	Displays a JavaScript representation of `object`.
`dirxml(object)`	Displays an XML JavaScript representation of `object` (as it would appear in the Elements panel).
`error(object, [, object,...])`	Identical to `log()` except that it emits an error message with a red icon and stack trace.
`group(object, [, object,...])`	Begins a new logging group that persists until the first call to `groupEnd()`. The browser visually groups all console output between those two calls.
`groupCollapsed(object, [, object,...])`	The same as `group()` except that the browser displays the group initially collapsed, instead of open (the default).
`groupEnd()`	Ends a group. See `group()` and `groupCollapsed()`.
`info(object, [, object,...])`	Identical to `log()`.

(Continues)

Table 2.1 *(Continued)*

Method	Description
`log(object, [, object,...])`	Displays each of the objects it is passed, concatenated into a space-delimited string. The first object can be a string with formatting characters, similar to `printf()` from the C language. Those formatting characters are: • `%s` (string) • `%d` or `%i` (integer) • `%f` (floating point) • `%o` (expandable DOM element) • `%0` (expandable JavaScript object) • `%c` (format with CSS you provide)
`profile(label)`	Starts a profile and assigns it the specified label. The profile runs until you call `profileEnd()`.
`profileEnd(label)`	Ends the profiler with the specified label.
`time(label)`	Starts a timer with a specified label. The timer runs until you call `timeEnd()`.
`timeEnd(label)`	Stops the timer with the specified label and displays the elapsed time in the console.
`timeline(label)`	Starts a timeline with a specified label. The timeline runs until you call `timelineEnd()`.
`timelineEnd(label)`	Stops the timeline with a specified label.
`timeStamp(label)`	Adds a timestamp event to a timeline when you are recording.
`trace()`	Prints a stacktrace.
`warn(object, [, object,...])`	The same as `log()` except that it emits a warning, complete with a yellow icon.

Several methods listed in Table 2.1 take arguments which are depicted in the table like this: `(object, [, object,...])`. The first argument is a string that can have