

C O R E

## HTML5

## 2D GAME PROGRAMMING



DAVID GEARY

FREE SAMPLE CHAPTER

SHARE WITH OTHERS



---

# **Core HTML5**

## **2D Game Programming**

---

*This page intentionally left blank*

---

# Core HTML5 2D Game Programming

---

**David Geary**



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

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at [corpsales@pearsoned.com](mailto:corpsales@pearsoned.com) or (800) 382-3419.

For government sales inquiries, please contact [governmentsales@pearsoned.com](mailto:governmentsales@pearsoned.com).

For questions about sales outside the United States, please contact [international@pearsoned.com](mailto:international@pearsoned.com).

Visit us on the Web: [informit.com/ph](http://informit.com/ph)

*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

Copyright © 2015 Clarity Training

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. To obtain permission to use material from this work, please submit a written request to Pearson Education, Inc., Permissions Department, One Lake Street, Upper Saddle River, New Jersey 07458, or you may fax your request to (201) 236-3290.

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* ..... xv

*Acknowledgments* ..... xxi

*About the Author* ..... xxiii

**Chapter 1: Introduction** ..... 1

1.1 Snail Bait ..... 3

1.1.1 Sprites: The Cast of Characters ..... 7

1.2 HTML5 Game Development Best Practices ..... 10

1.2.1 Pause the Game When the Window Loses Focus ..... 10

1.2.2 Implement a Countdown When the Window Regains Focus . 12

1.2.3 Use CSS for UI Effects ..... 12

1.2.4 Detect and React to Slowly Running Games ..... 14

1.2.5 Incorporate Social Features ..... 14

1.2.6 Put All the Game’s Images in a Single Sprite Sheet ..... 15

1.2.7 Store High Scores and Send Realtime, In-game Metrics to the Server ..... 16

1.3 Special Features ..... 16

1.4 Snail Bait’s HTML and CSS ..... 18

1.5 Snail Bait’s Humble Beginning ..... 25

1.6 The Use of JavaScript in This Book ..... 28

1.7 Conclusion ..... 31

1.8 Exercises ..... 31

**Chapter 2: Raw Materials and Development Environment** ..... 33

2.1 Use Developer Tools ..... 35

2.1.1 The Console ..... 35

2.1.2 Chrome Canary’s Frame Rate Counter ..... 40

2.1.3 Debugging ..... 42

2.1.4 Timelines ..... 44

2.1.5 Profiling ..... 49

2.2 Obtain Assets ..... 50

2.2.1	Graphics .....	50
2.2.2	Image Manipulation .....	51
2.2.3	Sound and Music .....	52
2.2.4	Animations .....	53
2.3	Use CSS Backgrounds .....	54
2.4	Generate Favicons .....	56
2.5	Shorten the Coding Cycle .....	58
2.6	Conclusion .....	59
2.7	Exercises .....	60
<b>Chapter 3: Graphics and Animation .....</b>		<b>61</b>
3.1	Draw Graphics and Images with the HTML5 canvas Element .....	64
3.1.1	Draw the Background .....	66
3.1.2	Draw the Runner .....	67
3.1.3	Draw Platforms .....	67
3.2	Implement Smooth HTML5 Animations .....	70
3.2.1	The requestAnimationFrame() Method .....	71
3.2.2	A requestAnimationFrame() Polyfill .....	72
3.3	Implement a Game Loop .....	75
3.4	Calculate Frame Rates .....	77
3.5	Scroll the Background .....	78
3.5.1	Translate the Coordinate System .....	79
3.5.2	Scroll Snail Bait's Background .....	81
3.6	Create Time-Based Motion .....	85
3.7	Reverse Scroll Direction .....	86
3.8	Draw Animation Frames .....	86
3.9	Use Parallax to Create the Illusion of Depth .....	87
3.10	Conclusion .....	90
3.11	Exercises .....	90
<b>Chapter 4: Infrastructure .....</b>		<b>93</b>
4.1	Encapsulate Game Functions in a JavaScript Object .....	95
4.1.1	Snail Bait's Constructor .....	95
4.1.2	Snail Bait's Prototype .....	97
4.2	Understand JavaScript's Persnickety this Reference .....	100
4.3	Handle Keyboard Input .....	103
4.4	Pause or Resume the Game When the Player Presses the p Key .....	105

4.5	Freeze the Game to Ensure It Resumes Exactly Where It Left Off .....	107
4.6	Pause the Game When the Window Loses Focus .....	108
4.7	Resume a Paused Game with an Animated Countdown .....	110
4.7.1	Display Toasts (Brief Messages) to Players .....	111
4.7.2	Snail Bait's Countdown .....	112
4.8	Conclusion .....	115
4.9	Exercises .....	116
<b>Chapter 5: Loading Screens .....</b>		<b>117</b>
5.1	Define Snail Bait's Chrome .....	120
5.1.1	Accessing Chrome Elements in JavaScript .....	122
5.2	Fade Elements In and Out with CSS Transitions .....	123
5.2.1	Fade Elements Into View .....	125
5.2.2	Fade Elements Out of View .....	127
5.2.3	The <code>snailbait-toast</code> Element's CSS .....	128
5.2.4	Revealing and Hiding Toasts .....	129
5.3	Fade Any Element In or Out That Has a CSS Transition Associated with Its Opacity .....	132
5.4	Implement the Loading Screen .....	135
5.5	Reveal the Game .....	140
5.6	Conclusion .....	144
5.7	Exercises .....	144
<b>Chapter 6: Sprites .....</b>		<b>147</b>
6.1	Sprite Objects .....	149
6.1.1	Sprite Properties .....	152
6.1.2	The <code>Sprite</code> Constructor .....	153
6.1.3	Sprite Methods .....	154
6.2	Incorporate Sprites into a Game Loop .....	156
6.3	Implement Sprite Artists .....	160
6.3.1	Stroke and Fill Artists .....	160
6.3.2	Image Artists .....	161
6.3.3	Sprite Sheet Artists .....	162
6.3.4	Define Sprite Sheet Cells .....	164
6.4	Create and Initialize a Game's Sprites .....	167
6.5	Define Sprites with Metadata .....	171
6.6	Scroll Sprites .....	174



6.7	Conclusion .....	176
6.8	Exercises .....	177
<b>Chapter 7: Sprite Behaviors .....</b>		<b>179</b>
7.1	Behavior Fundamentals .....	182
7.2	Runner Behaviors .....	184
7.3	The Runner's Run Behavior .....	187
7.4	Flyweight Behaviors .....	190
7.5	Game-Independent Behaviors .....	193
7.5.1	The Cycle Behavior .....	193
7.5.1.1	Sparkling Rubies and Sapphires .....	195
7.5.1.2	Flapping Wings and Throbbing Coins .....	197
7.6	Combine Behaviors .....	199
7.7	Conclusion .....	205
7.8	Exercises .....	206
<b>Chapter 8: Time, Part I: Finite Behaviors and Linear Motion .....</b>		<b>207</b>
8.1	Implement an Initial Jump Algorithm .....	209
8.2	Shift Responsibility for Jumping to the Runner .....	210
8.3	Implement the Jump Behavior .....	213
8.4	Time Animations with Stopwatches .....	214
8.5	Refine the Jump Behavior .....	217
8.6	Implement Linear Motion .....	220
8.6.1	Ascending .....	221
8.6.2	Descending .....	223
8.7	Pause Behaviors .....	225
8.8	Conclusion .....	227
8.9	Exercises .....	227
<b>Chapter 9: Time, Part II: Nonlinear Motion .....</b>		<b>229</b>
9.1	Understand Time and Its Derivatives .....	230
9.2	Use Animation Timers and Easing Functions to Implement Nonlinear Jumping .....	231
9.3	Implement Animation Timers .....	233
9.4	Implement Easing Functions .....	235
9.5	Fine-tune Easing Functions .....	239
9.6	Implement a Realistic Bounce Behavior .....	241
9.7	Randomize Behaviors .....	245

9.8	Implement Nonlinear Color Changes with Animation Timers and Easing Functions .....	247
9.8.1	The Pulse Behavior .....	249
9.9	Conclusion .....	251
9.10	Exercises .....	251
<b>Chapter 10: Time, Part III: Time Systems .....</b>		<b>253</b>
10.1	Snail Bait's Time System .....	255
10.2	Create and Start the Time System .....	257
10.3	Incorporate the Time System into Snail Bait .....	258
10.3.1	Use the Time System to Drive the Game's Animation .....	258
10.3.2	Implement a Game Method that Uses the Time System to Modify the Flow of Time .....	259
10.3.3	Factor the Time Rate into the Frame Rate Calculation .....	260
10.3.4	Pause and Resume the Game by Using the Time System .....	261
10.4	Redefine the Current Time for Stopwatches and Animation Timers .....	264
10.5	Implement the Time System .....	268
10.6	Conclusion .....	270
10.7	Exercises .....	270
<b>Chapter 11: Collision Detection .....</b>		<b>273</b>
11.1	The Collision Detection Process .....	275
11.2	Collision Detection Techniques .....	275
11.3	Snail Bait's Collision Detection .....	277
11.3.1	Sprite Collision Rectangles .....	278
11.3.2	The Runner's Collide Behavior .....	279
11.4	Select Candidates for Collision Detection .....	281
11.5	Detect Collisions Between the Runner and Another Sprite .....	282
11.6	Process Collisions .....	284
11.7	Optimize Collision Detection .....	286
11.7.1	Refine Bounding Boxes .....	286
11.7.2	Use Spatial Partitioning .....	288
11.8	Monitor Collision Detection Performance .....	289
11.9	Implement Collision Detection Edge Cases .....	291
11.10	Conclusion .....	295
11.11	Exercises .....	296

<b>Chapter 12: Gravity</b>	<b>297</b>
12.1 Equip the Runner for Falling	298
12.2 Incorporate Gravity	300
12.2.1 The Runner's Fall Behavior	302
12.2.2 Calculate Initial Falling Velocities	306
12.2.3 Pause When the Runner Is Falling	308
12.3 Collision Detection, Redux	308
12.4 Conclusion	310
12.5 Exercises	311
<b>Chapter 13: Sprite Animations and Special Effects</b>	<b>313</b>
13.1 Implement Sprite Animations	314
13.2 Create Special Effects	320
13.2.1 Shake the Game	321
13.2.2 Transition Between Lives	323
13.3 Choreograph Effects	329
13.3.1 Explode Bees	332
13.3.2 Detonate Buttons	333
13.4 Conclusion	335
13.5 Exercises	336
<b>Chapter 14: Sound and Music</b>	<b>337</b>
14.1 Create Sound and Music Files	339
14.2 Load Music and Sound Effects	340
14.3 Specify Sound and Music Controls	342
14.4 Play Music	343
14.5 Play Music in a Loop	344
14.6 Play Sound Effects	347
14.6.1 Create Audio Sprites	350
14.6.2 Define Sound Objects	351
14.6.3 Implement Multichannel Sound	353
14.6.3.1 Create Audio Channels	355
14.6.3.2 Coordinate with Sprite Sheet Loading to Start the Game	357
14.6.3.3 Play Sounds	358
14.7 Turn Sound On and Off	361

14.8	Conclusion .....	362
14.9	Exercises .....	362
<b>Chapter 15: Mobile Devices .....</b>	<b>363</b>	
15.1	Run Snail Bait on Mobile Devices .....	366
15.2	Detect Mobile Devices .....	368
15.3	Scale Games to Fit Mobile Devices .....	369
15.3.1	The viewport Meta Tag .....	371
15.3.2	Programmatically Resize Games to Fit Mobile Device Screens .....	376
15.4	Change Instructions Underneath the Game's Canvas .....	381
15.5	Change the Welcome Screen .....	383
15.5.1	Implement the Welcome Toast .....	384
15.5.1.1	Modify the Game's Start Sequence .....	385
15.5.1.2	Add HTML for the Mobile Welcome Toast .....	386
15.5.1.3	Define CSS for the Mobile Toasts .....	387
15.5.1.4	Implement Event Handlers for the Mobile Welcome Toast's Links .....	388
15.5.2	Draw Mobile Instructions .....	389
15.5.3	Implement the Mobile Start Toast .....	394
15.5.3.1	Implement the Start Link's Event Handler .....	395
15.5.4	Reveal the Mobile Start Toast .....	396
15.6	Incorporate Touch Events .....	396
15.7	Work Around Sound Idiosyncrasies on Mobile Devices .....	400
15.8	Add an Icon to the Home Screen and Run Without Browser Chrome .....	402
15.9	Conclusion .....	403
15.10	Exercises .....	404
<b>Chapter 16: Particle Systems .....</b>	<b>405</b>	
16.1	Smoking Holes .....	406
16.2	Use Smoking Holes .....	411
16.2.1	Define Smoking Hole Data .....	411
16.2.2	Create Smoking Holes .....	412
16.2.3	Add Smoking Holes to Snail Bait's sprites Array .....	413
16.2.4	Scroll Smoking Holes Every Animation Frame .....	413
16.3	Implement Smoking Holes .....	414

16.3.1	Disguise Smoking Holes as Sprites .....	415
16.3.2	Incorporate Fire Particles .....	417
16.3.2.1	Create Fire Particles .....	418
16.3.2.2	Draw and Update Fire Particles Every Animation Frame .....	421
16.3.3	Incorporate Smoke Bubbles .....	422
16.3.3.1	Create Smoke Bubbles .....	424
16.3.3.2	Draw and Update Smoke Bubbles Every Animation Frame .....	428
16.3.3.3	Emit Smoke Bubbles .....	430
16.3.3.4	Dissipate Smoke Bubbles .....	432
16.4	Pause Smoking Holes .....	434
16.5	Conclusion .....	435
16.6	Exercises .....	436
<b>Chapter 17:</b>	<b>User Interface .....</b>	<b>437</b>
17.1	Keep Score .....	438
17.2	Add a Lives Indicator .....	442
17.3	Display Credits .....	448
17.4	Tweet Player Scores .....	455
17.5	Warn Players When the Game Runs Slowly .....	458
17.5.1	Monitor Frame Rate .....	464
17.5.2	Implement the Running Slowly Warning Event Handlers ....	466
17.6	Implement a Winning Animation .....	467
17.7	Conclusion .....	472
17.8	Exercises .....	472
<b>Chapter 18:</b>	<b>Developer Backdoor .....</b>	<b>475</b>
18.1	Snail Bait's Developer Backdoor .....	477
18.2	The Developer Backdoor's HTML and CSS .....	479
18.3	Reveal and Hide the Developer Backdoor .....	481
18.4	Update the Developer Backdoor's Elements .....	483
18.5	Implement the Developer Backdoor's Checkboxes .....	484
18.5.1	Show and Hide Collision Rectangles .....	487
18.5.2	Enable and Disable the Running Slowly Warning .....	489
18.5.3	Show and Hide Smoking Holes .....	490
18.5.4	Update Backdoor Checkboxes .....	491

18.6	Incorporate the Developer Backdoor Sliders .....	492
18.6.1	Specify the HTML and CSS for the Backdoor's Sliders .....	494
18.6.2	Access Slider Readouts in Snail Bait's JavaScript .....	496
18.6.3	Create and Initialize the Backdoor's Sliders .....	497
18.6.4	Wire the Running Slowly Slider to the Game .....	498
18.6.5	Wire the Time Rate Slider to the Game .....	498
18.6.6	Wire the Game to the Time Rate Slider .....	499
18.6.7	Update Sliders Before Revealing the Backdoor .....	500
18.7	Implement the Backdoor's Ruler .....	502
18.7.1	Create and Access the Ruler Canvas .....	503
18.7.2	Fade the Ruler .....	504
18.7.3	Draw the Ruler .....	505
18.7.4	Update the Ruler .....	507
18.7.5	Drag the Canvas .....	507
18.8	Conclusion .....	513
18.9	Exercises .....	513

## **Chapter 19: On the Server: In-game Metrics, High Scores, and**

<b>Deployment .....</b>	<b>515</b>
19.1 Node.js and socket.io .....	517
19.2 Include socket.io JavaScript in Snail Bait .....	518
19.3 Create a Simple Server .....	520
19.4 Create a Socket on the Server .....	520
19.5 Start the Server .....	521
19.6 Create a Socket on the Client and Connect to the Server .....	522
19.7 Record In-game Metrics .....	523
19.8 Manage High Scores .....	526
19.8.1 The High Scores User Interface .....	527
19.8.2 Retrieve High Scores from the Server .....	530
19.8.3 Display High Scores on the Client .....	533
19.8.4 Monitor Name Input .....	534
19.8.5 Validate and Set the High Score on the Server .....	536
19.8.6 Redisplay High Scores .....	538
19.8.7 Start a New Game .....	539
19.9 Deploy Snail Bait .....	540
19.10 Upload Files to a Server .....	542

19.11 Conclusion .....	543
19.12 Exercises .....	544
<b>Chapter 20: Epilogue: Bodega's Revenge .....</b>	<b>545</b>
20.1 Design the User Interface .....	547
20.2 Create the Sprite Sheet .....	551
20.3 Instantiate the Game .....	552
20.4 Implement Sprites .....	553
20.4.1 The Turret .....	553
20.4.1.1 Create the Turret Sprite's Artist .....	554
20.4.1.2 Draw the Turret .....	555
20.4.2 Bullets .....	556
20.4.3 Birds .....	560
20.5 Implement Sprite Behaviors .....	563
20.5.1 Turret Behaviors .....	564
20.5.1.1 The Turret's Rotate Behavior .....	564
20.5.1.2 The Turret's Barrel Fire Behavior .....	566
20.5.1.3 The Turret's Shoot Behavior .....	569
20.5.2 Bullet Behaviors .....	571
20.5.3 Bird Behaviors .....	574
20.5.3.1 The Bird Move Behavior .....	575
20.5.3.2 The Bird Collide Behavior .....	577
20.5.3.3 The Bird Explosion Behavior .....	579
20.6 Draw the Bullet Canvas .....	580
20.7 Implement Touch-Based Controls for Mobile Devices .....	582
20.8 Conclusion .....	585
20.9 Exercises .....	585
<i>Glossary</i> .....	<i>587</i>
<i>Index</i> .....	<i>595</i>

---

# 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](http://corehtml5games.com/book/code/ch01). URLs for each of the book’s chapters follow the format [corehtml5games.com/book/code/ch??](http://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](http://corehtml5games.com)

Play Snail Bait: [corehtml5games.com/snailbait](http://corehtml5games.com/snailbait)

Play Bodega's Revenge: [corehtml5games.com/bodegas-revenge](http://corehtml5games.com/bodegas-revenge)

Download Snail Bait: [corehtml5games.com/book/downloads/snailbait](http://corehtml5games.com/book/downloads/snailbait)

Download Bodega's Revenge: [corehtml5games.com/book/downloads/bodegas-revenge](http://corehtml5games.com/book/downloads/bodegas-revenge)

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

# Introduction

## Topics in This Chapter

- 1.1 Snail Bait — p. 3
- 1.2 HTML5 Game Development Best Practices — p. 10
- 1.3 Special Features — p. 16
- 1.4 Snail Bait's HTML and CSS — p. 18
- 1.5 Snail Bait's Humble Beginning — p. 25
- 1.6 The Use of JavaScript in This Book — p. 28
- 1.7 Conclusion — p. 31
- 1.8 Exercises — p. 31

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](http://corehtml5games.com/snailbait), and you can find Bodega's Revenge at [corehtml5games.com/bodegas-revenge](http://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](#).

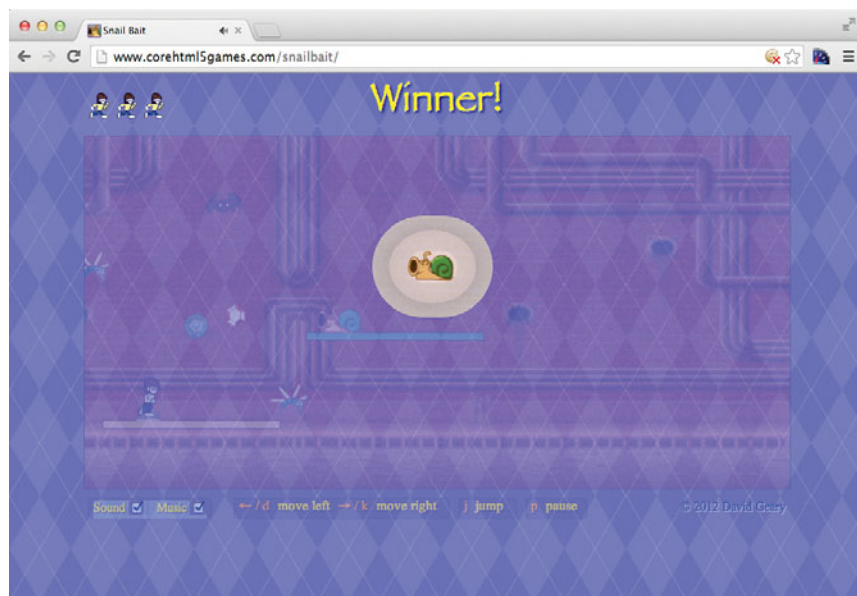


Figure 1.2 Snail Bait's winning animation

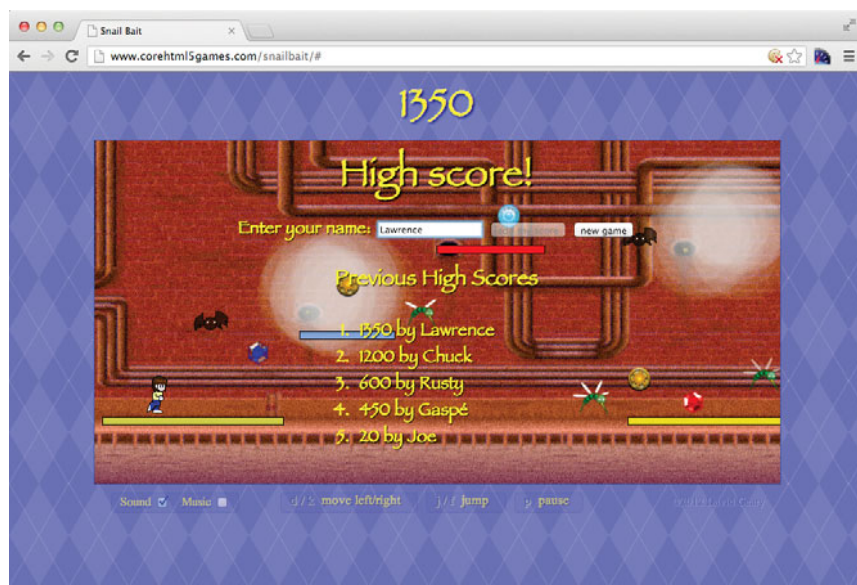
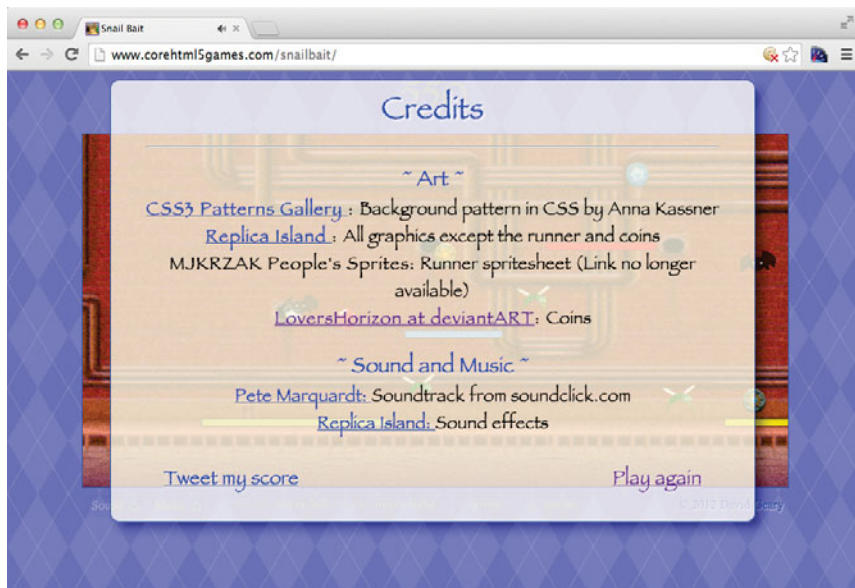


Figure 1.3 Snail Bait's high scores

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](http://replicaisland.net), and you can read about the Strategy design pattern at [http://en.wikipedia.org/wiki/Strategy\\_design\\_pattern](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.

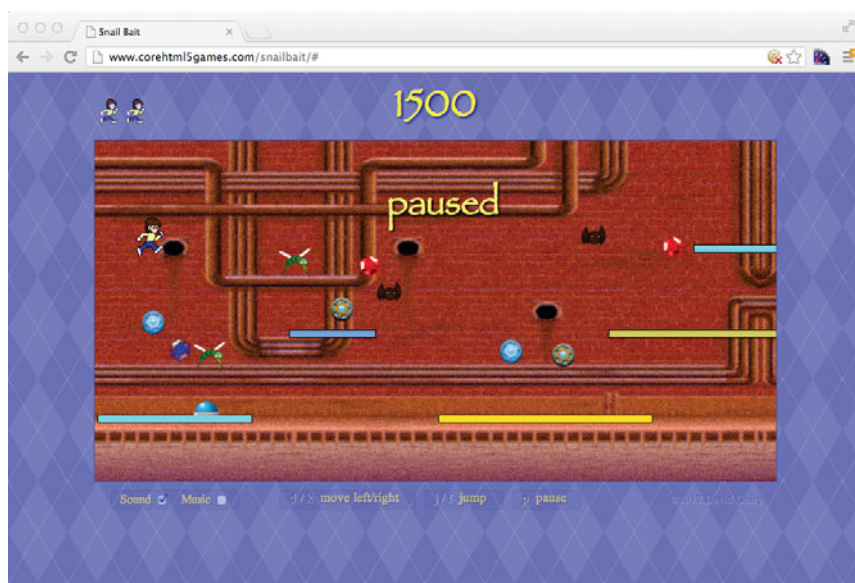
### 1.2.1 Pause the Game When the Window Loses Focus

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](#).



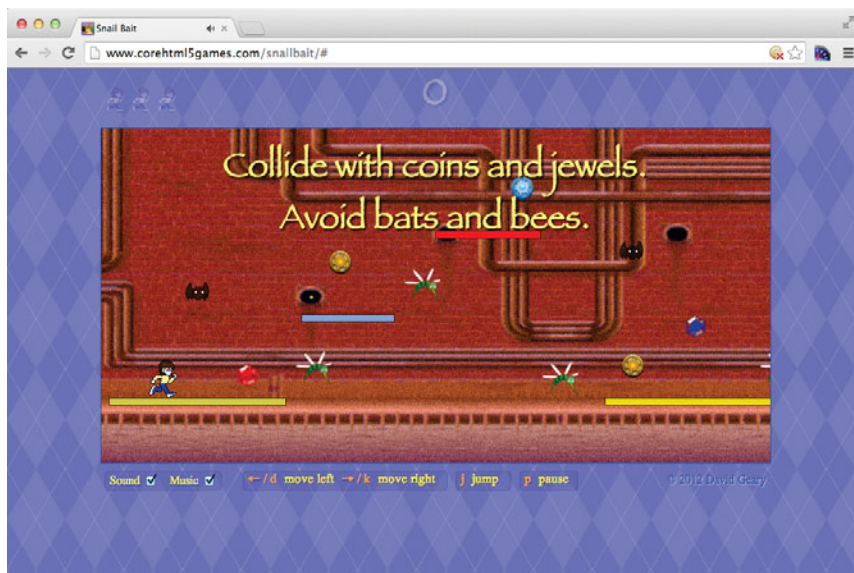
**Figure 1.6** Snail Bait’s countdown after the window regains focus

## 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 indicator 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.

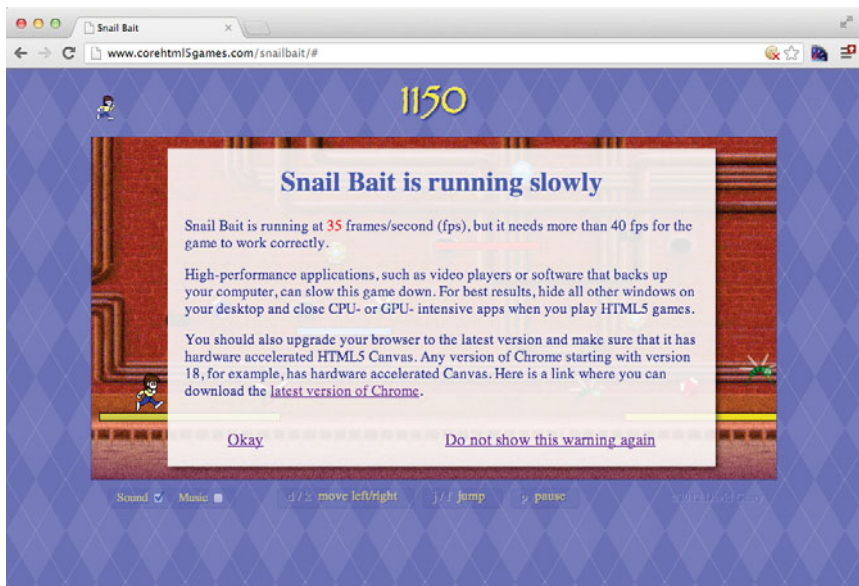
Snail Bait goes to all that trouble to focus attention on what's currently important. Initially, players should pay attention to the instructions below the game's canvas; once the game is underway, players will be more focused on their score and how many lives are remaining.



### 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](#).

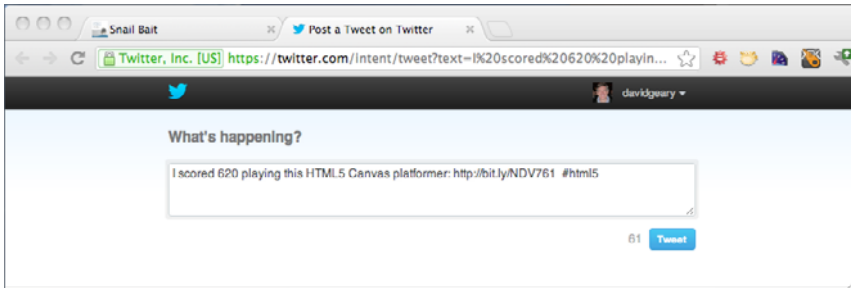


Figure 1.9 Snail Bait's Twitter integration

### 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.

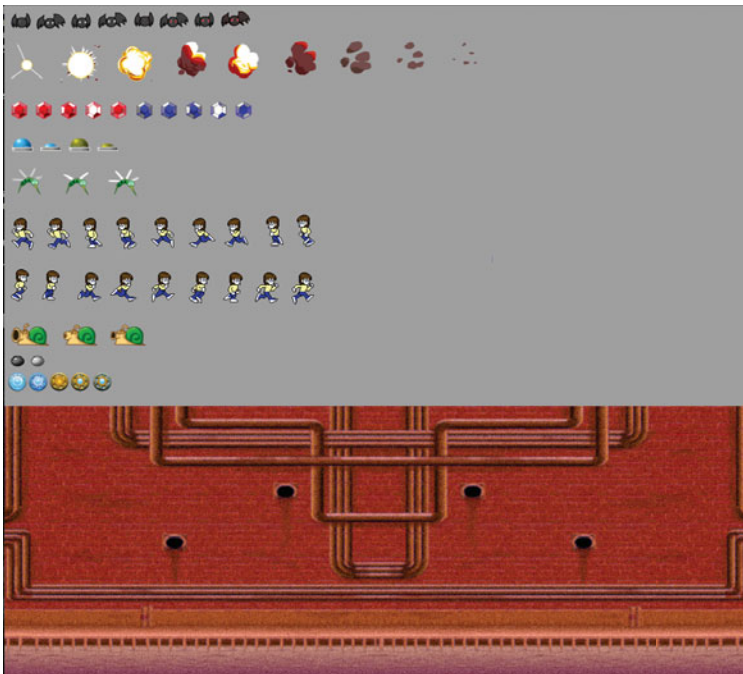


Figure 1.10 Snail Bait's sprite sheet (the gray background is transparent)



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.

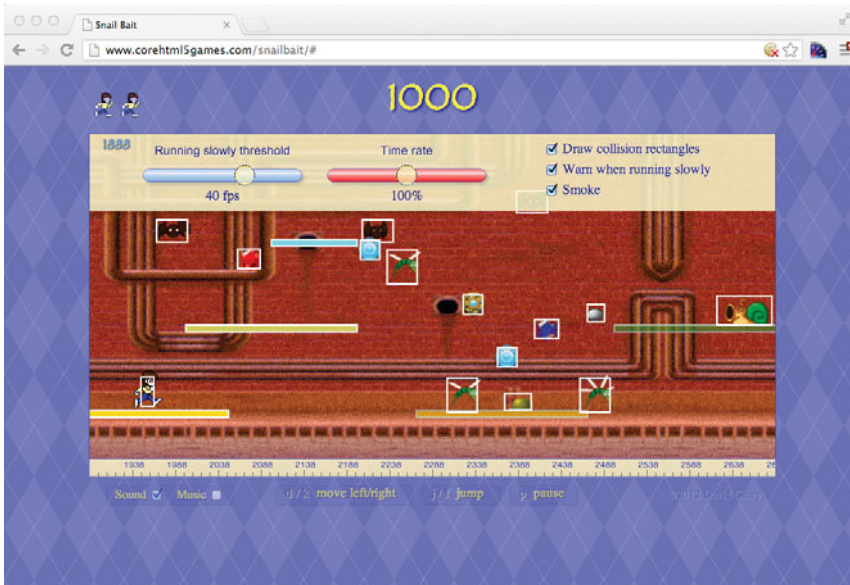


Figure 1.11 Snail Bait's developer backdoor

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 *misrepresent* 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**.

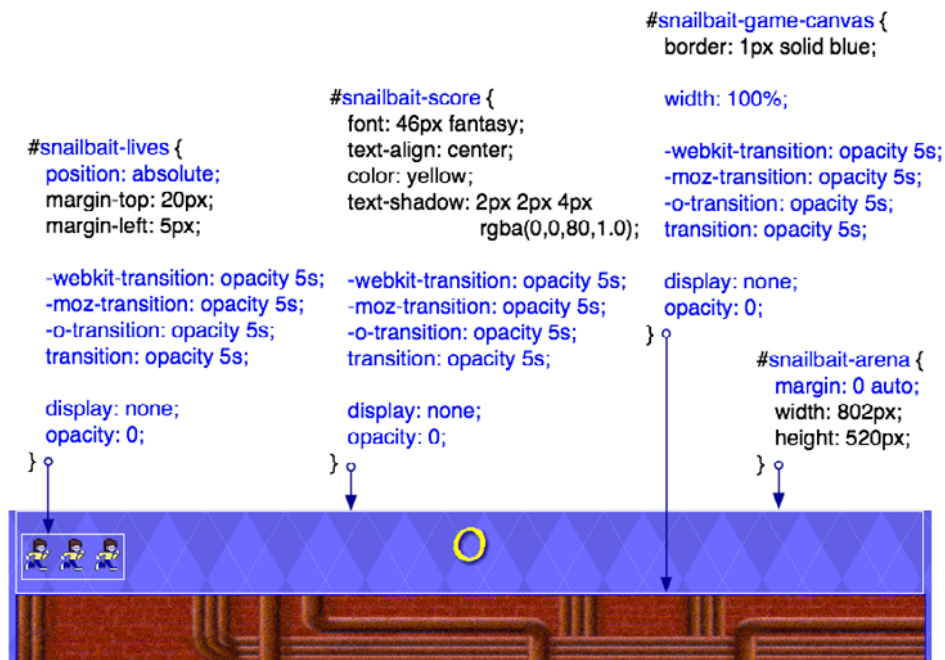


Figure 1.12 Snail Bait's CSS for the top half of the game

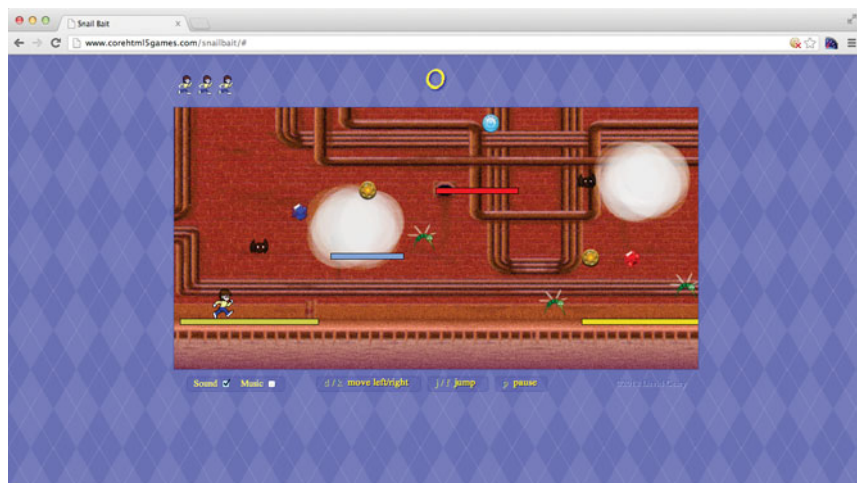
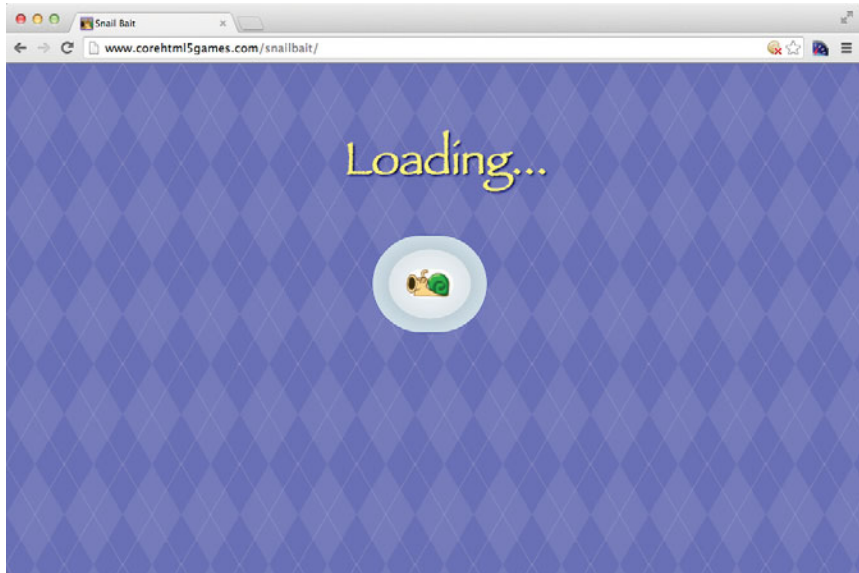


Figure 1.13 Snail Bait stays centered horizontally in the window

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,

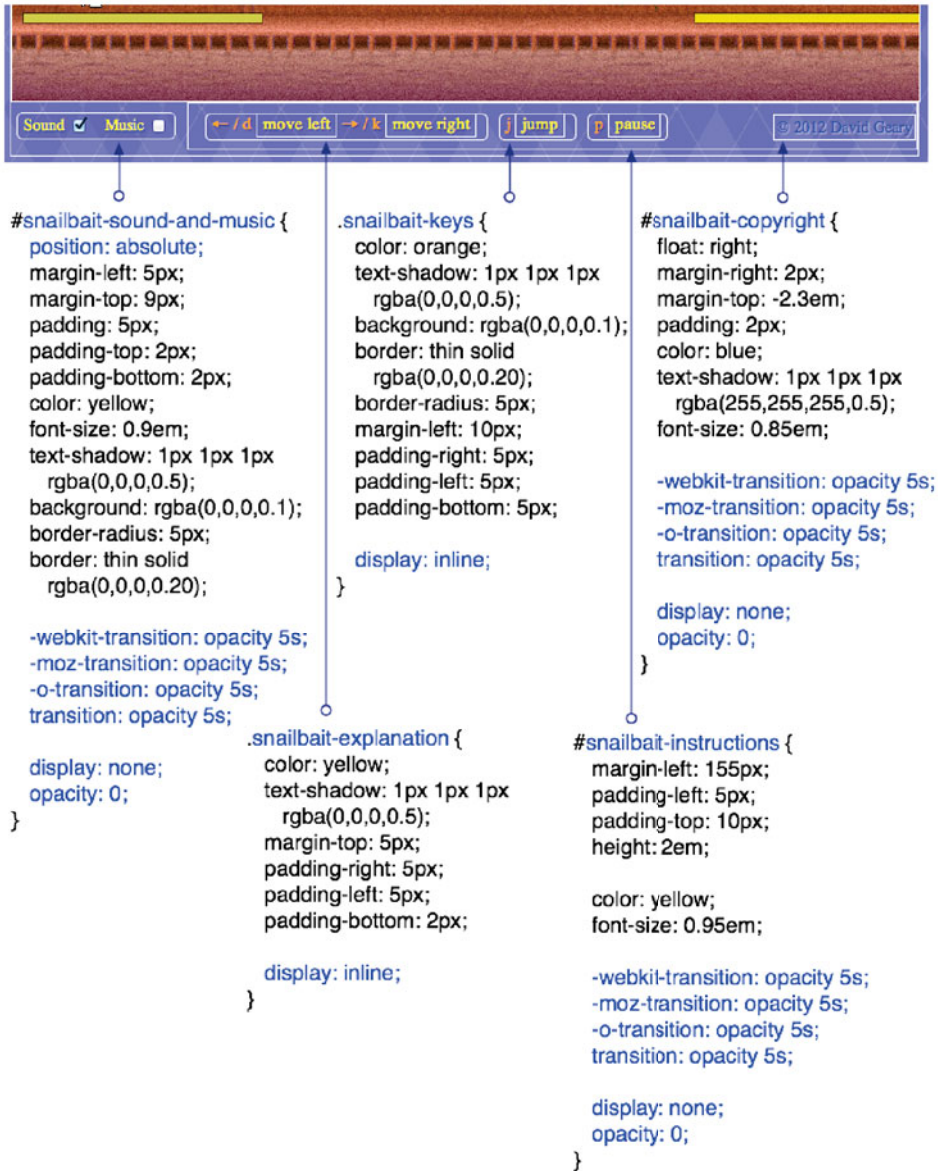


Figure 1.15 Snail Bait's CSS for the bottom of the game

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  $300 \times 150$  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-`.

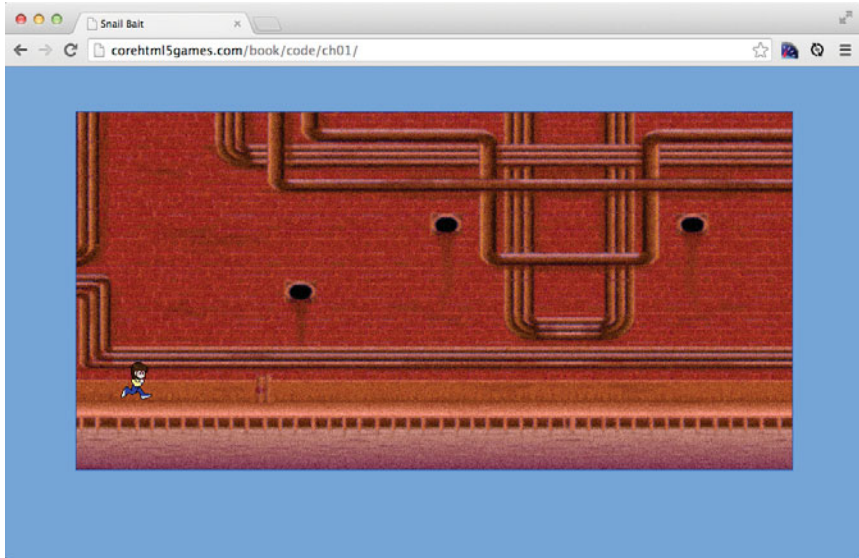
## 1.5 Snail Bait's Humble Beginning

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.

Name	Size	Kind
images	--	Folder
background.png	1.2 MB	Portable Network Graphics image
runner.png	845 bytes	Portable Network Graphics image
index.html	442 bytes	HTML document
snailbait.css	127 bytes	CSS
snailbait.js	668 bytes	JavaScript

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](http://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*

# Index

## A

- add my score button
  - disabling, 534, 539
  - enabling, 527, 535–536
  - event handler for, 536
- addBehaviors() method (SmokingHole), 417, 430–431
- addChangeListener() method (Slider), 493
- addEventListener() method (window)
  - during transitions, 327
  - for developer backdoor, 483
  - for jumps, 209–210
  - on size or orientation changes, 378
  - vs. onkeydown, 105
- addSpriteMethods() method (SmokingHole), 416, 421
- addSpriteProperties() method (SmokingHole), 415–416
- addSpritesToSpriteArray() method (SnailBait), 168, 413
- addTouchEventHandlers() method (SnailBait), 396–397
- adjustScore() method (collideBehavior), 441
- adjustVerticalPosition() method (BounceBehavior), 243–245
- Adobe Illustrator, 64
- advance() method (SpriteSheetArtist), 163–164
- advanceCursor() method (SmokingHole), 431
- advanceSpeedSamplesIndex() method (SnailBait), 466
- all.css, all.js files, 541–542
- Android
  - adding an icon to home screen on, 402
  - audio sprites on, 401
  - HTML5 applications on, 364–367
  - layout viewport on, 371
  - mobile instructions on, 390
  - remote debugging for, 365
  - scaling games on, 370
  - size of image files on, 164
  - viewport directives on, 375–376
- animate() function, 76–77, 86, 100
- animate() method (SnailBait), 101, 106, 156–160
  - double buffering with, 63
  - monitoring frame rate with, 464–465
  - naive implementation of, 100
  - using time system for, 258–259
- animated GIFs
  - creating, 54
  - for loading sequence, 135, 139
  - for winning sequence, 467–468
- animation frames
  - drawing, 86–87
  - last, time of, 108, 159
- animation loop. *See* game loop
- animation timers
  - duration of, 235
  - for jumps, 231–233
  - implementing, 233–235
  - redefining current time for, 264–265
- animations, 53–54, 61–90
  - implementing, 314–320
  - smooth, 63, 70–75
  - throttled heavily, 108–109
- AnimationTimer object
  - getElapsedTime() method, 234–235, 265
  - isExpired(), isPaused(), isRunning() methods, 234, 265
  - makeEaseXXXEasingFunction() methods, 232–233, 238–239, 242, 428
  - pause(), unpause() methods, 234, 263, 265
  - prototype object, 264–265
  - reset(), stop() methods, 234, 265
  - start() method, 234, 264
- Apache 2.0 license, 50–51
- appendFile() method (fs), 526
- appendTo() method (Slider), 493–494, 498
- Apple, 64, 373
- arc() method (canvas context), 64

- arena, 18
    - calculating size of, 379–380
    - HTML/CSS for, 18–24
    - resizing, 381
  - arguments variable (JavaScript), 133
  - armSnails() method (SnailBait), 202–203
  - artist property (sprites), 152
  - artists (for sprites), 9–10
    - benefits of, 151
    - implementing, 160–167
    - types of, 160–164
  - ascend() method (jumpBehavior), 221–222
  - ascendTimer property (runner), 218–219, 232
  - aspect ratio
    - cropping, 376–377
    - maintaining, 375
  - assert() method (console), 38
  - assets. *See* coins, jewels
  - Audacity, 53
    - creating audio sprite sheets in, 350–351
    - determining:
      - length of music in, 346–347
      - position and duration in, 351–352
  - audio channels, 353–361
    - creating, 355–356
    - getting available, 359
    - HTML for, 355–356
  - audio element (HTML)
    - for audio channels, 355
    - loop attribute, 344
    - on mobile devices, 400
    - preload attribute, 340–341
  - audio sprite sheets, 52–53, 350–351
    - on mobile devices, 401
    - seeking for audio in, 339, 359–360
  - audio.currentTime property, 360
  - audioChannels array (SnailBait), 355, 359
  - audioSpriteCountdown property (SnailBait), 357–358
  - authentication, 517
- B**
- backdoor, 16–17, 475–513
    - handling events for, 483
    - HTML/CSS for, 480
    - visibility of, 480–484
  - background
    - drawing, 66, 81–85, 547–548
    - left and right edges of, 84
    - loading, 28
      - starting game upon, 77
    - scrolling, 78, 86–86, 187, 307
      - direction of, 86
      - in developer backdoor, 502–513
      - key event handlers for, 103–105
      - slowing during transitions, 454
      - velocity of, 189–190
    - setting offset of, 81, 86–90, 175–176
  - background property (CSS), 480
  - BACKGROUND\_VELOCITY constant, 189
  - backgroundLoaded() method (SnailBait), 139
  - backgroundOffset property (SnailBait), 82–85
  - bats and bees, 4
    - colliding with, 285, 313–320, 442
    - exploding (bees only), 330–333
    - flapping wings, 197–199
  - beginPath() method (canvas context), 64, 283
  - behaviors (for sprites), 8, 179–205
    - and graphics context, 184
    - benefits of, 151
    - changing at runtime, 183
    - combining, 183, 199–205
    - game-independent, 193–199
    - generalizing, 195
    - implementing, 182–183
    - iterating over, 8
    - pausing/unpausing, 225–227, 262–263
    - randomizing, 245–247, 561
    - stateless. *See* flyweights
    - triggers for, 212, 214
  - behaviors property (sprites), 152
  - bgVelocity property (SnailBait), 189–190
  - birds
    - adjusting position of, 578
    - behaviors of, 575–579
    - creating, 560–562
    - randomizing properties of, 561–562
  - blue button, 8
    - creating, 335, 469
    - detonating, 330–331, 333–335, 469
    - pace behavior of, 190–193
  - blueButtonDetonateBehavior, 334–335
  - blur event handler, 105

- Bodega's Revenge, 545–585
    - on mobile devices, 582
    - playing online, 3
  - BodegasRevenge object, 552
    - createBirdCollideBehavior() method, 577–579
    - createBirdMoveBehavior() method, 575–577
    - createBirds() method, 560–562
    - createBulletArtist() method, 558
    - createBulletMoveBehavior() method, 571–574
    - createBullets() method, 556–558
    - createExplosionBehavior() method, 579
    - createTurret() method, 553–556
    - createTurretArtist() method, 554–555
    - createTurretBarrelFireBehavior() method, 567–568
    - createTurretShootBehavior() method, 569
    - drawBulletCanvas() method, 581–582
    - eraseBulletCanvas() method, 581
    - getBullet() method, 570, 577
    - getBulletLocation() method, 572
    - initializeBirdProperties() method, 561–562
    - isBulletInsideBird() method, 578–579
    - isBulletOutOfPlay() method, 572
    - loseOneBullet() method, 575–577, 580–581
    - lostBulletIndex property, 558, 570, 576–577, 582
    - polarToCartesian() method, 573
  - border-radius property (CSS), 453
  - bottom chrome. *See* instructions, Music
    - checkbox, Sound checkbox
  - BounceBehavior, 241–245
    - adjustVerticalPosition() method, 243–245
    - bouncing property, 243–244
    - constructor for, 242
    - execute() method, 243
    - pause(), unpause() methods, 245
    - resetTimer() method, 243–244
    - startBouncing() method, 243–244
  - bounding areas, 275–276
  - bounding boxes, 277–281
    - refining, 286–288
    - See also* collision rectangles
  - bounding volumes, 275
  - box-shadow property (CSS), 453
  - Braid game, 6, 179, 181
  - brighten() method (PulseBehavior), 250–251
  - browsers
    - audio/video formats in, 339–340
    - clamping frame rates in, 10–12
    - conditional breakpoints in, 42–43
    - errors/warnings in, 37
    - hardware acceleration in, 7, 50, 290
    - loop HTML attribute in, 344
    - profiling in, 49–50, 289–291
    - refreshing automatically, 58–59
    - setting sizes in, 381
    - specific functionality of, 72–75, 129
    - throttling heavily, 108–109
    - viewports in, 371–372
    - Web Audio API support in, 339
  - bullet canvas, 580–582
  - bullets
    - artist for, 558
    - creating, 556–558
    - drawing, 559
    - losing, 575–577, 580–581
    - moving, 557–558, 571–574
    - shooting, 563
  - buttons, 8
    - creating, 335, 469
    - detonating, 330–331, 333–335, 469–471
    - pace behavior of, 190–193
- ## C
- C++, timestamps in, 217
  - calculateArenaSize() method (SnailBait), 379–380
  - calculateAverageSpeed() method (SnailBait), 466
  - calculateCollisionRectangle() method (Sprite), 278–279, 487
  - calculateFps() function, 76–78
  - calculateFps() method (SnailBait), 260–261
  - calculateGameTime() method (TimeSystem), 255, 258, 268–270
  - calculateVerticalDrop() method (fallBehavior), 303–305
  - cannonSound object (SnailBait), 352

## Canvas

- 2D context, 64
  - arc() method, 64
  - beginPath() method, 64, 283
  - clearRect() method, 581
  - drawImage() method, 64–67, 162–163, 559
  - drawText() method, 79
  - fill() method, 65
  - fillRect() method, 64–65, 69
  - fillStyle attribute, 66
  - fillText() method, 391–393, 506
  - globalAlpha attribute, 66, 69, 155
  - isPointInPath() method, 65, 283, 578–579
  - lineTo(), moveTo() methods, 506
  - lineWidth attribute, 66
  - rect() method, 65, 283
  - restore() method, 65, 69, 155
  - rotate() method, 559
  - save() method, 65, 69, 155
  - stroke() method, 65
  - strokeRect() method, 64–65, 69
  - strokeStyle attribute, 66
  - translate() method, 65, 80–81, 559
- as immediate-mode graphics system, 63
- double buffering in, 63
- canvas element (HTML5), 20, 24
- CSS for, 19–20, 326
- dragging:
  - accidentally, on mobile devices, 397–399
  - in developer backdoor, 507–513
- drawing surface of, 66, 79
- hardware accelerated, 7, 50, 290
- implementing sliders with, 492
- not focusable, 103
- preventing zooming in/out for, 370–371, 376, 398–399
- revealing, 143
- saving/restoring context for, 389
- scaling, 25
- CANVAS\_WIDTH\_IN\_METERS constant, 302
- Cartesian coordinates, 572
- cells
  - bounding boxes of, 165–167
  - defining, 164–167
  - implementing animations with, 314–320
  - separate arrays of, and performance, 167

- CellSwitchBehavior, 314–320, 568, 579
  - execute() method, 315–316
  - revert(), switchCells() methods, 316–317
- change event handler, 343, 361, 489–491
- checkboxes
  - fading in/out, 12
  - implementing, 484–492
  - updating, 483–484, 491
- checkFps() method (SnailBait), 465
- checkHighScores() method (SnailBait), 530
- chrome
  - accessing in JavaScript, 122–123
  - defining, 120–123
  - fading in/out, 119, 140, 143
  - focusing attention on, 445
  - HTML for, 121–122
- Chrome browser
  - audio/video formats in, 339–340
  - conditional breakpoints in, 42–43
  - debugger in, 521
  - displaying timelines in, 46–47
  - free developer tools in, 35
  - hardware acceleration in, 7, 50, 290
  - live-editing JavaScript in, 42, 45–46
  - look-and-feel of, 35
  - profiling in, 49–50, 289–291
- Chrome Canary, 40–42
- clear() method (console), 38
- clearRect() method (canvas context), 581
- click event handler
  - for add my score button, 536–537
  - for new game button, 539
  - for Play again link, 455
  - for running slowly warning, 466–467
  - for Show how to use the controls link, 389
  - for Start link, 395–396
  - for Start the game link, 388
- clients
  - creating sockets on, 522
  - emitting messages from, 524–525, 530–532, 536
  - heads-up display on:
    - creating, 534
    - displaying, 538
    - hiding, 539
    - including socket.io, 519
- clientX, clientY properties, 511

- Cocoa API, 64
- coins, 4, 8
  - assigning values to, 440
  - bouncing, 241–245
  - colliding with, 284
    - and score, 438, 441
    - sound effects for, 347, 442
  - throbbing, 197–199
- coinSound object (SnailBait), 352
- collideBehavior, 184–186, 279–281
  - adding to runner, 151
  - adjustScore() method, 441
  - didCollide() method, 282–283, 291, 293–294
  - didRunnerCollideWithXXX() methods, 294–295
  - execute() method, 281
  - isCandidateForCollision() method, 281–282, 288
  - processAssetCollision() method, 284–285, 349, 441–442
  - processBadGuyCollision() method, 284–285, 320–321, 324–325
  - processCollision() method, 284, 333–334
  - processPlatformCollisionDuringJump() method, 285, 348
- collision detection, 273–295
  - and heavily throttled frame rates, 109
  - candidates for, 281–282
  - debugging, 487–489
  - edge cases of, 291–295, 310
  - inverting, 293
  - optimizing, 286–289
  - performance of, 288–291
  - processing, 284–286
  - with platforms, 308–310
- collision margins, 278–279, 287
- collision rectangles
  - calculating, 278–279
  - drawing, 487–489
  - See also* bounding boxes
- color, nonlinear changes of, 247–251
- Commodore Amiga, 149
- CommonJS, 521
- connect() method (socket.io), 522
- console object, 35–40
  - assert() method, 38
  - clear() method, 38
  - count() method, 38
  - debug() method, 36, 38
  - dir(), dirxml() methods, 38
  - error() method, 36–38
  - group(), groupXXX() methods, 38
  - info() method, 36, 38
  - log() method, 36, 39–40
  - logging to, 36
  - online API reference to, 40
  - profile(), profileEnd() methods, 39, 50
  - time(), timeEnd() methods, 39
  - timeline(), timelineEnd() methods, 39, 45–46
  - timestamp() method, 39
  - warn() method, 36–37, 39
- constants, 132
- constructors (JavaScript), 28
- coordinate system, 66
  - translating, 65, 79–82, 104, 158, 559, 566
- corehtml5games.com, 3, 165
- count() method (console), 38
- countdown, after regaining focus, 12, 45, 110–115
- countdown toast, 94, 110–112
  - fading in/out, 123–132
  - HTML/CSS for, 111
- countdownInProgress property (SnailBait), 113–115
- Cracker Jack, 71
- createAudioChannels() method (SnailBait), 354–356
- createBatSprites() method (SnailBait), 168–170, 197
- createBeeSprites() method (SnailBait), 168, 197–198, 332
- createBirdCollideBehavior() method (BodegasRevenge), 577–579
- createBirdMoveBehavior() method (BodegasRevenge), 575–577
- createBirds() method (BodegasRevenge), 560–562
- createBubbleArtist() method (SmokingHole), 427
- createBubbleSprite() method (SmokingHole), 425–426
- createBubbleSpriteTimer() method (SmokingHole), 428, 435
- createBulletArtist() method (BodegasRevenge), 558

- `createBulletMoveBehavior()` method (BodegasRevenge), 571–574
  - `createBullets()` method (BodegasRevenge), 556–558
  - `createButtonSprites()` method (SnailBait), 168, 190–193, 335, 469
  - `createCoinSprites()` method (SnailBait), 168, 198–199, 241–242, 245–247
  - `createDissipateBubbleBehavior()` method (SmokingHole), 432
  - `createElement()` method (browser), 356
  - `createExplosionBehavior()` method (BodegasRevenge), 579
  - `createFireParticle()` method (SmokingHole), 419–420
  - `createFireParticleArtist()` method (SmokingHole), 420–421
  - `createFireParticles()` method (SmokingHole), 418–419
  - `createPlatformSprites()` method (SnailBait), 168–171, 248
  - `createRubySprites()` method (SnailBait), 168, 196–197
  - `createRunnerSprite()` method (SnailBait), 168–171, 287, 317
  - `createSapphireSprites()` method (SnailBait), 168
  - `createServer()` method (http), 520
  - `createSmokeBubbles()` method (SmokingHole), 424–425
  - `createSmokingHoles()` method (SnailBait), 412
  - `createSnailSprites()` method (SnailBait), 29–30, 168, 201
  - `createSprites()` method
    - of BodegasRevenge, 553
    - of SnailBait, 168, 173, 412
  - `createTurret()` method (BodegasRevenge), 553–556
  - `createTurretArtist()` method (BodegasRevenge), 554–555
  - `createTurretBarrelFireBehavior()` method (BodegasRevenge), 567–568
  - `createTurretShootBehavior()` method (BodegasRevenge), 569
  - Creative Commons license, 34–35
  - credits, 6, 448–455, 550
    - CSS for, 451–453, 455
    - fading in/out, 123–132, 453–454
    - HTML for, 449–450
  - Crockford, Douglas, 30
  - cross-site scripting (XSS), 536
  - CSS (Cascading Style Sheets)
    - background properties in, 54–56
    - color strings in, 66
    - minifying and obfuscating, 541–542
    - namespacing in, 25
    - pixels in, 374
    - using for UI elements, 12–13
  - CSS Device Adaption, 376
  - CSS3 Patterns Gallery, 54–56, 547
  - CSS3 transitions, 13, 123–135
    - for credits, 447, 453
    - for game canvas, 326–327
    - for lives indicator, 444
    - for running slowly warning, 462
    - for score indicator, 440
    - for toasts, 128–129
  - CSVs (comma-separated values), 530
  - current time, 75, 159, 258–259
    - redefining, 217, 264
  - `currentTime` property
    - of audio, 360
    - of `musicElement`, 345–346
  - cursor
    - changing type of, 481, 508–509
    - recording original, 508–509
    - restoring, 482
  - `CycleBehavior`, 193–199, 201
- ## D
- `debug()` method (console), 36, 38
  - debugging, 42–43
    - adding breakpoints for, 35, 42–43
    - on servers, 521
    - remotely, for mobile devices, 365
  - Decorator pattern, 235
  - deployment, 540–542
  - `descend()` method (`jumpBehavior`), 223–224
  - `descendTimer` property (runner), 218–219, 232
  - `detectMobile()` method (SnailBait), 368
  - `detonating` property
    - of blue button, 334–335
    - of gold button, 470–471
  - developer backdoor. *See* backdoor
  - developer tools, 35–50
    - cost/benefit ratio of, 166, 479

- free, in major browsers, 35
- developerBackdoorVisible property (SnailBait), 481–482
- device pixels, 374
- didCollide() method (collideBehavior), 282–283, 291, 293–294
- didRunnerCollideWithXXX() methods (collideBehavior), 294–295
- dim() method (PulseBehavior), 250–251
- dimControls() method (SnailBait), 143
- dir(), dirxml() methods (console), 38
- direction property (BodegasRevenge), 564–566
- disguiseAsSprite() method (SnailBait), 415–417
- display property (CSS), 20–22, 111, 123–134
- dissipateBubble() method (SmokingHole), 433–434
- distanceAlongTrajectory property (BodegasRevenge), 559
- div element (HTML), 18
- Do not show this warning again button, 466–467
- documentElement property (document), 378
- DOM tree, 125–126
- Donkey Kong game, 6
- double buffering, 63
- dragGameCanvas() method (SnailBait), 512
- dragging property (SnailBait), 511
- Draw collision rectangles checkbox, 478, 489
  - accessing, 489
  - HTML/CSS for, 485–486
- draw() function, 70, 76, 82, 86–87, 89
- draw() method
  - of Slider, 493
  - of SmokingHole, 416–417, 421, 428
  - of SnailBait, 97, 100, 156, 160, 393–395, 507
  - of Sprite, 150, 154–155, 158, 160, 488
  - of SpriteSheetArtist, 163–164
  - of turretArtist, 555–556
- drawBackground() function, 66, 82, 86–87
- drawBulletCanvas() method (BodegasRevenge), 581–582
- drawCollisionRectangle() method (Sprite), 487
- drawFireParticles() method (SmokingHole), 421–422

- drawImage() method (canvas context), 28, 64–67, 162–163, 559
- drawMobileDivider() method (SnailBait), 391–392
- drawMobileInstructionsXXX() methods (SnailBait), 389–394
- drawPlatform() function, 68–69, 87–90
- drawPlatforms() function, 67–69, 87–90
- drawPlatforms() method (SnailBait), 156
- drawRulerXXX() methods (SnailBait), 505–506
- drawRunner() function, 67
- drawRunner() method (SnailBait), 156
- drawSmokeBubbles() method (SmokingHole), 429
- drawSprites() method (SnailBait), 158–159
- drawText() method (canvas context), 79
- drop shadow, 453
- duck typing, 435
- duration property (sound objects), 351–352

## E

- easing functions, 233–240
  - for falling, 239
  - for gravity, 308
  - online sources for, 239
- Easter eggs, 477
- elapsed property (Stopwatch), 215, 266
- elapsed time, 235, 243–244, 255
- electricityFlowingSound object (SnailBait), 352
- emit() method (socket.io), 524–526
- emitSmokeBubble() method (SmokingHole), 431
- end game sequence, 448–455
- endLifeTransition() method (SnailBait), 327–328
- Epoch, in programming languages, 217
- equipRunner() method (SnailBait), 210–211, 299
- equipRunnerForFalling() method (SnailBait), 286, 299–300
- equipRunnerForJumping() method (SnailBait), 211–212, 217–218, 231–232, 264, 299
  - with easing functions, 239
- erase() method (Slider), 493
- eraseBulletCanvas() method (BodegasRevenge), 581



- eraseRuler() method (SnailBait), 506
- error() method (console), 36–38
- event handlers
  - blur, 105
  - change, 343, 361, 489–491
  - click, 388–389, 395–396, 455, 466–467, 536–537, 539
  - keydown, 103–105, 209–210, 327, 482–483, 564, 567
  - keypress, 535
  - keyup, 564
  - mousedown, 509–511
  - mousemove, 509–510, 512
  - mouseup, 510, 512
  - onload, 77, 98, 102–103, 139
  - touchend, 397–399, 585
  - touchmove, 584
  - touchstart, 397–399, 583
- execute() method, 8, 154–155, 159, 182–184, 186–187
  - of blueButtonDetonateBehavior, 334–335
  - of BounceBehavior, 243
  - of CellSwitchBehavior, 315–316
  - of collideBehavior, 281
  - of fallBehavior, 302–303, 348
  - of goldButtonDetonateBehavior, 469–470
  - of jumpBehavior, 212
  - of PulseBehavior, 250
  - of runBehavior, 9, 187–189
  - of SmokingHole, 431–432
  - of snailShootBehavior, 204, 349
- explode() method (SnailBait), 319–320, 349
- explodeBehavior, 332
- exploding property
  - of BodegasRevenge, 579
  - of runner, 319–320
- explosions, 313–320
  - during a fall, 302
  - of bees, 330–333
  - sound effects for, 347
- explosionSound object (SnailBait), 352
- F**
- Facebook, posting scores in, 14
- fadeInElements() method (SnailBait), 132–135, 143
  - for credits, 447, 453
  - for developer backdoor, 481
  - for ruler, 504–505
  - for running slowly warning, 462
  - for top chrome, 446
- fadeOutElements() method (SnailBait), 132–135
  - for credits, 447, 453–454
  - for developer backdoor, 482
  - for high scores display, 539
  - for ruler, 504–505
  - for running slowly warning, 462
- fall() method (runner), 299–301
- fallBehavior, 184–185, 302–308
  - adding to runner, 151
  - calculateVerticalDrop() method, 303–305
  - creating runner sprite with, 299
  - execute() method, 302–303, 348
  - fallOnPlatform() method, 305, 348
  - isOutOfPlay() method, 305
  - moveDown() method, 302–306, 308
  - pause(), unpause() methods, 308
  - processCollision() method, 470–471
  - setSpriteVelocity() method, 303–305
  - willFallBelowCurrentTrack() method, 305
- falling property (runner), 300
- falls, 298–300
  - at the end of a jump, 298, 307
  - exploding during, 302
  - pausing during, 308
  - sound effects for, 347
  - through the bottom, 326, 442
- favicon.cc, 57
- favicons, 56–58
  - generating, 57, 547–548
  - specifying in HTML, 58
- feature detection, 369
- fill() method (canvas context), 65
- fillRect() method (canvas context), 64–65, 69
- fillStyle attribute (canvas context), 66
- fillText() method (canvas context), 391–393, 506
- finishAscent() method (jumpBehavior), 221–222
- finishDescent() method (jumpBehavior), 223–224, 307
- fire particles, 417–422
  - creating, 418–421
  - drawing and updating, 421–422

- no behaviors for, 422
- randomly varied, 419
- Firefox. *See* Mozilla Firefox
- fitScreen() method (SnailBait), 378
- flip books, 71
- flyweights, 190–193, 320, 559
- frame rate (fps), 61–62
  - calculating, 77–78, 258–261
  - current, 159
  - displaying, 459
  - monitoring, 14, 40–42, 464–466
  - throttled heavily, 10–12, 108–109
- frame rate indicator, 120
  - HTML for, 121
  - updating, frequency of, 106
- fs module (Node.js)
  - appendFile() method, 526
  - readFile() method, 531
- fuzzy objects, 406

**G**

- game components. *See* behaviors
- game engines, 149
- game loop, 75–77
  - incorporating sprites into, 156–159
- game object (BodegasRevenge)
  - creating, 552
  - turretRotation property, 556, 566
- gameOver() method (SnailBait), 328, 454, 530–531
- games
  - behavior-based, 205
  - calibrating, 523
  - deploying, 540–542
  - developing new features of, 477
  - ending, 448–455
  - level generators for, 70
  - pausing, 105–106, 261–264
  - resuming, 12, 45, 105–106, 110–115, 261–264
  - revealing, 140–144
  - running slowly, 14, 458–467
  - scaling, 369–381
  - shaking, 321–323
  - starting, 77, 140–142, 357–358, 400–401, 539–540
    - on mobile devices, 376–381, 385
- gameStarted property (SnailBait), 138–140, 357

- gameTime property (TimeSystem), 268
- Gartner, 363–364
- genfavicon.com, 547
- Gertie the dinosaur, 149
- getBullet() method (BodegasRevenge), 570, 577
- getBulletLocation() method
  - (BodegasRevenge), 572
- getElapsedTime() method
  - of AnimationTimer, 234–235, 265
  - of Stopwatch, 216, 267
- getElementById() method (document), 122–123
- getFirstAvailableAudioChannel() method
  - (SnailBait), 354, 359–360
- getViewportSize() method (SnailBait), 378–379
- GIF (Graphics Interchange Format), 54
- GIMP, 51–52
- globalAlpha attribute (canvas context), 66, 69, 155
- gold button, 8
  - creating, 469
  - detonating, 4, 469
  - pace behavior of, 190–193
- goldButtonDetonateBehavior, execute()
  - method, 469–470
- Google Nexus. *See* Android
- graphics
  - immediate- vs. retained-mode, 63
  - loading, 357–358
  - manipulating, 51–52
  - obtaining, 9–10, 50–51
- graphicsReady property (SnailBait), 357–358
- gravity, 297–310
- GRAVITY\_FORCE constant, 302, 307
- group(), groupXXX() methods (console), 38
- gzip, 543

## H

- hardware acceleration, 7, 50, 290
- hasMoreSmokeBubbles() method
  - (SmokingHole), 431
- heads-up display (HUD), 481, 526–540
  - accessing, 529–530
  - creating, 534
  - displaying, 538

- heads-up display (HUD) (*cont.*)
  - hiding, 539
  - HTML/CSS for, 527–529
  - updating, 530
- height directive (viewport), 375
- height property
  - of arena, 380–381
  - of sprites, 152
- hideCredits() method (SnailBait), 453–454
- hideDeveloperBackdoor() method (SnailBait), 482–483, 504, 509
- hideHighScores() method (SnailBait), 539
- hideToast() method (SnailBait), 131, 134–135
- high scores, 526–540
  - displaying on client, 533–534, 538–539
  - hiding, 539
  - retrieving from server, 530
  - storing on server, 16, 537–538
  - validating, 536–538
- high scores display. *See* heads-up display
- highScoreNamePending property (SnailBait), 530, 532–533, 535
- high-scores.txt, 530
- hOffset property (sprites), 152–153, 158, 176, 278, 414
- href property (SnailBait), 458
- HTML (HyperText Markup Language)
  - favicons in, 58
  - including JavaScript in, 75–76, 541
  - loading audio files in, 340–341
  - namespacing in, 25
  - special symbols in, 122
- HTML5
  - on mobile devices, 364–367
  - specifications for standard components, 494
  - unpredictability of environment for, 458–459
- http module (Node.js), 520
- HTTP requests
  - reducing, 15–16, 51–52, 162, 339
  - when games are running, 539
- HUD. *See* heads-up display
- I
- illusion of depth. *See* parallax effect
- image artists, 161
- images. *See* graphics
- img element (HTML), 135
- info() method (console), 36, 38
- in-game metrics
  - recording, 523–526
  - retrieving, 519
  - storing on server, 16, 518
- initial toast
  - on mobile devices, 383
  - revealing, 142
- initializeBirdProperties() method (BodegasRevenge), 561–562
- initializeContextForMobileInstructions() method (SnailBait), 391
- initializeDeveloperBackdoorSliders() method (SnailBait), 497–498
- initializeImages() function, 76
- initializeImages() method (SnailBait), 102, 138–139
- initializeSprites() method (SnailBait), 173–174, 202, 210–211, 440–441
- initial-scale directive (viewport), 375–376
- instructions, 120
  - CSS for, 21–22
  - fading in/out, 12–13
  - for mobile devices, 381–382
  - drawing, 389–394
  - HTML for, 121–122
- instructionsElement property (SnailBait), 382
- Internet Explorer browser
  - audio formats in, 339
  - free developer tools in, 35
  - hardware acceleration in, 7
  - profiling in, 49, 289
- iOS (iPad, iPhone, iPod)
  - adding an icon to home screen on, 402
  - and preload attribute, 341
  - downloading audio files on, 400–401
  - HTML5 applications on, 364–367
  - layout viewport on, 371
  - remote debugging for, 365
  - viewport directives on, 375–376
- isAscending() method (jumpBehavior), 221–222
- isBulletInsideBird() method (BodegasRevenge), 578–579
- isBulletOutOfPlay() method (BodegasRevenge), 572

- `isCandidateForCollision()` method (`collideBehavior`), 281–282, 288
- `isDescending()` method (`jumpBehavior`), 223–224
- `isDoneAscending()` method (`jumpBehavior`), 221–222
- `isDoneDescending()` method (`jumpBehavior`), 223–224
- `isExpired()` method (`AnimationTimer`), 234, 265
- `isOutOfPlay()` method (`fallBehavior`), 305
- `isPaused()`, `isRunning()` methods
  - of `AnimationTimer`, 234, 265
  - of `Stopwatch`, 216, 267
- `isPointInPath()` method (canvas context), 65, 283, 578–579
- `isSpriteInView()` method (`SnailBait`), 157–158

## J

- JavaScript
  - constructors in, 28
  - functions in, 100
  - global variables in, 95
  - including in HTML, 75–76
  - key codes of, 104
  - live-editing in, 42, 45–46
  - methods in, 97, 100
  - minifying and obfuscating, 541–542
  - `+new Date()` construct, 75, 217, 261, 263–264, 267, 269
  - `new` operator in, 29, 552
  - objects in, 28–29
  - profiling, 49–50, 289–291
  - prototypes in, 28, 97
  - regular expressions in, 402
  - running on server, 517
  - timestamps in, 217
  - variable-length argument lists in, 133
- jewels (rubies, sapphires), 4
  - assigning values to, 440
  - bouncing, 241–245
  - colliding with, 284
    - and score, 438, 441
    - sound effects for, 347, 442
  - creating, 196–197
  - sparkling, 195–197
- JSON (JavaScript Object Notation), 30
- `JUMP_DURATION`, `JUMP_HEIGHT` constants, 219

- `jump()` method (runner), 210–214, 218, 264
- `jumpApex` property (runner), 219
- `jumpBehavior`, 184–185
  - adding to runner, 151
  - animation timers for, 231–233
  - `execute()` method, 212
  - `finishDescent()` method, 307
  - implementing, 213
  - invoking every animation frame, 214
  - pausing/unpausing, 225
  - triggers for, 212, 214
  - using game time in, 264
  - using stopwatches for, 214, 217–220, 231
- jumping property (runner), 212, 214, 219
- jumps, 208–240
  - ascending, 221–223
  - descending, 223–224
  - event handlers for, 209–210
  - falling at the end of, 298, 307
  - linear, 220–224
  - nonlinear, 229–240

## K

- keyboard
  - controlling games with, 3
    - instructions for, 120
  - handling input from, 103–105
    - during life transitions, 328, 454
- keydown event handler, 564, 567
  - disregarding during transitions, 327
  - for developer backdoor, 482–483
  - for game window, 103–105
  - for jumps, 209–210
  - vs. `addEventListener()`, 105
- keypress event handler, 535
- keyup event handler, 564
- Kindle Fire, HTML5 applications on, 367
- `knobPercent` property (`SnailBait`), 492–493, 498–499, 501

## L

- `lastAnimationFrameTime` property (`SnailBait`), 77–78, 108
- `lastSlowWarningTime` property (`SnailBait`), 463
- `lastTimeTransducerWasSet` property (`TimeSystem`), 268
- layout viewport, 371
- `left` property (sprites), 152–153, 278

- levels
  - generating, 70
  - restarting, 328–329
- linear motion, 220–224
- lineTo() method (canvas context), 506
- lineWidth attribute (canvas context), 66
- link element (HTML), 57
- listen() method (socket.io), 520–521
- Little Nemo, 149
- live-editing, 42, 45–46
- lives, 4
  - losing, 4, 325, 442, 444, 446
    - recording location of, 523–526
  - transitioning between, 17–18, 253, 323–329, 499
- lives indicator, 120, 442–448
  - CSS for, 19–20, 444
  - fading in/out, 13, 444–445
  - HTML for, 443
  - updating, 446–447
- lives property (SnailBait), 325, 446
- lives-lost.txt, 524
- loading animation, 20, 53, 549
  - fading in/out, 119, 123–132
  - generating, 548
  - HTML/CSS for, 136–138
  - implementing, 135–140
  - importance of, 117
- loadingAnimationLoaded() method (SnailBait), 139
- log() method (console), 36, 39–40
- loop attribute (HTML), 344
- loseLife() method (SnailBait), 324–326, 446
- loseOneBullet() method (BodegasRevenge), 575–577, 580–581
- lostBulletIndex property (BodegasRevenge), 558, 570, 576–577, 582

## M

- makeEaseXXXEasingFunction() methods (AnimationTimer), 232–233, 238–239, 242, 428
- margin property (CSS), 18
- Mario Bros. game, 6
- Math.random() method, 246–247
- maximum-scale directive (viewport), 375–376
- McCay, Winsor, 149
- media queries, 369

- memory consumption, 559
- Mickey Mouse, 149
- minimum-scale directive (viewport), 375
- MKS Toolkit, 542
- mobile devices, 363–403
  - adding an icon to home screen on, 402
  - changing orientation of, 378
  - detecting, 368–369
  - dragging canvas in, 397–399
  - feature detection on, 369
  - games without browser chrome on, 402–403
  - instructions for, 381–382
    - drawing, 389–394
    - HTML for, 382
  - preventing zooming in/out on, 370–371, 376, 398–399
  - resolution of, 374
  - scaling games on, 369–381
  - size of image files on, 16, 164
  - sound effects on, 400–402
  - start toast for, 394–396
  - welcome toast for, 386–388
- mobile property (SnailBait), 368
- mobileInstructionsVisible property (SnailBait), 394–396
- Module pattern, 103
- motion
  - horizontal, 65, 78–86, 104
  - linear, 220–224
  - nonlinear, 229–251, 308
  - time-based, 85, 193
- mouse vs. touch events, 399
- mousedown event handler, 509–511
- mousemove event handler, 509–510, 512
- mouseup event handler, 510, 512
- moveBehavior (Bodega's Revenge), 557–558
- moveDown() method (fallBehavior), 302–306, 308
- moveTo() method (canvas context), 506
- Mozilla Firefox browser
  - audio formats in, 339
  - free developer tools in, 35
  - hardware acceleration in, 7
  - Ogg Theora support in, 340
  - profiling in, 49, 289
- mozRequestAnimationFrame() method (window), 72

- moz-transition property (CSS). *See* transition property
- MP3 format, 339
- MPEG-4 format, 340
- music (soundtrack), 337–362
  - creating files for, 339–340
  - editing, 53
  - length of, 346–347
  - loading, 340–341
  - obtaining, 10, 52
  - pausing, 343
  - playing, 343–344
  - turning on/off, 120
- Music checkbox
  - CSS for, 21–22
  - event handler for, 343
  - fading in/out, 12
  - HTML for, 121–122
  - specifying, 342
- musicCheckboxElement() method (SnailBait), 343–344
- musicElement.currentTime property, 345–346
- musicElement() method (SnailBait), 343
- musicOn property (SnailBait), 343–344

## N

- namespaces, 25
- +new Date() construct (JavaScript), 75, 217, 261, 263–264, 267, 269
- new game button, 539–540
- new operator (JavaScript), 29, 552
- Node Inspector, 521
- Node.js, 517–521
  - fs module, 526, 531
  - http module, 520
  - installing, 517
  - package manager of, 517
  - Passport module, 517
  - preventing hangups on, 521
  - validator module, 536
- nohup command, 521
- nonlinear color changes, 247–251
- nonlinear motion
  - for bouncing, 241–245
  - for gravity, 308
  - for jumping, 229–240
- npm package manager (Node.js), 517

## O

- objects (JavaScript), 28–29
- Ogg format, 339
- Ogg Theora format, 340
- Omni Graffiti, 551
- on() method (socket.io), 524–525
- onblur property (window), 105
- onload event handler, 77, 98, 102–103, 139
- ontouchstart() method (window), 368
- opacity property (CSS), 20, 123–138
  - for credits, 451
  - for game canvas, 326–327
  - for lives indicator, 444, 446
  - for running slowly warning, 461
  - for score indicator, 440, 446
- opacity property (sprites), 152
- OPAQUE constant, 447
- Opera browser
  - audio formats in, 339
  - free developer tools in, 35
  - hardware acceleration in, 7
  - Ogg Theora support in, 340
  - profiling in, 49, 289
- o-transition property (CSS). *See* transition property

## P

- paceBehavior, 190–193, 201
- parallax effect, 87–90
- particle systems, 405–435
- paths, drawing, 64–65
- pause() method, 262–263
  - of AnimationTimer, 234, 263, 265
  - of BounceBehavior, 245
  - of fallBehavior, 308
  - of jumpBehavior, 225
  - of PulseBehavior, 249, 263
  - of SmokingHole, 434–435
  - of Stopwatch, 215, 266
- paused property
  - of SnailBait, 105–109
  - of Stopwatch, 215, 266
- paused toast, 11
- pauseStartTime property (SnailBait), 108
- pausing, 105–106
  - during falls, 308
  - when window loses focus, 10–12, 108–110
  - while playing music, 344

- performance
    - and canvas scaling, 25
    - and hardware acceleration, 7
    - and large audio files, 339–340
    - and sprite sheet cells, 167
    - monitoring, 44–49
    - of collision detection, 288–291
    - vs. simplicity, 539
  - pianoSound object (SnailBait), 352
  - picasion.com, 54, 548
  - PIXELS\_PER\_METER constant, 302, 307
  - pixels:meter ratio, calculating, 301–302
  - platformers (platform games), 3, 6
    - illusion of depth in, 87
  - platformOffset property (SnailBait), 87
  - platforms, 8
    - colliding with, 286, 298–300, 306–310
      - sound effects for, 347
    - drawing, 67–70, 160–161, 170–171
    - pulsating, 247–251
    - putting sprites on, 174
    - scrolling, 87–90
  - platformUnderneath() method (SnailBait), 306
  - Play again link, 450–451, 455
  - playAudio() method (SnailBait), 354–361
  - playing property
    - for audio, 360–361
    - for keyboard input, 327–328
    - on mobile devices, 388, 396
  - playSound() method (SnailBait), 347–350, 354, 358–361
  - playtesting
    - from the middle of a level, 16, 502
    - in slow motion, 253, 498–499
  - polar coordinates, 572–573
  - polarToCartesian() method (BodegasRevenge), 573
  - pollMusic() method (SnailBait), 345–346
  - polyfills, 72–75
  - port numbers, 522
  - position property (CSS), 20, 22
  - position property (sound objects), 351–352
  - positionSprites() method (SnailBait), 173–174
  - power-ups, 254, 477
  - preload attribute (HTML), 340–341
  - preventDefault() method (SnailBait), 397–399, 511
  - processAssetCollision() method (collideBehavior), 284–285, 349, 441–442
  - processBadGuyCollision() method (collideBehavior), 284–285, 320–321, 324–325
  - processCollision() method
    - of collideBehavior, 284, 333–334
    - of fallBehavior, 470–471
  - processPlatformCollisionDuringJump() method (collideBehavior), 285, 348
  - processXXXTap() methods (SnailBait), 398–399
  - profile(), profileEnd() methods (console), 39, 50
  - profilers, 49–50, 289–291
  - prototypes (JavaScript), 28, 97
  - pulsate property (platforms), 248
  - pulsating, 247–251
    - duration of, 248
  - PulseBehavior, 249–251
    - brighten(), dim() methods, 250–251
    - execute() method, 250
    - pause(), unpause() methods, 249, 263
    - resetTimer() method, 250
    - startPulsing() method, 250
  - putSpriteOnPlatform() method (SnailBait), 174
  - putSpriteOnTrack() method (SnailBait), 285
- ## R
- radial gradient, 138
  - random() method (Math), 246–247
  - ray casting, 275–276
  - readFile() method (fs), 531
  - readouts
    - accessing in JavaScript, 496–497
    - updating, 483–484
  - rect() method (canvas context), 65, 283
  - rectangles, drawing, 65
  - Replica Island game, 9–10, 50–52, 183, 551, 571
  - requestAnimationFrame() method (window), 71–75
  - requestNextAnimationFrame() method (window), 73–77, 385
    - waiting between calls to, 106
  - require() method (Node.js), 520–521

- reset() method
  - of AnimationTimer, 234, 265
  - of Stopwatch, 216, 267
  - of TimeSystem, 269
- resetBubble() method (SmokingHole), 433–434
- resetTimer() method (BounceBehavior), 243–244, 250
- resizeElement() method (SnailBait), 381
- resizeElementsToFitScreen() method (SnailBait), 380–381
- restartGame() method (SnailBait), 454
- restartLevel() method (SnailBait), 328–329
- restartMusic() method (SnailBait), 345–346
- restore() method (canvas context), 65, 69, 155
- resuming (after pause), 11, 105–106
  - three-second countdown for, 12, 45, 110–115
- revealBottomChrome() method (SnailBait), 143
- revealCanvas() method (SnailBait), 143
- revealCredits() method (SnailBait), 453, 458
- revealDeveloperBackdoor() method (SnailBait), 481–484, 491, 504, 509
- revealGame() method (SnailBait), 142
- revealHighScores() method (SnailBait), 534
- revealInitialToast() method (SnailBait), 142–143
- revealMobileStartToast() method (SnailBait), 396
- revealRunningSlowlyWarning() method (SnailBait), 463–464
- revealToast() method (SnailBait), 112–115, 129–131
- revealTopChromeXXX() methods (SnailBait), 143–144, 445–446
- revealWinningAnimation() method (SnailBait), 468, 471–472
- revert() method (CellSwitchBehavior), 316–317
- rotate() method (canvas context), 559
- rotating property (BodegasRevenge), 564–565
- rotation, 555–556
- rounded corners, 453
- rubies. *See* jewels
- Ruby scripts, 58–59
- ruler, 502–513
  - drawing, 505–507
  - erasing, 506
  - HTML/CSS for, 503–504
  - updating, 507
  - visibility of, 504–505
- RUN\_ANIMATION\_RATE constant, 189
- runAnimationRate property (Sprite), 187–189, 319, 328
- runBehavior, 9, 184–190
  - adding to runner, 151
  - execute() method, 9, 187–189
- runner, 8
  - ascendTimer property, 218–219, 232
  - behaviors of, 184–187
  - colliding, 282–283
    - with bats and bees, 285, 313–320, 442
    - with snail bombs, 285, 291–295
  - creating, 151, 170–171, 184–185, 317
  - descendTimer property, 218–219, 232
  - drawing, 67, 161
  - exploding, 313, 319–320
  - fall() method, 299–301
  - falling property, 300
  - horizontal position of, 307
  - JUMP\_DURATION, JUMP\_HEIGHT constants, 219
  - jump() method, 210–214, 218, 264
  - jumpApex property, 219
  - jumping property, 212, 214, 219
  - keydown event listener for, 210
  - starting animation on, 189
  - stopFalling() method, 299–300
  - stopJumping() method, 212
  - verticalLaunchPosition property, 219
- RUNNER\_LEFT constant, 67
- runnerArtist object, 9
- runnerExplodeBehavior, 184–185, 318–319
  - adding to runner, 151
- running property (Stopwatch), 215, 266
- Running slowly threshold slider, 493
  - event handler for, 498
- running slowly warning, 14, 455, 458–467, 548, 550
  - enabling/disabling, 489–490
  - event handlers for, 466–467



running slowly warning (*cont.*)  
 HTML/CSS for, 460–462  
 initial invisibility of, 462  
 modifying threshold of, 489  
 runningSlowlyThreshold property  
 (SnailBait), 463

## S

Safari browser  
 audio/video formats in, 339–340  
 debugging iOS games in, 365  
 free developer tools in, 35  
 hardware acceleration in, 7  
 profiling in, 49, 289  
 Samsung Galaxy. *See* Android  
 sapphires. *See* jewels  
 save() method (canvas context), 65, 69, 155  
 score indicator, 120  
 CSS for, 19–20, 439–440  
 fading in/out, 13, 445  
 HTML for, 121, 438–439  
 initial invisibility of, 440  
 updating, 441  
 score property (SnailBait), 440, 532  
 scores, 4–5, 438–442  
 incrementing, 438, 441–442  
 posting in social networks, 14–15,  
 455–458  
 storing on server, 16, 537–538  
 script element (HTML), 519  
 security, preventing breaches of, 536  
 seekAudio() method (SnailBait), 354,  
 359–360  
 self reference (JavaScript), 103  
 Separating Axis Theorem, 277, 289  
 servers  
 connecting to, 522  
 creating, 520  
 creating sockets on, 520–521  
 debugging on, 521  
 emitting messages from, 531, 533  
 processing messages on, 525, 532  
 running, 521  
 updating files on, 537–538  
 uploading files to, 542–543  
 validating messages on, 537–538  
 setBackgroundOffset() function, 85–90  
 setBackgroundOffset() method  
 (SnailBait), 175–176

setBubbleSpriteProperties() method  
 (SmokingHole), 426  
 setInitialSmokeBubbleColor() method  
 (SmokingHole), 425, 427  
 setInterval() function, 70–71  
 for interface effects, 106  
 for playing music, 345–346  
 setOffsets() function, 89  
 setOffsets() method (SnailBait), 175  
 setPlatformXXX() functions, 89–90  
 setSpriteOffsets() method (SnailBait),  
 176, 414  
 setSpriteValues() method (SnailBait),  
 440–441  
 setSpriteVelocity() method  
 (fallBehavior), 303–305  
 setTimeout() function, 70–71, 74  
 for interface effects, 106, 321–328  
 for transitions, 125–127, 131  
 setTimeRate() method (SnailBait), 256,  
 259–260, 499–500  
 setTransducer() method (TimeSystem), 255,  
 259–260, 269–270  
 shake() method (SnailBait), 321–323  
 shim, shiv. *See* polyfills  
 shooting property (BodegasRevenge), 563,  
 567–569  
 Show how to use the controls link, 383,  
 388–389, 396  
 showCollisionRectangle property (Sprite),  
 488–489  
 showSlowWarning property (SnailBait), 463,  
 467, 490  
 showSmokingHoles property (SnailBait),  
 491  
 side-scroller games, 6  
 sliders, 492–501  
 accessing, 496  
 creating, 497  
 HTML/CSS for, 494–496  
 initializing, 497–498  
 manual redrawing, 500  
 separate JavaScript file for, 492–493  
 updating, 483–484  
 before revealing the backdoor, 500–501  
 smoke bubbles, 422–434  
 creating, 424–428  
 dissipating, 432–434  
 drawing and updating, 428–429

- emitting, 430–431
- expanding, speed of, 428
- Smoke checkbox, 491
  - accessing, 490
  - HTML/CSS for, 486
- smoking holes, 406–417
  - and performance, 49–50
  - creating, 412
  - defining, 411
  - disguising as sprites, 413, 415–417
  - JavaScript file for, 409–410
  - pausing, 434–435
  - scrolling, 413–414
  - showing/hiding, 490–491
- SmokingHole object
  - addBehaviors() method, 417, 430–431
  - addSpriteMethods() method, 416, 421
  - addSpriteProperties() method, 415–416
  - advanceCursor() method, 431
  - createBubbleArtist() method, 427
  - createBubbleSprite() method, 425–426
  - createBubbleSpriteTimer() method, 428, 435
  - createDissipateBubbleBehavior() method, 432
  - createFireParticle() method, 419–420
  - createFireParticleArtist() method, 420–421
  - createFireParticles() method, 418–419
  - createSmokeBubbles() method, 424–425
  - dissipateBubble() method, 433–434
  - draw() method, 416–417, 421, 428
  - drawFireParticles() method, 421–422
  - drawSmokeBubbles() method, 429
  - emitSmokeBubble() method, 431
  - execute() method, 431–432
  - hasMoreSmokeBubbles() method, 431
  - pause(), unpause() methods, 434–435
  - resetBubble() method, 433–434
  - setBubbleSpriteProperties() method, 426
  - setInitialSmokeBubbleColor() method, 425, 427
  - update() method, 416–417, 428–430
  - updateSmokeBubbles() method, 429
- snail
  - arming with bombs, 202–203
  - creating, 201
  - pace behavior of, 190, 201
  - Snail Bait, 3–543
    - characters of, 4, 7–10
    - code statistics of, 18
    - elements of, 3
    - HTML for, 22–27, 75–76
    - JavaScript for, 27–28
    - keyboard controlling of, 3
    - playing online, 3
    - stripped-down version of, 547
  - snail bombs, 4
    - colliding with, 285, 291–295
    - creating, 202
    - moving, 204
    - shooting, 203–204
      - sound effects for, 347
  - snail shooting sequence, 199–205
- SnailBait object
  - constructor of, 95–97
  - prototype of, 97–100
  - See also individual methods and properties*
- snailBait reference, 103
- snailBombMoveBehavior, 202–204
- snailShootBehavior, 203–204
  - execute() method, 204, 349
- social features, 14–15, 455–458
- socket.io, 517–526
  - connect() method, 522
  - creating sockets with, 520–521
  - emit() method, 524–526
  - including in Snail Bait, 518–520, 523
  - installing, 517
  - listen() method, 520–521
  - on() method, 524–525
- sockets
  - creating, 520–521
  - opening on clients, 522
- Sonic the Hedgehog game, 6
- Sound checkbox, 361–362
  - CSS for, 21–22
  - event handler for, 362
  - fading in/out, 12
  - HTML for, 121–122
  - specifying, 342
- sound effects, 337–362
  - creating files for, 339–340
  - editing, 53
  - loading, 340–341, 357–358
  - multichannel, 353–361
  - obtaining, 10, 52

- sound effects (*cont.*)
  - on mobile devices, 400–402
  - playing, 347–350, 358–361
  - turning on/off, 120, 361–362
- sound objects
  - creating, 352
  - properties of, 351–353
- soundclick.com, 52
- soundLoaded() method (SnailBait), 354, 357
- soundOn property (SnailBait), 361
- soundtrack. *See* music
- spatial partitioning, 288–289
- special effects, 320–329
- splitCSV() method (String), 532
- sprite artists. *See* artists
- sprite behaviors. *See* behaviors
- sprite containers, 411
- Sprite object, 150, 153
  - calculateCollisionRectangle() method, 278–279, 487
  - draw() method, 154–155, 488
  - drawCollisionRectangle() method, 487
  - runAnimationRate property, 187–189
  - showCollisionRectangle property, 488–489
  - update() method, 154–155, 157, 159, 186
- sprite sheet artists, 150, 162–164
- sprite sheet inspector, 165–167
- sprite sheets
  - cells of, 164–167, 314–317
  - for audio, 52–53
  - loading, 357–358
  - on mobile devices, 16, 164
  - single for all images, 15–16, 51–52
  - transparent background for, 551
- spriteOffset property (SnailBait), 174–175
- sprites, 7–10, 147–176
  - collision margins of, 278–279, 287
  - creating, 9, 150, 168–171
  - data-driven, 70
  - defining with metadata, 171–174
  - disguising JavaScript objects as, 415–417
  - drawing, 150, 158
    - from a sprite sheet, 162
  - in view, 157–158
  - incorporating into game loop, 156–159
  - properties of, 152
  - putting on platforms, 174
  - scrolling, 152–153, 174–176
  - setting offsets of, 176
  - specifying type of, 150
  - updating, 157, 186
    - based on current time, 159
- sprites array (SnailBait), 156–159, 167–171, 409
  - adding:
    - runner to, 171
    - smoking holes to, 413
  - defining, 167
  - iterating over, 168
- SpriteSheetArtist object, 163
- spriteSheetLoaded() method (SnailBait), 358, 400–401
- SQL injections, 536
- src attribute (HTML)
  - of img, 135, 139
  - of script, 519
- start game sequence, 117, 139–144
  - on mobile devices, 385, 400–401
- Start link
  - event handler for, 395–396
  - playing coin sound for, 400
- Start the game link, 388
- start toast, 548–549
  - for mobile devices, 394–396
- start() method
  - of AnimationTimer, 234, 264
  - of Stopwatch, 215, 266
  - of TimeSystem, 255, 269
- startBouncing() method (BounceBehavior), 243–244
- startDraggingGameCanvas() method (SnailBait), 510–511
- startGame() function, 76–77
- startGame() method (SnailBait), 140–142, 257–258, 354, 357–358, 385
- STARTING\_BACKGROUND\_OFFSET constant, 502
- startLifeTransition() method (SnailBait), 325–328
- startMusic() method (SnailBait), 345–346
- startPause property (Stopwatch), 215, 266
- startPulsing() method (PulseBehavior), 250
- startTime property (Stopwatch), 215, 266
- startToastTransition() method (SnailBait), 130–131, 134–135

- stop() method
    - of AnimationTimer, 234, 265
    - of Stopwatch, 215, 266
  - stopDraggingGameCanvas() method (SnailBait), 512
  - stopFalling() method (runner), 299–300
  - stopJumping() method (runner), 212
  - Stopwatch object, 214–217
    - elapsed, paused, running, startXXX, totalPausedTime properties, 215, 266
    - getElapsedTime() method, 216, 267
    - isPaused(), isRunning(), reset() methods, 216, 267
    - pause(), start(), stop() methods, 215, 266
    - unpause() method, 216, 266
  - stopwatches, 214–217
    - pausing, 225–227
    - redefining current time for, 217, 265–267
    - resuming, 217, 225–227
    - using for jumps, 214, 217–220, 231
  - Strategy design pattern, 9, 187
  - String.splitCSV() method, 532
  - stroke and fill artists, 160–161
  - stroke() method (canvas context), 65
  - strokeRect() method (canvas context), 64–65, 69
  - strokeStyle attribute (canvas context), 66
  - style change events, 126
  - SVG (Scalable Vector Graphics), 63
  - switchCells() method (CellSwitchBehavior), 316–317
- T**
- target property (CSS), 457
  - Texas Instruments 9918A video-display processor, 149
  - textAlign, textBaseline attributes (canvas context), 391
  - this reference (JavaScript), 97, 100–103
    - changing in the middle of a method, 102
  - thudSound object (SnailBait), 352
  - time, 207–270
    - current, 75, 159, 258–259
    - redefining, 217, 264
    - elapsed, 235, 243–244
    - modifying, 255
    - of last animation frame, 108, 159
  - time animations, 214–217
  - time rate
    - modifying, 17, 231, 253, 259–260
    - in developer backdoor, 478
    - setting, 256, 260, 498–500
    - slowing during transitions, 17, 253, 327, 499
  - Time rate slider, 498–500
  - time systems, 17, 253–270
    - creating, 257
    - implementing, 268–270
    - pausing/resuming games with, 261–264
    - starting, 257–258
    - using, 258–264
  - time(), timeEnd() methods (console), 39
  - time-based motion, 85, 193
  - timeline(), timelineEnd() methods (console), 39, 45–46
  - timelines, 44–49
    - starting/stopping programmatically, 45
  - timer property (TimeSystem), 268
  - timeStamp() method (console), 39
  - TimeSystem object
    - calculateGameTime() method, 255, 258, 268–270
    - gameTime, lastTimeTransducerWasSet, timer, transducer properties, 268
    - reset() method, 269
    - setTransducer() method, 255, 269–270
    - start() method, 255, 269
  - TOAST\_TRANSITION\_DURATION constant, 132, 134–135
  - toasts, 12, 111–112
    - CSS for, 128–129
    - fading in/out, 12, 112, 123–132
    - HTML/CSS for, 111
  - togglePaused() method (SnailBait), 105–115, 225, 261–262
    - for music, 344
  - togglePausedStateOfAllBehaviors() method (SnailBait), 225–227, 262
  - top chrome. *See* score indicator, lives indicator
  - top property (sprites), 152, 278
  - totalPausedTime property (Stopwatch), 215, 266
  - touch vs. mouse events, 399
  - touchend event handler, 397–399, 585
  - touchEnd() method (SnailBait), 398
  - touchmove event handler, 584

touchMove() method (SnailBait), 399  
 touchstart event handler, 397–399, 583  
 touchStart() method (SnailBait), 397, 399  
 track property, 304  
 trajectory property (BodegasRevenge), 559  
 transducer functions, 255–257  
     duration of, 270  
 transducer property (TimeSystem), 268  
 transition property (CSS), 123, 125, 128–129  
     versions of, 129  
 transitions. *See* CSS3 transitions  
 translate() method (canvas context), 65,  
     80–81, 559  
 TRANSPARENT constant, 447  
 triggers, 315  
 turnXXX() functions, 86  
 turnXXX() methods (SnailBait), 104,  
     189–190  
 turret, 553–556  
     artist for, 554–555  
     behaviors of, 564–570  
     creating, 553–556  
     direction property, 564–566  
     drawing, 555–556  
     rotating property, 564–566  
     shooting property, 563, 567–569  
 turretArtist.draw() method, 555  
 turretRotation property (game), 556, 566  
 Tweet my score link, 14–15, 450–451, 455–458  
     accessing in JavaScript, 457–458  
     HTML for, 457  
 TWEET\_EPILOGUE, TWEET\_PREAMBLE constants,  
     457–458  
 Twitter  
     posting scores in, 14–15  
     Web Intents, 455–458  
 type property (sprites), 152

## U

unPause() method, 262–263  
     of AnimationTimer, 234, 263, 265  
     of BounceBehavior, 245  
     of fallBehavior, 308  
     of jumpBehavior, 225  
     of PulseBehavior, 249, 263  
     of SmokingHole, 434–435  
     of Stopwatch, 216, 266  
 Unreal Engine, 149  
 update() method

    of SmokingHole, 416–417, 428–430  
     of Sprite, 154–155, 157, 159, 186  
 updateDeveloperBackdoorCheckboxes()  
     method (SnailBait), 483–484, 491  
 updateDeveloperBackdoorReadouts()  
     method (SnailBait), 483–484  
 updateDeveloperBackdoorSliders()  
     method (SnailBait), 483–484, 500–501  
 updateLivesElement() method (SnailBait),  
     446–447  
 updateRunningSlowlySlider() method  
     (SnailBait), 501  
 updateScoreElement() method (SnailBait),  
     441  
 updateSmokeBubbles() method  
     (SmokingHole), 429  
 updateSpeedSamples() method (SnailBait),  
     466  
 updateSprites() method (SnailBait),  
     157–159  
 updateTimeRateSlider() method  
     (SnailBait), 501  
 user interface (UI), 437–472  
     focusing attention on, 12–13, 445  
     latency of, 106  
 user-scalable directive (viewport), 375–376

## V

validator module (Node.js), 536–538  
 velocity of falling objects, 301  
     initial, 306–307  
 velocityX, velocityY properties (sprites),  
     152  
 verticalLaunchPosition property (runner),  
     219  
 video formats, 340  
 viewport meta tag, 373–376  
 @viewport rule (CSS), 376  
 visibility property (CSS), 128  
 visible property (sprites), 152  
 visible viewport, 371  
     getting width and height of, 378  
 volume property (sound objects), 351–353  
 VP8 format, 340

## W

Warn when running slowly checkbox,  
     489–491  
     accessing, 490

- HTML/CSS for, 485–486
- warn() method (console), 36–37, 39
- WASD keys, 7
- Web Audio API, 339
- web browsers. *See* browsers
- Web Intents, 455–458
- WebGL (Web Graphics Library), 290
- WebKit, 64
- webkitRequestAnimationFrame() method (window), 72
- webkit-transition property (CSS). *See* transition property
- webpages
  - background of, 54–56
  - scaling to fit, 371–372
- welcome toast, 384–389
  - HTML for, 386–388
- width directive (viewport), 374–376
- width property
  - of arena, 380–381
  - of sprites, 152
- willFallBelowCurrentTrack() method (fallBehavior), 305

- window object
  - addEventListener() method, 105, 209–210, 327
  - for developer backdoor, 483
  - on size or orientation changes, 378
  - keydown event handler, 564, 567
  - keyup event handler, 564
  - ontouchstart() method, 368
- windowHasFocus property (SnailBait), 113–115
- Windows, running UNIX scripts on, 542
- winning animation, 4–5, 467–472
  - revealing, 471

## X

- XSS (cross-site scripting), 536

## Y

- YUI Compressor, 540

## Z

- z-index property (CSS), 111, 455