RETRO WEB-BASED GAME PROGRAMMING

.

Retro Web-Based Game Programming

Second Edition

*K*enwright

BOOK TITLE:
Retro Web-Based Game Programming
Second Edition
ISBN-13: 979-8-642-07831-0

Edition: 003004020

Table of Contents

Preface **xi**

1 Introduction **1**
1.1 Internet . . . 1
1.2 Web Browsers (simple web pages to complex interactive ***video game*** solutions) . . . 1
1.3 Compatibility (Chrome, Firefox, 360 Secure Browser, ..) . . . 2
1.4 2D Games (HTML/CSS) . . . 2
1.5 Document Object Model (DOM) . . . 2
1.5.1 Recusing the Hierarchy . . . 4
1.6 No Canvas . . . 4
1.7 No Engines (Native Javascript) . . . 5
1.8 Libraries (jQuery and Ajax) . . . 5
1.8.1 Example, Waiting for the DOM to be loaded . . . 5

2 Game Mechanics **7**
2.1 What is a game? . . . 7
2.2 Win or lose . . . 7
2.2.1 Keeping players coming back for more . . . 7
2.2.2 Fun Factor . . . 8
2.3 Types of games (mazes, scrolling, ..) . . . 8
2.4 Essential Concepts . . . 8

3 CSS and HTML **11**
3.1 Visual aspects . . . 11
3.2 HTML is the content . . . 12
3.3 Classes and IDs . . . 12
3.3.1 ID or Class . . . 13
3.4 CSS Animations . . . 13
3.4.1 CSS @ Symbol . . . 13
3.4.2 @keyframes **Rule** . . . 14
3.5 Position (Relative, Fixed, Absolute) . . . 14
3.5.1 CSS Position and Helper Properties . . . 15
3.5.2 Static . . . 15

3.5.3 Relative . . . 16
3.5.4 Absolute . . . 17
3.5.5 Fixed . . . 18
3.5.6 Sticky . . . 19
3.6 Transforms . . . 19
3.7 Position (Translation) and Size . . . 19
3.8 Rotation . . . 20
3.8.1 Rotation around (predefined point) . . . 20
3.8.2 Direction and Rotation . . . 21
3.9 Depth (Z-Index) . . . 21
3.10 Sprite Sheets . . . 21
3.10.1 Showing Part of an Image . . . 21
3.10.2 Animating the Sprite Sheet . . . 22
3.11 CSS or Javascript . . . 22

4 Javascript Basics **23**
4.1 Creating CSS elements on the fly (CreateElement) . . . 23
4.2 Generate HTML for the page (document.write) . . . 24
4.3 Setting/Getting CSS Style information in Javascript . . . 25
4.4 Variables let, var (or const) . . . 25
4.5 Types of Variables . . . 27
4.5.1 null . . . 28
4.5.2 undefined . . . 28
4.5.3 boolean . . . 28
4.5.4 number . . . 28
4.5.5 string . . . 28
4.5.6 symbol . . . 29
4.5.7 objects . . . 29
4.6 typeof . . . 29
4.7 Mutable or Immutable Data Types . . . 30
4.7.1 Copy Array (or Object) . . . 30
4.7.2 Pass-by-Value or Pass-by-Reference . . . 31
4.8 Conditional logic . . . 31
4.9 Loops . . . 33
4.9.1 For loop . . . 33
4.9.2 While loop . . . 33
4.9.3 Do while loop . . . 34
4.10 Functions . . . 34
4.10.1 Modularize - functions to contain task . . . 34
4.11 Classes and Objects . . . 35
4.11.1 Four ways to define a Javascript object (class) . . . 35
4.11.2 1. Function definition . . . 35
4.11.3 2. Object literals . . . 36

4.11.4 3. Singleton using a function . . . 37
4.11.5 4. Classes (class keyword) . . . 38
4.12 `use strict;` . . . 38
4.13 debug assert (console) . . . 39
4.13.1 Console.log isn't available? . . . 39
4.14 Visually Debugging . . . 40
4.15 Creating Elements . . . 40
4.16 Debug Draw . . . 40
4.17 Debug Output . . . 40
4.18 Drawing Lines without Canvas . . . 41
4.19 User input . . . 43
4.19.1 Mouse input . . . 44
4.19.2 Keyboard Input . . . 44
4.19.3 Reacting to Key Events . . . 45
4.20 Timer Events (Timing Information) . . . 45
4.21 Events and Elements . . . 46
4.22 Game Loop . . . 46
4.22.1 requestAnimationFrame vs setInterval (or setTimeout) . 46
4.22.2 Document Object Model (Ready?) . . . 47
4.22.3 Timing Problems . . . 47
4.23 Rotating Elements to Face a Particular Direction . . . 47

5 Javascript Advanced 51
5.1 Data (JSON) . . . 51
5.1.1 Javasript JSON helper functions . . . 51
5.2 Pausing and Continuing Animations (Overriding CSS) . . . 52
5.3 Rotating and Animating (with CSS) . . . 52
5.4 Rotating Programmatically . . . 53
5.5 Memory Management . . . 54

6 Collisions and Intersections 55
6.1 Collision vs Intersection . . . 55
6.2 Elements On Screen (Screen-Box-Area) . . . 55
6.3 Mouse-Element Check (Point in Box) . . . 55
6.4 Axis-Aligned Bounding Box (AABB) . . . 56
6.5 Sphere-Sphere . . . 56
6.6 Shape Hierarchies/Decomposition . . . 58
6.7 Problems . . . 58

7 Simulations and Physics 59
7.1 Newton's Laws (Basics) . . . 60
7.2 Relationship between position, velocity and acceleration . . . 60
7.3 Forces to Control Movement . . . 61

7.4 Vectors (2-dimensional movement) 61
7.5 Time Step (Fixed) 63
7.6 Problems 64

8 Menus **65**
8.1 Press Start (or Splash Screen) 65
8.2 Popup news (toaster, queue, popup dialog or information) . . . 65
8.3 Basic Menu 66
8.4 Advanced Menu System 68

9 Projects **69**

10 Project: Duck Shoot (Extreme) **71**
10.1 Game Mechanics 71
10.2 Implementation 71
10.3 Enhancements 77

11 Project: Catapult (Sprout) **79**
11.1 Game Mechanics 79
11.2 Implementation 79
11.3 Enhancements 94

12 Project: Ship Scroller **95**
12.1 Game Mechanics 95
12.2 Implementation 95
12.3 Enhancements 103

13 Project: Breakout **105**
13.1 Mechanics 105
13.2 Implementation 105
13.3 Enhancements 113

14 Project: Tile **115**
14.1 Mechanics 115
14.2 Implementation 115
14.3 Enhancements 121

15 Project: Mission Command **123**
15.1 Mechanics 123
15.2 Implementation 123
15.3 Enhancements 150

Preface

About this Book

As anyone who's ever spent hours hunched over Asteroids or Candy Crush can attest, there's something special about retro video games. Sure they're fun, but they can also be absorbing, frustrating, challenging and complex. **Retro Web-Based Game Development** is an introductory text to help you get started developing your own interactive 2D games for the web. The book explains various concepts, such as, advanced CSS animation techniques and popular collision detection algorithms to enable you to create a variety of video game styles (board games to scrolling shooters). One thing is abundantly clear, game-based projects are an effective ways to nurture your web-based skills - creativity, critical thinking, communication and collaborative thought. Hence, this book is crafted to guide you through interactive web-based techniques using retro game projects, packed with resources for gaming gurus and hobbyists alike.

The **Retro Web-Based Gaming Book** contains:

- Complete Game Projects
- Coding Examples
- Debugging Advice
- Game Mechanics
- Collision Detection
- Physics
- User Interfaces
- Optimization Tips

Retro Web-Based Game Development is more than just creating games - it also teaching you the mechanics and principles to developing interactive and engaging browser solutions

Who is this Book for?

This book is designed for anyone who wants to get started developing interactive and dynamic web pages (using game-based examples as a spring-board). You will learn how to design and build fully-responsive and interactive web-based games that are both fun and dynamic (and extensible). The book introduces basic concepts and features, from responsive web design through to the latest Javascript, HTML and CSS technologies.

Examples: **Academics** The book would provide insightful examples and material to help ***teachers***, ***instructors*** or anyone involved in ***education*** and ***learning*** to develop bespoke ***interactive*** learning solutions (e.g., game-based projects to ***teach students*** mathematics, physics or programming principles in a creative and fun way)

Hobbiests The book offers multiple projects to help beginners master the topic of web technologies by implementing and enhancing simple self contained ***retro games*** (fun factor).

Web-Artists/Designers The book provides information and insights on how to stretch what the capabilities of websites, e.g., ***programmatically*** alter the content on the fly, interact and explore web content in new and interesting ways and more.

This book will open your mind to new ideas, while giving you the opportunity to acquire new skills and extensive knowledge. The material is practical based enabling you to take a hands-on approach to creating demos/and working solutions that you can use in the real-world (i.e., not just theory).

Why Learn Retro Game Programming?

One of the best parts of this book, is that developing **your own retro games** is incredibly **fun**. You'll certainly run into the same pitfalls that you face in regular development: times of frustration, not knowing how to proceed, and even feeling like you have to fight against the tools you're using. However, developing retro games with HTML/CSS/Javascript lends itself to an entertaining development process. You get to play and test your game as you go, and there's nothing like the feeling of finally having enough of your game made that you get to share it with others (through a browser).

Hopefully, after this book, you'll be convinced that retro game development with web technologies is for you. If you're wondering where to start, this book is for you. We'll take you through the core concepts and provide practical (simplified) examples for you to work through. It does have a somewhat steep learning curve, but it is incredibly flexible and powerful, and learning problem solve and develop various solutions with it will enable you to create practically any game you'd ever want to.

Prerequisites

While the material in this text is introductory, it assumes you to have some background programming knowledge (for example the ability to understand basic mathematical concepts and conditional logic). All of the demos and examples are explained, not to mention, available for you to run and test out. So they could be used in conjunction with other learning material (e.g., while you're learning Web-Design, ***Javascript***, ***HTML*** and ***CSS***).

Book Text Convention

Code samples will be given in blocks (highlighted and easy to identify and copy/type out):

```
var a = 2; // assign variable `a` the value 2
let b = 3;

// This is a one line comment
/*
   This is a multi-line comment
*/
```

Also throughout the book, various tips and ***comments*** worth emphasizing are highlighted:

Tip: Get into the habit of **commenting** your scripts.

Retro Gaming with Web-Based Technologies (HTML, CSS, Javascript).

1 Introduction

This Chapter provides an introduction to web-based development technologies. The history and the advantages of web-based development, both for video games, and other interactive applications.

1.1 Internet

The internet has revolutionized your world both with its technical ***capabilities*** and the ability to connect with billions of other users across the world. While the majority of the text focuses on browser based video game solutions, the internet is so much more. Especially, now that ***internet*** access is becoming popular on ***mobile*** phones (mobile apps), not to mention, the ***internet-of-things*** (IoT), social media and streaming content.

1.2 Web Browsers (simple web pages to complex interactive *video game* solutions)

The ***web***, in terms of user experience and functionality has changed enormously over the past decade, thanks to HTML5 and CSS3 and the evolution of web browsers (technologies).

The advantages of browser based solutions, such as, games, offers greater reach, accessibility from multiple device types and benefits from connecting with search, social media, email and affiliates. The shackles of limited features, user experience (UX) and functionality has disappeared. The combination of new web technologies, and extended Javascript resources, means you have the ability to tap into your device features, like location, camera, not to mention, the ability to work offline, so you can deliver interactive games with great user-experience - all via a web browsers.

As a simple example ***vibrate*** feature on a mobile phone:

```
window.navigator.vibrate(200); // vibrate for 200ms
window.navigator.vibrate([100, 30, 100, 30, 100, 30, 200, 30,
    ↪ 200, 30, 200, 30, 100, 30, 100, 30, 100]); // Vibrate 'SOS
    ↪ ' in Morse.
```

In Javascript, if your device supports the vibrate hardware and is enabled in your browser, then the `Navigator.vibrate(..)` method pulses the vibration hardware on the device. Of course, if the device doesn't support vibration, this method has no effect.

1.3 Compatibility (Chrome, Firefox, 360 Secure Browser, ..)

Always worth testing your retro games on different browsers. ***Cross browser testing*** important! If your retro game is not tested and debugged on different platforms and browsers, it won't work the same on all of them, causing inconvenience and sadness to your players.

While you might thing you'll need to download and install every single browser since the beginning of time - a good place to start testing for cross browser issues, is to lookup which browsers support which features.

For example, if you want to check your current browser's capabilities, there are automated solutions available.

Online tools, such as, `https://html5test.com/`, test which features your browser support and evaluating your **browser's ranking**.

1.4 2D Games (HTML/CSS)

Today people are overloaded with content and there is a trend for ever more realistic interactive 3D graphics, so you might be wondering if ***2D games*** are out dated? Boring? NO! A 2D game can be just as fun, engaging and exciting as a high-end 3D commercial game.

Previously, web developers relied mostly on Flash to create and deploy games and animations on the web - an expensive undertaking. Now with the latest HTML and CSS web-based games and animations can be both powerful and attractive while being easy to develop. As you'll learn in this book, it's relatively painless to create stunning interactive games with only HTML, CSS, and JavaScript (in fact it's fun).

1.5 Document Object Model (DOM)

The ***Document Object Model (DOM)*** is one of the most popular web-based concepts you need to master. The ***DOM*** is efficient (in terms of memory), and easy to work with, offering amenities like click event handlers and visual elements. It's relatively straightforward to create/remove and access objects in the DOM hierarchy (e.g., both through html and dynamically using Javascript).

For example, this is what a simple HTML web page DOM might look like:

```
document
  |
  +-root <html>
      |
      +-- element <head>
             +- element <title>
                    +- text "my title"
             ...
      +- element <body>
             +- element <h1>
                    +- text "a heading"
             +- element <a>
                    +- text "link text"
             ...
```

The equivalent html file for this DOM hierarchy would look like:

```
<html>
  <head>
     <title>my title</title>
  </head>
  <body>
     <h1>a heading</h1>
     <a>link text</a>
  </body>
</html>
```

You need to notice certain elements, such as `document` which contains the `root`. You'll always have access to the `document`, which means, you'll always be able to search and access any elements in the hierarchy. You'll give your elements unique `IDs` and `classNames` so that you're able to identify them.

To ***create***, ***add***, ***remove***, and ***find*** elements in yourfrom the DOM hierarchy on the fly (in Javascript) you have access to the following functions:

- `createElement(..)`
- `appendChild(..)`
- `removeChild(...)`
- `getElementById(...)`

Example (Adding element to the DOM hierarchy)

```
var node = document.createElement("div"); // Create<div> node
node.id = "myid"; // set the unique ID for the node
node.className = "someClass"; // associate the node with a class
document.body.appendChild( node ); // add the node element to the
    ↪  DOM hierarchy
```

Example (Removing element DOM hierarchy)

```
// any time we want the element, we can find it using its unique
    ↪ ID
var mydiv = document.getElementById("myid"); // Get the <div>
    ↪ element
mydiv.parentNode.removeChild( mydiv ); // Remove the element if
    ↪ we need to from the hierarchy
```

1.5.1 Recusing the Hierarchy

Each element in the ***hierarchy*** is connect (i.e., it knows who its parent and how it's children are). Using the **parentNode** and **childNodes** member variables you can at any time crawl or recurse over the Document Object elements.

As you become more familiar with the DOM API, you'll notice there is a `parentNode` and a `parentElement` accessor. What is the difference? The documentation says that `parentElement` is only for listing connected **elements** , however, you might ask, can objects in the DOM hierarchy not be elements? The `parentElement` property returns the **element** parent, while `parentNode` returns **any node** parent. These properties are usually the same: they both get the parent.

With the one exception of `document.documentElement`:

```
alert( document.documentElement.parentNode ); // document
alert( document.documentElement.parentElement ); // null
```

Similarly there are **childElements** and **childNodes** accessor functions.

1.6 No Canvas

The games and concepts in this text utilize the Document Object Model (DOM) and browser capabilities (leverage CSS/HTML features). You won't be using ***canvas***. The HTML ***canvas*** element is for low-level graphical control, such as, pixel-by-pixel drawing. Canvas offers finer grained control over rendering but comes at the cost of having to manage every detail manually, like hover state animations. Your biggest reason to use canvas is when you start to develop detailed graphical effects (on the fly) in the browsereded. However, for the retro games you'll develop here, you'll not need canvas, and will use graphical components like gifs and jpgs for visual details not to mention style sheet animations/effects (CSS).

1.7 No Engines (Native Javascript)

The focus of this book is on developing interactive retro games from the ground up using Native Javascript/CSS/HTML, and not external game ***engines***/pre-written game-frameworks. You'll learn about the low-level mechanics, collision detection algorithms and optimization techniques - implement them as you need them for your game.

This provides you with a more comprehensive understanding, not to mention more flexibility (and power).

1.8 Libraries (jQuery and Ajax)

AJAX is an acronym standing for **Asynchronous JavaScript and XML** and this technology will help you load data from servers without a browser page refresh (e.g., upload and download high score data on the fly without needing to refresh the page). As you start to master game development with web-technologies, you'll find Ajax an invaluable tool for any online communication (multiplayer or downloading/uploading information). The ***AJAX*** syntax (when used with ***jQuery***) is just Javascript (and is greatly simplified, e.g., an AJAX call can be a single line - with jQuery):

```
$.ajax({ url: "test.html", success: function(){/*do stuff here*/
    ↪ }});
```

JQuery is a great tool which provides a rich set of methods to develop next generation web application (compact syntax).

While not covered in this book, it's still worth trying out other libraries and frameworks, once you've mastered the basic gaming concepts. You might want to explore integrating some of your game programming concepts with **React** and **Angular**.

The important thing to remember when developing your games, is that you understand the concepts, and can implement them either in vanilla Javascript or another language (i.e., won't be dependent on a specific library like jQuery).

1.8.1 Example, Waiting for the DOM to be loaded

The jQuery way:

```
$(document).ready(() => {
    //...
})
```

The Javascript/DOM way:

```
document.addEventListener("DOMContentLoaded", () => {
  //...
})
```