HTML5 による 2D ゲームの開発: Snail Bait の紹介

初のプラットフォーム・テレビ・ゲームの開発を始める

David Geary
Author and speaker
Clarity Training, Inc.

2012年 9月 27日

この連載では、HTML5 のエキスパートである David Geary が、HTML5 で 2D テレビ・ゲームを実装する方法について順を追って説明します。今回はその第 1 回として、完成したゲームを紹介した後で、このゲームの実装をゼロの状態から開始します。HTML5 でゲームを作成したいと思いながらも、その詳細のすべてをマスターする時間がなかった読者にとって、この連載記事は最適です。

このシリーズの他の記事を見る

ソフトウェア開発の素晴らしい点は、常識の範囲内で、想像できるものはどんなものでも画面上で実現できることです。ソフトウェア開発者は、他の分野の技術者の妨げとなっているような物理的な制約によって妨げられることがないことから、長年、グラフィックス API や UI ツールキットを使用して創造的で魅力的なアプリケーションを実装してきました。ソフトウェア開発のジャンルのうち、最も創造的なジャンルがゲームのプログラミングであることは間違いありません。創造性の観点で言うと、ゲームに対して思い描いていたビジョンを実現することほど努力が実る作業はほとんどありません。

プラットフォーム・テレビ・ゲーム

ドンキーコング、マリオブラザーズ、ソニック・ザ・ヘッジホッグ、ブレイドは、どれも有名なベストセラーのゲームであり、どれもプラットフォーム・ゲームです。プラットフォーム・ゲームは一時、テレビ・ゲームの販売額全体の 3分の1 にも達したことがあります。現在、市場でのプラットフォーム・ゲームのシェアは大幅に低くなっていますが、成功しているプラットフォーム・ゲームは今でもたくさんあります。

ただし、「実る」という言葉は「容易である」ということを意味するのではなく、実際には逆です。ゲームを実装するには、プログラミングを十分に理解すること、グラフィックスやアニメーションを深く理解すること、そして大量の数学的知識を豊かな芸術性や創造性と融合することが求められます。これは、テレビ・ゲームの場合はなおさらです。しかもそれは出発点にすぎません。成功を収めているゲーム開発者は、ゲームプレイとは関係の無いさまざまな側面 (スコアボード、操作説明、残機が増減したときのアニメーションやステージが変わったときのアニメーション、ゲーム終了シーケンスなど) の実装のみならず、ゲームプレイやグラフィックスを精緻化することでゲームをより完成度の高いものにするために、多くの時間を費やしています。

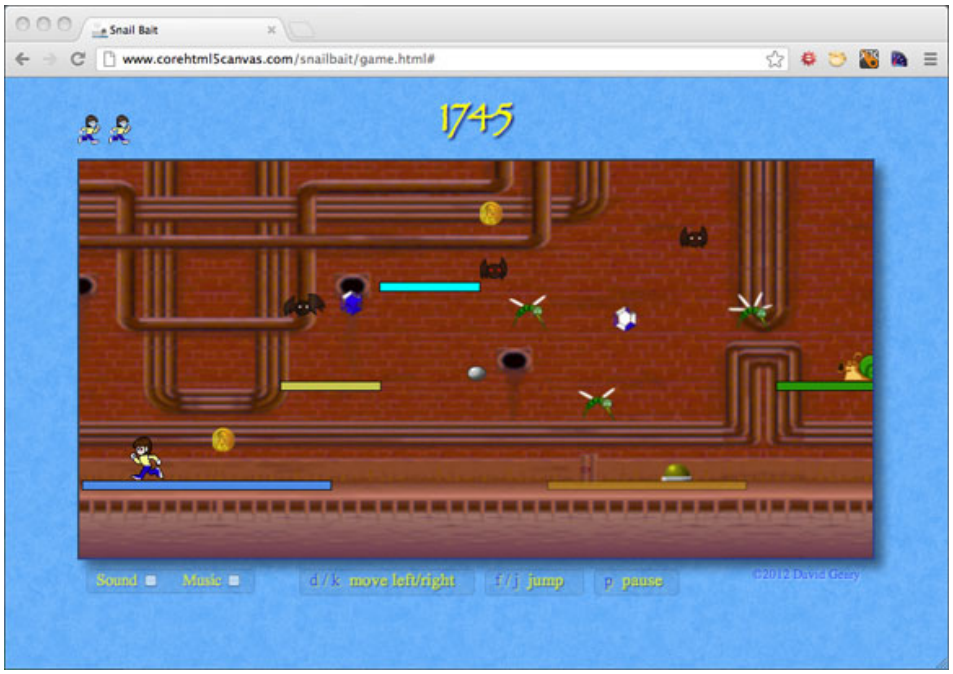
この連載の目標は、HTML5 によるテレビ・ゲームの実装方法を説明し、皆さん自身がゲームの実装を始められるようにすることです。

「この連載について」の動画でしゃべっている内容を見るには、ここをクリックしてください。

この連載で取り上げるゲーム: Snail Bait

この連載では、主に HTML5 の Canvas API を使用してプラットフォーム・テレビ・ゲームを実装する方法について説明します。そこで取り上げるゲームは、図 1 に示す「Snail Bait」です。このゲームはオンラインでプレイすることができます (「参考文献」でゲームへのリンクを参照してください)。皆さんのブラウザーに Canvas 用のハードウェア・アクセラレーション機能があることを確認してください (この機能はごく最近、Chrome バージョン 18 以降をはじめとする、ほとんどのブラウザーに実装されました)。この機能がない場合、Snail Bait の実行速度は非常に遅くなります (囲み記事「HTML5 の Canvas のパフォーマンス」を参照してください)。

図 1. Chrome で Snail Bait を実行している様子



Snail Bait に使用されている HTML5 技術

- Canvas (2D API)
- スクリプト・ベースのアニメーションのためのタイミング制御
- audio 要素
- CSS3 (トランジションとメディア・クエリー)

Snail Bait は古典的なプラットフォーム・ゲームです。主人公 (ここでは単純にランナーと呼ぶことにします) は水平方向に移動する宙に浮いたプラットフォームに沿って走ったり、プラットフォームとプラットフォームの間をジャンプしたりします。ランナーにとっての最終的な目標は、そのステージの最終段階である、金色のボタンの付いた振動するプラットフォームに到達することです。ランナー、振動するプラットフォーム、金色のボタンを示したものが図 1 です。

プレイヤーはキーボードでランナーを操作します。ランナーは「d」で左に移動し、「k」で右に移動し、「j」または「f」でジャンプします。また、「p」を押すとゲームが一時中断します。

プレイヤーはゲーム開始時に残機を 3 つ持っています。図 1 を見るとわかるように、ゲームのキャンバスの左上には残機の数を示すランナー・アイコンが表示されます。ランナーがステージの最後まで到達する冒険の中では、蜂、コウモリ、カタツムリなどの敵を避けなければならない一方、コイン、ルビー、サファイアなどの価値あるアイテムをキャッチしなければなりません。ランナーが敵にぶつかると、ランナーは爆発して命を失い、そのステージの始めに戻らなければなりません。ランナーがアイテムにぶつかると、スコアが加算され、ご褒美として心地よい効果音が鳴ります。

WASD とは？

コンピューター・ゲームでは慣習により、プレイの操作に「w」、「a」、「s」、「d」といったキーを使用することが多くあります。この慣習が生まれた主な理由は、これらのキーを使用すると右利きのプレイヤーがマウスとキーボードを同時に使用できるからです。また、これらのキーを使用すると右手が自由になり、スペースバーや、「CTRL」、「ALT」などの修飾キーを押すことができます。Snail Bait ではマウスや修飾キーからの入力を受けとらないため、WASD を使用していません。しかし、ゲームのコードを変更して任意のキーの組み合わせを使用できるようするのは簡単です。

敵はほとんどの場合、ランナーが自分達にぶつかるのを待って、うろついているだけです。ただしカタツムリは一定周期でカタツムリ爆弾 (図 1 で中央近くに表示されている銀色の玉) を発射します。他の敵と同じように、このカタツムリ爆弾がランナーにぶつかると、ランナーは爆発します。

ゲームが終了するのは、プレイヤーが残機をすべて失った場合、または振動するプラットフォームに到達した場合のいずれかです (金色のボタンの上に着地するとボーナス・ポイントが付きます)。どちらの場合もゲームの終了時には図 2 のようなクレジットが表示されます。

図 2. ゲームのクレジット

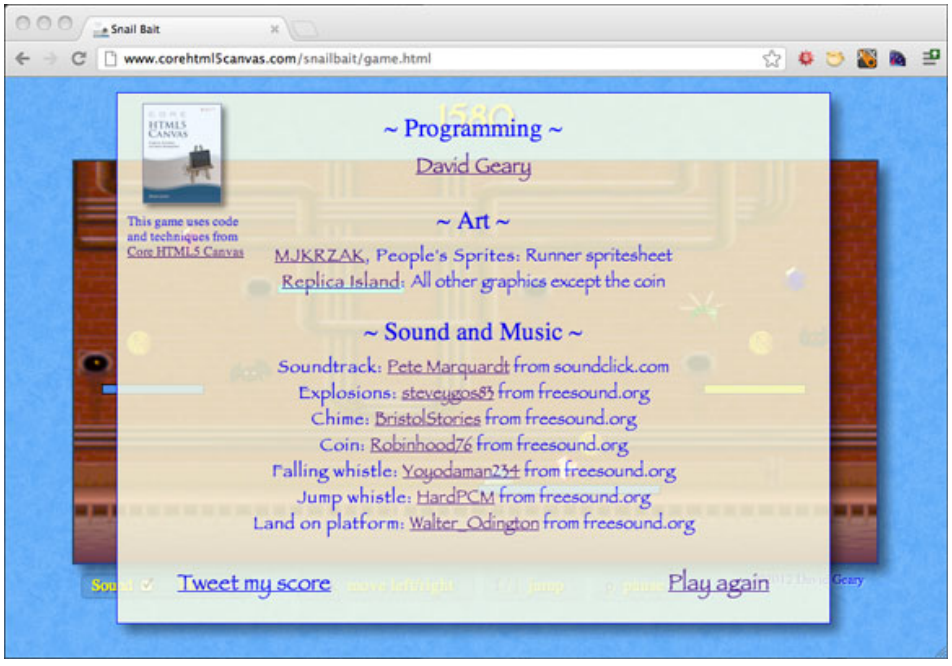


図 1 を見てもわかりませんが、ユーザーが動きを操作するランナーを除き、すべてのものが常にスクロールしています。このスクロール動作から、Snail Bait の種類をさらに細かく分類すると、水平スクロール・プラットフォーム・ゲームに分類することができます。ただし、このゲームにおける動きはスクロールのみではありません。そのため、スプライトやスプライトのビヘイビアも使用しています。

スプライト: 登場するキャラクター

HTML5 の Canvas のパフォーマンス

最近まで、ほとんどのブラウザーは CSS トランジション用にハードウェア・アクセラレーションを実装していましたが、Canvas 用のハードウェア・アクセラレーションは実装していませんでした。Canvas での動作は従来から比較的高速で、特に SVG (Scalable Vector Graphics) のような他のグラフィックス・システムと比べると高速でしたが、ハードウェア・アクセラレーションなしの Canvas はハードウェア・アクセラレーションを使用する他のシステムにはとてもかないません。

最近のブラウザーはすべて、Canvas 要素にハードウェア・アクセラレーションを使用しており、iOS 5 も同様です。つまり、アニメーションがスムーズな Canvas ベースのテレビ・ゲームが、今やデスクトップ・マシンのみならず Apple のモバイル機器でも可能になったのです。

背景を除き、Snail Bait ではすべてのものがスプライトです。スプライトはゲームのキャンバスに描画できるオブジェクトで、Canvas API の一部ではありませんが、実装するのは簡単です。このゲームのスプライトは以下のとおりです。

- プラットフォーム (生物ではないオブジェクト)
- ランナー (主人公)
- 蜂とコウモリ (敵)
- ボタン (アイテム)
- ルビーとサファイア (アイテム)
- コイン (アイテム)
- カタツムリ (敵)
- カタツムリ爆弾 (敵)

右から左にスクロールする他、このゲームのスプライトのほとんどすべてには、独立した独自の動きがあります。例えば、ルビーとサファイアは上下に動きますが、その速さは毎回同じではありません。またボタンとカタツムリは自分達が配置されたプラットフォーム上をゆっくりと行ったり来たりします。

レプリカアイランド

スプライトのビヘイビアという概念は、Strategy デザイン・パターンの一例であり、オープンソースの Android プラットフォーム・ゲームとして人気のレプリカアイランドに発想を得ています。Snail Bait のグラフィックスの大部分はレプリカアイランドのものを (使用許可を得た上で) 使用しています。「参考文献」で、ウィキペディアの「Strategy パターン」へのリンクとレプリカアイランドのホームページへのリンクを参照してください。

このような独立した動きは、数あるスプライトのビヘイビアの 1 つです。スプライトは、この動きとはまったく関係のないビヘイビアを持つことが可能です。例えば、ルビーとサファイアは上下に動く他、きらきらと輝きます。

各スプライトには、ビヘイビアの配列があります。ビヘイビアは、`execute()` メソッドを持つオブジェクトにすぎません。このゲームでは、すべてのアニメーション・フレームで各ビヘイビアの `execute()` メソッドを呼び出します。そのメソッドの中で、ビヘイビアはゲームの状況に応じて、そのビヘイビアに関連するスプライトを何らかの方法で操作します。例えば、「k」を押してランナーを右に移動すると、ランナーの「横に移動する」ビヘイビアにより、すべてのアニメーション・フレームでランナーは右に移動し、それはランナーの移動する方向が変更されるまで続きます。「足踏みする」という別のビヘイビアでは、周期的にランナーの画像を変更し、あたかもランナーが足踏みしているかのように見せます。この 2 つのビヘイビアを組み合わせると、ランナーはあたかも左方向または右方向に走っているように見えます。

このゲームのスプライトと各スプライトのビヘイビアを一覧にしたものが表 1 です。

表 1. Snail Bait のスプライトとそのビヘイビア

スプライト	ビヘイビア
プラットフォーム	• 水平方向に移動する (ランナーとカタツムリ爆弾以外のスプライトはすべてプラットフォームと一緒に移動します)
ランナー	• 足踏みする • 横に移動する • ジャンプする • 落下する • 敵にぶつかって爆発する • アイテムにぶつかってポイントを獲得する
蜂とコウモリ	• 空中に浮かぶ • 羽ばたきする
ボタン	• 横にゆっくり動く • つぶれる • 状況によって異なり、敵を爆発させるか、そのステージを終了する
コイン、ルビー、サファイア	• きらきら輝く • 上下に動く • 横にゆっくり動く
カタツムリ	• 横にゆっくり動く • 爆弾を発射する
カタツムリ爆弾	• (プラットフォームよりも速く) 右から左に移動する • ランナーにぶつかって消える

スプライトとスプライトのビヘイビアについての詳細は連載の今後の記事で説明しますが、ここでは差し当たり、大まかな概要を説明するために、このゲームで「`#### (runner)`」スプライトを作成する方法をリスト 1 に示します。

リスト 1. スプライトを作成する

```
var runInPlace = {  // Just an object with an execute method
   execute: function (sprite, time, fps) {
      // Update the sprite's attributes based on the time and frame rate
   }
};

var runner = new Sprite('runner', // name
                        runnerPainter, // painter
                        [ runInPlace,... ]); // behaviors
```

このリストでは、`runInPlace` オブジェクトが定義され、他のビヘイビアを含む配列に入れられて、「ランナー」スプライトのコンストラクターに渡されています。ゲームが実行されている間は、すべてのアニメーション・フレームで `runInPlace` オブジェクトの `execute()` メソッドが呼び出されます。

HTML5 のゲームを開発する場合のベスト・プラクティス

無料で入手可能なアセット

ほとんどのゲーム開発者は、グラフィックスや効果音、音楽に関して何らかの支援を必要とします。幸いなことに、さまざまなライセンスの下で、無料で入手できるアセットが豊富にあります。

Snail Bait では以下のアセットを使用しています。

- freesound.org の効果音
- soundclick.com のサウンドトラック
- panelmonkey.org のランナー・スプライト (このサイトはハッキングされてしまいました)
- 他のグラフィックスはすべてレプリカアイランドのものです。

この連載ではゲームの開発に関するベスト・プラクティスについて説明しますが、最初に HTML5 特有のベスト・プラクティスとして以下の 5 点について説明します。

- ウィンドウがフォーカスを失った場合にはゲームを一時中断する
- ウィンドウが再度フォーカスを得た場合にはカウントダウンを実行する
- CSS3 トランジションを使用する
- 実行速度が遅いゲームを検出し、それに対応する
- ソーシャル機能を組み込む

この 5 点について連載の今後の記事で詳細に説明しますが、ここでは 5 点のそれぞれを簡単に見ていきます。

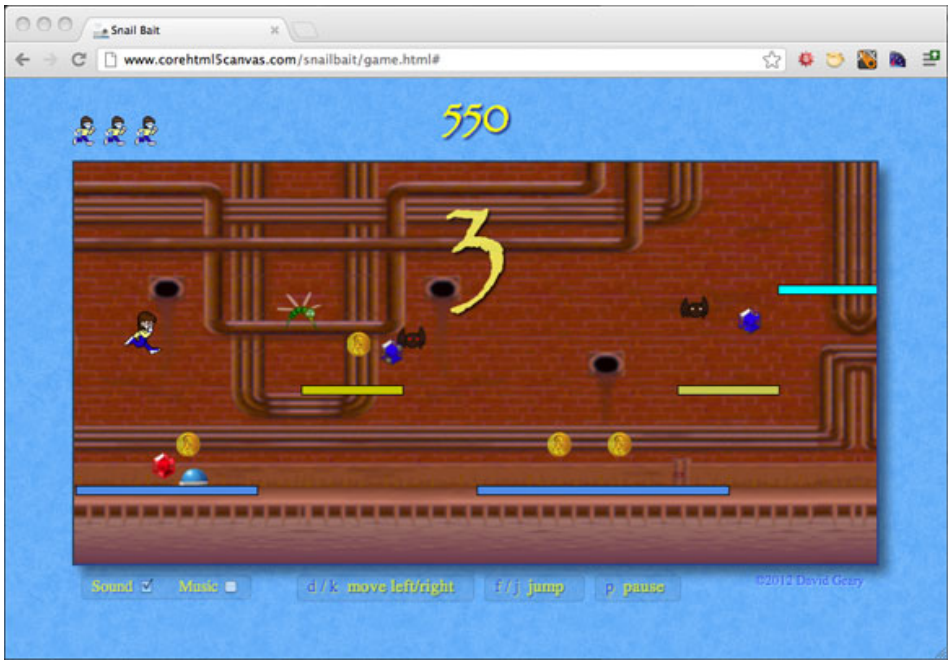
1. ウィンドウがフォーカスを失った場合にはゲームを一時中断する

ブラウザーで HTML5 のゲームを実行している時に別のタブや別のブラウザー・ウィンドウにフォーカスを移すと、ほとんどのブラウザーはゲームのアニメーションを実行するフレーム・レートを大幅に下げ、CPU やバッテリーなどのリソースを節約しようとします。このフレーム・レート制限により、ほぼ間違いなく衝突検出アルゴリズムが破綻します。これは衝突検出アルゴリズムが「ゲームはある値以上のフレーム・レートで実行される」ことを想定しているためです。フレーム・レートが制限されて衝突検出アルゴリズムが破綻するのを防ぐために、ウィンドウがフォーカスを失った場合には自動的にゲームを一時中断する必要があります。

2. ウィンドウが再度フォーカスを得た場合にはカウントダウンを実行する

ゲームのウィンドウが再度フォーカスを得た場合には、ユーザーがゲーム再開の準備をできるように、何秒間かの余裕を与えるのが望ましい考えです。Snail Bait では、ウィンドウが再度フォーカスを得た場合には図 3 に示すように 3 秒間のカウントダウンを使用します。

図 3. Snail Bait が自動的に一時中断している様子



3. CSS3 トランジションを使用する

図 4 はゲームがロードされた時のスクリーン・ショットです。

図 4. CSS3 のエフェクト

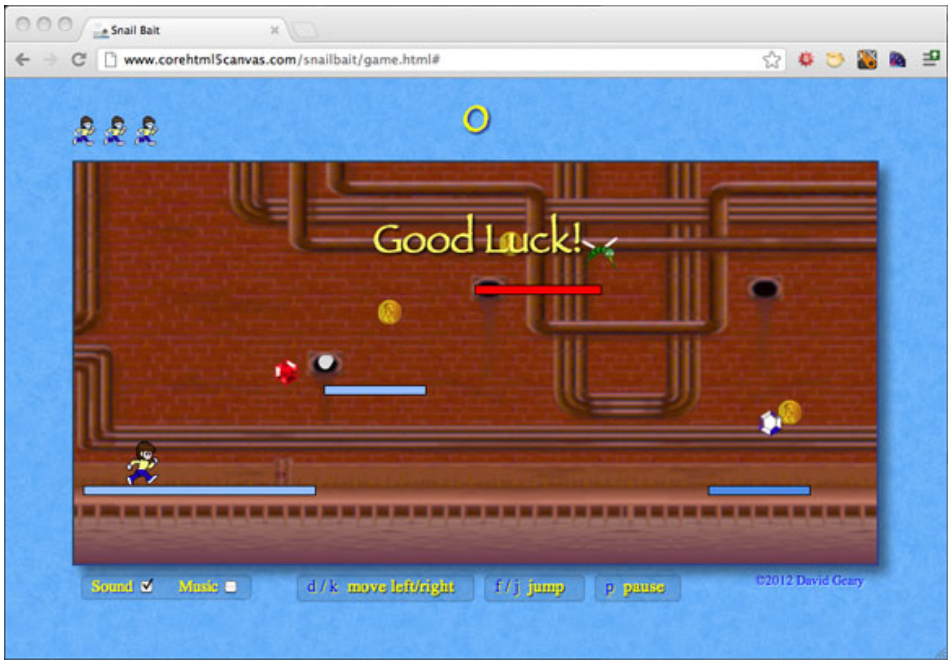


図 4 で注意する点が 2 つあります。第 1 に、ゲーム開始のメッセージ (プレイヤーに対して短時間表示されます) として「Good Luck!」が表示されています。このメッセージはゲームがロードされるとフェードインし、5 秒後にフェードアウトします。第 2 に、ゲームのキャンバスの下に

「Sound (音)」と「Music (音楽)」のためのチェックボックスと、(どのキーを押すと、どの機能が実行されるのかを説明した) 操作説明が表示されていることに注意してください。ゲーム開始時にはチェックボックスと操作説明は図 4 のように完全に見えていますが、プレイが開始されると (図 3 に示すように) これらの要素はゆっくりとフェードアウトしてほとんど見えなくなり、目障りになりません。

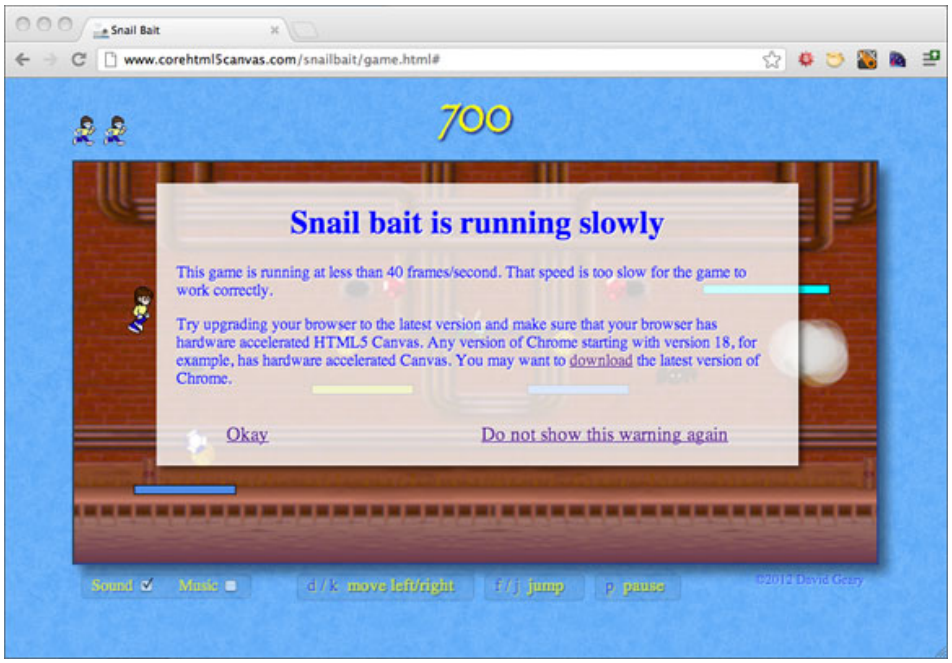
Snail Bait は CSS3 トランジションを使用して要素の明るさを下げ、メッセージの表示をフェードアウトしています。

4. 実行速度が遅いゲームを検出し、それに対応する

厳密に制御された環境で実行されるコンソール・ゲームとは異なり、HTML5 のゲームは変動性が高く、予測困難で混乱に満ちた環境で実行されます。プレイヤーが別のタブで YouTube の動画を再生している場合や、そうでなくても CPU や GPU に過度の負荷をかけている場合には、許容できないほどゲームの実行速度が遅くなることは珍しくありません。また、プレイヤーが使用しているブラウザーの能力がゲームの要求を満たすことができないブラウザーが使用される可能性も十分にあります。

皆さんはゲーム開発者として、そうした不幸な事態の組み合わせを想定し、それに対応する必要があります。Snail Bait は常にフレーム・レートを監視し、特定のしきい値を下回ることが度々あって何秒もの間続いたことを検出すると、実行速度が遅いというメッセージを図 5 のように表示します。

図 5. フレーム・レートが遅いことを検出したときの表示

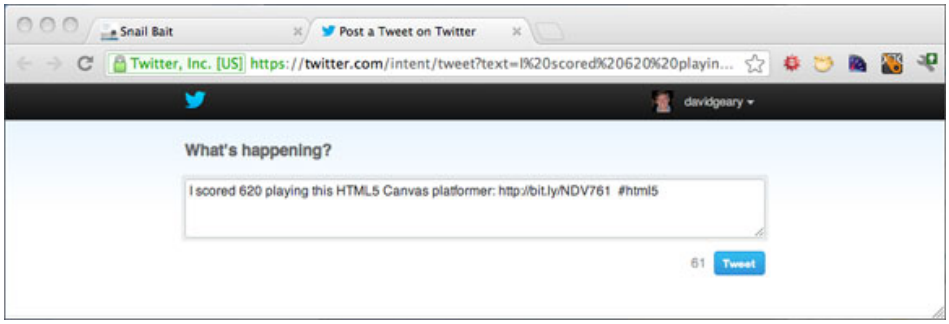


5. ソーシャル機能を組み込む

成功しているゲームのほとんどすべてには、Twitter や Facebook にスコアを投稿するなどのソーシャルな側面が組み込まれています。Snail Bait のプレイヤーが、ゲーム終了時に表示される

「Tweet my score (スコアをツイートする)」リンクをクリックすると、Snail Bait は別のタブで Twitter にアクセスし、そのスコアを発表する図 7 のようなツイートを自動的に作成します。

図 7. ツイートのテキスト



これでゲームの概要を理解できたので、今度は少しコードを見ていきましょう。

Snail Bait の HTML と CSS

Snail Bait のコード統計

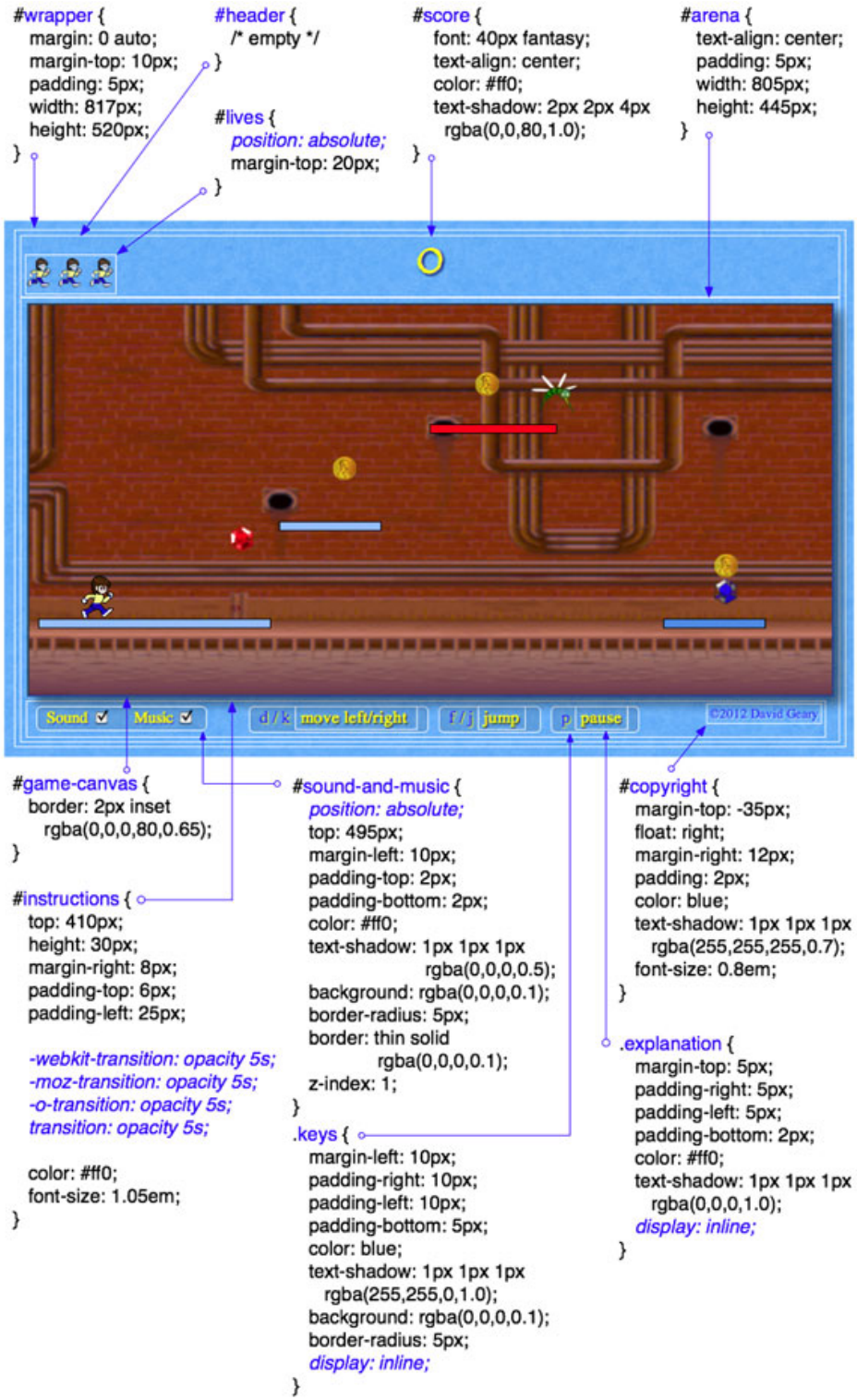
コードの行数

- HTML: 276
- CSS: 410
- JavaScript: 3,898

Snail Bait は HTML、CSS、JavaScript で実装されていますが、囲み記事「Snail Bait のコード統計」を見るとわかるように、コードの大半は JavaScript です。実際、この連載の今後の記事では主に JavaScript を取り上げ、HTMLと CSS3 については時折触れるのみとなります。

図 8 は、このゲーム本体に関わる HTML 要素とそれらに対応する CSS を示しており、メッセージやクレジットといった他の要素の HTML と CSS は省略しています。

図 8. このゲームの HTML と CSS (box-shadow プロパティーは省略)



CSS の大部分は特に注目すべきものではありませんが、図 8 では注目に値する属性のいくつかを青字の斜体で示してあります。第 1 に (青字の斜体にはしてありませんが)、`wrapper` 要素の `margin` 属性を「`0 auto`」に設定しています。この設定により、ラッパーとラッパー内にあるすべてのものがウィンドウ内で横方向に中央揃えされます。第 2 に、`lives` 要素と `sound-and-`

music 要素の位置が「absolute」になっています。これらの要素がデフォルトの位置 (relative) のままにされると、これらの DIV はキャンバスの幅にまで拡大され、近くにある要素 (それぞれ score と instructions) はこれらの DIV の下に移動することになってしまいます。最後に、keys と explanation という CSS クラスでは、関連付けられた要素を同じ行に配置するために、display 属性の値を「inline」に設定しています。

リスト 2 に図 8 の CSS を示します。

リスト 2. game.css (抜粋)

```
#arena {
   text-align: center;
   padding: 5px;
   width: 805px;
   height: 445px;
}

#copyright {
   margin-top: -35px;
   float: right;
   margin-right: 12px;
   padding: 2px;
   color: blue;
   text-shadow: 1px 1px 1px rgba(255,255,255,0.7);
   font-size: 0.8em;
}

.explanation {
   color: #ff0;
   text-shadow: 1px 1px 1px rgba(0,0,0,1.0);
   display: inline;
   margin-top: 5px;
   padding-right: 5px;
   padding-left: 5px;
   padding-bottom: 2px;
}

#game-canvas {
   border: 2px inset rgba(0,0,80,0.62);
   -webkit-box-shadow: rgba(0,0,0,0.5) 8px 8px 16px;
   -moz-box-shadow: rgba(0,0,0,0.5) 8px 8px 16px;
   -o-box-shadow: rgba(0,0,0,0.5) 8px 8px 16px;
   box-shadow: rgba(0,0,0,0.5) 8px 8px 16px;
}

#instructions {
   height: 30px;
   margin-right: 8px;
   padding-top: 6px;
   padding-left: 25px;

   -webkit-transition: opacity 2s;
   -moz-transition: opacity 2s;
   -o-transition: opacity 2s;
   transition: opacity 2s;

   color: #ff0;
   font-size: 1.05em;
   opacity: 1.0;
}

.keys {
   color: blue;
   text-shadow: 1px 1px 1px rgba(255,255,0,1.0);
```

```
   background: rgba(0,0,0,0.1);
   border: thin solid rgba(0,0,0,0.20);
   border-radius: 5px;
   margin-left: 10px;
   padding-right: 10px;
   padding-left: 10px;
   padding-bottom: 5px;
   display: inline;
}

#sound-and-music {
   position: absolute;
   top: 495px;
   margin-left: 10px;
   color: #ff0;
   text-shadow: 1px 1px 1px rgba(0,0,0,0.5);
   background: rgba(0,0,0,0.1);
   border-radius: 5px;
   border: thin solid rgba(0,0,0,0.20);
   padding-top: 2px;
   padding-bottom: 2px;
   z-index: 1;
}

#wrapper {
   margin: 0 auto;
   margin-top: 20px;
   padding: 5px;
   width: 817px;
   height: 520px;
}
```

図 8 の HTML を示すリスト 3 を見るとわかるように、このゲームの HTML には大量の `DIV` と 1 つの canvas 要素、そしていくつかの画像とチェックボックスがあることがわかります。

リスト 3. game.html (抜粋)

```
<!DOCTYPE html>
<html>
   <!-- Head.....................................................-->

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

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

   <body>
      <!-- Wrapper.................................................-->

      <div id='wrapper'>
         <!-- Header..................................................-->

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

            <div id='score'>0</div>
            <div id='fps'></div>
         </div>

         <!-- Arena...................................................-->
```

```
        <div id='arena'>
           <!-- The game canvas.....................................-->

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

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

           <div id='sound-and-music'>
              <div class='checkbox-div'>
                 Sound <input id='sound-checkbox'
                              type='checkbox' checked/>
              </div>

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

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

           <div id='instructions'>
              <div class='keys'>
                 d / k

                 <div class='explanation'>
                    move left/right
                 </div>
              </div>

              <div class='keys'>
                 f / j

                 <div class='explanation'>
                    jump
                 </div>
              </div>

              <div class='keys'>
                 p

                 <div class='explanation'>
                    pause

                 </div>
              </div>
           </div>

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

           <div id='copyright'> ©2012 David Geary</div>
        </div>
     </div>

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

     <script src='js/stopwatch.js'></script>
     <script src='js/animationTimer.js'></script>
     <script src='js/sprites.js'></script>
     <script src='js/requestNextAnimationFrame.js'></script>
     <script src='js/behaviors/bounce.js'></script>
     <script src='js/behaviors/cycle.js'></script>
     <script src='js/behaviors/pulse.js'></script>
     <script src='game.js'></script>
```

```
  </body>
</html>
```

すべてのアクションは canvas 要素で実行されます。このキャンバスには何よりもまず、2D ゲームを実装するための強力な API を備えた 2D コンテキストがあります。canvas 要素内のテキストは、ブラウザーが HTML5 の Canvas をサポートしない場合にのみブラウザーに表示されるフォールバック・テキストです。

このゲームの HTML と CSS に関して最後に触れておくこととして、キャンバスの幅と高さが canvas 要素の width 属性と height 属性で指定されていることに注意してください。これらの属性は、canvas 要素に囲まれた描画サーフェスのサイズと canvas 要素のサイズの両方に関係しています。

一方、CSS を使用して canvas 要素の幅と高さを設定すると、canvas 要素のサイズしか設定されません。描画サーフェスの幅と高さはそれぞれ 300 ピクセルと 150 ピクセルというデフォルトの値のままです。これはつまり、ほぼ間違いなく canvas 要素のサイズと canvas 要素の描画サーフェスのサイズが一致せず、その場合はブラウザーが canvas 要素に合わせて描画サーフェスのサイズを調整する、ということです。ほとんどの場合、それは望ましいことではないので、CSS によって canvas 要素のサイズを設定するのは避けた方が無難です。

小さなキャンバスに描画し、CSS によって拡大する

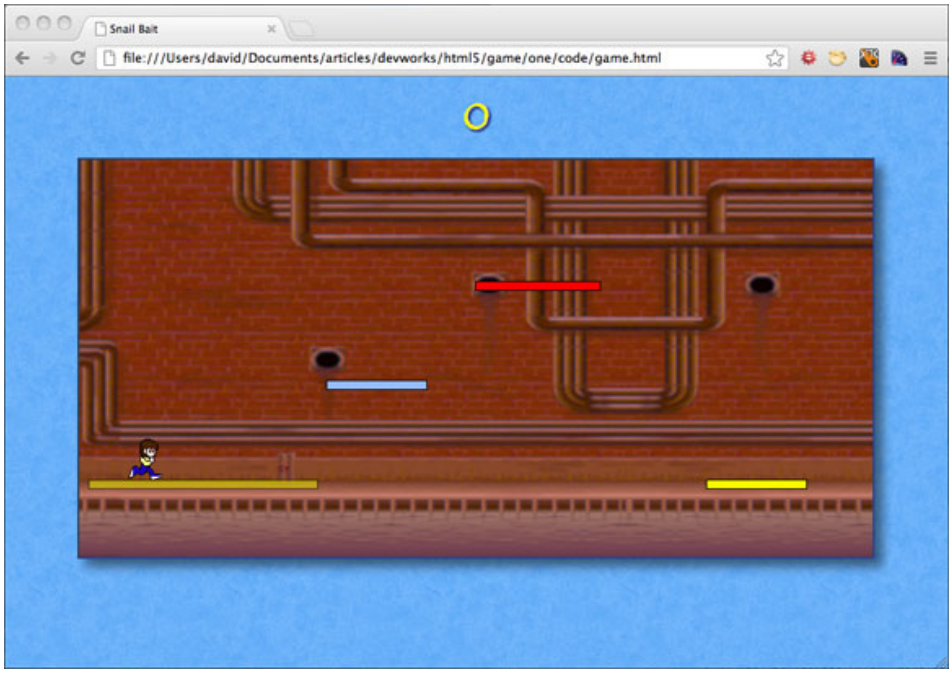
一部のゲームは意図的に小さなキャンバスに描画し、CSS を使用してプレイ可能なサイズにキャンバスを拡大しています。こうすることで、キャンバス内の数多くのピクセルを操作するということがなくなり、パフォーマンスが向上します。また通常は CSS によるキャンバス拡大はハードウェア・アクセラレーターによって行われるため、拡大処理のコストは最小限で済みます。しかし現在は、最近のブラウザーの最新バージョンのほぼすべてが Canvas 用のハードウェア・アクセラレーション機能を備えているため、ほとんどの場合、フルサイズのキャンバスを描画しても速さはほとんど同じです。

パルプ・フィクションなどの優れた映画と同様、既に話の結末がわかりました。では最初に戻りましょう。

Snail Bait の簡素な始まりの部分

図 9 はこのゲームの始まるところを示しており、単純に背景、プラットフォーム、ランナーを描画しています。始めは、プラットフォームとランナーはスプライトではなく、ゲームのプログラムによって直接これらが描画されます。背景とランナーを作成するコードを入手するには「ダウンロード」を参照してください。

図 9. 背景とランナーを描画する



このゲームの始めの部分の HTML をリスト 3 に示します。この部分はリスト 2 の HTML を簡略化したものにすぎません。

リスト 3. game.html (ゲームの始めの部分)

```
<!DOCTYPE html>
<html>
   <!-- Head..................................................-->

   <head>
      <title>Snail Bait</title>
      <link rel='stylesheet' href='game.css'/>
   </head>

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

   <body>
      <!-- Wrapper..............................................-->

      <div id='wrapper'>
         <!-- Header..............................................-->

         <div id='header'>
            <div id='score'>0</div>
         </div>

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

         <div id='arena'>
            <!-- The game canvas.....................................-->

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

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

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

リスト 4 にゲームの始めの部分の JavaScript を示します。

リスト 4. game.js (ゲームの始めの部分)

```
// --------------------------- DECLARATIONS ---------------------------

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

   // Constants.....................................................

   PLATFORM_HEIGHT = 8,
   PLATFORM_STROKE_WIDTH = 2,
   PLATFORM_STROKE_STYLE = 'rgb(0,0,0)',

   STARTING_RUNNER_LEFT = 50,
   STARTING_RUNNER_TRACK = 1,

   // Track baselines
   //
   // Platforms move along tracks. The constants that follow define
   // the Y coordinate (from the top of the canvas) for each track.

   TRACK_1_BASELINE = 323,
   TRACK_2_BASELINE = 223,
   TRACK_3_BASELINE = 123,

   // Images

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

   // Platforms
   //
   // Each platform has its own fill style, but the stroke style is
   // the same for each platform.

platformData = [  // One screen for now
      // Screen 1.....................................................
      {
         left:      10,
         width:     230,
         height:    PLATFORM_HEIGHT,
         fillStyle: 'rgb(255,255,0)',
         opacity:   0.5,
         track:     1,
         pulsate:   false,
      },

      {  left:      250,
         width:     100,
         height:    PLATFORM_HEIGHT,
         fillStyle: 'rgb(150,190,255)',
         opacity:   1.0,
         track:     2,
         pulsate:   false,
      },

      {  left:      400,
         width:     125,
         height:    PLATFORM_HEIGHT,
```

```
         fillStyle: 'rgb(250,0,0)',
         opacity:   1.0,
         track:     3,
         pulsate:   false
      },

      {  left:      633,
         width:     100,
         height:    PLATFORM_HEIGHT,
         fillStyle: 'rgb(255,255,0)',
         opacity:   1.0,
         track:     1,
         pulsate:   false,
      },
   ];

// ------------------------ INITIALIZATION ----------------------------

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

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

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

function calculatePlatformTop(track) {
   var top;

   if      (track === 1) { top = TRACK_1_BASELINE; }
   else if (track === 2) { top = TRACK_2_BASELINE; }
   else if (track === 3) { top = TRACK_3_BASELINE; }

   return top;
}

function drawPlatforms() {
   var pd, top;

context.save(); // Save context attributes on a stack

   for (var i=0; i < platformData.length; ++i) {
      pd = platformData[i];
      top = calculatePlatformTop(pd.track);

context.lineWidth = PLATFORM_STROKE_WIDTH;
context.strokeStyle = PLATFORM_STROKE_STYLE;
context.fillStyle = pd.fillStyle;
context.globalAlpha = pd.opacity;

      // If you switch the order of the following two
      // calls, the stroke will appear thicker.

context.strokeRect(pd.left, top, pd.width, pd.height);
context.fillRect  (pd.left, top, pd.width, pd.height);
   }

context.restore(); // Restore context attributes
}

function drawRunner() {
context.drawImage(runnerImage,
```

```
      STARTING_RUNNER_LEFT,
      calculatePlatformTop(STARTING_RUNNER_TRACK) - runnerImage.height);
}

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

function startGame() {
draw();
}

// Launch game

initializeImages();
```

この JavaScript は `canvas` 要素にアクセスするのに続いて、キャンバスの 2D コンテキストへの参照を取得します。それから、そのコンテキストの `drawImage()` メソッドを使用して背景とランナーの画像を描画します。ここでは、`drawImage()` の 3 つの引数としての変数を使用してキャンバス上の特定の場所 `(x,y)` で画像を描画しています。

`drawPlatforms()` 関数は、コンテキストの線の幅と種類、塗りつぶしスタイル、グローバルなアルファ値などの属性を設定した後、長方形を描いて塗りつぶすことで、プラットフォームを描画します。`context.save()` と `context.restore()` を呼び出していることに注意してください。これらの呼び出しの間で設定している属性は一時的なものです。これらのメソッドについては、この連載の次回の記事で説明します。

ゲームは背景画像がロードされると開始されます。現状では、開始といっても単純に背景、スプライト、ランナーを描画するにすぎません。次の課題は、これらの静的な画像に生命を吹き込むことです。

次回は

この連載の次回の記事では、キャンバス・コンテキストの 2D API の概要を説明してから、アニメーションについて、さらには背景をスクロールすることで各要素の動きを設定する方法について説明します。また、パララックスを実装してプラットフォームが背景よりも近くに見えるようにする方法や、アニメーションのフレーム・レートに関わらずスプライトのアニメーションを必ず一定の速さにする方法についても説明します。ではまた次回お会いしましょう。

ダウンロード

内容	ファイル名	サイズ
Code for Snail Bait's background and runner	j-html5-game1.zip	718KB

著者について

David Geary

『Core HTML5 Canvas』の著者、David Geary は HTML5 Denver User's Group の共同設立者でもあり、Swing と JavaServer Faces に関するベストセラーの本を含め、Java に関する 8 冊の本の著者でもあります。また彼は、JavaOne、Devoxx、Strange Loop、NDC、OSCON などのカンファレンスで頻繁に講演を行っており、JavaOne Rock Star にも 3 度選ばれています。彼は連載記事、「JSF 2 の魅力」と「GWT の魅力」を developerWorks に寄稿しました。Twitter の @davidgeary で彼をフォローしてください。