HTML5 による 2D ゲームの開発: 舞台の設定

ゲーム・オブジェクトの実装、一時中断、フリーズ、フリーズ解除、キーボード入力

David Geary
Author and speaker
Clarity Training, Inc.

2012年 11月 29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。今回はゲームのコードをオブジェクトの中にカプセル化する方法、ゲームの一時中断と再開を実装する方法、ゲームを再開する際のカウントダウンの実装に CSS3 のトランジションを使用する方法について説明します。

このシリーズの他の記事を見る

ゲームの開発では、ゲームプレイとは直接関係のない多くの側面を扱います。説明の表示、ゲームの一時中断、異なるステージ間の遷移、ゲームのクレジットの表示などは、ゲーム開発者がゲームそのものの他に実装しなければならない機能の一部にすぎません。

あるゲームのインスピレーションが浮かんできたとき、普通、そこにはハイ・スコアの表示方法やステージ間の遷移方法の細部までは含まれていません。そのため、ゲームのインフラストラクチャーについて十分考えずにゲームプレイのメカニズムの実装を開始してしまいがちです。しかし大半のプロジェクトと同様、後からインフラストラクチャーの機能を追加しようとすると、最初からインフラストラクチャーを考慮した場合よりも多くの手間がかかるものです。

この連載の前回の記事では、Snail Bait のゲームプレイの基本要素であるグラフィックスとアニメーションについて説明しました。今回は一旦ゲームプレイに関する話題から離れて、ゲームのインフラストラクチャーの一部を実装します。まず始めに、Snail Bait のコードを `Game` オブジェクトの中にカプセル化する方法について説明します。私がこのゲームを最初に実装したときには、このカプセル化のステップから作業を始めたのですが、カプセル化によってコードをオブジェクト内に実装することでグラフィックスとアニメーションの説明がわかりにくくなるのは避けたかったため、この記事では `Game` オブジェクトについての説明を今まで先延ばしにしてきました。

またこの記事では、Snail Bait を一時中断してフリーズさせる方法と、その後でフリーズを解除してゲームを再開し、再開時にカウントダウンをアニメーションで表示する方法について説明します。この記事の最後ではゲームプレイのメカニズムに戻り、キーボード・イベントを処理してランナーの垂直方向の位置を制御する方法について説明します。

この記事では以下の内容を説明します。

- ゲームの機能をオブジェクトの中にカプセル化する方法
- ゲームを一時中断する方法と再開する方法
- ウィンドウがフォーカスを失った場合にゲームを自動的に一時中断する方法
- ウィンドウが再度フォーカスを得た場合にゲームを再開し、カウントダウンをアニメーションで表示する方法
- ユーザーに対して一時的にメッセージ (「トースト」と呼ばれます) を表示する方法
- キーボード入力を処理する方法

これらを説明する中で、JavaScript オブジェクトを定義する方法とインスタンス化する方法、CSS3 のトランジションを使用する方法、それらのトランジションと `setTimeout()` を組み合わせてステップ単位のアニメーションを実装する方法についても説明します。

Game オブジェクト

この連載のここまでの段階で、Snail Bait のすべての関数を実装し、またそれらの関数の変数のいくつかを「グローバル」変数として実装しました。もちろん、これではいけません。グローバル変数による悪影響をよく理解していない人は、この問題について Douglas Crockford 氏や Nicholas Zakas 氏など、JavaScript の権威が解説した記事を「参考文献」に挙げてありますので、それらの記事を参照してください。

グローバル変数を使用する代わりに、ここから先では Snail Bait の関数と変数のすべてをオブジェクトの中にカプセル化します。リスト 1 と リスト 2 を見るとわかるように、このオブジェクトは 2 つの部分で構成されます (この記事の完全なサンプル・コードは「ダウンロード」セクションから入手することができます)。

リスト 1 はこのゲームのコンストラクター関数です。この関数は Game オブジェクトの属性を定義します。

リスト 1. ゲームのコンストラクター (コードの一部を抜粋)

```
var SnailBait = function (canvasId) {
this.canvas  = document.getElementById(canvasId);
this.context = this.canvas.getContext('2d');

   // HTML elements

this.toast = document.getElementById('toast'),
this.fpsElement = document.getElementById('fps');

   // Constants

this.LEFT = 1;
this.RIGHT = 2;
   ...

   // Many more attributes are defined in the rest of this function
};
```

リスト 2 はこのゲームのプロトタイプです。このプロトタイプは Game オブジェクトのメソッドを定義します。

リスト 2. ゲームのプロトタイプ (コードの一部を抜粋)

```
SnailBait.prototype = {
   // The  and  methods were
   // discussed in the  in this series.


draw function (now) {
      this.setPlatformVelocity();
      this.setOffsets();

      this.drawBackground();

      this.drawRunner();
      this.drawPlatforms();
   },

drawRunner: function () {
      this.context.drawImage(this.runnerImage,
         this.STARTING_RUNNER_LEFT,
         this.calculatePlatformTop(this.runnerTrack) - this.RUNNER_HEIGHT);
   },
   ...

   // Many more methods are defined in the rest of this object
};
```

この連載で新しい機能を追加する際には、メソッドの追加や削除を行ったり、メソッドの実装の一部を変更したりします。この記事を最後まで説明した時点で存在する Snail Bait のメソッドを一覧にしたものが表 1 です。

表 1. 開発段階 (この記事を最後まで説明した時点) における Snail Bait のメソッド (呼び出される順に並んでいます)

メソッド	説明
`initializeImages()`	ゲームの画像を初期化します。背景画像に対する `onload` ハンドラーが `start()` を呼び出します。
`start()`	`requestAnimationFrame()` を呼び出すことによってゲームを開始します。`requestAnimationFrame()` は最初のアニメーション・フレームを描画する時になると `animate()` メソッドを呼び出します。
`splashToast() [1]`	一時的なメッセージをプレイヤーに対して表示します。
`animate() [2]`	ゲームが一時中断されていない場合、このメソッドは次のアニメーション・フレームを描画して `requestNextAnimationFrame()` を呼び出し、`animate()` に対する次の呼び出しをスケジューリングします。ゲームが一時中断されている場合には、`animate()` は 200ms 待った後に `requestNextAnimationFrame()` を呼び出します。
`calculateFps()`	最後のアニメーション・フレームからの経過時間に基づいてフレーム・レートを計算します。
`draw()`	次のアニメーション・フレームを描画します。
`setTranslationOffsets()`	背景およびプラットフォームを変換するためのオフセットを設定します。
`setBackgroundTranslationOffset()`.	背景を変換するためのオフセットを現在の時刻に応じて設定します。
`setPlatformTranslationOffset()`	プラットフォームを変換するためのオフセットを現在の時刻に応じて設定します。
`setPlatformVelocity()`	背景速度の倍数としてプラットフォームの速度を設定し、穏やかなパララックス効果を生み出します。

`drawBackground()`	Canvas の座標系を変換して背景を 2 度描画し、Canvas の座標系を最初の位置に逆変換します。
`drawRunner()` [3]	`drawImage()` を使用してランナーを描画します。
`drawPlatforms()` [3]	2D コンテキストの `strokeRect()` と `fillRect()` を使用して長方形のプラットフォームを描画します。
`calculatePlatformTop()`	トラックが指定されると、プラットフォーム最上部の Y 座標を計算します (プラットフォームは 3 つの水平トラックのいずれかの上を移動します)。
`turnLeft()`	背景とプラットフォームを右にスクロールします。
`turnRight()`	背景とプラットフォームを左にスクロールします。
`togglePaused()` [1]	ゲームの一時中断と再開の状態を切り換えます。

[1] この記事で導入したメソッド
[2] ブラウザーによって呼び出されるメソッド
[3] この連載の次回の記事で置き換えられるメソッド

関数とメソッド

オブジェクトのメンバーである JavaScript 関数は「メソッド」と呼ばれ、独立した関数は単純に「関数」と呼ばれます。

表 1 に記載したメソッドの大部分は、この連載の過去 2 回の記事で導入したメソッドです (これらのメソッドは過去 2 回の記事では単なる関数にすぎませんでした)。この記事では、`togglePaused()` と `splashToast()` という 2 つの新しいメソッドについて説明し、また他のメソッド (`animate()` など) に対する変更についても説明します。

リスト 1 とリスト 2 の JavaScript は関数とプロトタイプを定義していますが、`SnailBait` オブジェクトのインスタンス化はしていません。それを次のセクションで行います。

ゲームを開始する

SnailBait のグローバル・オブジェクト

リスト 1 とリスト 3 を見るとわかるように、Snail Bait のグローバル・オブジェクトは `SnailBait` 関数と `SnailBait` オブジェクトの 2 つしかありません。

リスト 3 はゲームを開始する JavaScript です。このリストの最初の部分は `SnailBait` の 3 つのメソッド (`animate()`、`start()`、`initializeImages()`) の実装を示しています。

リスト 3. ゲームを開始する

```
SnailBait.prototype = {
   ...

   // The 'this' variable in the animate() method is
   // the window object, so the method uses snailBait instead

animate: function (now) {
snailBait.fps = snailBait.calculateFps(now);
snailBait.draw(now);
```

```
requestNextAnimationFrame(snailBait.animate);
   },

start: function () {
      this.turnRight();                    // Sets everything in motion
      this.splashToast('Good Luck!', 2000); // "Good Luck" is displayed for 2 seconds

requestNextAnimationFrame(this.animate);
   },

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

      this.background.onload = function (e) {

         // ...the 'this' variable is the window object,
         // so this function uses snailBait instead.

snailBait.start();
      };
   },
}; // End of SnailBait.prototype


// Launch game

var snailBait = new SnailBait(); // Note: By convention, the object
                                 // reference starts with lowercase, but
                                 // the function name starts with uppercase

snailBait.initializeImages();
```

JavaScript の **this** オブジェクトは要注意です

Java などの昔ながらのオブジェクト指向言語を使用した経験のある人であれば、オブジェクトの this 変数は必ずそのメソッドに関連付けられたオブジェクトを指すと想定します。

JavaScript で最も危険な点は、this 変数が変化する可能性があることです。リスト 2 で、animate() メソッドの this 変数と背景画像の onload イベント・ハンドラーは、SnailBait オブジェクトではなく window オブジェクトを参照します。そのため、これらのメソッドは SnailBait オブジェクトに直接アクセスします。

リスト 3 の JavaScript は `SnailBait` オブジェクトをインスタンス化し、このオブジェクトの `initializeImages()` メソッドを呼び出します。それによって背景画像の `onload` イベント・ハンドラーが設定されます。背景画像がロードされると、`onload` イベント・ハンドラーが `start()` メソッドを呼び出します。

`start()` メソッドは `turnRight()` を呼び出し、それによって背景とプラットフォームが動き出します。また `start()` メソッドは `splashToast()` も呼び出し、この `splashToast()` が「Good Luck!」というメッセージを 2 秒間表示します。最後に、`start()` は `requestNextAnimationFrame()` のポリフィルを呼び出します。ポリフィルについては、この連載の第 2 回の記事で説明しました (第 2 回の記事の「`requestAnimationFrame()` のポリフィル」セクションを参照) が、このポリフィルによって最終的にゲームの `animate()` メソッドが呼び出されます。

`animate()` メソッドは現在のフレームを描画してから `requestNextAnimationFrame()` を呼び出して自分自身をコールバック関数として指定し、アニメーションを継続させます。

このようにしてゲームが開始されます。次に、ゲームの開始後にゲームを一時中断する方法について説明します。

ゲームを一時中断する

HTML5 のゲーム (特にテレビ・ゲーム) は、一時中断ができなければなりません。リスト 4 では Snail Bait のゲーム・ループを変更し、ゲームを一時中断したり再開したりできるようにしています。

リスト 4. 一時中断と再開

```
var SnailBait = function (canvasId) {
   ...
   this.paused = false,
   this.PAUSED_CHECK_INTERVAL = 200; // milliseconds
   ...
};

SnailBait.prototype = {
animate: function (now) {
      if (snailBait.paused) {

         // Check again in snailBait.PAUSED_CHECK_INTERVAL milliseconds

setTimeout( function () {

            requestNextAnimationFrame(snailBait.animate);

         }, snailBait.PAUSED_CHECK_INTERVAL);
      }
      else {

         // The game loop from

         snailBait.fps = snailBait.calculateFps(now);
         snailBait.draw(now);
         requestNextAnimationFrame(snailBait.animate);
      }
   },

togglePaused: function () {
      this.paused = !this.paused;
   },
};
```

`togglePaused()` メソッドは単純にゲームの `paused` 変数を切り換えます。paused 変数が `true` の場合 (つまりゲームが一時中断されている場合)、`animate()` メソッドはゲーム・ループを実行しません。

一時中断されたゲームを再開すべきかどうかを (フレーム・レートが 60fps として) 毎秒 60 回確認する必要はありません。そうした頻繁な確認は非効率です。そこでリスト 4 の `animate()` メソッドは 200ms 待った後に `requestNextAnimationFrame()` のポリフィルを呼び出し、このポリフィルは次のアニメーション・フレームを描画する時になると `animate()` に対する次回の呼び出しをスケジューリングします。

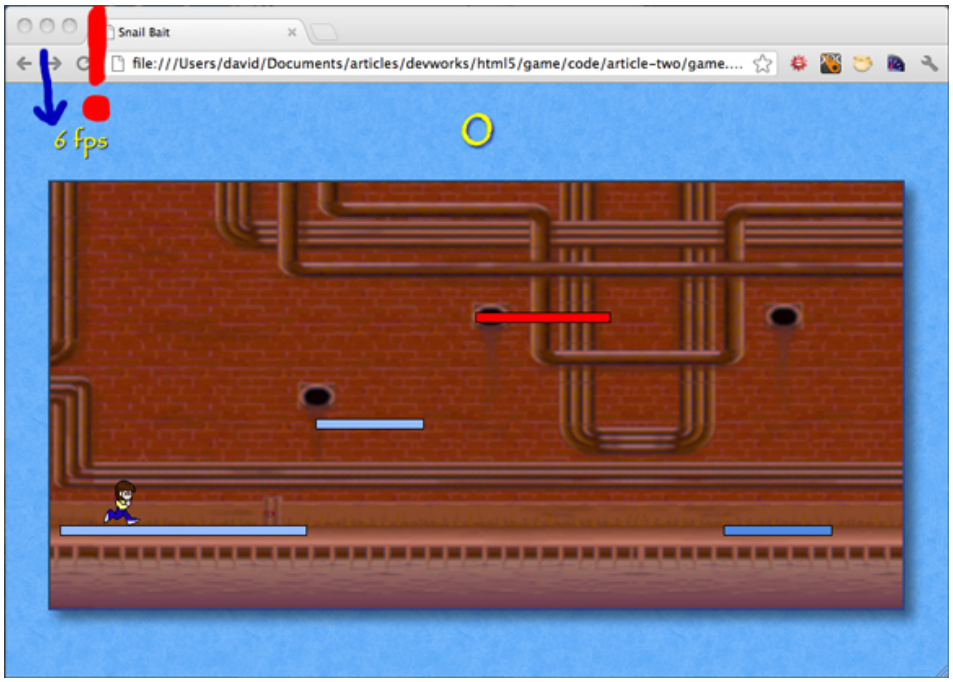
ウィンドウがフォーカスを失うと自動的に一時中断する

W3C の「Timing control for script-based animations (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)」仕様には、`requestAnimationFrame()` によって実装されるアニメーションに関して次のように記述されています。

> ページが現在非表示の場合、そのページのアニメーションの動作を大幅に制限することにより、更新の頻度を抑え、CPU パワーをほとんど使用しないようにすることができます。

「大幅に制限する」という表現の意味は、ブラウザーが図 1 に示すような非常に低いフレーム・レート (通常は 1 fps から 10 fps の間) でアニメーションのコールバックを呼び出す、ということです (図 1)。図 1 はウィンドウが再度フォーカスを得た直後に 6 fps というフレーム・レートが表示される状態を示しています。

図 1. Snail Bait がフォーカスを失った後に再度フォーカスを得た時の画面



フレーム・レートが大幅に制限されると、通常はフレーム・レートに基づいて衝突の発生 (または発生の可能性) を判断している衝突検出アルゴリズムが大混乱に陥ります。フレーム・レートの大幅な低下による衝突検出の混乱を防ぐためには、ゲームの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し、再度フォーカスを得た場合にゲームを再開するようにします。その方法を示したものがリスト 5 です。

リスト 5. 自動的な一時中断

```
window.onblur = function () { // window looses focus
   if (!snailBait.paused) {
      snailBait.togglePaused();
   }
};

window.onfocus = function () { // window regains focus
   if (snailBait.paused) {
      snailBait.togglePaused();
   }
};
```

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必要がありますが、それだけではなく、ゲームを一時中断している間はゲームを「フリーズ」させる必要もあります。

ゲームをフリーズさせる

ゲームを一時中断するためには単にアニメーションを停止するだけでは不十分です。「ゲームは中断したそのままの状態から再開できなければなりません。」リスト 4 はその要件を満たすように見えます。つまり、ゲームが一時中断している間は何も起こらないので、ゲームは一時中断する前とまったく同じ状態で再開するはずであるかのように思えます。しかしそうではありません。なぜなら、Snail Bait を含め、すべてのアニメーションにとっての基本要素は時刻だからです。

第 2 回の記事で説明したように (第 2 回の記事の `requestAnimationFrame()` のセクションを参照)、`requestAnimationFrame()` は指定されたコールバック関数に時刻を渡します。Snail Bait の場合、そのコールバックは `animate()` メソッドであり、その `animate()` メソッドがその時刻を `draw()` メソッドに渡します。

ゲームが一時中断されると、アニメーションは実行されなくなりますが、それと無関係に時刻は進み続けます。Snail Bait の `draw()` メソッドは `animate()` から受け取る時刻に基づいて次のアニメーション・フレームを描画するため、リスト 4 に示した `togglePaused()` の実装では、一時中断されたゲームが再開されると、ゲームにとっては時刻が進んでしまうことになります。

リスト 6 に、一時中断されたゲームが再開された時に Snail Bait の時刻が突然シフトしてしまうのを防ぐ方法を示します。

リスト 6. ゲームをフリーズする

```
var SnailBait = function (canvasId) {
   ...
   this.paused = false,
   this.pauseStartTime = 0,
   this.totalTimePaused = 0,
   this.lastAnimationFrameTime = 0,
   ...
};

SnailBait.prototype = {
   ...
   calculateFps: function (now) {
      var fps = 1000 / (now - this.lastAnimationFrameTime);
      this.lastAnimationFrameTime = now;

      if (now - this.lastFpsUpdateTime > 1000) {
```

```
            this.lastFpsUpdateTime = now;
            this.fpsElement.innerHTML = fps.toFixed(0) + 'fps';

         }

         return fps;
      },

      togglePaused: function () {
         var now = +new Date();

         this.paused = !this.paused;

         if (this.paused) {
            this.pauseStartTime = now;
         }
         else {
            this.lastAnimationFrameTime += (now - this.pauseStartTime);
         }
      },
};
```

リスト 6 は、ゲームの一時中断があった場合には中断時間を考慮するように Snail Bait の `togglePaused()` メソッドと `calculateFps()` メソッドを変更しています。

ここでは、前のアニメーション・フレームのフレーム・レートを計算するために、最後のフレームを描画した時刻を現在の時刻から引き、その値を 1,000 で割っています。こうすることで、フレーム・レートはミリ秒単位ではなく秒単位で得られます (フレーム・レートの計算については、第 2 回の記事の「アニメーションのフレーム・レート (フレーム/秒: fps) を計算する」セクションを参照)。

ゲームが再開されると、ゲームの中断時間を最後のアニメーション・フレームの時刻に加算します。この加算によって実質的に一時中断の影響が解消され、ゲームは中断した時点のそのままの状態から再開されます。

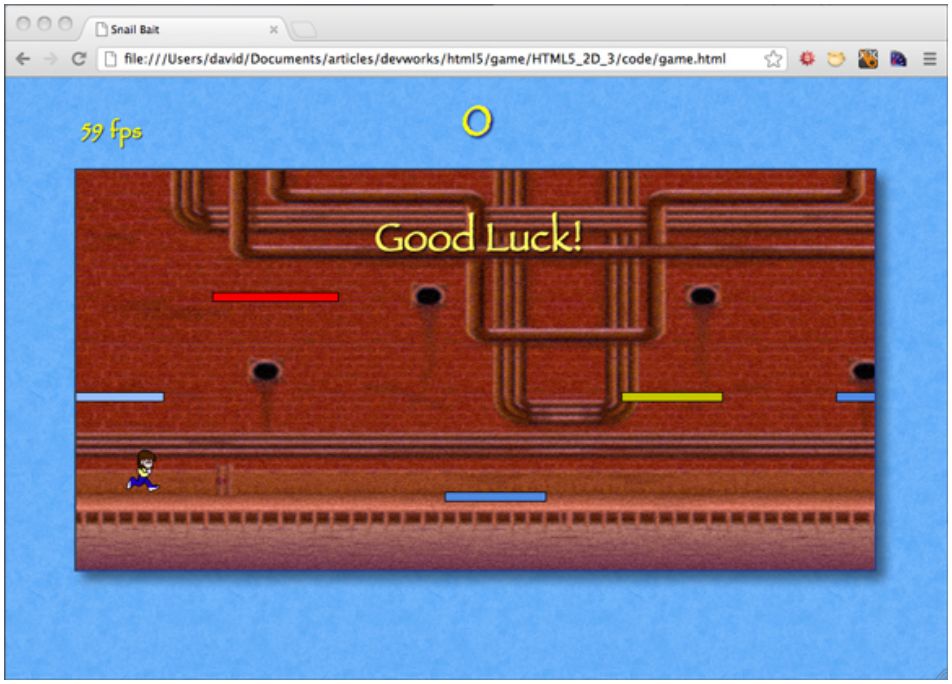
ウィンドウがフォーカスを得たときにゲームのフリーズを解除する

ゲームが再開される際に、プレイヤーは操作を再び開始するための準備の時間をもらえると、スムーズにゲームの再開へと遷移できてありがたいものです。そしてその間、ゲーム再開までの残り時間を画面に表示するのが賢明です。Snail Bait では、ゲーム再開までの残り時間が、トーストに示されるカウントダウンとして、画面に表示されるように実装しています。そこで、この点についてトーストの概要を含めて説明しましょう。

トースト

トーストはプレイヤーのために一時的にゲームに表示されるものです。例えば図 2 では「Good Luck!」というトーストを表示しています。

図 2. トースト



Snail Bait そのものと同様、Snail Bait のトーストは HTML、CSS、JavaScript の組み合わせによって実装されます。それを以下の 3 つのリストに示します。

リスト 7 に示すのは、トーストの HTML です。

リスト 7. トーストの HTML

```
<!DOCTYPE html>
<html>
   <head>
      ...
   </head>

   <body>
      <div id='wrapper'>
         <!-- Toast.....................................................-->

<div id='toast'></div>
         ...

      </div>
      ...
  </body>
</html>
```

Snail Bait の「Good Luck!」トーストを実装するための CSS がリスト 8 です。

リスト 8. トーストの CSS

```
#toast {
   position: absolute;
   ...

   -webkit-transition: opacity 0.5s;
   -moz-transition: opacity 0.5s;
   -o-transition: opacity 0.5s;
   transition: opacity 0.5s;

   opacity: 0;
z-index: 1;
display: none;
}
```

リスト 9 に示すのは、「Good Luck!」トーストの JavaScript です。

リスト 9. トーストの JavaScript

```
var SnailBait =  function () {
   ...
   this.toast = document.getElementById('toast'),
   this.DEFAULT_TOAST_TIME = 3000, // 3 seconds
   ...
};

SnailBait.prototype = {
   ...
start: function () {
      ...
      snailBait.splashToast('Good Luck!');
   },

splashToast: function (text, howLong) {
      howLong = howLong || this.DEFAULT_TOAST_TIME;

toast.style.display = 'block';
      toast.innerHTML = text;

setTimeout( function (e) {
         toast.style.opacity = 1.0; // After toast is displayed
      }, 50);

      setTimeout( function (e) {
         toast.style.opacity = 0; // Starts CSS3 transition

         setTimeout( function (e) {
            toast.style.display = 'none'; // Just before CSS3 animation concludes
         }, 480);
      }, howLong);
   },
   ...
}
```

上記 3 つのリストの実装、そしてリスト 7 でわかるように、トーストは単なる `DIV` にすぎません。`DIV` の CSS コードを記載するリスト 8 では、さらに興味深いことがわかります。`DIV` の位置は `absolute` です。つまりトーストは他の `DIV` の前や後ろではなく、他の `DIV` の上または下に表示されます。また `toastDIV` の `z-index` は 1、つまりトーストは必ずゲームのキャンバス (デフォルトで `z-index` は 0) の上に表示されます。最後に、`toast` 要素の CSS は 0.5 秒の遷移を定義し、その遷移を opacity プロパティーに関連付けています。`opacity` プロパティーが変更されると、CSS は

トースト DIV の透明度を 0.5 秒間のスムーズなアニメーションで変更前の透明度から新しい値の透明度へと変化させます。

指定した時間だけトーストを表示するリスト 9 の splashToast() メソッドは、それよりもさらに興味深いものです。Snail Bait がデフォルトの表示時間 (3 秒) で splashToast() を呼び出すと、トーストは 0.5 秒でフェードインされ、2.5 秒間という短い時間表示され、0.5 秒でフェードアウトします。その仕組みは以下のとおりです。

まず splashToast() メソッドは toastDIV の display 属性を block に設定します。通常はこれにより、DIV が見えるようになりますが、opacity 属性が最初は 0 であるため、toastDIV は見えない状態のままです。次に splashToast() は引数として渡されたテキストを toastDIV の innerHTML に設定します。しかし透明度の設定はそのままなので、テキストを設定してもやはり toastDIV は見えるようになりません。

toast DIV を見えるようにするには、透明度を 1.0 に設定します。この設定により、リスト 8 で指定された遷移に従って CSS3 のアニメーションがトリガーされますが、「実際にアニメーションがトリガーされるのは、この設定を囲む奇妙な setTimeout() の結果として、後で (この場合は 50ms 後に) 透明度が設定された場合のみです」。その理由は以下のとおりです。

CSS3 の遷移は、中間的な状態を持つ要素のプロパティーに対してのみ指定することができます。例えば (2 つのランダムな数字を選ぶとして) 0.2 から 0.3 へと透明度を変更する場合、0.21、0.22 等々という中間的な透明度があります。

遷移に中間的な状態が必要なことは理にかなっています。中間的な状態がないと、遷移のアニメーションの詳細を明確に指定することができません。そのため、例えば中間的な状態がない display プロパティーに対して遷移を指定することはできません。それだけではなく、display プロパティーを変更すると、「CSS3 はそれまで他のプロパティーに指定されていたすべての遷移も無視するようになります」。これも理にかなっています。なぜなら、相反する 2 つのことをするように CSS3 に指示しているからです。つまり display プロパティーを変更して要素を即座に表示させる一方、例えば opacity プロパティーによる遷移で、その要素をゆっくりとフェードインさせて表示することになるからです。同時に両方を行うことはできないため、CSS3 は display プロパティーを変更することを選択します。

半透明の DIV とイベント

先ほどの splashToast() に関する説明の後、そもそもなぜこのメソッドが DIV の display プロパティーを操作するのか、と思った人がいるかもしれません。なぜ単純に DIV の透明度を操作して、DIV を見えるようにしたり、見えなくしたりしないのでしょう。その答えは、意図的な場合を除き、見えない DIV が存在することそれ自体が賢明なことではないためです。なぜなら、見えない DIV の存在が予想外の形で (例えばインターセプトするイベントによって) 知られてしまう可能性があるからです。

splashToast() が DIV の display プロパティーと opacity プロパティーを同時に設定すると CSS3 は透明度の遷移を無視するため、splashToast() メソッドは display プロパティーを設定した「後で」(もっと具体的に言えば、約 50ms 後に)、透明度を 1.0 に設定します。

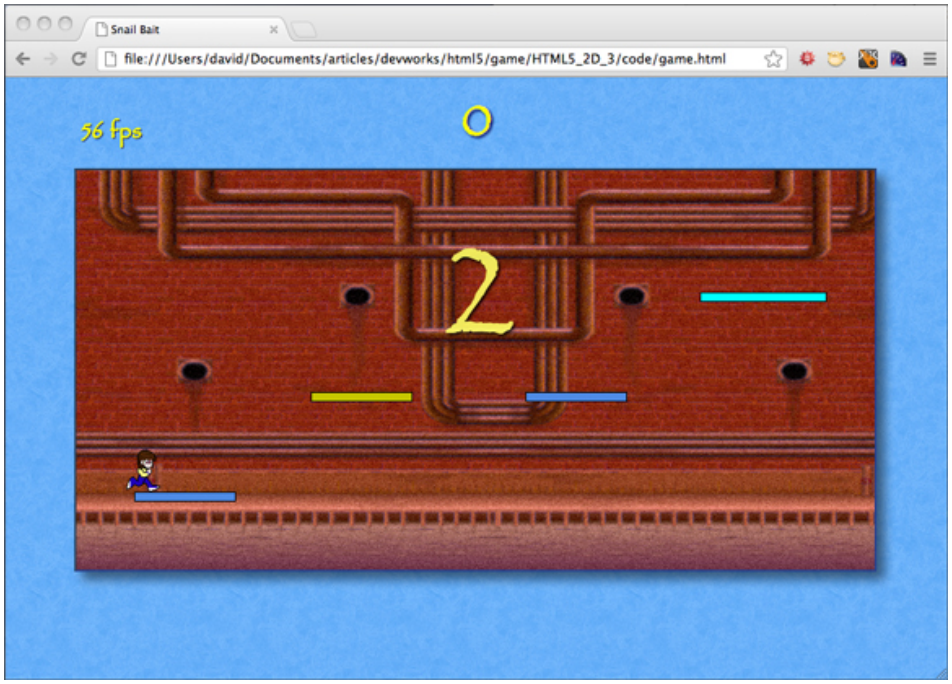
最後に、必要な表示時間が経過すると、splashToast() は DIV の opacity プロパティーを 0 にリセットし、それによって再度 CSS3 のアニメーションがトリガーされて 0.5 秒間継続します。CSS3

のアニメーションの起動後 2 秒経過すると、`splashToast()` メソッドは `display` プロパティーを `0` にリセットします。

Snail Bait のフリーズを解除する

Snail Bait は一時中断後に再開されると、3 秒間のカウントダウンを表示して、プレイヤーに準備する時間を与えます (図 3 を参照)。

図 3. フリーズを解除する間のカウントダウン



リスト 10 に、カウントダウンの JavaScript を示します。

リスト 10. カウントダウンの JavaScript

```
var SnailBait = function (canvasId) {
   ...
   this.toast = document.getElementById('toast'),
};


window.onblur = function (e) {  // Pause if unpaused
   if (!snailBait.paused) {
      snailBait.togglePaused();
   }
};

window.onfocus = function (e) {  // unpause if paused
   var originalFont = snailBait.toast.style.fontSize;

   if (snailBait.paused) {
      snailBait.toast.style.font = '128px fantasy';

      snailBait.splashToast('3', 500); // Display 3 for one half second

      setTimeout(function (e) {
         snailBait.splashToast('2', 500); // Display 2 for one half second

         setTimeout(function (e) {
```

```
            snailBait.splashToast('1', 500); // Display 1 for one half second

            setTimeout(function (e) {
               snailBait.togglePaused();

               setTimeout(function (e) { // Wait for '1' to disappear
                  snailBait.toast.style.fontSize = originalFont;
               }, 2000);
            }, 1000);
         }, 1000);
      }, 1000);
   }
};
```

Snail Bait のウィンドウが再度フォーカスを得ると、`splashToast()` メソッドを使用してカウントダウンが開始されます。カウントダウンの各数値は 0.5 秒間でフェードインされ、0.5 秒間でフェードアウトされます。カウントが 0 になると `onfocus` ハンドラーによってゲームが再開されます。

ただし、カウントダウン中にプレイヤーが別のウィンドウや別のタブをアクティブにすると、リスト 10 のコードは適切に動作しません。なぜなら、ウィンドウがフォーカスを得ているかどうかにかかわらず、カウントダウンが終了するとゲームが再開されるからです。この問題は `windowHasFocus` フラグを使用することで容易に修正することができます。それをリスト 11 に示します。

リスト 11. カウントダウンの間にフォーカスを失う場合を考慮する

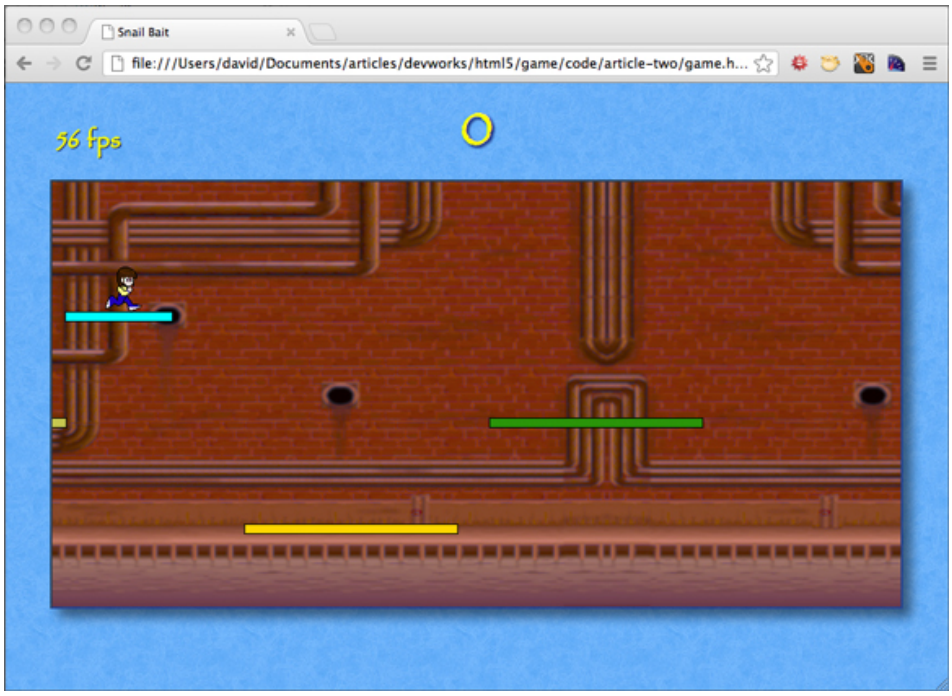
```
var SnailBait = function (canvasId) {
   ...
   this.windowHasFocus = true,
   ...
};
...

SnailBait.prototype = {
   ...

   splashToast: function (text, howLong) {
      howLong = howLong || this.DEFAULT_TOAST_TIME;

      toast.style.display = 'block';
      toast.innerHTML = text;

      setTimeout( function (e) {
         if (snailBait.windowHasFocus) {
            toast.style.opacity = 1.0; // After toast is displayed
         }
      }, 50);

      setTimeout( function (e) {
         if (snailBait.windowHasFocus) {
            toast.style.opacity = 0; // Starts CSS3 transition
         }

         setTimeout( function (e) {
            if (snailBait.windowHasFocus) {
               toast.style.display = 'none';
            }
         }, 480);
      }, howLong);
   },
   ...
```

```
};
...

window.onblur = function (e) {  // pause if unpaused
   snailBait.windowHasFocus = false;

   if (!snailBait.paused) {
      snailBait.togglePaused();
   }
};

window.onfocus = function (e) {  // unpause if paused
   var originalFont = snailBait.toast.style.fontSize;

   snailBait.windowHasFocus = true;

   if (snailBait.paused) {
      snailBait.toast.style.font = '128px fantasy';

      snailBait.splashToast('3', 500); // Display 3 for one half second

      setTimeout(function (e) {
         snailBait.splashToast('2', 500); // Display 2 for one half second

         setTimeout(function (e) {
            snailBait.splashToast('1', 500); // Display 1 for one half second

            setTimeout(function (e) {
               if ( snailBait.windowHasFocus) {
                  snailBait.togglePaused();
               }

               setTimeout(function (e) { // Wait for '1' to disappear
                  snailBait.toast.style.fontSize = originalFont;
               }, 2000);
            }, 1000);
         }, 1000);
      }, 1000);
   }
};
```

キーボード入力

Snail Bait では、プレイヤーはキーボードを使用してランナーを制御します。そこでこの記事の締めくくりとして、このゲームがキーボード入力をどのように処理するかを簡単に説明します。「d」キーと「k」キーでランナーはそれぞれ左と右に移動し、「j」キーと「f」キーでランナーはそれぞれジャンプまたは落下します。図 4 はランナーが 3 番目のプラットフォーム・トラックに飛び移った後の状態を示しています。

図 4. ランナーがトラック間を飛び移った後の状態



「フォーカス可能な」HTML 要素に対してのみ、キーボード・イベント・リスナーを追加することができます。`canvas` 要素はフォーカス可能ではないため、Snail Bait は `onkeydown` イベント・ハンドラーを `window` オブジェクトに追加します (リスト 12)。

リスト 12. キーボード入力に応答する

```
var runnerTrack = 1,
    BACKGROUND_VELOCITY = 42;

function turnLeft() {
   bgVelocity = -BACKGROUND_VELOCITY;
}

function turnRight() {
   bgVelocity = BACKGROUND_VELOCITY;
}

window.onkeydown = function (e) {
   var key = e.keyCode;

   if (key === 80 || (paused && key !== 80)) {  // p
      togglePaused();
   }

   if (key === 68 || key === 37) { // d or left arrow
      turnLeft();
   }
   else if (key === 75 || key === 39) { // k or right arrow
      turnRight();
   }
   else if (key === 74) { // j
      if (runnerTrack === 3) {
         return;
      }
      runnerTrack++;
```

```
   }
   else if (key === 70) { // f
      if (runnerTrack === 1) {
         return;
      }
      runnerTrack--;
   }
};
```

Snail Bait のゲーム・ループが常に実行されていることを理解することが重要です。`animate()` 関数はブラウザーが次のアニメーション・フレームを描画する準備が整うと、その都度ブラウザーによって呼び出され、そのたびに `animate()` は (リスト 2 の) `draw()` を呼び出します。

ゲーム・ループは常に実行されているため、キーボード・イベント・ハンドラーは単純にゲームの変数を設定することしかしません。例えば「k」を押してランナーを右に移動させると、イベント・ハンドラーは `bgVelocity` を `BACKGROUND_VELOCITY = 42` (ピクセル/秒) に設定し、「d」を押してランナーを左に移動させると、イベント・ハンドラーは `bgVelocity` を -42 ピクセル/秒に設定します。後でゲームが次のアニメーション・フレームを描画するときまで、これらの設定は有効になりません。

次回は

この連載の次回の記事では、動きの少ない Snail Bait のグラフィックスを「スプライト」として知られるアニメーション化されたオブジェクトに変える方法について説明します。いくつかの異なる方法でスプライトを描画する方法について、スプライトシートから描画する方法を含めて説明し、さらにそれらのスプライトを Snail Bait の既存のコードに組み込む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
Sample code	j-html5-game3.zip	3.9MB

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。