HTML5 による 2D ゲームの開発: スプライトのビヘイビアを実装する

Snail Bait のキャラクターにビヘイビアを追加する

David Geary

2013年 6月 13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。今回の記事では、あらゆるテレビ・ゲームに不可欠な要素である、スプライトのビヘイビアを実装する方法について説明します。

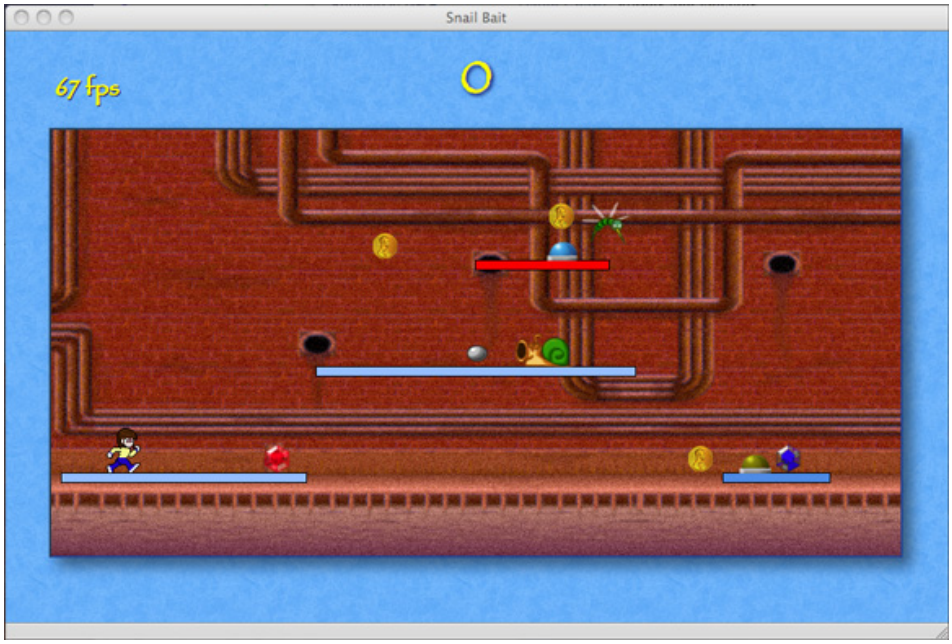
このシリーズの他の記事を見る

素晴らしいストーリーには素晴らしいキャラクターが登場します。また本や映画と同じように、テレビ・ゲームにも興味深い振る舞い (ビヘイビア) をするキャラクターが必要です。例えば、米国でベストセラーとなったテレビ・ゲーム「Braid (ブレイド)」の主人公は、時間を「操る」ことができます。この独創的なビヘイビアによって、Braid (ブレイド) は同時期に作られたゲームとは一線を画しています。

ビヘイビアは、あらゆるテレビ・ゲームの核心部分です。前回の記事で実装した、動きのない Snail Bait のスプライトにビヘイビアを追加すると、途端に Snail Bait がこれまでよりも面白いものになります (図 1)。

© Copyright IBM Corporation 2013

図 1. 今回の記事の終わりの時点での Snail Bait の状態



前回の記事の「スプライト・オブジェクト」セクションで説明したように、Snail Bait ではスプライトのアクション (例えば、走る、ジャンプする、爆発するなど) をスプライト・オブジェクトに直接実装するのではなく、ビヘイビアという別のオブジェクトに実装を委ねることで、アクションを制御します。

図 1 は、カタツムリがカタツムリ爆弾を発射しているところを示しており、他にも (静止画としての図 1 を見ただけではわかりませんが) 以下のようなビヘイビアがあります。

- ランナーが走る
- ボタンがそれぞれのプラットフォーム上をゆっくりと行ったり来たりする
- ルビーとサファイアがきらきら輝く

これらのビヘイビアをまとめたものが表 1 です。

表 1. この記事で説明するビヘイビア

スプライト	ビヘイビア	説明
ボタン、カタツムリ	`paceBehavior`	プラットフォームに沿ってゆっくりと行ったり来たりします。
ランナー	`runBehavior`	ランナーが走っているときの一連の画像を順番に、繰り返し描画することで、ランナーが走っているように見せます。
カタツムリ	`snailShootBehavior`	カタツムリの口からカタツムリ爆弾を発射します。
カタツムリ	`cycleBehavior`	スプライトのさまざまな画像を繰り返し描画します。
カタツムリ爆弾	`snailBombMoveBehavior`	カタツムリ爆弾がキャンバス上に表示されている間、カタツムリ爆弾を左方向へ水平に移動します。

時間を操作する

Braid (ブレイド) ではゲームの主人公である Tim (ティム) が時間を「操り」ますが、テレビ・ゲームではどれもが見事な方法で時間を操作しています。この記事を読むと、ビヘイビアはその根底で常に時間の流れと関係していることがわかるはずです。この連載の次回、そ

してさらにその次の記事では、時間を操作することによってリニアでない動きを実装する方法について説明します。リニアでない動きは、走ったり、ジャンプしたりといった、リアルな動きをする上での基本となります。

この表 1 に記載したビヘイビアは、連載第 1 回の記事の表「Snail Bait のスプライトとそのビヘイビア」を見るとわかるように、ゲームで使用されるビヘイビア全体の半分もありません。また、これらのビヘイビアはスプライトのビヘイビアのうち最も基本的なものでもあります。例えば、今後の記事を読むとわかりますが、ジャンプは他のビヘイビアよりもかなり複雑です。それでもなお、この記事で説明する単純なビヘイビアの実装からは、以下の内容をはじめとする数多くのことを学ぶことができます。

- ビヘイビアの実装方法と、スプライトにビヘイビアを割り当てる方法
- スプライトの一連の画像を順番に、繰り返し描画する方法
- メモリー使用量を減らすために Flyweight ビヘイビアを作成する方法
- ビヘイビアを組み合わせる方法
- ビヘイビアを使用して爆弾を発射する方法

ビヘイビアの基本

レプリカアイランドのビヘイビア

ビヘイビアという概念は、オープンソースの Android 用ゲームとして人気の高い、レプリカアイランドに発想を得ています。また、Snail Bait のグラフィックスの多くは、レプリカアイランドのものを使用しています。「参考文献」には、レプリカアイランドへのリンクと、レプリカアイランドの作成者がビヘイビアについて説明しているブログ記事へのリンクを挙げてありますので、参照してください。

どのようなオブジェクトでも、`execute()` メソッドを持たせることによって、ビヘイビア・オブジェクトにすることができます。ビヘイビア・オブジェクトの `execute()` メソッドは、スプライト、時刻、ゲームのアニメーションのフレーム・レートという 3 つの引数を取ります。そして、時刻とアニメーションのフレーム・レートに応じてスプライトの状態を変更します。

ビヘイビアは強力であるため、以下のことが可能です。

- ビヘイビアによって、スプライトとスプライトの動作を切り離しておくことができます。
- 実行時にスプライトのビヘイビアを変更することができます。
- どのスプライトでも機能するビヘイビアを実装することができます。
- ステートレスなビヘイビアを Flyweight として使用することができます。

表 1 に挙げたビヘイビアの実装の詳細を説明する前に、ランナーのビヘイビア全体について見ていくことで、ビヘイビアの大まかな概要 — 具体的には、ビヘイビアの実装方法と、ビヘイビアをスプライトと関連付ける方法 — を説明します。

ランナーのビヘイビア

Snail Bait のランナーには表 2 に挙げた 4 つのビヘイビアがあります。

表 2. ランナーのビヘイビア

ビヘイビア	説明

`runBehavior`	スプライト・シートのランナーのセルを順番に、繰り返し描画することで、ランナーが走っているように見せます。
`jumpBehavior`	ジャンプのすべての側面 (上昇、下降、着地) を制御します。
`fallBehavior`	ランナーが落下するときの垂直の動きを制御します。
`runnerCollideBehavior`	ランナーと他のスプライトとの衝突を検出し、それに対応した動作をします。

ここではランナーのビヘイビアをオブジェクトの配列によって定め、この配列を `Sprite` コンストラクターに渡します (リスト 1)。

リスト 1. SnailBait のランナーを作成する

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

   this.runner = new Sprite('runner',              // Type
                            this.runnerArtist,    //  Artist
                            [this.runBehavior,  //  Behaviors
                           this.jumpBehavior,
                           this.fallBehavior,
                           this.runnerCollideBehavior

                            ]);
};
```

ランナーのビヘイビアをリスト 2 に示します。実装の詳細は省略してあります。

リスト 2. ランナーのビヘイビア・オブジェクト

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

   this.runBehavior = {
      execute: function(sprite, time, fps) { // sprite is the runner
         ...
      }
   };
   this.jumpBehavior = {
      execute: function(sprite, time, fps) { // sprite is the runner
         ...
      }
   };
   this.fallBehavior = {
      execute: function(sprite, time, fps) { // sprite is the runner
         ...
      }
   };
   this.runnerCollideBehavior = {
      execute: function(sprite, time, fps) { // sprite is the runner
         ...
      }
   };
};
```

アニメーションの各フレームで、Snail Bait は一連のスプライトに対して繰り返し処理を行い、各スプライトの `update()` メソッドを呼び出します (リスト 3)。

リスト 3. ビヘイビアを実行する

```
Sprite.prototype = {
   update: function (time, fps) {
      for (var i=0; i < this.behaviors.length; ++i) {
         if (this.behaviors[i] === undefined) { // You never know
            return;
         }

         this.behaviors[i].execute(this, time, fps);
      }
   }
};
```

`Sprite.update()` メソッドはスプライトのビヘイビアに対して繰り返し処理を行い、各ビヘイビアの `execute()` メソッドを呼び出します。Snail Bait は、表示されているすべてのスプライトに関連したすべてのビヘイビアを継続的に (アニメーションのフレームごとに 1 度) 呼び出します。つまりビヘイビアの `execute()` メソッドは、あまり頻繁には呼び出されない他の大部分のメソッドとは異なり、常に回転している小さなモーターのようなものです。

これで、スプライトとビヘイビアを関連付ける方法を理解できたので、ここからはビヘイビアを実装する作業に専念したいと思います。

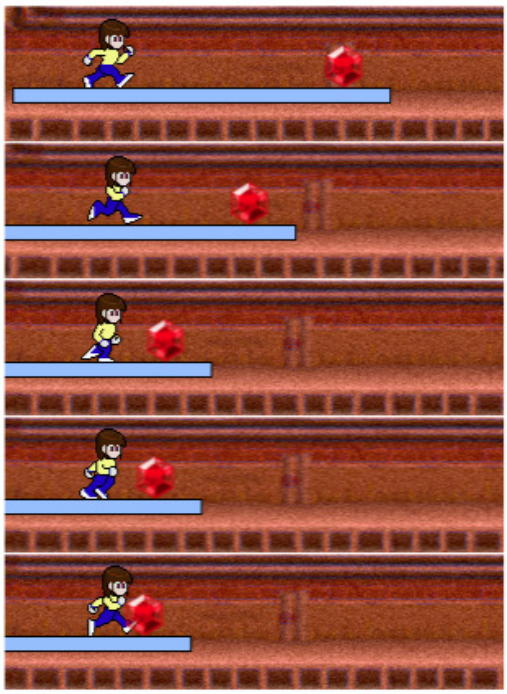
Strategy デザイン・パターン

ビヘイビアは、アルゴリズムをオブジェクトにカプセル化する Strategy デザイン・パターン (「参考文献」を参照) を実装したものです。実行時に、これらのアルゴリズムをさまざまに組み合わせ、一連のビヘイビアを 1 つのスプライトに割り当てることができます。ビヘイビアを使用すると、個々のスプライトの中にアルゴリズムを直接ハードコーディングするよりも柔軟性を高めることができます。

走る

Snail Bait では、ランナーが走っているように見せるために、次の 2 つのことを行います。第 1 に、連載第 2 回の「背景をスクロールさせる」セクションで説明したように、このゲームは常に背景をスクロールさせ、ランナーが水平方向に移動しているように見せます。第 2 に、ランナーの「走る」ビヘイビアは、このゲームのスプライト・シートに含まれる一連のランナーの画像を順番に、繰り返し描画します (図 2)。

図 2. ランナーが走っているときのシーケンス



リスト 4 のコードは、ランナーが走っているビヘイビアを実装したものです。

リスト 4. ランナーの `runBehavior`

```
var SnailBait =  function () {
   ...
   this.BACKGROUND_VELOCITY = 32, // pixels/second
   this.RUN_ANIMATION_RATE = 17, // frames/second
   ...

   this.runAnimationRate,

   this.runBehavior = {
      // Every  milliseconds, this behavior advances the
      // runner's artist to the next frame of the sprite sheet.

      lastAdvanceTime: 0,

      execute: function(sprite, time, fps) {
         if (sprite.runAnimationRate === 0) {
            return;
         }

         if (this.lastAdvanceTime === 0) {  // skip first time
            this.lastAdvanceTime = time;
         }
         else if (time - this.lastAdvanceTime > 1000 / sprite.runAnimationRate) {
            sprite.artist.advance();
            this.lastAdvanceTime = time;
         }
      }
   },
   ...
};
```

runBehavior オブジェクトの execute() メソッドによって、ランナーのアーティストは一定の時間間隔で、スプライト・シートに含まれるランナーの画像シーケンスの次の画像へと処理を進めます。(Snail Bait のスプライト・シートについては、連載第 4 回の「スプライト・アーティストとスプライト・シート」セクションを参照してください。)

runBehavior によってどの程度の頻度でランナーの画像が進められるかにより、ランナーが走る速さが決まります。その時間間隔は、ランナーの runAnimationRate 属性によって設定されます。ゲーム開始時にはランナーは走っていないため、ランナーの runAnimationRate は、最初はゼロです。しかしプレイヤーが左または右に方向を変えると、Snail Bait は runAnimationRate 属性を毎秒 17 フレームに設定し (リスト 5)、ランナーが走り出します。

リスト 5. 方向を変えると、走るアニメーションが開始される

```
SnailBait.prototype = {
   ...

   turnLeft: function () {
      this.bgVelocity = -this.BACKGROUND_VELOCITY;
      this.runner.runAnimationRate = this.RUN_ANIMATION_RATE; // 17 fps#### 4 ###
      this.runnerArtist.cells = this.runnerCellsLeft;
      this.runner.direction = this.LEFT;
   },

   turnRight: function () {
      this.bgVelocity = this.BACKGROUND_VELOCITY;
      this.runner.runAnimationRate = this.RUN_ANIMATION_RATE; // 17 fps#### 4 ###
      this.runnerArtist.cells = this.runnerCellsRight;
      this.runner.direction = this.RIGHT;
   },

};
```

時間の流れ

ランナーの「走る」ビヘイビアと同じように、ほとんどすべてのビヘイビアの根底には時間があります。また、ゲームのアニメーションは常に実行されているため、ゲームの動作を変更するための多くの関数 (リスト 5 の turnLeft() や turnRight() など) では、単にゲームの変数を設定しなおすことによってゲームの動作を変更します。ゲームが次のアニメーション・フレームを描画するときに、これらの変数がゲームの動作に影響を与えます。

turnLeft() メソッドと turnRight() メソッドは、ゲームのキーボード・イベント・ハンドラーによって呼び出され、ランナーの画像シーケンスをどの程度速く繰り返すかを、先ほど説明した runAnimationRate 属性によって制御します。この 2 つのメソッドは背景のスクロール速度を表す bgVelocity 属性を設定することにより、ランナーがどの程度速く左から右へと移動するかも制御します。

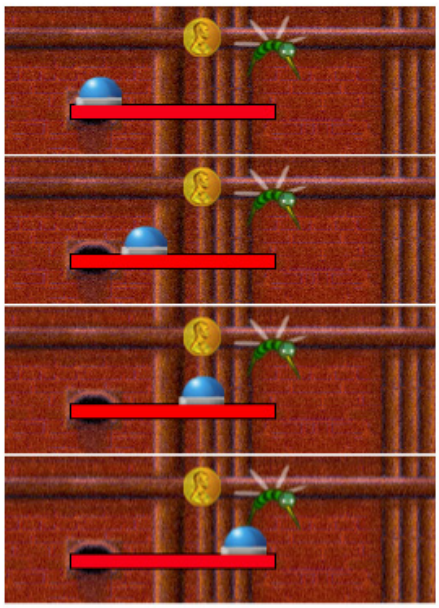
Flyweight ビヘイビア

前のセクションで説明した、ランナーの「走る」ビヘイビアは、状態 — スプライトの画像を最後に進めた時刻 — を保持します。状態を保持することで、ランナーとビヘイビアが緊密に結合されます。そのため、例えば走っているように見せたい別のスプライトがある場合には、そのスプライトの「走る」ビヘイビアが必要になります。

状態を保持しないビヘイビアは、もっと柔軟になります。例えば、状態を保持しないビヘイビアは、Flyweight として使用することができます。Flyweight は、ある 1 つのオブジェクトの単一イ

ンスタンスであり、他の多くのオブジェクトによって同時に使用されます。図 3 は、状態を持たない「横にゆっくり動く」ビヘイビアであり、このビヘイビアによってスプライトはプラットフォーム上をゆっくりと行ったり来たりするようになります。このビヘイビアの単一インスタンスは、Snail Bait のボタンやカタツムリで使用され、これらのスプライトはいずれもそれぞれの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 (図 3)。

図 3. ボタンが横にゆっくり動いているときのシーケンス



リスト 6 は Snail Bait の `createButtonSprites()` メソッドを示しています。このメソッドは各ボタンに対し、「横にゆっくり動く」ビヘイビアのみを追加します。

リスト 6. 横にゆっくり動くボタンを作成する

```
SnailBait.prototype = {
   ...

   createButtonSprites: function () {
      var button,
          buttonArtist = new SpriteSheetArtist(this.spritesheet,
                                               this.buttonCells),
      goldButtonArtist = new SpriteSheetArtist(this.spritesheet,
                                               this.goldButtonCells);

      for (var i = 0; i < this.buttonData.length; ++i) {
         if (i === this.buttonData.length - 1) {
            button = new Sprite('button',
                                goldButtonArtist,
                                [ this.paceBehavior ]);
         }
         else {
            button = new Sprite('button',
                                buttonArtist,
                                [ this.paceBehavior ]);
         }

         button.width = this.BUTTON_CELLS_WIDTH;
         button.height = this.BUTTON_CELLS_HEIGHT;
```

```
         button.velocityX = this.BUTTON_PACE_VELOCITY;
         button.direction = this.RIGHT;

         this.buttons.push(button);
      }
   },
   ...
};
```

リスト 7 は `paceBehavior` オブジェクトを示しています。

リスト 7. 「横にゆっくり動く」ビヘイビア

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

   this.paceBehavior = {
      checkDirection: function (sprite) {
         var sRight = sprite.left + sprite.width,
             pRight = sprite.platform.left + sprite.platform.width;

         if (sRight > pRight && sprite.direction === snailBait.RIGHT) {
            sprite.direction = snailBait.LEFT;
         }
         else if (sprite.left < sprite.platform.left &&
                  sprite.direction === snailBait.LEFT) {
            sprite.direction = snailBait.RIGHT;
         }
      },

      moveSprite: function (sprite, fps) {
         var pixelsToMove = sprite.velocityX / fps;

         if (sprite.direction === snailBait.RIGHT) {
            sprite.left += pixelsToMove;
         }
         else {
            sprite.left -= pixelsToMove;
         }
      },

      execute: function (sprite, time, fps) {
         this.checkDirection(sprite);
         this.moveSprite(sprite, fps);
      }
   },
```

「横にゆっくり動く」ビヘイビアは、スプライトの水平位置を変更します。このビヘイビアでは、タイム・ベースの動きを実装しており、スプライトの速度 (ピクセル/秒) をアニメーションのフレーム・レート (フレーム/秒) で割って結果 (ピクセル/フレーム) を得ることで、現在のアニメーション・フレームでスプライトを何ピクセル移動するかを計算します (タイム・ベースの動きについての詳細は、連載第 2 回の「タイム・ベースで動かす」セクションを参照)。

ゲームに固有ではないビヘイビア

Flyweight と状態

paceBehavior を Flyweight として使用できるのは、paceBehavior がステートレスであるためです。なぜ paceBehavior がステートレスかというと、paceBehavior は状態 — 各スプライトの位置と方向 — をスプライト自体の中に格納するからです。

この記事で説明した最初のビヘイビア — `runBehavior` — は、1 つのスプライトに緊密に結合されたステートフルなビヘイビアです。その次に説明した `paceBehavior` ビヘイビアは、自らを個々のスプライトから切り離したステートレスなビヘイビアであるため、単一インスタンスを複数のスプライトで使用することができます。

ビヘイビアは、さらに一般化することができます。つまり、個々のスプライトからビヘイビアを切り離すだけでなく、ゲームそのものからも切り離すことができます。Snail Bait で使用しているビヘイビアには、どのようなゲームにも使用できるビヘイビア (以下に記載します) が 3 つあります。

- `bounceBehavior`
- `cycleBehavior`
- `pulseBehavior`

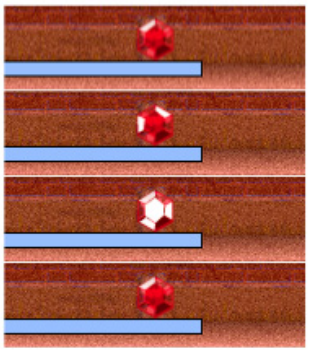
「バウンドする」ビヘイビア (bounceBehavior) は、スプライトをバウンドさせ、「繰り返す」ビヘイビア (cycleBehavior) は、スプライトの一連の画像を繰り返し描画し、「振動する」ビヘイビア (pulseBehavior) は、スプライトの透明度を操作してスプライトが振動しているように見せます。

「バウンドする」ビヘイビアと、「振動する」ビヘイビアは、どちらも今後の記事で説明するリニアでないアニメーションに関係します。一方、「繰り返す」ビヘイビアは、あるスプライトの一連の画像をリニアなアニメーションで繰り返し描画するものです。そこで、どのゲームにも使用できるビヘイビアの実装についての説明では、この「繰り返す」ビヘイビアの実装を使用することにします。

きらきら輝くルビー

Snail Bait のルビーとサファイアは、きらきら輝きます (図 4)。

図 4. ルビーがきらきら輝いているときのシーケンス



Snail Bait のスプライト・シートには、ルビーとサファイア両方の画像シーケンスが含まれており、これらの画像を繰り返し描画することで、きらきら輝いているように見せることができます。

リスト 8 は、ルビーを作成する Snail Bait のメソッドを示しています (ここには示していませんが、これとほぼ同じメソッドで、サファイアを作成することができます)。この `createRubySprites()` メソッドは、「繰り返す」ビヘイビア (CycleBehavior) を作成しています。

このビヘイビアでは、ルビーがきらきら輝く画像シーケンスの 1 つの画像を 100 ミリ秒表示すると次の画像を表示し、最後の画像の表示が完了して先頭の画像に戻った後、500 ミリ秒経過すると次のシーケンスを開始します。

リスト 8. ルビーを作成する

```
SnailBait.prototype = {
   ...
   createRubySprites: function () {
      var ruby,
          rubyArtist = new SpriteSheetArtist(this.spritesheet, this.rubyCells);

      for (var i = 0; i < this.rubyData.length; ++i) {
         ruby = new Sprite('ruby', rubyArtist,
                           [ new CycleBehavior(100,     // animation duration
                                               500) ]); // interval between animations
         ...
      }
   },
   ...
};
```

リスト 9 は、「繰り返す」ビヘイビアを示しています。

リスト 9. `CycleBehavior`

```
// This behavior advances the sprite artist through
// the sprite's images at a specified animation rate.

CycleBehavior = function (duration, interval) {
   this.duration = duration || 0;  //  milliseconds
   this.interval = interval || 0;
   this.lastAdvance = 0;
};

CycleBehavior.prototype = {
   execute: function(sprite, time, fps) {
      if (this.lastAdvance === 0) {
         this.lastAdvance = time;
      }

      // During the interval start advancing if the interval is over

      if (this.interval && sprite.artist.cellIndex === 0) {
         if (time - this.lastAdvance > this.interval) {
            sprite.artist.advance();
            this.lastAdvance = time;
         }
      }
      // Otherwise, if the behavior is cycling, advance if duration is over

      else if (time - this.lastAdvance > this.duration) {
         sprite.artist.advance();
         this.lastAdvance = time;
      }
   }
};
```

Generalizing behaviors

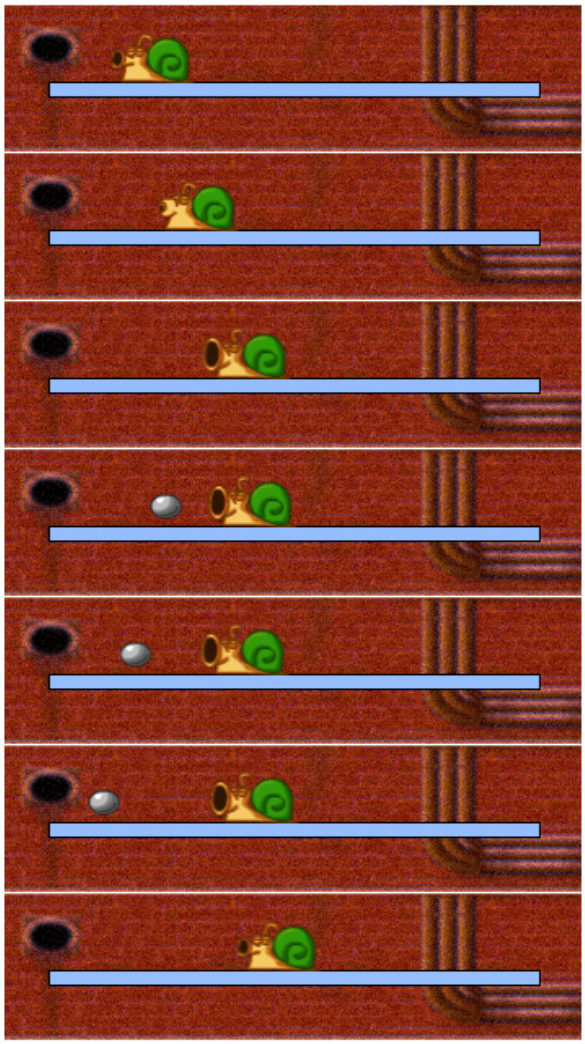
さまざまな状況でビヘイビアを使用できるように、ビヘイビアを一般化できないか検討することは賢明なことです。

この「繰り返す」ビヘイビアは、スプライト・シート・アーティストを持つ任意のスプライトで機能します。つまり、このビヘイビアは Snail Bait 専用ではなく、他のゲームで再利用することができます。リスト 4 のランナー・スプライト専用の「走る」ビヘイビアは、Snail Bait に固有ではないリスト 9 の「繰り返す」ビヘイビアと多くの共通点があり、実際、この「繰り返す」ビヘイビアは、「走る」ビヘイビアから派生したものです。(この「走る」ビヘイビアをより一般的な「繰り返す」ビヘイビアにすることもできますが、「走る」ビヘイビアではランナーのアニメーションのフレーム・レートも考慮に入れています。)

ビヘイビアを組み合わせる

個々のビヘイビアには、走る、横にゆっくり動く、きらきら輝く、などの特定のアクションがカプセル化されています。また、ビヘイビアを組み合わせて、もっと複雑なエフェクトを作り出すこともできます。例えば、カタツムリがプラットフォーム上をゆっくりと行ったり来たりしながら、周期的にカタツムリ爆弾を発射するように組み合わせることができます (図 5)。

図 5. カタツムリが爆弾を発射しているときのシーケンス



カタツムリが爆弾を発射しているときのシーケンスには、以下の 3 つのビヘイビアが組み合わされています。

- `paceBehavior`
- `snailShootBehavior`
- `snailBombMoveBehavior`

`paceBehavior` と `snailShootBehavior` はカタツムリと関連付けられており、`snailBombMoveBehavior` はカタツムリ爆弾と関連付けられています。Snail Bait はスプライトを作成する際、`paceBehavior` と `snailShootBehavior` という 2 つのビヘイビアを Sprite コンストラクターの中で指定します (リスト 10)。

リスト 10. カタツムリを作成する

```
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
                                                1500) // 1.5 seconds between sequences
                            ]);

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

         snail.velocityX = this.SNAIL_PACE_VELOCITY;
         snail.direction = this.RIGHT;

         this.snails.push(snail); // Push snail onto snails array
      }
   },
};
```

カタツムリ・オブジェクトの `cycleBehavior` は、スプライト・シートに含まれるカタツムリの各画像 (図 6) を描画するシーケンスを、前のシーケンスの完了後 1.5 秒の間隔を置いて繰り返し行います。各画像の表示時間は、300 ミリ秒です。その結果、カタツムリが周期的に口を開いたり閉じたりするように見えます。カタツムリ・オブジェクトの `paceBehavior` は、プラットフォーム上でカタツムリを行ったり来たりさせます。

図 6. カタツムリが爆弾を発射しているときのシーケンスに使用するスプライト・シートの画像

カタツムリ爆弾オブジェクトは `armSnails()` メソッドによって作成され (リスト 11)、ゲームが開始されると Snail Bait はこのメソッドを呼び出します。このメソッドは Snail Bait のすべてのカタツムリ・オブジェクトに対して処理を繰り返します。その処理の中では、カタツムリ・オブジェクトごとにカタツムリ爆弾オブジェクトを作成し、各カタツムリ爆弾オブジェクトに `snailBombMoveBehavior` を追加し、カタツムリ・オブジェクトの位置を参照することにより、カタツムリ爆弾オブジェクトの位置を決め、その位置をオブジェクトに格納します。

リスト 11. カタツムリに爆弾を装備する

```
SnailBait.prototype = {
   ...

   armSnails: function () {
      var snail,
          snailBombArtist = new SpriteSheetArtist(this.spritesheet, this.snailBombCells);

      for (var i=0; i < this.snails.length; ++i) {
         snail = this.snails[i];

         snail.bomb = new Sprite('snail bomb',
                                 snailBombArtist,
                                 [ this.snailBombMoveBehavior ]);

         snail.bomb.width  = snailBait.SNAIL_BOMB_CELLS_WIDTH;
         snail.bomb.height = snailBait.SNAIL_BOMB_CELLS_HEIGHT;

         snail.bomb.top = snail.top + snail.bomb.height/2;
         snail.bomb.left = snail.left + snail.bomb.width/2;
         snail.bomb.visible = false;

         this.sprites.push(snail.bomb);
      }
   },
};
```

カタツムリ・オブジェクトの snailShootBehavior は、カタツムリからカタツムリ爆弾を発射します (リスト 12)。

リスト 12. カタツムリ爆弾を発射する

```
SnailBait.prototype = {
   ...

   this.snailShootBehavior = { // sprite is the snail
      execute: function (sprite, time, fps) {
         var bomb = sprite.bomb;

         if (! bomb.visible && sprite.artist.cellIndex === 2) {
            bomb.left = sprite.left;
            bomb.visible = true;
         }
      }
   },

};
```

ビヘイビア・ベースのゲーム

ビヘイビア・ベースのゲームでは、基本となるインフラストラクチャーの実装が終わると、ゲームを肉付けする作業のほとんどがビヘイビアの実装になります。アニメーション、フレーム・レート、背景のスクロールなど、ゲームの基礎となるメカニズムに関する事項から解放され、ビヘイビアの実装のみにほぼ専念することで、ゲームに生命を吹き込むことができます。また実行時にさまざまなビヘイビアを組み合わせることができるため、シナリオのプロトタイプを (ビヘイビアを組み合わせることによって) 短時間で作成することができます。

`snailShootBehavior` はカタツムリ・オブジェクトに関連付けられているため、このビヘイビアの `execute()` メソッドに渡されるスプライトはカタツムリ・オブジェクトです。

カタツムリ・オブジェクトは、カタツムリ爆弾オブジェクトへの参照を保持しているため、`snailShootBehavior` はカタツムリ・オブジェクトを通じて爆弾オブジェクトにアクセスします。そして `snailShootBehavior` はカタツムリの現在の画像が図 6 の右端の画像かどうか、つまりカタツムリが口を開けようとしているところかどうかを確認します。そうである場合には、`snailShootBehavior` はカタツムリの口に爆弾を配置し、爆弾が見えるようにします。

つまり、カタツムリ爆弾を発射するには、適切な条件の下で爆弾を配置し、爆弾が見えるようにする必要があるのです。その後、爆弾を移動するのは、`snailBombMoveBehavior` の役目です (リスト 13)。

リスト 13. カタツムリ爆弾の「移動する」ビヘイビア

```
SnailBait = function () {
   this.SNAIL_BOMB_VELOCITY = 450,
   ...
};

SnailBait.prototype = {
   this.snailBombMoveBehavior = {
      execute: function(sprite, time, fps) {  // sprite is the bomb
         if (sprite.visible && snailBait.spriteInView(sprite)) {
            sprite.left -= snailBait.SNAIL_BOMB_VELOCITY / fps;
         }

         if (!snailBait.spriteInView(sprite)) {
            sprite.visible = false;
         }
      }
   },
```

カタツムリ爆弾が表示されている限り、`snailBombMoveBehavior` は `snailBait.SNAIL_BOMB_VELOCITY` で規定される速さ (毎秒 450 ピクセル) でカタツムリ爆弾を左に移動します。爆弾が表示領域外に移動されたら、snailBombMoveBehavior は爆弾を見えないようにします。

次回は

この連載の次回の記事では、ランナーの「ジャンプする」ビヘイビア (ジャンプ・ビヘイビア) について調べながら、時間とビヘイビアについてさらに詳しく説明します。その中で、ジャンプの時間を測定するために JavaScript でストップウォッチを実装する方法を紹介します。この、アニメーションの時間を測定する、という基本的な手法は、皆さんが作成するゲームで大いに活用することができます。

ダウンロード可能なリソース

内容	ファイル名	サイズ
Sample code	j-html5-game5.zip	1.2MB

関連トピック

- 『Core HTML5 Canvas』(David Geary 著、Prentice Hall、2012年): Canvas API とゲーム開発について広範にわたって説明している David Geary の著書です。関連する Web サイトとブログにもアクセスしてください。
- Snail Bait: HTML5 対応の任意のブラウザーで Snail Bait をオンラインでプレイしてみてください (Chrome のバージョン 18 またはそれ以降のバージョンが最適です)。
- HTML5 Canvas を使用した目を見張るアプリケーション: Strange Loop 2011 での David Geary のプレゼンテーションを見てください。
- 「HTML5 Game Development」: NDC (Norwegian Developer's Conference) 2011 での David Geary のプレゼンテーションを見てください。
- 「Platform games」: Wikipedia でプラットフォーム・ゲームについての説明を読んでください。
- 「横スクロール」テレビ・ゲーム: ウィキペディアで横スクロール・テレビ・ゲームについての説明を読んでください。
- 「Strategy パターン」: ウィキペディアで Strategy デザイン・パターンの項目を調べてください。
- レプリカアイランド: Android 用として人気の、オープンソースのプラットフォーム・ゲームのソースをダウンロードしてください。

© Copyright IBM Corporation 2013
(www.ibm.com/legal/copytrade.shtml)
商標
(www.ibm.com/developerworks/jp/ibm/trademarks/)