ソース説明

RefTest.htm

HTML5用の定型ヘッダ. HTML4と比べて、DOCTYPEの指定等が大幅に簡略化されている.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
```

ヘッダ部にてスクリプトファイル(RefTest.js)を指定しておく.

```
<title>REFLEXES TEST</title>
<script type="text/javascript" src="RefTest.js"></script>
```

ゲーム画面を全画面/スクロールバー無しにするためのスタイルを設定する.
この設定によって, 無駄なスペースを無くし, またブラウザサイズの変更等でゲーム画面がはみ出して邪魔なスクロールが出てしまう状況を防ぐことができる.

```
<style>
body {
  position: absolute;
  left:     0px;
  top:      0px;
  right:    0px;
  bottom:   0px;
  margin:   0px;
  overflow: hidden;
  background-color:#444;
}
</style>
</head>
```

canvasのみのページを用意する.
(キャンバスのサイズはボディサイズに応じて変化するため, ここでは指定しない.)

```
<body id="idBody">
 <canvas id="idCanvas"></canvas>
</body>
</html>
```

RefTest.js

ゲームで使用する変数を定義.
慣例として定数は全て大文字を, 変数は小文字の名前を使用する.
・idBody,idCanvas,ctxは描画等で頻繁に参照されるため、ここで取得しておく.
・size,gparam,balls,scoreはの後の処理で設定するため、ここでは空のオブジェクトや配列だけを定義する.
・stateはゲーム状態を表す. 初期状態として0(ゲーム開始前)を設定しておく.

```
window.onload = function () {
    var BALL_NUM = 15,  // ゲームに使用するボールの数
        RANK_NUM = 5,   // 表示するランキング数
        RANK_MSG = ["EXCELLENT!!", "Fantastic!", "Grate!", "Very good!", "Good!", "Clear!"],
        idBody   = document.getElementById("idBody"),
        idCanvas = document.getElementById("idCanvas"),
        ctx      = idCanvas.getContext('2d'),
        size     = {},  // サイズ依存情報
        gparam   = {},  // ゲームパラメータ
        balls    = [],  // ボール情報
        score    = [],  // スコアランキング
        state    = 0;   // ゲーム状態(0:開始前、1:ゲーム中、2:終了)
```

複数個所で使われる処理を関数にしている.
msec2strでは渡されたミリ秒整数値(v)に応じて, vが0以上の場合はs.xxx形式の文字列に0未満やundefined等の時はハイフン文字列を返すようにしている. (後者を利用して、ランキング未登録のスコアをハイフンで表示している.)

```
    // 範囲指定での乱数取得
    function randRange(min, max) {
        return Math.random() * (max - min) + min;
    }
    // ボールの色範囲定義
    function ballRGB() {
        return (randRange(80, 255) | 0) + ",";
    }
    // ミリ秒整数をs.xxx形式の文字列に変換
    function msec2str(v) {
        return (v >= 0) ? ("000" + v).replace(/0*(.+)(...)/, "$1.$2s") : "---------";
    }
```

文字の修飾描画のため、canvasのcontext2Dにメソッドを追加しておく.
このメソッドによって, 文字センタリング, 文字色指定(opt.), シャドウ指定(opt.), 浮き文字指定(opt.)での描画が簡単に行えるようになる.

なお, メソッドの中でコンテキスト自身のsave/restoreを行っているが, この処理により, メソッドの呼び出し側はコンテキストの属性がメソッド内で変更されたかどうかを意識しなくても良くなる.

```
    // context2D拡張: 修飾文字センタリング描画
    ctx.centerText = function (str, x, y, rgb, shadow, hover) {
        x -= this.measureText(str).width / 2;
        y += size.ball / 4;
        this.save();
        if (hover) { // 浮き文字色指定
            this.fillStyle = hover;
            this.fillText(str, x + size.ball / 20, y + size.ball / 20);
        }
        if (rgb) {  // フォント色指定
            this.fillStyle = rgb;
            if (shadow) { // シャドウ色指定
                this.shadowColor = shadow;
                this.shadowBlur  = size.ball / 3;
            }
        }
        this.fillText(str, x, y);
        this.restore();
    };
```

矩形の修飾描画のため、canvasのcontext2Dにメソッドを追加しておく.
このメソッドによって, 矩形のグラデーション描画が簡単に行えるようになる.

なお, 上記のcenterTextメソッドと同様に, このメソッドでも先頭と最後にコンテキストのsave/restoreを行い, コンテキストの属性がメソッド呼び出し前後で保持されるようしている.

```
    // context2D拡張: 矩形グラデーション描画
    ctx.gradRect = function (x, y, w, h, gx, gy, rgbs) { // rgbs.length >= 2
        var i    = rgbs.length,
            len1 = i - 1;
        this.save();
        this.fillStyle = this.createLinearGradient(x, y, x + w * gx, y + h * gy);
        while (i--) {
            this.fillStyle.addColorStop(i / len1, rgbs[i]);
        }
        this.fillRect(x, y, w, h);
        this.restore();
    };

    // 描画処理 ////////////////////////////////////////////////////////////////
    function drawBalls() {
        var i = BALL_NUM, b;
        while ((b = balls[--i]) !== undefined) {
            if (b.r > 0.1) {
                ctx.beginPath();
                ctx.arc(b.vec[0].pos, b.vec[1].pos, size.ball / b.r, 0, Math.PI * 2, true);
                ctx.fillStyle = b.rgb + (b.r * 0.9) + ")"; // 塗りつぶし色
                ctx.fill();
                ctx.strokeStyle = "rgba(255,255,255," + b.r + ")"; // 枠色
                ctx.stroke();
                ctx.centerText(i + 1, b.vec[0].pos, b.vec[1].pos, (b.r < 1) ? 0 : "#000"); // ボール番号
            }
        }
    }
```

ここで, 全てのボールを描画している.
ループ内のbが各ボールを指し, b.rがそのボールの状態を示している.
b.rは初期値は1に設定されるが, クリアされたボールは80%づつ減衰していき, これがボールの透過率、ボール半径の倍率に適用されてarc(円)描画される.
これにより, クリアするとボールが破裂したように見える.
なお, 十分減衰したボール(b.r <= 0.1)は描画しない.

gparam.drawfuncにステート別の描画処理を定義している.
state=0はゲーム開始前のため、表題とクリックで開始する旨の表示を行う.

```
    gparam.drawfunc = [function () {
        // 表題描画
        ctx.gradRect(size.hx / 4, size.hy / 2 - size.ball, size.hx * 1.5, size.ball * 2, 1, 1, ["#f00", "#400", "#f00"]);
        ctx.centerText("REFLEXES TEST",   size.hx, size.hy / 2, "#ff6", "#fd0", "#100");
        ctx.centerText("Click to start!", size.hx, size.hy + size.ball, "#fff", "#f88");
    }, function () {
```

state=1はゲーム中画面. ここではボールと経過時間を表示するのみ.

```
        // 本編描画
        drawBalls();
        ctx.centerText("Time: " + msec2str(Date.now() - gparam.startms), size.hx, size.hy, "rgba(255,255,255,0.5)");
    }, function () {
```

state=2はゲームクリア後の状態.
ここでは, クリアした時のスコア(時間)と記録されているランキングを表示する.
もし, 今回のスコアがランクインしていた場合にはランクに応じたクリアメッセージを表示するほか, 該当のランキング位置を(透過率を変化することで)点滅させ, 何位になったかが一目で分かる様にしている.
メソッドの最後でボールを表示しているが, これはゲームクリアしても, 破裂したボールの残像が暫く残るため処理を入れている.

```
        // 結果描画
        var sy = idCanvas.height / 9,
            i = RANK_NUM;
        ctx.centerText(RANK_MSG[gparam.rank], size.hx, sy, "#ffd", "#ffa", "#880");
        ctx.centerText("Your score is " + msec2str(gparam.startms), size.hx, sy * 2, "#ddf", "#33f");

        // ランキング
        ctx.gradRect(size.hx / 2, sy * 2.5, size.hx, sy, 1, 1, ["#6f6", "#070"]);
        ctx.strokeStyle = "#bfb";
        ctx.strokeRect(size.hx / 2, sy * 2.5, size.hx, sy * 6);
        ctx.centerText("RANKING", size.hx, sy * 3, "#fff", "#8f8");
        while (i--) {
            ctx.centerText(i + 1 + ": " + msec2str(score[i]), size.hx, sy * (i + 4),
                           (i === gparam.rank) ? "rgba(255,255,128," + (Date.now() % 1000 / 500) + ")" : "#fff", "#ff0");
        }
        drawBalls(); // ボールは残像があるためクリア後も描画しておく
    }];
```

Copyright (c) 2012 GPS Labo All rights reserved.

ソース説明

ステート共通の背景を描画した後に，上記の3つのステート別メソッドへ振り分けるための関数．

```
function draw() {
    ctx.gradRect(0, 0, idCanvas.width, idCanvas.height, 0, 1, ["#000", "#700"]); // 背景
    gparam.drawfunc[state]();
}
```

この処理は、ゲームを進めていくために定期的に呼び出される関数．

クリア後のボールは破裂表現のため80%づつ減衰させ，クリア前のボールは加速度をランダムに変化させながら，移動させる．
なお，ゲームバランス難易度調整の為，加速度や速度速度が大きくなりすぎないようにチェックしながらボールの移動パラメータを設定している．
また，ボールが画面外に飛び出さないようにキャンバスサイズで壁衝突の判定を行い，範囲外なら跳ね返るようにしている．
全てのボールの状態が更新されたら，最後にdraw関数を呼び出して，新しい場所にボールを描画する．
この処理の繰り返しによって，ボールが動いているように見える．

```
// タイマー更新 ////////////////////////////////////////////////////////
function ontimer() {
    var i = BALL_NUM, j, k, b;
    while ((b = balls[--i]) !== undefined) {
        if (i < gparam.count) {
            // ボールのフェードアウト
            b.r *= 0.8;
        } else {
            // ボール移動(加速度ランダム変化)
            for (j = 0; j < 2; ++j) { // (x,y)
                k = b.vec[j];
                k.pos += (k.v = Math.min(Math.max(k.v + randRange(-size.accel, size.accel), -size.speed), size.speed));
                if (k.pos < size.ball || size.area[j] <= k.pos) {
                    // 壁衝突
                    k.pos = ((k.pos < size.ball) ? size.ball : size.area[j]) * 2 - k.pos;
                    k.v  *= -1;
                }
            }
        }
    }
    draw();
}
```

gparam.clickfuncに、ステート別のクリック(タッチ)処理を定義している．

ここではstate=0(ゲーム開始前)のクリック処理として，全ボールをランダムに配置してクリアボール数や経過時間を初期化し，ゲームを開始する．
なお，ここで上記ontimer関数を定期的に呼び出すための設定を行っているが，このゲームでは30FPS程になるよう呼び出し間隔を設定している．

```
// クリック or タッチ制御 ////////////////////////////////////////////////
gparam.clickfunc = [function () {
    // ゲーム開始
    var i;
    for (i = 0; i < BALL_NUM; i++) { // 全ボールをランダムに設置
        balls[i] = {vec : [{pos: randRange(size.ball, size.area[0]), v: 0},
                           {pos: randRange(size.ball, size.area[1]), v: 0}],
                    r   : 1,
                    rgb : "rgba(" + ballRGB() + ballRGB() + ballRGB()};
    }
    gparam.count    = 0;
    gparam.timerid  = setInterval(ontimer, 1000 / 30); // 30fps
    gparam.startms  = Date.now();
    state = 1;
}, function (e) {
    // ボールヒット判定
    var i = (e.touches) ? e.targetTouches[0] : e,   // タッチ or マウス
        vec = balls[gparam.count].vec;
    if (Math.pow(vec[0].pos - i.clientX, 2) + Math.pow(vec[1].pos - i.clientY, 2) <= Math.pow(size.ball, 2)) {
        // ボールヒット!
        if (++gparam.count >= BALL_NUM) { // クリア(最終ボール)判定
            gparam.startms = Date.now() - gparam.startms;

            // ランキングチェック
            for (i = 0; i < RANK_NUM; ++i) {
                if (!score[i] || score[i] > gparam.startms) { // ランキング更新?
                    score.splice(i, 0, gparam.startms);
                    score.splice(RANK_NUM);
                    break;
                }
            }
            gparam.rank = i; // 今回のランキング位置 (ランク外はRANK_NUM)
            state = 2;
        }
    }
    //else {お手つきのペナルティは特に無し}
}, function () {
    // スタート画面復帰
    clearTimeout(gparam.timerid);
    state = 0;
}];
```

ここではstate=1(ゲーム中)のクリック処理として，クリック位置にターゲットのボールがあるか検査している．
ボールにヒットしていれば，そのボールをクリアして、さらに全てのボールをクリアしていれば，ゲームクリアのためランキング判定処理を行う．
なお，クリック位置にボールが無い場合にも特にペナルティはないが，もし必要であれば，if文最後のelse箇所のコメントに処理を記述すれば，お手付き発生時の動作を追加することができる．

ここではstate=2(ゲームクリア)時のクリック処理．state=0の時に設定したontimerを停止する処理のみ．

この処理はブラウザのサイズに応じて変化するパラメータの計算のためのメソッドです．size.ballにて高さ/幅の小さい方の1/5のサイズを基準として設定し，その他のパラメータも，そのサイズを基に計算している．
これにより縦画面や横画面などの様々な画面サイズに対応することができる．

また，ゲーム中に画面サイズが変更された場合を考慮し，サイズ変更完了後には再度描画処理を呼び出し，新しいサイズで画面を更新している．

```
// 画面サイズ依存のパラメータ設定 ////////////////////////////////////////
function onresize() {
    // 画面サイズ、ボールサイズ設定等
    size.hx     = idBody.offsetWidth  / 2;
    size.hy     = idBody.offsetHeight / 2;
    size.ball   = Math.min(size.hx, size.hy) / 5 | 0; // ボールサイズ
    size.area   = [(idCanvas.width  = idBody.offsetWidth)  - size.ball,
                   (idCanvas.height = idBody.offsetHeight) - size.ball];
    size.speed  = size.ball / 5;    // 最大速度  (ここで難易度調整)
    size.accel  = size.ball / 100;  // 最大加速度(ここで難易度調整)

    // 描画サイズパラメータはボールサイズ基準でスケーリング
    ctx.font  = size.ball * 0.6 + "pt Arial";
    ctx.lineWidth = size.ball / 40;

    // サイズ変更後は再描画
    draw();
}
```

起動時の画面サイズで初期化する．

ブラウザのサイズ変更とクリック(またはタッチ)の時に実行される処理を定義している．

```
// 初期化処理 ////////////////////////////////////////////////////////////
onresize();
window.addEventListener("resize", onresize, false);
idBody.addEventListener((navigator.userAgent.search(/Android|iPad|iPhone|iPod/) < 0) ? "mousedown" : "touchstart",  function (e) {
    gparam.clickfunc[state](e);
    draw();
    e.preventDefault(); // iOSでのフリック動作抑制のため
}, false);
};
```

Copyright (c) 2012 GPS Labo All rights reserved.