Web technology for engineers
Laboration Pong

John-Patrik Nilsson
e-mail: jpatrik.nilsson@gmail.com
Skype: j-p.nilsson

April 6, 2010

1 Abstract

This report describes an example of how to program the classic game Pong using JavaScript (ECMAScript) and HTML.

2 Laboration

2.1 Introduction & purpose

This document intends to describe how a laboration about JavaScript was conducted. The main objective of the laboration was to learn basic knowledge about the JavaScript language. To achieve this, the laboration consisted of several lesser, concrete, objectives (assignments) which is described in detail below.

2.2 Method & tools

A website consisting of an HTML document and a JavaScript document to be used as a starting point were supplied at the start of this laboration. The website consisted of a representation of the classical game Pong, a game in which two players tries to win over the other by directing a ball to a specific place by the use of a paddle.

The laboration consisted of solving six assignments, which in turn consisted of modifying and/or enhancing the JavaScript code supplied at the start of the laboration. As an extra in-depth assignment, the JavaScript document providing the website functionality were re-written from scratch.

The result of this laboration is the website code consisting of the HTML document as well as two JavaScript documents, and they can be found in the Appendix of this document.

The assignments which this laboration consisted of were solved merely by the use of a text editor (vim) for code editing and the Mozilla Firefox browser for testing purposes.

3 Assignments

3.1 1. Comment the code

The first assignment was to interpret the supplied JavaScript code and to insert comments in the code document so as to make it more readable. This was really needed, the code was not easy to read or to interpret, the code was functioning, but it proved very hard to add new functionality to it and to debug it.

3.2 2. Increase the speed of the game

By this point the new JavaScript document which was written from scratch were used instead of the one supplied at the start of the laboration.

The movement of the ball were governed by calls to a moveBall function during intervals. For this, the setInterval function of the JavaScript API was used. The setInterval function calls another function (in this case moveBall) in intervals measured in milliseconds.

The original JavaScript document called this function only once, and used the ball movement in pixels to represent the speed of the game. That is, for the game speed to increase, the ball had to move additional pixels per interval. The ball thus "skipped" forward; the faster the game went, the more pixels the ball skipped ahead. The advantage of this was that the game didn't require more system resources.

The new JavaScript document does not use ball movement to represent speed. Instead this functionality is implemented by adding more calls to setInterval and thus also moveBall. This results in that moveBall gets called with increasing frequency, and this makes the ball travel faster. The advantage of this approach is that the ball moves more smoothly, as it always just moves one pixel, it's just that it moves one pixel with faster intervals. Another advantage of this is that there is less risk that the game will produce errors when calculating movement. For example; say that the paddle is 10 pixels wide, if the ball speed is implemented using "skips" and the ball moves with 14 pixel each interval, there is a risk that the ball will "skip" over the paddle, ignoring it. This is never the issue if the ball always just moves one pixel. However, there is one disadvantage with implementing this approach; each call to setInterval dramatically increase the system resources (particularly the CPU usage) consumed by the JavaScript. This is however negotiated by the code written for this laboration by setting a max limit of intervals which is to be added to the game.

To make the game more interesting and real, the speed increase is configured to occur randomly when the ball hits the side of a paddle which faces toward the center.

3.3 3. Increase & decrease the paddle sizes

The functionality of increasing and decreasing the height of the paddles were implemented by adding functions to the paddle objects in the JavaScript code. These functions modified the height of the HTML DOM elements representing the paddles.

3.4 4. Add goals

Two red rectangular objects were added to the HTML document at either side. These were intended to represent the goals where the players would ideally try to place the ball to score. This functionality was implemented by the use of conditional if/else blocks and simple coordinates to determine if the ball was in the respective goal.

3.5 5. Paddle movement

The event manager of the game was extented to enable the paddles to move horizontally in addition to vertically. The exact same logic for horizontal movement as for vertical movement was used.

3.6 6. Sound

This functionality was not implemented, but could very well be by adding a dummy HTML element to the HTML document as a placeholder, and then modify the contents of that element to embed sound whenever sound is required.

The HTML document addition:

```
<span id=dummyspan></span>
```

The JavaScript document addition:

```
var playSound = function () {
    document.getElementById('dummyspan').innerHTML =
    "<embed src=\"success.wav\" hidden=true autostart=true loop=false>";
}
```

A Resources

A.1 References

http://javascript.crockford.com/code.html
http://javascript.crockford.com/private.html
http://thepenry.net/jsrandom.php
http://www.phon.ucl.ac.uk/home/mark/audio/play.htm

A.2 Code listing

A.2.1 Pong.html

This is the website HTML document.

```
<html>
<head>
<script type="text/javascript" language="JavaScript"
    src="pingpong.js"></script>
</head>
<body onKeyPress="captureKey(event);">
```

```
<div id="court" style="position:absolute;top:100px;left:30px;
    z-index:1;border:solid blue 4px;width:750px;height:400px;">

<img src="ball.gif" id="ball1" style="position:absolute;
    top:10px;left:25px;z-index:2;">

<div id="leftpaddle" style="position:absolute;top:4px;left:15px;
    z-index:1;height:70px;width:10px;background-color:black;"></div>
<div id="centerline" style="position:absolute;top:0px;left:375px;
    z-index:1;height:400px;border-left:dashed blue 1px;"></div>
<div id="rightpaddle" style="position:absolute;top:4px;left:725px;
    z-index:1;height:70px;width:10px;background-color:black;"></div>

<div id="messageBox" style="position:absolute;top:100px;left:235px;
    z-index:3;border:yellow solid 10px;
padding:10px;width:250px;height:125px;background-color:white;
    visibility:hidden;"></div>

<div id="leftgoal" style="position: absolute; top: 150px; left: -6px;
    width: 6px; height: 100px; background-color: red;"></div>

<div id="rightgoal" style="position: absolute; top: 150px;
    left: 750px; width: 6px; height: 100px; background-color: red;"></div>
</div>
<button onclick="startGame();">Start Play!</button><br />
<div>Left paddle keyboard bindings: left: a, right: d, up: w, down: s. <br />
Right paddle keyboard bindings: left: j, right: l, up: i, down: k. <br />
Increase paddle sizes: 6, decrease paddle sizes: 5.
</div>
<div>Paddle must be still to hit the ball!</div>
</body>
</html>
```

A.2.2 pingpong.js

This is the re-written JavaScript document.

```
/*
 * Author: mindbug.
 */

/*
 * Problem: The ball sometimes does not "get hit" by a paddle, especially
 * if the paddle moves.
 * This might have something do to with the paddles movement; it moves
 * x pixels, whereas the ball moves 1. It also might have something to do
 * with the width of different elements, so that the logic is correct,
 * but not the visual representation.
 *
 * Note about functions,
 *   function blah() {}
 * gets translated to
 *   var blah = function () {};
 * by the compiler at runtime. This program uses only the latter
 * for consistency.
 */

/*
 * TODO
 * Don't call this.getDom() so much!
 */
var ball = {
    /*
     * The direction is the opposite of what it should be, logically,
     * because the ball starts in contact with the pad,
     * so the first move it does is to change dir to the right.
     */
    direction: -1,
    ydirection: 1,
    getDom: function () {
        return document.getElementById("ball1");
    },
    getWidth: function () {
        return parseInt(this.getDom().width);
    },
    getHeight: function () {
        return parseInt(this.getDom().height);
    },
    changeXDir: function () {
        this.direction = this.direction * -1;
    },
    going: function (dir) {
        switch (dir) {
        case 'left':
```

```
            if (this.direction < 0) {
                return true;
            } else {
                return false;
            }
        break;
        case 'right':
            if (this.direction >= 0) {
                return true;
            } else {
                return false;
            }
        break;
        case 'up':
            if (this.ydirection < 0) {
                return true;
            } else {
                return false;
            }
        break;
        case 'down':
            if (this.ydirection >= 0) {
                return true;
            } else {
                return false;
            }
        break;
        }
},
changeYDir: function () {
    this.ydirection = this.ydirection * -1;
},
prepServe: function (side) {
    this.getDom().style.top = "10px";
    this.ydirection = 1;
    switch (side) {
    case 'left':
        this.getDom().style.left = "25px";
        this.direction = -1;
    break;
    case 'right':
        this.getDom().style.left = 725 - this.getWidth() + "px";
        this.direction = 1;
    break;
    }
},
move: function () {
    this.getDom().style.left = this.edge('left') +
        this.direction + 'px';
    this.getDom().style.top = this.edge('top') +
```

```
            this.ydirection + 'px';
    },
    edge: function (dir) {
        switch (dir) {
        case 'left':
            return parseInt(this.getDom().style.left);
        break;
        case 'right':
            return this.edge('left') + this.getWidth();
        break;
        case 'top':
            return parseInt(this.getDom().style.top);
        break;
        case 'bottom':
            return this.edge('top') + this.getHeight();
        break;
        }
    }
};

var court = {
    edge: function (dir) {
        switch (dir) {
        case 'left':
            return 0;
        break;
        case 'right':
            return 750;
        break;
        case 'top':
            return 0;
        break;
        case 'bottom':
            return 400;
        break;
        }
    }
};

/*
 * Constructor example, without the use of the 'new' operator.
 * Uses object literal: {}.
 */
var makePad = function (html_id) {
    return {
    getDom: function () {
        return document.getElementById(html_id);
    },
    getWidth: function () {
        return parseInt(this.getDom().style.width);
```

```
},
getHeight: function () {
    return parseInt(this.getDom().style.height);
},
getLeft: function () {
    return parseInt(this.getDom().style.left);
},
getRight: function () {
    return this.getLeft() + this.getWidth();
},
getTop: function () {
    return parseInt(this.getDom().style.top);
},
increaseSize: function () {
    this.getDom().style.height = this.getHeight() + 10 + "px";
},
decreaseSize: function () {
    this.getDom().style.height = this.getHeight() - 10 + "px";
},
getBottom: function () {
    return this.getTop() + this.getHeight();
},
prepServe: function (side) {
    this.getDom().style.top = "4px";
    this.getDom().style.height = "70px";
    switch (side) {
    case 'left':
        this.getDom().style.left = "15px";
    break;
    case 'right':
        this.getDom().style.left = "725px";
    break;
    }
},
move: function (dir) {
    switch (dir) {
    case 'left':
        this.getDom().style.left = this.getLeft() - 10 + 'px';
    break;
    case 'right':
        this.getDom().style.left = this.getLeft() + 10 + 'px';
    break;
    case 'up':
        this.getDom().style.top = this.getTop() - 10 + 'px';
    break;
    case 'down':
        this.getDom().style.top = this.getTop() + 10 + 'px';
    break;
    }
},
```

```
    edge: function (dir) {
        switch (dir) {
        case 'left':
            return this.getLeft();
        break;
        case 'right':
            return this.getRight();
        break;
        case 'top':
            return this.getTop();
        break;
        case 'bottom':
            return this.getBottom();
        break;
        }
    }
    };
};

/*
 * Create two objects by using the constructor function above.
 */
var leftPad = makePad('leftpaddle');
var rightPad = makePad('rightpaddle');

/*
 * The following function is used to increase the speed of the
 * game. The amount of calls to setInterval should be severely limited
 * because they are very CPU-intensive.
 *
 * The speed is increased by doing more calls to setInterval, which
 * results in that moveBall() gets called with increased frequency.
 */
var intervals = [];
var increaseGameSpeed = function () {
    if (Math.floor(Math.random() * 5 + 1) == 1 &&
        intervals.length <= 3) {
        intervals.push(setInterval('moveBall()', 20));
    }
};

var startGame  = function () {
    if (intervals.length < 1) {
        intervals.push(setInterval('moveBall()', 10));
    }
};

/*
 * Controls the ball and the game. This logic could have been
 * included in the ball object, but it was a design decision to
```

```
 * keep the ball from knowing about its surroundings.
 */
var moveBall = function () {
    if (touchesNet('left')) {
        score('right');
    } else if (touchesNet('right')) {
        score('left');
    }

    if (touchesSide('left') && ball.going('left')) {
        ball.changeXDir();
    } else if (touchesSide('right') && ball.going('right')) {
        ball.changeXDir();
    }
    if (touchesSide('top') && ball.going('up')) {
        ball.changeYDir();
    } else if (touchesSide('bottom') && ball.going('down')) {
        ball.changeYDir();
    }
    ball.move();
};

/*
 * Determine if some side of the ball touches anything.
 * This function also increases the speed of the game when
 * the ball hits a pad, however this is not the ideal place to
 * implement that functionality.
 */
var touchesSide = function (side) {
    switch (side) {
    case 'top':
        /*
         * If top side touches upper wall,
         * else if it touches left(right)Pads bottom side.
         */
        if (ball.edge('top') == court.edge('top')) {
            return true;
        } else if (ball.edge('top') == leftPad.edge('bottom') &&
            ball.edge('left') <= leftPad.edge('right') &&
            ball.edge('right') >= leftPad.edge('left')) {
            return true;
        } else if (ball.edge('top') == rightPad.edge('bottom') &&
            ball.edge('left') <= rightPad.edge('right') &&
            ball.edge('right') >= rightPad.edge('left')) {
            return true;
        } else {
            return false;
        }
    break;
    case 'bottom':
```

```
    /*
     * If bottom side touches bottom wall,
     * else if it touches left(right)Pads top side.
     */
    if (ball.edge('bottom') == court.edge('bottom')) {
        return true;
    } else if (ball.edge('bottom') == leftPad.edge('top') &&
        ball.edge('left') <= leftPad.edge('right') &&
        ball.edge('right') >= leftPad.edge('left')) {
        return true;
    } else if (ball.edge('bottom') == rightPad.edge('top') &&
        ball.edge('left') <= rightPad.edge('right') &&
        ball.edge('right') >= rightPad.edge('left')) {
        return true;
    } else {
        return false;
    }
break;
case 'left':
    /*
     * If left side of ball touches left wall,
     * else if left side of ball touches right side of any pad.
     */
    if (ball.edge('left') == court.edge('left')) {
        return true;
    } else if (ball.edge('left') == leftPad.edge('right') &&
        ball.edge('top') <= leftPad.edge('bottom') &&
        ball.edge('bottom') >= leftPad.edge('top')) {
        /* Increase the speed. Ideally this should not
         * be called here, but instead in the moveBall()
         * function.
         */
        increaseGameSpeed();
        return true;
    } else if (ball.edge('left') == rightPad.edge('right') &&
        ball.edge('top') <= rightPad.edge('bottom') &&
        ball.edge('bottom') >= rightPad.edge('top')) {
        return true;
    } else {
        return false;
    }
break;
case 'right':
    /*
      * If right side of the ball touches the right wall,
      * else if the right side of the ball touches the left
     * side of any pad.
     */
    if (ball.edge('right') == court.edge('right')) {
        return true;
```

```
        } else if (ball.edge('right') == leftPad.edge('left') &&
            ball.edge('top') <= leftPad.edge('bottom') &&
            ball.edge('top') >= leftPad.edge('top')) {
            return true;
        } else if (ball.edge('right') == rightPad.edge('left') &&
            ball.edge('top') <= rightPad.edge('bottom') &&
            ball.edge('top') >= rightPad.edge('top')) {
            /* Increase the speed. Ideally this should not
             * be called here, but instead in the moveBall()
             * function.
             */
            increaseGameSpeed();
            return true;
        } else {
            return false;
        }
    break;
    }
};

/*
 * Objects could have been made to represent the goals, but instead
 * the static numbers were used, as per the Pong.html file.
 */
var touchesNet = function (side) {
    if (ball.edge(side) == court.edge(side) &&
        ball.edge('top') <= 250 &&
        ball.edge('bottom') >= 150) {
        return true;
    } else {
        return false;
    }
};

var captureKey = function (e) {
    var keycode;
    /*
     * This is a cross-browser hack.
     */
    if (e.keyCode) {
        keycode = e.keyCode;
    } else {
        keycode = e.which;
    }
    switch (String.fromCharCode(keycode)) {
        case 'w':
            leftPad.move('up');
        break;
        case 's':
            leftPad.move('down');
```

```
        break;
        case 'a':
            leftPad.move('left');
        break;
        case 'd':
            leftPad.move('right');
        break;
        case 'i':
            rightPad.move('up');
        break;
        case 'k':
            rightPad.move('down');
        break;
        case 'j':
            rightPad.move('left');
        break;
        case 'l':
            rightPad.move('right');
        break;
        case '5':
            leftPad.decreaseSize();
            rightPad.decreaseSize();
        break;
        case '6':
            leftPad.increaseSize();
            rightPad.increaseSize();
        break;
    }
};

var scorePop;
var score = function (side) {
    /*
     * Clear the intervals which calls moveBall()
     * and remove them from the array.
     */
    while (intervals.length > 0) {
        clearInterval(intervals.pop());
    }
    msgBox = document.getElementById('messageBox');
    msgBox.style.visibility = "visible";
    scorePop = "<p>Point for the "+side+" side!<br /> ";
        scorePop += "To serve the ball again, press the button " +
        "after closing this pop-up.</p>";
        scorePop += "<a href=\"javascript:return false\"" +
        "onClick=\"prepServe('" +
        side + "');return false;\">X Close</a>";
         msgBox.innerHTML = scorePop;
};
```

```
var prepServe = function (side) {
        msgBox.style.visibility='hidden';
    ball.prepServe(side);
    /*
     * TODO
     * Fix these, they have stupid names.
     */
    leftPad.prepServe('left');
    rightPad.prepServe('right');
};
```

A.2.3 commented-pingpong.js

This is the original JavaScript document with additional comments.

```
/**
 * @author Lisha
 */
/** @comments J-P
 */

// ECMAScript calculates x and y coordinates as if origo (0) was at the upper
// left corner of the document.

var dirx = 1;
// diry sets the vertical direction of the ball.
var diry = 1;
// spdx and spdy are speed variables controlling how fast the ball
// will travel horizontally and vertically respectively.
var spdx = 10;
var spdy = setRand();

var imgLeftInt;
var imgTopInt;
var imgHeight;
var imgWidth;
var winWidth;
var winHeight;
var theBall;

// timer control var
var t=0;
// paddle location vars
var rInt, lInt;

function startGame() {
// setInterval calls and loops animBall() every 80 millisecs.
// if t != 0, it means that theres an ongoing game, so a new
// game will not be started.
 if(t==0){
 t = setInterval('animBall()', 80);
 }
}

// If this function gets called every 80 milliseconds, why does it
// include so many initiations which only should exist once?
function animBall() {
 theBall = document.images['ball1'];
 // find out the size of the ball.
 imgLeftInt = parseInt(theBall.style.left);
 imgTopInt = parseInt(theBall.style.top);
 imgHeight =  parseInt(theBall.height);
```

```
imgWidth =  parseInt(theBall.width);
// This variable sets the x coordinate the left edge of the ball must have for
// the left player to win, that is, imgLeft > winWidth.
// The right player wins if the left edge of the ball is lower than 10,
// that is, imgLeftInt < 10.
// So in essence, the left edge of the ball image, imgLeftInt, is between 10
// and winWidth, else the game stops and some player scores.
// But the million dollar question is why these variables are set every
// time animBall is called.
winWidth = 735;
// This variable should really be called courtHeight or something similar,
// since it has nothing to do with winning or scoring.
winHeight = 390;

// Initiate the paddles.
PadlLoc();
// Initiate and determine the innermost edges of the paddles,
// that is, the horizontal coordinates that the ball will bounce
// against.
padRight=parseInt(rpadl.style.left);
padLeft=parseInt(lpadl.style.left)+parseInt(lpadl.style.width);
// If the right edge of the ball (imgLeftInt+imgWidth) is touching
// the right paddle, call the rPadlHit function.
if ((imgLeftInt+imgWidth)== padRight){
rPadlHit();
// Or if the left edge of the ball touches the left paddle,
// call the lPadlHit function.
} else     if ((imgLeftInt)== padLeft){
lPadlHit();
}
// Then determine which way the ball should go and call the
// appropriate functions.
if(dirx == 1){                              // if I should go right...
goRight();
} else {                                    // otherwise, I'd better go left!
goLeft();
}

if(diry == 1) {                            // if I should go down...
goDown();
} else {                                   // otherwise, I'll go up!
goUp();
   }

}
// These functions gets called whenever the ball hits a paddle.
// The ball has hit the right paddle.
function rPadlHit(){
    // rInt is the y coordinate for the top edge of the right
    // paddle.
```

```
    // rBottom is set to the y coordinate for the bottom edge
    // of the right paddle.
     rBottom = rInt+parseInt(rpadl.style.height);
    // rTop is set to the y coordinate for the top edge of the
    // right paddle minus the ball height.
    // imgHeight represents the height of the ball image.
    // rTop is calculated weirdly because we do not have the
    // vertical coordinate for the balls lower edge.
     rTop = rInt-imgHeight;
         // If the top of the ball is less than (over) the lower edge
    // of the right paddle. AND
    // IF the top of the ball (plus its height so that we get
    // the bottom) is more than (below) the top edge of the
    // right paddle.
     if ((imgTopInt < rBottom ) && (imgTopInt > rTop)){
         // Set the direction of the ball to left.
    dirx=0;
     }
}
// The ball has hit the left paddle.
// Same explanation as the function above, just switch the logic.
function lPadlHit(){
    lBottom = lInt+parseInt(lpadl.style.height);
    lTop = lInt-imgHeight;
    if ((imgTopInt < lBottom ) && (imgTopInt > lTop)){
        dirx=1;
    }
}
// This function initiates and returns the html elements
// representing the right and left paddles, as well as int variables
// representing their respective y coordinates as seen from the top.
function PadlLoc() {
    rpadl = document.getElementById('rightpaddle');
    rInt = parseInt(rpadl.style.top);
    lpadl = document.getElementById('leftpaddle');
    lInt = parseInt(lpadl.style.top);
    return rpadl,rInt, lpadl, lInt;
}
// These two functions controls the horizontal movement of the ball, as well as
// determines if someone has scored or not, and if so, which player.
// The movement speed is controlled by the variable spdx, and is fixed.
// If someone scored the function calls another function called score.
function goRight() {
     theBall.style.left = imgLeftInt+spdx +"px";
     if (imgLeftInt >  (winWidth-(imgWidth))){
    score('left');
     }
}
function goLeft() {
     theBall.style.left = (imgLeftInt-spdx) +"px";
```

```
     if (imgLeftInt <  10){
    score('right');
     }
}
// These two functions controls the vertical movement of the ball.
// The movement speed is controlled by the spdy variable, which is
// randomized between 2 and 7 as per the setRand function.
function goDown() {
    theBall.style.top = imgTopInt+spdy+"px";
    // If the bottom of the ball (imgTopInt+imgHeight) is going to go (+spdy)
    // beyond the bottom of the court (winHeight) by the next call to goDown,
    // change the vertical direction
    // of the ball movement and set a new random speed.
    if (imgTopInt >  (winHeight-(imgHeight+spdy))){
        diry = 0;
        spdy= setRand();
    }
}

function goUp() {
    theBall.style.top = (imgTopInt-spdy) +"px";
    // If the top of the ball is too near the top of the court (5)
    // change the vertical direction of the ball movement and
    // set a new random speed.
    if (imgTopInt < 5){
        diry = 1;
        spdy= setRand();
    }
}
// Return a random number in the interval 2 to 7.
function setRand() {
        randnum= Math.floor(Math.random()*6)+2;
        return randnum;
}
// These two functions controls the right and left paddle movements.
function rPadMove(dir) {
    // If the direction to move is up and if it is possible to move
    // up without going beyond the court boundaries.
    if(dir==1 && rInt>5){
        // Move the right paddle up 12 pixels.
           rpadl.style.top = (rInt-12)+"px";
    // Or if the direction to move is down and if it is possible
    // to do so without going beyond the court boundaries.
    } else if (dir==0 && rInt < 320){
        // Move the left paddle down 12 pixels.
         rpadl.style.top = (rInt+12)+"px";
    }
}
    // Precisely the same logic as the above function, just reverse
    // the variables and directions.
```

```
 function lPadMove(dir) {
    if(dir==1 && lInt > 5){
        lpadl.style.top = (lInt-12)+"px";
    } else if(dir==0 && lInt < 320){
        lpadl.style.top = (lInt+12)+"px";
    }
 }

// variables to determine what a key press means
var rup = '9';  // right paddle up
var rdn = '0';  // right paddle down
var lup = '1';  // left paddle up
var ldn = '2';  // left paddle down
function CaptureKey(e) {
    // We want to get the keycode for the pressed key which
    // triggered the event.
    // The following if block achieves this, and is necessary
    // to make it function in all major browsers.
    // Some browsers uses e.keyCode, and others uses e.which,
    // to get the correct keycode.
   if (e.keyCode) {
        keycode=e.keyCode;
   }
     else {
         keycode=e.which;
     }
    // This following line obviously converts the keycode to
    // a string object.
   move=String.fromCharCode(keycode);

      // The rest of the function interprets the commands given and
    // executes the correct function.
      if (move == "s") {
          startGame();
      }
      if (move == rup) {
        rPadMove(1);
     }
      if (move == rdn) {
        rPadMove(0);
     }
      if (move == lup) {
          lPadMove(1);
     }
      if (move == ldn) {
        lPadMove(0);
     }
}
// This function determines what should be done if anyone scores.
function score(side) {
```

```
// Stop the timer, that is, stop the ball from animating.
    clearInterval(t);
    // Again, why does msgBox get initiated every time someone
    // scores?
    msgBox = document.getElementById('messageBox');
        // Show the score box.
    msgBox.style.visibility = "visible";

    scorePop = "<p>Point for the "+side+" side!<br /> ";
    scorePop += "To serve the ball again, press \"s\" after closing this pop-up.</p>";
    scorePop += "<a href=\"javascript:return false\" onClick=\prepNewGame('"+side+"');retu

     msgBox.innerHTML = scorePop;
}

function prepNewGame(side){
// Reset the timer ID variable (t).
// Reset all values.
    t=0;
    msgBox.style.visibility='hidden';
    theBall.style.top = "10px";
// determine which side serves and set the correct positions for the
// ball and the paddles.
    if(side == "left"){
    theBall.style.left = "25px";
    rpadl.style.top = "5px";
    lpadl.style.top = "5px";
    } else {
    theBall.style.left = (winWidth-40) + "px";
    rpadl.style.top = "5px";
    lpadl.style.top = "5px";
    }

}
```