IE Developer Center Column:
HTML5 Game Development Techniques

Seth McEvoy, Microsoft

Detecting Collisions with SVG Bounding Boxes

SVG (Scalable Vector Graphics) is more than a graphics markup language. SVG has many additional capabilities because it provides *methods* that make it possible to perform actions on objects. These SVG methods have valuable features that can enhance web game development. This article will show you how the SVG **getBBox** method uses bounding boxes to create two types of game action: tracking the size and position of moving objects, and determining if objects have collided.

HTML5 inline SVG DOM (Document Object Model) scripting will be used in this article to develop these SVG object detection techniques with JavaScript. With these techniques, you will be able to find moving objects and see if they collide with other objects. They can be used to develop different aspects of game play such as:

- Locating game objects like:
  - Friends or enemies
  - Treasure
  - Fuel sources
  - Tools
  - Food
  - Target or goal
- Avoiding collisions with characters or landscape
- Navigating around obstacles
- Finding correct pathways
- Chasing or escaping from characters
- Tracking moving objects

- Hitting a target
- Detecting collisions

These bounding box techniques will be demonstrated with an annotated stand-alone inline SVG HTML5 code example that shows how to use the SVG **getBBox** method to enable a smart bomb to chase a space ship through a game field. The code example will: explain how to create the space ship and the smart bomb, demonstrate the procedures for moving and tracking both objects, and show how to develop the techniques for determining when the bomb collides with the ship.

This article discusses the code in seven parts. The first six parts will explain key concepts that are necessary to develop with this technique. Part seven will provide debugging information on: how to use the SVG **Text** element to display variable values from your code while the game is running, and how you can make bounding boxes visible to help with debugging. The concepts that will be covered are:

1. Using HTML5 and JavaScript to set up a game shell.
2. Creating SVG objects and properties with DOM scripting.
3. Moving a game object using a game loop.
4. Processing key presses with an event handler.
5. Detecting collisions with bounding boxes.
6. Displaying your score.
7. Debugging SVG code.

Using HTML5 and JavaScript to Set Up a Game Shell

The first step necessary to create a program that uses this SVG collision detection technique is to set up the game shell by creating the standard HTML5 elements for head, body, and script. Be sure to add an ID to the body element because you'll need to reference it later.

Here's what a typical HTML5 game shell looks like:

```
<!DOCTYPE HTML>
<html>

    <head>
        <script type="text/javascript">
```

```
        ...

         </script>
    </head>

    <body id="myBody">
        <h1>
            Smart Bomb
        </h1>
    </body>

</html>
```

Next you need to define global variables and initialize a multi-dimensional JavaScript array for target tracking:

```
// Global variables
var xShip = 250;       // Ship x start position.
var yShip = 250;       // Ship y start position.
var rShip = 30;        // Ship radius.
var speedS = 5;        // Speed of ship.
var edge = rShip;      // How close to the edge?
var xBomb = 50;        // Bomb x start position.
var yBomb = 50;        // Bomb y start position.
var speedB = 20;       // Speed of bomb.
myDebug = 1;           // Debugging mode.
myMoves = 0;           // Number of moves for your ship.
bombMoves = 0;         // Number of moves for the bomb.
// Namespace for SVG.
svgNS = "http://www.w3.org/2000/svg";
```

After you have defined global variables to initialize the data for the space ship and smart bomb, add two event listeners and a timer for the game loop. The first listener will listen for keys on the keyboard and will call the **whatKey** function. The second listener will listen for the page load and call the **runFirst** function. A game loop is needed to allow code to run at regular intervals and uses the **setInterval** timer to call **myGameLoop** every second. The code looks like this:

```
// Add keyboard event listener to detect keydown presses.
window.addEventListener("keydown", whatKey, false);

// Add load event listener to run first function.
 window.addEventListener("load", runFirst, false);

// Add a timer to run the game loop every 1 second.
var myInterval = window.setInterval(myGameLoop, 1000);
```

The final component necessary to create a game shell is the addition of the empty **runFirst** function which will hold all the object creation code:

```
// This function will run when the document loads.
function runFirst() {

   ...

   }
```

Creating SVG Objects and Properties with SVG DOM Scripting

Once you have developed an HTML5 game shell, you can begin using SVG to create objects and properties that will make up the game components. The first step needed to create any SVG object is to set up the SVG parent container and define its boundaries. This container uses DOM scripting that calls directly into the SVG engine. By coding directly to the engine, it is possible to modify the object parameters using exact datatypes. This makes it easy to develop games with SVG that are fast and efficient.

The code in this section will be placed inside the empty **runFirst** function. Here is the code needed to create the SVG parent element container and a rectangle to show its boundaries:

```
// Create SVG parent element.
mySvg = document.createElementNS("http://www.w3.org/2000/svg", "svg");
mySvg.width.baseVal.valueAsString = "300px";
mySvg.height.baseVal.valueAsString = "300px";

// Create boundaries.
myBox = document.createElementNS("http://www.w3.org/2000/svg", "rect");
myBox.x.baseVal.value = 0;
myBox.y.baseVal.value = 0;
myBox.width.baseVal.value = 300;
myBox.height.baseVal.value = 300;
myBox.style.setProperty("stroke-width","1","");
myBox.style.setProperty("stroke","seagreen","");
myBox.style.setProperty("fill","none","");

// Append box to SVG parent element.
mySvg.appendChild(myBox);

// Append SVG parent to body of web page.
document.getElementById("myBody").appendChild(mySvg);
```

The next step is to create the game objects. In this code example, the objects will consist of a maroon space ship that you will fly and a limegreen smart bomb that is chasing you. The code for creating the space ship and smart bomb is developed in the same standard SVG DOM scripting object creation format. First you create an element using the SVG namespace, then you define the properties, and finally you attach the element to the SVG parent and the document. Here is what the code looks like for the ship and the bomb:

```
// Create ship.
myShip = document.createElementNS(svgNS, "circle");
myShip.cx.baseVal.value = xShip;
myShip.cy.baseVal.value = yShip;
myShip.r.baseVal.value = rShip;
myShip.style.setProperty("fill","maroon","");
mySvg.appendChild(myShip);

// Create bomb.
myBomb = document.createElementNS(svgNS, "circle");
myBomb.cx.baseVal.value = xBomb;
myBomb.cy.baseVal.value = yBomb;
myBomb.r.baseVal.value = 10;
myBomb.style.setProperty("fill","limegreen","");
mySvg.appendChild(myBomb);
```

Moving a Game Object using a Game Loop

Now that the objects and properties have been defined, the next step is to create a game loop that runs on a repeating cycle. Games often have a game loop that runs code at regular intervals so that events can take place without user input. In this game example, the loop is set up to move the smart bomb every second. The loop will perform the following tasks:

a. 1. Calculate a small random move for the smart bomb. This adds an element of surprise since you will never know exactly where the bomb will move next.

b. 2. Determine where the space ship is by using the SVG **getBBox** method. Every SVG object has a bounding box that contains the **x**, **y**, **width**, and **height** coordinates that contain the box. Even though an object may have a very complicated path that defines it, the bounding box will give you the basic location and size of the object. This can be used for both tracking and detecting collisions.

c. 3. Move the bomb toward the ship by using the ship's bounding box to determine which direction to move.

Here is the code that shows the game loop and the function it calls to move the bomb:

```
function myGameLoop() {

        // Debugging output. If myDebug == 0, hide text element.
        if (myDebug == 0) myText.setAttributeNS(null,"display","none");

        // Move the bomb.
        moveBomb();

        // Count bomb moves.
        bombMoves++;
        }

function moveBomb() {

        // Calculate random move.
        xMove = Math.floor(Math.random()*speedB) - (speedB/2) + 1;
        yMove = Math.floor(Math.random()*speedB) - (speedB/2) + 1;

        // Add move to current bomb position.
        xBomb = xBomb + xMove;
        yBomb = yBomb + yMove;

        // Find the ship.
        bb = myShip.getBBox();
        targetX = bb.x + rShip;
        targetY = bb.y + rShip;

        // Calculate move toward ship.
        if (targetX > xBomb) xBomb = xBomb + 10;
        if (targetY > yBomb) yBomb = yBomb + 10;
        if (targetX < xBomb) xBomb = xBomb - 10;
        if (targetY < yBomb) yBomb = yBomb - 10;

        // Move bomb.
        myBomb.cx.baseVal.value = xBomb;
        myBomb.cy.baseVal.value = yBomb;
```

```
}
```

Processing key presses with an event handler

In order to move the ship, you need to add a event handler that will respond to key presses. The handler function has two purposes. The first is to move the ship whenever you press a key. The code processes the key presses and moves the ship left or right, up or down. It also makes sure that the ship can't run outside the game field.

This is how the code looks:

```
// Handle keydown event.
function whatKey(evt){

        // Check for arrow key codes.
        switch (evt.keyCode) {

        // Check key presses.
        // Left arrow keycode is 37.
        case 37:
        if (xShip > edge) {
                xShip = xShip - speedS;
                }
        break;

        // Right arrow keycode is 39.
        case 39:
        if (xShip < (300 - edge)) {
                xShip = xShip + speedS;
                }
        break;

        // Down arrow keycode is 40.
        case 40:
        if (yShip < 300 - edge) {
                yShip = yShip + speedS;
                }
        break;

        // Up arrow keycode is 38.
        case 38:
        if (yShip > edge) {
                yShip = yShip - speedS;
                }
        break;
        }

        // Set new center coordinates ship.
        myShip.cx.baseVal.value = xShip;
        myShip.cy.baseVal.value = yShip;

        // Count ship moves.
        myMoves++;
        }
```

Detecting Collisions with SVG Bounding Boxes

In the "Moving a Game Object using a Game Loop" section, the SVG **getBBox** method was used to calculate the location of the ship the smart bomb is chasing. The **getBBox**

can also be used to detect collisions. It does this by using **getBBox** every time the bomb moves and checking to see if the bomb's own bounding box overlaps the bounding box of the ship. If the two bounding boxes overlap enough, a flag will be set to indicate that the game is over. The following code is inserted in the **moveBomb** function after the code that moves the bomb:

```
// Do the bounding boxes overlap?
myFlag = 0;
bbb = myBomb.getBBox();
xClose = Math.abs(bb.x - bbb.x + 20);
yClose = Math.abs(bb.y - bbb.y + 20);
if ((xClose < 21) && (yClose < 21)) myFlag = 1;
```

Displaying the Score and Ending the Game

SVG has a **Text** object that provides a way to display text anywhere on the screen. This can be used to display the final score when the game ends. You can do that with the following steps:

a. 1. Create a **Text** element with a text node.

b. 2. Use the DOM **textContent** property to put text into the text node.

In order to create an SVG **Text** element, put this code at the end of the **runFirst** function:

```
// Create text element for messages.
myText = document.createElementNS(svgNS,"text");
myText.setAttributeNS(null,"x",10);
myText.setAttributeNS(null,"y",270);

// Create node for actual text.
myTextNode = document.createTextNode("Use arrow keys.");
myText.appendChild(myTextNode);
mySvg.appendChild(myText);
```

SVG **Text** elements require a **TextNode** sub-element that contains the text. Initially it says "Use arrow keys.", but you can change it later by using the **textContent** property at the end of the **moveBomb** function. This code will only run if the **myFlag** flag is set, indicating that a collision happened. The code looks like this:

```
if (myFlag == 1) {
        // Display score.              t1 = "You moved " + myMoves + " times in ";  t2 =
bombMoves + " ticks."         myTextNode.textContent = t1 + t2;
```

At this point, you need to add code to stop the game loop and event handler because the game loop is running and the keypress event handler is still looking for key presses. In order to do this, add the following code after the code that displays the score:

```
        // Stop game loop.            clearInterval(myInterval);
        // Remove event listener.             window.removeEventListener("keydown",
```

```
whatKey, false);              }
```

clearInterval stops the game loop by using the **myInterval** object that identifies the interval taking place and **removeEventListener** removes the listener that was listening for the **keydown** event.

Debugging SVG Code

When you are developing and debugging your code, it is often helpful to see the values of variables when the program is running. The SVG **Text** object makes it easy to view the internal workings of your code. You can display the value of a variable by creating an SVG **Text** object and then use **textContent** to copy the value of the variable to the text node. This SVG **Text** object technique was used in this game example to display the final score, but you can also use the **Text** object technique to see what's going on under the hood while you are coding. If you want to hide the **Text** object, set the **display** attribute of the object to "none".

Bounding boxes can be used as a debugging tool. They make it possible to see the exact outline of an object's bounding box. This can be valuable because you can see how close an object's bounding box is to another object. You can make a bounding box visible by drawing an SVG **rect** rectangle defined by the object's bounding box. The following code demonstrates how this works. It displays a bounding box around the space ship and shows the exact area that the bounding box defines:

```
        // Optional: draw bounding box if you want to see it.
        boundShip = document.createElementNS(svgNS, "rect");        boundShip.x.baseVal.value =
bb.x;   boundShip.y.baseVal.value = bb.y;     boundShip.width.baseVal.value = rShip * 2;
        boundShip.height.baseVal.value = rShip * 2;
        boundShip.style.setProperty("stroke-width","3","");
        boundShip.style.setProperty("stroke","magenta","");
        boundShip.style.setProperty("fill","none","");       mySvg.appendChild(boundShip);
```

Note that bounding boxes are always rectangular, even if the underlying shape is not a box. The bounding box will always be the smallest box the shape can fit in and if the shape changes size or moves, the bounding box will change or move with it.

Complete SVG Code Example

The following listing is the complete SVG bounding box code example. It includes all of the code used in this article. In order to run this example, copy this code to a text file and save it with the .html file extension. It can be run by opening it in any of the major browsers.

```
<!DOCTYPE HTML>
<html>

    <head>
        <script type="text/javascript">
```

```
// Global variables
var xShip = 250;       // Ship x start position.
var yShip = 250;       // Ship y start position.
var rShip = 30;               // Ship radius.
var speedS = 5;               // Speed of ship.
var edge = rShip;      // How close to the edge?
var xBomb = 50;               // Bomb x start position.
var yBomb = 50;               // Bomb y start position.
var speedB = 20;       // Speed of bomb.
myDebug = 1;           // Debugging mode.
myMoves = 0;           // Number of moves for your ship.
bombMoves = 0;         // Number of moves for the bomb.
// Namespace for SVG.
svgNS = "http://www.w3.org/2000/svg";

// Add load event listener to run first function.
window.addEventListener("load", runFirst, false);

// Add keyboard event listener to detect keydown presses.
window.addEventListener("keydown", whatKey, false);

// Add a timer to run the game loop every 1 second.
var myInterval = window.setInterval(myGameLoop, 1000);

// This function will run when the document loads.
function runFirst() {

        // Create SVG parent element.
        mySvg = document.createElementNS(svgNS, "svg");
        mySvg.width.baseVal.valueAsString = "300px";
        mySvg.height.baseVal.valueAsString = "300px";

        // Create game area boundaries.
        myBox = document.createElementNS(svgNS, "rect");
        myBox.x.baseVal.value = 0;
        myBox.y.baseVal.value = 0;
        myBox.width.baseVal.value = 300;
        myBox.height.baseVal.value = 300;
        myBox.style.setProperty("stroke-width","1","");
        myBox.style.setProperty("stroke","seagreen","");
        myBox.style.setProperty("fill","none","");

        // Append box to SVG parent element.
        mySvg.appendChild(myBox);

        // Append SVG parent to body of web page.
        document.getElementById("myBody").appendChild(mySvg);

        // Create ship.
        myShip = document.createElementNS(svgNS, "circle");
        myShip.cx.baseVal.value = xShip;
        myShip.cy.baseVal.value = yShip;
        myShip.r.baseVal.value = rShip;
        myShip.style.setProperty("fill","maroon","");
        mySvg.appendChild(myShip);

        // Create Bomb.
        myBomb = document.createElementNS(svgNS, "circle");
        myBomb.cx.baseVal.value = xBomb;
        myBomb.cy.baseVal.value = yBomb;
        myBomb.r.baseVal.value = 10;
        myBomb.style.setProperty("fill","limegreen","");
        mySvg.appendChild(myBomb);

        // Create text element for messages.
        myText = document.createElementNS(svgNS,"text");
        myText.setAttributeNS(null,"x",10);
        myText.setAttributeNS(null,"y",270);

        // Create node for actual text.
        myTextNode = document.createTextNode("Use arrow keys.");
```

```
                        myText.appendChild(myTextNode);
                        mySvg.appendChild(myText);
                        }

// Handle keydown event.
function whatKey(evt){

        // Check for arrow key codes.
        switch (evt.keyCode) {

        // Check key presses.
        // Left arrow keycode is 37.
        case 37:
        if (xShip > edge) {
                xShip = xShip - speedS;
                }
        break;

        // Right arrow keycode is 39.
        case 39:
        if (xShip < (300 - edge)) {
                xShip = xShip + speedS;
                }
        break;

        // Down arrow keycode is 40.
        case 40:
        if (yShip < 300 - edge) {
                yShip = yShip + speedS;
                }
        break;

        // Up arrow keycode is 38.
        case 38:
        if (yShip > edge) {
                yShip = yShip - speedS;
                }
        break;
        }

        // Set new center coordinates ship.
        myShip.cx.baseVal.value = xShip;
        myShip.cy.baseVal.value = yShip;

        // Count ship moves.
        myMoves++;
        }

function myGameLoop() {

        // Debugging output. If myDebug == 0, hide text element.
        if (myDebug == 0) myText.setAttributeNS(null,"display","none");

        // Move the bomb.
        moveBomb();

        // Count bomb moves.
        bombMoves++;
        }

function moveBomb() {

        // Calculate random move.
        xMove = Math.floor(Math.random()*speedB) - (speedB/2) + 1;
        yMove = Math.floor(Math.random()*speedB) - (speedB/2) + 1;

        // Add move to current bomb position.
        xBomb = xBomb + xMove;
        yBomb = yBomb + yMove;

        // Find the ship.
```

```
        bb = myShip.getBBox();
        targetX = bb.x + rShip;
        targetY = bb.y + rShip;

        // Calculate move toward ship.
        if (targetX > xBomb) xBomb = xBomb + 10;
        if (targetY > yBomb) yBomb = yBomb + 10;
        if (targetX < xBomb) xBomb = xBomb - 10;
        if (targetY < yBomb) yBomb = yBomb - 10;

        // Move bomb.
        myBomb.cx.baseVal.value = xBomb;
        myBomb.cy.baseVal.value = yBomb;

        // Do the bounding boxes overlap?
        myFlag = 0;
        bbb = myBomb.getBBox();
        xClose = Math.abs(bb.x - bbb.x + 20);
        yClose = Math.abs(bb.y - bbb.y + 20);
        if ((xClose < 21) && (yClose < 21)) myFlag = 1;

        // If they overlap, the game is over.
        if (myFlag == 1) {

                /* Optional: draw bounding box if you want to see it.
                boundShip = document.createElementNS(svgNS, "rect");
                boundShip.x.baseVal.value = bb.x;
                boundShip.y.baseVal.value = bb.y;
                boundShip.width.baseVal.value = rShip * 2;
                boundShip.height.baseVal.value = rShip * 2;
                boundShip.style.setProperty("stroke-width","3","");
                boundShip.style.setProperty("stroke","magenta","");
                boundShip.style.setProperty("fill","none","");
                mySvg.appendChild(boundShip);
                */

                // Display score.
                t1 = "You moved " + myMoves + " times in ";
                t2 = bombMoves + " ticks."
                myTextNode.textContent = t1 + t2;

                // Stop game loop.
                clearInterval(myInterval);
                // Remove event listener.
                window.removeEventListener("keydown", whatKey, false);
                }
        }

        </script>
    </head>

    <body id="myBody">
        <h1>
            SVG Smart Bomb
        </h1>
    </body>

</html>
```