Build a Web Game in an Hour with Visual Studio and ASP.NET

Michael Oneppo(https://www.sitepoint.com/author/moneppo/)

August 27, 2015 +7

This article is part of a web development series from Microsoft. Thank you for supporting the partners who make SitePoint possible.

This article discusses:

- Basic game development philosophy
- Using Web technologies for game development
- Adding game controls and AI

Technologies discussed:

Visual Studio 2013 Pro, Visual Studio 2013 Community, ASP.NET
Code download (http://download.microsoft.com/download/9/0/9/909383C3-E5F2-4EF8-BDE5-7D8B6C9AD164/Code_OneppoGame0315.zip) (.zip)

You don't need an entirely new skill set to develop games. In fact, your current Web development skills in HTML, JavaScript, CSS and so on are just fine for a wide range of games. When you build a game with Web technologies, it will run on pretty much any device with a browser.

To prove this, I'll demonstrate building a game from scratch using Web technologies and just two external libraries, and I'll do it in less than one hour. I'll cover a variety of game development topics, from basic design and layout, controls and sprites, to artificial intelligence (AI) for a simple opponent. I'm even going to develop the game so it works on PCs, tablets and smartphones. If you

have some experience with programming as a Web developer or another development domain, but no experience writing games, this article will get you started. If you give me one hour, I promise to show you the ropes.

Get Up and Running

I'll do all development in Visual Studio (http://www.visualstudio.com/?WT.mc_id=13419-DEV-sitepoint-article43), which will allow fast execution of the Web app as I make changes. Be sure to have the latest version of Visual Studio so you can follow along. I used Visual Studio 2013 Pro, but updated the code with Visual Studio 2013 Community. Also if you have a Mac or Linux, Visual Studio Code (http://code.visualstudio.com/?WT.mc_id=13419-DEV-sitepoint-article43) is available cross-platform nowadays.

This app will require no server code, so I start by creating a new, empty Web page project in Visual Studio. I'll use the empty C# template for a Web site by selecting the Visual C# option after selecting File | New | ASP.NET Empty Web Site.

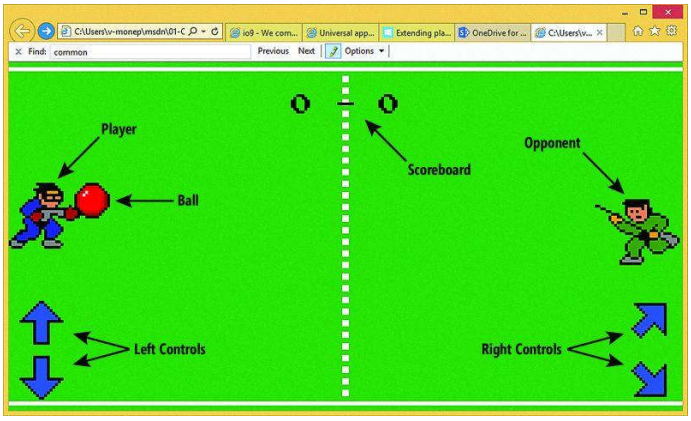
The index HTML file requires just three resources: jQuery, a main style sheet and a main JavaScript file. I add an empty CSS file to the project called style.css and an empty JavaScript file called ping.js to avoid errors when loading the page:

```
<!DOCTYPE html>
<html>
<head>
  <script src="http://ajax.aspnetcdn.com/ajax/jQuery/jquery-2.1
  <script src="ping.js"></script>
  <link rel="stylesheet" href="style.css"></script>
</head>
<body>
</body>
</html>
```

Also don't forget to test this app (or any other) for that matter across browsers & devices. While the code I wrote is interoperable with modern browsers like Chrome, Firefox, and Microsoft Edge (http://blogs.windows.com/msedgedev/2015/05/06/a-break-from-the-past-part-2-saying-goodbye-to-activex-vbscript-attachevent/?WT.mc_id=13419-DEV-sitepoint-article43), it's always a best practice to double-check. Now you can do that with free virtual machines (http://dev.modern.ie/tools/vms/?utm_source=SitePoint&utm_medium=article43&utm_campaign=SitePoint) and other tools like http://www.browserstack.com (http://www.browserstack.com).

Basic Design

The game I'm building is a variant of Pong that I call Ping. Ping has essentially the same rules as Pong, except that either player grabs the ball when it comes to them and can then fire the ball back either directly or at an angle up or down. It's often best to draw how you would like the game to look before you build it. For this game, the overall layout I want to see is shown below.



Once I've developed the game design layout, it's just a matter of adding each element to HTML to build the game. One thing to note, though, is I'll group the scoreboard and controls to ensure they sit together. So one by one, you can see I've added the elements, as shown below:

```
<div id="arena">
  <div id="score">
    <h1>
      <span id="playerScore">0</span>
     <span id="opponentScore">0</span>
   </h1>
 </div>
 <div id="player"></div>
 <div id="opponent"></div>
 <div id="ball"></div>
 <div id="controls-left">
   <div id="up"></div>
   <div id="down"></div>
  </div>
  <div id="controls-right">
    <div id="left"></div>
    <div id="right"></div>
  </div>
</div>
```

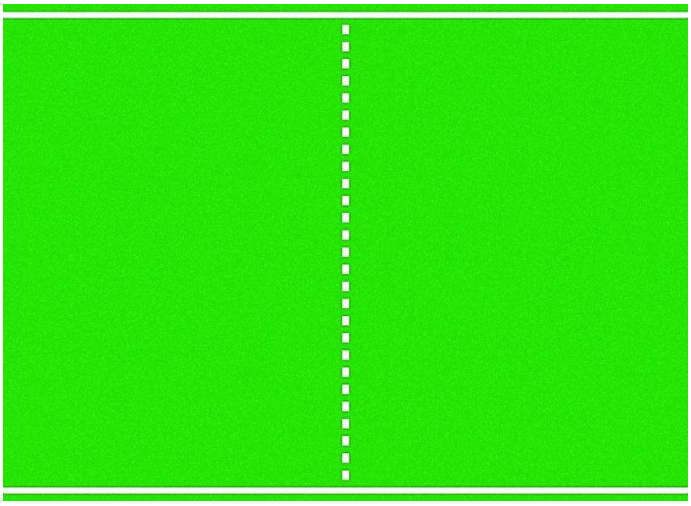
Play with Style

If you were to load this page, you wouldn't see anything because there's no style applied. I've already set up a link to a main.css file in my HTML, so I'll place all my CSS in a new file with that name. The first thing I'll do is position everything on the screen. The body of the page needs to take up the whole screen, so I'll set that up first:

```
body {
  margin: 0px;
  height: 100%;
}
```

Second, I need to have the arena fill the whole screen with the arena background image (see image below) applied:

```
#arena {
  background-image: url(arena.png);
  background-size: 100% 100%;
  margin: 0px;
  width: 100%;
  height: 100%;
  overflow: hidden;
}
```



Next, I'll position the scoreboard. I want this to appear top and center, over the other elements. The command position: absolute lets me place it wherever I want and left: 50% places it halfway across the top of the window, but starting at the leftmost side of the scoreboard element. To ensure it's perfectly centered, I use the transform property and the z-index property ensures it's always at the top:

```
#score {
  position: absolute;
  z-index: 1000;
  left: 50%;
  top: 5%;
  transform: translate(-50%, 0%);
}
```

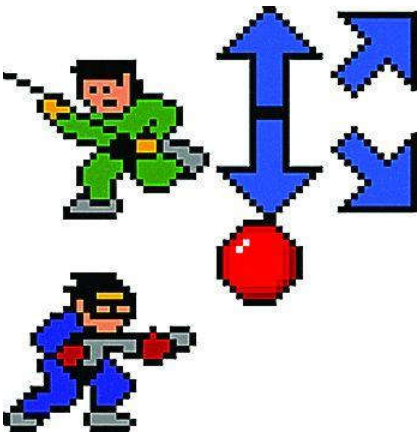
I also want the text font to be retro-themed. Most modern browsers let me include my own fonts. I found the appropriate Press Start 2P font from codeman38 (zone38.net). To add the font to the scoreboard, I have to create a new font face:

```
@font-face {
  font-family: 'PressStart2P';
  src: url('PressStart2P.woff');
}
```

Now, the scores are in an h1 tag, so I can set the font for all h1 tags. Just in case the font is missing, I'll provide a few backup options:

```
h1 {
  font-family: 'PressStart2P', 'Georgia', serif;
}
```

For the other elements, I'll use a sprite sheet of images. A sprite sheet contains all the images I need for the game in one file (see image below).



Any element that has an image on this sheet will have a sprite class assigned. Then, for each element, I'll use `background-position` to define what part of the sprite sheet I want to show:

```
.sprite {
  background-image: url("sprites.png");
  width: 128px;
  height: 128px;
}
```

Next, I'll add the `sprite` class to all elements that will use the sprite sheet. I'll have to briefly jump back to HTML to do this:

```
<div id="player" class="sprite"></div>
<div id="opponent" class="sprite"></div>
<div id="ball" class="sprite"></div>
<div id="controls-left">
  <div id="up" class="sprite"></div>
  <div id="down" class="sprite"></div>
</div>
<div id="controls-right">
  <div id="left" class="sprite"></div>
  <div id="right" class="sprite"></div>
</div>
```

Now I need to indicate the positions of each sprite on the sheet for each element. Again, I'll do this using background-position:

```
#player {
  position: absolute;
  background-position: 0px 128px;
}
#opponent {
  position: absolute;
  background-position: 0px 0px;
}
#ball {
  position: absolute;
  background-position: 128px 128px;
}
#right {
  background-position: 64px 192px;
}
#left {
  background-position: 64px 0px;
}
#down {
  background-position: 128px 192px;
}
#up {
  background-position: 128px 0px;
}
```

The position: absolute property on the player, opponent and ball will let me move them around using JavaScript. If you look at the page now, you'll see the controls and the ball have unnecessary pieces attached to them. This is because the sprite sizes are smaller than the default 128 pixels, so I'll adjust these to the right size. There's only one ball, so I'll set its size directly:

```
#ball {
  position: absolute;
  width: 64px;
  height: 64px;
  background-position: 128px 128px;
}
```

There are four control elements (buttons the user can press to move the player about), so it behooves me to make a special class for them. I'll also add a margin so they have a little space around them:

```
.control {
  margin: 16px;
  width: 64px;
  height: 64px;
}
```

After adding this class, the game has much better looking controls:

```
<div id="controls-left">
  <div id="up" class="sprite control"></div>
  <div id="down" class="sprite control"></div>
</div>
<div id="controls-right">
  <div id="left" class="sprite control"></div>
  <div id="right" class="sprite control"></div>
</div>
```

The last thing I need to do is position the controls so they're by the user's thumbs when the page is running on a mobile device. I'll stick them to the bottom corners:

```
#controls-left {
  position: absolute;
  left: 0; bottom: 0;
}
#controls-right {
  position: absolute;
  right: 0; bottom: 0;
}
```

One nice thing about this design is everything is set with relative positions. This means the screen can be a number of different sizes while still making the game look good.

Follow the Bouncing Ball

Now I'll make the ball move around. For the JavaScript code, I've referenced a file called ping.js in HTML, just as I did with the CSS. I'll add this code to a new file with that name. I'm going to make objects for the ball and each of the players, but I'll use the factory pattern for the objects.

This is a simple concept. The Ball function creates a new ball when you call it. There's no need to use the new keyword. This pattern reduces some of the confusion around the this variable by clarifying the available object properties. And because I only have an hour to make this game, I need to minimize any confusing concepts.

The structure of this pattern, as I make the simple Ball class:

```
var Ball = function( {
  // List of variables only the object can see (private variabl
  var velocity = [0,0];
  var position = [0,0];
  var element = $('#ball');
  var paused = false;
  // Method that moves the ball based on its velocity. This met
  // internally and will not be made accessible outside of the
  function move(t) {
  }
  // Update the state of the ball, which for now just checks
  // if the play is paused and moves the ball if it is not.
  // This function will be provided as a method on the object.
  function update(t) {
    // First the motion of the ball is handled
    if(!paused) {
      move(t);
    }
  }
  // Pause the ball motion.
  function pause() {
    paused = true;
  }
  // Start the ball motion.
  function start() {
    paused = false;
  }
  // Now explicitly set what consumers of the Ball object can u
  // Right now this will just be the ability to update the stat
  // and start and stop the motion of the ball.
  return {
    update:       update,
    pause:        pause,
    start:        start
}
```

To create a new ball, I simply call this function I've defined:

```
var ball = Ball();
```

Now I want to make the ball move and bounce around the screen. First, I need to call the update function at some interval to create an animation of the ball. Modern browsers provide a function meant for this purpose called requestAnimationFrame. This takes a function as an argument, and will call that passed-in function the next time it runs its animation cycle. This lets the ball move around in smooth steps when the browser is ready for an update. When it calls the passed-in function, it will give it the time in seconds since the page was loaded. This is critical for ensuring animations are consistent over time. In the game, the use of `requestAnimationFrame` appears as follows:

```
var lastUpdate = 0;
var ball = Ball();

function update(time) {
  var t = time - lastUpdate;
  lastUpdate = time;
  ball.update(t);
  requestAnimationFrame(update);
}

requestAnimationFrame(update);
```

Note that `requestAnimationFrame` is called again in the function, as the ball has finished updating. This ensures continuous animation.

While this code will work, there may be an issue where the script starts running before the page is fully loaded. To avoid this, I'll kick off the code when the page is loaded, using jQuery:

```
var ball;
var lastUpdate;
$(document).ready(function() {
  lastUpdate = 0;
  ball = Ball();
  requestAnimationFrame(update);
});
```

Because I know the speed of the ball (velocity) and the time since its last update, I can do some simple physics to move the ball forward:

```
var position = [300, 300];
var velocity = [-1, -1];
var move = function(t) {
  position[0] += velocity[0] \* t;
  position[1] += velocity[1] \* t;
  element.css('left', position[0] + 'px');
  element.css('top', position[1] + 'px');
}
```

Try running the code and you'll see the ball move at an angle and off the screen. This is fun for a second, but once the ball goes off the edge of the screen, the fun stops. So the next step is to make the ball bounce off the edges of the screen, as implemented in Figure 7. Add this code and running the app will show a continuously bouncing ball.

A Moveable Player

Now it's time to make the Player objects. The first step in fleshing out the player class will be to make the move function change the position of the player. The side variable will indicate which side of the court the player will reside, which will dictate how to position the player horizontally. The y value, passed into the move function, will be how much up or down the player will move:

```
var Player = function (elementName, side) {
  var position = [0,0];
  var element = $('#'+elementName);
  var move = function(y) {
  }
  return {
    move: move,
    getSide:      function()  { return side; },
    getPosition:  function()  { return position; }
  }
}
```

We can then lay out player movement, stopping the motion if the player sprite reaches the top or bottom of the window.

```
var move = function(y) {
  // Adjust the player's position.
  position[1] += y;
  // If the player is off the edge of the screen, move it back.
  if (position[1] <= 0)  {
    position[1] = 0;
  }
  // The height of the player is 128 pixels, so stop it before
  // part of the player extends off the screen.
  if (position[1] >= innerHeight - 128) {
    position[1] = innerHeight - 128;
  }
  // If the player is meant to stick to the right side, set the
  // to the right edge of the screen.
  if (side == 'right') {
    position[0] = innerWidth - 128;
  }
  // Finally, update the player's position on the page.
  element.css('left', position[0] + 'px');
  element.css('top', position[1] + 'px');
}
```

I can now create two players and have them move to their appropriate side of the screen:

```
player = Player('player', 'left');
player.move(0);
opponent = Player('opponent', 'right');
opponent.move(0);
```

Keyboard Input

So in theory you can move the player, but it won't move without instruction. Add some controls to the player on the left. You want two ways to control that player: using the keyboard (on PCs) and tapping the controls (on tablets and phones).

To ensure consistency between touch inputs and mouse inputs on various platforms, I'll use the great unifying framework Hand.js (handjs.codeplex.com). First, I'll add the script to HTML in the head section:

```
<script src="hand.minified-1.3.8.js"></script>
```

I'll then use `Hand.js` and jQuery to control the player when you press keyboard keys A and Z, or when you tap the controls.

```
var distance = 24;  // The amount to move the player each step.
$(document).ready(function() {
  lastUpdate = 0;
  player = Player('player', 'left');
  player.move(0);
  opponent = Player('opponent', 'right');
  opponent.move(0);
  ball = Ball();
  // pointerdown is the universal event for all types of pointe
  // a mouse, a stylus and so on.
  $('#up')    .bind("pointerdown", function() {player.move(-dis
  $('#down')  .bind("pointerdown", function() {player.move(dist
  requestAnimationFrame(update);
});
$(document).keydown(function(event) {
  var event = event || window.event;
  // This code converts the keyCode (a number) from the event t
  // letter to make the switch statement easier to read.
  switch(String.fromCharCode(event.keyCode).toUpperCase()) {
    case 'A':
      player.move(-distance);
      break;
    case 'Z':
      player.move(distance);
      break;
  }
  return false;
});
```

Catch the Ball

As the ball bounces around, I want to let players catch it. When it's caught, the ball has an owner, and it follows the motion of that owner. I'll add functionality to the ball's move method, allowing for an owner, which the ball will then follow:

```
var move = function(t) {
  // If there is an owner, move the ball to match the owner's p
  if (owner !== undefined) {
    var ownerPosition = owner.getPosition();
    position[1] = ownerPosition[1] + 64;
    if (owner.getSide() == 'left') {
      position[0] = ownerPosition[0] + 64;
    } else {
      position[0] = ownerPosition[0];
    }
  // Otherwise, move the ball using physics. Note the horizonta
  // has been removed -- ball should pass by a player if it
  // isn't caught.
  } else {
    // If the ball hits the top or bottom, reverse the vertical
    if (position[1] - 32 <= 0 || position[1] + 32 >= innerHeigh
      velocity[1] = -velocity[1];
    }
    position[0] += velocity[0] \* t;
    position[1] += velocity[1] \* t;
  }
  element.css('left', (position[0] - 32) + 'px');
  element.css('top',  (position[1] - 32) + 'px');
}
```

Currently, there's no way to get the position of a Player object, so I'll add the `getPosition` and `getSide` accessors to the Player object:

```
return {
  move: move,
  getSide:      function()  { return side; },
  getPosition:  function()  { return position; }
}
```

Now, if the ball has an owner, it will follow that owner around. But how do I determine the owner? Somebody has to catch the ball. Let's determine when one of the player sprites touches the ball. When that happens, I'll set the owner of the ball to that player.

```
var update = function(t) {
// First the motion of the ball is handled.
if(!paused) {
    move(t);
}
// The ball is under control of a player, no need to update.
if (owner !== undefined) {
    return;
}
// First, check if the ball is about to be grabbed by the playe
var playerPosition = player.getPosition();
  if (position[0] <= 128 &&
      position[1] >= playerPosition[1] &&
      position[1] <= playerPosition[1] + 128) {
    console.log("Grabbed by player!");
    owner = player;
}
// Then the opponent...
var opponentPosition = opponent.getPosition();
  if (position[0] >= innerWidth - 128 &&
      position[1] >= opponentPosition[1] &&
      position[1] <= opponentPosition[1] + 128) {
    console.log("Grabbed by opponent!");
    owner = opponent;
}
```

If you try playing the game now, you'll find the ball bounces off the top of the screen, and you can move the player to catch it. Now, how do you throw it? That's what the right-hand controls are for—aiming the ball. Let's add a "fire" function to the player, as well as an aim property.

```
var aim = 0;
var fire = function() {
  // Safety check: if the ball doesn't have an owner, don't not
  if (ball.getOwner() !== this) {
    return;
  }
  var v = [0,0];
  // Depending on the side the player is on, different directio
  // The ball should move at the same speed, regardless of dire
  // with some math you can determine that moving .707 pixels o
  // x and y directions is the same speed as moving one pixel i
  if (side == 'left') {
    switch(aim) {
    case -1:
      v = [.707, -.707];
      break;
    case 0:
      v = [1,0];
      break;
    case 1:
      v = [.707, .707];
    }
  } else {
    switch(aim) {
    case -1:
      v = [-.707, -.707];
      break;
    case 0:
      v = [-1,0];
      break;
    case 1:
      v = [-.707, .707];
    }
  }
  ball.setVelocity(v);
  // Release control of the ball.
  ball.setOwner(undefined);
}
// The rest of the Ball definition code goes here...
return {
  move: move,
  fire: fire,
  getSide:      function()  { return side; },
  setAim:       function(a) { aim = a; },
  getPosition:  function()  { return position; },
}
```

We can then augment the keyboard function to set the player's aim and fire functions. Aiming will work slightly differently. When the aiming key is released, the aim will return to straightforward.

```
$(document).keydown(function(event) {
  var event = event || window.event;
  switch(String.fromCharCode(event.keyCode).toUpperCase()) {
    case 'A':
      player.move(-distance);
      break;
    case 'Z':
      player.move(distance);
      break;
    case 'K':
      player.setAim(-1);
      break;
    case 'M':
      player.setAim(1);
      break;
    case ' ':
      player.fire();
      break;
  }
  return false;
});
$(document).keyup(function(event) {
  var event = event || window.event;
  switch(String.fromCharCode(event.keyCode).toUpperCase()) {
    case 'K':
    case 'M':
      player.setAim(0);
      break;
  }
  return false;
});
```

The final addition will be touch support on all controls. I'll make the controls on the right change the aim of the player. I'll also make it so touching anywhere on the screen fires the ball:

```
$('#left')  .bind("pointerdown", function() {player.setAim(-1);
$('#right') .bind("pointerdown", function() {player.setAim(1);}
$('#left')  .bind("pointerup",   function() {player.setAim(0);}
$('#right') .bind("pointerup",   function() {player.setAim(0);}
$('body')   .bind("pointerdown", function() {player.fire();});
```

Keep Score

When the ball passes a player, I want to change the score and give the ball to that player. I'll use custom events so I can separate scoring from any of the existing objects. The update function is getting long, so I'll add a new private function called checkScored:

```
function checkScored() {
  if (position[0] <= 0) {
    pause();
    $(document).trigger('ping:opponentScored');
  }
  if (position[0] >= innerWidth) {
    pause();
    $(document).trigger('ping:playerScored');
  }
}
```

The code below reacts to those events to update the score and hand over the ball. Add this code to the bottom of the JavaScript document.

```
$(document).on('ping:playerScored', function(e) {
  console.log('player scored!');
  score[0]++;
  $('#playerScore').text(score[0]);
  ball.setOwner(opponent);
  ball.start();
});
$(document).on('ping:opponentScored', function(e) {
  console.log('opponent scored!');
  score[1]++;
  $('#opponentScore').text(score[1]);
  ball.setOwner(player);
  ball.start();
});
```

Now when the ball makes it past your opponent (which isn't that difficult, as the opponent doesn't move) your score will go up, and the ball will be handed to the opponent. However, the opponent will just hold onto the ball.

Get Smart

You almost have a game. If only you had someone with whom to play. As a last step, I'll show how to control the opponent with simple AI. The opponent will try to stay parallel with the ball as it moves about. If the opponent catches the ball, it will move randomly and fire the ball in a random direction. To make the AI feel a little more human, I'll add delays in everything done. This isn't highly intelligent AI, mind you, but it will be something to play the game against.

When designing this kind of system, it's good to think in states. The opponent AI has three possible states: following, aiming/shooting and waiting. I'll be the state between following actions to add a more human element. Start with just that for the AI object:

```
function AI(playerToControl) {
  var ctl = playerToControl;
  var State = {
    WAITING: 0,
    FOLLOWING: 1,
    AIMING: 2
  }
  var currentState = State.FOLLOWING;
}
```

Depending on the state of the AI, I'll want it to do a different action. Just like the ball, I'll make an update function I can call in `requestAnimationFrame` to have the AI act according to its state:

```
function update() {
  switch (currentState) {
    case State.FOLLOWING:
      // Do something to follow the ball.
      break;
    case State.WAITING:
      // Do something to wait.
      break;
    case State.AIMING:
      // Do something to aim.
      break;
  }
}
```

The `FOLLOWING` state is straightforward. The opponent moves in the vertical direction of the ball, and the AI transitions to the `WAITING` state to inject some slowed reaction time. The code below shows these two states:

```
function moveTowardsBall() {
  // Move the same distance the player would move, to make it f
  if(ball.getPosition()[1] >= ctl.getPosition()[1] + 64) {
    ctl.move(distance);
  } else {
    ctl.move(-distance);
  }
}
function update() {
  switch (currentState) {
    case State.FOLLOWING:
      moveTowardsBall();
      currentState = State.WAITING;
    case State.WAITING:
      setTimeout(function() {
        currentState = State.FOLLOWING;
      }, 400);
      break;
    }
  }
}
```

The AI alternates between having to follow the ball and wait a split second. Now

add the code to the game-wide update function:

```
function update(time) {
  var t = time - lastUpdate;
  lastUpdate = time;
  ball.update(t);
  ai.update();
  requestAnimationFrame(update);
}
```

When you run the game, you'll see the opponent follow the ball's movements—not a bad AI in less than 30 lines of code. Of course, if the opponent catches the ball, it won't do anything. So for the last trick of the hour, it's time to handle the actions for the `AIMING` state.

I want the AI to move randomly a few times and then fire the ball in a random direction. Let's add a private function that does just that. Adding the `aimAndFire` function to the `AIMING` case statement makes a fully functional AI against which to play.

```
function repeat(cb, cbFinal, interval, count) {
  var timeout = function() {
    repeat(cb, cbFinal, interval, count-1);
  }
  if (count <= 0) {
    cbFinal();
  } else {
    cb();
    setTimeout(function() {
      repeat(cb, cbFinal, interval, count-1);
    }, interval);
  }
}

function aimAndFire() {

  // Repeat the motion action 5 to 10 times.

  var numRepeats = Math.floor(5 + Math.random() \* 5);
  function randomMove() {
    if (Math.random() > .5) {
      ctl.move(-distance);
    } else {
      ctl.move(distance);
    }
  }

  function randomAimAndFire() {

    var d = Math.floor( Math.random() \* 3 - 1 );
    opponent.setAim(d);
    opponent.fire();

    // Finally, set the state to FOLLOWING.

    currentState = State.FOLLOWING;
  }

  repeat(randomMove, randomAimAndFire, 250, numRepeats);

}
```

Wrapping Up

By now, you have a full-fledged Web game that works on PCs, smartphones and tablets. There are many possible improvements to this game. It will look a little awkward in portrait mode on a smartphone, for example, so you need to make sure you're holding the phone in landscape for it to work properly. This is just a small demonstration of the possibilities for game development for the Web and beyond.

Thanks to technical expert Mohamed Ameen Ibrahim for reviewing this article.

More hands-on with JavaScript

This article is part of the web development series from Microsoft tech evangelists on practical JavaScript learning, open source projects, and interoperability best practices including Microsoft Edge (http://blogs.windows.com/msedgedev/2015/05/06/a-break-from-the-past-part-2-saying-goodbye-to-activex-vbscript-attachevent/?WT.mc_id=13419-DEV-sitepoint-article43) browser and the new EdgeHTML rendering engine (http://blogs.windows.com/msedgedev/2015/02/26/a-break-from-the-past-the-birth-of-microsofts-new-web-rendering-engine/?WT.mc_id=13419-DEV-sitepoint-article43).

We encourage you to test across browsers and devices including Microsoft Edge – the default browser for Windows 10 – with free tools on dev.modern.IE (http://dev.modern.ie/tools/?utm_source=SitePoint&utm_medium=article43&utm_campaign=SitePoint):

- Scan your site for out-of-date libraries, layout issues, and accessibility (http://dev.modern.ie/tools/staticscan/?utm_source=SitePoint&utm_medium=article43&utm_campaign=SitePoint)
- Use virtual machines for Mac, Linux, and Windows (http://dev.modern.ie/tools/vms/windows/?utm_source=SitePoint&utm_medium=article43&utm_campaign=SitePoint)
- Remotely test for Microsoft Edge on your own device (https://remote.modern.ie/?utm_source=SitePoint&utm_medium=article43&utm_campaign=SitePoint)
- Coding Lab on GitHub: Cross-browser testing and best practices (https://github.com/deltakosh/interoperable-web-development)

In-depth tech learning on Microsoft Edge and the Web Platform from our engineers and evangelists:

- Microsoft Edge Web Summit 2015 (http://channel9.msdn.com/events/WebPlatformSummit/2015/?WT.mc_id=13419-DEV-sitepoint-article43) (what to expect with the new browser, new supported web platform standards, and guest speakers from the JavaScript community)
- Woah, I can test Edge & IE on a Mac & Linux! (https://channel9.msdn.com/Events/WebPlatformSummit/2015/Woah-I-Can-Test-Edge--IE-on-a-Mac--Linux/?WT.mc_id=13419-DEV-sitepoint-article43) (from Rey Bango)
- Advancing JavaScript without Breaking the Web (http://channel9.msdn.com/Events/WebPlatformSummit/2015/Advancing-JavaScript-without-breaking-the-web/?WT.mc_id=13419-DEV-sitepoint-article43) (from Christian Heilmann)
- The Edge Rendering Engine that makes the Web just work (https://channel9.msdn.com/Events/WebPlatformSummit/2015/The-Microsoft-Edge-Rendering-Engine-that-makes-the-Web-just-work/?WT.mc_id=13419-DEV-sitepoint-article43) (from Jacob Rossi)
- Unleash 3D rendering with WebGL (https://channel9.msdn.com/Events/WebPlatformSummit/2015/Unleash-3D-rendering-with-WebGL-and-Microsoft-Edge/?WT.mc_id=13419-DEV-sitepoint-article43) (from David Catuhe including the vorlon.JS (http://vorlonjs.com) and babylonJS (http://babylonjs.com) projects)
- Hosted web apps and web platform innovations (https://channel9.msdn.com/Events/WebPlatformSummit/2015/Hosted-

web-apps-and-web-platform-innovations/?WT.mc_id=13419-DEV-sitepoint-article43) (from Kevin Hill and Kiril Seksenov including the manifold.JS (http://manifold.js.com) project)

More free cross-platform tools & resources for the Web Platform:

Visual Studio Code for Linux, MacOS, and Windows (https://code.visualstudio.com/?WT.mc_id=13419-DEV-sitepoint-article43) Code with node.JS (https://www.microsoftvirtualacademy.com/en-US/training-courses/building-apps-with-node-js-jump-start-8422/?WT.mc_id=13419-DEV-sitepoint-article43) and free trial on Azure (https://azure.microsoft.com/en-us/pricing/free-trial/?WT.mc_id=13419-DEV-sitepoint-article43)

Tags: asp.net (https://www.sitepoint.com/tag/asp-net-tag/), visual studio (https://www.sitepoint.com/tag/visual-studio/)

Was this helpful?

Michael Oneppo

(https://www.sitepoint.com/author/moneppo/)

COURSES > (https://www.sitepoint.com/premium/topics/all?q=&content_types[]=Course&utm_source=sitepoint&utm_medium=relatedpremiu Dev @ Microsoft)

3:07:36

JavaScript: Next Steps

M. David Green

★★★★★

(https://www.sitepoint.com/premium/courses/javascript-next-steps-2921/?utm_source=sitepoint&utm_medium=relatedpremiumarticlefooter&utm_camp Dev @ Microsoft)

1:49:07

Your First Meteor 1.2 Application

David Turnbull

★★★★½

(https://www.sitepoint.com/premium/courses/your-first-meteor-1-2-applicatic utm_source=sitepoint&utm_medium=relatedpremiumarticlefooter&utm_camp Dev @ Microsoft)

1:11:20

React The ES6 Way

Darin Haener

★★★★☆

(https://www.sitepoint.com/premium/courses/react-the-es6-way-2914/?utm_source=sitepoint&utm_medium=relatedpremiumarticlefooter&utm_camp Dev @ Microsoft)

BOOKS > (https://www.sitepoint.com/premium/topics/all?q=&content_types[]=Book&utm_source=sitepoint&utm_medium=relatedpremium Dev @ Microsoft)

Hugo Giraudel

★★★★½

(https://www.sitepoint.com/premium/books/jump-start-sass/?utm_source=sitepoint&utm_medium=relatedpremiumarticlefooter&utm_camp
Dev @ Microsoft)

M. David Green

★★★★½

(https://www.sitepoint.com/premium/books/scrum-novice-to-ninja/?utm_source=sitepoint&utm_medium=relatedpremiumarticlefooter&utm_camp
Dev @ Microsoft)

ECMAScript 2015: A SitePoint Anthology

James Hibbard

★★★★½

(https://www.sitepoint.com/premium/
utm_source=sitepoint&utm_medium=r
Dev @ Microsoft)

SCREENCASTS >
(https://www.sitepoint.com/premium/topics/all?q=&content_types[]=ScreenCast&utm_source=sitepoint&utm_medium=relatedpr
Dev @ Microsoft)

Finding Bugs in Your Commits with Git Bisect

Shaumik Daityari

(https://www.sitepoint.com/premium/
bisect/?
utm_source=sitepoint&utm_medium=r
Dev @ Microsoft)

Creating Custom Sass Functions

Guilherme Muller

(https://www.sitepoint.com/premium/tutorials/creating-custom-sass-functio

About

Our Story (/about-us/)
Advertise (/advertising/)
Press Room (/press/)
Reference (http://reference.sitepoint.com/css/)
Terms of Use (/legals/)
Privacy Policy (/legals/#privacy)
FAQ (https://sitepoint.zendesk.com/hc/en-us)
Contact Us (mailto:feedback@sitepoint.com)
Contribute (/write-for-us/)

Visit

SitePoint Home (/)
Forums (https://www.sitepoint.com/community/)
Newsletters (/newsletter/)
Premium (/premium/)
References (/sass-reference/)
Shop (https://shop.sitepoint.com)
Versioning (https://www.sitepoint.com/versioning/)

Connect

(https://www.sitepoint.co
(/newsletter/)
(https://www.facebook.cc
(http://twitter.com/sitepo
(https://plus.google.com/