游戏开发

在一小时内开发网络游戏

Michael Oneppo

下载代码示例

您无需使用全新的技能集来开发游戏。事实上，HTML、JavaScript 和 CSS 等中的当前 Web 开发技能广泛适用于各种游戏。在使用 Web 技术制作游戏时，它几乎可以在带有浏览器的所有设备上运行。

为了证明这一点，我将演示如何使用 Web 技术和两个外部库在不到一小时的时间内从零开始制作一个游戏。我将介绍各种游戏开发主题，从基本设计和布局、控件和子画面到简单对手的人工智能 (AI)。我甚至要开发可在 PC、平板电脑和智能手机上运行的游戏。如果您对作为 Web 开发人员编程或其他开发领域有一些体验，但没有编写游戏的经验，本文将帮助您入门。请给我一小时，我一定向您展示相关窍门。

启动并运行

我将在 Visual Studio 中进行所有的开发工作，这将会允许随着我所进行的修改而快速执行 Web 应用。确保您安装的是最新版本的 Visual Studio（下载地址 bit.ly/1xEjEnX），那么您可以继续。我使用的是 Visual Studio 2013 Pro，但是 Visual Studio 2013 Community 中更新了代码。

此应用无需任何服务器代码，因此我首先在 Visual Studio 中新建空的网页项目。选择“文件”|“新建”|“ASP.NET 空白网站”后，选择 Visual C# 选项即可为网站使用空白的 C# 模板。

索引 HTML 文件仅需要以下三种资源：jQuery、主样式表和主 JavaScript 文件。在加载网页时，我将一个空的 CSS 文件添加到名称为 style.css 的项目和一个名称为 ping.js 的空 JavaScript 文件以避免错误：

```
<!DOCTYPE html>
<html>
<head>
  <script src="http://ajax.aspnetcdn.com/ajax/jQuery/jquery-2.1.1.min.js"></script>
  <script src="ping.js"></script>
  <link rel="stylesheet" href="style.css"></script>
</head>
<body>
</body>
</html>
```

基本设计

我制作的游戏是我称作 Ping 的 Pong 的变体。除了有一点不同之外（当球击向任一个玩家时，该玩家会抓住球，然后直接把球回击回去，或者按照一定的角度向上或向下击球），Ping 和 Pong 在本质上具有相同的规则。通常，在制作游戏之前，最好绘制一下您预期的游戏外观。对于此游戏，我想要看到的整体布局显示在**图 1** 中。

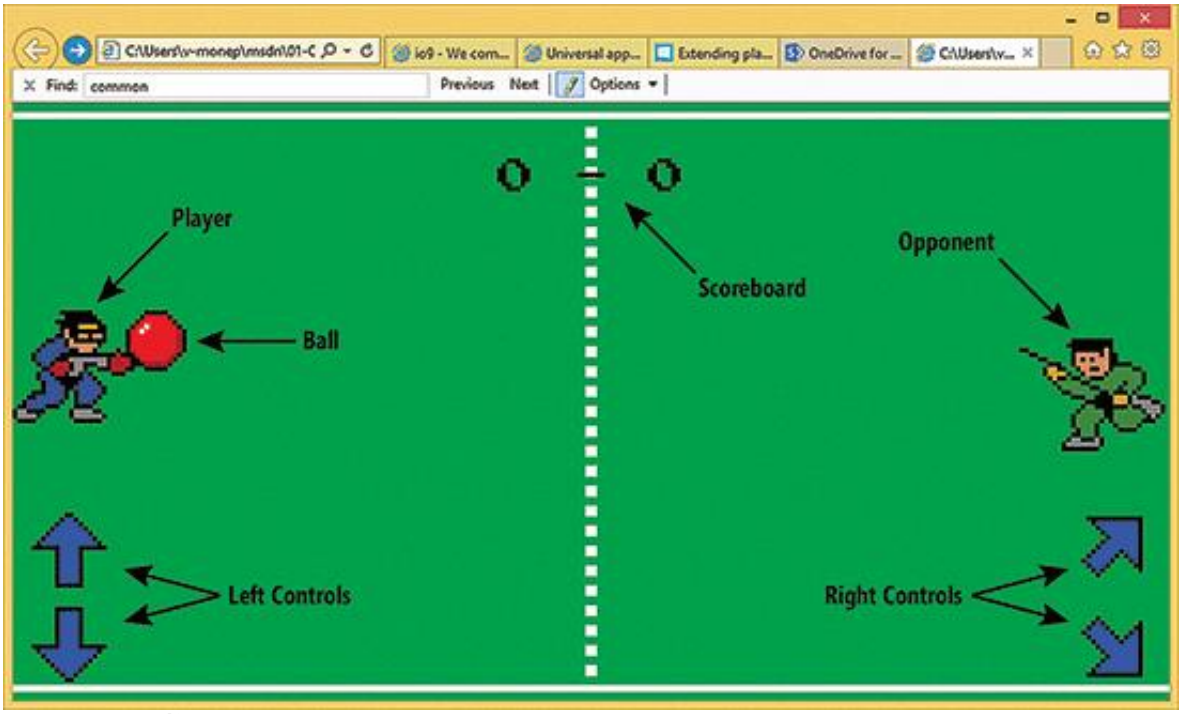

图 1 Ping 的整体设计

开发游戏设计布局后，剩下需要做的就是向 HTML 中添加每个元素以制作游戏。但需要注意的是，我将计分板和控件组合到一起，以确保它们并排出现。那么，如您所见，我已经逐个添加了相应元素（如**图 2** 中所示）。

图 2 初始 HTML 布局

```
<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>
```

设置样式

如果您要加载此页面，则看不到任何内容，因为没有应用任何样式。我已经创建一个指向我的 HTML 中的 main.css 文件的链接，因此我可以将所有 CSS 存放到含有该名称的新文件。我要做的第一件事情是定位屏幕上所有内容。页面的主体需要占据整个屏幕，因此我首先对其进行设置：

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

```
}
```

其次，我需要让竞技场充满整个屏幕，将竞技场背景图像应用到整个屏幕（请参见**图 3**）：

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

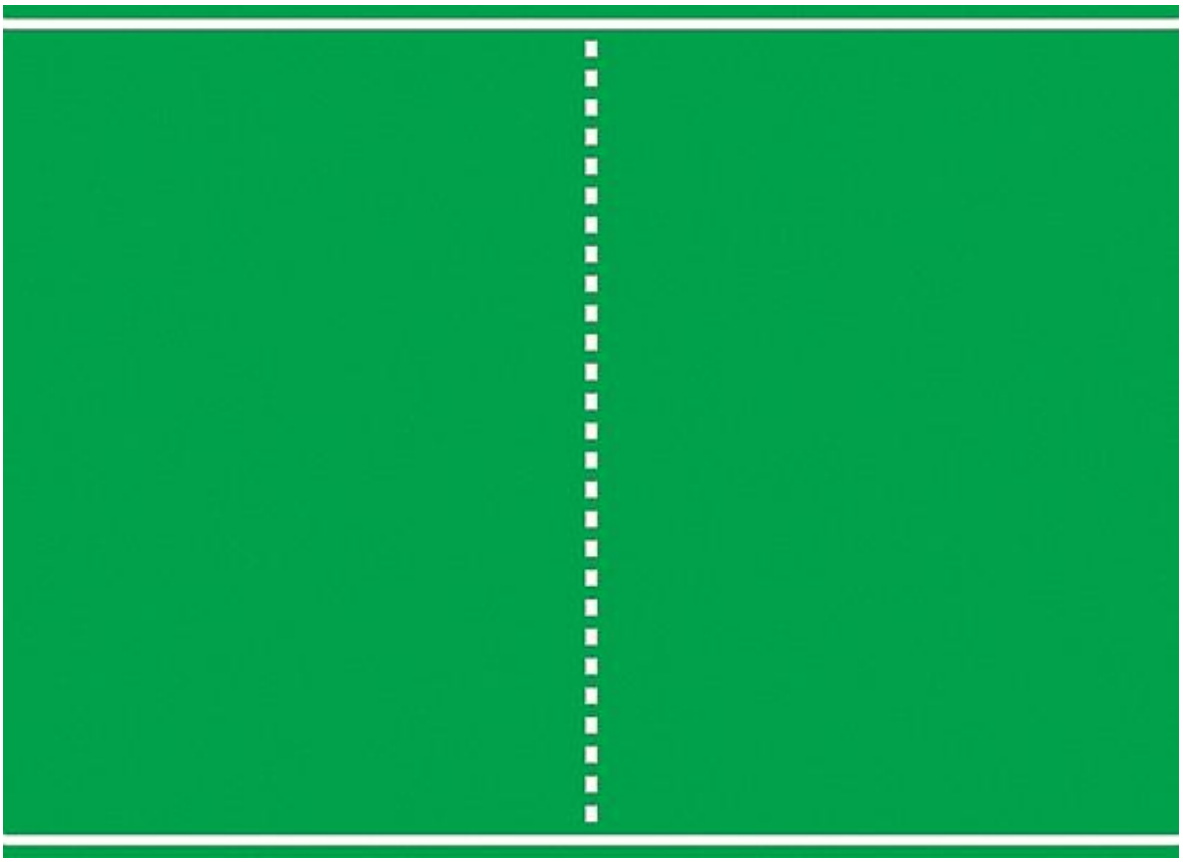

图 3 竞技场背景图像

接下来，我将设置计分板的位置。我希望计分板显示在顶部中心，位于其他元素之上。命令位置：绝对可以让我将其放到我想要的任何位置，左侧：50%，将其放到窗口顶部一半的位置，但是从计分板元素的最左侧开始放置。若要使其完全居中，我会使用转换属性和 Z-索引属性来确保其始终位于顶部：

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

我还希望文本字体具有复古风。大部分新型浏览器可允许我添加我自己的字体。我从 codeman38 (zone38.net) 中发现适当的 Press Start 2P 字体。若要向计分板添加该字体，我必须创建新的字体：

```
@font-face {
  font-family: 'PressStart2P';
```

```
  src: url('PressStart2P.woff');
}
```

现在，分数位于 h1 标记中，因此我可以为所有 h1 标记设置字体。为了防止该字体缺失，我将提供几个备用选项：

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

对于其他元素，我将使用图像的子画面表单。在一个文件中，子画面表单包含该游戏所需的所有图像（请参见**图 4**）。

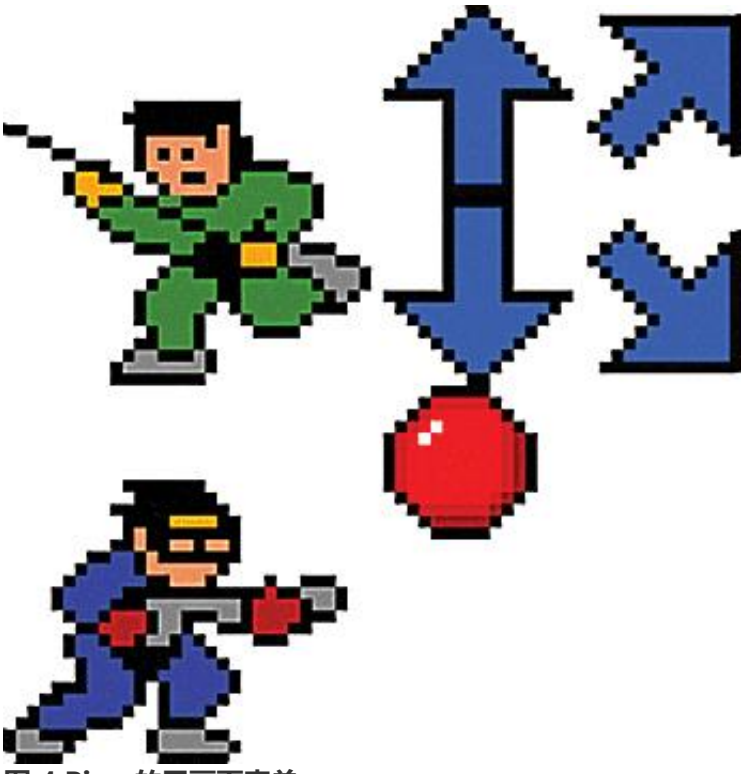

图 4 Ping 的子画面表单

含有此表单上的图像的所有元素都含有指定的 sprite 类。然后，对于每个元素，我将使用背景定位来定义我要显示的子画面表单的部分：

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

接下来，我将向使用该 sprite 类的所有元素添加该类。我需要暂时跳转回 HTML 以执行此操作：

```
<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>
```

现在，我需要在每一个元素的表单上指出每个子画面的位置。同样，我可以使用背景定位来执行此操作，如**图 5** 中所示。

图 5 为子画面表单添加偏移值

```
#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;
}
```

位置：玩家、对手和球的绝对属性可以让我通过 JavaScript 来移动它们。如果您现在查看该页面，您将发现有一些不必要的东西附加在这些控件和球上。这是因为子画面尺寸小于默认的 128 像素，所以我将它们调整到适当的尺寸。由于只有一个球，因此我将直接设置其尺寸：

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

因为有四个控件元素（用户可以按下以移动玩家的按钮），所以我理应为它们创建一个特殊类。我还将添加边距，以便让它们周围有一点空间：

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

添加此类后，游戏的控件看上去更加不错：

```
<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>
```

我需要做的最后一件事情是定位控件，以便于在移动设备上运行该页面时，这些控件在用户的手指范围内。我将它们放到底部角落中：

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

此设计的一大优点在于所有东西的位置都是相对的。这意味着，尽管屏幕可以有各种不同的尺寸，但游戏看上去始终良好。

追逐跳动的球

现在我要让球来回移动。对于 JavaScript 代码，我将引用 HTML 中名为 ping.js 的文件，就像我使用 CSS 执行的操作一样。我将此代码添加到含有该名称的新文件。我打算为该球和每个玩家创建对象，但是我将使用适用于这些对象的工厂模式。

这是一个简单的概念。当您调用 Ball 函数时，它会新建一个球。无需使用新关键字。通过清晰化可用的对象属性，此模式会减少有关此变量的混乱。而且，由于我只有一个小时的时间来制作此游戏，因此我需要最大程度地减少概念的混乱。

当我构建简单的 Ball 类时，此模式的结构如**图 6** 中所示。

图 6 Ball 类

```
var Ball = function( {
  // List of variables only the object can see (private variables).
  var velocity = [0,0];
  var position = [0,0];
  var element = $('#ball');
  var paused = false;
  // Method that moves the ball based on its velocity. This method is only used
  // internally and will not be made accessible outside of the object.
  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 use.
  // Right now this will just be the ability to update the state of the ball,
  // and start and stop the motion of the ball.
  return {
    update:       update,
    pause:        pause,
    start:        start
}
```

若要制作一个新的球，我只需调用已定义的这一函数：

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

现在，我要使球在屏幕上移动和跳动。首先，我需要按照某个间隔调用更新函数，以创建球的动画。新型浏览器提供一个用于此用途的函数，称为 requestAnimationFrame。这会将某个函数视作参数，并且会在下次运行其动画循环时调用该传入的函数。当浏览器准备好进行更新时，此举可以让球流畅地移动。当它调用传入的函数时，它将为该函数提供自加载了页面后的时间（以秒为单位）。这对于确保动画始终一致很关键。在该游戏中，requestAnimationFrame 的用法如下所示：

```
var lastUpdate = 0;
var ball = Ball();
function update(time) {
  var t = time - lastUpdate;
  lastUpdate = time;
  ball.update(t);
  requestAnimationFrame(update);
}
requestAnimationFrame(update);
```

请注意，requestAnimationFrame 在函数中被再次调用，因为球已完成更新。这可确保动画的连续。

虽然此代码可以工作，但是可能存在问题，即脚本会在页面完全加载之前就开始运行。若要避免这种情况，在加载页面时，我将使用 jQuery 启动代码：

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

由于我知道球的速度（速率）和最后一次更新到现在的时间，因此我可以执行一些简单的物理操作让球向前移动：

```
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');
  }
```

尝试运行代码，您将看到球按照某个角度移动并往屏幕边缘移动。这种乐趣会持续一秒钟，但是当球移出屏幕的边缘时，这种乐趣就结束了。因此，下一步是使球从屏幕的边缘弹回来，如**图 7** 中实现的那样。添加此代码并运行该应用将显示连续跳动的球。

图 7 球跳动的简单物理运动

```
var move = function(t) {
  // If the ball hit the top or bottom, reverse the vertical speed.
  if (position[1] <= 0 || position[1] >= innerHeight) {
   velocity[1] = -velocity[1];
  }
  // If the ball hit the left or right sides, reverse the horizontal speed.
  if (position[0] <= 0 || position[0] >= innerWidth) {
   velocity[0] = -velocity[0];
  }
  position[0] += velocity[0] * t;
  position[1] += velocity[1] * t;
  element.css('left', (position[0] - 32) + 'px');
  element.css('top', (position[1] - 32) + 'px');
}
```

可移动的玩家

现在可以制作 Player 对象了。充实 player 类的第一步是让 move 函数更改玩家的位置。side 变量将指出玩家将处于场地的哪一边，这将指明如何水平定位玩家的位置。传入到 move 函数的 y 值是玩家向上或向下移动的高度：

```
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; }
  }
}
```

图 8 展示了玩家的移动情况，同时在玩家子画面到达窗口的顶部或底部时停止移动。

现在，我可以创建两个玩家并且让它们移动到屏幕的相应侧：

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

图 8 玩家子画面的移动控件

```
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 any
  // 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 player position
  // 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');
}
```

键盘输入

那么理论上，您可以移动该玩家，但是在没有指示的情况下它不会移动。向左侧的玩家添加一些控件。您希望通过两种方式控制该玩家：（在电脑上）使用键盘和（在平板电脑和手机上）轻按控件。

若要确保各种平台上的触摸输入和鼠标输入之间的一致性，我将使用极佳的统一框架 Hand.js (handjs.codeplex.com)。首先，我将在 HTML 标头部分中添加脚本：

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

图 9 演示了在键盘上按下 A 键和 Z 键或轻按控件时如何使用 Hand.js 和 jQuery 来控制玩家。

图 9 添加触摸和键盘控件

```
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 pointers -- a finger,
  // a mouse, a stylus and so on.
  $('#up')    .bind("pointerdown", function() {player.move(-distance);});
  $('#down')  .bind("pointerdown", function() {player.move(distance);});
  requestAnimationFrame(update);
});
$(document).keydown(function(event) {
  var event = event || window.event;
  // This code converts the keyCode (a number) from the event to an uppercase
  // 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;
});
```

抓住球

当球跳来跳去时，我想要玩家抓住该球。抓住球后，该球就有了所有者，并且它会遵循该所有者的动作。**图 10** 向该球的移动方法中添加相应功能，允许球有所有者，球将遵循所有者发出的动作。

图 10 让球遵循其所有者发出的动作

```
var move = function(t) {
  // If there is an owner, move the ball to match the owner's position.
  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 horizontal bouncing
  // 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 speed.
    if (position[1] - 32 <= 0 || position[1] + 32 >= innerHeight) {
      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');
}
```

目前，无法获取 Player 对象的位置，因此我将向 Player 对象添加 getPosition 和 getSide 访问器：

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

现在，如果球有了所有者，则它会遵循该所有者发出的动作。但是，我如何确定所有者呢？必须有人抓住球。**图 11** 显示其中一个玩家子画面触摸到球的确定方法。出现这种情况时，我将把球的所有者设置为该玩家。

Figure 11 球和玩家的碰撞检测

```
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 player.
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;
}
```

如果您现在试图玩此游戏，您会发现球从屏幕的顶部弹回，而且您可以移动玩家去抓住球。现在，如何将球扔出去呢？这就是右侧控件所要做的，即瞄准球。**图 12** 向玩家添加了一个“fire”函数和一个 aim 属性。

图 12 瞄准并发球

```
var aim = 0;
var fire = function() {
  // Safety check: if the ball doesn't have an owner, don't not mess with it.
  if (ball.getOwner() !== this) {
    return;
  }
  var v = [0,0];
  // Depending on the side the player is on, different directions will be thrown.
  // The ball should move at the same speed, regardless of direction --
  // with some math you can determine that moving .707 pixels on the
  // x and y directions is the same speed as moving one pixel in just one directic
  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; },
}
```

图 13 扩大键盘功能，以设置玩家的瞄准和触发功能。瞄准的工作情况可能稍有不同。释放瞄准键后，瞄准将返回为径直。

图 13 设置玩家的瞄准功能

```
$(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;
});
```

最终添加内容将在所有控件上支持触摸。我将让右侧的控件更改玩家的瞄准情况。此外，我还要让触摸屏幕的任意位置都可以触发球：

```
$('#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();});
```

记录得分

当球传递到玩家后，我希望更改分数并将该球记入该玩家。我将使用自定义事件，这样我可以对任何现有对象分别计分。随着 update 函数越来越长，我将添加一个名为 checkScored 的新私有函数：

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

图 14 显示与这些事件互动的代码，以更新分数并传球。将此代码添加到 JavaScript 文档的底部。

图 14 更新计分板

```
$(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();
});
```

现在，当向对手传递球（这并不是很难，因为对手没有移动）时，您的分数将会增加，而且球将传递到对手。但是，对手只是抓住球。

进行智能化

游戏几乎已经成型了。要是有个玩家一起玩就更好了。作为最后一步，我将演示如何使用简单 AI 来控制对手。对手会尝试在球移动时与其保持平行。如果对手抓住了球，它将随机移动并朝着任意方向发球。若要使 AI 有一点人性化，我将在所做的动作中添加延迟。提醒您，这并非高度智能的 AI，但是可以将其作为对手一起玩游戏。

在设计此类系统时，最好要郑重地考虑。对手 AI 有三种可能状态：跟随、瞄准/发射和等待。我将是跟随动作之间的状态，以添加更多的人性化元素。就这样和 AI 对象开始吧：

```
function AI(playerToControl) {
  var ctl = playerToControl;
  var State = {
    WAITING: 0,
    FOLLOWING: 1,
```

```
    AIMING: 2
  }
  var currentState = State.FOLLOWING;
}
```

根据 AI 的状态，我希望它做不同的动作。就像球一样，我将构建一个可以在 requestAnimationFrame 中调用的 update 函数，以让 AI 根据其状态做出动作：

```
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;
  }
}
```

FOLLOWING 状态非常简单。对手按照球的垂直方向移动，而且 AI 转变到 WAITING 状态，以注入一段缓慢的反应时间。**图 15** 显示这两个状态。

图 15 简单的 FOLLOWING AI

```
function moveTowardsBall() {
  // Move the same distance the player would move, to make it fair.
  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;
    }
  }
}
```

使用**图 15** 中的代码，AI 在跟随球和等待一秒之间进行交替。现在，向游戏范围的 update 函数添加该代码：

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

```
  }
```

在运行游戏时，您将发现对手跟随着球的运动（即，不到 30 行代码就创建出一个还不错的 AI）。当然，如果对手抓住球，则它不会发出任何动作。因此，这一小时内的最后一个技巧该是处理 AIMING 状态的动作了。我希望 AI 随机移动几次，然后往任意方向将球发出。**图 16** 添加了执行上述动作的私有函数。向 AIMING 选择语句添加 aimAndFire 函数将使 AI 的功能更加完备，从而与其一起玩游戏。

图 16 瞄准并触发 AI

```
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);
}
```

总结

到目前为止，您拥有了一个可在电脑、智能手机和平板电脑上运行的、成熟的网络游戏。还可以对该游戏进行很多潜在的改进。例如，在智能手机上纵向玩此游戏有点困难，因此您需要确保横向拿着手机才能让该游戏正常工作。对于适用于 Web 以及 Web 以外的游戏开发的种种可能，这里只是做了一个小小的演示。

Michael Oneppo 是富有创造性思维的技术专家并且以前是 Microsoft Direct3D 团队的项目经理。他最近的工作包括担任 Library For All（非盈利技术）的 CTO 以及致力于探索获得 NYU 交互式电信计划方面的硕士学位。

衷心感谢以下技术专家对本文的审阅：Mohamed Ameen Ibrahim

MSDN Magazine Blog

14 Top Features of Visual Basic 14: The Q&A
Wednesday, 1月 7 by Michael Desmond - MSDN Magazine
Big Start to the New Year at MSDN Magazine
Friday, 1月 2 by Michael Desmond - MSDN Magazine

More MSDN Magazine Blog entries >

Current Issue

Browse All MSDN Magazines

Subscribe to MSDN Flash newsletter

Receive the MSDN Flash e-mail newsletter every other week, with news and information personalized to your interests and areas of focus.

关注我们

注册 MSDN 时事通讯

此页面有用吗？ 是 否

开发人员中心

Windows

学习资源

Microsoft 虚拟学院

第 9 频道

社区

论坛

博客

支持

自助支持