Case Study - Auto-Resizing HTML5 Games

- Preparing the Page
- Resizing the Game
- Summary

In the summer of 2010, we created Sand Trap, a game that we entered in a competition on HTML5 games for mobile phones. But most mobile phones either displayed only part of the game or made the game too small—making it completely unplayable. So we took it upon ourselves to make the game fluidly adjust to match any resolution. After a bit of re-programming and using ideas outlined in this article, we had a game that scaled across any modern browser, whether it ran in a desktop or a mobile device.

Screenshot of thwack fullscreen

screenshot of thwack smaller in browser window

This approach worked well for Sand Trap, so we used the same method on our latest game, Thwack!. The game automatically adjusts screen resolutions to fit both full-screen and custom-sized windows, as shown in the screenshots below.

Implementing this required taking advantage of both CSS and JavaScript. Using CSS to fill the whole screen is trivial, but CSS does not allow you to maintain the same width-to-height ratio to prevent stretching of the canvas and game area. That's where JavaScript comes in. You can rescale document elements with JavaScript and trigger the resize on window events.

Preparing the Page

The first step is to designate the area on the page where the game will take place. If you include this as a div block, you can place other tags or a canvas element within it. By setting it up correctly, these child elements will inherit the scaling of the parent div block.

If you have two parts to your game area, a play area and a score-keeping area, it might look like this:

```
<div id="gameArea">
  <canvas id="gameCanvas"></canvas>
  <div id="statsPanel"></div>
</div>
```

Once you have a basic document structure, you can give these elements a few CSS properties to prepare them for resizing. Many of the CSS properties for “gameArea” are manipulated directly by JavaScript, but in order for them to work, set up a few other CSS properties starting with the parent gameArea div block:

```
#gameArea {
  position: absolute;
  left:     50%;
  top:      50%;
}
```

This puts the top left corner of the canvas in the center of the screen. The JavaScript auto-resize function described in the next section manipulates additional CSS properties to resize the game area and center it in the window.

Since the game area is automatically resized according to the dimensions of the window, you do not want the dimensions in pixels for the gameArea div block’s child elements; instead, you want it in percentages. Pixel values do not allow inner elements to scale with the parent div as it changes. However, it may be helpful to start with pixels and then convert them to percentages once you have a layout that you like.

For this example, start with the game area being 300 pixels tall and 400 pixels wide. The canvas covers the entire game area, and a semitransparent stats panel runs along the bottom at 24 pixels tall, as shown in Figure 1.

Dimensions of gameArea child elements in pixels

```
<figcaption>
Figure 1: Dimensions of gameArea child elements in pixels
</figcaption>
```

Translating these values to percentages makes the canvas 100% in width and 100% in height (of gameArea, not the window). Dividing 24 by 300 gives the height of the stats panel at 8%, and, since it will cover the bottom of the game area, it’s width will also be 100%, as seen in Figure 2.

Dimensions of gameArea child elements in percentages

Figure 2: Dimensions of gameArea child elements in percentages

Now that you have determined the dimensions of the game area and its child elements, you can put together the CSS properties for the two inside elements as follows:

```
#gameCanvas {
  width: 100%;
  height: 100%;
}
```

```
#statsPanel {
  position: absolute;
  width: 100%;
  height: 8%;
  bottom: 0;
  opacity: 0.8;
}
```

Resizing the Game

Now you are ready to create a function to handle the window being resized. First, grab a reference to the parent gameArea document element.

```
var gameArea = document.getElementById('gameArea');
```

Since you are not concerned about the exact width or height, the next piece of information you need to set is the ratio of width to height. Using your earlier reference of a game area of 400 pixels wide and 300 pixels high, you know that you want to set the aspect ratio at 4 units wide and 3 units high.

```
var widthToHeight = 4 / 3;
```

Since this function is called whenever the window is resized, you also want to grab the window's new dimensions so you are able to adjust your game's dimensions to match. Find this by using the window's innerWidth and innerHeight properties.

```
var newWidth = window.innerWidth;
var newHeight = window.innerHeight;
```

Just as you determined the ratio of width to height you want, now you can determine the window's current width to height ratio:

```
var newWidthToHeight = newWidth / newHeight;
```

This allows you to decide whether to make the game fill the screen vertically or horizontally, as shown in Figure 3.

Fitting the gameArea element to the window while maintaining aspect ratio

Figure 3: Fitting the gameArea element to the window while maintaining aspect ratio

If the desired game area shape is wider than the window's shape (and height is shorter), you need to fill up the window horizontally and leave margins along the top and bottom. Likewise, if the desired game area shape is higher than the window's shape (and the width is narrower), you need to fill up the window vertically and leave margins along the left and right.

To do this, test your desired ratio of width to height with the current window's ratio of width to height and make the appropriate adjustments as follows:

```
if (newWidthToHeight > widthToHeight) {
  // window width is too wide relative to desired game width
  newWidth = newHeight * widthToHeight;
  gameArea.style.height = newHeight + 'px';
  gameArea.style.width = newWidth + 'px';
} else { // window height is too high relative to desired game height
  newHeight = newWidth / widthToHeight;
  gameArea.style.width = newWidth + 'px';
  gameArea.style.height = newHeight + 'px';
}
```

Now that you have adjusted the width and height of the game area, you need to center things up by placing a negative margin on the top that is half of the height and on the left that is half of the width. Remember that CSS is already placing the top-left corner of the gameArea div at the exact center of the window, so this centers the game area in the window:

```
gameArea.style.marginTop = (-newHeight / 2) + 'px';
gameArea.style.marginLeft = (-newWidth / 2) + 'px';
```

You would also want to automatically adjust the font size. If you have all of the child elements using em, you can simply set the fontSize CSS property of the gameArea div block to a value determined by its size.

```
gameArea.style.fontSize = (newWidth / 400) + 'em';
```

Lastly, you want to make the canvas drawing dimensions match its new width and height. Note that the rest of the game code must keep game engine dimensions separate from the canvas drawing dimensions to accommodate for a dynamic canvas resolution.

```
var gameCanvas = document.getElementById('gameCanvas');
gameCanvas.width = newWidth;
gameCanvas.height = newHeight;
```

So the completed resize function might look something like this:

```
function resizeGame() {
    var gameArea = document.getElementById('gameArea');
    var widthToHeight = 4 / 3;
    var newWidth = window.innerWidth;
    var newHeight = window.innerHeight;
    var newWidthToHeight = newWidth / newHeight;

    if (newWidthToHeight > widthToHeight) {
        newWidth = newHeight * widthToHeight;
        gameArea.style.height = newHeight + 'px';
        gameArea.style.width = newWidth + 'px';
    } else {
        newHeight = newWidth / widthToHeight;
        gameArea.style.width = newWidth + 'px';
        gameArea.style.height = newHeight + 'px';
    }

    gameArea.style.marginTop = (-newHeight / 2) + 'px';
    gameArea.style.marginLeft = (-newWidth / 2) + 'px';

    var gameCanvas = document.getElementById('gameCanvas');
    gameCanvas.width = newWidth;
    gameCanvas.height = newHeight;
}
```

Now, you want these adjustments to be made automatically whenever the window is resized or, in the case of mobile devices, the screen orientation is changed. Handle these events by having them call your resizeGame() function like so:

```
window.addEventListener('resize', resizeGame, false);
window.addEventListener('orientationchange', resizeGame, false);
```

If the window is resized too high or the screen’s orientation is vertical, you are making width 100% of the window, and if the window is resized too wide or the screen’s orientation is horizontal, you are making height 100% of the window. The remaining dimension is sized according to the predetermined width-to-height aspect ratio.

Summary

Gopherwood Studios has used versions of this structure for all of our HTML5 games and it has proved very useful for accommodating multiple screen resolutions and various mobile devices. Additionally, with the aid of a full screen browser, this gives our web games an immersive experience more akin to traditional desktop gaming than many browser-based games. We look forward to more innovations in web games as HTML5 and web technologies continue to evolve.

□□□ #3
□ □□□ □□□ 2026-10-31 21:54:23 UTC
□ cocobook-sync □□□ 2026-11-04 17:37:14 UTC