How to Make an HTML5 Game, Part 2: Jumping, Falling, and Stopping

If you missed Part 1, you might want to check it out before we continue, since this tutorial builds off of the HTML5 game engine we started earlier. And for those of you that did read Part 1, you probably felt very overwhelmed. Sorry! The good news is, the hard part's over. Making the engine was the biggest step. Now we want to take that engine and turn it into a 2D platformer, just like *Super Markup Man*.

First, download this zip file. It's the same engine as before with some extra code added to it. Run the game by opening the *index.html* file in a browser. The character can now jump and stand on platforms. If you open the *graphics.js* file from the *scripts* folder in a text editor, you'll see that we're loading the new platform graphic on line 53:

var plankImage = loadImg('images/plank.png');

We'll draw the platforms on the canvas inside the **render()** function on line 63. Like the player, the platforms are objects with their own **draw()** function. Because we have more than one platform, though, they're stored inside an "array." We could just write the **draw()** function for each one like this:

platforms[0].draw();
platforms[1].draw();
platforms[2].draw();

But why do that when we can use a **for** loop? Then if we change the size, or **length**, of the array by adding more platforms, we don't have to worry about coming back here and writing more code. The **for** loop will handle it for us.

Those are the only changes to *graphics.js*. Open *game.js* now. On line 5 is the beginning of the platforms array. Inside the **init()** function, on line 36, there's another **for** loop that creates all of the platforms. Each platform has a position, a size, and a **draw()** function. Nothing too complicated, right? And if we decide we want more platforms, we can change the **for** loop to say **i < 5** or **i < 100**. But let's not get carried away!

Scroll down now to the **update()** function on line 64. Last time, we were using the **update()** function to check whenever the player touched the border of the canvas. Well, that was kind of boring, so I got rid of that code. Instead, we want the player to be able to stand on top of platforms. We can accomplish this by checking for "collisions" on every game update. The **for** loop on line 69 looks for a collision between the player and every platform and stops the player if a collision is found.

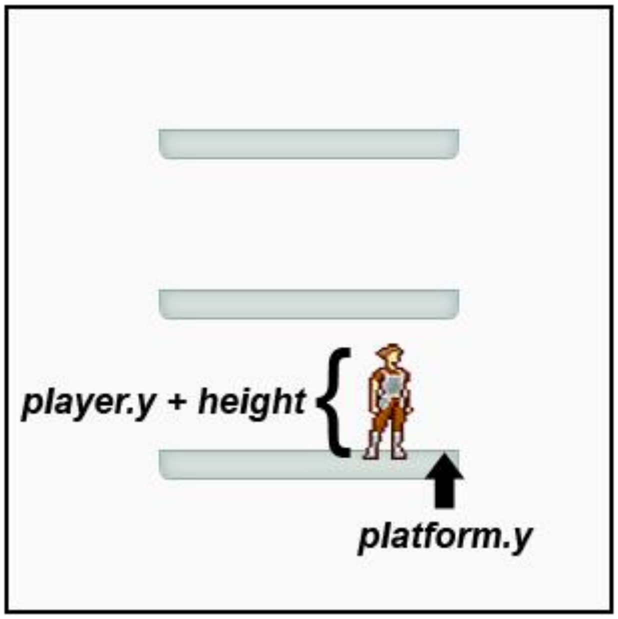
We don't care if the player hits a platform if he/she is holding the "down" key, though. That will let the player fall through it. In the **if** statement on line 72, we can check for that using the **keys** array:

if (!(keys.down in keysDown) && player.velocity >= 0) { }

The other part of the **if** statement looks at the player's **velocity**. That's a new value in the *player.js* file. It determines how fast the player falls. If the player's **velocity** is less than zero, that means the player is moving upwards, or jumping. If **velocity** is greater than zero, the player must be falling. So we only want to look for collisions if the player is falling AND the player isn't holding "down."

Once we know those two things are true, we can move on to the next **if** statement… and, oh boy, this one's ugly. There's no easy way to ask Javascript, "Hey, Javascript, did these two things touch?" Our only option is to compare the two objects' **x** and **y** values to see if the player is "inside" the platform. As soon as that is true, we'll bump the player back out by changing his/her **y** value:

player.y = platforms[i].y – player.height;



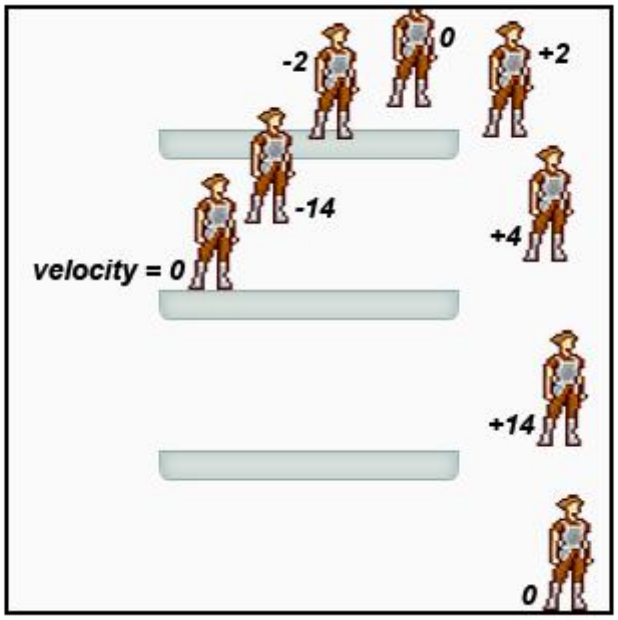
Don't forget to take into account the player's **height**. This move happens so quickly, though, that you don't even notice the player was inside anything. At the same time, we turn the player's ability to jump back on and reset his/her **velocity** (falling speed). Let's open the *player.js* file now to see how these new values are being used.

In *player.js*, the new variables are created on line 13: **velocity, jumpPower,** and **jumping**. These all control the player's up/down movement. On line 29, we're still using **keys.up in keysDown** to check for an "up" key press, but the player is only allowed to do this once until they touch the ground again. That's why the **jumping** variable changes from false to true and back again in various places. Inside this **if** statement, the player's **velocity** is set to the player's **jumpPower** (which is -14).

The **velocity** is super important, because it’s essentially our gravity. Gravity should always be affecting the player. On line 45, there’s a final formula that changes the player’s position based on his/her current **velocity:**

this.y += this.velocity;

When the player’s **velocity** is -14, that means he/she is jumping. Remember, subtracting from the player’s **y** position will make him/her move upwards. The lower the **y** value, the closer to the top the player is. If we leave the **velocity** at -14, though, the player will just fly off the screen. We need gravity to do its thing. So on line 37, we add a little to the **velocity**. Eventually, after several updates are called, the **velocity** will start to equal a positive number. When that’s applied to our final formula, the player will move back down**.**



Tada! We now have a 2D platformer where you can run, jump, fall, and collide with objects. But we’re not done yet. Next up, we’ll look at how to animate our character so he doesn’t look like a cardboard cutout floating around the screen. Take a look at *Super Markup Man* again and pay attention to how the character moves. Are you ready to do that?