

Select Solutions

to JavaScript Coding Activity

Add a background image to the scene.

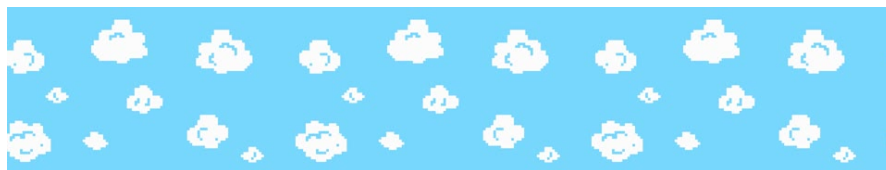
- In the styles.css file, the size of the gameWindow is set:

```
#gameWindow{  
  position: relative;  
  width: 800px;  
  height: 400px;  
  background-color: black;  
  overflow: hidden;  
  margin: 0 auto;  
}
```

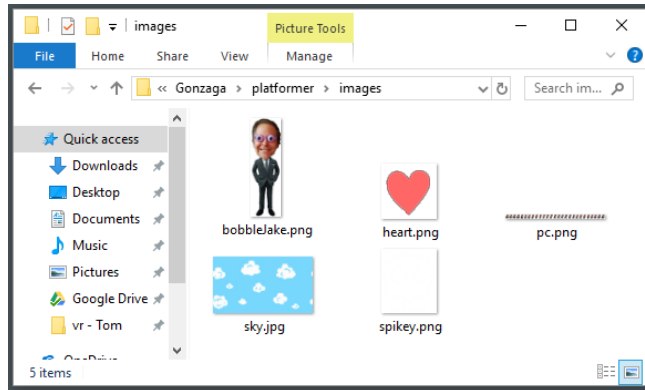
- Use PhotoShop, Illustrator, or some other software to create an image with an 800px width and a 400px height.
- Alternatively, you could create a smaller image that has no seams when tiled. Consider the sky image below.



- You can tell by looking at the sides that the image will tile without showing a seam. The image is repeated three times below.



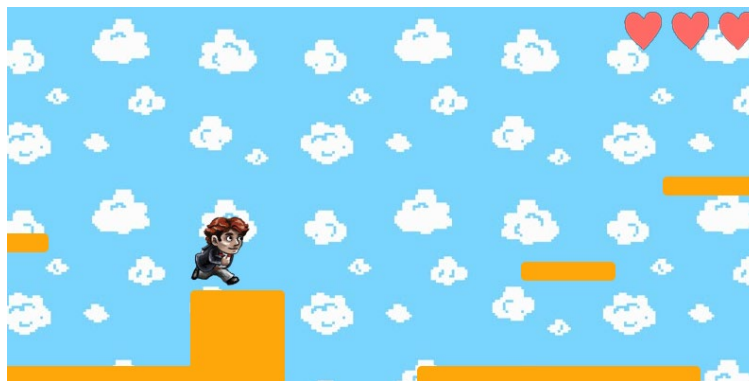
- Once ready, save the image in the images folder of the project.



- To skip the challenge and use the sky image above, you may download it from burkecs.org/platformer/saveDrE/images/sky.jpg
- Delete the CSS that sets the background-color of the gameWindow and add in the CSS that sets the background-image using the name and location of the file you created.

```
#gameWindow{
  position: relative;
  width: 800px;
  height: 400px;
  background-image: url("images/sky.jpg");
  overflow: hidden;
  margin: 0 auto;
}
```

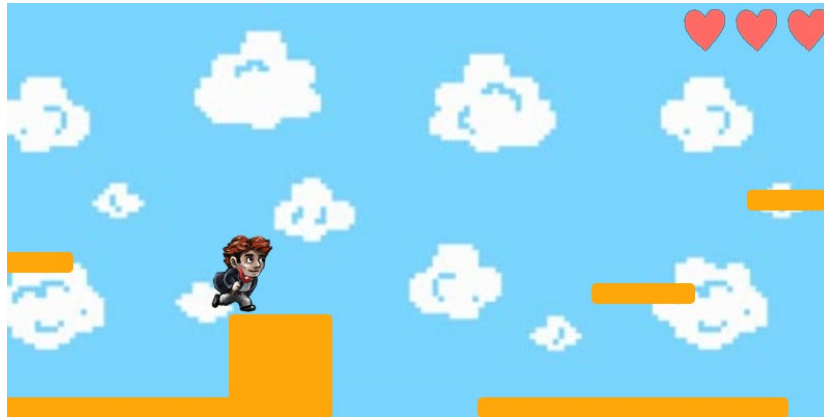
- When you refresh the page to play the game again, the image should appear!



- You may also want to set the size of your image in CSS, though this may pixelate the image if the resolution is low. You may change the image size using the background-size property. In the code snippet below, the size of the image is set to be twice as large.

```
height: 400px;  
background-image: url("images/sky.jpg");  
background-size: 640px 360px;  
overflow: hidden;
```

- And that, my friends, is how you add a background image to the game



Modify the speed or jump ability of the player character.

- In the engine.js file, the “player character” element is represented by the variable *pc*. In the *init()* function you can see where the *JUMP_FORCE* and *SPEED* properties of the *pc* element are initialized.

```
pc = document.getElementById('pc');  
pc.style.width = '72px';  
pc.style.height = '81px';  
pc.JUMP_FORCE = 18;  
pc.SPEED = 5;
```

- Simple change these values to your liking. Save the file and refresh the game. I thought the following values work well.

```
pc.JUMP_FORCE = 17;  
pc.SPEED = 10;
```

- And that, my friends, is how you modify the speed and jump ability of the player character.

Add platforms and “Spikey Guys” to build out full levels.

- The code for setting up each level can be found in the nextLevel() function of the engine.js file. The code for level 1 is shown below.

```
if(level==1){  
  drE.style.left = '1600px';  
  drE.style.top = '250px';  
  
  addPlatform(0, 380, 500, 20);  
  addPlatform(150, 240, 100, 20);  
  addPlatform(640, 380, 300, 20);  
  addPlatform(400, 300, 100, 100);  
  addPlatform(900, 180, 100, 20);  
  addPlatform(750, 270, 100, 20);  
  addPlatform(1180, 380, 1000, 20);  
  
  addSpikeyGuy(1300, 322);  
  
  sndMusic.src = 'audio/level_1_music.mp3';  
  sndMusic.volume = 0.2;  
  sndMusic.currentTime = 0;  
  sndMusic.play();  
}  
else if(level==2){
```

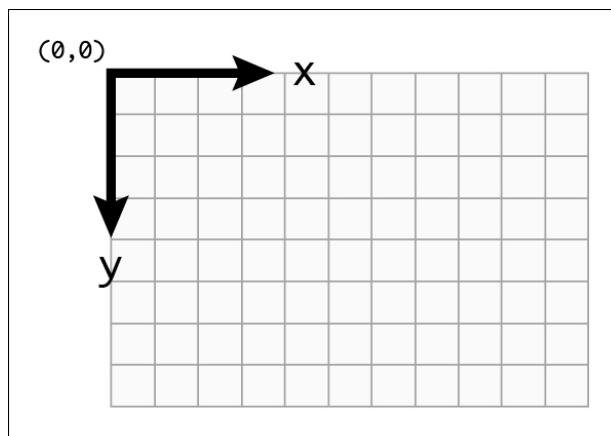
- Notice the addPlatform() function is used to add platforms and the addSpikeyGuy() is used to add “spikey guys”.
- What do all these numbers mean though? You can figure this out by looking at the variable names in the function definitions

found later in the code. Consider first the definition of the `addPlatform()` function.

```
function addPlatform(x, y, w, h){
  var p = document.createElement('DIV');
  p.className = 'platform';
  p.style.left = x + 'px';
  p.style.top = y + 'px';
  p.style.width = w + 'px';
  p.style.height = h + 'px';

  platforms.push(p);
  gameWindow.appendChild(p);
}
```

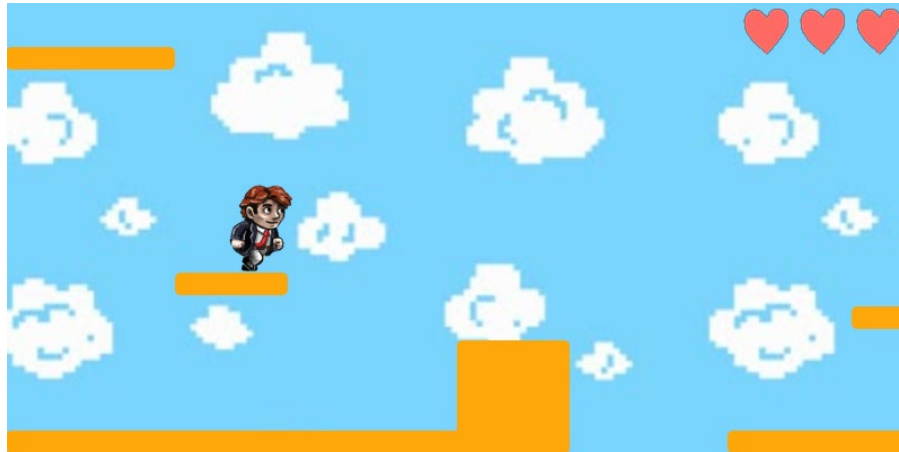
- The first two arguments (or parameters) are used for setting the position of the top left corner of the platform while the last two arguments are used to set the size of the platform.
 - The first argument is the x value, or the horizontal location of the platform.
 - The second argument is the y value, or the vertical location of the platform.
 - The third argument represents the width of the platform.
 - The fourth argument represents the height of the platform.
- It is important when positioning elements to understand that the origin is at the top-left corner of the `gameWindow` so increasing the x value moves the platform to the right while increasing the y value moves the platform down.



- Consider the following line of code

```
addPlatform(0, 40, 150, 20);
```

- The command adds a platform whose top-left corner is positioned in the gameWindow at (0, 40). That is 0 pixels from the left and 40 pixels from the top. The size of the platform is set to a width of 150 pixels and a height of 20 pixels.



- Adding Spikey Guys requires passing the x and y to set the location. As all Spikey Guys are intended to be the same size, the function has no arguments related to the width and height.

```
function addSpikeyGuy(x, y){
  var spikey = document.createElement('IMG');
  spikey.className = 'spikey';
  spikey.src = "images/spikey.png";
  spikey.style.left = x + 'px';
  spikey.style.top = y + 'px';
  spikey.style.width = '50px';
  spikey.style.height = '58px';

  spikeyGuys.push(spikey);
  gameWindow.appendChild(spikey);
}
```

- And that, my friends, is how you add platforms and “Spikey Guys” to build out complete levels for the game. This tedious task may be simplified by planning out your levels using [graph paper](#).

Add a countdown timer for each level.

- Many of the elements in the gameWindow are added through JavaScript, but some are added directly in HTML as you can see by looking at the index.html file.

```
<div id="gameWindow">
  <div id="pc"></div>
  <div id="drE"></div>
  <div id="lifebar"></div>
  <div id="btnContinue" onclick="nextLevel();">CONTINUE</div>
</div>
```

- Add a DIV element that will show the number of seconds remaining for the level being played. Give the DIV an id that describes the element and temporarily enter some text so you will be able to see how it looks in the game.

```
<div id="gameWindow">
  <div id="txtTime">TEXT</div>
  <div id="pc"></div>
  <div id="drE"></div>
  <div id="lifebar"></div>
  <div id="btnContinue" onclick="nextLevel();">CONTINUE</div>
</div>
```

- In the styles.css file, add CSS properties to style the DIV element as you wish.

```
#txtTime{
  position: absolute;
  left: 540px;
  top: 10px;
  width: 80px;
  padding: 10px;
  text-align: center;
  background-color: rgba(0, 0, 0, 0.25);
  color: white;
}
```

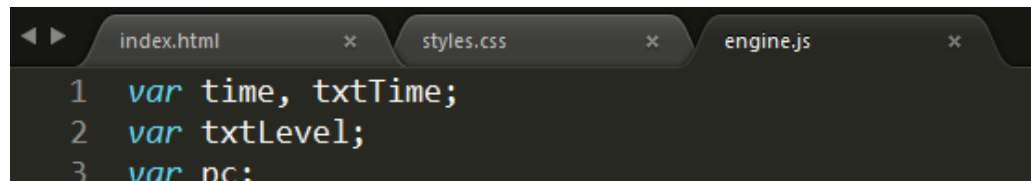
- Be sure to save the HTML and CSS files. With the CSS above, the time will be displayed at the top in front of the hearts.



- Once you have the time showing the way you want, remove the temporary text that you had placed in the txtTime DIV.

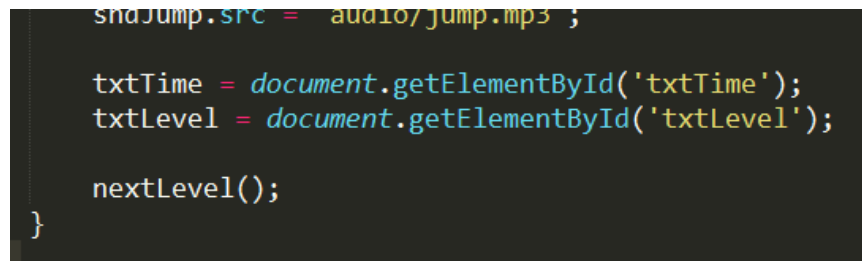
```
<div id="gameWindow">
  <div id="txtTime"></div>
  <div id="pc"></div>
```

- In the engine.js file, declare a *txtTime* variable at the top of the code that will be used to reference the txtTime DIV element and a *time* variable that will store the number of seconds remaining.



```
1 var time, txtTime;
2 var txtLevel;
3 var pc;
```

- In the init() function, initialize the value of the txtTime value. It may be logical to do this just before the line that sets the value of the txtLevel variable.

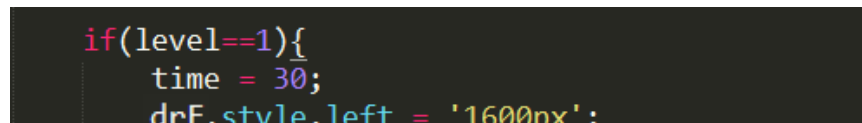


```
sndJump.src = audio/jump.mp3 ;

txtTime = document.getElementById('txtTime');
txtLevel = document.getElementById('txtLevel');

nextLevel();
}
```

-
- In the nextLevel() function, set the value of time in each if statement. For instance, if you want to allow 30 seconds to complete level 1, you would write:

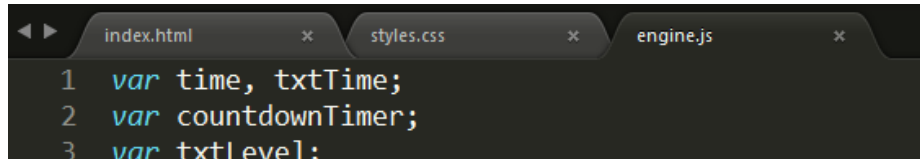


```
if(level==1){
  time = 30;
  drE.style.left = '1600px';
```

- Be sure to set the time for all your levels.
- When should we reduce the number of seconds? One method is to set up a function that executes every one second. This would be done similarly to how the gameTimer was set up to run the gameloop() function every 50 milliseconds to create a framerate of 20 frames per second).


```
gameTimer = setInterval(gameLoop, 50);
}
```

- To do this, declare a *countdownTimer* variable at the top of the code.



```
index.html x styles.css x engine.js x
1 var time, txtTime;
2 var countdownTimer;
3 var txtLevel;
```

- At the bottom of the nextLevel() function, use setInterval() to start a timer so that it executes the reduceTime() function that we will create next every second (i.e. 1000 milliseconds).
- This is also a good place to update the time in the txtTime DIV so that the DIV will not be empty at the start of the game.

```
gameTimer = setInterval(gameLoop, 50);
countdownTimer = setInterval(reduceTime, 1000);
txtTime.innerHTML = time;
}
```

- Create the reduceTime() function that will be executed every second.
 - Be sure not to write this code inside the code block of another function. It could be written between the end of the nextLevel() function and before the beginning of the addLife() function.

```
function reduceTime(){
    time--;
    txtTime.innerHTML = time;
    if(time <= 0){
        die("Oh no, you ran out of time.");
    }
}
```

- This function reduces the time by one, updates the txtTime DIV to show the new time, and then calls the die() function if the time is less than or equal to zero.

- Finally, we need to stop the timer at the appropriate times so the time does not continue decreasing when it should not be. In the die() function, add a clearInterval() for the countdownTimer.

```
function die(msg){
  clearInterval(gameTimer);
  clearInterval(countdownTimer);
  sndMusic.pause();
}
```

- And towards the end of the gameloop() function, add another clearInterval() to stop the countdown timer when the player runs into Dr. Enfield.

```
function reduceTime(){
  time--;
  txtTime.innerHTML = time;
  if(time <= 0){
    die("Oh no, you ran out of time.");
  }
}
```

- Now the countdown should be working, displaying, and the player should be dying when the time runs out.

