

COMP4021  
Internet Computing

# Making Games With Sprites

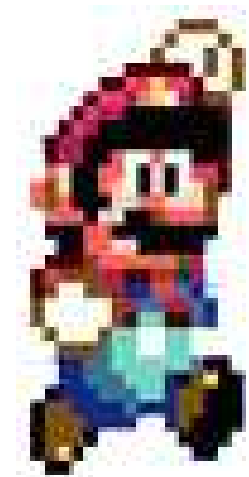
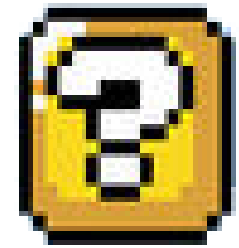
Gibson Lam

# Canvas and Games

- You can use canvas to make games on web pages
- In this presentation, we will look at using HTML and canvas to build 2D games with sprites
  - Drawing sprites
  - Working with animation
  - Playing audio in a web page

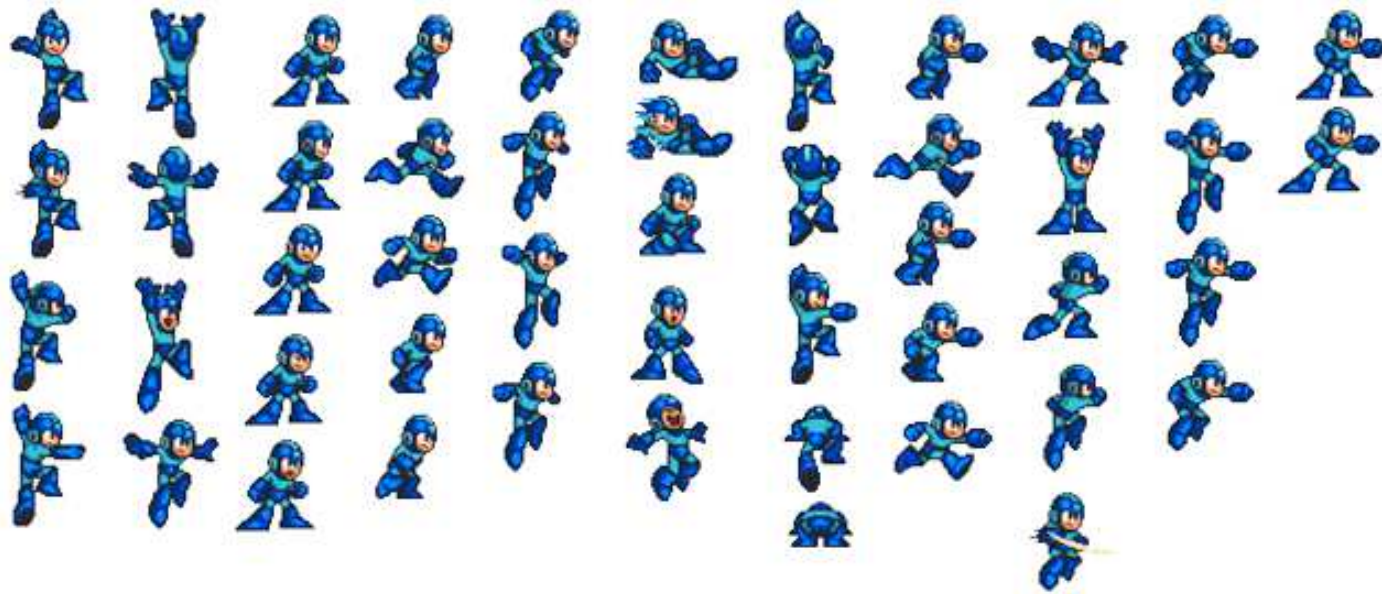
# Drawing Sprites in Canvas

- Sprites are images in games, which store the appearances of game objects
- These images are usually drawn over the game background with some clever transparency handling
- You can do that in canvas using `drawImage()` with transparent images, i.e. PNG files



# Sprite Sheets

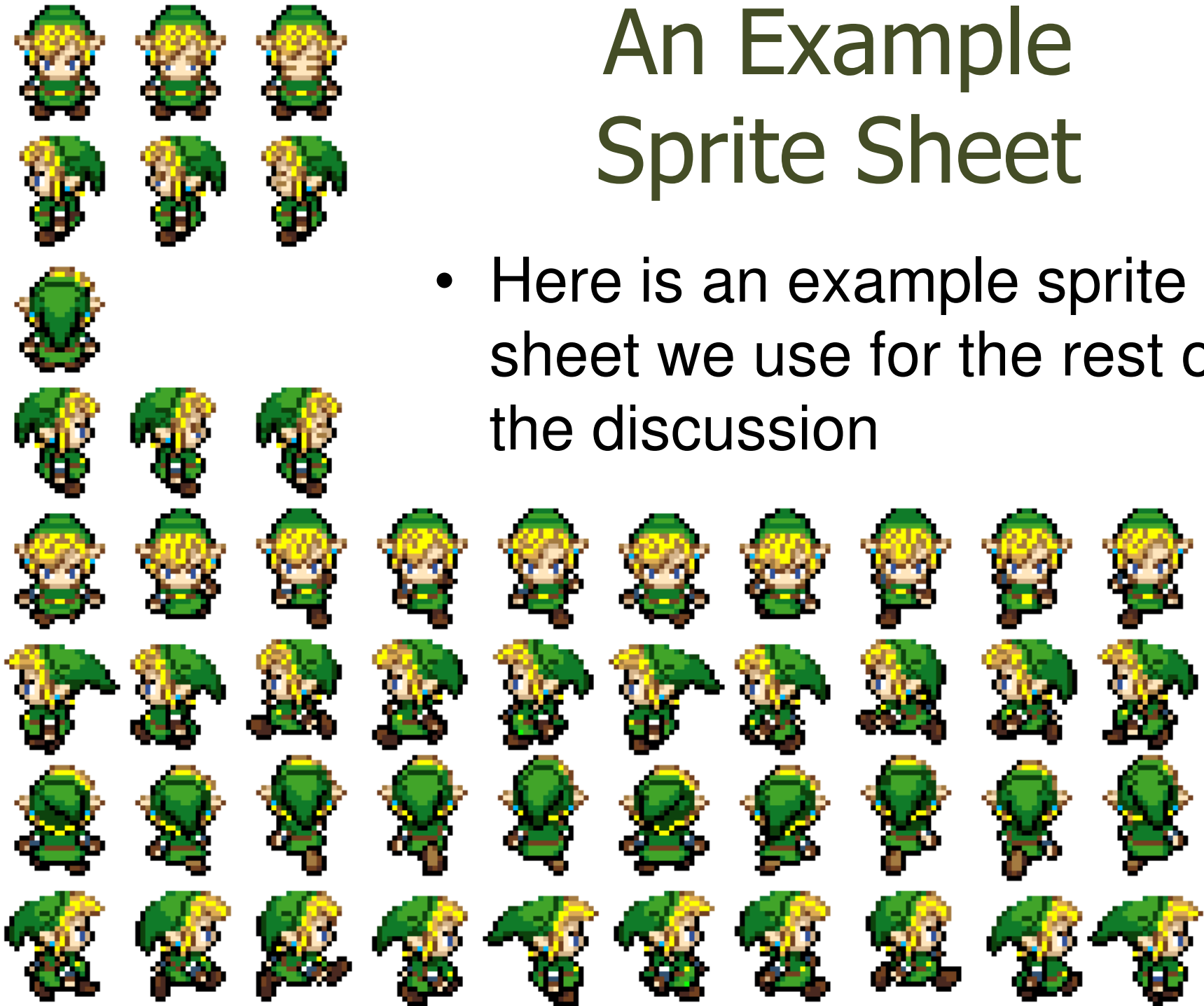
- A sprite sheet is a single image containing a collection of sprites from typically the same character or object



- You make sprite animations by drawing the appropriate sprites over time

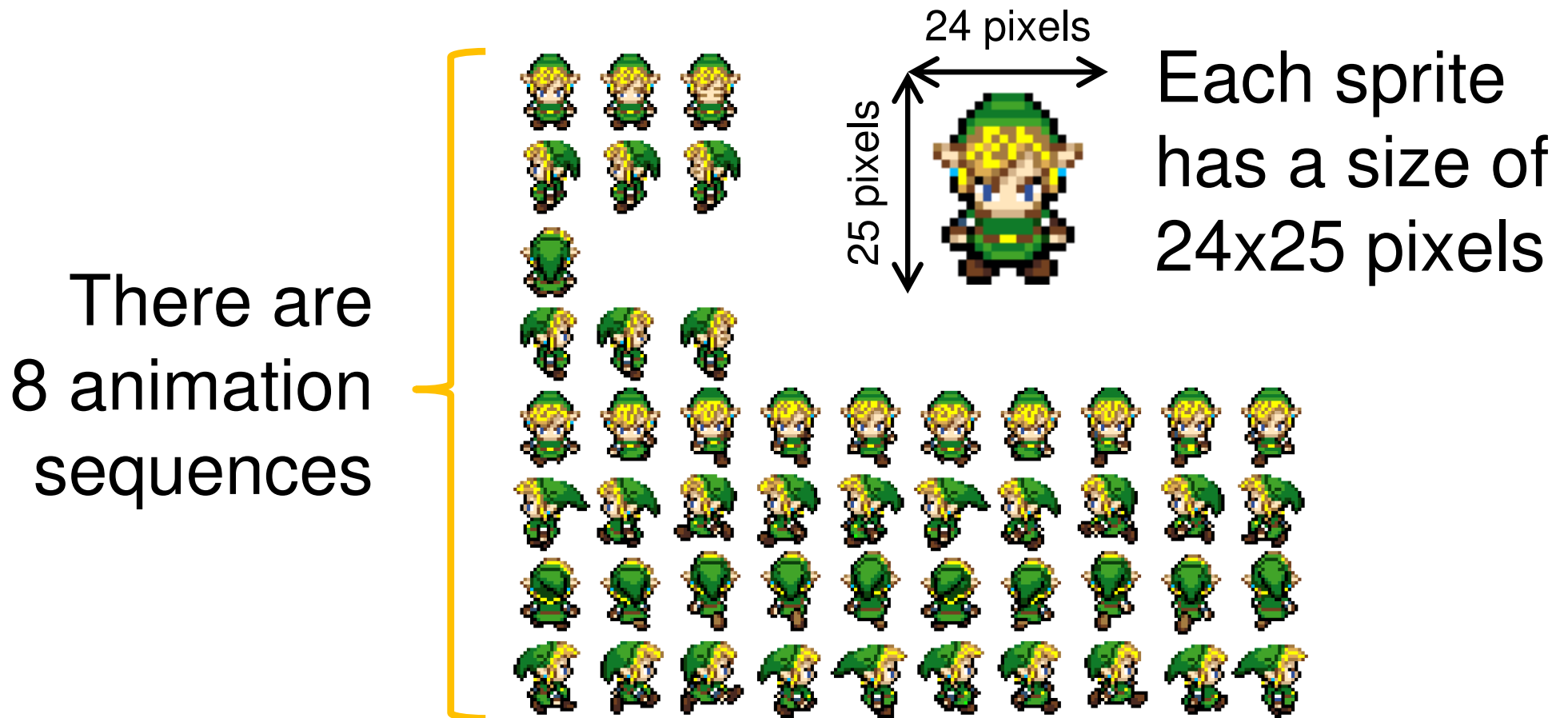
# An Example Sprite Sheet

- Here is an example sprite sheet we use for the rest of the discussion



# The Sprite Sheet Sequences

- For the example sprite sheet:



# Making an Animation

- Let's make an animation of the following sequence of sprites in canvas:



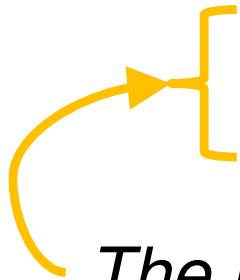
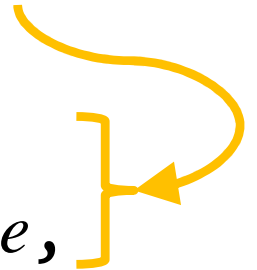
- There are 10 sprites in the above animation sequence but you only need to draw one of them at any time
- The `drawImage()` function allows you to draw only from a small area of the sprite sheet

# drawImage() Parameters

- The drawImage() function has 9 parameters, as shown below:

```
drawImage(  
    image,  
    x in source image, y in source image,  
    width in source image, height in source image,  
    x in canvas, y in canvas,  
    width in canvas, height in canvas);
```

*The area in the  
source image*



*The location and size to put the image in the canvas area*

- They tell the function where to extract from the source image and where to put the content in the canvas area



# Drawing a Sprite



*Each sprite has a size of 24x25*

- For example, to draw the 4th sprite from the above example at (50, 50) in canvas, you can use the following code:

*Parameters  
for the image  
in the canvas  
area*

```
context.drawImage(image,  
    24*3, 100, 24, 25,  
    50, 50, 24, 25);
```

*Parameters  
for the sprite  
in the sprite  
sheet*

# Working With Animation

- You can create an animation by drawing different sprite images at around the same place over time
- To do that, you need a way to continuously update the canvas area
- JavaScript provides a useful function for you to work with animation:

`requestAnimationFrame(...function name...)`

# Request Animation Frame

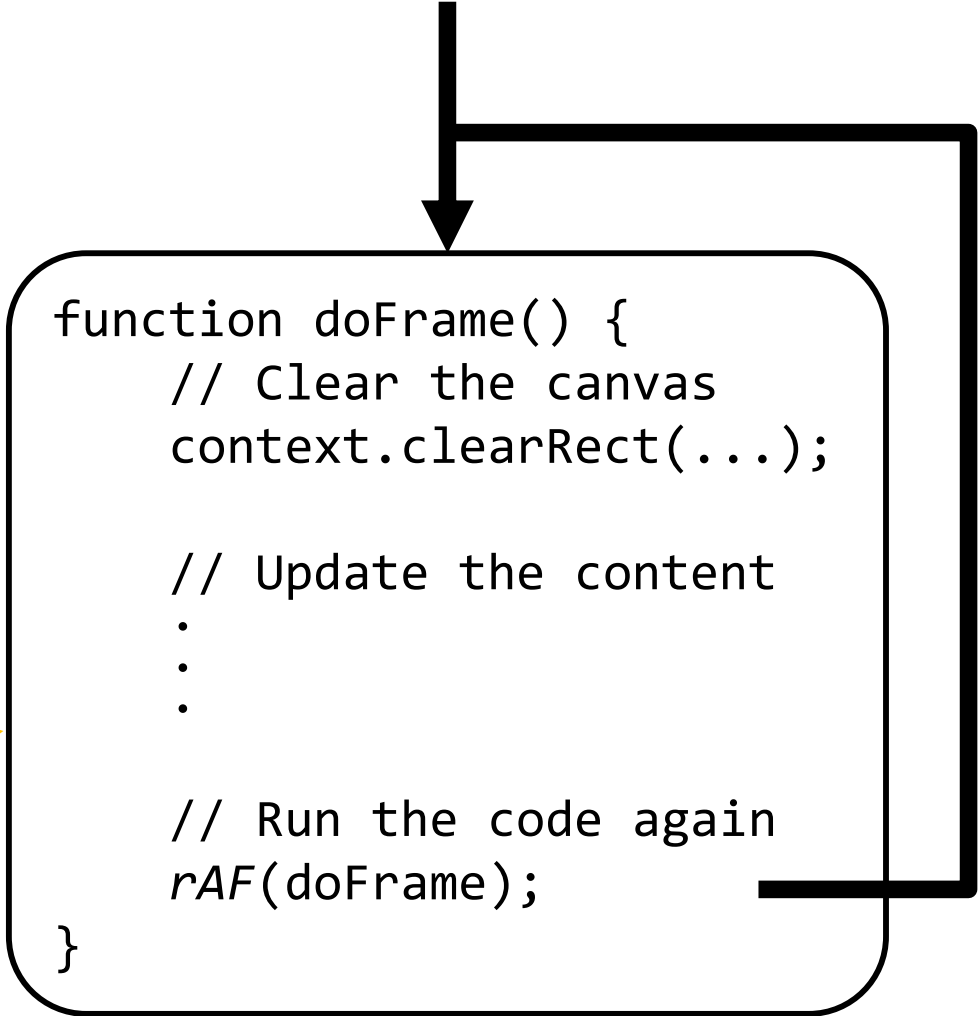
- `requestAnimationFrame()` works in a similar way to `setTimeout()`  
*(As the function name is very long, we will sometimes write `rAF()` to save space)*
- You use `rAF()` to run some code, not immediately but after a certain amount of time
- However, unlike `setTimeout()`, you do not control the timing to run the code
- `rAF()` makes sure your code runs in 60 fps (frames per second), i.e. your code runs approximately 60 times a second

# Canvas Animation Loop

- Let's assume that you have written a function `doFrame()` to do something for one 'frame' of an animation
- An 'animation loop' can then be created like this:

*This function runs  
60 times a second*

`rAF(doFrame);`

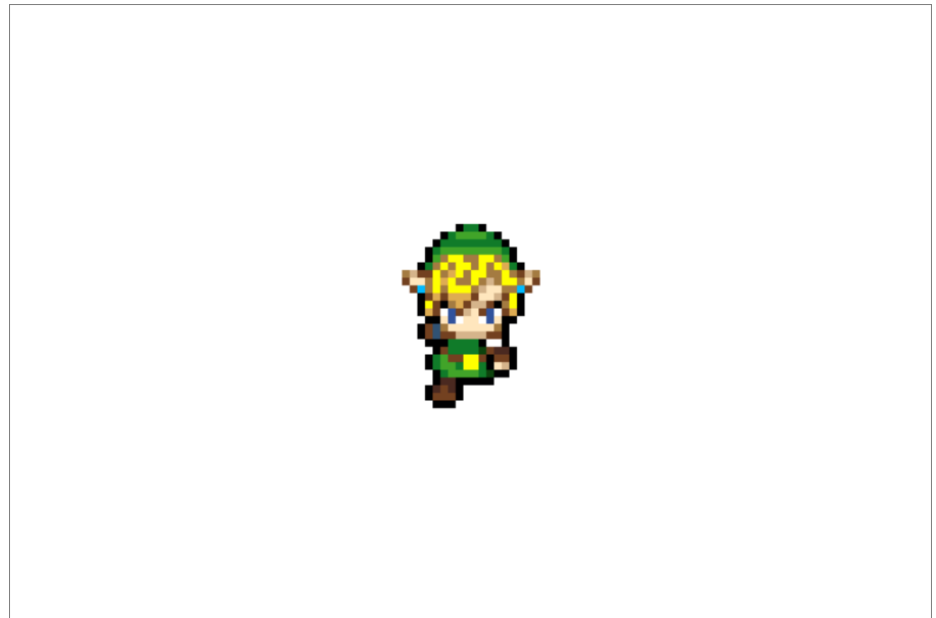


```
function doFrame() {  
    // Clear the canvas  
    context.clearRect(...);  
  
    // Update the content  
    :  
    .  
  
    // Run the code again  
    rAF(doFrame);  
}
```

The diagram illustrates the animation loop. A thick black arrow points from the `rAF(doFrame);` call at the top to the `doFrame` function block. Inside the function block, a curved arrow points from the `rAF(doFrame);` call back to the top of the function block, indicating a recursive call. A yellow curved arrow points from the text 'This function runs 60 times a second' to the `doFrame` function block.

# Making a Sprite Animation

- In the function `doFrame()` we draw one of the sprites in the animation sequence and then moves on to the next sprite the next time the function is run
- The code is described on the next few slides




# The Example Code 1/5

- First, the code loads the sprite sheet image
- Once the image is loaded, the code starts the animation loop using *rAF()* :

```
const sheet = new Image();  
sheet.onload = function() {  
    requestAnimationFrame(doFrame);  
};  
sheet.src = "sprite_sheet.png";
```

*The animation loop starts here*



# The Example Code 2/5


- Then a JavaScript object is set up to store the information of the sprite sequence:

```
const sequence = {  
  x: 0, y: 100, width: 24, height: 25,  
  targetX: 240, targetY: 135,  
  targetWidth: 120, targetHeight: 125,  
  index: 0, count: 10  
};
```

*Sprite position and size*

*Canvas image position and size*

*Current sprite and the total number of sprites*



0 1 2 ...

# The Example Code 3/5

- Inside `doFrame()`, the code first clear the area containing the sprite:

```
function doFrame() {  
    context.clearRect(  
        sequence.targetX,  
        sequence.targetY,  
        sequence.targetWidth,  
        sequence.targetHeight);  
    ...  
}
```

*This only  
clears the  
area with  
the sprite*



# The Example Code 4/5

- Afterwards, the current sprite is drawn at the same position:

```
context.imageSmoothingEnabled = false;
```

```
context.drawImage(
```

```
    sheet,
```

*Sprite image information*

```
    sequence.x +
```

```
    sequence.index * sequence.width,
```

```
    sequence.y,
```

```
    sequence.width, sequence.height,
```

*Canvas  
image  
information*

```
    sequence.targetX, sequence.targetY,
```

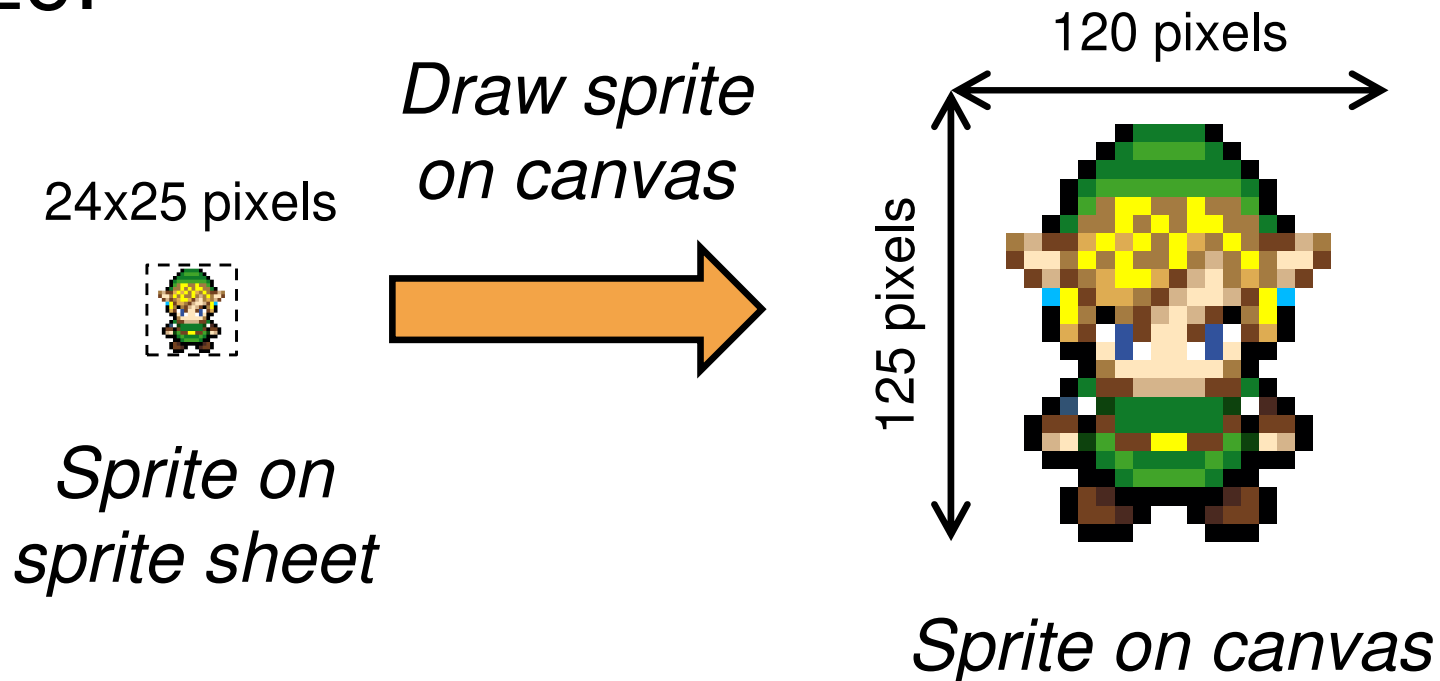
```
    sequence.targetWidth, sequence.targetHeight
```

```
);
```

*Draw the  
image  
without  
blurring*

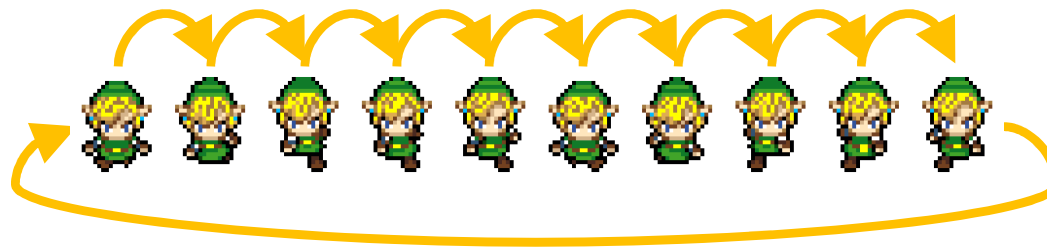
# Sprite Size on Canvas

- Because the sprite size is 24x25 pixels and the size on canvas is 120x125 pixels, the sprites are drawn 5 times their original size:



# The Example Code 5/5

- Finally, the code moves to the next sprite in the sequence and runs the function again:



*Update the index  
(current sprite) in  
the sequence*

...

```
sequence.index++;  
if (sequence.index >= sequence.count)  
    sequence.index = 0;
```

```
requestAnimationFrame(doFrame);
```

```
}
```

# Animation Timing

- If you run the example in the previous slides, the character will be animated too fast
- The 'walking' animation of the character should be much slower
- You need to write your own code to maintain a different animation timing than the 60 fps that *rAF()* gives you

# The Timestamp Parameter

- To control your own timing, you can use the timestamp parameter passed to you by *rAF()*, i.e.:

```
requestAnimationFrame(doFrame);
```

```
function doFrame(timestamp) {  
    ...  
}
```



*A timestamp  
parameter containing  
the time information*

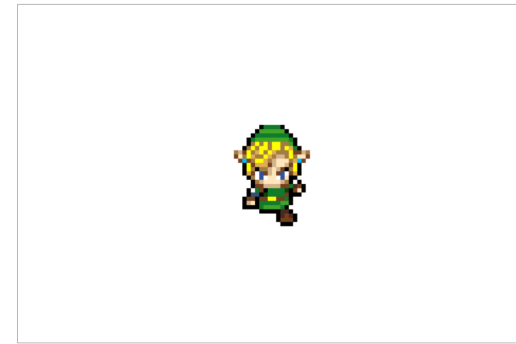
- This parameter gives you the amount of time passed in milliseconds since the page is loaded

# Using the Timestamp

- You use the `timestamp` parameter to determine the timing between frames
- You can also compare it against the value returned by `performance.now()`
  - It gives you the current time in the same unit as the `timestamp` parameter
- In the next example, the sprite animation has an adjustable timing by comparing the timestamps against the timing between two sprites

# Comparing the Timing

- In this example, the sprite is drawn only if the time difference between frames fits the timing of the sprites



Sprite Timing:  
100 milliseconds

```
let last = performance.now();
```

```
function doFrame(now) {
```

```
    const timing = $("#timing").val();
```

```
    if (now - last >= timing) {
```

```
        ...draw the sprite...
```

```
        last = now
```

```
    }
```

```
    ...
```

```
}
```

*Time  
difference  
so far*

Sprite Timing:



100 milliseconds

*Get the  
timing  
from a  
slider*

# More Sprites

- The example sprite sheet contains 8 animation sequences
- Code can be set up so that different sequences are shown at the same time, using different timings
- You can also use multiple sprite sheets to create more characters / objects



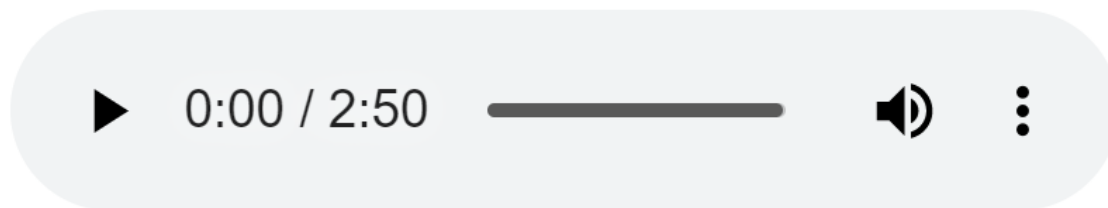


# Putting Audio on Web Pages

- You can use the `<audio>` element to put an audio file on a web page, for example:

*Showing the audio controls*

```
<audio controls src="zelda_main.mp3">  
</audio>
```



*The audio file*

# Using the Audio Object

- However, just like drawing images using canvas, we want to control the audio playback by JavaScript
- That means you would want to use the audio object to load an audio file in your script, i.e:

```
const song = new Audio("zelda_main.mp3");
```


- The audio is loaded into the memory instead of the DOM

# Playing and Stopping Audio

- After loading an audio object, you can play the audio using `.play()` and pause it using `.pause()`
- If you want to restart the audio after pausing it, you will need to set the playback time back to 0, i.e.:

```
song.currentTime = 0;
```

```
song.play();
```



*This plays the audio from the start*

# Issue About Playing Audio

- Chrome (and some other browsers) does not allow web pages to play audio right after the pages are loaded
- That means you would not be able to ‘auto play’ some background music for a game
- You will need to start playing audio inside some user-initiated events, i.e. onclick