Animation using HTML 5, CSS & JS

Aim: Learn the basics of adding animation to your web pages using HTML 5, Cascading Style Sheets (CSS) and JavaScript (JS).

Register your attendance

Please make sure we register your attendance.

Animation with HTML 5, CSS and JS

Cascading Style Sheets (CSS) and JavaScript (JS) provide the means of adding sophisticated animations to web pages. You have already used CSS **transitions** in your multimedia application. Now you will try using CSS **keyframe** animation and HTML **Canvas** animation using timed functions in JS.

In this practical you will add two new screens (pages) to the web application that you began building earlier in the multimedia practicals. Again, complete instructions are not given and you will need to create your own HTML5+CSS+JS code to implement what is asked for. So you may need to consult learning resources as you proceed. Some suitable resources are listed below.

Resources

To learn more and to complete this practical you should consult the ***W3Schools*** tutorials and references on HTML, CSS and JS at **http://www.w3schools.com/**

Preparatory work

1. Open the folder (eg **WebMM)** that you were working in for the previous multimedia practicals.
2. To see a *finished* version, to give you an idea of what you are aiming for, open **MMfinal.html** in your preferred browser and try out the two new buttons, "Keyframe" and "JS Anim".
    a. On the Keyframe screen, click on the gray "bullet" to the left of the cowboy.
3. You might like to leave this file open in your web browser so you can refer back to it as you progress with building your own application.
4. Now open the file that you were working on earlier, called **MMapp.html,** in your browser for viewing and in your favourite IDE for editing.

CSS Keyframe Animation

Firstly, we are going to explore *keyframe animations* in CSS, which are a powerful way of adding dynamic behaviour to a web page, in addition to the *transition* used on the Home page.

5. In your app in your IDE, create a new division, "**scene4**", with associated menu button, labelled "Keyframe". As initial content, add a suitable level 2 heading (<h2>), and display the cowboy image (Media/cowboy.png; width 272 pixels, height 301 pixels) in a paragraph with centred text (using CSS; this will centre the image in the content panel).
6. Our bullet is created simply using an empty division ( ). Below the paragraph containing the cowboy image, create an empty with the ID "bullet". If you look back towards the

top of your app file you will see that there is a style definition "#bullet" that defines both the appearance and position of our bullet (this style is automatically applied to your new provided it has the ID of "bullet"). Note the rounded right-hand end of the bullet, obtained by specifying the border radius. Try your scene in the browser to make sure everything displays correctly.

7. Now we will add a keyframe animation to our bullet to make it "fire". In your IDE, in the <style> sheet, just above the style definition, "#bullet", add the following CSS keyframe animation definition:

```
@keyframes animbullet {
0% {left: 120px; background-color:  silver;}
100% {left: 270px; background-color:  silver;}
}
```

8. This animation defines the starting position and colour (identical to that of the #bullet style) and the final position and colour of our bullet. It is possible to add as many keyframes (each positioned between 0 and 100%) and as many style specifications for each keyframe as you want. Later you will add further keyframes to make the bullet change shape and colour as it hits the cowboy!
9. To apply this animation to our bullet, add the following rules to the style definition "#bullet" to specify that the keyframe animation called "animbullet" should be applied to our bullet over a duration of 2 seconds:

```
animation-name: animbullet;
animation-duration: 2s;
animation-fill-mode: forwards;
animation-play-state: paused;
```

10. Currently the animation is paused so that it does not actually run (if you now try your app in the browser you will see that nothing has changed and the bullet stays where it is.) To allow your user to "fire" the bullet by clicking on it add the following:
    a. In a new Javascript <script> block below the end of your "scene4" , create a Javascript function called "shootBullet()" that contains the single line of code:

```
document.getElementById("bullet").style.animationPlayState = "running";
```

    b. Add the attribute, "onclick=shootBullet()" to the bullet ; this tells the browser to execute the JS function "shootBullet()"when the is clicked with the mouse.
    c. Rerun your app in the browser and "fire" your bullet: it should move to the right and finish somewhere over the middle of the cowboy.
11. To be able to refire the bullet every time you visit the "Keyframe" scene, add the following as the first line of code of the "clickBtn(btn)" function at the top of your file:

```
document.getElementById("bullet").style.animationPlayState = "paused";
```

12. Now change the background colour to red in the final (100%) keyframe, so that you have:

```
100% {left: 270px; background-color: red;}
```

13. Refresh your app in the browser and rerun the animation. Now the bullet gradually changes from silver to red over the full duration of the animation. Let's make the bullet only change to red when it reaches the cowboy: insert this line above the 100% line to define a new keyframe between 0% and 100%:

```
90% {left: 270px; background-color: silver;}
```

14. Again, save, refresh and rerun the animation to see what happens now.
15. Now your challenge is to make the bullet morph into a big red blood spot, as in the final version (you might like to replay that now to see what it looks like). The final version changes the width, height and border radius of the bullet at the end of the animation. So try doing this. You do not need to achieve something identical to the final version, so be creative. Add keyframes as you need them. You can also alter the speed of the animation by changing the duration in the "#bullet" style definition.
16. Finally, let's add some sound effects. Add the following line just below your "bullet" :

```
<audio id="gunshot" src="gunshot.wav"></audio>
```

17. To make this sound clip of a gunshot play each time you fire the bullet, add the following line to the "shootBullet()" JS function:

```
document.getElementById("gunshot").play();
```

18. If you have your own headphones with you then plug them into the computer and rerun the animation (unfortunately, without headphones, you will just have to imagine the gunshot).

Javascript time-based animation

In this exercise you will create an animation by using Javascript to draw on an HTML canvas. We will make use of the "setInterval()" function to redraw the canvas at regular intervals, with some drawn objects changing position each time, creating the animation. This is a very flexible technique that is widely used, particularly in game engines (as are explored in the games module, CSCU9N6). You may not finish this in this practical session, so work on it in your own time and seek help when you need it from the module instructors.

19. Create a new "**scene5**" division in your app, with an associated menu button, labelled "JS Anim". For content, add a suitable heading, followed by an HTML canvas of size 400 by 280 pixels and with an ID="myCanvas2" (to be different from the canvas created for scene3). You might like to give this canvas a border (using CSS) so you can see its edges.
20. Following the recipe used in scene3, create a new <script> below this scene and define variables that refer to the canvas (e.g. c2) and the drawing context (e.g. ctx2).
21. Add two further variables that specify the x and y coordinates of our "ball" object that we are going to animate, giving both variables the initial value of 50.
22. Below these add the following function definition for drawing a circle ("ball"):

```
function drawBall(x,y) {
```

```
ctx2.fillStyle = "black";
ctx2.beginPath();
ctx2.arc(x, y, 20, 0, Math.PI*2, true);
ctx2.fill();
}
```

23. Below this, create a function called “animate()” with no parameters. Give this function a single line of code that calls drawBall() with your x and y position variables as parameters.
24. Save all this and try it out in your browser. Hopefully you will see a black ball drawn on your canvas. But so far it does not move…
25. Add three more variables to your script, one to define the speed of your ball (give it a value of 3) and two to define the direction of travel in the x and y directions, respectively (give both these a value of 1).
26. Before your call to drawBall() in the function animate(), add two lines of code that make use of your new variables to adjust the position of the ball according to its speed and direction of travel: depending on your variable names these lines will be something like:

```
xpos+=speed*dirX;
ypos+=speed*dirY;
```

27. Now to create the animation, we need to call the animate() function at a regular interval, so just below your variable declarations add the following line:

```
setInterval(animate,20);
```

28. This will call animate() every 20 milliseconds (or 50 times a second), which is a reasonable rate for smooth animation. Save all this and give it a try in your browser. What happens? Have we forgotten something?
29. Actually, we have forgotten two things. Firstly, we have to erase the ball before we redraw it in a new position. The easiest way to do this is to simply clear the canvas completely before drawing the ball. So add this as the first line of animate() and give it a try:

```
ctx2.clearRect(0, 0, c2.width, c2.height);
```

30. But where has my ball gone? It moved and just kept on going… So the remaining challenge is to add further lines of code to animate() that change the direction of travel of the ball whenever it reaches an edge of the canvas so that the ball appears to bounce off the edge and remains within the canvas (as in MMfinal.html). You will need to change the direction of travel in the x or y directions from +1 to -1 and back again, as needed. Good luck!