Client-Side Web Development

Assignment 2 - Part 1: JavaScript
2018

You must implement a fruit machine game in HTML, CSS and JavaScript.

Clicking on the gears icon should cause the settings menu to appear over the page.

You should be able to select the colour scheme you want for the page by selecting the option.

You should create two styles for the page for this purpose.

The close box should hide the settings panel.

Click on the handle and the fruits scroll past in their windows (see screencast).

After a second the first window stops scrolling.

After another second the second window stops scrolling.

Finally, after second, the final window stops scrolling.

If at least 2 of the fruits are the same then you win and you cover the page/viewport with an animated gif background of falling confetti.

This confetti should appear to fall down over the page (see screencast).

You can animate a change in the height of the element with the confetti background. (This confetti element should be hidden again after a short delay, e.g. 3 seconds).

The handle can be made up of divs (or similar elements) sized and shaped (via border-radius, etc.) to look like the components of the handle.

The handle can be made up of divs (or similar elements) sized and shaped (via border-radius, etc.) to look like the components of the handle.

It can be positioned to the side of the fruit machine using **absolute** positioning.

When you click on the handle the handle seems to depress and then return to its original height.

This can be done by simply changing the height of the "stick" element.

You are provided a transparent sprite for the fruits.

Note that each fruit is a square. If you make the background image 100px across (e.g. make it 100% the width of a 100px wide element) then it is 500px tall.

To implement the scrolling fruit we set the background to repeat on the y-axis and change the position of the background image.

Once we know what fruit was chosen for each box we must move the background by an appropriate amount.

Once we know what fruit was chosen for each box we must move the background by an appropriate amount.

If you set the sprite background image to be **x** pixels across then it will be **5 * x** pixels tall.

Here we set up the sprite as the background to the fruit window (in red here).

If you want to show the next fruit move the background vertically by **-x** pixels.

Before

To show the third fruit we move by **-2 * x** pixels.

After

To show the third fruit we move by **-2 * x** pixels.

-4x

I.e. to show fruit **z** we move the background by **-(z-1) * x** pixels.

E.g. assume **x** is 100 pixels.

1st fruit (z = 1) = -(1-1) * 100px = **-0px**

2nd fruit (z = 2) = -(2-1) * 100px = **-100px**

3rd fruit (z = 3) = -(3-1) * 100px = **-200px**

etc.

Since the background repeats we can keep moving it upward and there will always be fruit to show.

Since the background repeats we can keep moving it upward and there will always be fruit to show.

Since the height of the sprite is **5 * x** pixels then moving the sprite up by **5 * x** pixels has no effect on the graphic being shown since it wraps around to the same position in the sprite.

Similarly, moving

-3 * x

is the same as

(-5 * x) + (-3 * x)

(e.g. move it up by the height of the sprite - which leaves it showing the same fruit - plus an extra 3 * x)

Similarly, moving

-3x

is the same as

(-5x X 4) + (-3x)

(e.g. move it up 4 times the height of the sprite - which leaves it showing the same fruit - plus an extra 3 * x)

Why move it by multiples of the height if it shows the same fruit?

Because we can animate that movement to give the effect of scrolling fruit (see screencast).

Note that ideally you will be moving the background from wherever it stopped, not from the the top.

Here we want to show fruit 1 (this is the **target**)

E.g. We move by

-(**target**-1) * **x** px

-(1-1) * **x** px

-0px

To select fruit 2 we move by

-(target-1) * x pixels

-(2-1) * x pixels

-1 * x pixels

-x pixels

Now to select fruit 3 we only move a further -**x** since we moved **-x** pixels already.

I.e. Assume the number of the currently displayed fruit is stored in **current**.

To move to fruit 3 (**target = 3**) from fruit 2 (**current = 2**) we move by:

-(**target**-**current**) * **x** pixels.

-(3-2) * **x** pixels

-1 * **x** pixels

-**x** pixels

Assume the current background position of the sprite is stored in **bgPos**

Assume the current fruit you are displaying (as a number) is stored in **current**.

Also assume the number of the fruit you want to show is in **target**

Finally, assume the height of a fruit is **100px** (and the height of the sprite is 500px - this may be different for your design).

We can calculate the background position as follows:

```
bgPos = bgPos - 500+100*(target-current);
```

This amount simply wraps the background to the same fruit. But its purpose is to add extra distance to animate.

```
bgPos = bgPos - 500+100*(target-current);
```

This is the height of a single fruit

```
bgPos = bgPos - 500+100*(target-current);
```

We multiply that by the numerical offset (as discussed)

```
bgPos = bgPos - 500+100*(target-current);
```

We subtract this value from the stored background position of the element.

```
bgPos = bgPos - 500+100*(target-current);
```

```
document.getElementById("fruitwindow1").style.backgroundPositionY = bgPos+"px";
```

Don't forget to keep the equivalent of the **target** and **current** variables up-to-date each time you show a new fruit.

E.g.

```
current = target;
```

```
document.getElementById("fruitwindow1").style.backgroundPositionY = bgPos+"px";
```

To generate a random number between 1 and 5 (inclusive) the following code will suffice:

```
var target = Math.floor(Math.random()*5+1);
```

The amount you move the 2nd fruit should be ***roughly*** twice the distance as the first fruit since we will animate it for twice as long. The third fruit is roughly 3 times as long (not counting the offsets for the individual fruits).

```
bgPos1 -= 500+100*(target1-current1);
bgPos2 -= 1000+100*(target2-current2);
bgPos3 -= 1500+100*(target3-current3);
```

The amount you move the 2nd fruit should be ***roughly*** twice the distance as the first fruit since we will animate it for twice as long. The third fruit is roughly 3 times as long (not counting the offsets for the individual fruits).

```
bgPos1 -= 500+100*(target1-current1);
bgPos2 -= 1000+100*(target2-current2);
bgPos3 -= 1500+100*(target3-current3);
```

The transition animation (in the CSS) for each fruit should then be of different & proportionate durations (e.g. 1 second, 2 seconds,and 3 seconds).

Since the last fruit is displayed 3 seconds after you click the handle you must make sure the confetti (if it is required) doesn't appear until after that 3 seconds (e.g. you can use **setTimeout()**).

Client-Side Web Development

Assignment 2 - Part 2: CSS
2018

You must create the responsive page shown here.

The content is contained in a centred section in the desktop version, but edge to edge in the mobile version.

Header

Mobile

Desktop

The header consists of the page title and a menu of 3 items.

On the desktop the title and menu are vertically aligned as shown.

The menu and title are at either end of their centred container.

Note the boxes in the diagram above represents how the elements look visually. Not necessarily how they are implemented.

On the mobile version the menu appears **above** the title (edge to edge) with a different background colour. And they are distributed evenly across the screen. The title is centred.

Calls to Action

Desktop

Mobile

The calls to action are made up of three parts.

Action Description

Image Section

Heading

Note that the description and heading have changed their order (heading comes first now) in the mobile version.

Action Description

? (Removed)

Image Section

Heading

In the mobile version there is a line after each Call to Action (Except the last).

The heading text is larger than the action text.

In the desktop version each Call to Action is the same height.

The titles and actions are whatever height they need to be (based on their content) and the picture takes up the remaining space.

The Calls to Action are the same width and distributed evenly across the container (going edge to edge within the container).

Regardless the size of the image area, as much of the image should be visible as possible (see screencast).

This can best be implemented by making the image a background image and using **background-size**.

Desktop

Mobile

Member Profile Pics

On the desktop the members' profile pictures appear in a centred container (with a border on top and bottom).

The pictures then automatically scroll through the container (see screen cast)

On mobile the images go from edge to edge (with no border on the container).

These are the same images as in the desktop version but with a less severe border-radius.

vs

Copyright

Place a centred copyright notice at the bottom of the page.

You can choose your colours and images for the page

You can also choose your web font(s)

You can get sample profile images from this site: https://uifaces.co

The breakpoint is up to you but can be around the 500px mark.

Technical Tips

The animation (for both mobile and desktop) goes as follows. An element containing the profile images is positioned just off to the side of the container.

You need to make sure the element containing the picture element is wide enough for the number of pictures you use.

If you are using **absolute** positioning, or **floating,** or setting its **display** to **inline-block** the element's width will shrink to fit its contents. In order for the element to be wide enough for all the profile pictures we must make sure they don't wrap.

This can be done by either using a flex box (i.e. create a row that doesn't wrap) or setting **white-space** to **nowrap** (if using inline-block on the profile pictures themselves).

We then animate the position of this element so it moves across the container.

Screen grab shows effect with **overflow: hidden** removed from container element.

Eventually all the images have moved through the container.

We then have the element with the profile pictures fade out (via opacity).

We then move the images back to their original position .....

.... and repeat the animation.

See the screencast for a better idea of how it works.

You can set the **overflow** property of the container to **hidden** so that you only see the profile pictures that appear inside the container.

You can move the element by changing one of the following (depending on how you displayed the images).

`margin-left`

`left` (with absolute positioning)

`translateX()`

We need to move the element containing the images to the left by the same amount as its own width.

When using **translateX** with a percentage value, the percentage value is relative to the element's own size and not its parent.

E.g. this code moves an element left by its own width.

```
transform: translateX(-100%);
```

You can animate the images fading away by changing the **opacity** property.

This animation, and the movement, can be carried out with key frame animation .

E.g. first move the pictures, then fade them out, then repeat the animation.

As long as the page works as specified here you can implement it any way you consider appropriate. However, it is particularly well suited to flex boxes.

Responsive
Search Google or type a URL
CIT Chat Forum
Register
Login
Messages
Register
Step 1
Login
Step 2
Leave Message for other users
Step 3
Latest Members

Flex box

(Row)

Flex box

(Column)

Flex box

(Column)

Responsive
Search Google or type a URL
CIT Chat Forum
Register
Login
Messages
Register
Step 1
Login
Step 2
Leave Message for other users
Step 3
Latest Members

Flex box

(Row)

Flex box

(Column)

Responsive
Search Google or type a URL
CIT Chat Forum
Register
Login
Messages
Register
Step 1
Login
Step 2
Leave Message for other users
Step 3
Latest Members

Flex box

(Row)

Latest Members

© 2018 CSWD

Flex box

(Row)

Submission

Note: Remaining **labs** will be dedicated to working on this assignment.

Attendance is still expected.

Lectures from this point out will continue to cover new material which may not necessarily be applicable to the assignment. however attendance is still required as tips and aspects of this assignment may be covered in classes.

The assignments (especially the CSS assignment) can be implemented with flex box. The screengrabs of the CSS assignment solution were implemented that way.

However, we will be covering Grids over the next few days. You may if you wish use them for your solution (or any alternative technique as long as it meets the assignment requirements).

See provided screencasts for more details on each part of the assignment.

Submit your code on Blackboard by the due date for the assignment.

All work must be your own (see Assignment Guideline Document).