Intro #2: Animation

Animation

So far, we've only set up static scenes. We've learnt how to position different kinds of objects on the screen, and change how they look; but we haven't moved them or changed them after initialisation.

Animation means to give an appearance of movement. We do this by **changing values over time**.

- First, let's start a **new project**.

  You can do this by saving your last file with a different name (game1.js would be sensible), then reopen game.js. If you do that you'll need to delete everything after `await uw.begin()`, except for the first three lines that create the world and give it a title and background.

- Then in this new file, make a Sprite with your favourite emoji. (You can search for emojis here: once you've found the one you want, click on it and it will be copied to your clipboard.)

  ```
  var face = new Sprite
  face.costume = '🐷'
  ```

👉 Save, and check your Sprite appears!

Forever

We need a way to react to time passing. We do this by using `forever` to run code on every *frame*.

An app draws a new frame on the screen roughly 60 times a second (or 60 **FPS**, frames per second).

We can use `forever` to make our sprite spin!

- Add a `forever` block, right after you create your `Sprite`.

```
face.forever(() => {

})
```

The `forever` blocks are fiddly to type; sorry about that. Make sure you get the brackets exactly right!

- Rotate your Sprite a small amount every frame.

```
face.angle += 1
```

*Add this **inside** the `forever` block.*

In JavaScript, `n += 1` is shorthand for `n = n + 1`. You can read it as **"change by"**.

We're increasing the angle of the sprite a little bit on every frame, so our sprite will slowly rotate clockwise.

👉 Save, and you should see your Sprite start to spin!

- **Challenge:** make the sprite rotate anticlockwise.

👉 Make sure you can get your Sprite to spin the other way.

Now we know the basics of animation, lets try some simple movement.

- **Instead** of rotating, make the Sprite move right.

```
face.angle += 1
```

Delete this line.

```
face.posX += 2
```

👉 Save, and the Sprite should move to the right instead of rotating.

The numbers inside the forever are how much to move on each frame, so they control **how fast** the animation happens.

- Make your sprite move a bit faster.

👉 Make sure you can get your Sprite to move faster.

- **Challenge**: experiment with animating other properties, like gradually increasing scale:

  ```
  face.scale *= 1.05
  ```

Note that multiplying the scale by the fraction `1.05` is the same as increasing it by 5%.

👉 See if you can get your Sprite to get bigger.

Events

Let's have our face shoot out a projectile toward our finger when we tap.

- Delete (or comment out) the `forever` block for now – we won't need it until later.

  ```
  face.forever(() => {
      face.angle += ...
      face.posX += ...
      face.scale += ...
  })
  ```

👉 Make sure your sprite stops moving.

- Use `on('tap')` to detect when the screen is tapped.

```
world.onTap(e => {
    alert("dont touch this")
})
```

Make sure you get the brackets right!

The code inside the `on(...` block runs whenever the screen is tapped. So whenever you tap (or click) the screen, the message "don't touch this" should appear.

👉 Save, and check that the message appears when you tap the screen.

That gets boring quickly, so let's make a projectile.

- When the screen is tapped, create a bullet under your finger.

```
    ~~alert("dont touch this")~~
```

Delete this.

```
    var bullet = new Sprite
    bullet.costume = '👾'
    bullet.posX = e.fingerX
    bullet.posY = e.fingerY
```

*Add this **inside** the `world.onTap` block.*

👉 Save, and check that a bullet appears wherever you tap.

What's going on here? `e` is an **event** object, telling us the details of the **tap** event. The `e.fingerX` and `e.fingerY` attributes tell us the X & Y coordinates where the tap happened.

Now let's try moving our projectile!

- Add a `forever` block to move the Sprite we created after the tap.

```
bullet.forever(() => {
    bullet.posX += 3
    bullet.posY += 3
})
```

*Make sure this is **inside** the `world.onTap` block still.*

Notice that the `forever` block has to be inside the `world.onTap` block. Just as we can't use a name like `cow` before it's been created, we can't use the name `bullet` **outside** the block that it was created inside. This phenomenon is called "scope".

👉 Save, and check that a bullet appears and starts moving, wherever you tap.

Moving at an angle

Sometimes we want to move things at an angle – and to do this, we need **trigonometry**!

It'd be really neat if our bullets started at the face, and travelled outward toward the point where we tapped. Then it would feel more like a "cannon" sort-of game.

- **Challenge**: make the bullets start at the same position as the `face`.

  Make sure you do this! If you don't, then the bullets won't move in the right direction later :-)

👉 Make sure the bullets always start from the face.

To move our bullets, we only need to know two things from trigonometry:

1 You can use `.posX += uw.sin(angle)` and `.posY += uw.cos(angle)` to move a sprite at an angle.

2 You can use `angle = uw.atan2(dx, dy)` to work out the angle you need to move something in a direction.

Currently, our bullets all go at 45°, which is dull. Let's fire them out at a random angle.

- **Challenge**: create the bullets with a random initial direction.

  Hint: you want to set their `angle` to a random number between `1` and `360`.

👉 Check the bullets are created pointing in a random direction.

- Now move them at that angle by replacing your `forever` block.

  ```
  bullet.posX += 3
  bullet.posY += 3
  ```

  We don't want this anymore.

  ```
  bullet.posX += 4 * uw.sin(bullet.angle)
  bullet.posY += 4 * uw.cos(bullet.angle)
  ```

  Type this instead, inside the `bullet.forever` block.

👉 Check the bullets move in the same direction as they're facing.

By multiplying `sin` and `cos` by 4, we increase the speed of movement.

So we've done the first part – moving the bullets in a direction.

Now to work out the correct angle! We need to use `atan2` for this. This is a special version of inverse *tan()* which takes two numbers – a difference in X and a difference in Y – and returns the correct angle.

We'll call the difference in X `dx`, and the difference in Y will be `dy`.

- Delete your code for setting the bullet to a random angle.

  ```
  bullet.angle = ...
  ```

  Delete your existing line of code for setting `bullet.angle`.

- Work out the difference in X and Y between the face and your finger.

```
var dx = e.fingerX - face.posX
var dy = e.fingerY - face.posY
```

- Use `atan2` to get the correct angle.

```
bullet.angle = uw.atan2(dx, dy)
```

👉 Check the bullets move toward your mouse.

Congrats! We've just made a… face-cannon… thing.

Conditions

Animation is all very well, and we've learnt how to react to *events*. But how can we react to other things changing?

A **condition** returns a Boolean value: either `true` or `false`. They look like this:

1 `x === y` means *x is equal to y*

2 `x !== y` means *x is not equal to y*

3 `x < y` means *x is less than y*

4 `x <= y` means *x is less than or equal to y*

5 `!p` means **not** – *`true` if p is `false`, and vice-versa*

6 `p && q` means **and** – *`true` only if both p and q are `true`*

7 `p || q` means **or** – *`true` if either of p or q is `true`*

Here are some examples. These are all `true`:

1 `1 !== 2`
2 `42 === 42`
3 `"moo" === "moo"`
4 `10 < 42`

```
5  !(false)
6  true && true
7  false || true
```

Finally, there's a really important condition built-in to you-win, called **.isTouching(otherSprite)**. More about that later!

Let's see what we can do with conditions.

- First, make our cannon-face move.

```
var xVel = 2

face.forever(() => {
    face.posX += xVel
})
```

Add this at the bottom of your program, just before the end `})`.

This time, we're giving its speed a name: `xVel`. We're doing this so we can change it later.

Just as you can give a name to Sprites, you can give a number a name to "remember" it for later. You can change it later, too.

Currently, our face moves to the right (increasing X).

- Make it bounce off the right edge.

```
if (world.width < face.right) {
    xVel = -2
}
```

Make sure the `if` *block is* **inside** *the* `face.forever` *block.*

We do this by comparing his `right` edge to the width of the world. If his right edge is greater than the width of the world, then he's gone off the right-hand side of the screen; so we set his velocity negative, so he starts moving left instead.

- Flip the costume, for good measure…

```
face.flipped = true
```

  *Add this **inside** the `if` block.*

- **Challenge:** make it bounce off the left edge too.

  You'll need to add a second `if` block inside the `face.forever`.

  Hint: the left-hand side of the world is where *X = 0*…

- Check that the bullets still come from the face, and head toward your finger!

Fin

Excellent work! You've learnt how to:

- Do something over time with `forever`
- **Change attributes** using +=
- *Move* things over time (**animation!**)
- React to the **orientation** of the phone
- How to **detect taps** using `.onTap`
- How to move things at an **angle**
- Using `if` for **conditions**
- `destroy`-ing Sprites when you're finished with them

What next?

If you want something else to try, here are some extensions to try! Why not:

- Carry on moving the bullets at an angle, but have their picture stay upright.

  Hint: you'll need a `var` statement for this–but make sure you get it in the right place!

Hint #2: the relevant phenomenon here is called "scope", and it's hard to understand, so ask someone to explain it to you!

- Generally have a play around with the thing you just made, or more generally, everything you've learnt in chapters 1 and 2.

- Continue on to the next thing once you're ready! 🙃