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, start a **new project**, and make a Sprite with your favourite emoji.

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

Forever

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

```
forever(() => {
    // do stuff
})
```

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.angle += 1
```

Delete this.

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

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

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

By increasing the angle of the sprite on every frame, our sprite will slowly rotate clockwise.

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

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

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

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

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

You can make the sprite move faster by increasing the number `2`, since the sprite will move further on each frame.

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

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

Orientation

We can read the phone's accelerometer to get the angle the phone's being held at, compared to gravity.

First, we need to initialise a `Phone` object, so that `you-win` knows to start reading the phone's orientation sensors.

- Put this near the top of your program, after the `import` lines.

  You might find a commented out-line already, in which case just uncomment it!

  ```
  // var phone = new Phone
  ```

  ```
  var phone = new Phone
  ```

- Rotate the sprite when the phone rotates.

  ```
  forever(() => {
      // whatever you had before
  })
  ```

  ```
  forever(() => {
      face.angle = phone.zAngle
  })
  ```

- **Challenge**: what happens if you use negative `zAngle`?

  (This might not make sense, depending on whether you're using an iPhone. THe idea is that the face always stays upright while you rotate the phone! Feel free to skip this if it doesn't make sense.)

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.

  ```
  forever(() => {
      face.angle = phone.zAngle
  })
  ```

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

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

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.

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

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

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

```
world.onTap(e => {
    var bullet = new Sprite
    bullet.costume = '👾'
    bullet.posX = e.fingerX
    bullet.posY = e.fingerY

    // TODO: move the bullet ...
})
```

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

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 coordinates of the tap.

Now let’s try moving our projectile!

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

```
world.onTap(e => {
    // ... [make the bullet] ...

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

Since the `forever` block is inside the `on`, it too runs on every `tap`; and the `bullet` variable that it refers to is the one we just created. (This phenomenon is called "scope"; more about that in Intro #4: Functions.

Trig

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 :-)

I'm not going to explain all of trig here, since that would be boring. For the purposes, we only need to know two things:

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`.

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

```
world.onTap(e => {
    // ... [make the bullet, with random angle] ...

    forever(() => {
        bullet.posX += 4 * UW.sin(bullet.angle)
        bullet.posY += 4 * UW.cos(bullet.angle)
    })
})
```

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.

- Use `atan2` to get the correct angle.

```
world.onTap(e => {
    var bullet = new Sprite
    // ... [go to face] ...
    bullet.angle = UW.atan2(e.fingerX - face.posX, e.fingerY -
face.posY)

    // ... [move at an angle] ...
})
```

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** is an expression that's 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 !== 2
42 === 42
"moo" === "moo"
10 < 42
!(false)
true && true
false || true
```

Finally, there's a really important condition built-in to moo. It's a function attached to a Sprite, and it's 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

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

This time, we're using a variable to store its speed. We'll need this to be a variable so we can change it later.

Just as you can store objects (such as `Sprites`) in a variable, you can store a number in a variable to "remember" it for later. You can change it later, too.

We're moving him to the right (increasing x).

- Now, let's have it bounce off the right edge.

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

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 = !face.flipped
```

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

  You'll need to add a second `if` block.

  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!

Finally, if we don't destroy the bullets, eventually the game will get really slow! Let's fix that, by destroying them once they're completely off the screen:

- Destroy the bullets once they're completely off-screen.

  Challenge: where should this go?

  ```
  if (!bullet.isOnScreen()) {
      bullet.destroy()
  }
  ```

  This should go inside the bullet's `forever`, just after the code to move it.

The `destroy()` function attached to a Sprite removes it from the screen permanently. This also stops any `forever` loops attached to it.

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.

  Come back when you're ready for the next thing 🙃