The Infinite Jumping Game

We're going to make a *Doodle Jump* clone!

The game is about bouncing on platforms to get higher and higher…forever. Your score is based on how high you can jump. You'll tilt the phone left and right to control the game.

Starting out

- Set the `width` and `height` of your world.

```
world.width = 300
world.height = 460
```

- **Challenge:** create a sprite for your player character. Call it `player`.

  Have a look back at chapter one, if you've forgotten how to create sprites!

  Make sure the player is just the right size on the screen of your phone: not too big, not too small.

- Set the `background` color of your world, to taste.

Gravity

The first thing to do is to make our player fall down.

We can simulate gravity using a **constant acceleration**. As you should know from Maths or Physics class, that means we need to keep track of **velocity** (speed and direction). Let's add a variable for just that.

- Add a variable called `velY`, which stores a number.

```
var velY = -2
```

- Add a `forever` loop for the player.

```
player.forever(() => { // player
    player.posY += velY
})
```

Save. You'll notice the player falls at a constant speed, which isn't right at all! Let's change velocity over time, to simulate gravity:

- Decrease velocity, so the player falls.

~~`var velY = -2`~~

```
var velY = 0

player.forever(() => { // player
    player.posY += velY
    velY -= 2               // <-- add this line
})
```

Platforms

Now our player falls down! Let's add some platforms for him to bounce on. (We won't do the bouncing part just yet; that'll have to wait for later.)

- Create a `platform`. This is going to be a `Polygon` instead of a `Sprite`.

```
var platform = new Polygon
platform.points = [[-30, 0], [30, 0], [30, 10], [-30, 10]]
platform.posY = 100
```

- Set the `fill` and `outline` colors for your Polygon, to taste.
- Tweak the size of your platform as necessary. You can do this by changing its `points`.

Save. Check the platform appears below the player!

Bouncing

Currently our player falls through the platform entirely. Let's sort that out.

We can use `isTouching` to check for collisions between different objects. `isTouching` returns a Boolean value (`true` or `false`), which we can use inside an `if` condition.

- Check if the platform is touching the player.

```
platform.forever(() => {
    if (player.isTouching(platform)) {
        // ...jump...
    }
})
```

- To jump, we want to **set** the player's Y velocity to be a large, positive number. Let's try `10`.

```
        velY = 10
```

Add this inside the `if` condition you just wrote.

Functions

A game with just one platform isn't very impressive, so we need to add some more. But we don't want to copy/paste the code every time we make a platform– that would be no good at all!

What we want to do is to move the code for making a platform into a **function**. A function (aka. procedure, aka. "custom block" from Scratch) lets you give a **name** to a piece of code, so you can refer to it by name.

- Move the platform code into a function, by adding `function makePlatform() {` at the beginning of that section, and `}` at the end.

```
function makePlatform() {
  //...[all your platform code]...
}
```

*Put the `function` bits **around** your existing code.*

You'll want to re-indent your code at this point–try selecting the code inside the function, and pressing the `Tab` key.

Save. You'll notice the platform disappears. This is because we haven't **called** our function; we've defined it, but not used it yet! Let's do that now.

- Make a platform by calling your new function.

```
makePlatform()
```

Add this at the end of your program, before the final `});`.

Now we can easily make more platforms!

- Add some more platforms.

```
makePlatform()
makePlatform()
makePlatform()
// ...
```

The platforms are all being made at the same height - we really want to make each platform higher than the last. Let's make a variable at the beginning of our program, to keep track of the height of the last one…

- Create a `lastY` variable.

```
var lastY = 100
```

Add this before the `makePlatform` function.

Make sure not to put this *inside* `makePlatform`, or you'll get a **different** version of the `lastY` variable each time. We need to remember it between calls to our function, so it has to go outside.

Now let's use it.

- Use `lastY` inside `makePlatform`.

```
    platform.posY = lastY
```

- **Challenge:** increase `lastY` every time you make a platform.

  *Make sure you do this **inside** `makePlatform`!*

Scrolling

Now we've got some platforms to climb, we need to add scrolling so we can move up the platforms.

In *Doodle Jump*, the screen only ever scrolls up; never down. This is because the way for the game to end, is for the player to fall off the bottom of the screen!

We can scroll by changing `world.scrollY`. The `.scrollX` and `.scrollY` values are an **offset** added to the position of everything in the world, so you can use them to scroll around.

We only want to scroll up, never down. We can do this by comparing the player's height, `player.posY`, with the current scroll position, `world.scrollY`.

- Scroll up when the player is near the top of the screen.

```
if (world.scrollY < player.posY - 300) {
    world.scrollY = player.posY - 300
}
```

You can tweak the number `300` to taste, to control how far up the screen the player has to go for the game to start scrolling.

More Platforms!

When we scroll up far enough, we run out of platforms.

We could keep adding more by copy/pasting the `makePlatform()` line, but that doesn't seem sensible; however many we'll have, we'll run out eventually! Let's delete all but one of our `makePlatform()` calls, and have our game make 'em automatically.

To do this, we need to compare the current scroll position, `world.scrollY`, with the Y position of the last platform we made: which we stored in the `lastY` variable.

- **Challenge:** if the last platform is below the *top* of the screen, make a new platform.

  *Write this inside the `forever` loop for the **player**.*

  Hint: you'll want to add the height of the screen to `world.scrollY`.

Bouncing, take 2

At this point, you might notice that when we touch a platform, it gives us a push upwards… even if we're already moving upwards! That's just plain wrong.

We could check whether we're touching the bottom or the top of the platform, but that sounds tricky.

Instead, let's check that we're moving downwards. We can do this by checking the `velY` variable, since that gives us our current speed and direction: it's positive when we're moving upward, and negative when we're moving downwards.

- **Challenge**: change the `if` so that we only bounce if both we're touching a platform **and** we're moving downwards.

  You learnt about conditions in chapter 2.

  *Make sure you get the brackets in the right place! The **condition** for an if always has to be written in brackets.*

```
if ( … && … ) {
```

Make sure the bouncing looks right now.

Movement

In *Doodle Jump*, the player moves when you tilt the phone.

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

Like last time, 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
```

Now we can use `phone.zAngle` to get the **angle** at which the phone is held. But to move the player horizontally, we need a **difference in X**.

Think back to the bullets from chapter two: to turn an angle into a difference in X, we use trigonometry: specifically, `UW.sin`.

- Move the player horizontally when the phone is tilted.

```
player.posX += 5 * UW.sin(phone.zAngle)   // adjust `5` to
taste
```

  Inside the `forever` for the player.

The number `5` is a multiplier controlling how **fast** the player moves when you tilt the phone. Adjust it until it feels right to you.

Wrapping

In *Doodle Jump*, the player can "wrap" from one side of the screen to the other: when you go off the right, you appear back on the left side, and vice-versa.

- Wrap when we go past the right-hand side of the screen.

  ```
  if (player.posX > world.width) {
    player.posX = 0
  }
  ```

  Inside the `forever` for the player.

- Challenge: wrap when you go past the left-hand side of the screen.

What next?

- Different kinds of platforms!

  Use `UW.randomChoice` to pick the kind of platform:

- Trampolines which bounce you higher.

  Bonus: an animation which rotates the player after he jumps off a trampoline!

- Springs which bounce you even higher.
- Platforms which move from side to side.
- Randomly vary the distance between different kinds of platforms.

  The springs can have a larger distance after them.

  You can increase the minimum distance as you get higher, so it gets more difficult the higher up you get.

- Add enemies which you have to avoid.

- Let the player shoot bullets at the enemies to kill them!