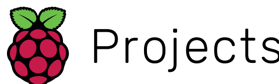
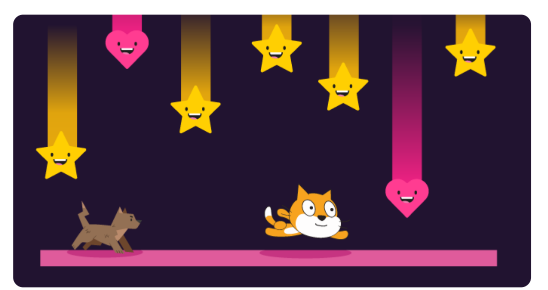


Falling stars

Make a platform-style game and add your own levels, powers, and characters.



Step 1 Introduction

These are the Advanced Scratch Sushi Cards, and with them you'll make a platform-style game to which you can later add your own levels, powers, and characters!

What you will make

Here is an example of the game you will build. Feel free to improve on my "art".

Use the arrow keys to move the cat around. Watch out for the dog!

What you will learn

- Cloning sprites
- Using variables inside cloned sprites
- Creating and using My blocks blocks
- Reusing code in several places with My blocks blocks
- Messages to trigger behaviours across sprites
- Using variables to configure your program

What you will need

Hardware

- A computer capable of running Scratch 3

Software

- Scratch 3 (either online (https://scratch.mit.edu/projects/editor/) or offline (https://scratch.mit.edu/download/))

Step 2 Setting things up

Because you're learning how to code in Scratch and not how to build a physics engine (code that makes things in a computer game behave like real-world objects, e.g. they don't fall through floors), you'll be starting with a project I've created that already has the basics for moving, jumping, and detecting platforms built in.

You should take a quick look at the project, including the details on this card, because you'll be making some changes to it later, but you don't need to understand everything it's doing!

Get the project

The first thing you'll need to do is to get a copy of the Scratch code from here (https://scratch.mit.edu/projects/454114430).

To use the project offline, download it by clicking See Inside, then go to the File menu and click Download to your computer. Then you can open the downloaded file in Scratch on your computer.

You can also use it directly in Scratch in your browser by just clicking See Inside and then Remix.

Take a look at the code

The physics engine of the game has a variety of pieces in it, some of which work already and some of which don't yet. You can test this out by running the game and trying to play it.

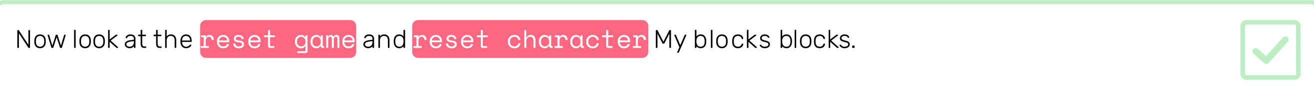
You'll see that you can lose lives, but nothing happens when you run out. Also, the game only has one level, one type of thing to collect, and no enemies. You're going to fix all of that, and a then do a bit more!

Take a look at how the code is put together.

It uses lots of My blocks blocks, which are great for splitting your code up into pieces so you can manage it better. A My blocks block is a block you make up out of a lot of other blocks, and you can give some instructions to it. You'll see how it works in an upcoming step!

'My blocks' blocks are really useful

In the code above, the main game `forever` loop calls the `main-physics` My blocks block to do a whole lot of stuff! Keeping the blocks separated like this makes it easy to read the main loop and understand what happens in the game, without worrying about how it happens.


Now look at the `reset game` and `reset character` My blocks blocks.

They do pretty normal things, such as setting up variables and making sure the character rotates properly

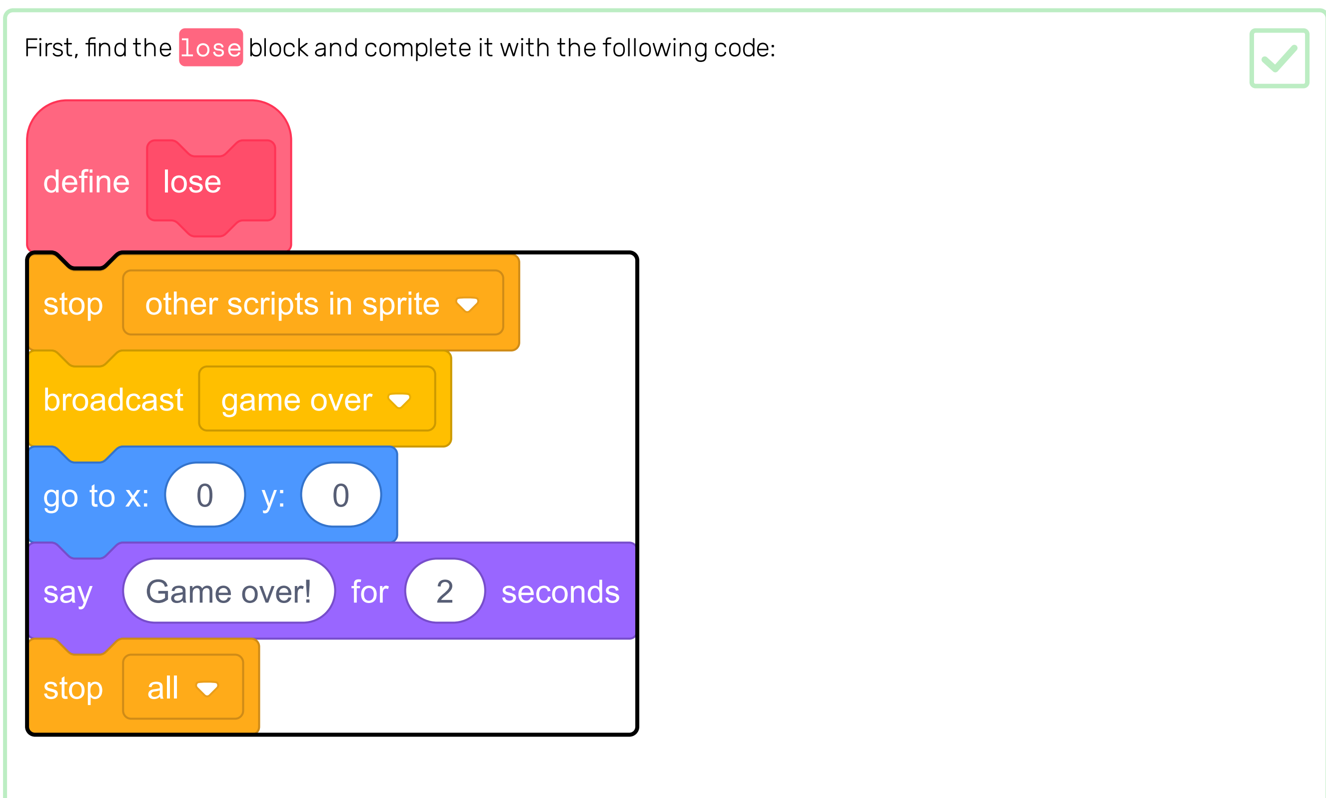
- `reset-game` calls `reset-character`, showing you that you can use a My blocks block inside another My blocks block
- The `reset-character` My blocks block gets used in two different places in the main loop. This means you can change two places in your main game loop by only changing the code inside of the My blocks block, which saves you a lot of work and helps you avoid mistakes.

Step 3 Losing the game

First things first! You need a way to make the game end when the player has run out of lives. At the moment that doesn't happen.

You may have noticed that the `lose` My blocks block in the scripts for the Player Character sprite is empty. You're going to fill this in and set up all the pieces needed for a nice 'Game over' screen.

First, find the `lose` block and complete it with the following code:



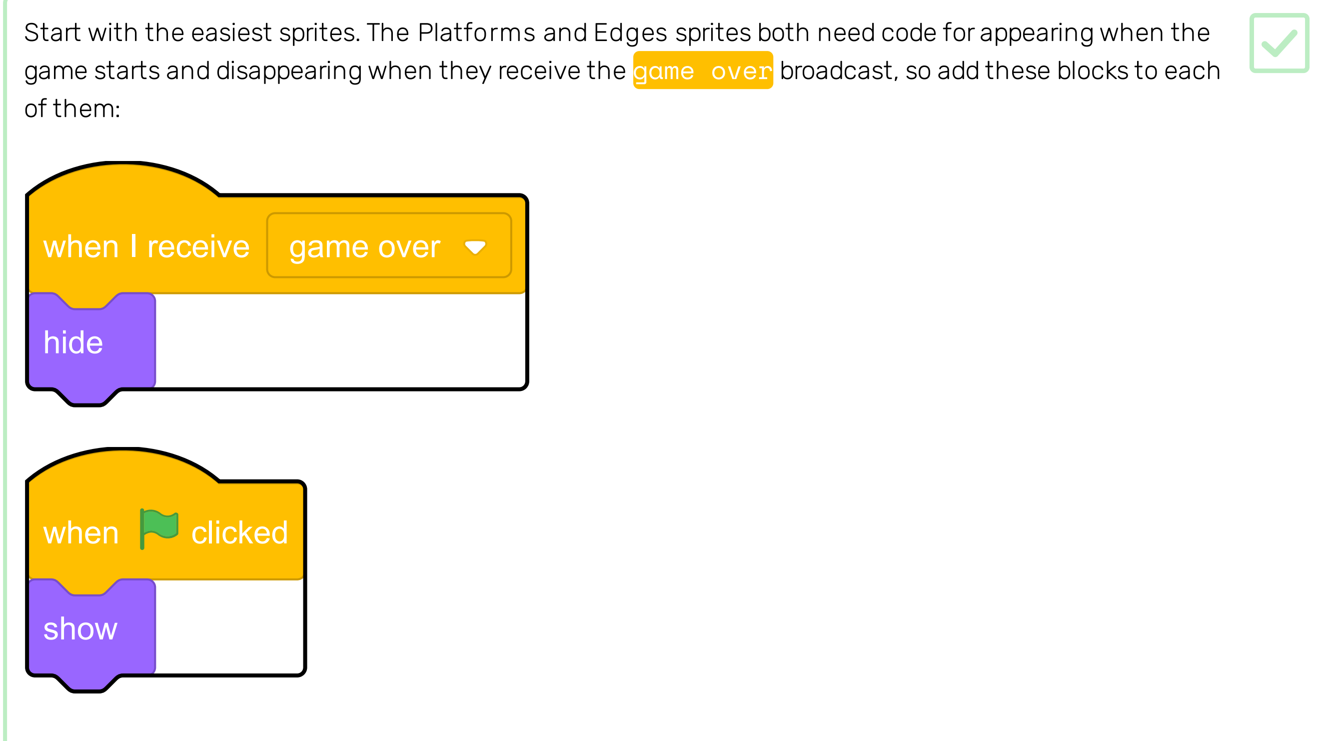
What does this code do?

Whenever the `lose` block runs now, what it does is:

1. Stop the physics and other game scripts acting on the Player Character
2. Tell all the other sprites that the game is over by broadcasting a `game over` message they can respond to and change what they're doing
3. Move the Player Character to the centre of the Stage and have them tell the player that the game is over
4. Stop all scripts in the game

Now you need to make sure all the sprites know what to do when the game is over, and how to reset themselves when the player starts a new game. Don't forget that any new sprites you add also might need code for this!

Hiding the platforms and edges

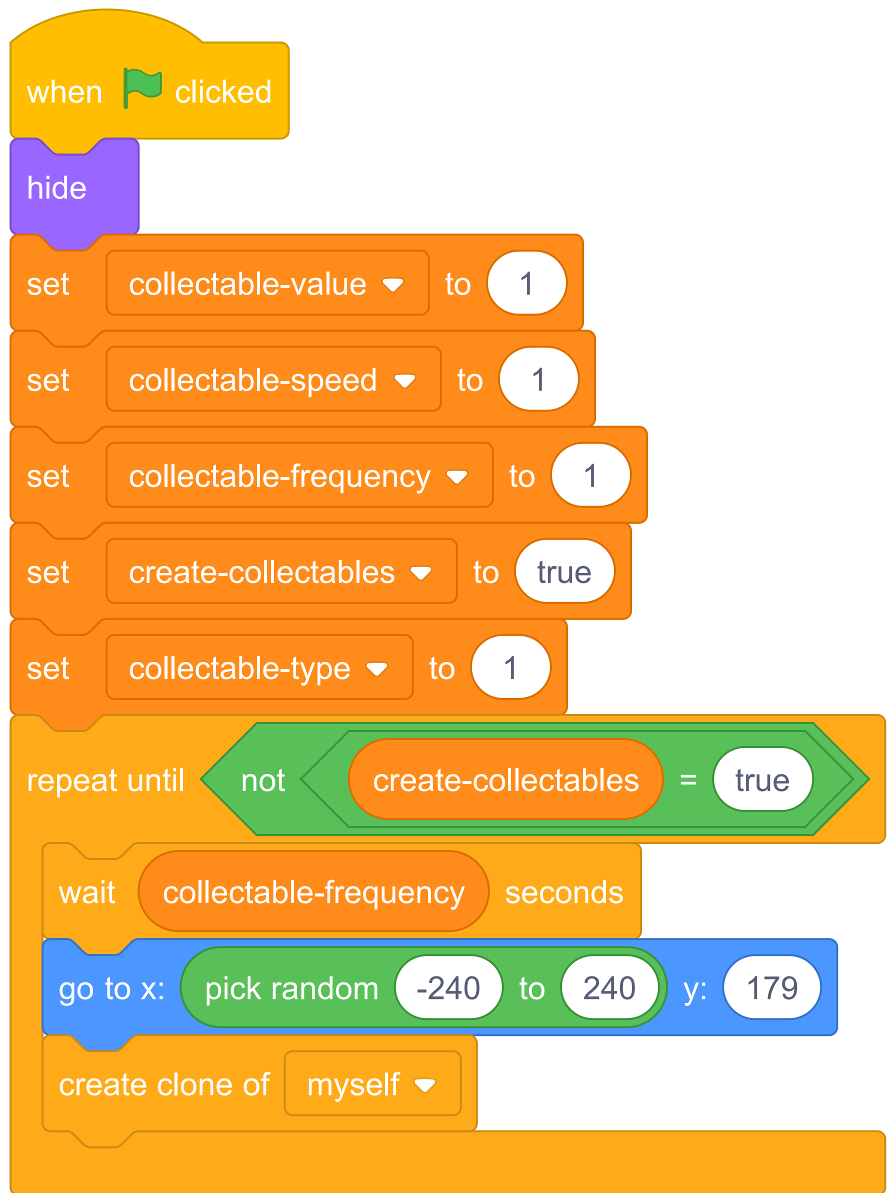


Stopping the stars

Now, if you look at the code for the Collectable sprite, you'll see it works by cloning itself. That is, it makes copies of itself that follow the special `when I start as a clone` instructions.

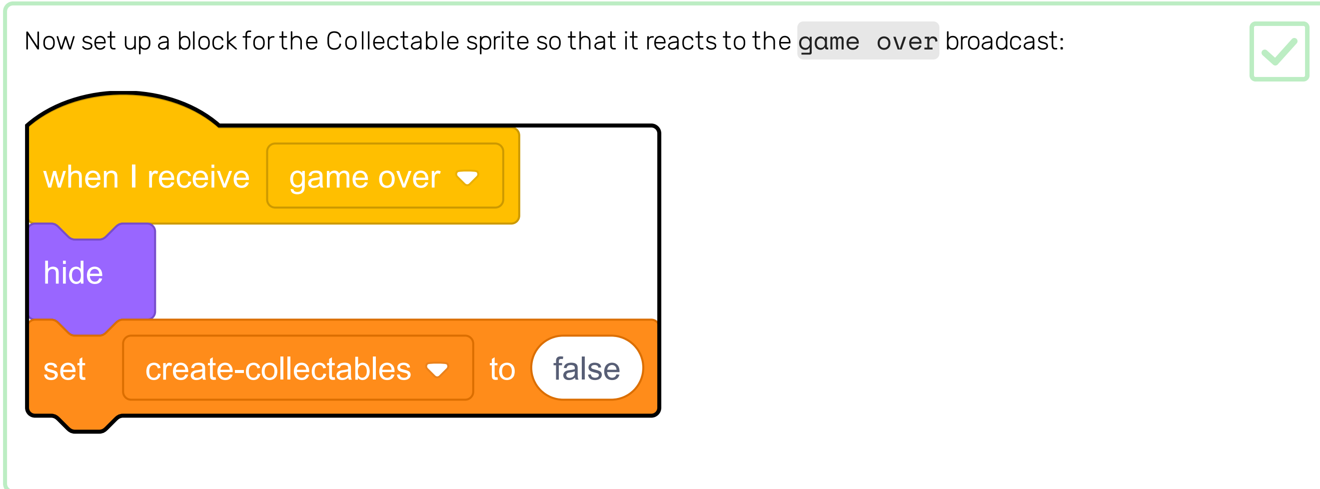
We'll talk more about what makes clones special when we get to the step about making new and different collectables. For now, what you need to know is that clones can do almost everything a normal sprite can, including receiving `broadcast` messages.

Look at how the Collectable sprite works. See if you can understand some of its code:



1. First it makes the original Collectable sprite invisible by hiding it
2. Then it sets up the control variables — we'll come back to these later
3. The `create-collectables` variable is the on/off switch for cloning: the loop creates clones if `create-collectables` is `true`, and does nothing if it's not

Now set up a block for the Collectable sprite so that it reacts to the `game over` broadcast:



This code is similar to the code controlling the Platforms and Edges sprites. The only difference is that you're also setting the `create-collectables` variable to `false` so that no new clones get created when it's 'Game over'.

Note that you can use the `create-collectables` variable to pass messages from one part of your code to another!

Step 4 Power-ups

At the moment you have just one type of collectable: a star that gains you one point when you grab it. On this card, you're going to create a new type of collectable, and you'll do it in a way that will make adding other types of collectables easy. Then you can invent your own power-ups and bonuses and really make the game your own!

I've already included some pieces to do this with the `collectable-type` variable and the `pick-costume` My blocks block. You're going to need to improve on them though.

Let's have a look at how the collectable works right now.

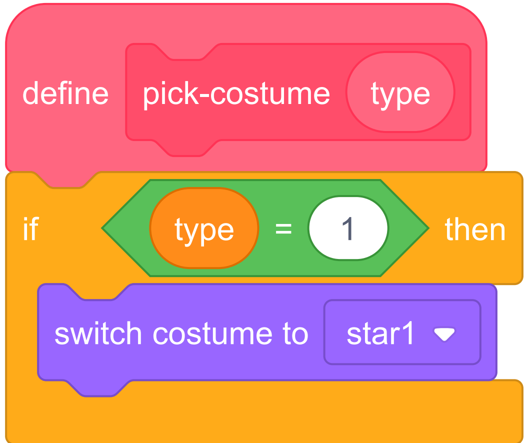
In the scripts for the Collectable sprite, find the `when I start as a clone` code. The blocks you should look at are the ones that give you points for collecting a star:

and this one that selects a costume for the clone:

pick-costume collectable-type

How does picking a costume work?

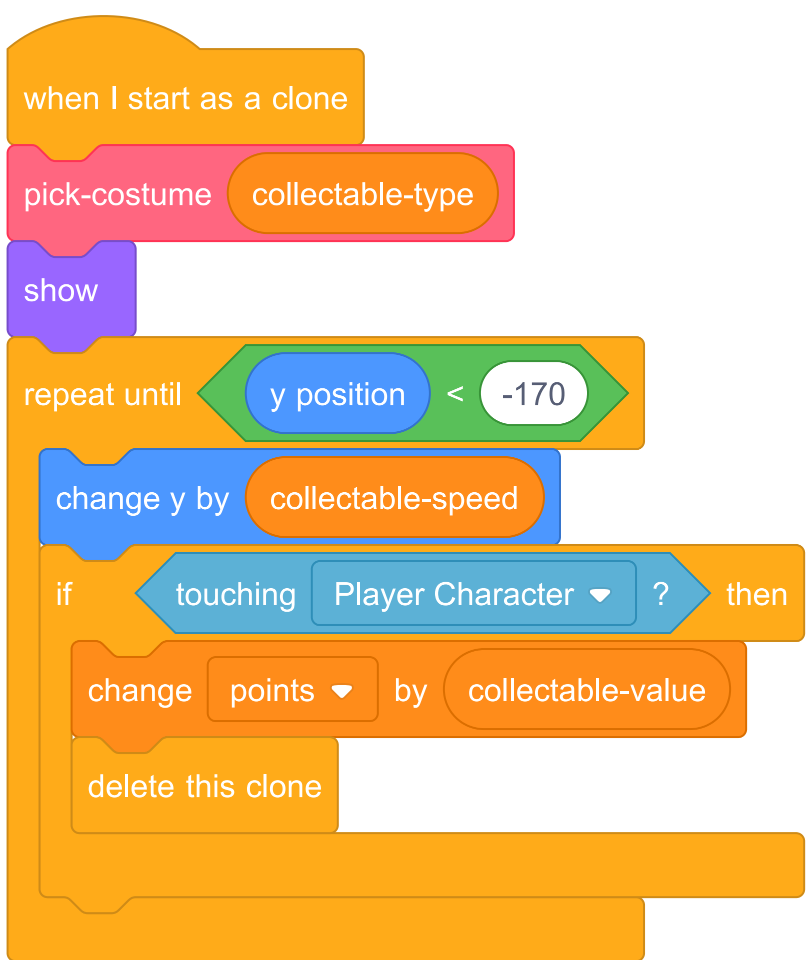
The `pick-costume` block works a bit like the `lose` block, but it has something extra: it takes an input variable called `type`.



When the `pick-costume` block runs, what it does is this:

1. It looks at the `type` input variable
2. If the value of `type` is equal to `1`, it switches to the `star1` costume

Take a look at the part of the script that uses the block:



You can see that the `collectable-type` variable gets passed to the `pick-costume` block. Inside the code for `pick-costume`, `collectable-type` is then used as the input variable (`type`).

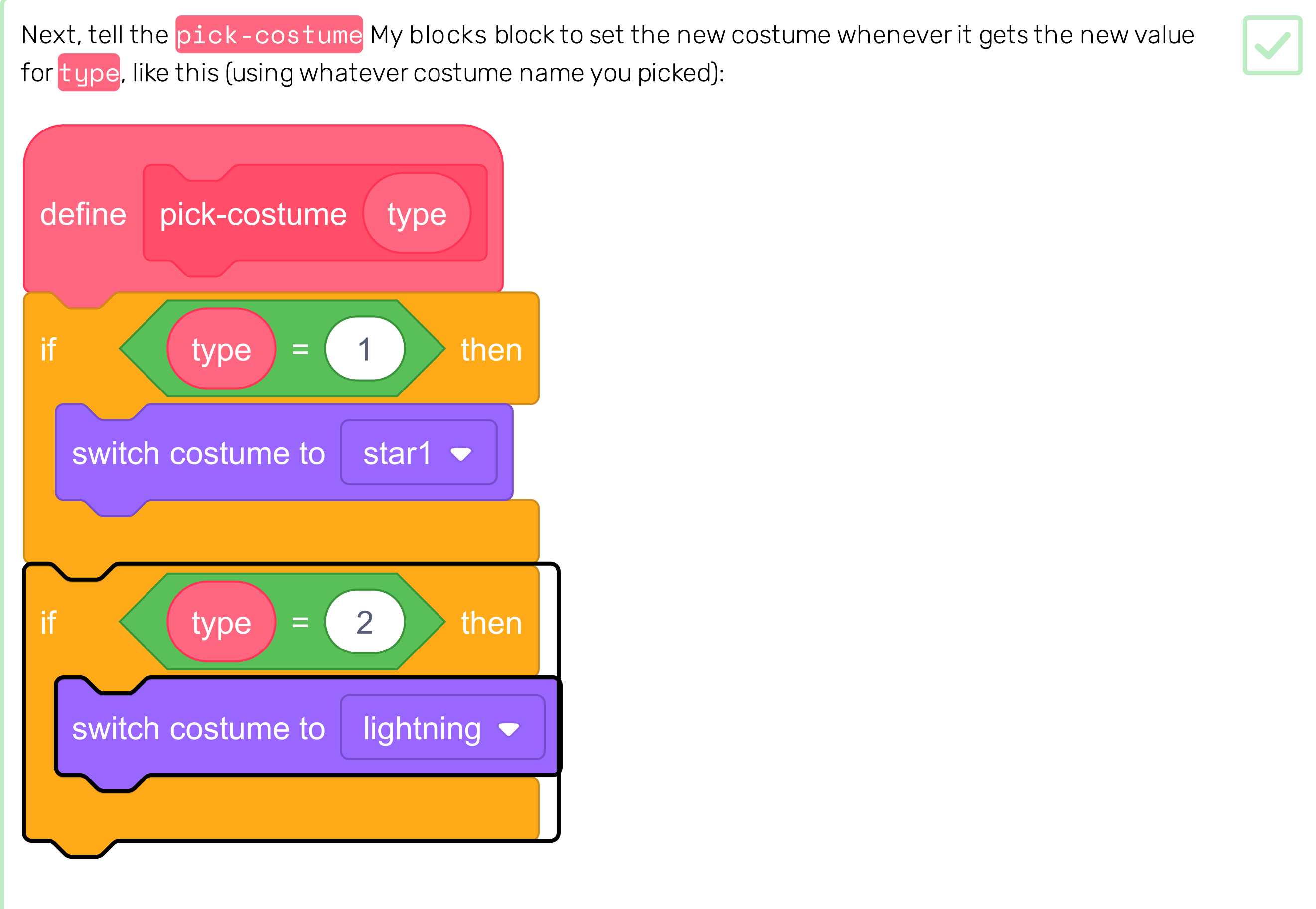
This means that the value of `collectable-type` decides which costume the sprite clone gets.

Add a costume for the new power-up

Of course, right now the Collectable sprite only has one costume, since there's only one type of collectable. You're about to change that.

Add a new costume to the Collectable sprite for your new power-up. I like the lightning bolt, but pick whatever you like.

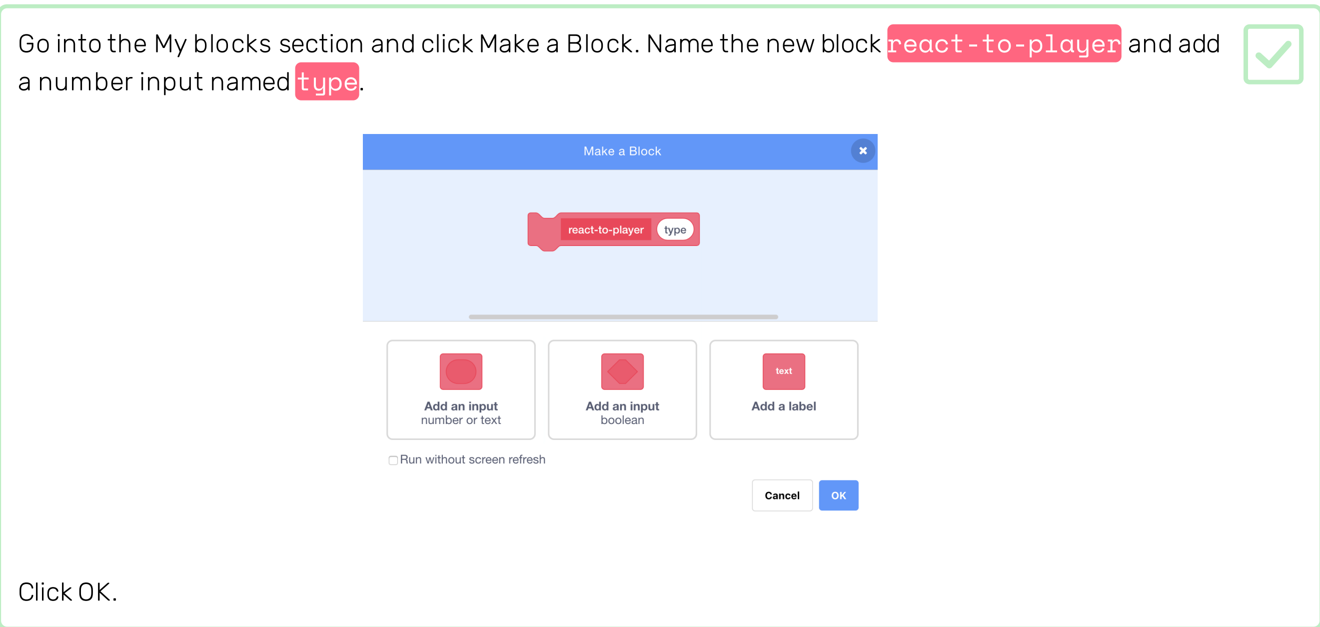
Next, tell the `pick-costume` My blocks block to set the new costume whenever it gets the new value for `type`, like this (using whatever costume name you picked):



Create the power-up code

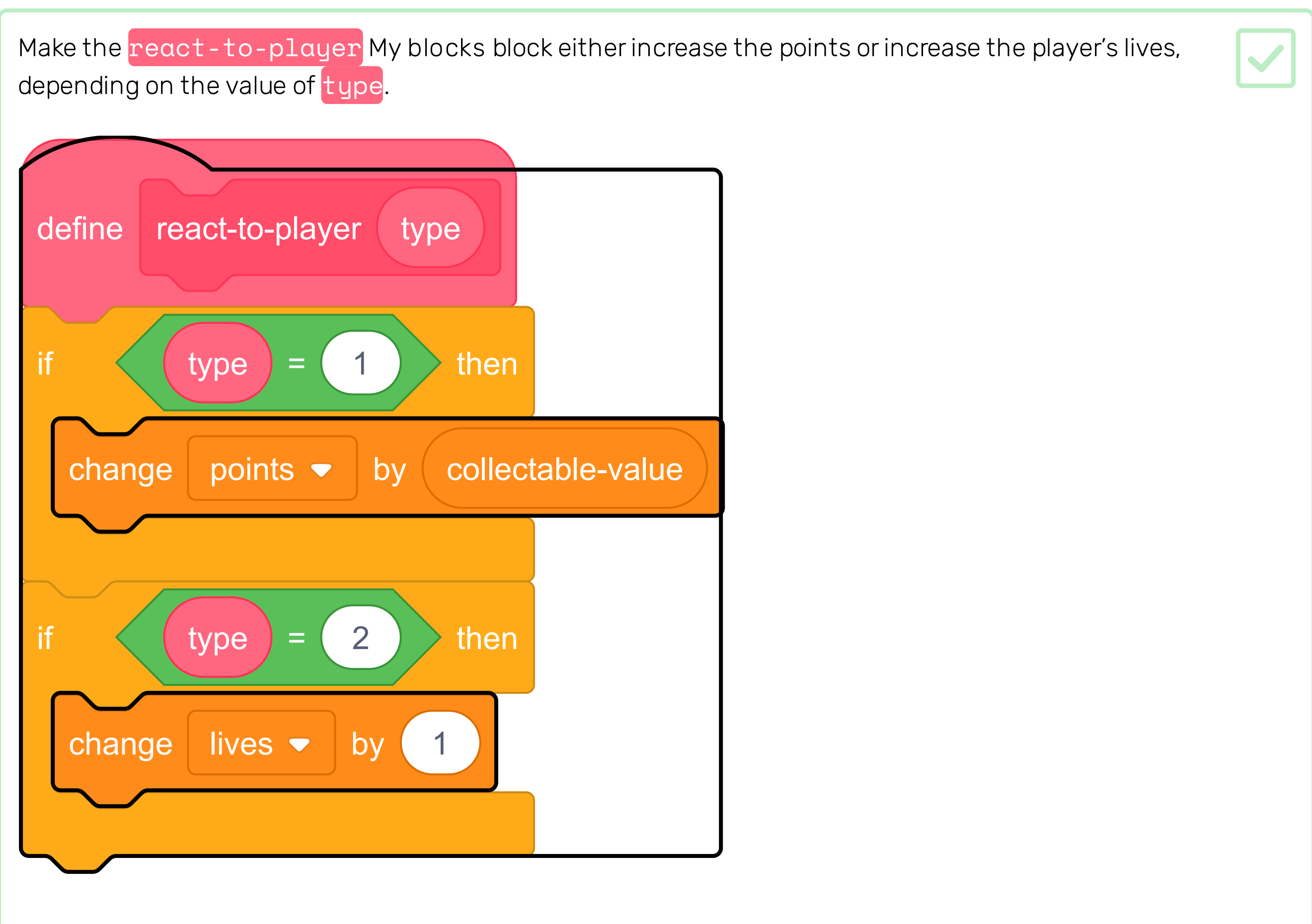
Now you need to decide what the new collectable will do! We'll start with something simple: giving the player a new life. In the next step, you'll make it do something cooler.

Go into the My blocks section and click Make a Block. Name the new block `react-to-player` and add a number input named `type`.

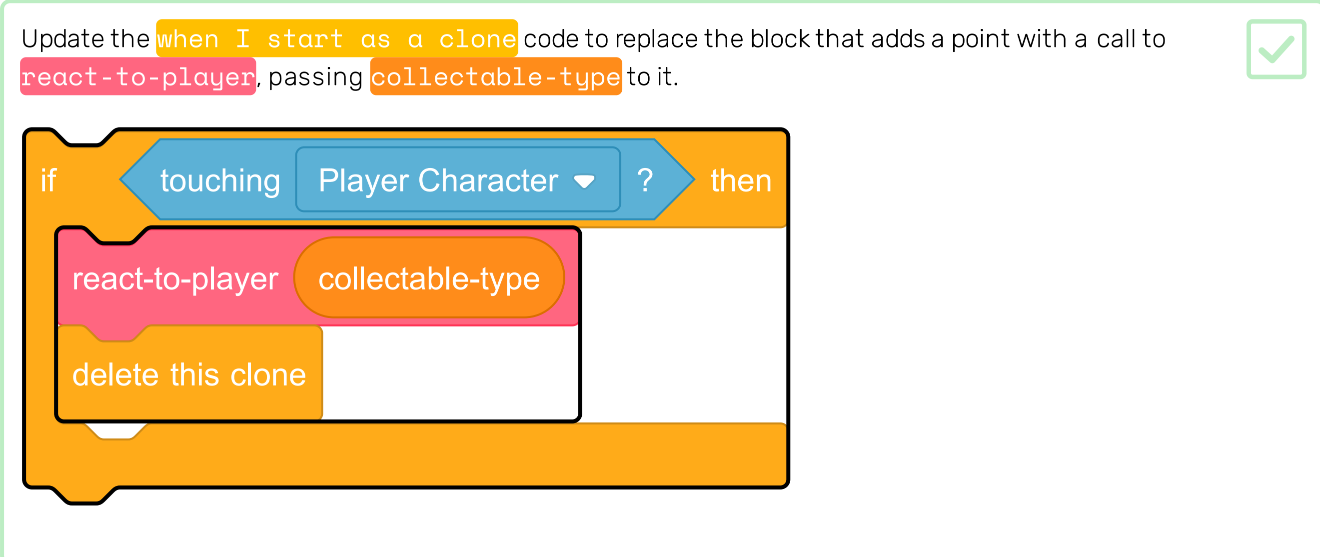


Click OK.

Make the `react-to-player` My blocks block either increase the points or increase the player's lives, depending on the value of `type`.



Update the `when I start as a clone` code to replace the block that adds a point with a call to `react-to-player`, passing `collectable-type` to it.



By using this new `react-to-player` My blocks block, stars still add a point, but the new power-up you've created adds a life.

Using `collectable-type` to make different collectables appear at random

Right now, you might be wondering how you'll tell each collectable the game makes what type it should be.

You do this by setting the value of `collectable-type`. This variable is just a number. As you've seen, it's used to tell the `pick-costume` and `react-to-player` blocks what costume, rules, etc. to use for the collectable.

Working with variables in a clone

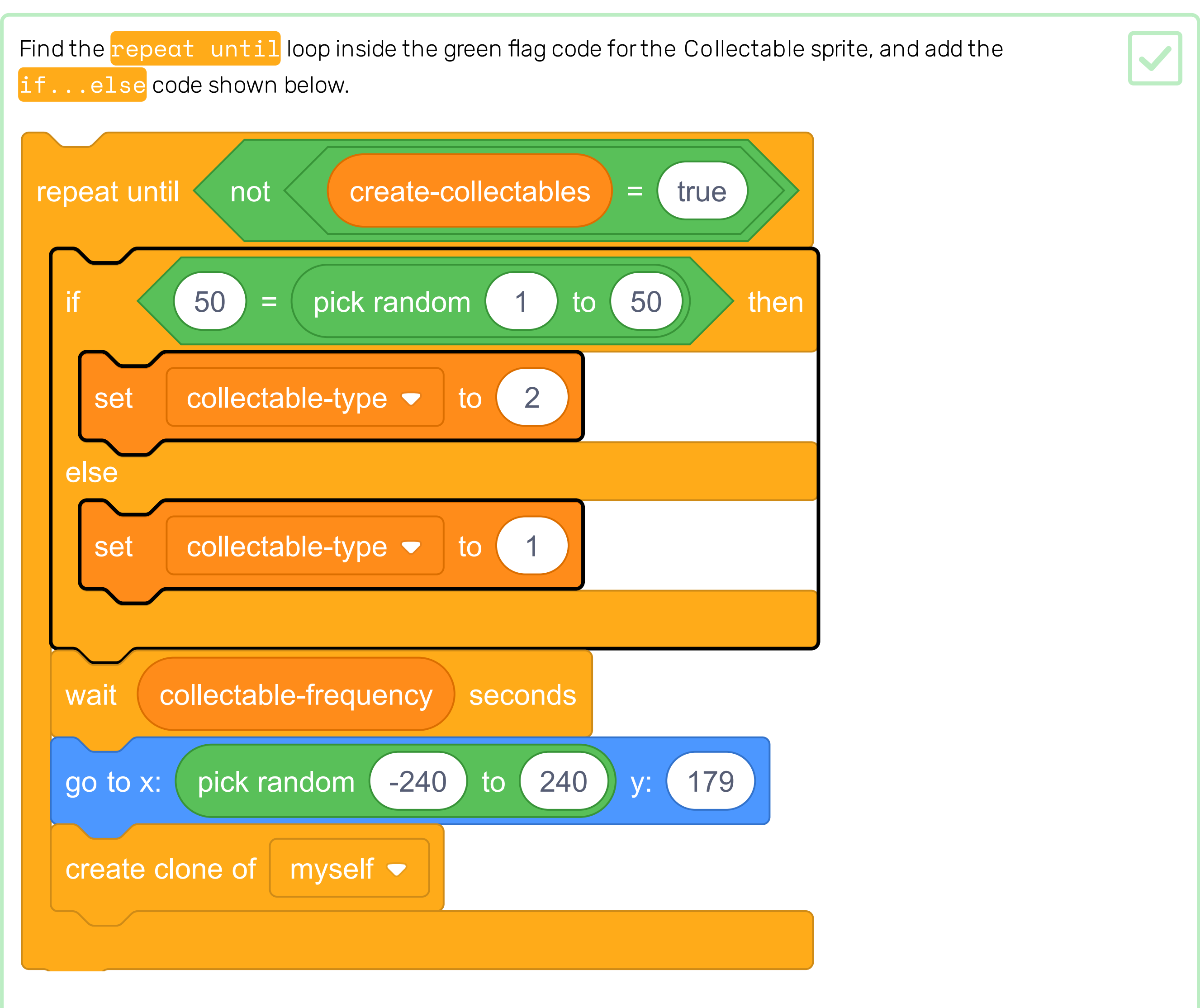
For each clone of the Collectable sprite, you can set a different value for `collectable-type`.

Think of it like creating a new copy of the Collectable sprite with the help of the value that is stored in `collectable-type` at the time the Collectable clone gets created.

You might be wondering whether changing the value of `collectable-type` will turn all the collectables on the Stage into the same type. That doesn't happen, because one of the things that makes clones special is that they cannot change the values of any variables they start with. Sprite clones effectively have constant values. That means that when you change the value of `collectable-type`, this doesn't affect the Collectable sprite clones that are already in the game.

You're going to set the `collectable-type` to either `1` or `2` for each new clone that you make. To keep the game interesting, pick between the numbers at random to make a random collectable every time.

Find the `repeat until` loop inside the green flag code for the Collectable sprite, and add the `if...else` code shown below.



This code gives a 1-in-50 chance of setting the `collectable-type` to `2`. After all, you don't want to give the player the chance to collect an extra life too often, otherwise the game would be too easy.

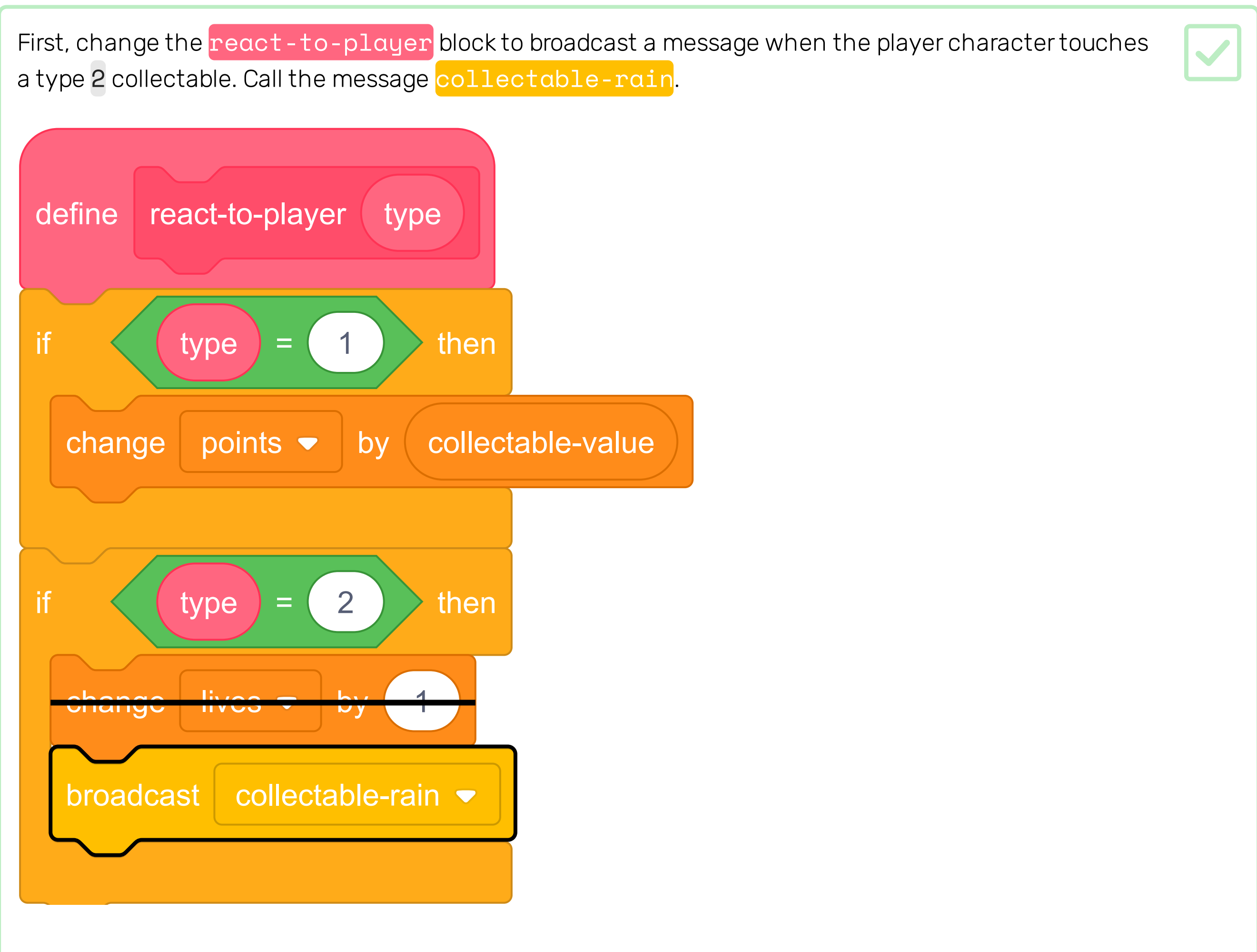
Now you have a new type of collectable that sometimes shows up instead of the star, and that gives you an extra life instead of a point when you collect it.

Step 5 Super power-ups!

Now that you have a new power-up collectable working, it's time to make it do something really cool: Let's make it 'rain' power-ups for a few seconds, instead of just giving out an extra life.

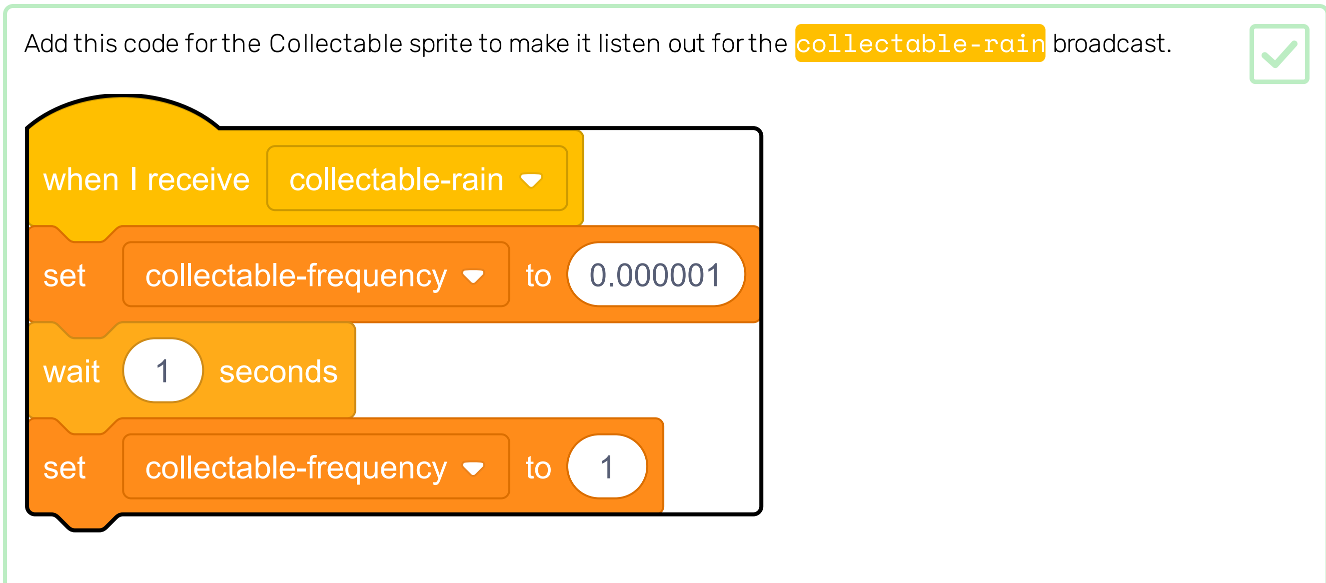
For this, you're going to use another `broadcast` message.

First, change the `react-to-player` block to broadcast a message when the player character touches a type `2` collectable. Call the message `collectable-rain`.



Now you need to create a new piece of code inside the Collectable sprite scripts that will start whenever the `collectable-rain` message is broadcast.

Add this code for the Collectable sprite to make it listen out for the `collectable-rain` broadcast.

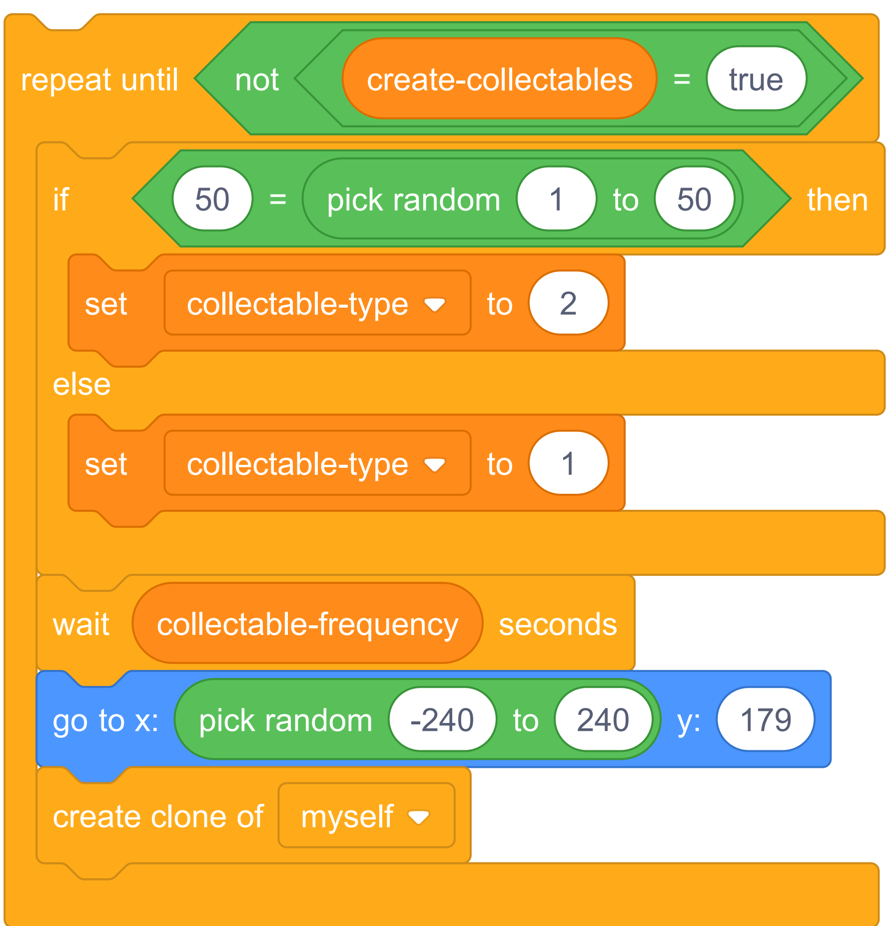


What does the new code do?

This piece of code waits to receive a broadcast, and responds by setting the `collectable-frequency` variable to a very small number, then waiting for one second, and then changing the variable back to `1`.

Let's look at how the `collectable-frequency` variable is used to find out why this makes it rain collectables.

In the main game loop, the part of the code that makes Collectable sprite clones gets told by the `collectable-frequency` variable how long to wait between making one clone and the next:



You can see that the `wait` block here pauses the code for the length of time set by `collectable-frequency`.

If the value of `collectable-frequency` is `0.000001`, the `wait` block only pauses for one millionth of a second, meaning that the `repeat until` loop will run many more times than normal. As a result, the code is going to create a lot more power-ups than it normally would, until `collectable-frequency` is changed back `1`.

Can you think of any problems that might cause? There'll be a lot more power-ups…what if you kept catching them?

Challenge: get creative!

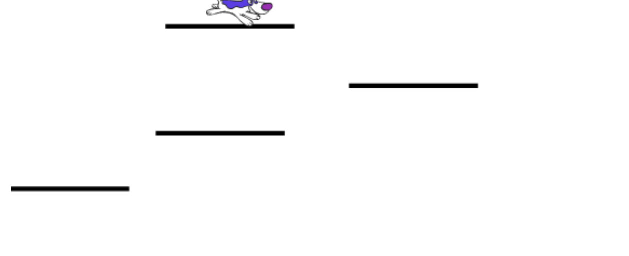
Based on the previous two cards you can now make as many different power-up collectables as you want! What about one that gives out 20 times the usual number of points, or adds three lives, or makes it so the player can’t run out of lives for a period of time? Come up with two cool power-ups and see if you can make them!

Step 6 Adding some competition

Your game works and now you can collect points, get special powers from power-ups, and lose. We're getting somewhere! Maybe it'd be fun to add some competition though — what about including a character that moves around a little, but that you're not supposed to touch? This will be similar to enemies in the traditional platform games like Super Mario that we're inspired by here.

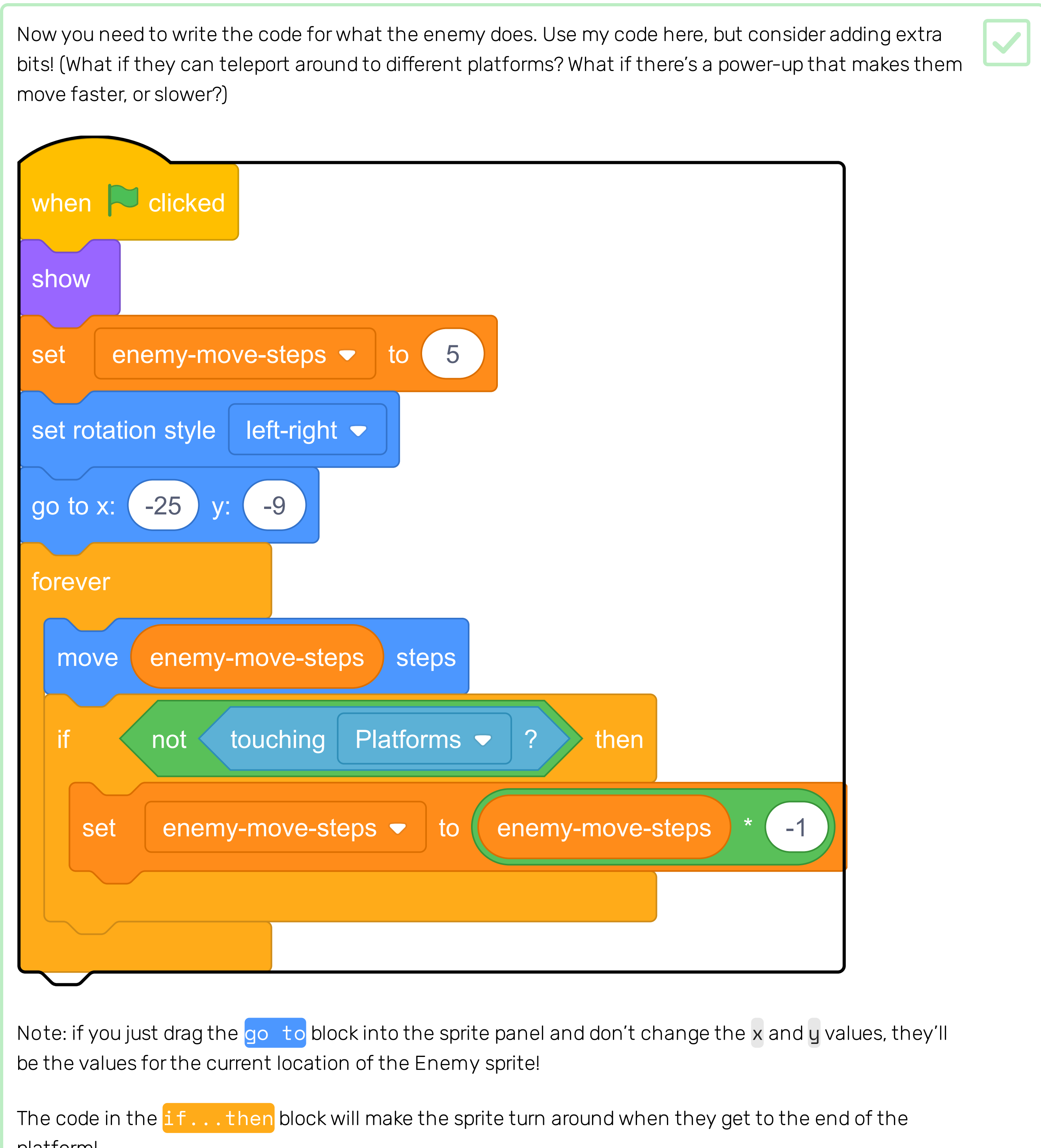
First, pick a sprite to add as your enemy. Because our player character is a cat, I chose a dog. There are lots of other sprites you could add though. I also renamed the sprite Enemy, just to make things clearer for me.

Resize the sprite to the right size, and place it somewhere appropriate to start. Here's what mine looks like:



Write the easiest code first: set up its block for reacting to the `game over` message to make the enemy disappear when the player loses the game.

Now you need to write the code for what the enemy does. Use my code here, but consider adding extra bits! (What if they can teleport around to different platforms? What if there's a power-up that makes them move faster, or slower?)

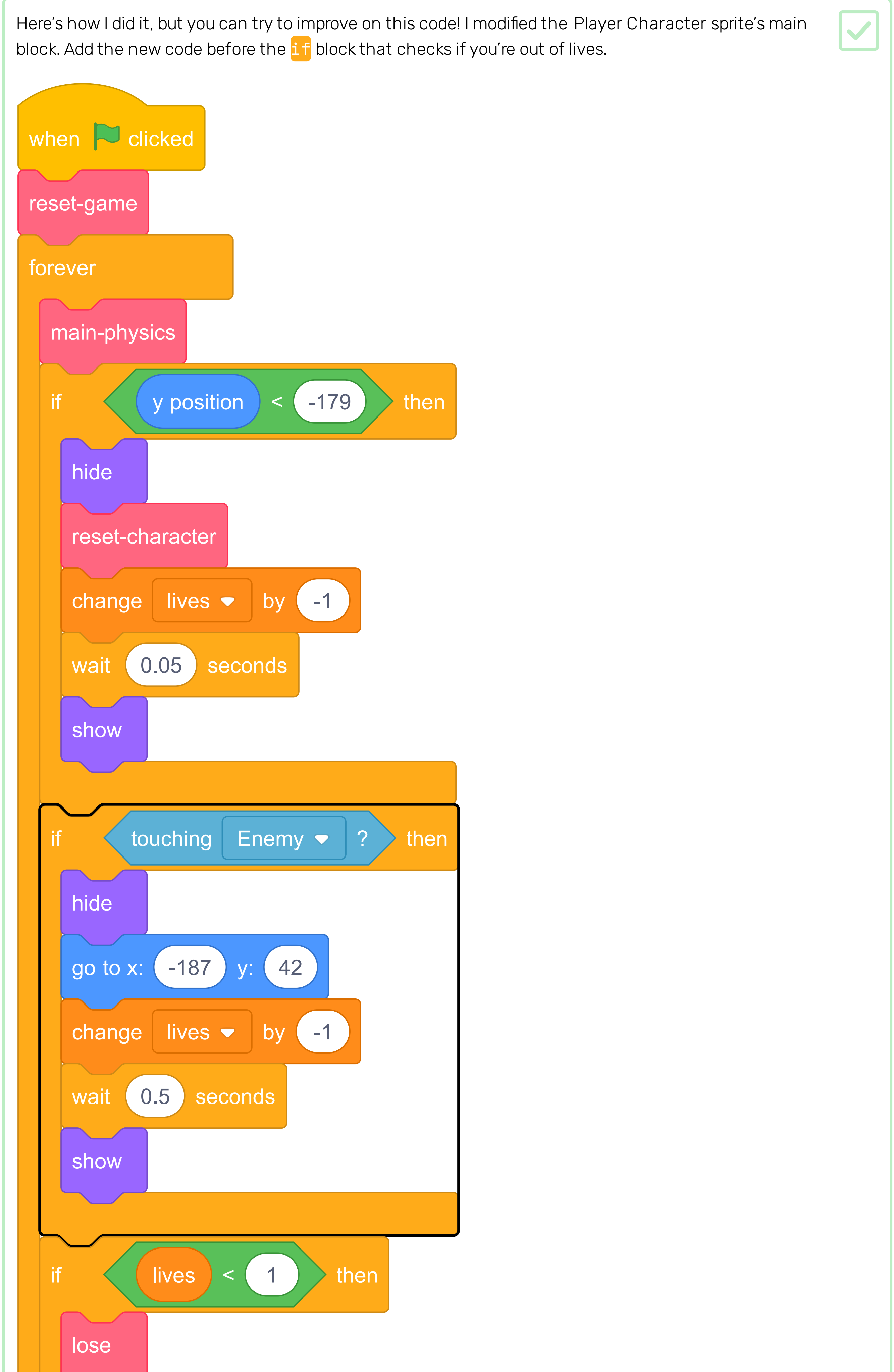


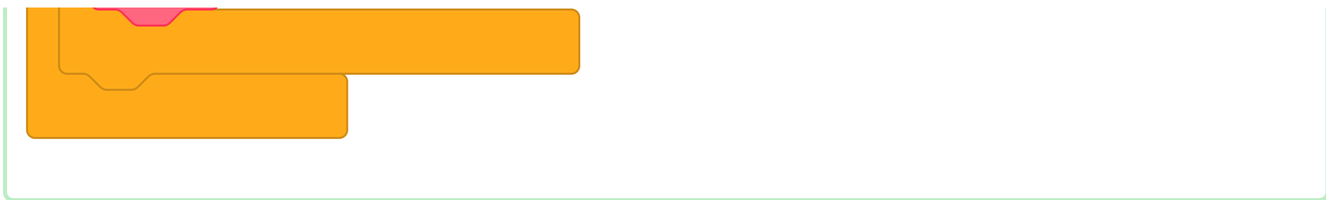
Note: if you just drag the `go to` block into the sprite panel and don't change the `x` and `y` values, they'll be the values for the current location of the Enemy sprite!

The code in the `if...then` block will make the sprite turn around when they get to the end of the platform!

The next thing you'll need is for the player to lose a life when their Player Character sprite touches the Enemy sprite. Also, you need to make sure the sprites stop touching really quickly, since otherwise the code that checks for touching will keep running and the player will keep losing lives.

Here's how I did it, but you can try to improve on this code! I modified the Player Character sprite's main block. Add the new code before the `if` block that checks if you're out of lives.





The new code hides the Player Character sprite, moves it back to its starting position, reduces the `lives` variable by `1`, and after half a second makes the sprite re-appear.

Step 7 Level 2

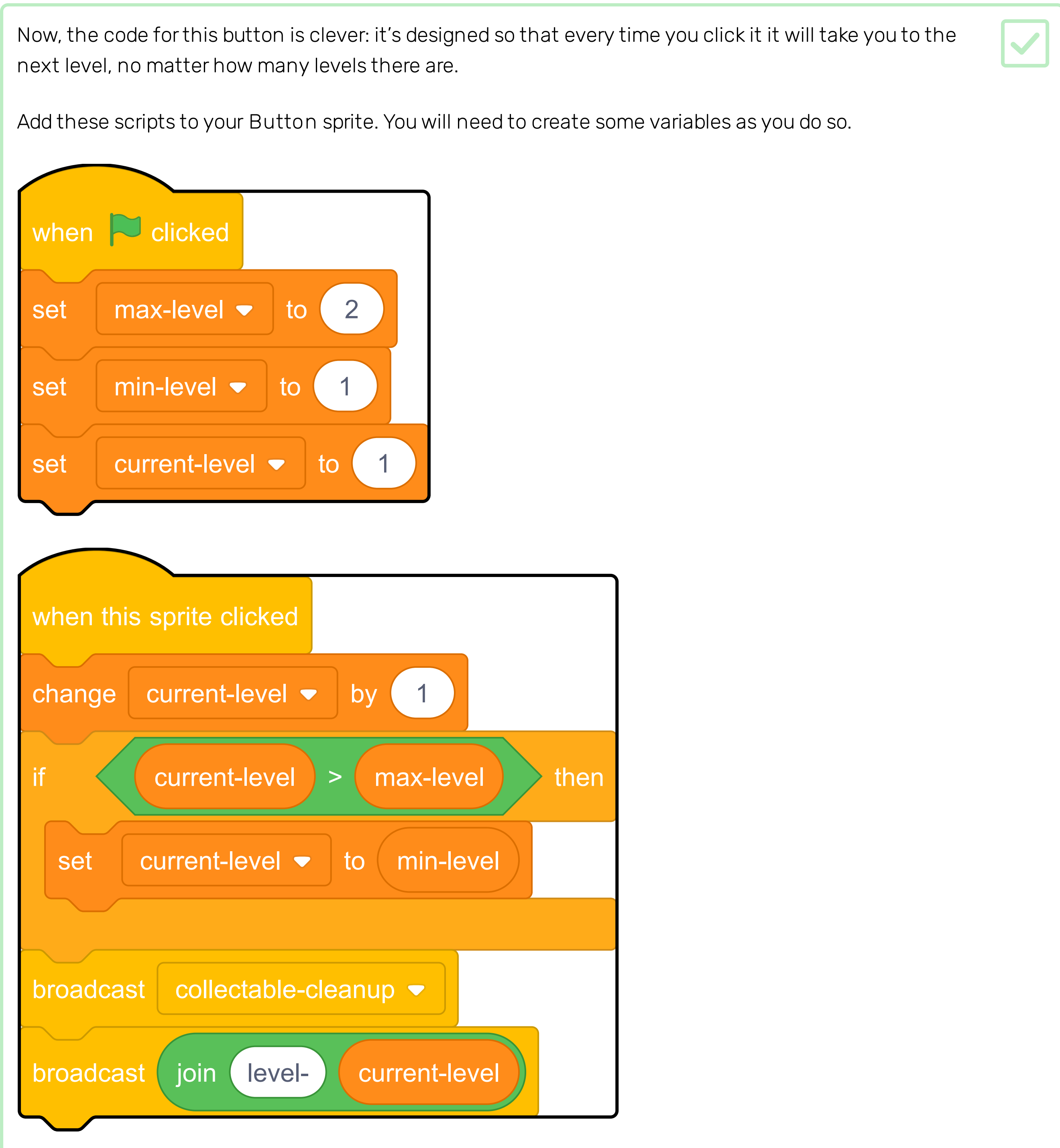
With this step, you're going to add a new level to the game that the player can get to by just pressing a button. Later, you can change your code to make it so they need a certain number of points, or something else, to get there.

Moving to the next level

First, create a new sprite as a button by either adding one from the library or drawing your own. I did a bit of both and came up with this:

Now, the code for this button is clever: it's designed so that every time you click it it will take you to the next level, no matter how many levels there are.

Add these scripts to your Button sprite. You will need to create some variables as you do so.



Can you see how the program will use the variables you created?

- `max-level` stores the highest level
- `min-level` stores the lowest level
- `current-level` stores the level the player is on right now

These all need to be set by the programmer (you!), so if you add a third level, don't forget to change the value of `max-level`! `min-level` will never need to change, of course.

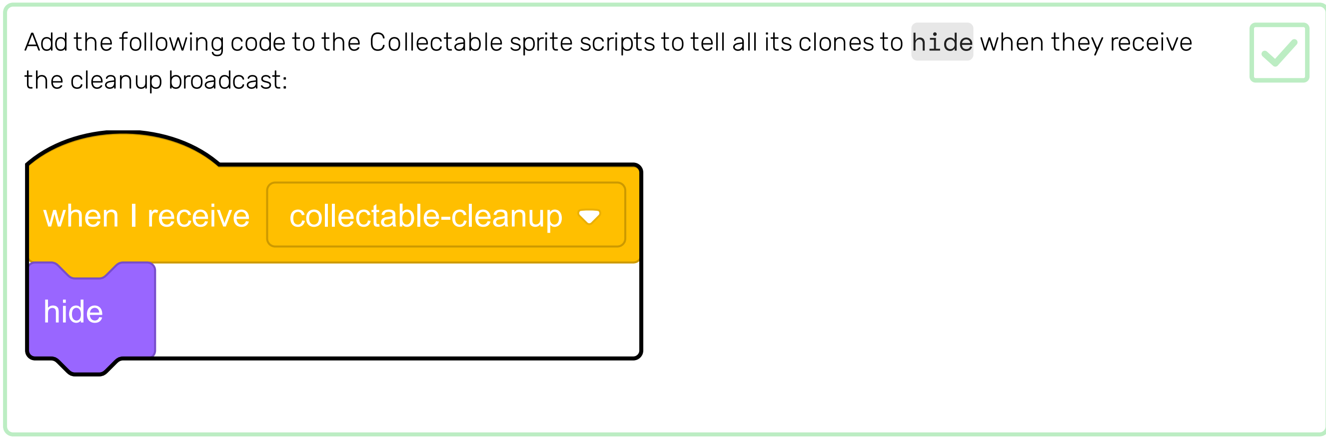
The broadcasts are used to tell the other sprites which level to display, and to clear up the collectables when a new level starts.

Make the sprites react

The Collectable sprite

Now you need to get the other sprites to respond to these broadcasts! Start with the easiest one: clearing all the collectables.

Add the following code to the Collectable sprite scripts to tell all its clones to `hide` when they receive the cleanup broadcast:

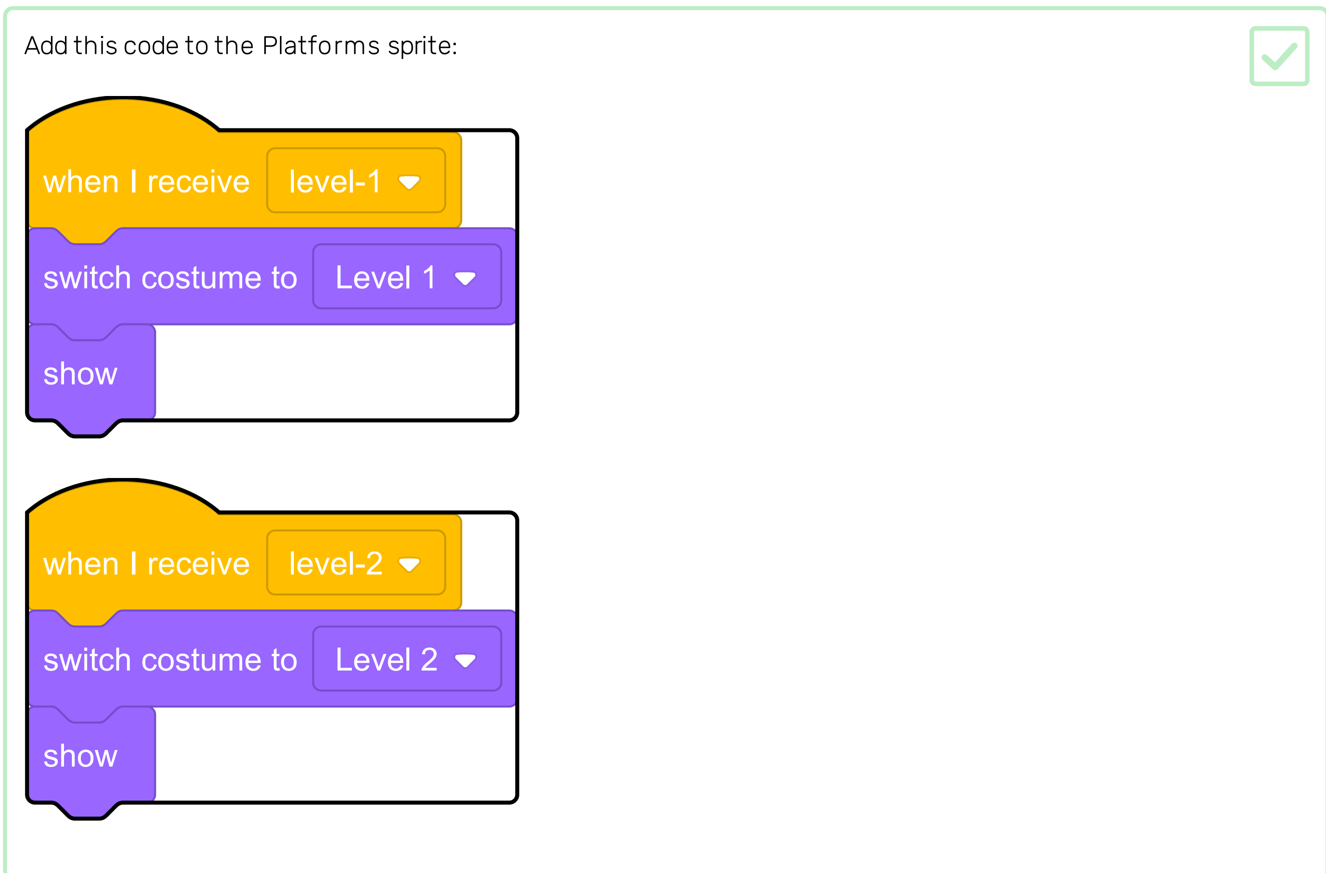


Since one of the first things any new clone does is show itself, you don't have to worry about unhiding collectables!

The Platforms sprite

Now to switch the Platforms sprite. You can design your own new level later if you like, but for now let's use the one I've already included — you'll see why on the next step!

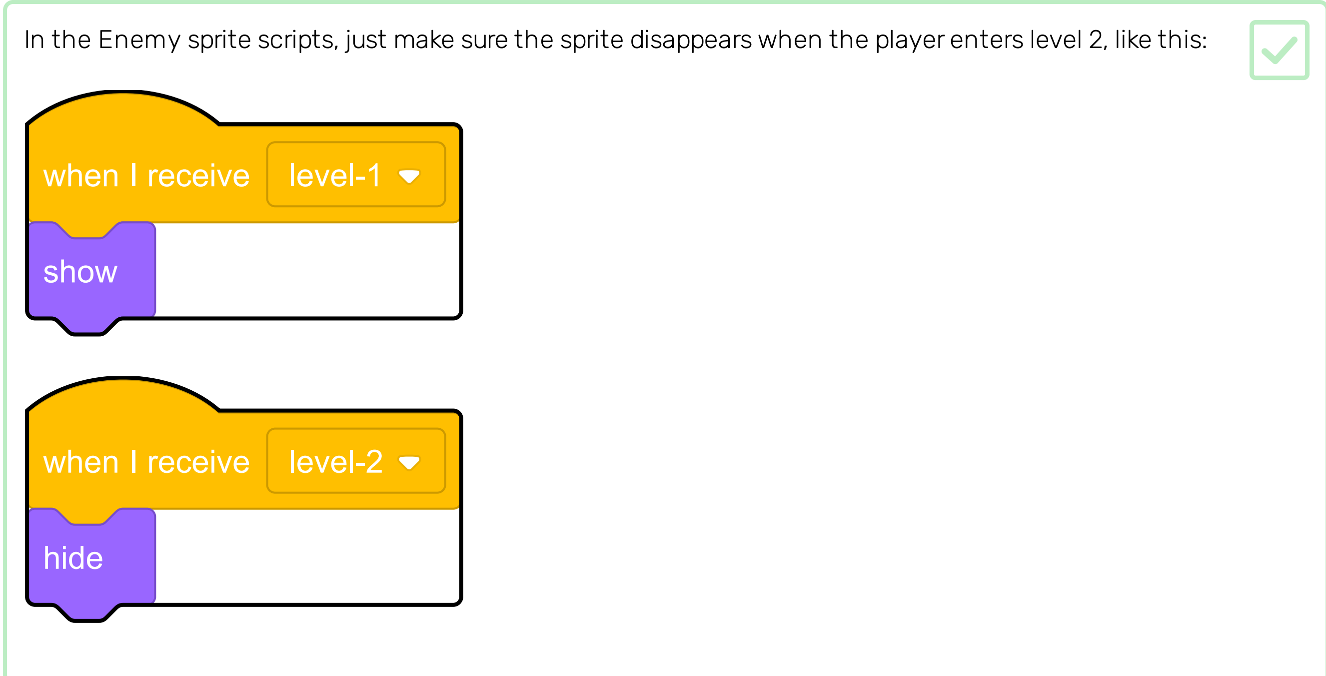
Add this code to the Platforms sprite:



It receives the `joined` messages of `level-` and `current-level` that the Button sprite sends out, and responds by changing the Platforms costume.

The Enemy sprite

In the Enemy sprite scripts, just make sure the sprite disappears when the player enters level 2, like this:

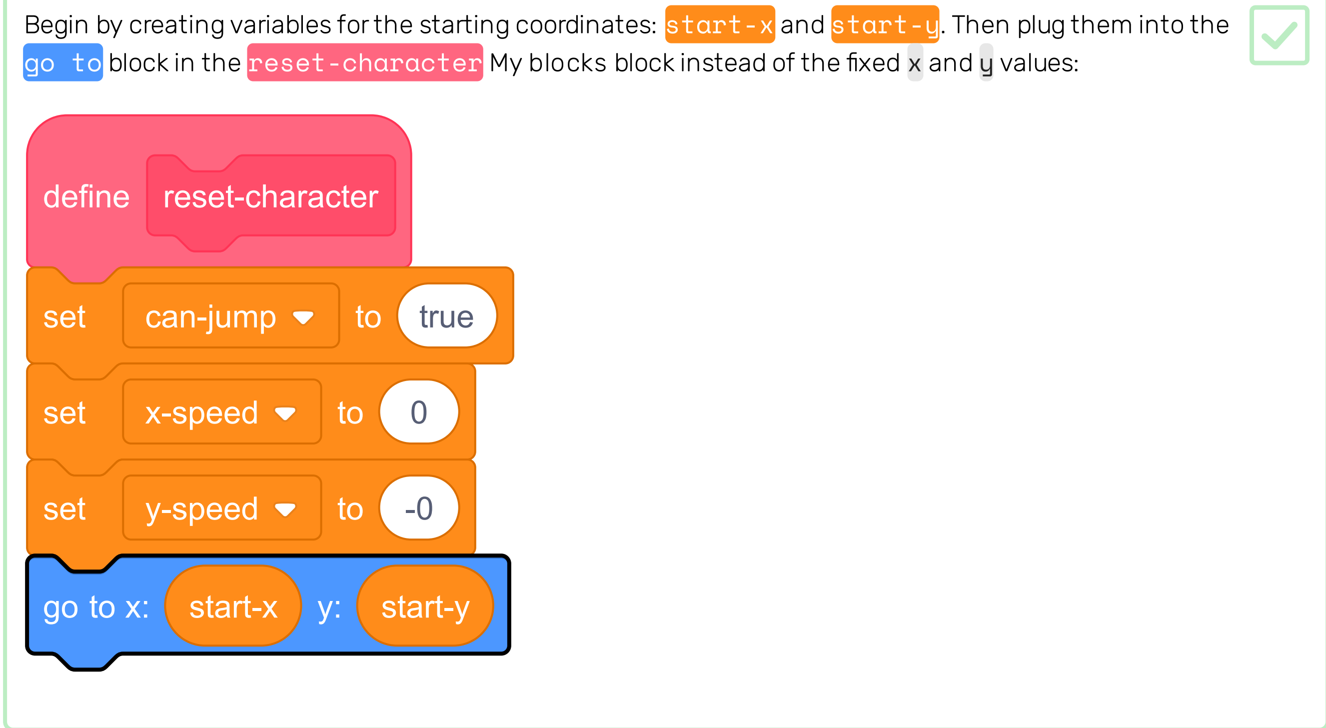


If you prefer, you can make the enemy move to another platform instead. In that case, you would use a `go to` block instead of the `show` and `hide` blocks.

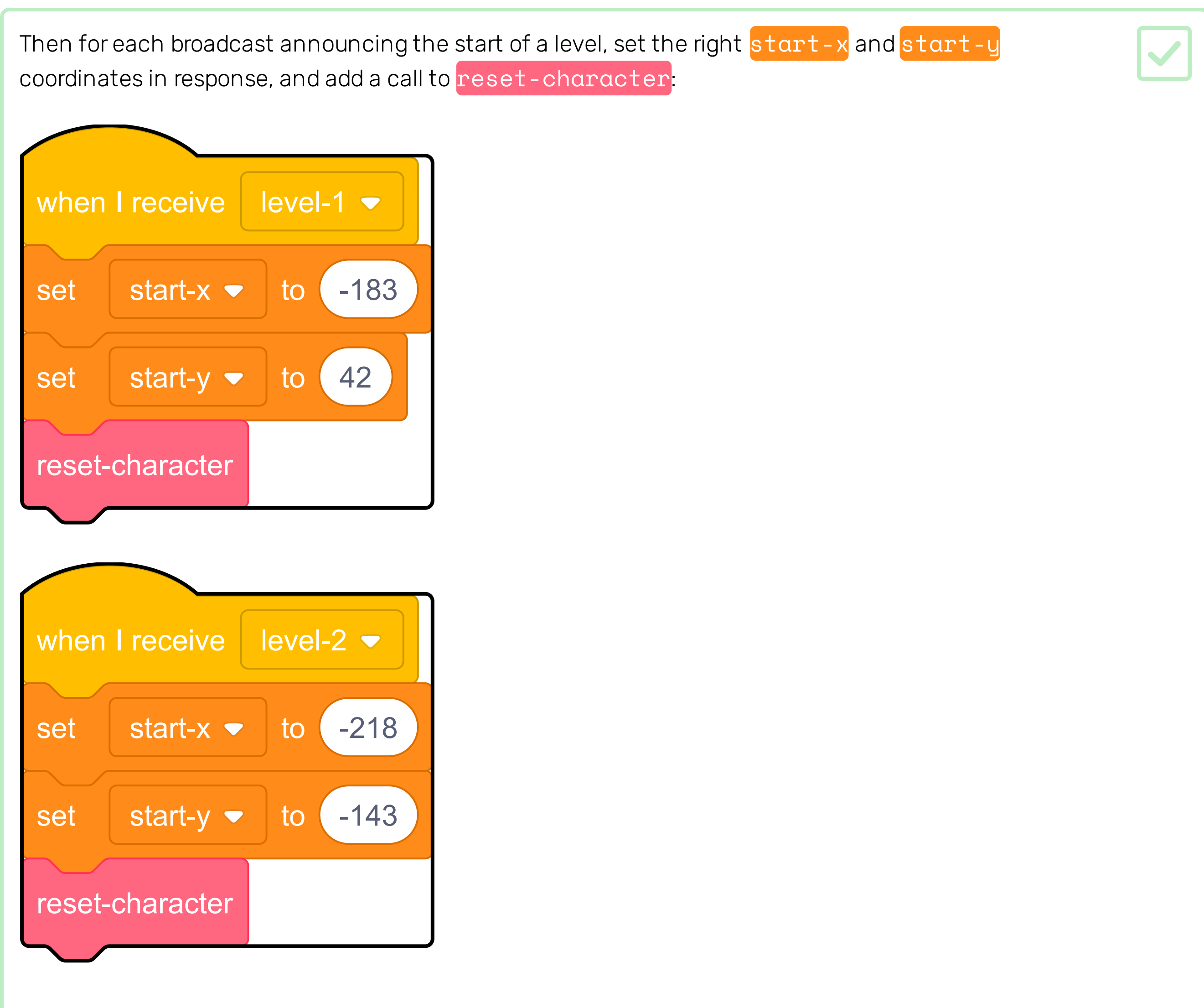
Make the Player Character appear in the right place

Whenever a new level starts, the Player Character sprite needs to go to the right place for that level. To make this happen, you need to change where the sprite gets its coordinates from when it first appears on the Stage. At the moment, there are fixed `x` and `y` values in its code.

Begin by creating variables for the starting coordinates: `start-x` and `start-y`. Then plug them into the `go to` block in the `reset-character` My blocks block instead of the fixed `x` and `y` values:



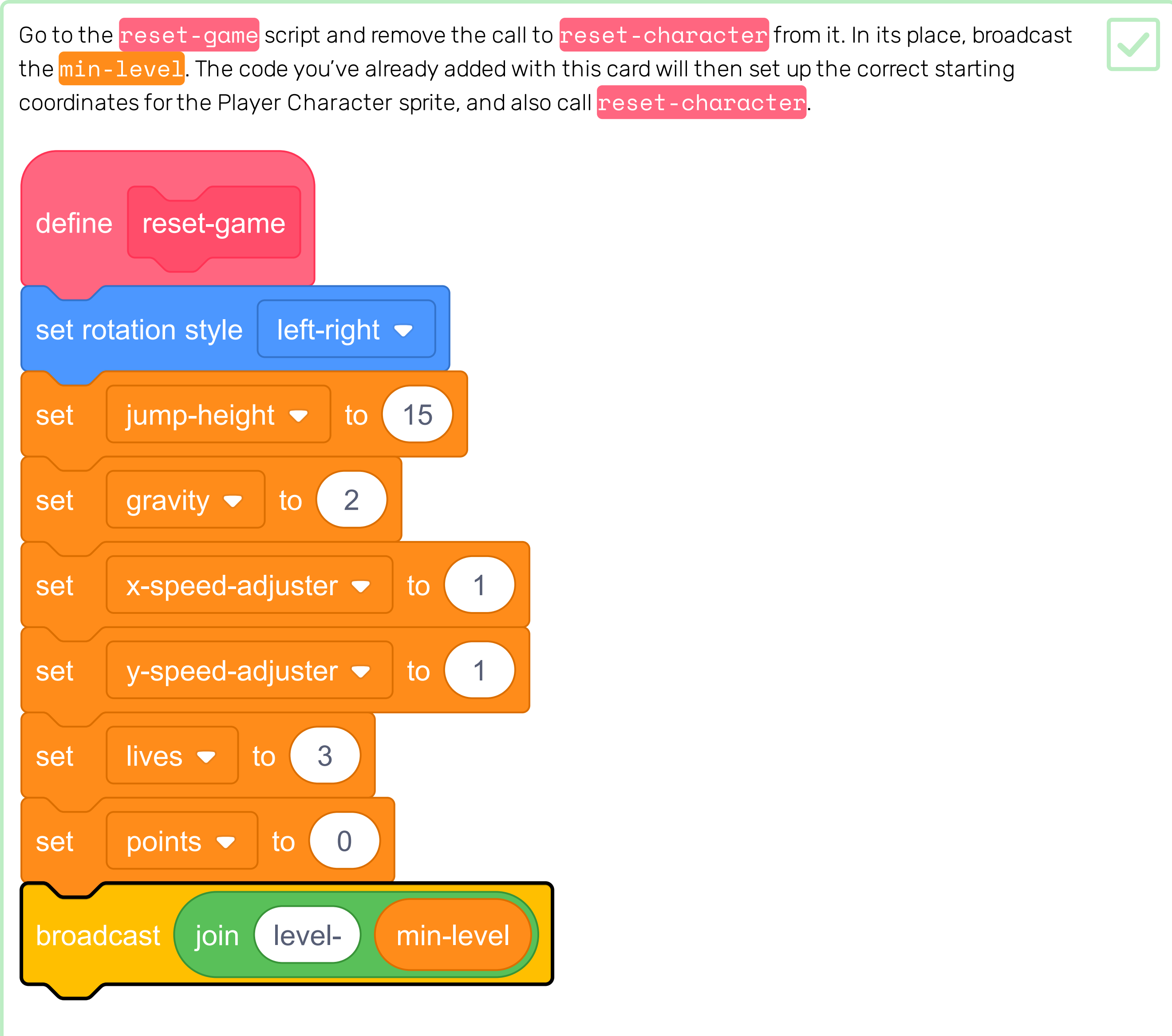
Then for each broadcast announcing the start of a level, set the right `start-x` and `start-y` coordinates in response, and add a call to `reset-character`:



Starting at Level 1

You also need to make sure that every time someone starts the game, the first level they play is level 1.

Go to the `reset-game` script and remove the call to `reset-character` from it. In its place, broadcast the `min-level`. The code you've already added with this card will then set up the correct starting coordinates for the Player Character sprite, and also call `reset-character`.



Resetting the Player Character versus resetting the game

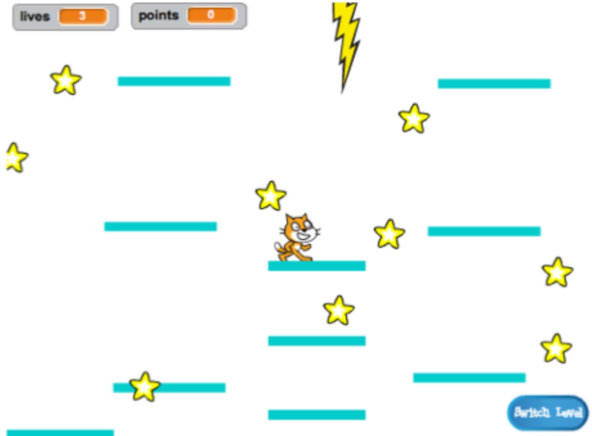
Notice that the first block in the Player Character sprite's main green flag script is a call to the `reset-game` My blocks block.

This block sets up all the variables for a new game and then calls the `reset-character` My blocks block, which places the character back in its correct starting position.

Having the `reset-character` code in its own block separate from `reset-game` allows you to reset the character to different positions without having to reset the whole game.

Step 8 Moving platforms

The reason I asked you to use my version of level 2 is the gap you might have noticed in the middle of the layout. You're going to create a platform that moves through this gap and that the player can jump on and ride!



First, you'll need the sprite for the platform.

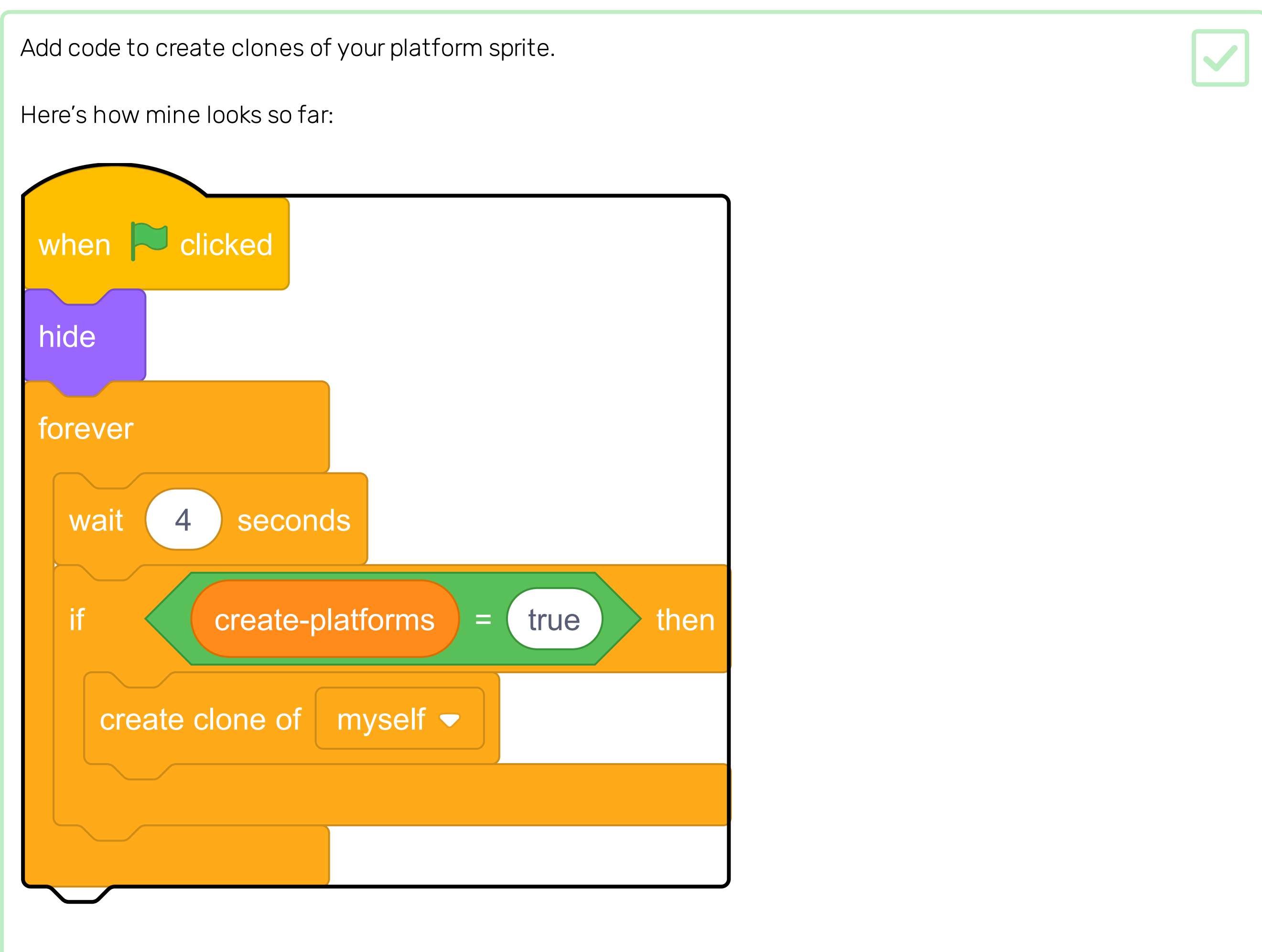
Add a new sprite, name it Moving-Platform, and using the costume customisation tools in the Costumes tab to make it look like the other platforms (use vector mode).

Now, let's add some code to the sprite.

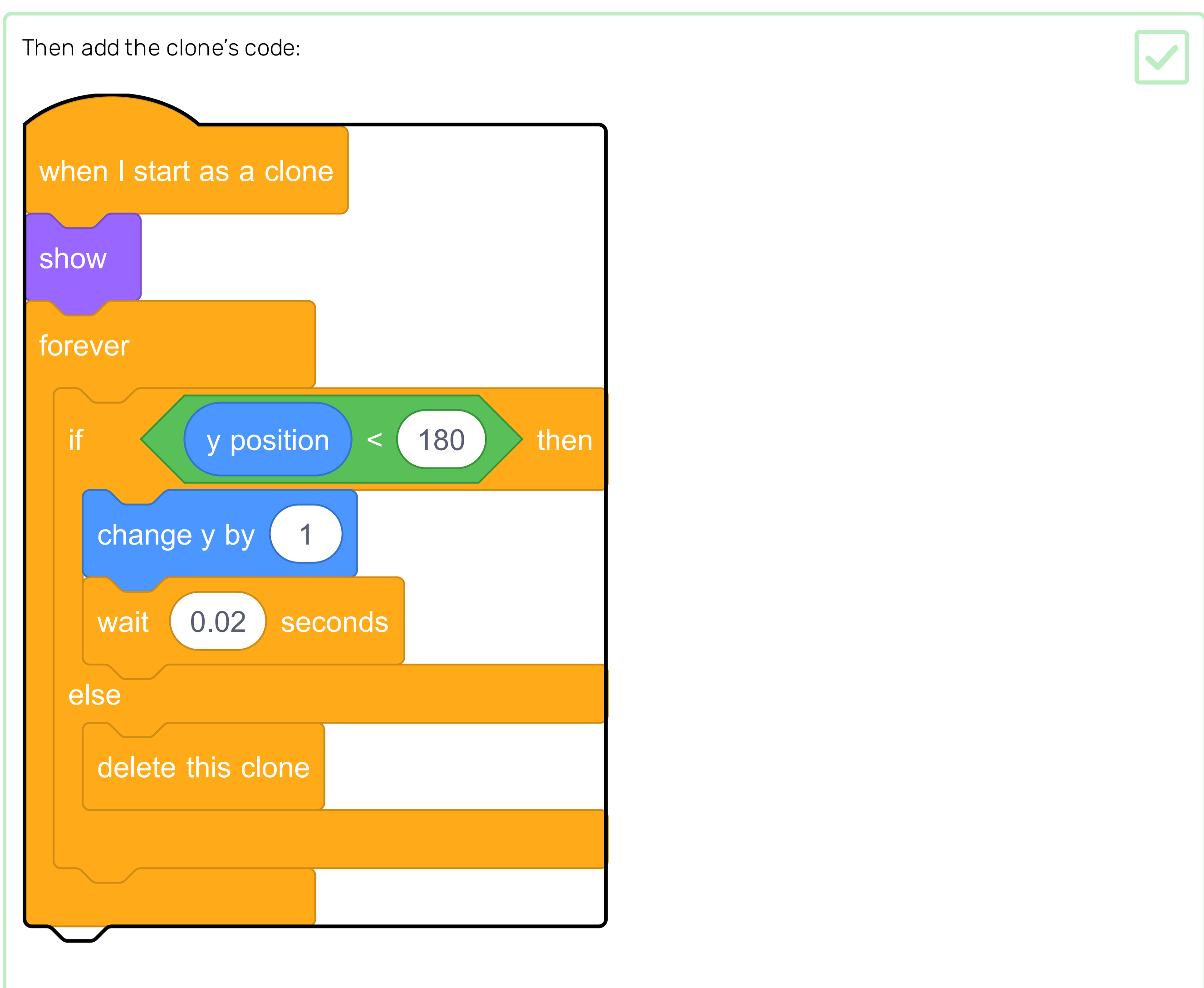
Begin with the basics: to make a never-ending set of platforms moving up the screen, you'll need to clone the platform at regular intervals. I picked `4` seconds as my interval. You also need to make sure that there's an on/off switch for making the platforms, so that they don't show up in level 1. I'm using a new variable called `create-platforms`.

Add code to create clones of your platform sprite.

Here's how mine looks so far:

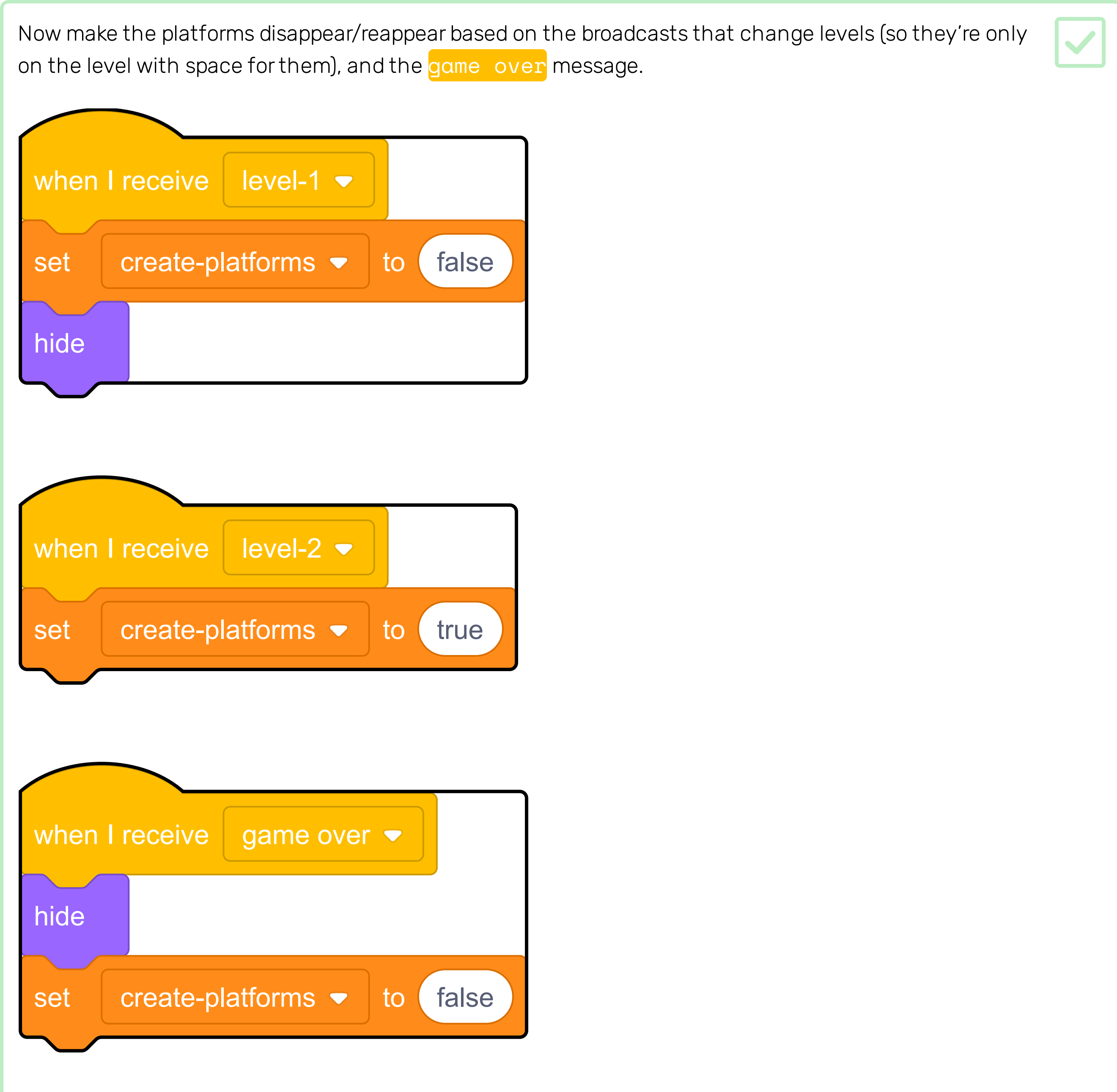


Then add the clone's code:



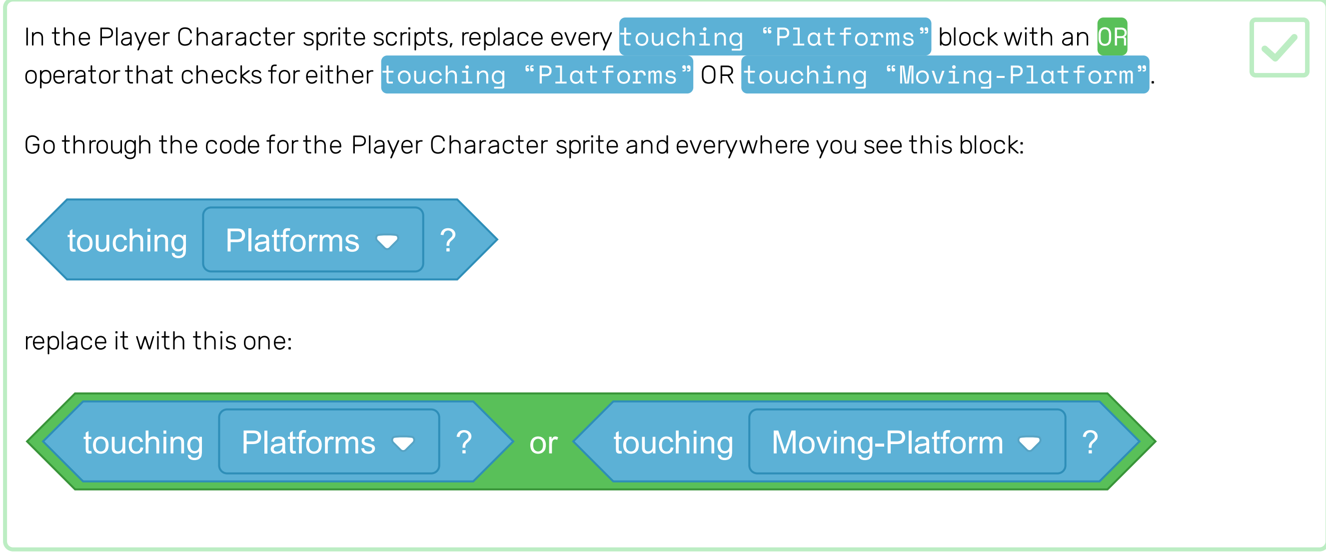
This code makes the Moving-Platform clone move up to the top of the screen, slowly enough for the player to jump on and off, and then disappear.

Now make the platforms disappear/reappear based on the broadcasts that change levels (so they're only on the level with space for them), and the `game over` message.



Now, if you try to actually play the game, the Player Character falls through the platform! Any idea why?

It's because the physics code doesn't know about the platform. It's actually a quick fix:



In the Player Character sprite scripts, replace every `touching “Platforms”` block with an `OR` operator that checks for either `touching “Platforms”` OR `touching “Moving-Platform”`.

Go through the code for the Player Character sprite and everywhere you see this block:

```
touching (Platforms ▼) ?
```

replace it with this one:

```
<touching (Platforms ▼) ?> or <touching (Moving-Platform ▼) ?>
```

Step 9 What next?

You've got a complete game now, but there's a whole lot more you can do with it! Here are a few ideas to get you started:

High scores

- Keep a list of the names and scores of people who've gotten high scores in the game! You'll need to use the `ask` block to the get their name.

New power-ups

- Try adding some new power-ups, for example for:
  - Immunity to enemies
  - More lives
  - Bigger player character
  - Smaller player character

Scrolling levels

- Can you figure out how to make the levels scroll along, so the player character can move through them from left to right? Or at least make it *look* like that's what's happening?

Completing levels

- Right now, the levels never end. What if, instead of pushing a button, you needed a certain number of points to get to the next level?

Play With physics

- Try changing some of the values in the physics engine, like the `gravity`, `jump height`, `x-speed-adjuster`, and `y-speed-adjuster`. How do they change the game?

- Can you use the physics to make power-ups?

More levels

- Add more levels! Make better art! Use the Stage background to make the game look cooler while still keeping platforms easy to work with for you as the coder.

Sound effects

- This game is totally silent right now! Try adding background music and sound effects using the blocks in the Sound section!

Secrets

- Think of secret bonuses, cheat codes, and other 'easter eggs' you can hide in the game for players to discover. Try to code some of them!

Different characters

- Add more character sprites and let the player pick one. Make the characters different in attributes like size, how high they jump, maybe even how many lives they have, and how many points they get from collectables!

Published by Raspberry Pi Foundation (https://www.raspberrypi.org) under a Creative Commons license (https://creativecommons.org/licenses/by-sa/4.0/).
View project & license on GitHub (https://github.com/RaspberryPiLearning/cd-advanced-scratch-sushi)