

## Week 3

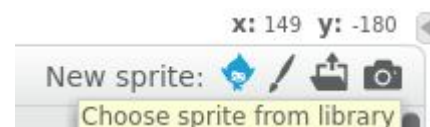
Last week we made a grid of cells which our SpaceChem-like robot will work in. This time we'll work out how to make it so the player can put the 'direction instruction' arrows onto the grid.


This worksheet will use [the Scratch project from the end of Week 2](#), but you should be able to use your own project if you prefer.

### Start with the 'right arrow'

We'll start with one of the direction arrows, and then it will be easy to use the same ideas for the other direction.

It's quickest to start with an arrow sprite from the library. If there's time at the end, you can draw your own arrow instead. Use the leftmost 'New sprite' button, and choose something. I used 'Arrow1'.



Click green flag to draw your grid, and then shrink your arrow using the  tool until it fits nicely into a cell.

### Let the player drag the arrow

We will write a short script which makes the sprite follow the mouse until the player lets go of the mouse button. The 'drag' should start when the player clicks on the arrow, so we'll start the script with the 'when this sprite clicked' hat-block.



Then, until the player lets go of the mouse button, we want the arrow to go to where the mouse is. This is what the 'repeat until' control block is for.



#### **Add this script to the arrow sprite:**

You'll see we haven't filled in the 'question' part of the 'repeat until' yet. We want to keep going to the mouse pointer until the mouse button is 'up'. Scratch doesn't directly have a way of asking whether the mouse

button is 'up', only whether it's 'down':



Another way of asking 'is the button up?' is to ask 'is the button *not* down?', so this is what we have to do:



#### **Use this block to fill in the script:**



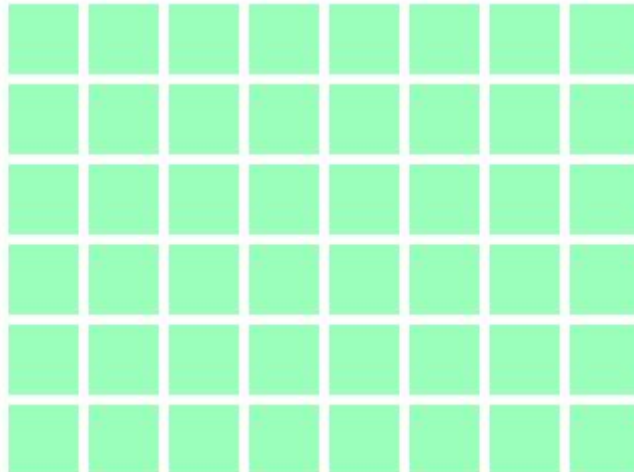
Test it!

Now if you go to full-screen mode, using the button under the Scratch logo at top-left, you should be able to drag the arrow around with the mouse.



## Dragging new arrows

But the player needs to be able to put more than one arrow on the grid to program their robot. We'll use clones for this. We'll have the original arrow off to the side:



and let the player drag clones of it onto the grid.

Whenever the original arrow is clicked, we'll make a clone, and *that clone* will be what follows the mouse pointer until the mouse button is up. So **change** the 'when this sprite clicked' script, and **add** a 'when I start as a clone' script:

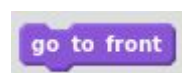


Test it!

If you try this, you should be able to drag lots of arrows off the original one and onto the grid.

## Clones should be in front

A small improvement: The clone appears 'behind' the original, which looks a bit odd. Put a block just under the 'when I start as a clone' to fix this.



## Moving the clones around on the grid

If you play with this, you'll notice another problem: If you put an arrow on the grid, and then try to move that arrow to a different grid cell, you get another arrow! We don't want this to happen. If you drag a clone, it should just *move* that clone, not make another one.

We need each arrow to know whether it's the original, or a clone. This is the same problem as we had when each grid-cell had to know which row and column it was in, and we'll use the same solution.

**Make a variable** so each arrow can remember whether it's the original or a clone:

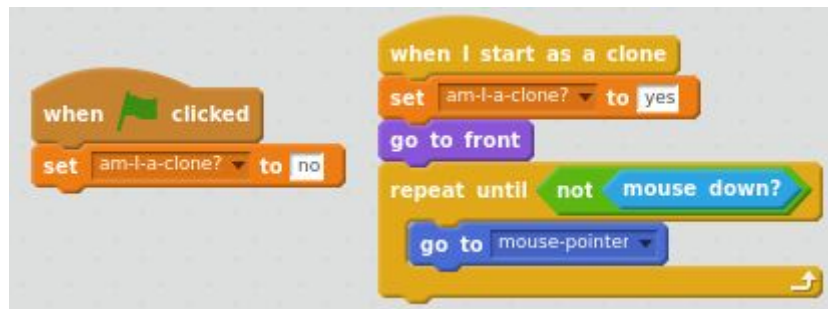
Variable name:

(Make sure you choose 'For this sprite only'.)

☐ For all sprites ☒ For this sprite only

We need the original to have "no" for this variable, and the clones to have "yes", so add blocks to make this happen.

**Add a 'when green-flag clicked' script**, and **change the 'when I start as a clone' script**:



(You should still have your 'when this sprite clicked' script.)

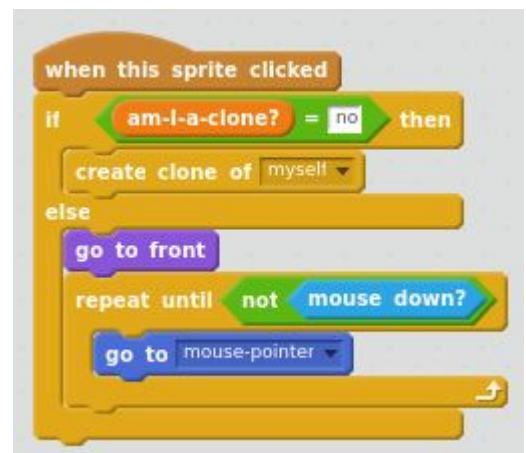
And then, when an arrow is clicked, we need to do two different things depending on whether it is the original or a clone that's been clicked:

- If it's the original, then create a new clone.
- Else it's a clone, so let the player drag that clone around.

We'll use the 'if / then / else' block for this.

**Change the 'when this sprite clicked' script** to the one at the right:

(Your 'when I start as a clone' script stays the same.)  
You can duplicate the required blocks from 'when I start as a clone' to save time.




Test it!

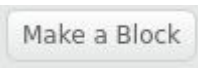
This should now be working — you should be able to drag new arrows onto the grid, and move arrows which are already on the grid. Go to full-screen and check it is working!

## Making our own Scratch block for 'drag and drop'

As your programs get more complicated, one good way to keep them understandable is to give names to parts of your scripts. You can see that we've used 'go to front; repeat until not mouse down; go to mouse-pointer' twice. We'll give a name to this set of blocks.

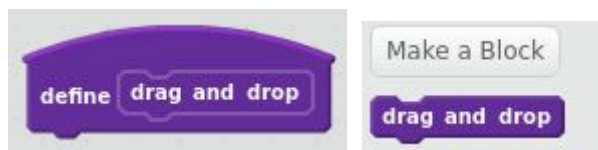
It's worth trying to think of a good name. This stack of blocks lets us drag the arrow around (by moving the mouse) and then drop it onto the grid (by letting go of the mouse button). So I'm going to call it 'drag and drop'.

To make your own block, use the  More Blocks section, bottom-right, and click

. Type 'drag and drop' into the purple block in the window which pops up:

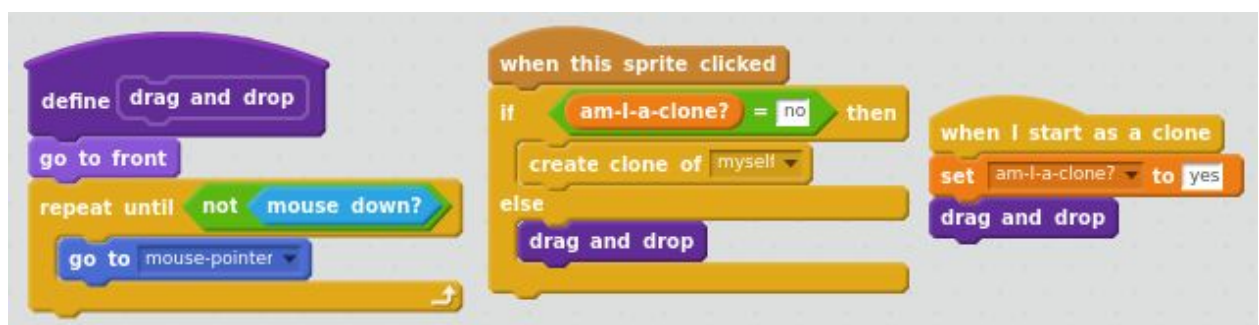
Click 'OK'.

Scratch will put a big purple hat-block into your scripts section, and a 'drag and drop' purple block into the 'More Blocks' palette:



The hat-block lets you 'define' what you mean by 'drag and drop', by connecting blocks underneath it. Then when you use your new 'drag and drop' block, Scratch will use the blocks in the definition. This is probably easier to understand once you see it working:

**Add blocks** to finish the 'define drag and drop' script, and **change** your 'when this sprite clicked' and 'when I start as a clone' scripts:



Your program is now much easier to read and work with.

Test it!

Check your game still works!

## Make arrow 'snap' to centre of cell


This is working quite well, but the player can drop an arrow anywhere on the grid, including across two or more cells. It will be very difficult to make sure the robot follows the correct arrows if it's not clear which cell the arrows are on. We want the arrows to always be in the centre of a cell. We'll add some extra blocks to our 'drag and drop' definition to fix this.




This will involve quite a bit of maths, so don't worry too much if you don't understand every detail of it. If you get the general idea, and are able to make it work in your project by copying the scripts in this sheet, then that's fine.

## Think about a simpler problem first

We'll work out what to do by thinking of a simpler problem. Suppose we wanted to let somebody put a coin on a ruler, but only at the marks 0cm, 10cm, 20cm, 30cm, etc. If they put it on 32cm, we'll move it to the closest allowed place, 30cm. How can we work out where to move the coin to once they've put it down?


We want to 'round to nearest 10cm'. Scratch does have a  block, but it rounds to the nearest whole number. We can work out 'how many steps of 10cm along the ruler' the person has put their coin by dividing:

$$(\text{number of 10cm steps}) = (\text{place on ruler}) \div 10\text{cm}.$$

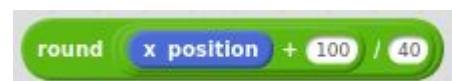
If they put the coin on 42cm, this will work out to 4.2 steps, which we can then  to 4 steps, and multiply back by the 10cm to get 40cm.

## Work out 'x' coordinate of nearest cell centre

Our grid is like this, but the steps aren't 10cm. In my project, one 'cell step' is 40 Scratch

units. Yours might be different. It's the number in the  formula the cell sprite uses.

Also, our 'ruler' doesn't start at zero, because we had to move the cells left and down to make the grid fit. So we need to undo the subtraction before dividing. 'Undoing subtracting' is 'adding', so to find out which cell the arrow is nearest to, we can start with the 'x' value the sprite is at, and calculate:



(The slash '/' is used for 'divide' in Scratch and many other computer languages.)

There are more calculations to do, so to stop this getting too complicated we'll make a variable to give a name to what we've done so far.

### Make a 'for this sprite' variable 'nearest-cell-x':

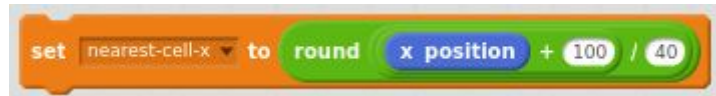
Variable name:

We'll set this variable to the calculation we've just worked out.

☐ For all sprites ☒ For this sprite only



**Make this block, ready for use in a script later:**



And then to turn this back into a Scratch-style 'x' number, we do the same 'multiply by 40 and subtract 100' as we used for the cells.

**Make this block, ready for use in a script later:**



(Remember the star '\*' is 'multiply'.)

Be very careful you get the right green blocks inside each other!

## Work out 'y' of nearest cell centre

In the same way, we can work out what 'y' number we need to nudge the arrow to for it to be in the centre of a cell vertically. It's very similar, except you might need a different number instead of the '100', depending what you used for your cells. For my project, I needed '120'.

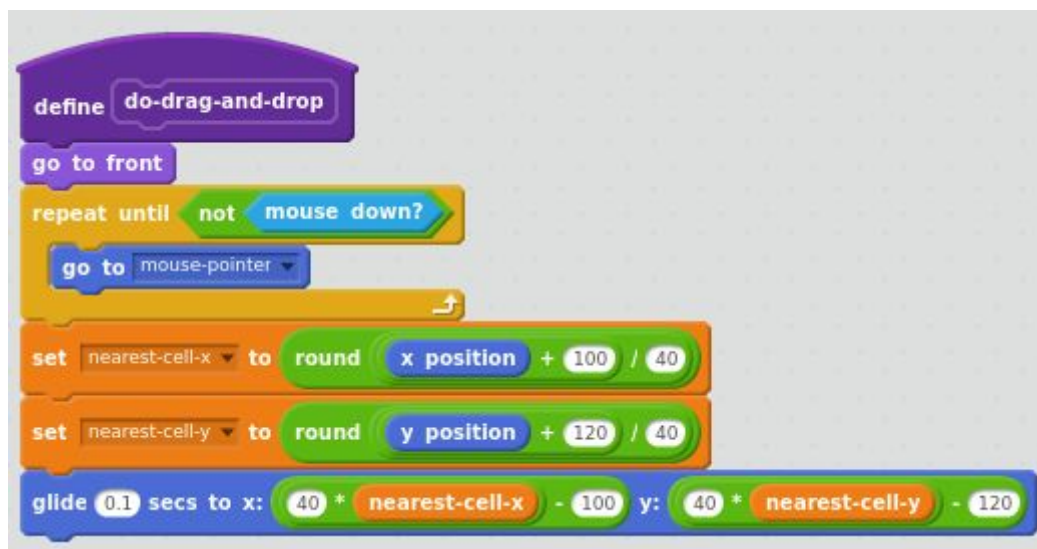
**Make a 'nearest-cell-y' variable, and then make these two blocks, ready for use later:**



## Nudge the arrow to its nearest cell's centre

Now put this all together!

**Add three blocks to the bottom of our definition of 'drag and drop', to make it:**



You've already made the 'set nearest-cell-x' and 'set nearest-cell-y' blocks, and the green round-ended blocks for the 'glide' block.

Here I used a 'glide' of 0.1 seconds, but you could instead use 'go to' if you want the arrow to go immediately to its cell centre.


This is another good reason to have made our own 'drag and drop' block — we only have to have one copy of this very complicated script in our sprite.

## Arrows in other directions

You can now duplicate the arrow sprite, and choose a different costume for each one, to get all four direction arrows. (The library 'Arrow 1' sprite already has all four costumes.)

Give the sprites sensible names!

## Challenge: Dropping an arrow outside the grid deletes it

At the moment, the player can drop the arrow outside the grid. Can you make it so that if the player does this, then that arrow disappears? You will need the  block.

## Key points

Use 'repeat until' to keep doing something until a questions gets the answer "yes".

Use the 'not' operator block to turn "no" into "yes" and vice versa.

Make clones behave differently to the original sprite, with if/then/else.

Create a custom block to make our program easier to understand and work with.

Use the 'round' block, with other arithmetic blocks, to find the nearest cell.

---


## Extra information (not part of our project)

### For other projects

**Don't do this for our arrow**, but sometimes you can use a simpler way of letting the player drag a sprite round the stage. There is a 'can drag in player' tick-box, which you can work with using the blue circled 'i' of a sprite:

Click here on the blue 'i' to get to:



Tick 'can drag in player', and then click the  at top-left to go back.

This won't work for the clones we will need to work with shortly, which is why we wrote our own scripts.

## About this document

Main content is copyright 2017 Ben North, and is hereby licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/). This and other worksheets available at <https://bennorth.github.io/scratch-worksheets-2017-2018/>.

Images contain material copyright The Scratch Team, used under a [Creative Commons Attribution-ShareAlike 2.0 license](https://creativecommons.org/licenses/by-sa/2.0/).

Scratch is developed by the Lifelong Kindergarten Group at the MIT Media Lab. See <http://scratch.mit.edu>.