Getting started in Scratch: Hello World and a Maze Game

Intro to Scratch and “Hello World”

Hi everyone welcome to Scratch!

You may have heard of Scratch before since it has become one of the most popular first ways to teach programming.

In this booklet we’re going to be walking you through an introduction to Scratch and give a number of worked examples of making games. We’re going to be making

- our own version of the famous first program “Hello World”
- a maze game for trolling your friends
- a clone of the infamous meme game *Flappy Bird*
- a platformer with scrolling and obstacles
- a shoot’em’up with multiple waves of enemies and powerups
- an interactive story with multiple characters and scenes

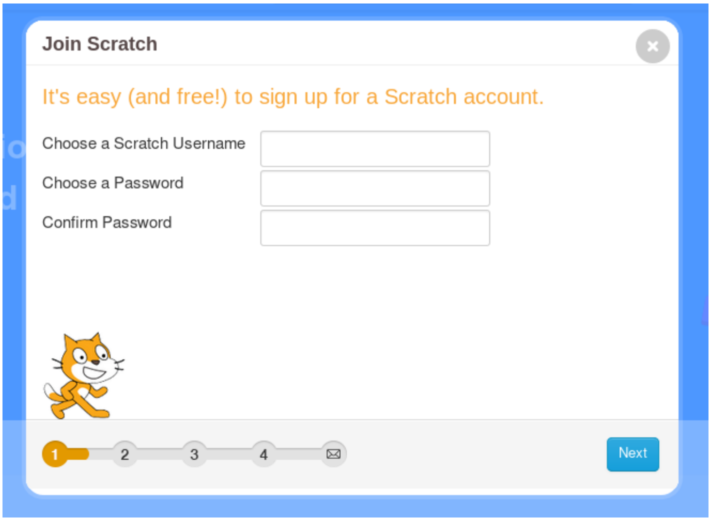
First, our “Hello World”!

“Hello World” is tradtionally the first program you learn to make in a programming language, going all the way back to the 1970s. The point is just to make a running program that displays the text “Hello World”.

Our Scratch version of this famous exercise is going to give us a little program where you can move the Scratch cat around and press the space bar to make the cat say “Hello World”!

Making an account

To start, go ahead and make a Scratch account if you don't have one by clicking on `Join Scratch`.

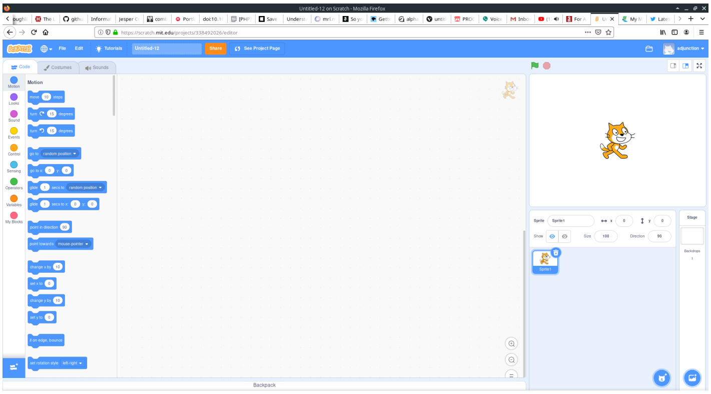


Join Scratch

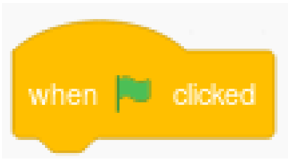
It's easy (and free!) to sign up for a Scratch account.

Choose a Scratch Username	
Choose a Password	
Confirm Password	

Once you have your account, click on `create` to start making your first program. You should be looking at a screen like this



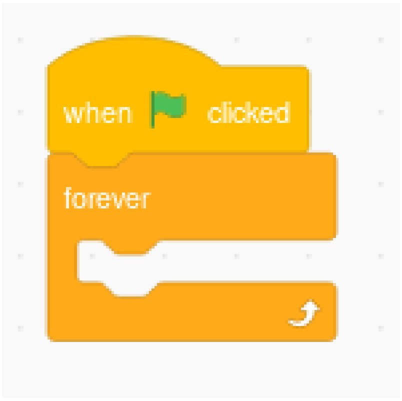
We're going to start our program with the "when green flag clicked" block



Everything we attach to this block is going to happen every time we hit the green flag at the start of the program.

Now, we need to make the part that listens for our button presses.

The first thing is to go to the `Control` section and then grab a block called `forever` and connect it to the first block. You should have something that looks like this:

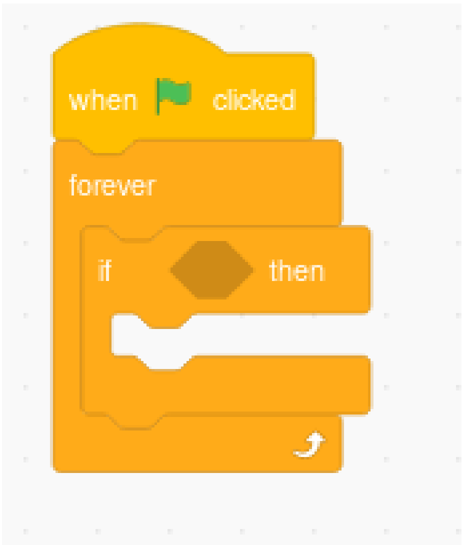


What this block does: it takes more code inside it and runs that code over and over until you stop the program. We're going to put the rest of our code inside it so we can move the cat around or display our message whenever we want and *not just once.*

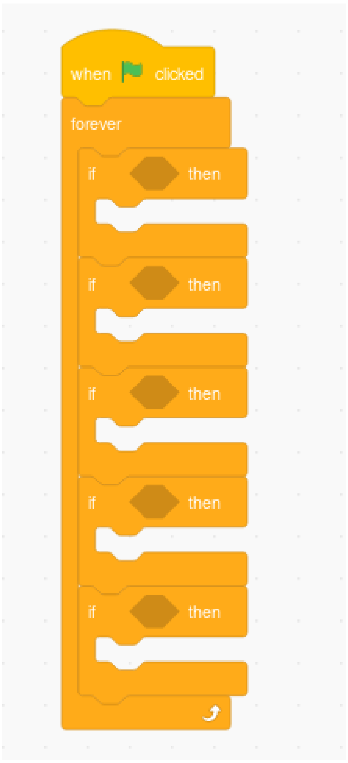
So how do we tell if a key is being pressed in Scratch? To start we need to know how to ask a question in our program.

In Scratch, like most programming languages you ask questions with something called `if`.

So grab the block labeled `if then` under the `Control` section.



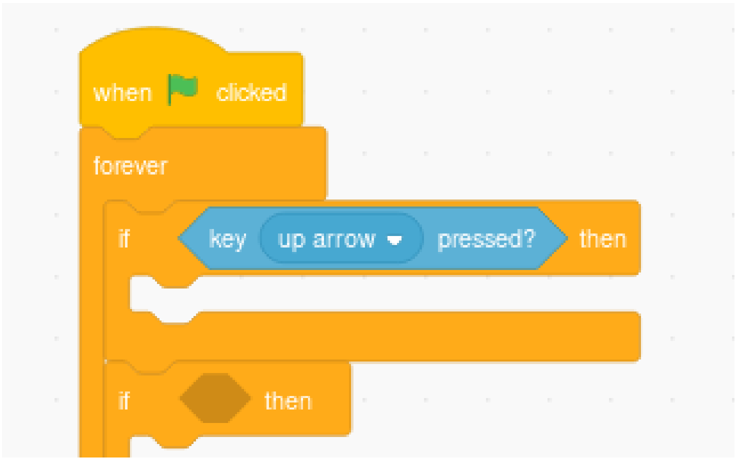
In fact, we're going to need four more of those `if then`s underneath the first one so it looks like



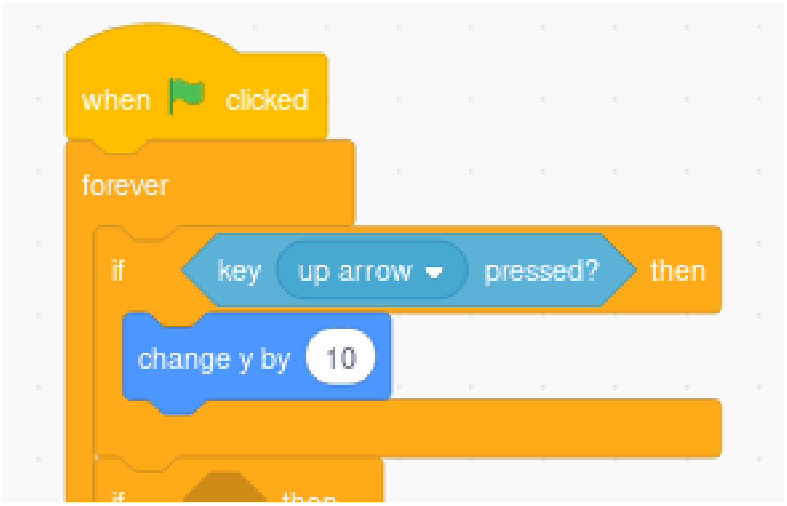
The diamond shape space between the `if` and `then` is where you put the question you want to ask the program. In this first case it's going to be the question

"Is the up key pressed?"

To ask this question in Scratch we'll grab a block in the `Sensing` section that says `key space pressed?` and place it in the diamond spot of the first `if`, then click the black triangle to change it so it says `up arrow`.



To make our cat move when the up key is pressed we need to put code that moves our cat *up the screen.* This bit of code can be found in the `Motion` section. We need to change the *y-position* of our sprite, since the y-position is the up-and-down position on the screen. Put the `change y by 10` block inside the first `if`.



We can already test our code even without filling out the rest.

Go ahead and hit the green flag and test your code. You should be able to press the up key in order to move your cat up.

Does it work? If it doesn't check your code against the screenshot.

Now, hit the stop button to end the program.

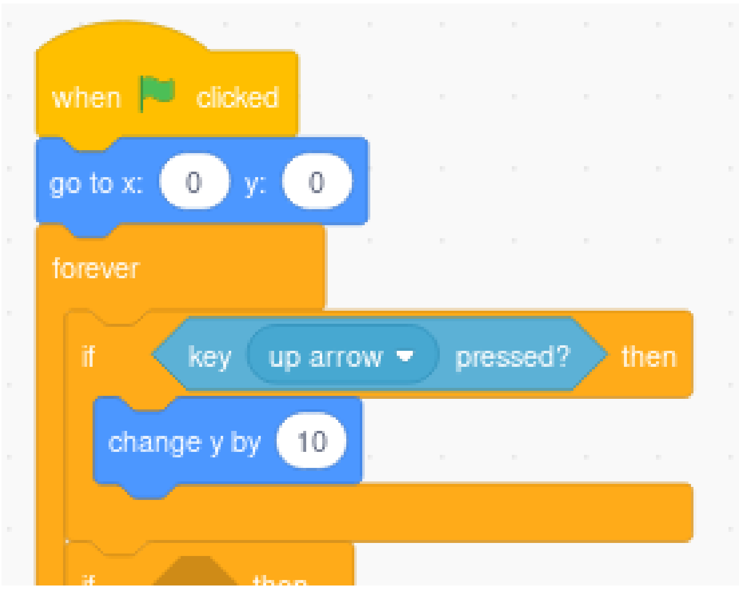
Notice anything annoying? Like the fact that the cat doesn't go back to the center of the screen?

Here let's try adding some more code to reset the program each time, this time using the `go to x: 0 y: 0` block under the `Motion` section and placing it just before our `forever` block.

Question:

Why do we need to place the `go to x: 0 y: 0` block *before* the `forever`? What would happen if you put it inside the `forever`?

Your code should look like *this*

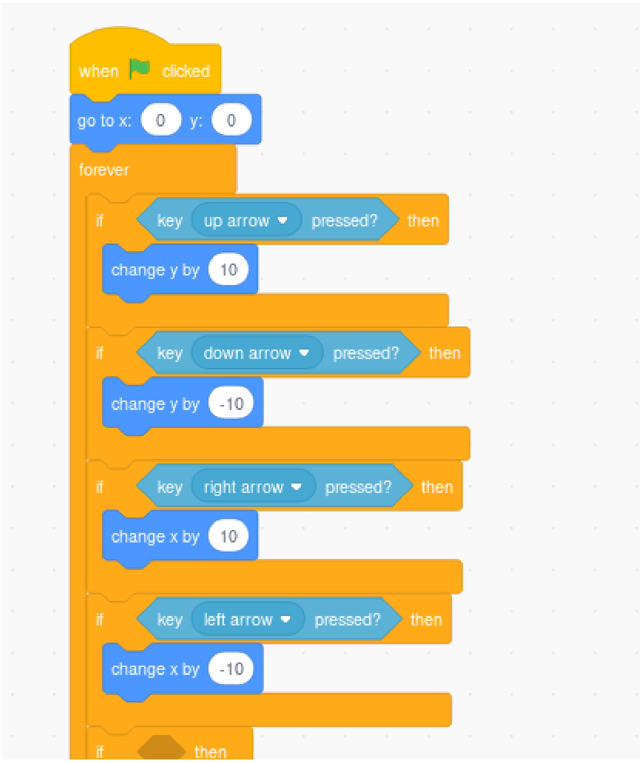


```
when 🏁 clicked
go to x: 0 y: 0
forever
    if key up arrow ▾ pressed? then
        change y by 10
    if then
```

Things to try:

Change what position the cat goes to at the start of the program. Try sending it to a random position. Can you find a block to help you do that?

Given our example of the up key can you fill in the code for the rest of the arrow keys. If possible, give it a try before checking the next screenshot

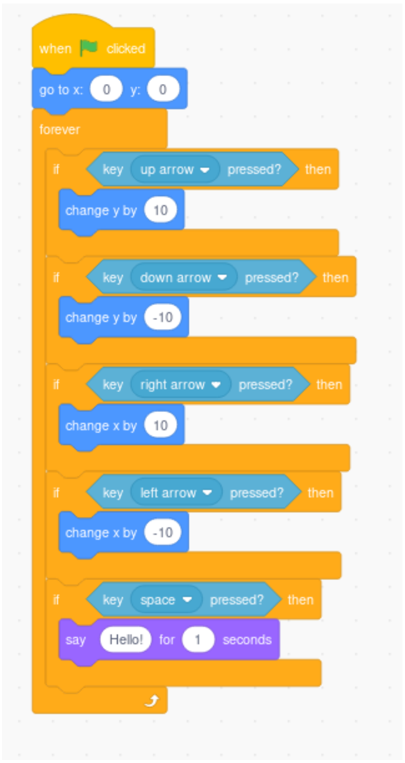


```
when 🏴 clicked
go to x: 0 y: 0
forever
  if key up arrow ▾ pressed? then
    change y by 10
  if key down arrow ▾ pressed? then
    change y by -10
  if key right arrow ▾ pressed? then
    change x by 10
  if key left arrow ▾ pressed? then
    change x by -10
  if  then
```

Now we need to finish our program by filling in the last key: the space bar. Place one last `key space pressed` inside the diamond of the final `if` and go to `Looks` to find a block called `say Hello! for 2 seconds`. Place that inside our last if statement and change the text to something else, like “Hello World!”, and change the time to something like 0.5 seconds instead.

Click the green flag to start your program. Does it work the way you want?

If not, check the screenshot below!



```
when 🏴 clicked
go to x: 0 y: 0
forever
    if key up arrow ▾ pressed? then
        change y by 10
    if key down arrow ▾ pressed? then
        change y by -10
    if key right arrow ▾ pressed? then
        change x by 10
    if key left arrow ▾ pressed? then
        change x by -10
    if key space ▾ pressed? then
        say Hello! for 1 seconds
```

Things to try:

- Add a sound effect every time you hit the space button, using blocks from the `Sound` section
- Add code to your `forever` loop that spins the cat around slowly
- Change your code so that instead of saying a message when the space bar is pressed the cat says a message when the mouse cursor is touching it

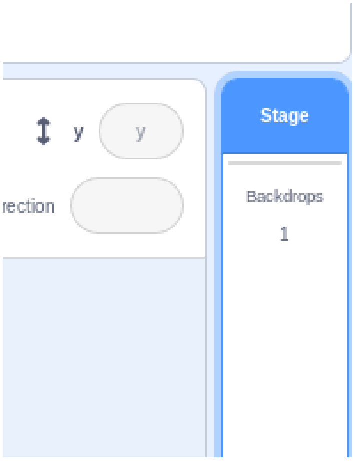
A maze game

The next game we're going to make is a *maze* game. In this section we're going to learn how to

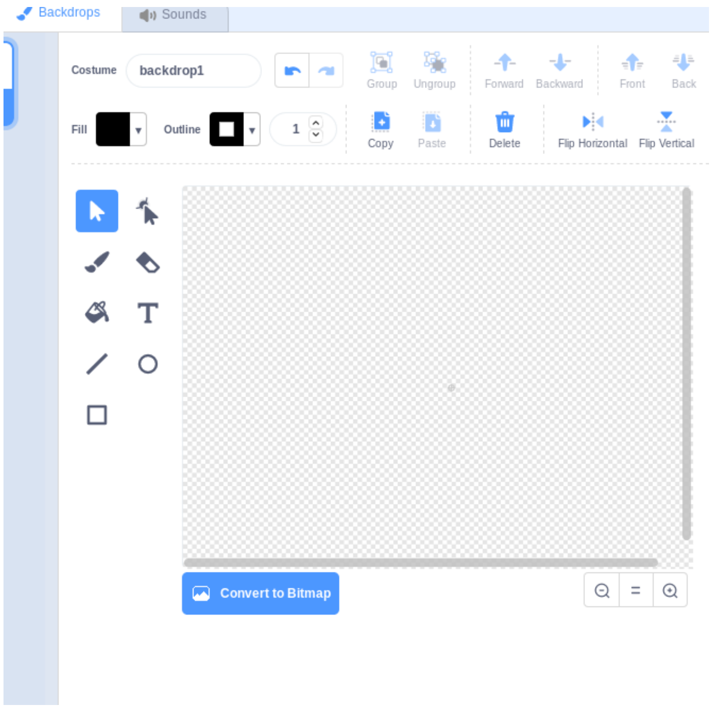
- make mazes with the backdrop drawing tools
- make the walls of our mazes solid to our player
- transition between levels after completing each maze

Let's start a new project. The first thing we're going to do is draw our first maze.

Click on the bar all the way on the right-hand side of the screen that says "Stage" like in the screenshot below



and then on the tab that says "Backdrops" in the upper left, so that your screen looks like this

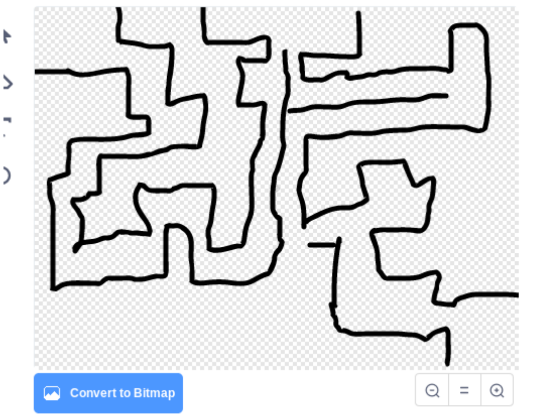


We can start either by selecting the rectangle tool, which is the one that looks like a square, and drawing our maze with big solid pieces *or* we can use the brush tool and freehand draw the maze. I'm going to do the latter, myself!

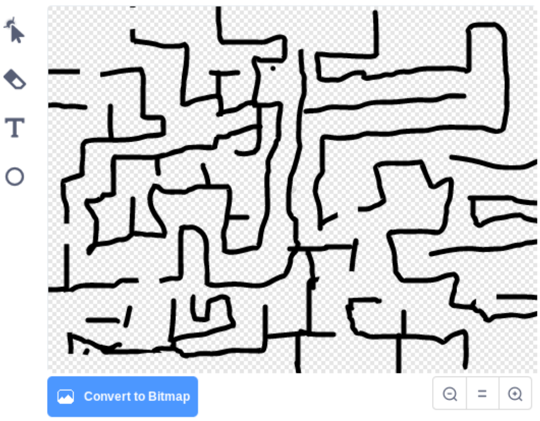
One tip for drawing a maze is to essentially draw a big pipe from the beginning to the end of the maze, so that you know it's solvable, and then draw extra lines or erase walls to add to the *subterfuge* of it all.

I already know that I want to make the upper-left corner the start and the lower-right corner the end of the maze.

So I started by drawing this



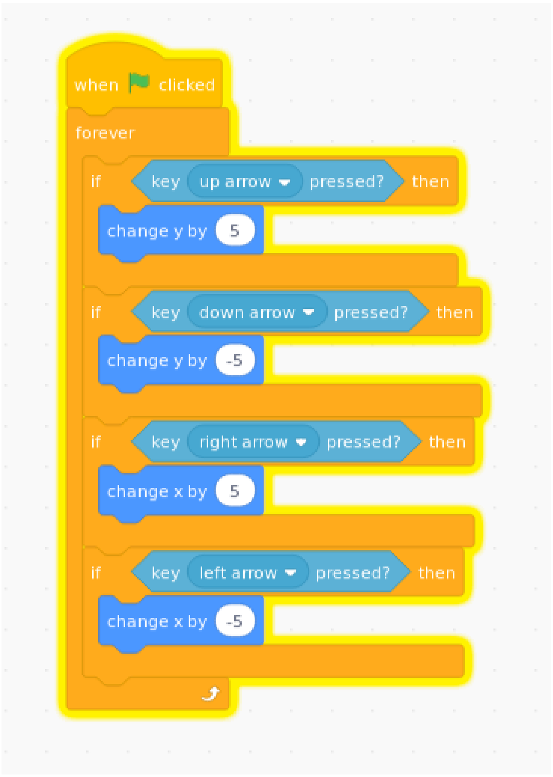
and I ended with this



We're at the point of writing code to make the cat move and be blocked by the walls.

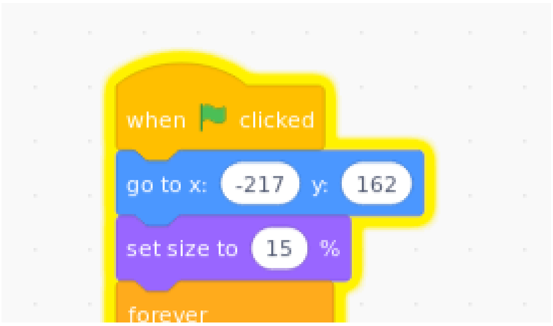
To start we put in our basic 4-directional movement code using a combination of the

- `when green flag clicked`
- `if then`
- `is key pressed?`



```
when 🏴 clicked
forever
    if key up arrow ▾ pressed? then
        change y by 5
    if key down arrow ▾ pressed? then
        change y by -5
    if key right arrow ▾ pressed? then
        change x by 5
    if key left arrow ▾ pressed? then
        change x by -5
```

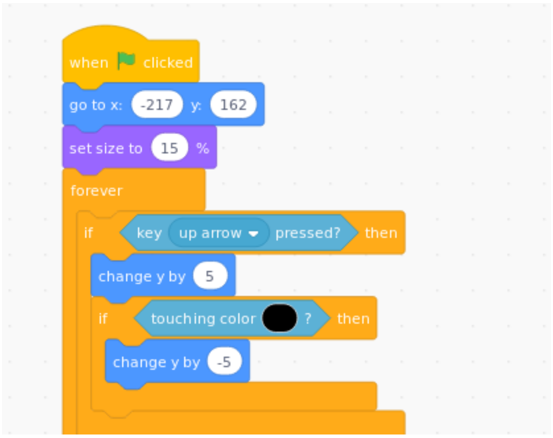
If you run your game you should be able to verify that your sprite moves. Ugh. It's way too big, though, isn't it? Let's make it smaller at the start of our game and reset it to the start of the maze every time we hit the green flag, like this



```
when [flag] clicked
go to x: -217 y: 162
set size to 15 %
forever
```

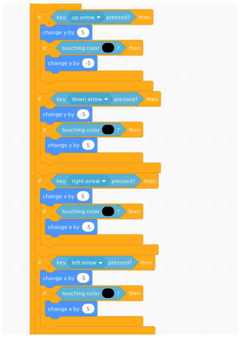
To make the walls solid, first note that we've drawn our maze to be black. We're thus going to check whether our sprite is touching the color black to determine if we're pushing up against the wall and, if we are, undo the movement we just attempted.

Now we're going to put new `if`s inside of our current `if` and add the check that we're not touching the wall. It'll look like this



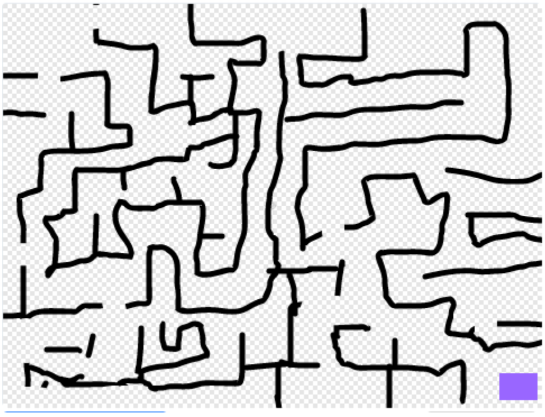
```
when ⚑ clicked
go to x: -217 y: 162
set size to 15 %
forever
    if key up arrow ▾ pressed? then
        change y by 5
        if touching color ● ? then
            change y by -5
```

Given this example, do you think you can fill in all the other directions? Try it and then check your answer against the next screenshot

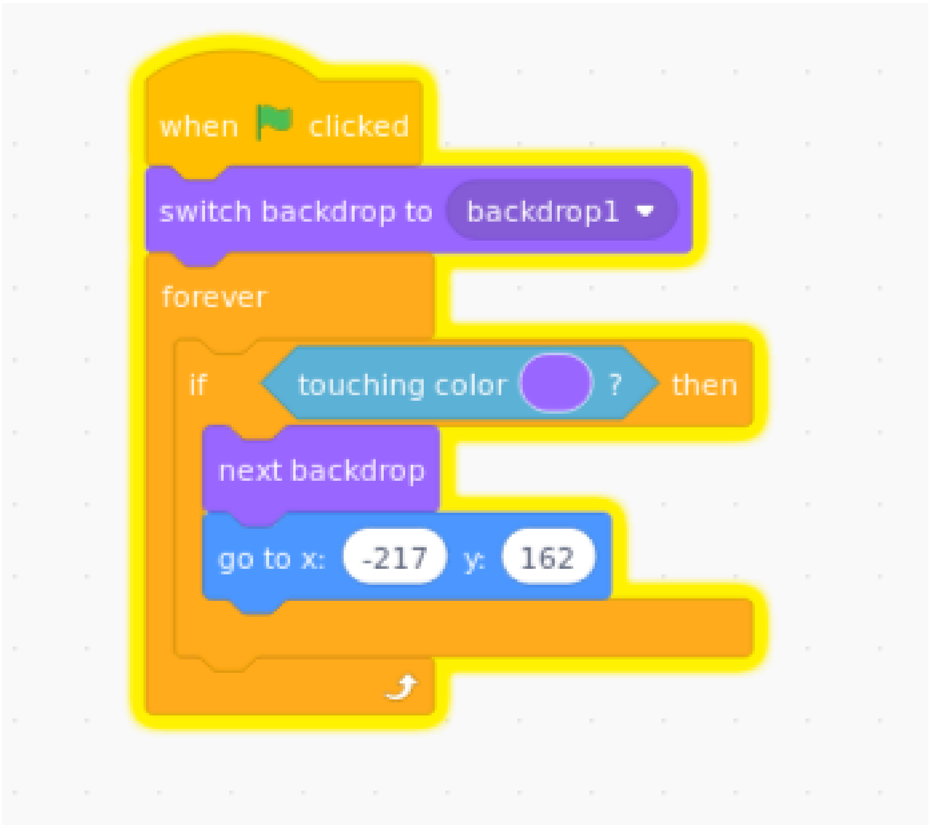


```
    if key up arrow ▾ pressed? then
        change y by 5
        if touching color ● ? then
            change y by -5
    if key down arrow ▾ pressed? then
        change y by -5
        if touching color ● ? then
            change y by 5
    if key right arrow ▾ pressed? then
        change x by 5
        if touching color ● ? then
            change x by -5
    if key left arrow ▾ pressed? then
        change x by -5
        if touching color ● ? then
            change x by 5
```

The last things we need to are add the code so that you can switch levels! That's going to be pretty easy. First, place a rectangle of color at the end of the maze. It doesn't matter *what* color exactly just make sure it isn't the same color as the maze walls.



Finally, we're going to add a *new* loop starting with the `when green flag clicked` block and a `forever` and check to see if our sprite is touching a specific color and, if it is, we switch to the next backdrop and reset our position back to the beginning. As this is our last coding challenge, see if you can do it without looking at the next screenshot!



```
when [flag] clicked
switch backdrop to (backdrop1 ▾)
forever
    if <touching color ( )?> then
        next backdrop
        go to x: (-217) y: (162)
```

Now, all you need to do is add new backdrops and your whole maze game will keep working correctly.

Things to try:

- Add a “You win screen” to congratulate someone who can get through all your troll-y mazes
- Add a *jumpscare* to your maze for when you finish a level
- Ultimate troll hard mode: instead of having the walls block the player, rewrite your code so that touching a wall sends you back to the beginning!