# CODERDOJO TRAMORE 2022 – SCRATCH TUTORIAL OCTOBER 2022

## How to code your own Platformer game using Scratch!

This tutorial is a written break down of the beginner's guide to writing your own platformer game in scratch, as posted by the amazing Griffpatch on youtube.

This tutorial will describe the steps you need, but if you would like to see the original video at any point the link is as follows: -

https://youtu.be/D16hTnDGweo

GriffPatch has a starter project in Scratch Online that you can remix into your own project space. It contains the basic assets for your starter Platformer game.

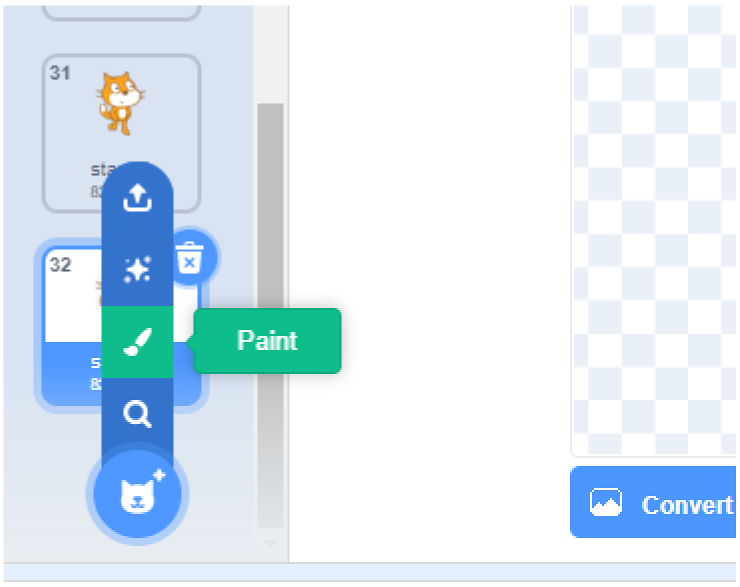
Click here to view it:

Platforming Resources (YouTube) on Scratch (mit.edu)

click the "Remix" to copy this starter project to your own project space. You will need to be signed in to Scratch online with your own account for this.

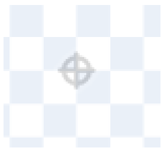
Now to begin we will make a new simple non-animating costume for our player sprite. Add it as a new costume at the end of the "Resources" sprite. And you can rename this sprite to "Player".

Start Painting a new costume using "Paint" in the new costume menu:

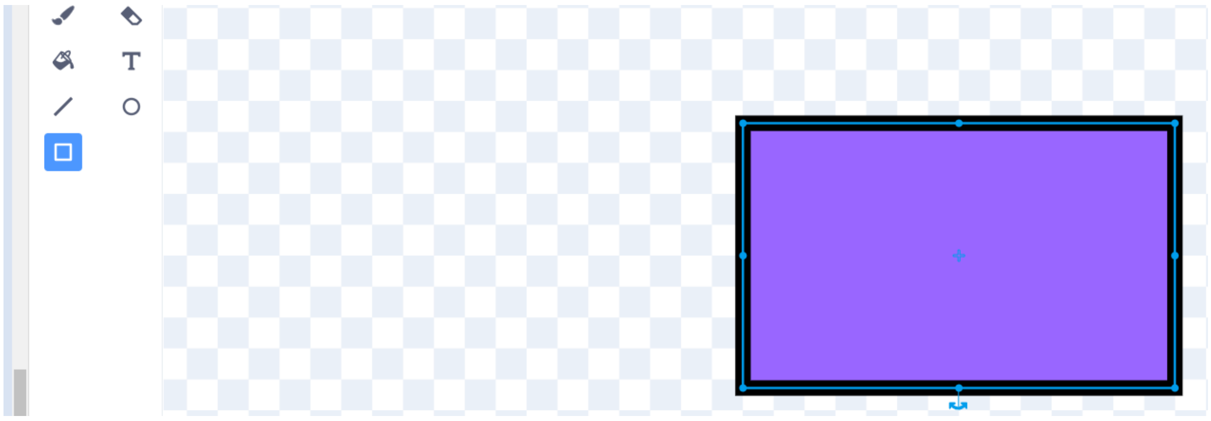


Let's make our initial player just a square block with eyes!

1st draw a purple square with a black outline. But make sure to keep the square exactly centered on the + center marker in the middle of the painting area: -



And it should be about 7 x 7 of the grey and white squares, that is 7 or 8 across and same down: -

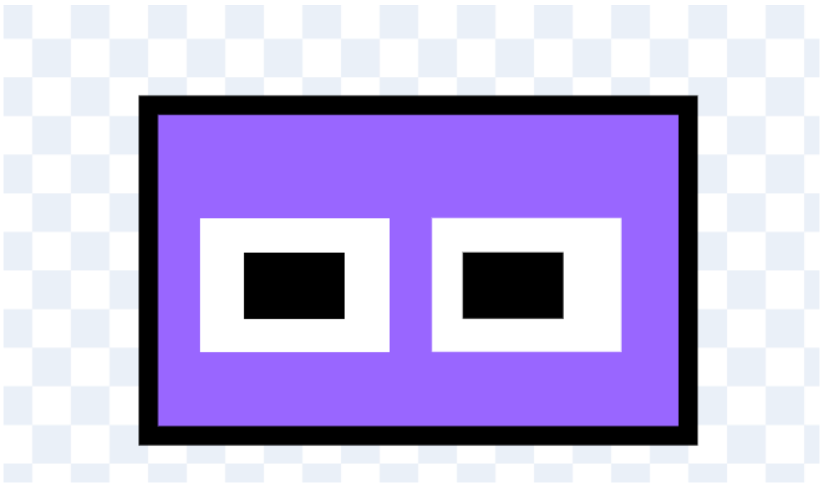


A tip to get it exactly on the center of the paint area – if you click on it to lift it up again, the center shows through it and you can align it with the center of the square.

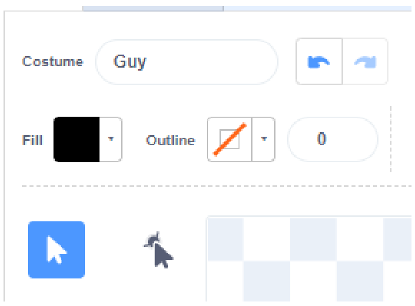
Ok now draw two white squares (turn off the fill border or set to white) for eyes and place two smaller black squares for the pupils and our basic player sprite is ready to go!

Tip: draw one white square and then use copy and paste to duplicate one the exact same size, you can do the same technique for the pupils also: -

Player sprite when you're done should look something like this: -



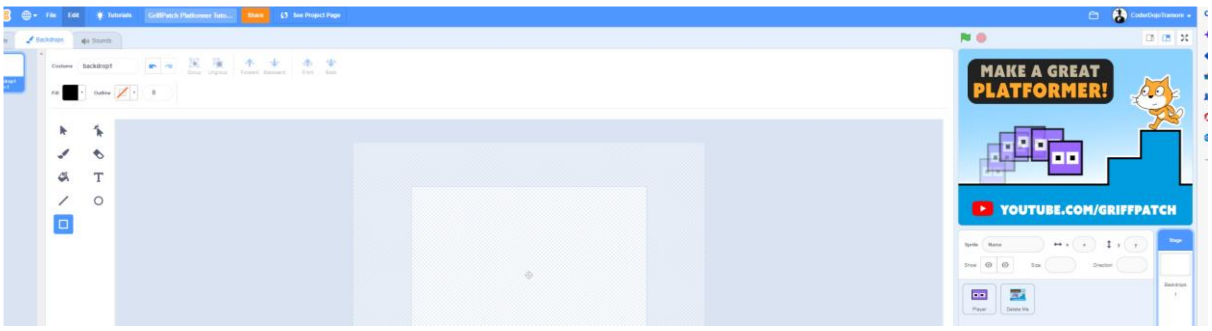
Let's rename this new costume to .. "Guy"



Now the actual platform levels can be done using EITHER sprites or backdrops.

To start with, griff shows how to do it using backdrops.

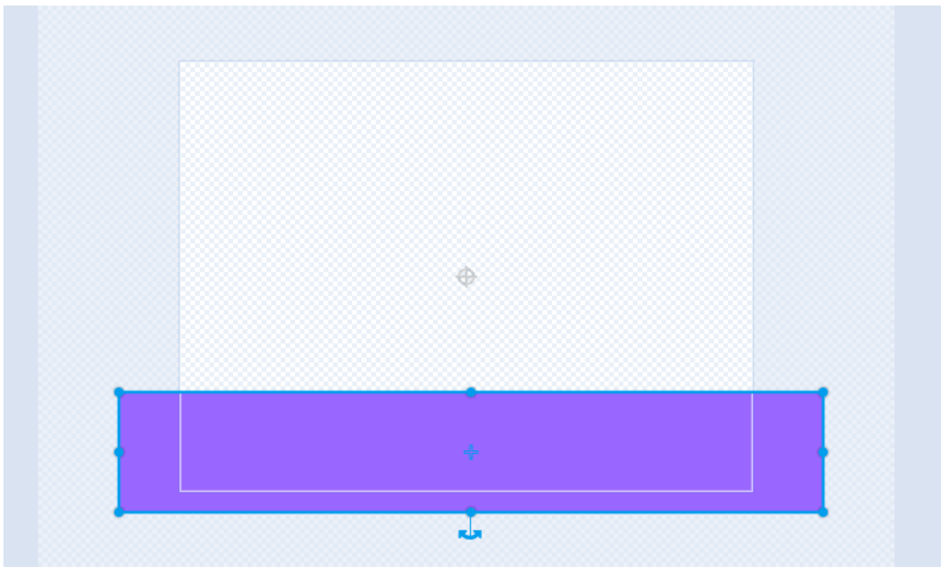
So click on the stage section, select the backdrop tool and select the “square” tool: -



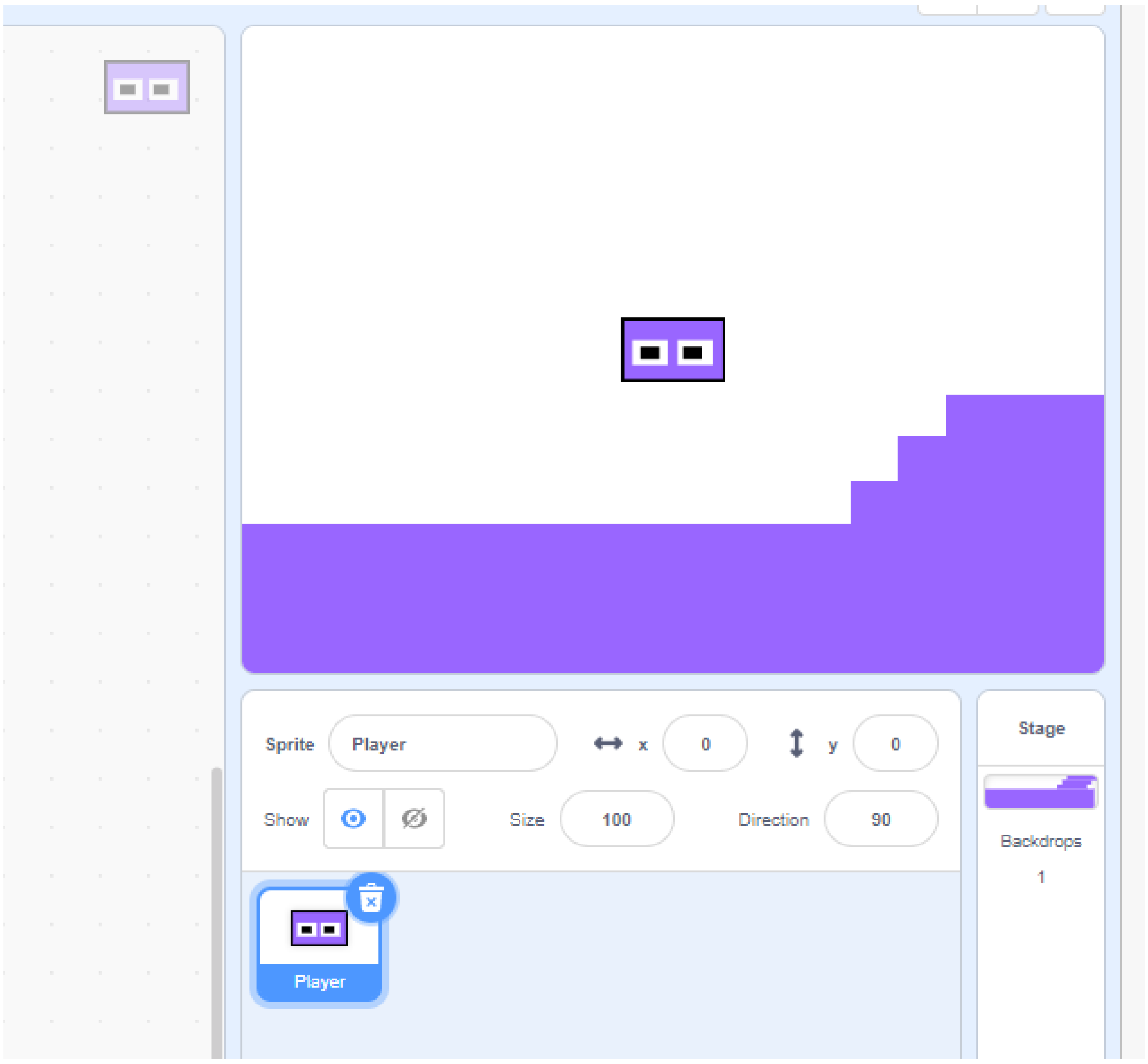
And draw a nice starter “floor” for our level: -

[Tip – to get the colour back the exact same purple as Guy sprite, you can go back to Costumes and instead of selecting a colour from the picker you can use the eyedropper and touch the body of Guy – this switch of colour to purple will persist even when you then go back to the backdrop builder]

Its ok to go outside the bright area – the bright area is the visible area when the backdrop is drawn on screen: -

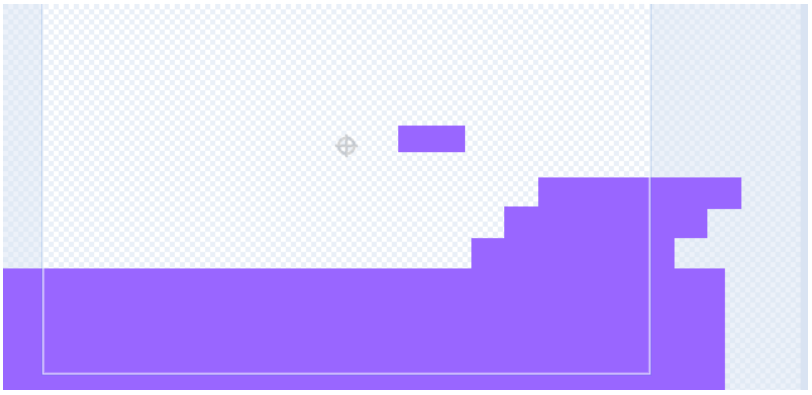


Draw one narrow rectangle to make the 1st step of some stairs, and copy and paste it to a 2nd and 3rd step also – arrange them like this; -



You can also delete the “Delete Me” sprite – this will allow you to see player on screen along with your backdrop.

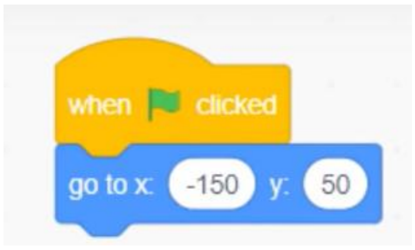
Back to the level design, finally let’s add one floating platform – just to test our skill!



Now we can start to code!

Select the Player sprite and the “Code” tab.

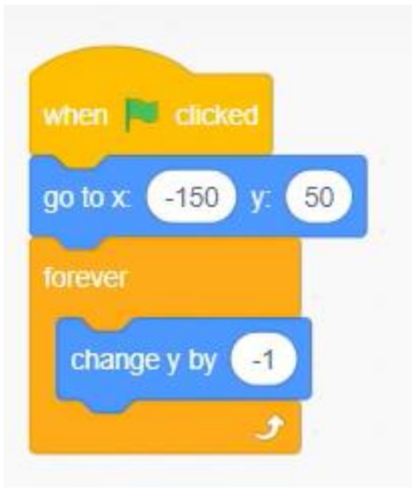
Add a Green Flag block. And set the players starting position to x -150 y 50.



```
when ⚑ clicked
go to x: -150 y: 50
```

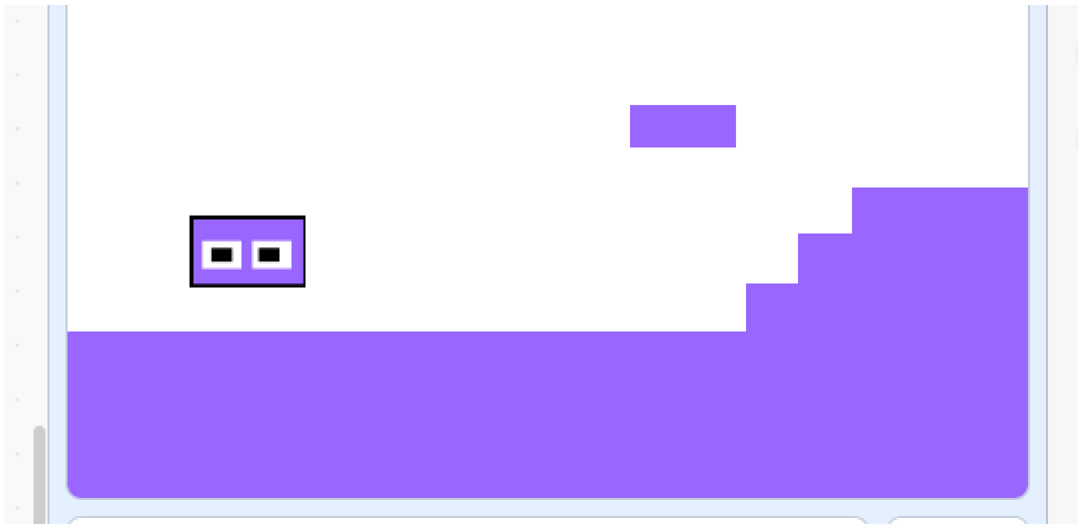
Now we also need to model some gravity to make Guy fall down to the floor to begin the game.

Let’s start with a Forever Loop and simply for now reduce Y by 1 each loop (we need to change by -1 to do this). In scratch, making the Y position value of a sprite smaller moves it **down** the screen.



```
when ⚑ clicked
go to x: -150 y: 50
forever
    change y by -1
```

Click the green flag and you should see Guy very slowly move down the screen at a constant rate (and through the floor at the moment).

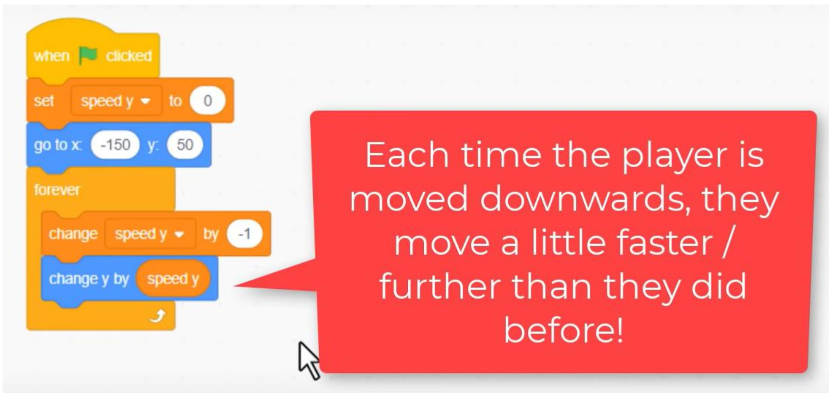


But that’s not how Gravity works in the real world! – Gravity is **an accelerating force** – what that means is your speed **increases** every instant as you fall and to make things more realistic we need to model this now.

So let’s make a variable to keep track of Guy’s vertical speed.

Let’s call it “speed y” and set it to be “For this sprite only” as it is just needed for Guy’s speed and nowhere else.

Then add the following blocks as shown – speed y is set to 0 initially but decreased by 1 each loop (think of it as bigger in a downwards direction) and y is then changed by speed y each loop also:-


Each time the player is moved downwards, they move a little faster / further than they did before!

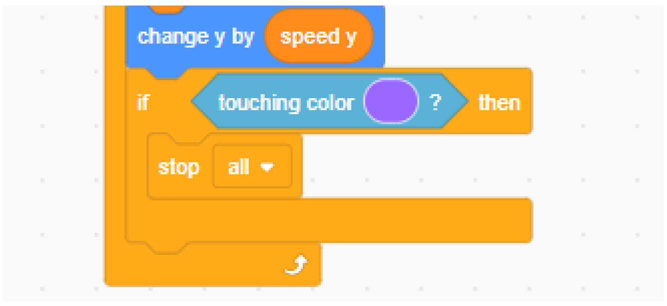
```
when [flag] clicked
set speed y to 0
go to x: -150 y: 50
forever
    change speed y by -1
    change y by speed y
```

Now if you click the Green Flag, Guy starts not moving then gradually accelerates right through to the bottom of the screen!

But of course.. we want him to land on the floor, not fall through it! So let’s fix that next..

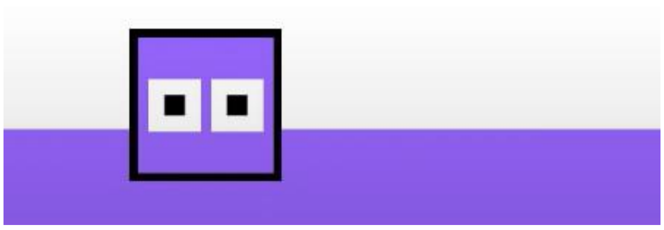
So let’s try the”touching color” block in sensing and place it in an IF that stops the script completely once it is true and let’s see what happens: -

Add this code to the bottom of the forever loop, you can use the colour picker in the touching colour block to exactly match it to your level colour (which should be all the same colour!): -



```
    change y by speed y
    if touching color ? then
        stop all
```

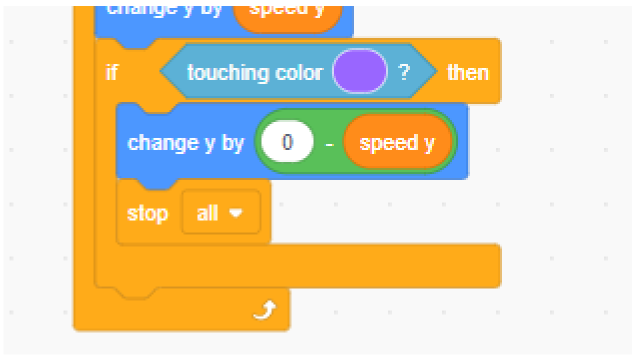
Run this and you should see that Guy actually only stops when he is buried in the ground!!



So we need to cater for this and fix it somehow.

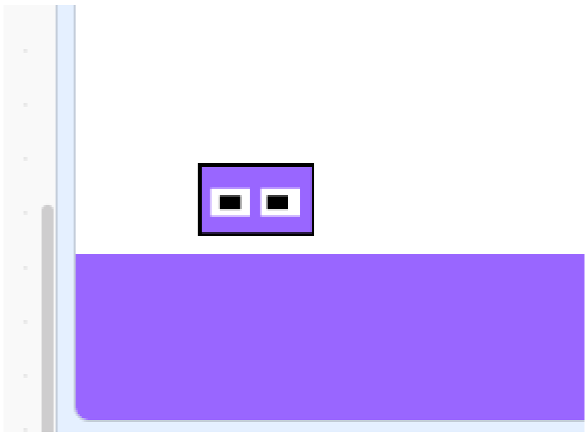
So we might say, well let’s move back by the last amount of “speed y”. Let’s try that.

Before stopping, add the last speed y value back to y. As y is negative, the easiest way to do this is to to change y by (0 – speed y), so add this then as below and try it again: -



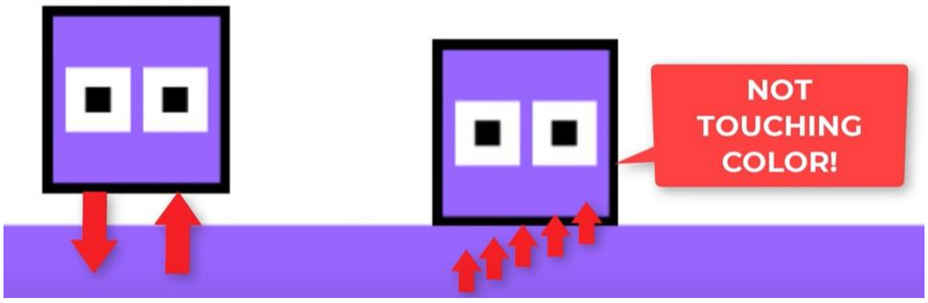
```
change y by speed y
if touching color ? then
    change y by 0 - speed y
    stop all
```

You should see that Guy is now back of the ground – but now he’s hovering -!



So the amount that we move him back up is very dependent on how fast he was travelling just before he touched the floor – if he fell from higher the gap to the floor would be even bigger. So we’re not quite there yet.

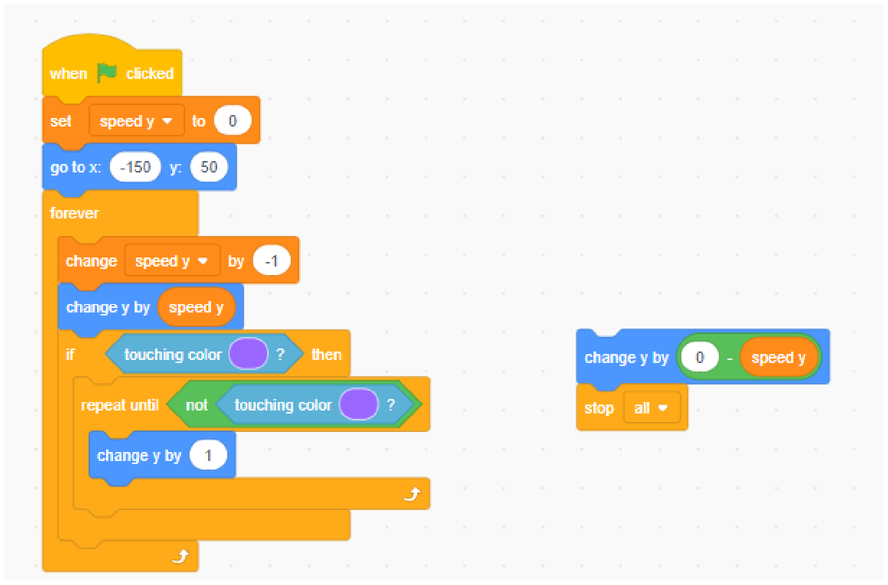
We need to make our technique a little fancier.. and what Griff has dreamt up is we move our player a very small amount at a time, and keep going until our player is no longer “touching colour”



So let's replace the change Y and stop all with instead a repeat loop that repeats until Guy is NOT touching the level colour. And each loop lets increase Y by 1 pixel.

So change the code to this: -

TIP – you can right click on the touching colour block in the IF and select duplicate. This is quicker and saves you having to set the colour again.



```
when 🏴 clicked
set speed y ▾ to 0
go to x: -150 y: 50
forever
    change speed y ▾ by -1
    change y by speed y
    if touching color ● ? then
        repeat until not touching color ● ?
            change y by 1

change y by 0 - speed y
stop all ▾
```

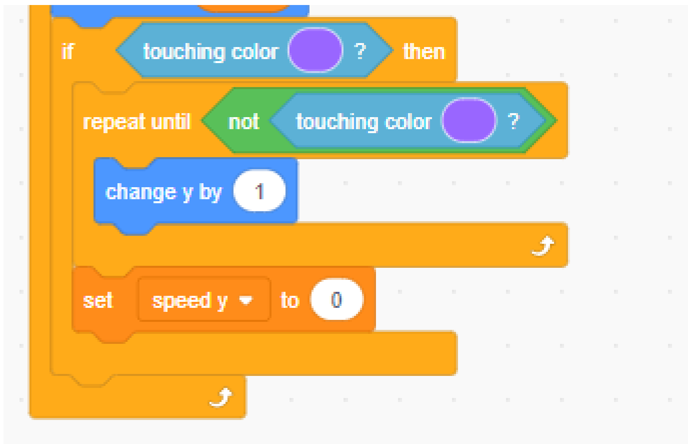
Right let's run that and see what happens now..

What you should see is Guy slowly rises out of the floor now, but then he falls back in over and over, like quicksand.

The reason for this is we still have the downward speed y. We've hit the ground so we should no longer have any vertical speed, this means speed y should now be SET TO ZERO.

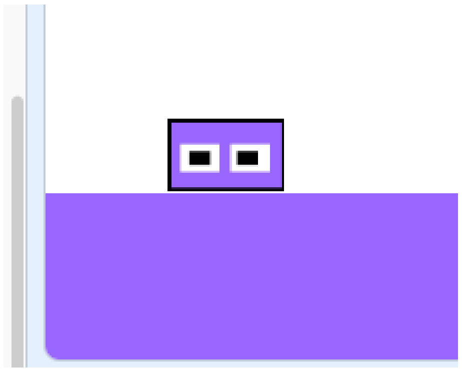
Let's add this and see are we in business: -

Note: it's inside the IF block:



```
if <touching color ( ) ?> then
    repeat until <not <touching color ( ) ?>>
        change y by (1)
    end
    set [speed y v] to (0)
end
```

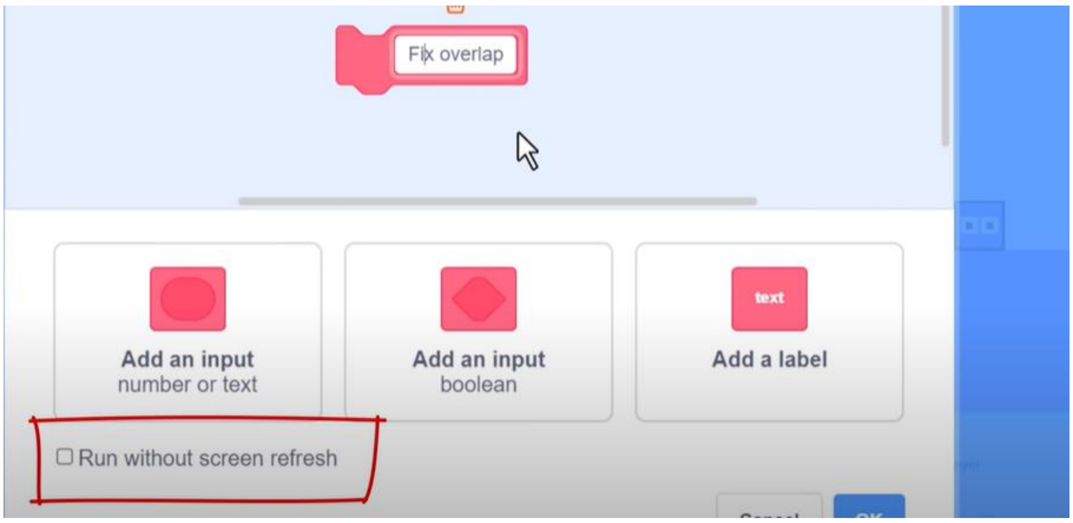
OK that's good, now Guy springs back up on to the floor: -



But the next problem is he moves very slowly back up. Can we improve this?

The main trick whenever we have a screen update that is happening slowly and we want it to happen almost instantaneously, is to use a **custom block**. And this is because within custom blocks you can set an option so that the block doesn't actually update the screen while it is executing.

If you haven't used them before custom blocks are created in the "My Blocks" section. Create a new custom block and call it "Fix Overlap" and most importantly select the **Run without Screen Refresh** checkbox: -



Fix overlap

Add an input
number or text

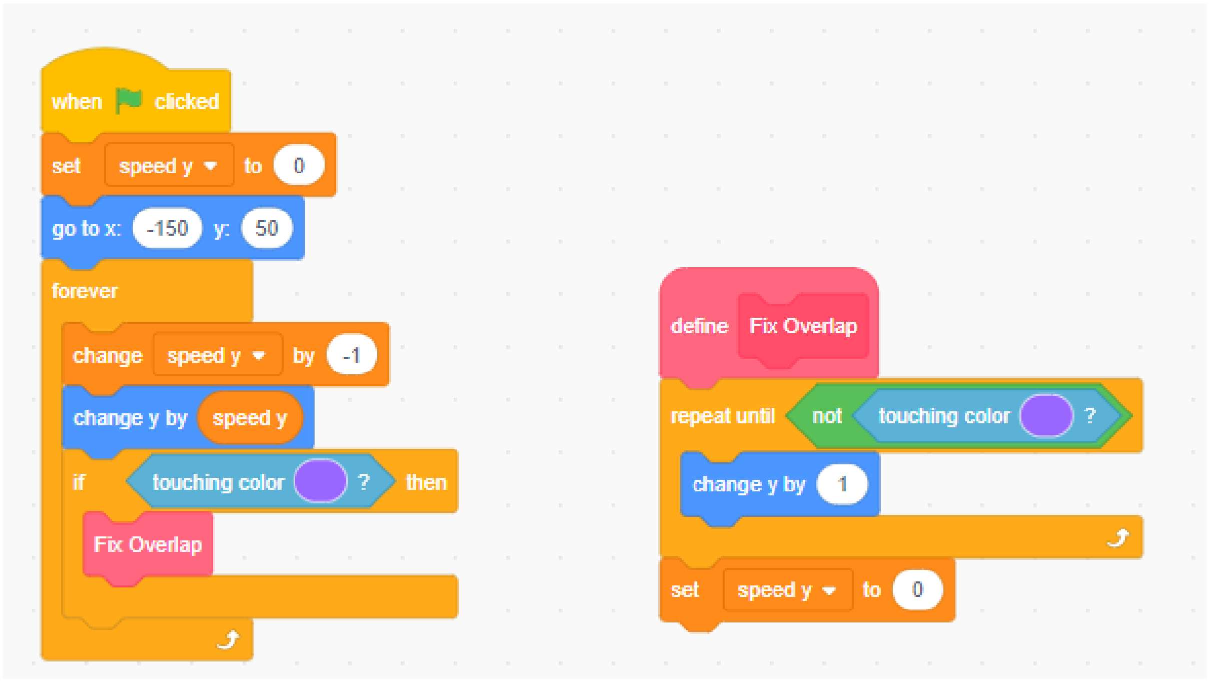
Add an input
boolean

Add a label

☐ Run without screen refresh

Link the repeat loop and the speed zeroing to the definition of Fix Overlap.

And then simply add the Fix Overlap block in the IF.



```
when 🏳 clicked
set speed y ▾ to 0
go to x: -150 y: 50
forever
    change speed y ▾ by -1
    change y by speed y
    if touching color ● ? then
        Fix Overlap
```

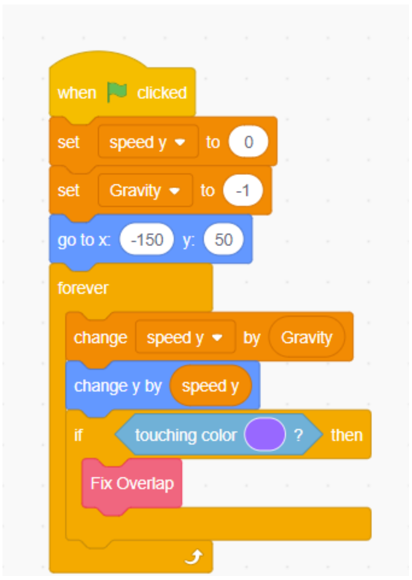
```
define Fix Overlap
repeat until not touching color ● ?
    change y by 1
set speed y ▾ to 0
```

Annnddd.. if you've done everything right, Guy should now appear to **instantly** stop on the floor.

## Make Gravity a variable:

Now at the moment the rate at which speed Y changes each loop is manually hardcoded as minus 1. Let's make a variable Gravity and select "for all sprites", and set it to -1 at the start of the program and put this variable in the change speed y block instead of -1.

So now your code should look like: -



```
when 🏳 clicked
set speed y ▾ to 0
set Gravity ▾ to -1
go to x: -150 y: 50
forever
    change speed y ▾ by Gravity
    change y by speed y
    if touching color ● ? then
        Fix Overlap
```

Once you have that done, feel free to play with some different values for Gravity and run the program to see how that affects Guy's falling. They should all be negative values however (or Guy will fall "up"!).

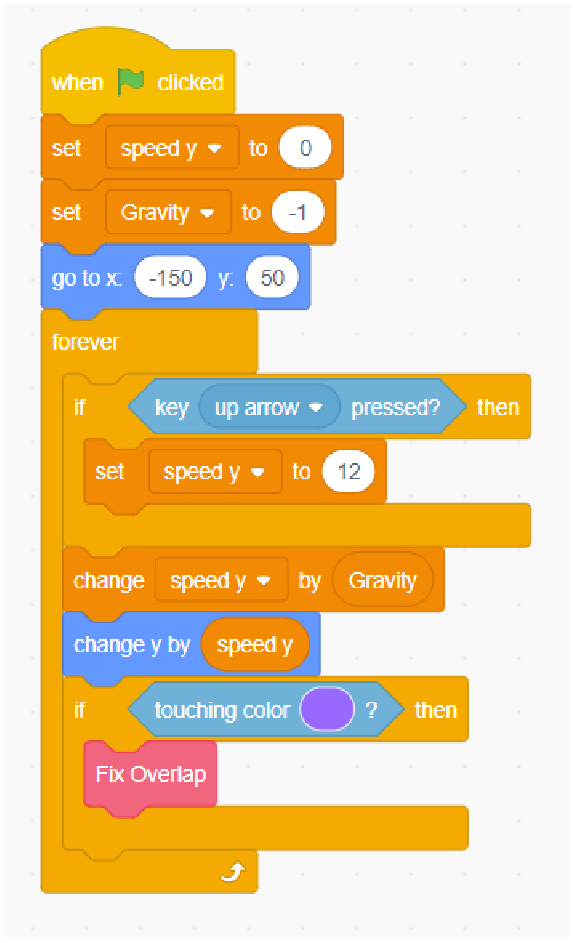
But once you're done, set Gravity back to -1 before continuing.

## JUMPING

Now that we have gravity in place, we should be able to code jumping.

This is really as simple now as adding an IF for detecting up arrow press, and then setting speed y to a positive value, say 12 to begin with.

The code you need then is : -



```
when [flag] clicked
set speed y to 0
set Gravity to -1
go to x: -150 y: 50
forever
    if key up arrow pressed? then
        set speed y to 12
    change speed y by Gravity
    change y by speed y
    if touching color ? then
        Fix Overlap
```

But.. there's an issue.. as it stands there is nothing to stop you pressing up arrow again while you're in the air to get another speed boost and you can carry on like this up into the sky like a rocket. This obviously isn't how the real world works .. you shouldn't be able to jump unless you are on the ground!

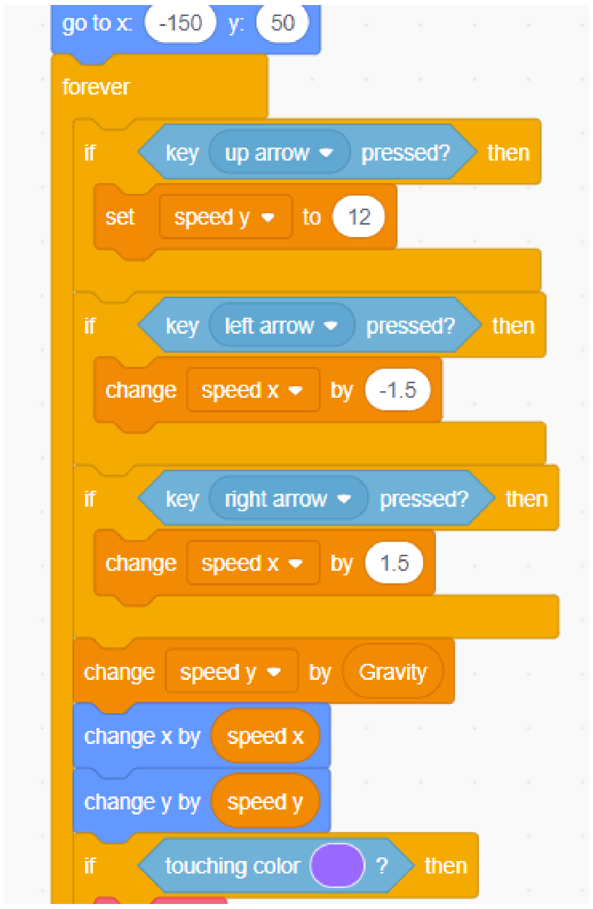
We'll come back to this and fix this later.

## WALKING LEFT AND RIGHT

Let’s enable moving left and right.

Create a new variable “speed x” (for this sprite only) and set it as a negative value when left arrow pressed (because x gets smaller as sprites move to the left) or a positive value if the right arrow key is pressed and change X in the same place we change Y in our forever loop.

So we need this code now : -



```
go to x: -150 y: 50
forever
    if key up arrow ▾ pressed? then
        set speed y ▾ to 12
    if key left arrow ▾ pressed? then
        change speed x ▾ by -1.5
    if key right arrow ▾ pressed? then
        change speed x ▾ by 1.5
    change speed y ▾ by Gravity
    change x by speed x
    change y by speed y
    if touching color 🟣 ? then
```

Because we CHANGE x by 1.5 every loop that left or right is pressed, this acts like an acceleration just like gravity does when pulling Guy downwards. So long as you press in a certain direction you will get faster and faster. And Guy will climb the stairs! This happens for free because when he moves into the first step, he is automatically lifted up by our gravity fix – happy days!

Give it a try – you can also jump at the same time.

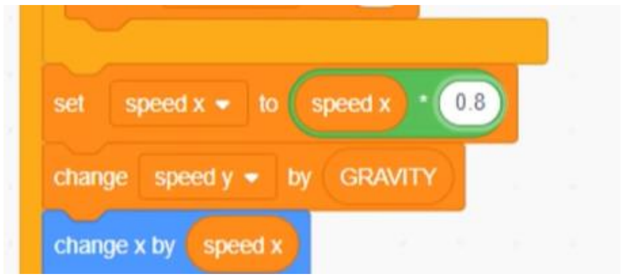
But the issue now is, this is like sliding on ice – Guy moves very quickly and there’s no way to stop once you start!

What we need to add is some **friction** – friction is force that pushes back when you move against something, and acts like a brake.

A good way to model this is to decrease the current speed by some factor, say 20%, ever loop.

The way we do this mathematically in Scratch is to multiply the speed by 0.8. This means set the speed to 8/10 of its current value, i.e. make it smaller. If we stop pressing an arrow key, this multiplication will quite quickly make speedX gets so small that it will become equal to zero because scratch numbers have only a certain accuracy possible.

So we simply add in our friction brake then by adding this block: -



```
set speed x to (speed x * 0.8)
change speed y by GRAVITY
change x by speed x
```

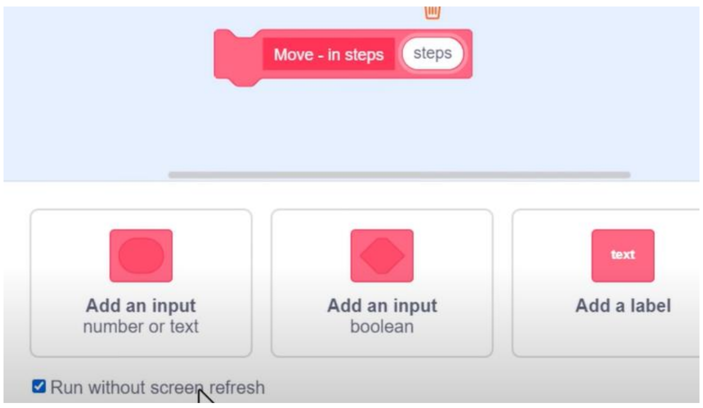
Try it again - now it should be much more controllable!

## MOVE IN STEPS

Now Guy is currently vaulting up the steps because we are moving in big jumps of pixels each time and as a result he is ending up “in” the steps and then getting moved up. But this is a platformer, we want to only climb things by jumping so let’s make the movements left and right more sophisticated to make this happen.

Make another custom block call this one “Move – In Steps” and click the “Add an input number or text” button. This allows you to say you want to “pass in” a number for the custom block to use every time you use it in your program.

You can give this a name by clicking in the bubble that appears – call it “steps” & as we did the last time, make it run with screen refresh: -

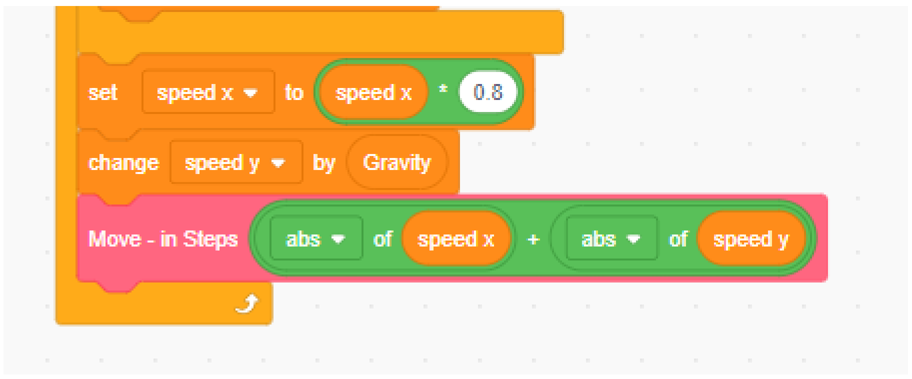


Move - in steps (steps)

☑ Run without screen refresh

SpeedX and SpeedY basically tells us how many pixels we have to move in each frame update. So we will add them together and divide the movement up into their total (behind the scenes, as screen refresh is off).

So we can remove all the code after "change speed y by Gravity" and just add this 1 line of code: -



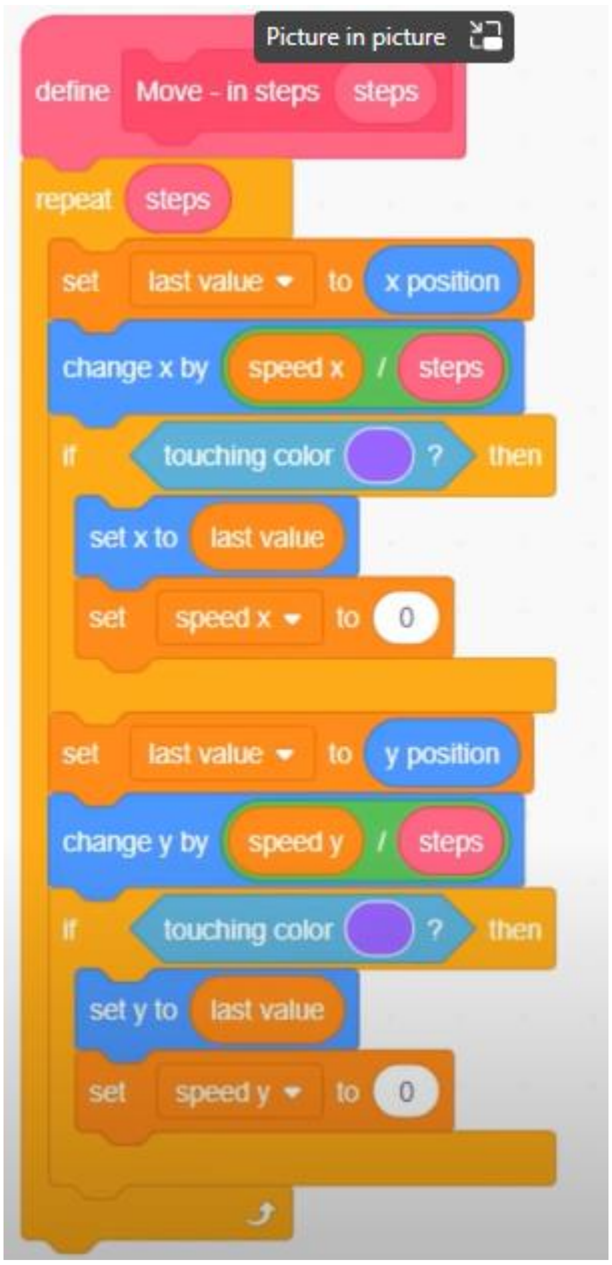
```
set speed x to (speed x * 0.8)
change speed y by Gravity
Move - in Steps ((abs of speed x) + (abs of speed y))
```

This is basically passing SpeedX + SpeedY as the number of steps, the abs is a mathematical function "absolute" and we need it because speed x and speed y are negative numbers – abs turns negative numbers into positive (and leaves positive numbers positive!)

Now we just need to build that Move in Steps custom block.

Careful put together the following, note you will need to create a new variable "last value" (for Guy sprite only).

Another tip is, where you which to use "Steps" value passed in to the block call in the code, just drag the Steps bubble from the define block at the top and put it in place where you need it.



```
define Move - in steps (steps)
repeat (steps)
    set [last value] to (x position)
    change x by ((speed x) / (steps))
    if <touching color [ ]?> then
        set x to (last value)
        set [speed x] to (0)
    set [last value] to (y position)
    change y by ((speed y) / (steps))
    if <touching color [ ]?> then
        set y to (last value)
        set [speed y] to (0)
```

This is getting a little complicated but basically it works like this: -

It does a loop “Steps” times – and remember steps will be the current speed X PLUS speed Y.

Each loop it moves Guy first Speed X divided by steps and checks for collision. If a collision is detected, it resets X back to the last X position on screen recorded and zeroes the X speed.

And the same then is done in the Y direction.

By dividing Speed X and Speed Y by “Steps” each move, this means that when the loop is finished Guy will have moved the full Speed X and Speed Y.

And all this happens "invisibly" and therefore very quick because we turned off screen refreshing for this custom block!

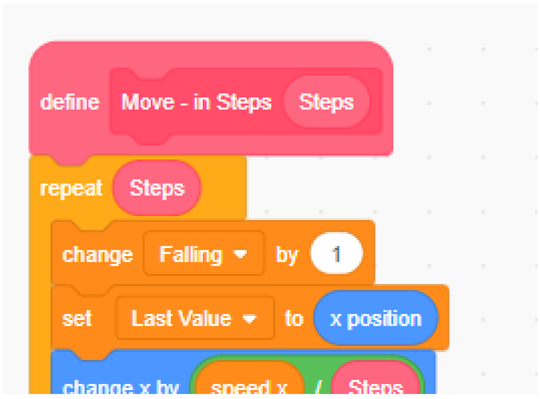
So after you have coded this, Guy will stop whenever he hits into anything horizontally or vertically, and you'll now have to jump up the stairs and on to the platform.

## FIXING JUMPING

One thing you've probably noticed is you can jump as many times as you like, even in the air at present – which of course you can't do in real life, so let's tackle this issue next.

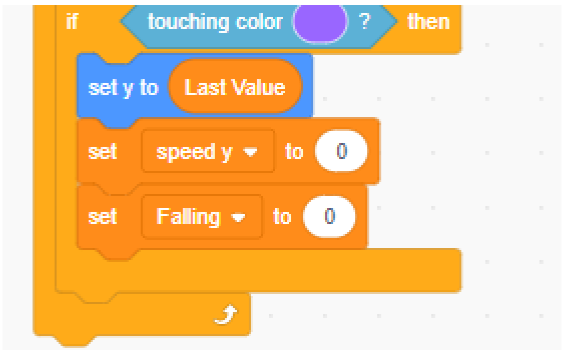
Let's make a new variable "Falling" (for Guy only). This variable will act like a flag to let us know if Guy is currently falling or not, and we'll use this to control when you are allowed to jump again.

Falling is going to count upwards the whole time we are "in the air". So at the start of each move in steps block, let's change it by + 1.



```
define Move - in Steps (Steps)
repeat (Steps)
    change [Falling ▾] by (1)
    set [Last Value ▾] to (x position)
    change x by ((speed x) / (Steps))
```

Next just after we set "Speed Y" to zero after any vertical collision, let's set Falling back to zero as well: -



```
if <touching color ( ) ?> then
    set y to (Last Value)
    set [speed y ▾] to (0)
    set [Falling ▾] to (0)
```

And then lastly we put an IF around the set of Speed Y in the up arrow detect section and only allow it if Falling < 3.

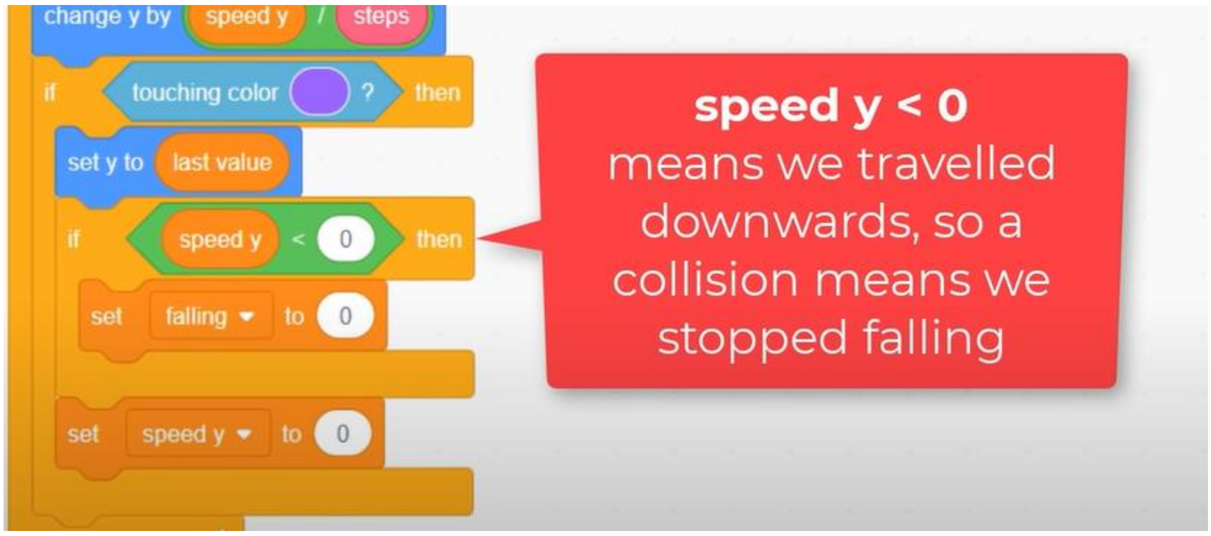
Why 3? 3 let's you still jump a little bit after moving off the edge of a platform 😊 and this makes the game a little friendlier and not as frustrating.

If falling is a bigger number 3 we no longer can jump until falling gets reset to zero on the ground again. Actually depending on your PC you might need to make this a little bigger maybe 7 or 8 – when it is set right you should be able to jump even when moving quickly let and right.

Test it out – does it work well – **there is still a bug – can you find it??**

We're nearly there but the slight bug remaining is, it may be possible to "hang" under your platform by holding down the up key. This is because this is a collision in the Y direction BUT its not with a floor!

So to fix this last wrinkle add this: -



```
change y by (speed y / steps)
if <touching color ( )?> then
    set y to (last value)
    if <(speed y) < (0)> then
        set falling to (0)
    end
    set speed y to (0)
end
```

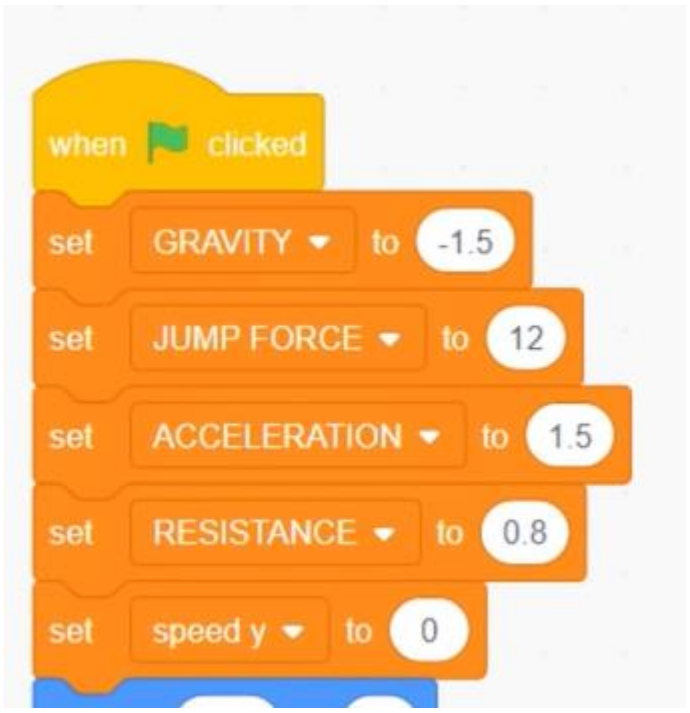
**speed y < 0**
means we travelled downwards, so a collision means we stopped falling

**NOTE:** we've moved it above the set speed Y to 0 also.

And we're all good! All your basic platformer mechanics are now in place 😊

Finally, let's make it a bit easier to tweak our world physics, until the game has a good feel.

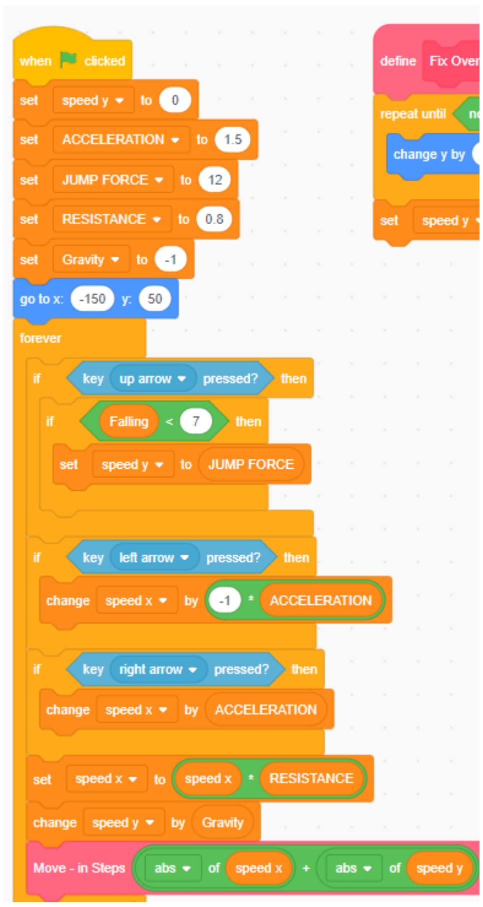
Create all the variables after Gravity here and set them as shown – **note create them "for all sprites": -** Also let's make the Gravity a little strong by increasing it to -1.5.



```
when 🏴 clicked
set GRAVITY ▾ to -1.5
set JUMP FORCE ▾ to 12
set ACCELERATION ▾ to 1.5
set RESISTANCE ▾ to 0.8
set speed y ▾ to 0
```

And then put these variables in the code where these numbers are currently, with one exception we need to make acceleration negative for moving left, by multiplying by minus 1.

You main block should look something like this when all done: -



```
when 🏴 clicked
set speed y ▾ to 0
set ACCELERATION ▾ to 1.5
set JUMP FORCE ▾ to 12
set RESISTANCE ▾ to 0.8
set Gravity ▾ to -1
go to x: -150 y: 50
forever
    if key up arrow ▾ pressed? then
        if Falling < 7 then
            set speed y ▾ to JUMP FORCE
    if key left arrow ▾ pressed? then
        change speed x ▾ by (-1 * ACCELERATION)
    if key right arrow ▾ pressed? then
        change speed x ▾ by ACCELERATION
    set speed x ▾ to (speed x * RESISTANCE)
    change speed y ▾ by Gravity
    Move - in Steps ((abs ▾ of speed x) + (abs ▾ of speed y))
```

```
define Fix Over
repeat until n
    change y by
set speed y
```

Now we have a lot of variables displaying on screen at this stage, you can remove them by unticking them in the variables creation section.

So now you can try tinkering with all 4 of Gravity / Jump force / Acceleration and Resistance. Have fun and leave them set at something that feels just right for your game.

**COMING NEXT – Adding Levels!**