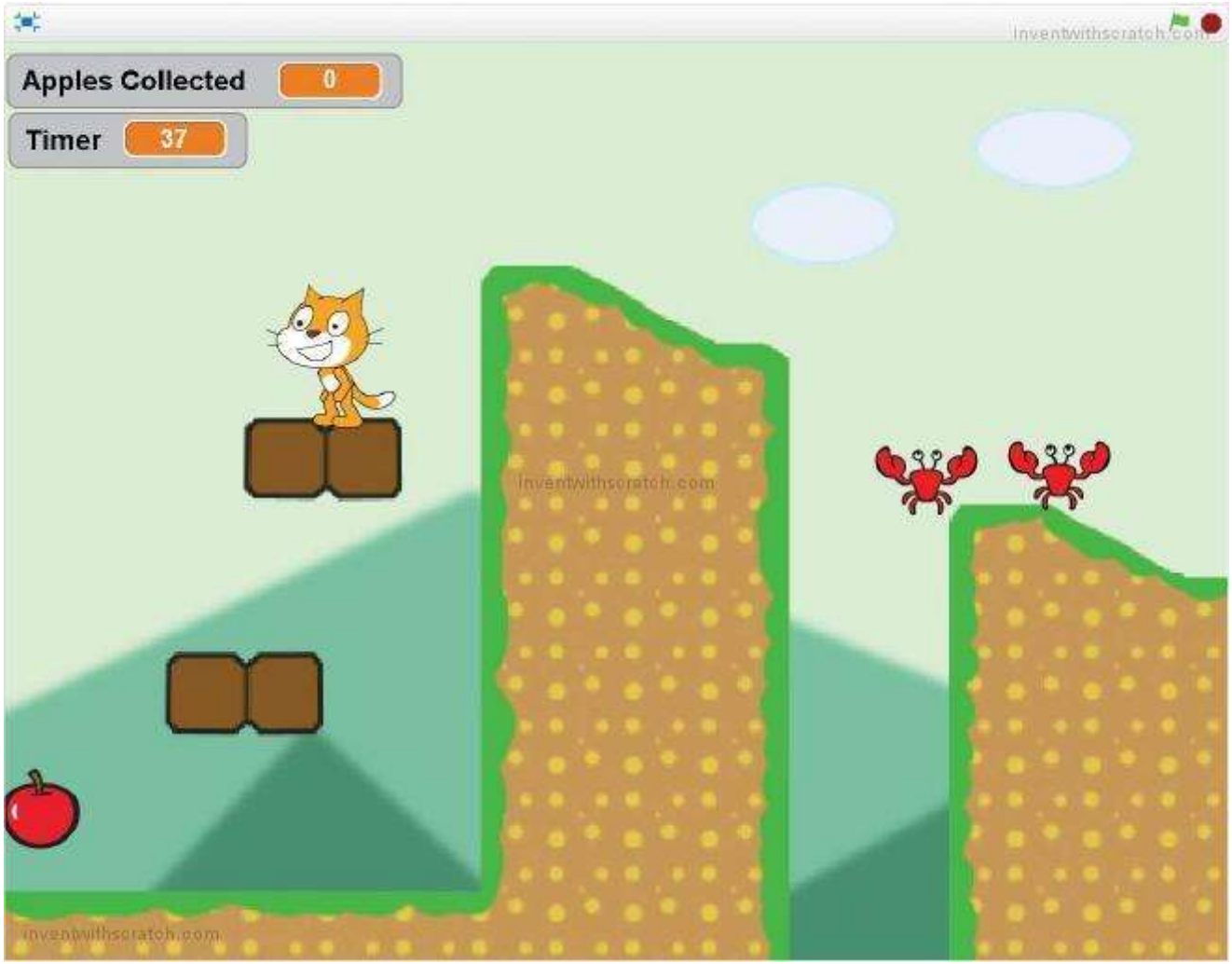
9
MAKING AN ADVANCED PLATFORMER

The first *Super Mario Bros.* game was introduced in 1985 and became Nintendo's greatest video game franchise and one of the most influential games of all time. Because the game involves making a character run, jump, and hop from platform to platform, this game style is called a *platformer* (or *platform* game).

In the Scratch game in this chapter, the cat will play the part of Mario or Luigi. The player can make the cat jump around a single level to collect apples while avoiding the crabs who will steal them. The game is timed: the player has just 45 seconds to collect as many apples as possible while trying to avoid the crabs!

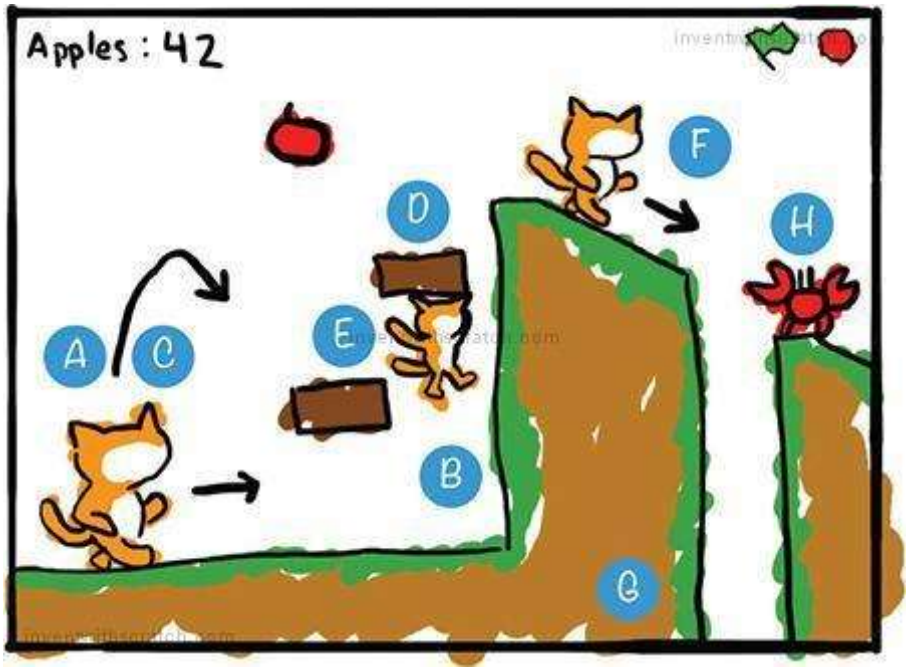
Before you start coding, look at the final program at *https://www.nostarch.com/scratchplayground/*.



Get ready to program a more complicated game than those in previous chapters!

SKETCH OUT THE DESIGN

Let's sketch out on paper what the game should look like. The player controls a cat that jumps around while apples appear randomly. The crabs walk and jump around the platforms randomly, too.



Here's what we'll do in each part:

A. Create gravity, falling, and landing

B. Handle steep slopes and walls

C. Make the cat jump high and low

D. Add ceiling detection

E. Use a hitbox for the cat sprite

F. Add a better walking animation

G. Create the level

H. Add crab enemies and apples

This platform game is the most ambitious one in the book, but anyone can code it if they follow the steps in this chapter. Let's code each part one step at a time.

If you want to save time, you can start from the skeleton project file, named *platformer-skeleton.sb2*, in the resources ZIP file. Go to *https://www.nostarch.com/scratchplayground/* and download the ZIP file to your computer by right-clicking the link and selecting **Save link as** or **Save target as**. Extract all the files from the ZIP file. The skeleton project file has all the sprites already loaded, so you'll only need to drag the code blocks into each sprite.

CREATE GRAVITY, FALLING, AND LANDING

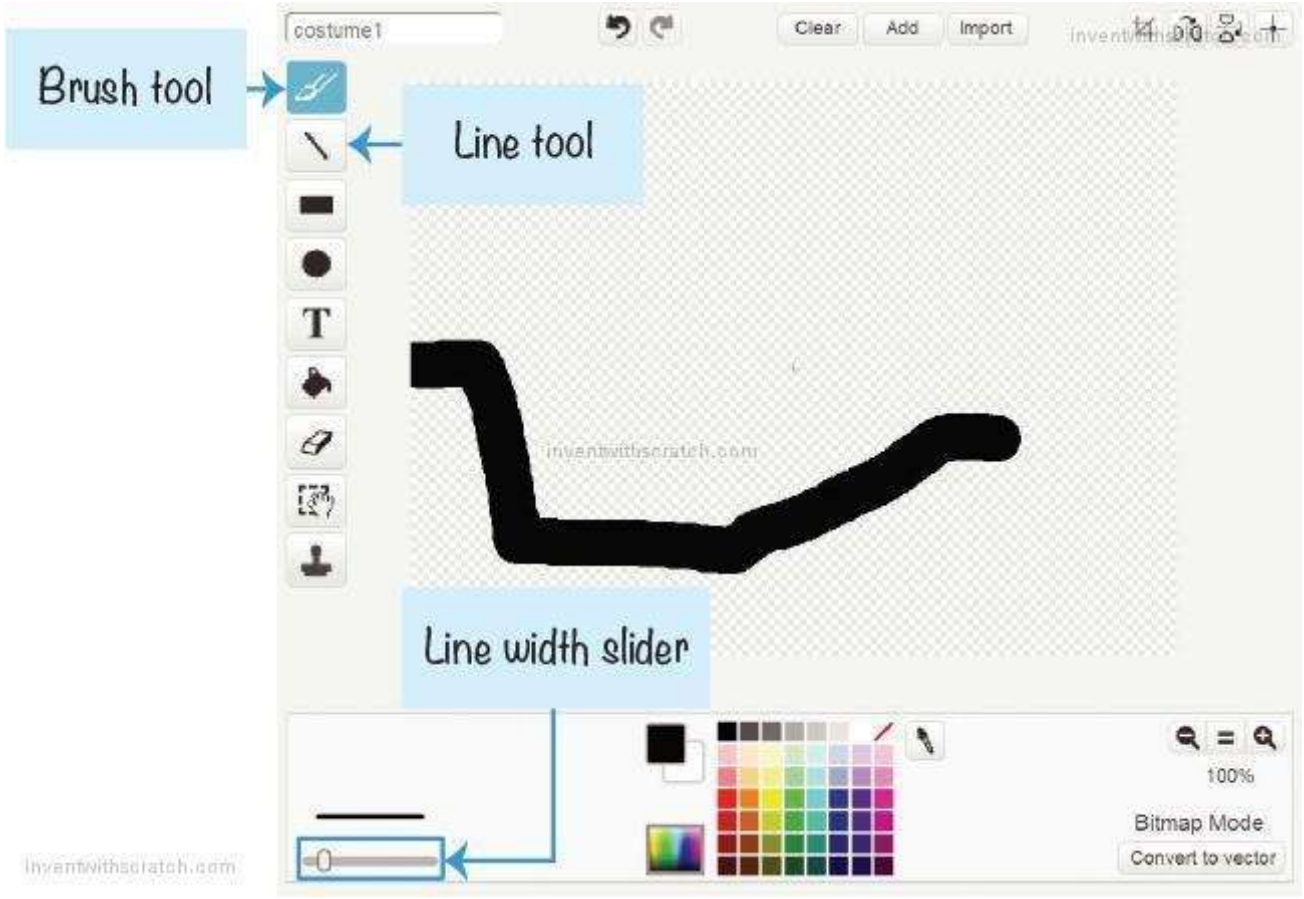
In the first part, we'll add gravity, falling, and landing code, similar to the *Basketball* game in Chapter 4. The important difference is that in the platform game, the cat lands when it touches a ground sprite rather than the bottom of the Stage. Coding is a bit trickier, because we want the ground to have hills and eventually platforms!

To start, click the text field at the top left of the Scratch editor and rename the project from *Untitled* to *Platformer*.

1. Create the Ground Sprite

Let's use a simple shape for the ground in the first few scripts, just to explore how the code will work.

Click the **Paint new sprite** button next to New sprite to create a temporary ground sprite while you learn about platforming code. In the Paint Editor, use the Brush or Line tool to draw a shape for the ground. You can make the lines thicker by using the Line width slider in the bottom-left corner of the Paint Editor. Be sure to draw a gentle slope on the right and a steep slope on the left.

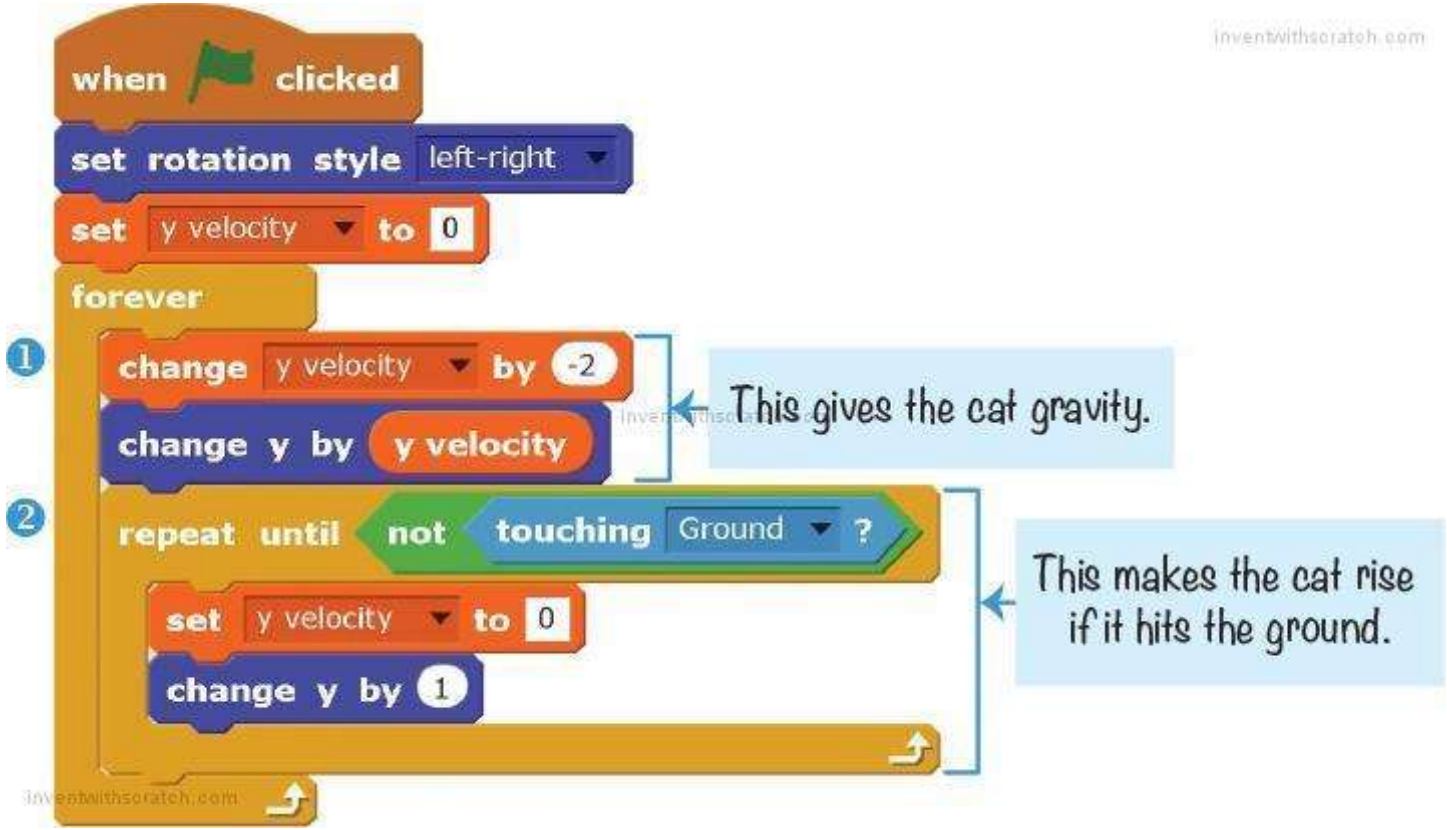


Open the sprite's Info Area and rename the sprite Ground. Also, rename the Sprite1 sprite Cat.

2. Add the Gravity and Landing Code

Now that we have a sprite for the ground, we need the cat to fall and land on it.

Select the Cat sprite. In the orange *Data* category, click the **Make a Variable** button and create a For this sprite only variable named y velocity. Then add the following code to the Cat sprite:



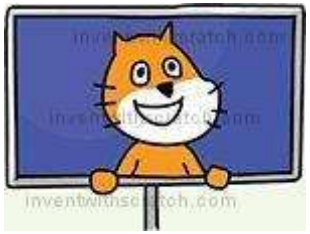
This code performs two actions in its **forever** loop: it makes the Cat sprite fall until it touches the Ground sprite ❶ and then lifts up the Cat sprite if it is deep in the ground ❷.

With these two code sections, the cat will fall down, hit the ground, and then rise if necessary, eventually settling on top of the Ground sprite.

The falling code at ❶ subtracts 2 from the y velocity variable and then moves the Cat sprite's y position by y velocity, making the cat fall faster and faster. If you programmed the *Basketball* game in Chapter 4, the falling code should be familiar.

But the **repeat until** block ❷ will loop until the Cat sprite is no longer touching the Ground sprite. (If the cat is still in the air and falling, it will not be touching the ground, so the code in the loop is skipped.) Inside this loop, the y velocity is set to 0 so that the Cat stops falling any farther. The **change y by 1** block will lift up the Cat sprite a little. The **repeat until not touching Ground** block continues lifting the sprite until it is no longer sunk into the Ground sprite. This is how the cat stays on top of the ground, no matter what the shape of the Ground sprite is.

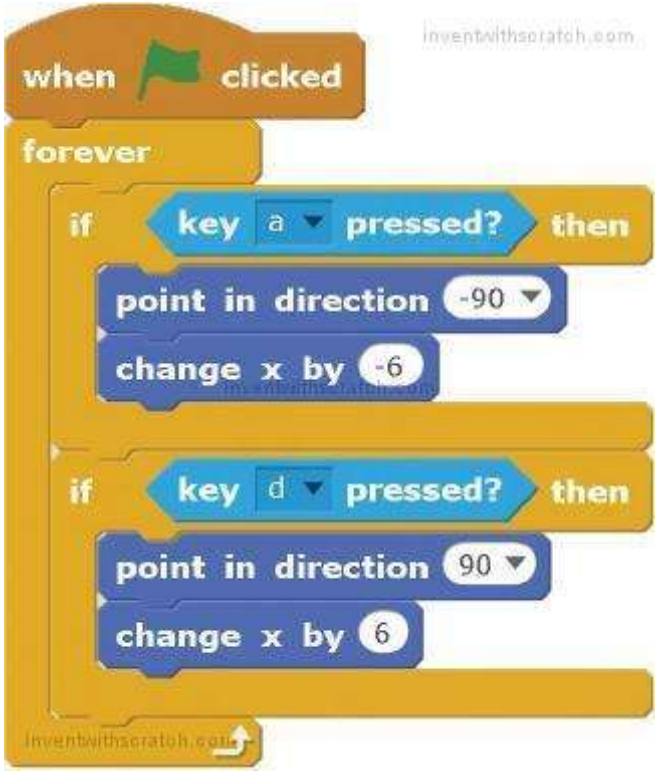
SAVE POINT



Click the green flag to test the code so far. Drag the cat up with the mouse and let go. Make sure the cat falls and sinks into the ground a bit and then slowly lifts out of it. Then click the red stop sign and save your program.

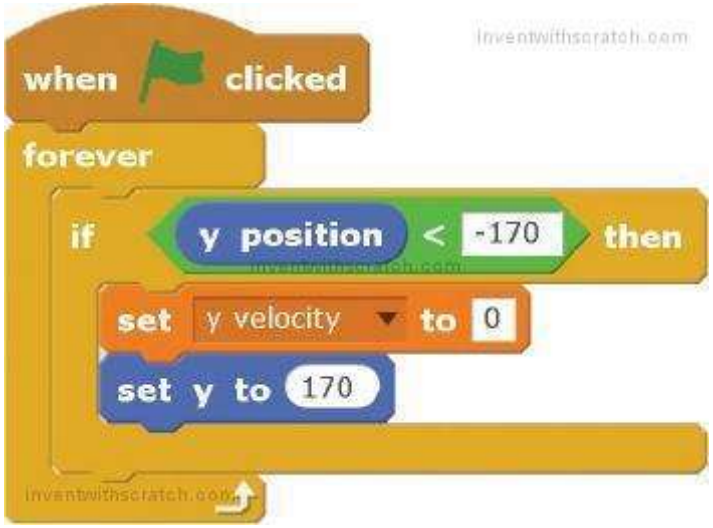
3. Make the Cat Walk and Wrap Around the Stage

The cat also needs to walk left and right using the WASD keys, so add the following script to the Cat sprite:



This code is very straightforward: pressing A points the cat to the left (-90) and moves the x position by -6 (to the left); pressing D makes the cat point to the right and moves the x position by 6 (to the right).

Next, add the following script to make the Cat sprite wrap around to the top if it falls to the bottom of the Stage.



This code is very similar to the wrap-around code we wrote in the *Asteroid Breaker* game in Chapter 8. We'll write wrap-around code for moving left and right later.

SAVE POINT

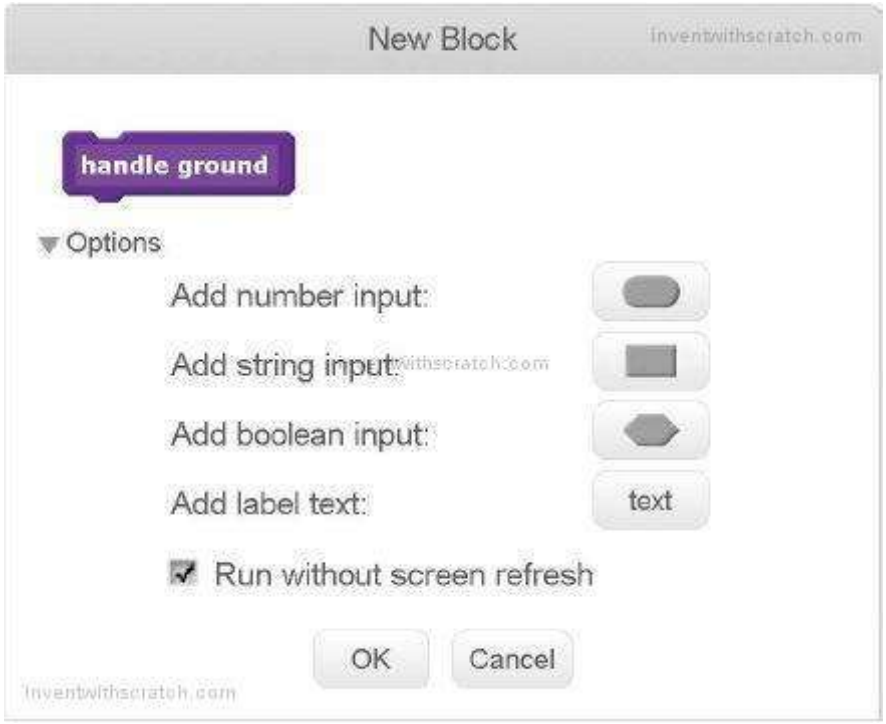
Click the green flag to test the code so far. Press the A and D keys to make the cat walk up and down the slopes. If the Cat sprite walks off the edge of the Ground sprite and falls to the bottom of the Stage, the Cat sprite should reappear at the top. Then click the red stop sign and save your program.

This platformer has many scripts, so you might get lost if you get confused. If your program isn't working and you can't figure out why, load the project file *platformer1.sb2* from the resources ZIP file. Click **File** ▶ **Upload from your computer** in the Scratch editor to load the file, and continue reading from this point.

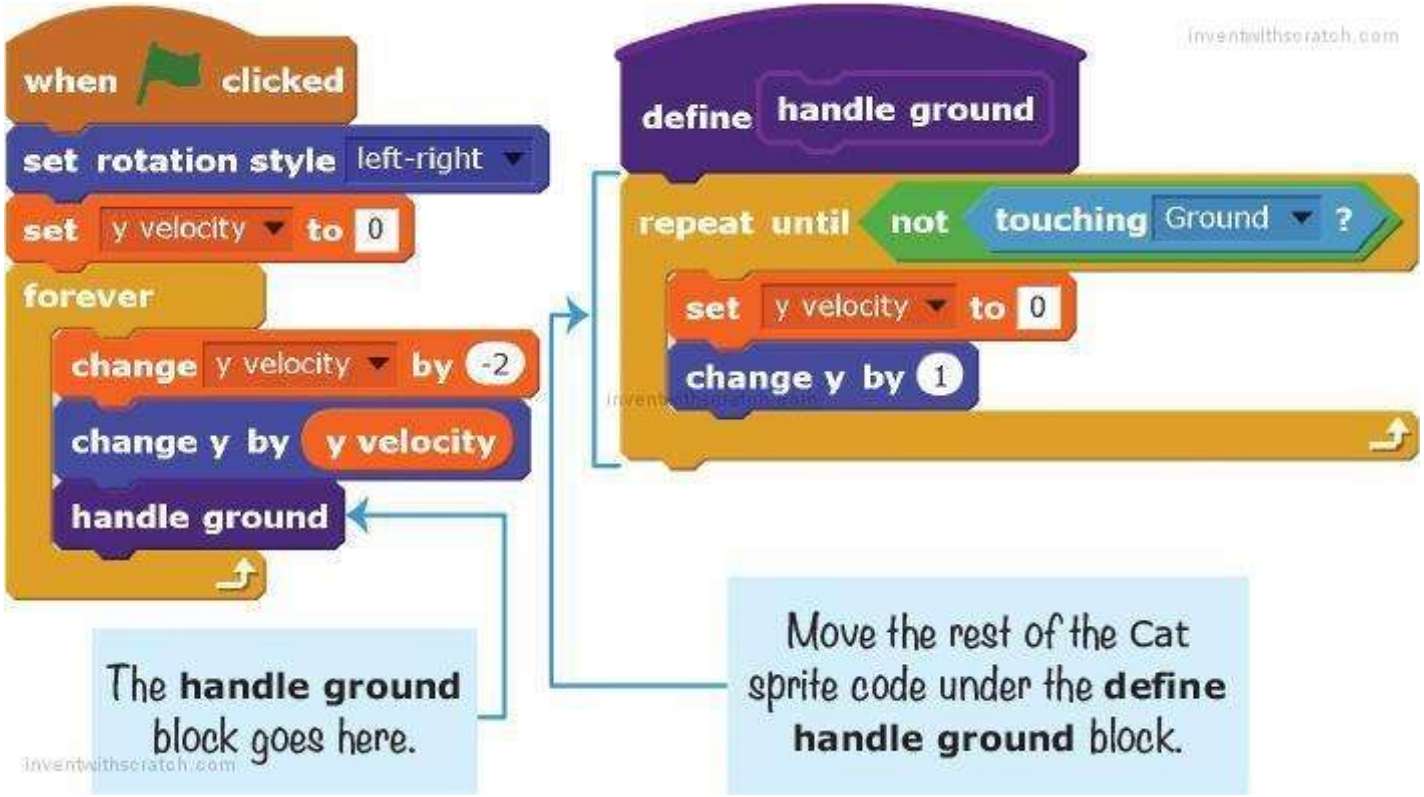
4. Remove the Ground Lift Delay

The big problem with the code right now is that the Cat sprite is lifted from inside the ground to on top of it very slowly. This code needs to run so fast that the player only sees the sprite on top of the ground, not in it.

The dark purple custom blocks can help us do this. Go to the *More Blocks* category, and click the **Make a Block** button. Name the block **handle ground**, and then click the gray **Options** triangle. Check the checkbox next to **Run without screen refresh**.



The **define handle ground** block should now appear in the Scripts Area. Change the code for the Cat sprite to make it use the **handle ground** block. The **handle ground** block goes where the **repeat until not touching Ground** blocks were, and that loop is moved under **define handle ground**.



This code works exactly as it did before, but now the **handle ground** block has Run without screen refresh checked, so the loop code runs in Turbo Mode. Lifting the cat now happens instantly, so it looks like the cat never sinks into the ground.

SAVE POINT

Click the green flag to test the code so far. Make the cat walk around or use the mouse to drop the cat from the top of the Stage as before. Now the Cat sprite should never sink into the ground. Then click the red stop sign and save your program.

If you're lost, open *platformer2.sb* in the resources ZIP file and continue reading from this point.

HANDLE STEEP SLOPES AND WALLS

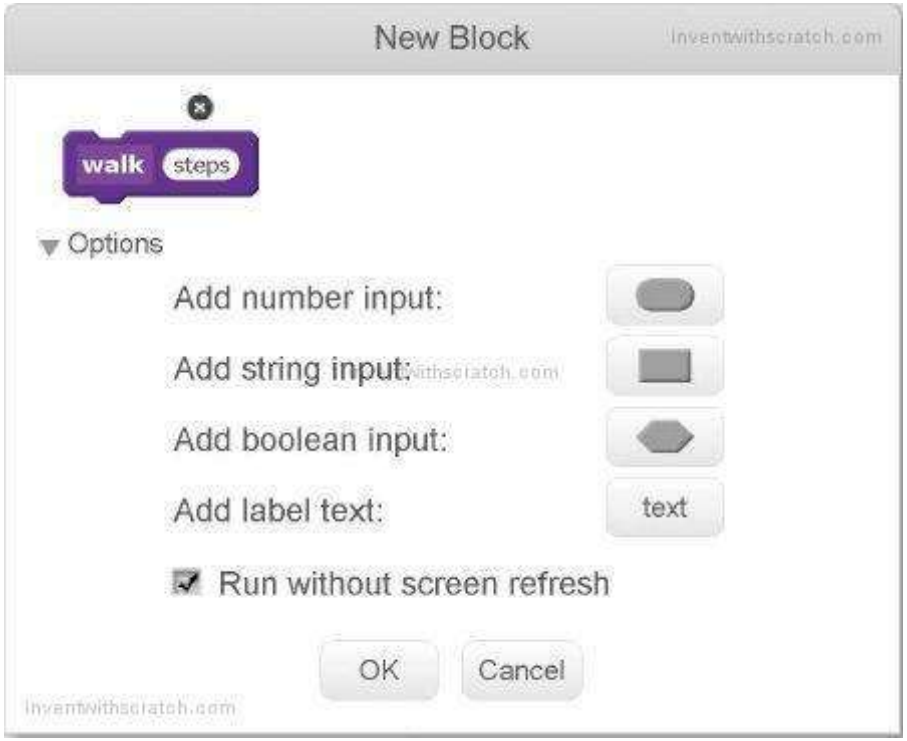
The Ground sprite has hills and slopes that the cat can walk on, and you can change the Ground sprite to pretty much any shape in the Paint Editor. This is a big improvement for the player compared to just walking on the bottom of the Stage, as in the *Basketball* game. But now the problem is that the Cat sprite can walk up the steep slope on the left as easily as it can walk up the gentle slope on the right. This isn't very realistic. We want the steep slope to block the cat. To do this, we'll make a small change to the walking code blocks.

At this point, the sprites are becoming overcrowded with lots of different scripts. So, right-click on the Scripts Area, and select **clean up** to reorganize the scripts into neat rows.

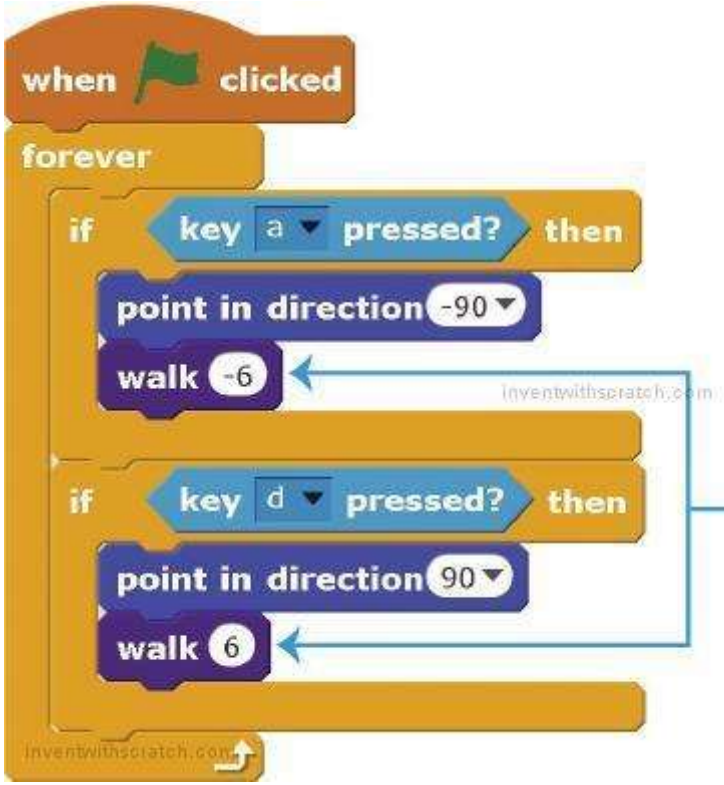
5. Add the Steep Slope Code

Now we need to edit the Cat sprite's walking code and add some new code, too. Instead of simply changing the x position by a particular value, we'll use a new custom block. Let's call it **walk** and give this new custom block an *input* called **steps**. An input is somewhat like a variable, but you can only use it in the custom block's **define** block.

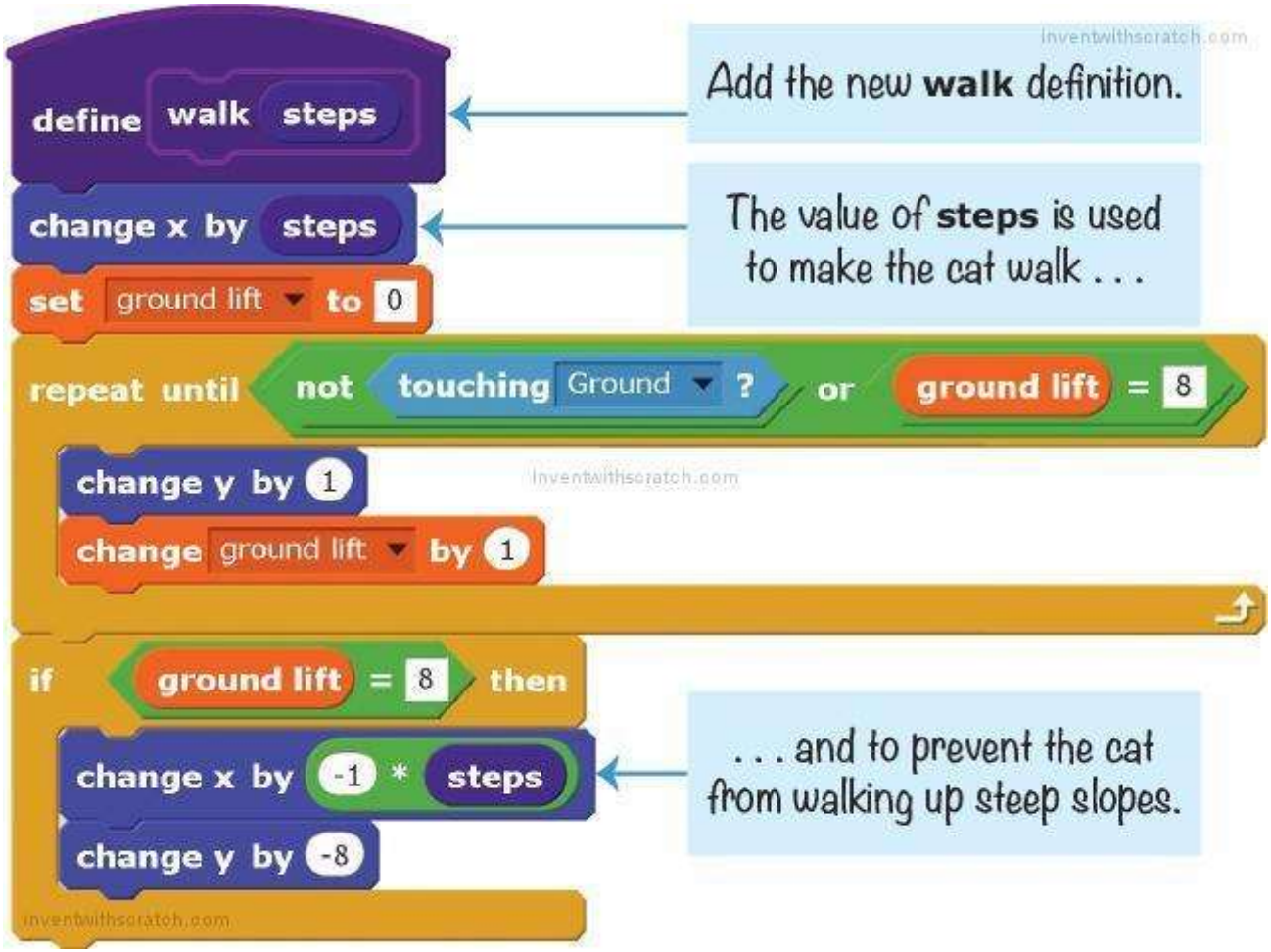
Click the **Make a Block** button in the dark purple *More Blocks* category to make the **walk** block. Be sure to click the **Add a number input button** to make the **steps** input. When we want to call the new **walk** block, we'll have to define that input with a number of **steps** to take. Make sure you check the **Run without screen refresh** checkbox!



This code also requires you to make a variable for the Cat sprite named ground lift (which should be For this sprite only). We'll use this new variable to determine whether a slope is too steep for the cat to walk up. This code is a little complicated, but we'll walk through it step-by-step. For now, make the Cat sprite's code look like the following:



Add calls to the **walk** block, which has an input for the number of **steps**.

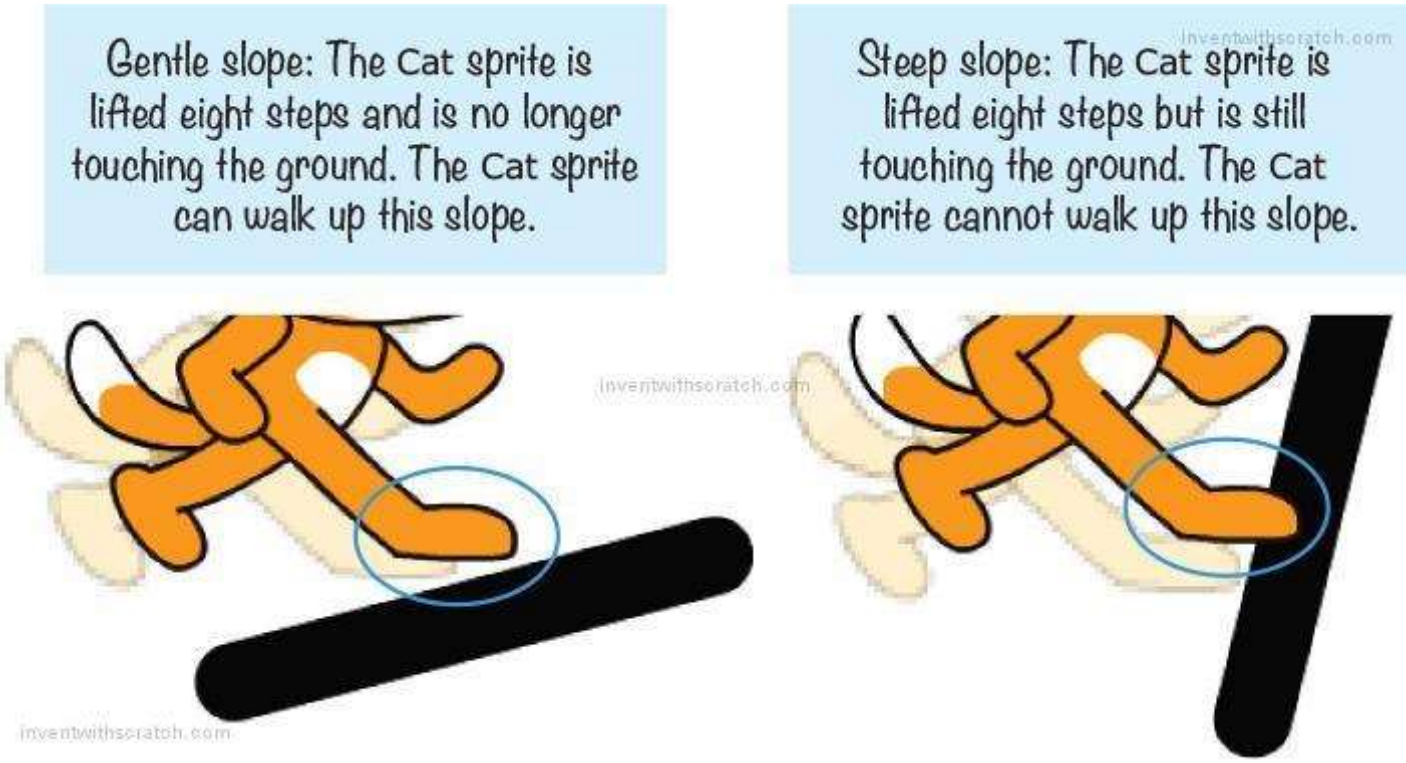


We want the cat to walk six units, as we did earlier, so we use -6 and 6 in the walking script when we call **walk**. In the **define walk** block, the **steps** input block is used in the **change x by** blocks. This makes the code more compact, because we can use the same script for moving the cat to the left (with the **walk -6** block) or to the right (with the **walk 6** block).

The code in the **repeat until** loop uses the ground lift variable to determine if the slope is a walkable slope or a wall that should block the Cat sprite's progress. The ground lift variable starts at 0 and changes by 1 each time the **repeat until** loop lifts the Cat sprite's y position by 1. This loop continues looping until the sprite is no longer touching the ground or ground lift is equal to 8.

If ground lift is less than 8, then the slope isn't that steep. The sprite can walk up the slope, so the **define walk** script doesn't have to do anything else.

But if **ground lift = 8**, the **repeat until** loop stops looping. This code means "the sprite has been lifted up by 8 but it's still touching the Ground sprite, so this must be a steep slope." In that case, we need to undo the lift *and* the walking movement. The **change y by -8** and **change x by -1 * steps** blocks undo the Cat sprite's movement. Multiplying the **walk** input by -1 gives the opposite number of the input and variable.



This code is exactly like the code in the *Maze Runner* game in Chapter 3 that blocks the player from walking through walls.

SAVE POINT

Click the green flag to test the code so far. Use the A and D keys to make the cat walk around. The cat should be able to walk up the gentle slope on the right, but the steep slope on the left should stop the cat. Click the red stop sign and save your program.

If you're lost, open *platformer3.sb* in the resources ZIP file and continue reading from this point.

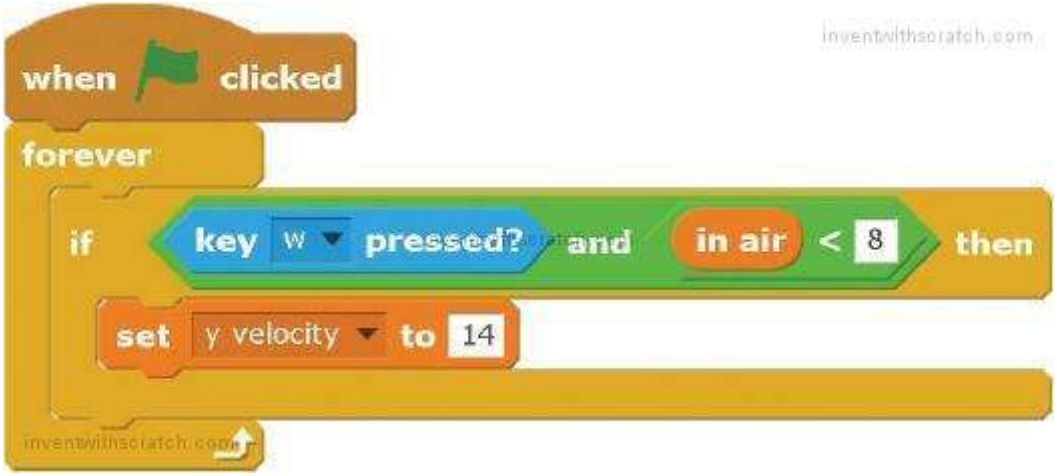
MAKE THE CAT JUMP HIGH AND LOW

With the walking code done, let's add jumping. In the *Basketball* game, we changed the falling variable to a positive number. This meant that the player jumped the height of that variable's value each time. But in many platform games, the player can do a short jump by pressing the jump button quickly or jump higher by holding down the jump button. We want to use high and low jumping in this platform game, so we'll have to come up with something a little more advanced than the *Basketball* game's jumping code.

6. Add the Jumping Code

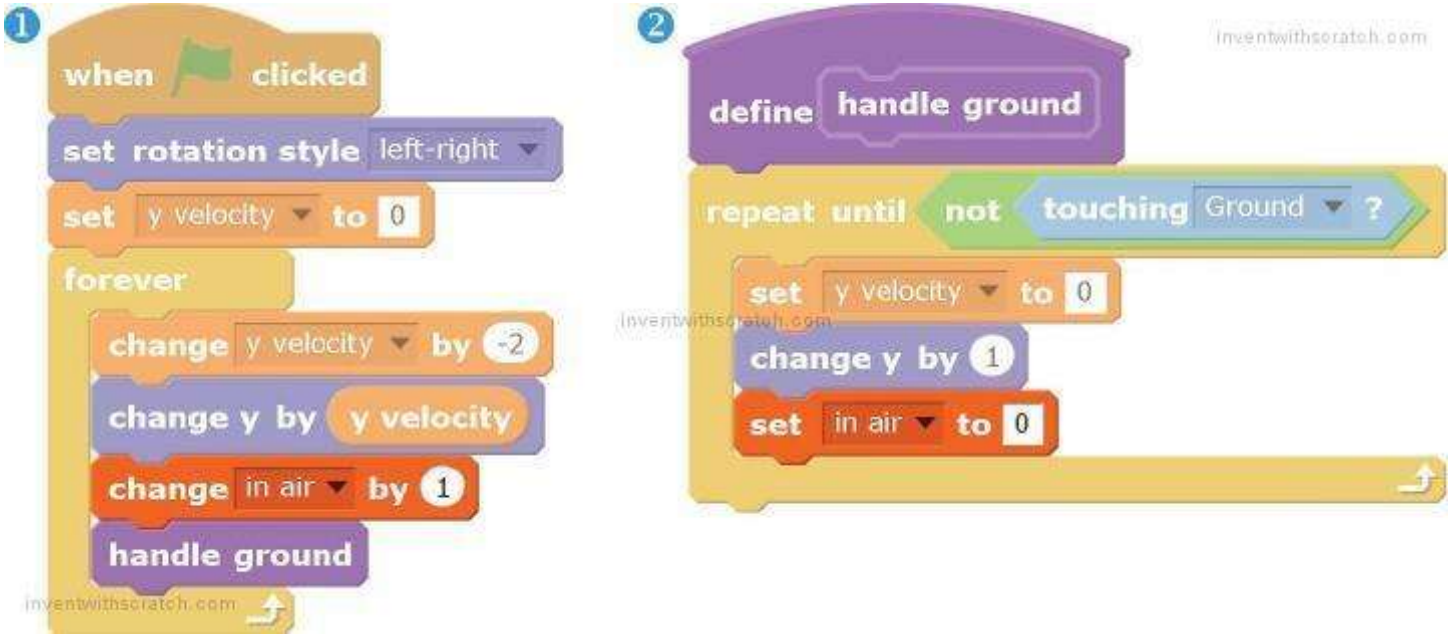
Let's first create a For this sprite only variable named in air. This variable will be set to 0 whenever the Cat sprite is on the ground. But in air will start increasing when the Cat sprite is jumping or falling. The larger the in air value is, the longer the cat will have been off the ground and in the air.

Add this script to the Cat sprite:



The **forever** loop keeps checking whether the W key is being held down. If it is, this will give the Cat sprite a velocity of 14—that is, the Cat sprite will be moving up. But note that there are two conditions for the cat to continue moving up—the player must hold down W *and* the in air variable must be less than 8.

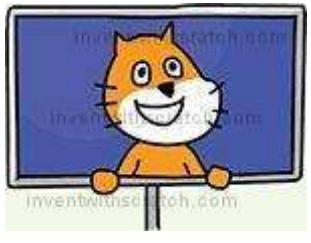
Let's edit two of the existing Cat sprite scripts to add the in air variable that limits how high the cat can jump.



When the player first holds down the W key to make the cat jump, the y velocity variable is set to 14. This makes the code in the **forever** loop in script ❶ change the Cat sprite's y position by the positive y velocity, moving it upward. At the start of the jump, the in air variable is increasing but is still less than 8. So if the player continues to hold down the W key, y velocity keeps getting set to 14 instead of decreasing because of the **change y velocity by -2** block. This causes the jump to go upward longer than if the player had held down the W key for just one iteration through the loop. But eventually in air will become equal to or greater than 8, so it won't matter if the W key is pressed. Remember that both conditions—**key w pressed** *and* **in air < 8**—must be true for the code inside the **if then** block to run.

At this point, the y velocity variable will decrease as expected, and the Cat sprite will eventually fall. In script ❷, when the cat is on the ground, the in air variable is reset to 0.

SAVE POINT



Click the green flag to test the code so far. Press the W key to jump. Quickly pressing the key should cause a small jump. Holding down the W key should cause a higher jump. Make sure the cat can jump only while it's on the ground and can't do double jumps. Then click the red stop sign and save your program.

If you're lost, open *platformer4.sb* in the resources ZIP file and continue reading from this point.

ADD CEILING DETECTION

The cat can walk on the ground, and now walls will stop the cat from moving through them. But if the player bumps the Cat sprite's head against a platform from below, the Cat sprite will rise above it! To solve this problem, we need to make some adjustments to the lifting code to add *ceiling detection*.

7. Add a Low Platform to the Ground Sprite

Add a short, low platform to the Ground sprite's costume, as shown in the following figure. Make sure it is high enough for the cat to walk under but low enough that the cat can jump up and touch it.

This platform should be low enough that the cat could hit its head on it. If it can't, then redraw the platform a little bit lower.

SAVE POINT

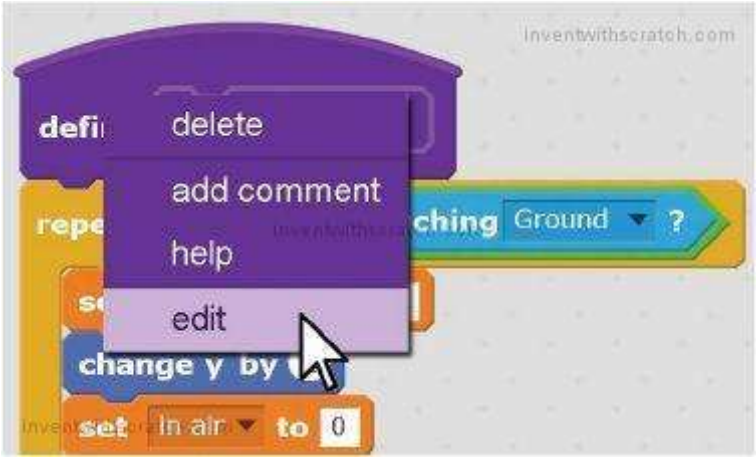
Click the green flag to test the code so far. Make the cat jump under the low platform. Notice that when the Cat sprite touches the platform, it ends up above the platform. This is the bug we need to fix. Click the red stop sign.

8. Add the Ceiling Detection Code

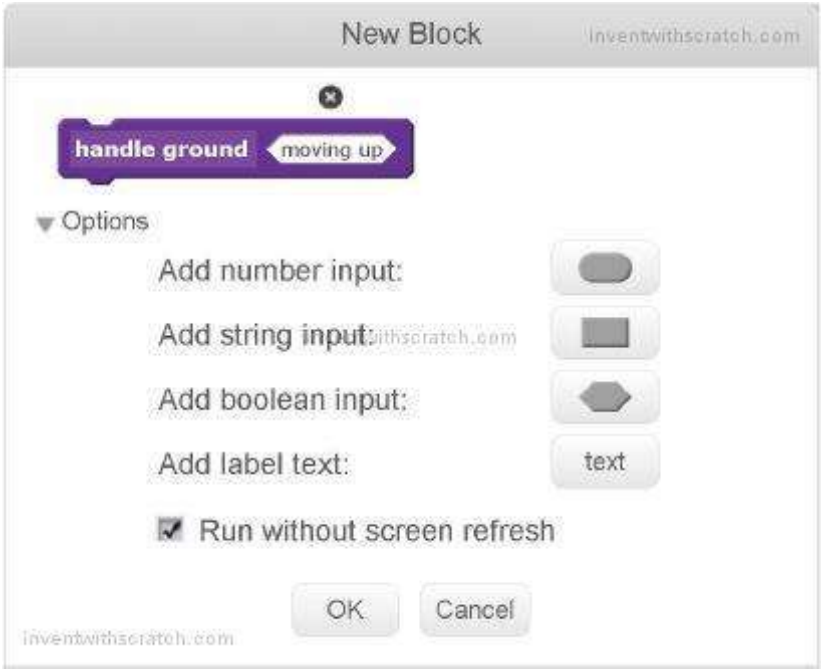
The problem with the code is in the custom **handle ground** block. This code always assumes the Cat sprite is falling from above, and if the Cat sprite is touching the Ground sprite, it should be lifted above it. The Ground sprite represents any solid part that the cat can't move through, including ceilings. We need to change the code so that if the Cat sprite is jumping up when it touches the Ground sprite, the cat *stops* rising because it bumps its head. We know the Cat sprite is moving upward when its y velocity is greater than 0. So let's edit the custom **handle ground** block to add a new Boolean input named **moving up**.

A *Boolean* is a true or false value. We use a Boolean input because we need to know if y velocity is greater than 0 when the **handle ground** block is first called. This true or false value is stored in the **moving up** input just like a variable would store it. If we put **y velocity > 0** in the **if then** block instead of **moving up**, then the cat would end up rising above the ceiling instead of bumping against it.

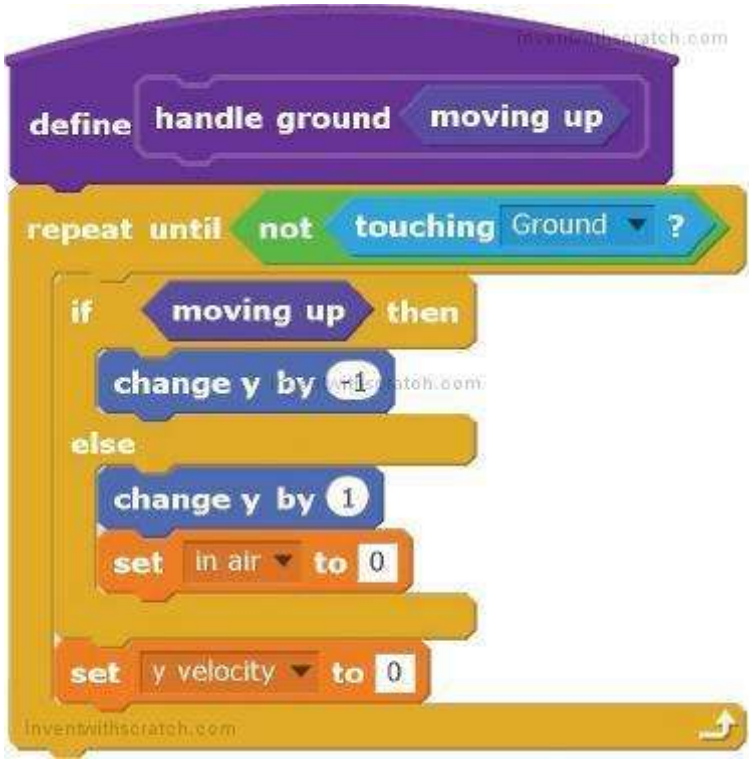
Right-click the **define handle ground** block and select **edit** from the menu.



Click the gray triangle next to **Options** to expand the window, and click the button next to **Add boolean input.** Name this new input field **moving up** and then click **OK**.

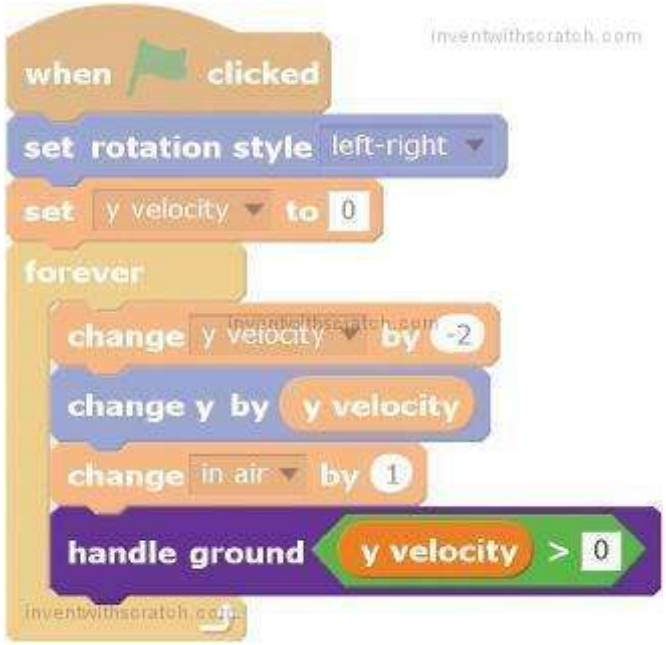


This will add a new **moving up** block that you can drag off the **define handle ground** block just like you do with blocks from the Blocks Area. This **moving up** block will be used in a new **if then else** block. Modify the **define handle ground** block's code to match this.



If the cat is moving up, the **change y by -1** block makes the cat look like it's bumping its head. Otherwise, the script behaves as it did previously, raising the cat so it sits above the ground.

Next, we have to edit the **handle ground** call in this script. We'll add a Boolean condition to determine whether the sprite is moving up, which is **y velocity > 0**.



The **handle ground y velocity > 0** block sets the **moving up** input to true if the y velocity is greater than 0 (that is, if the sprite is jumping and moving up). If y velocity is not greater than 0, then the sprite is either falling down or still, causing the **moving up** input to be set to false.

This is how the **define handle ground** block decides whether it should run **change y by -1** (so the Cat sprite can't go up through the ceiling) or run **change y by 1** (so the Cat sprite is lifted up out of the ground). Either way, if the Cat sprite is touching the Ground sprite (which it is if the code inside the **repeat until not touching Ground** block is running), the y velocity variable should be set to 0 so that the Cat sprite stops falling or jumping.

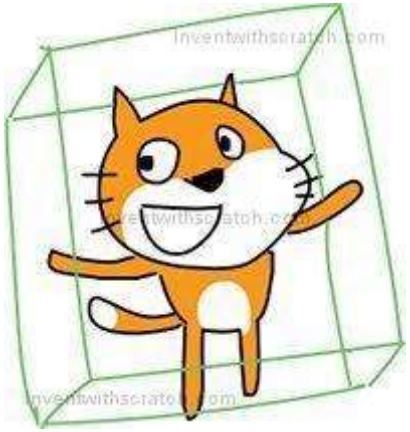
SAVE POINT

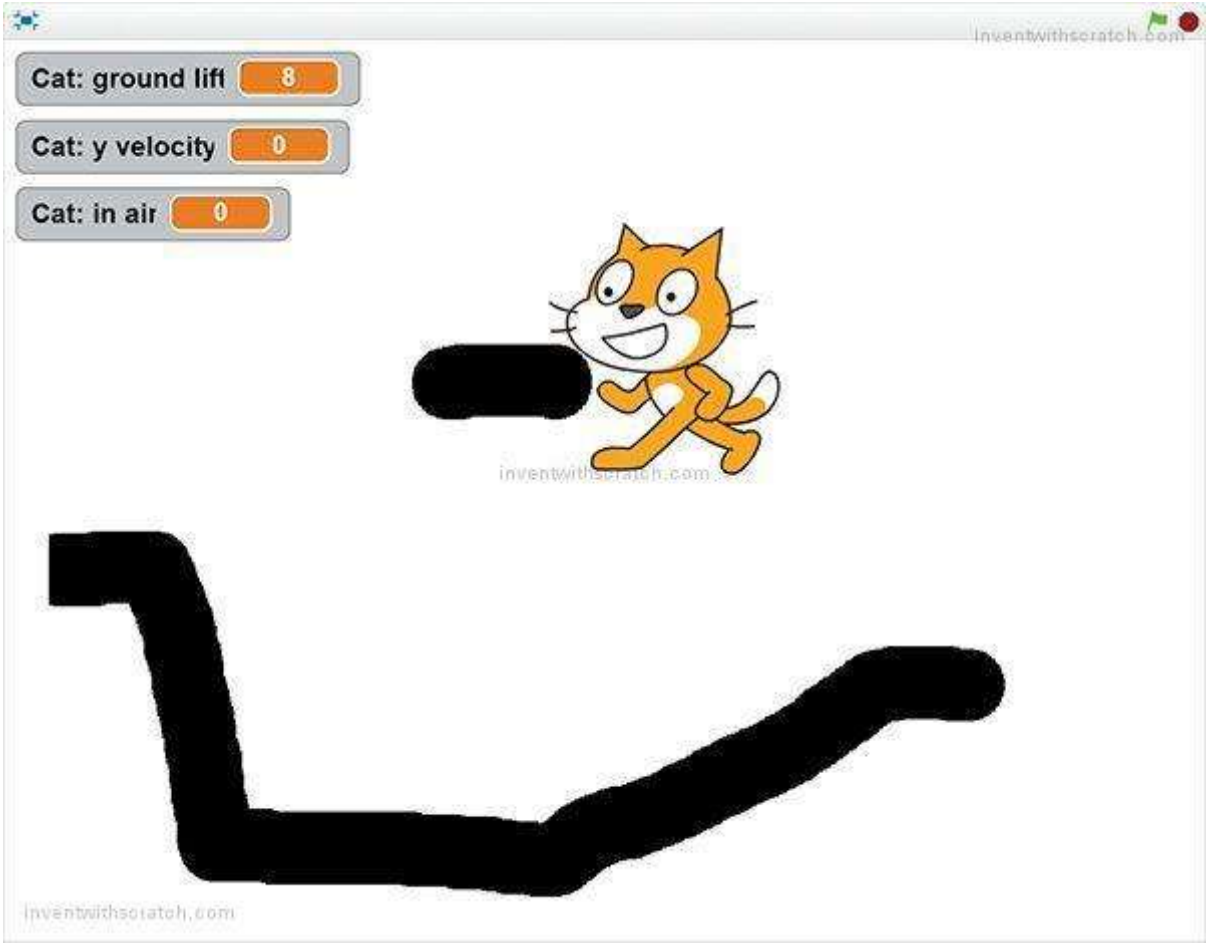
Click the green flag to test the code so far. Make the cat walk under the low platform and jump. Make sure the cat bumps into the platform but does not go above it. Then click the red stop sign and save your program.

If you're lost, open *platformer5.sb* in the resources ZIP file and continue reading from this point.

USE A HITBOX FOR THE CAT SPRITE

There’s another problem with the game. Because the code relies on the Cat sprite touching the Ground sprite, any part of the Cat sprite can be “standing” on the ground, even its whiskers or cheek! In this figure, the cat isn’t falling because its cheek has “landed” on the platform, which isn’t very realistic.

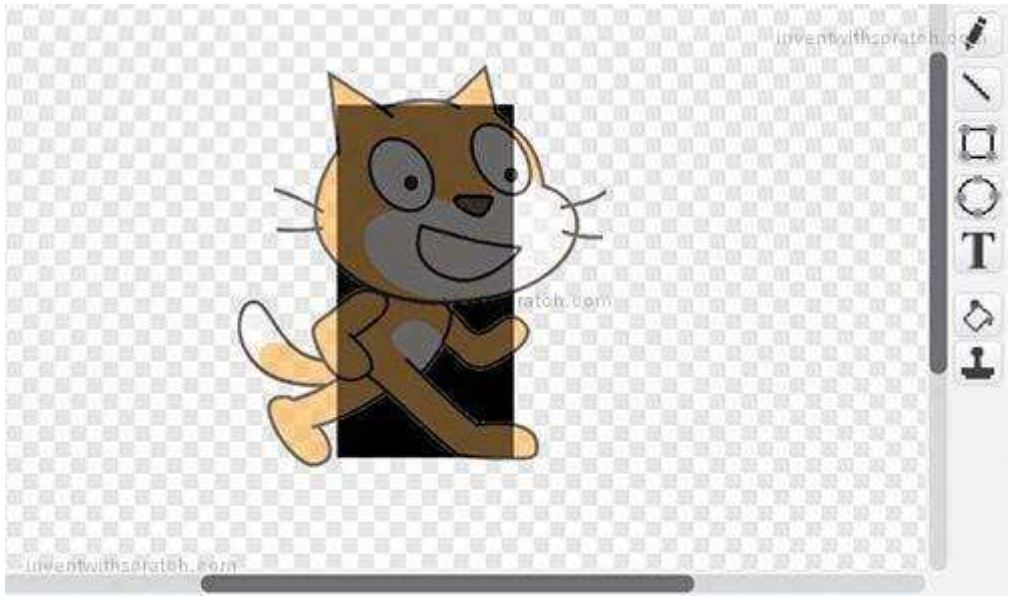




Fortunately, we can fix this problem by using the hitbox concept we used in the *Basketball* game.

9. Add a Hitbox Costume to the Cat Sprite

Click the Cat sprite's **Costumes** tab. Then click the **Paint new costume** button and draw a black rectangle that covers most (but not all) of the area of the other two costumes. The following figure shows a slightly transparent first costume in the same image so you can see how much area the black rectangle covers.

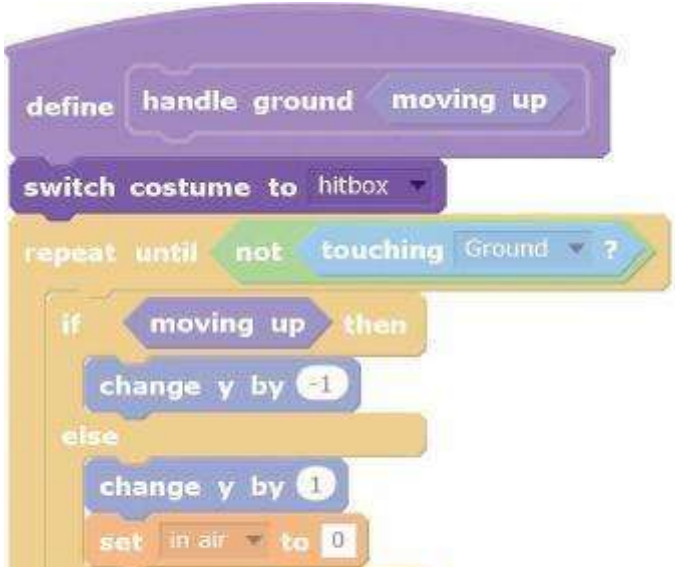


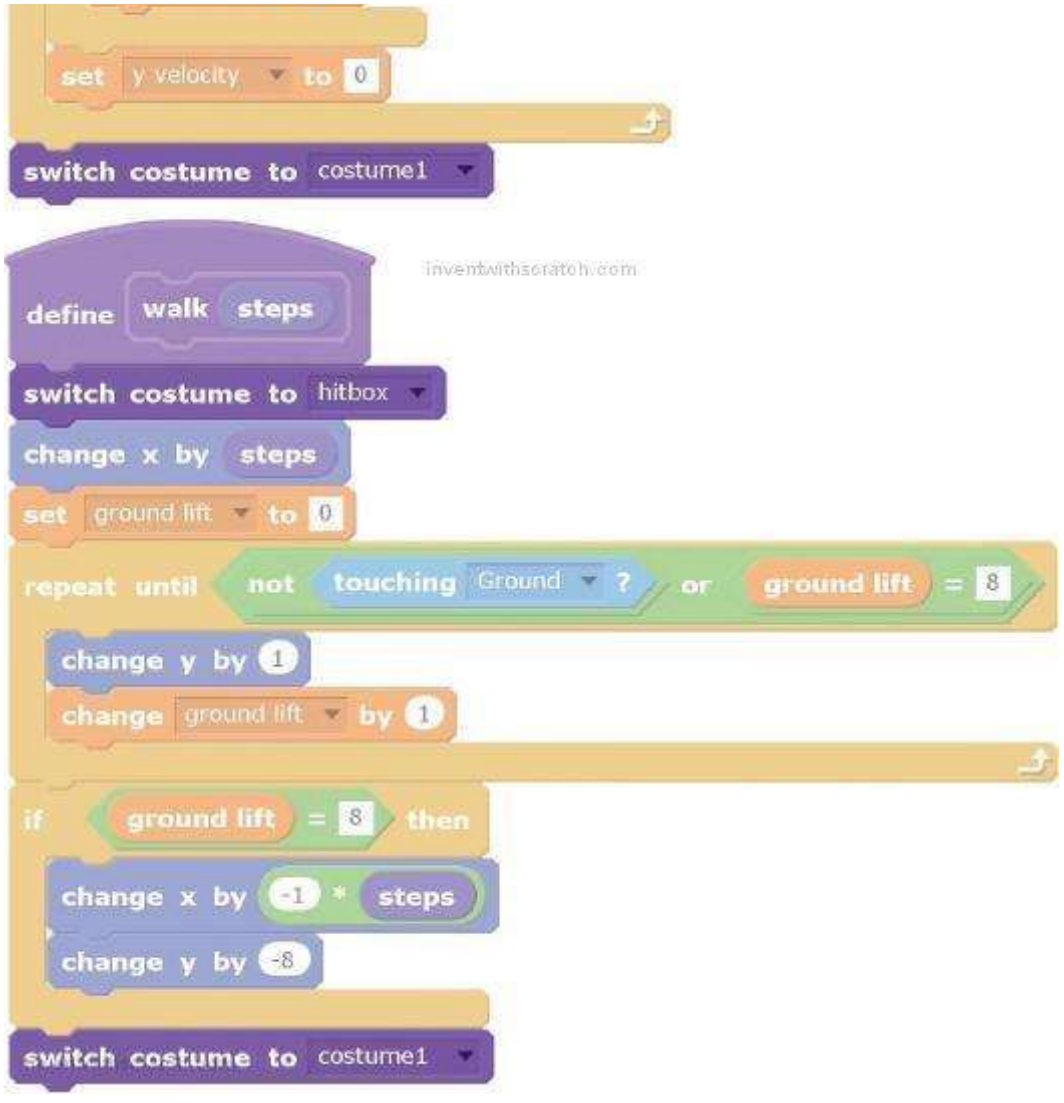
Name this costume hitbox. Whenever the *Platformer* program's code checks whether the Cat sprite is touching the Ground sprite, we'll switch the costume to the black rectangle hitbox costume before the check and then back to the regular costume after the check. By doing so, the hitbox will determine whether the cat is touching the ground.

These costume switches will be handled with the dark purple custom blocks that have the option Run without screen refresh checked, so the hitbox costume will never be drawn on the screen.

10. Add the Hitbox Code

We'll add **switch costume to** blocks to the start and end of both dark purple custom blocks. Modify the Cat sprite's code to look like this:





These blocks from the purple *Looks* category will change the costume to the hitbox. Because the hitbox is a simple rectangle that doesn't have protruding parts that could "catch" on platforms, like the cat's head and whiskers could, the game will behave in a more natural way.

SAVE POINT

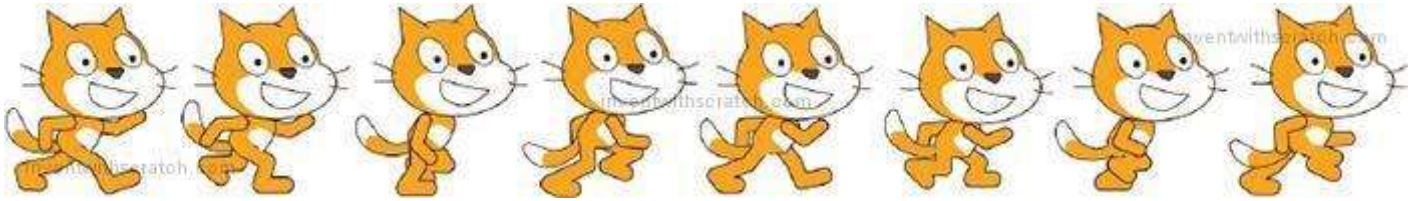
Click the green flag to test the code so far. Make the cat jump around, and make sure the cat can't hang off the platform by its cheek or tail. Then click the red stop sign and save your program.

If you're lost, open *platformer6.sb* in the resources ZIP file and continue reading from this point.

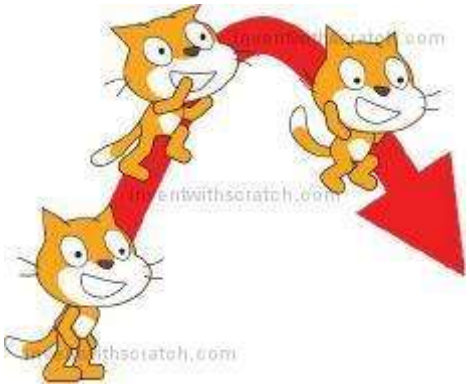
ADD A BETTER WALKING ANIMATION

The Cat sprite that a Scratch project starts with has two costumes named costume1 and costume2.

You can make a simple walking animation by switching back and forth between these two costumes. However, a Scratcher named griffpatch has created a series of walking costumes for the Scratch cat.



griffpatch has also made costumes for standing, jumping, and falling.



Using these costumes will make the *Platformer* game look more polished than using the two simple costumes that the Cat sprite comes with. We just need to add some animation code that switches between these costumes at the right time. griffpatch has created several cool Scratch programs using these costumes: *https://scratch.mit.edu/users/griffpatch/*.

11. Add the New Costumes to the Cat Sprite

To add the new costumes, you must upload the costume files into your Scratch project. You'll find the eight walking images and the standing, jumping, and falling images in the resources ZIP file. The filenames for these images are *Walk1.svg*, *Walk2.svg*, and so on up to *Walk8.svg*, as well as *Stand.svg*, *Jump.svg*, and *Fall.svg*.

Then, in the Scratch editor, click the Cat sprite's **Costumes** tab. Click the **Upload costume from file** button next to New sprite, and select *Stand.svg* to upload the file. This creates a new sprite with *Stand.svg* as its costume.

Delete the original costume1 and costume2 costumes, but keep the hitbox costume. Put the costumes in the following order (it's important that you match this order exactly):

1. Stand
2. Jump
3. Fall
4. Walk1
5. Walk2
6. Walk3
7. Walk4
8. Walk5
9. Walk6
10. Walk7
11. Walk8

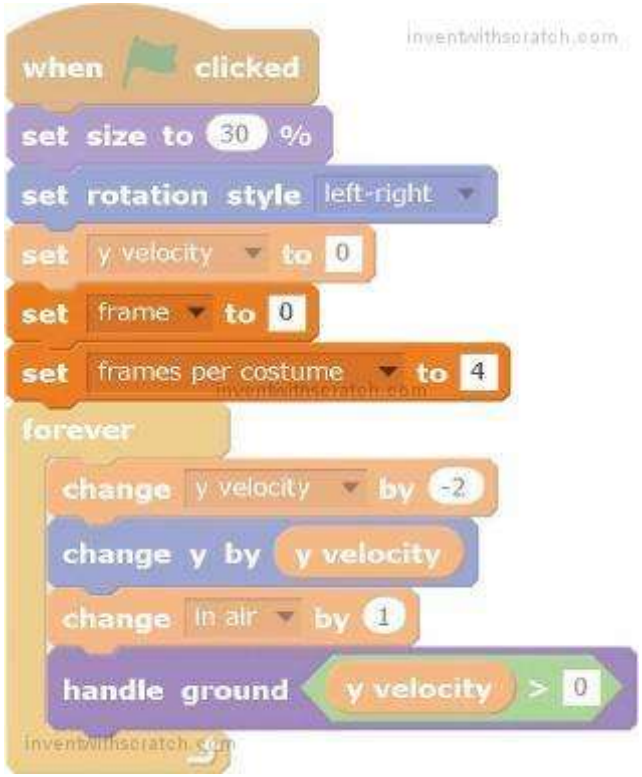
12. hitbox

Each costume has not only a name (like Walk1, Jump, or Fall) but also a number. The costume number is based on the costume's order in the Costumes tab. For example, the costume at the top is named Stand, but it is also known as costume 1. The costume beneath it is named Jump, but it is also known as costume 2. The code we add in the next step will refer to costumes by their names and numbers.

12. Create the Set Correct Costume Block

With all these different costumes, it will be a bit tricky to determine which frame we need to show and when. We'll use the idea of animation frames: several frames shown together quickly make a moving image, just like a flip book.

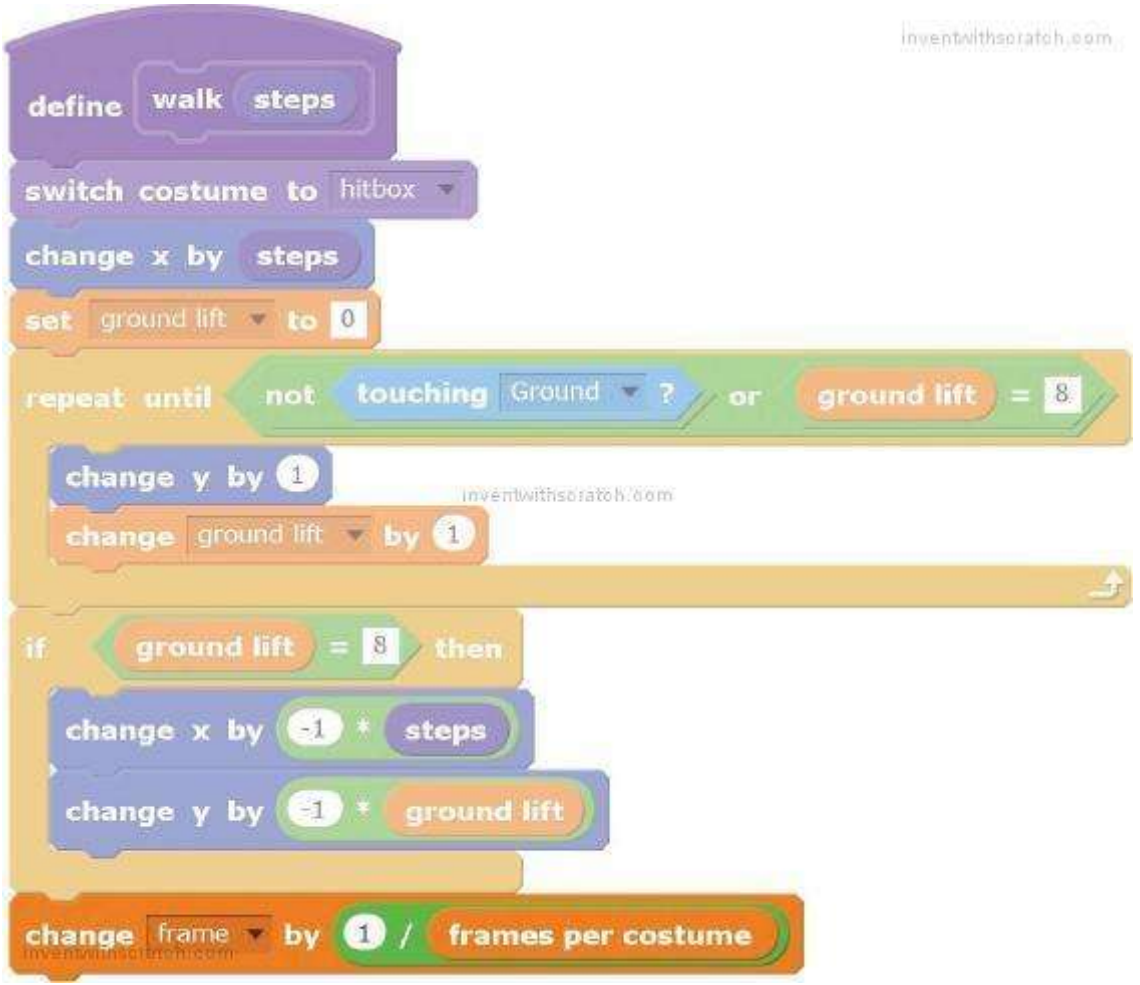
To keep track of the frames, create two For this sprite only variables named frame and frames per costume. Then add two **set** blocks for these initial variables to the Cat sprite's **when green flag clicked** script.



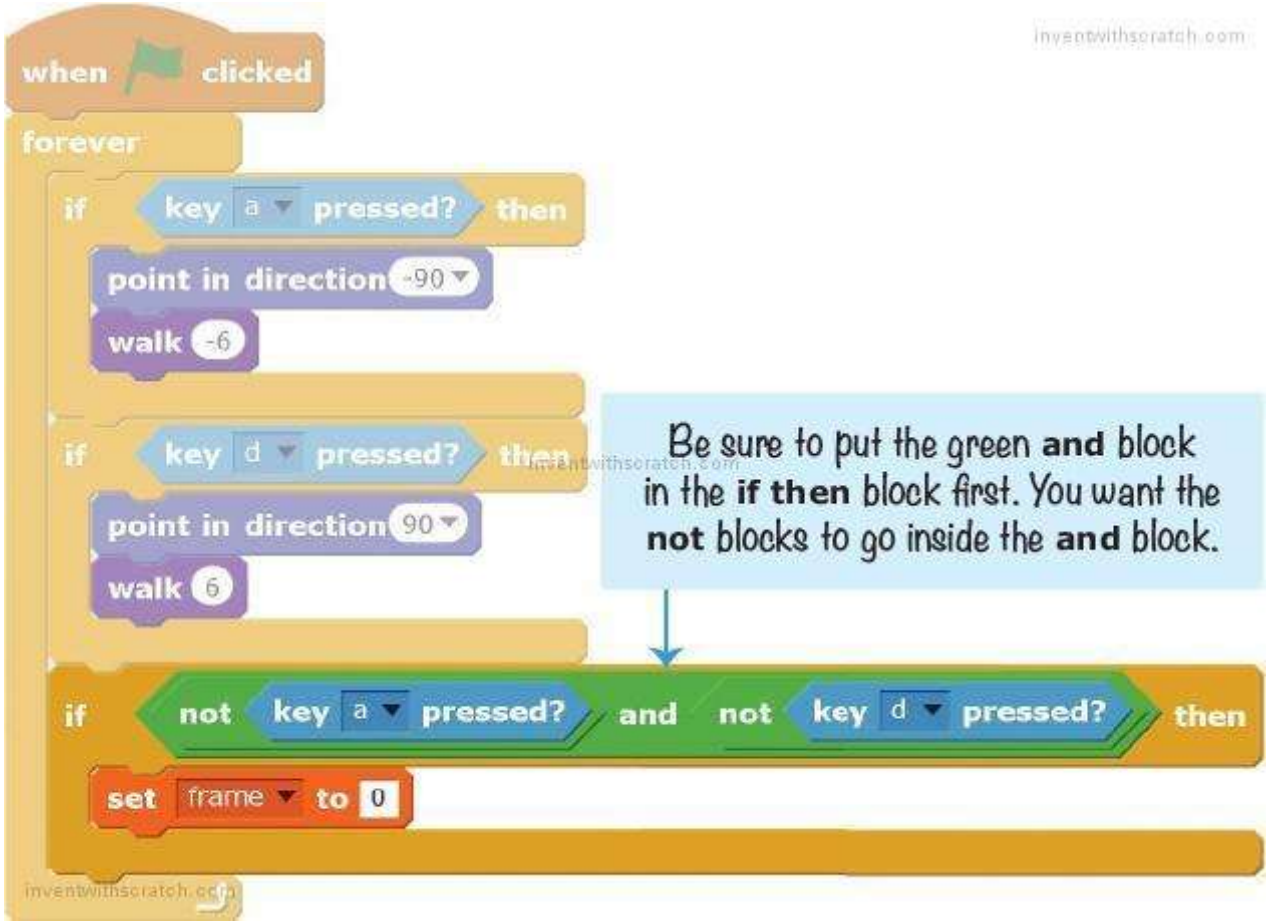
Now the setup is done.

As the player moves the Cat sprite left or right, we want the frame variable to increase. The frames per costume variable keeps track of how fast or slow the animation runs.

Let's edit the code in the **define walk** custom block to increase the frame variable by an amount calculated from frames per costume.

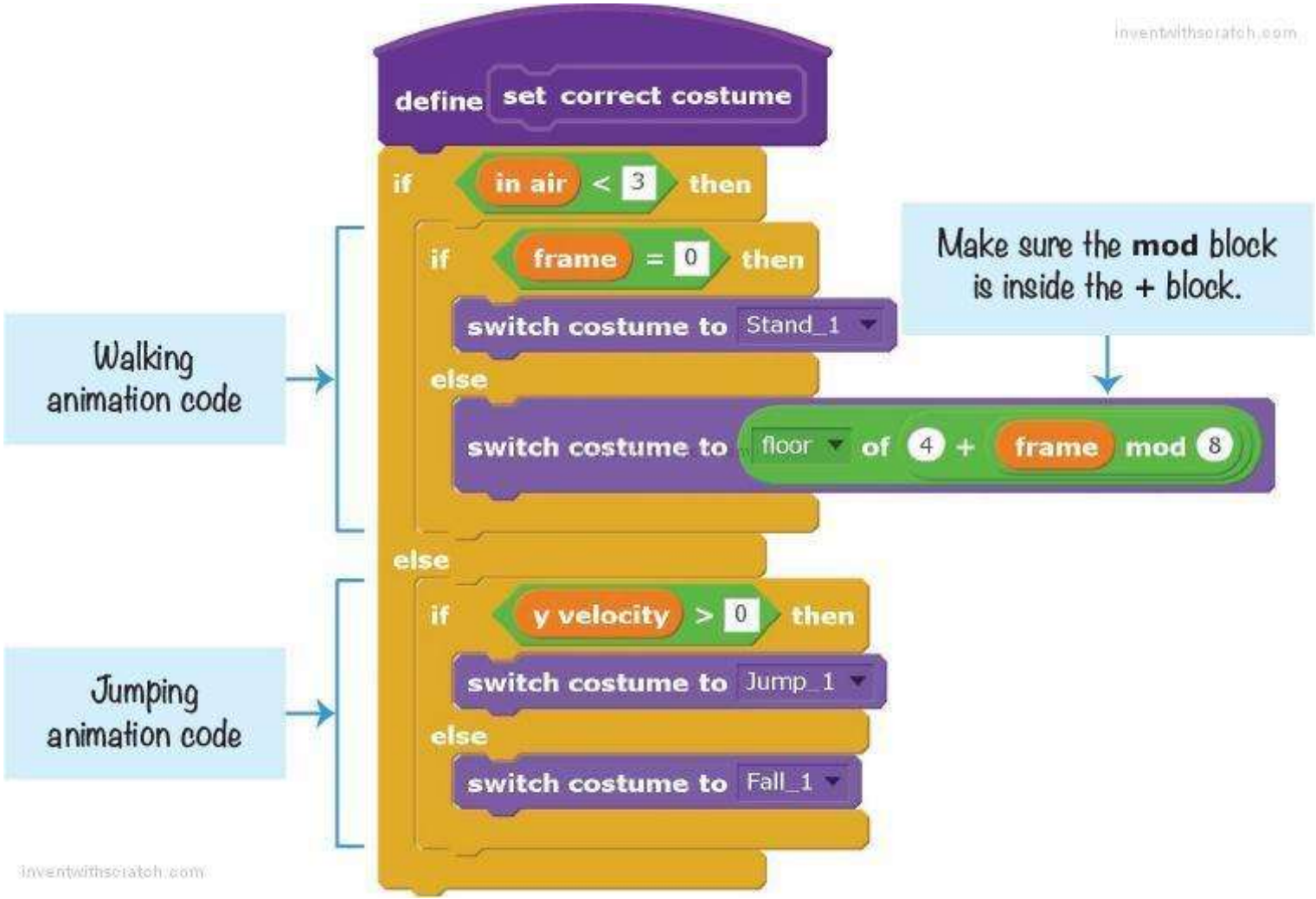


When the cat is standing still (that is, not moving left or right), the frame variable should be reset to 0. Modify the Cat sprite's existing **when green flag clicked** script to add a third **if then** block that resets the frame variable.



Now let's write some code to determine which costume to show. We'll use this code in a few places in the scripts we've written, so let's make a custom block.

In the dark purple *More Blocks* category, click the **Make a Block** button and name this block **set correct costume**. Click the gray **Options** triangle, check the option **Run without screen refresh**, and then click **OK**. Add the following blocks to the Cat sprite, starting with the new **define set correct costume** block.



If the Cat sprite is on the ground (or has just started jumping or falling so that in air is less than 3), then we want to display either the standing costume or one of the walking costumes. Remember that the **when green flag clicked** script keeps setting frame to 0 if the player isn't pressing the A key or D key. So when frame is 0, the **switch costume to Stand** block displays the Stand costume. Otherwise, we must calculate which of the eight walking costumes to show. This calculation refers to costumes by their numbers, which are based on their order in the Costumes tab.

Which walking costume is shown is decided by the **switch costume to floor of 4 + frame mod 8** blocks. Wow, that looks complicated! Let's break it down to better understand each part.

The **mod** block does a *modulo* mathematical operation, which is the remainder part of division. For example, 7 / 3 = 2 remainder 1, so 7 mod 3 = 1 (the remainder part). We'll use **mod** to calculate which costume number to display.

The frame variable keeps increasing, even though we only have eight walking costumes. When frame is set to a number from 0 to 7, we want it to display costumes 4 to 11. This is why our code has the **4 + frame** blocks. But when frame increases to 8, we want to go back to costume 4, not costume 12.

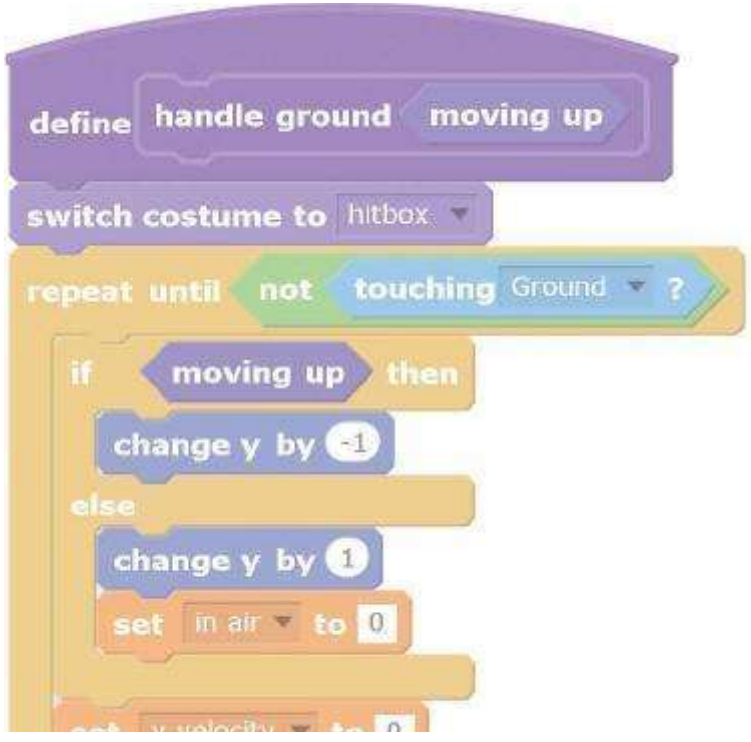
The **mod** block helps us do this wrap around with the costume numbers. We can control the costume that is displayed by using a math trick: because 8 mod 8 = 0, a frame value of 8 will show the first walking costume! We have to add 4 to this number, because the first walking costume is actually costume 4. (Remember, costume 1, costume 2, and costume 3 are the standing, jumping, and falling costumes, respectively.) This sum is then used with the **floor** block.

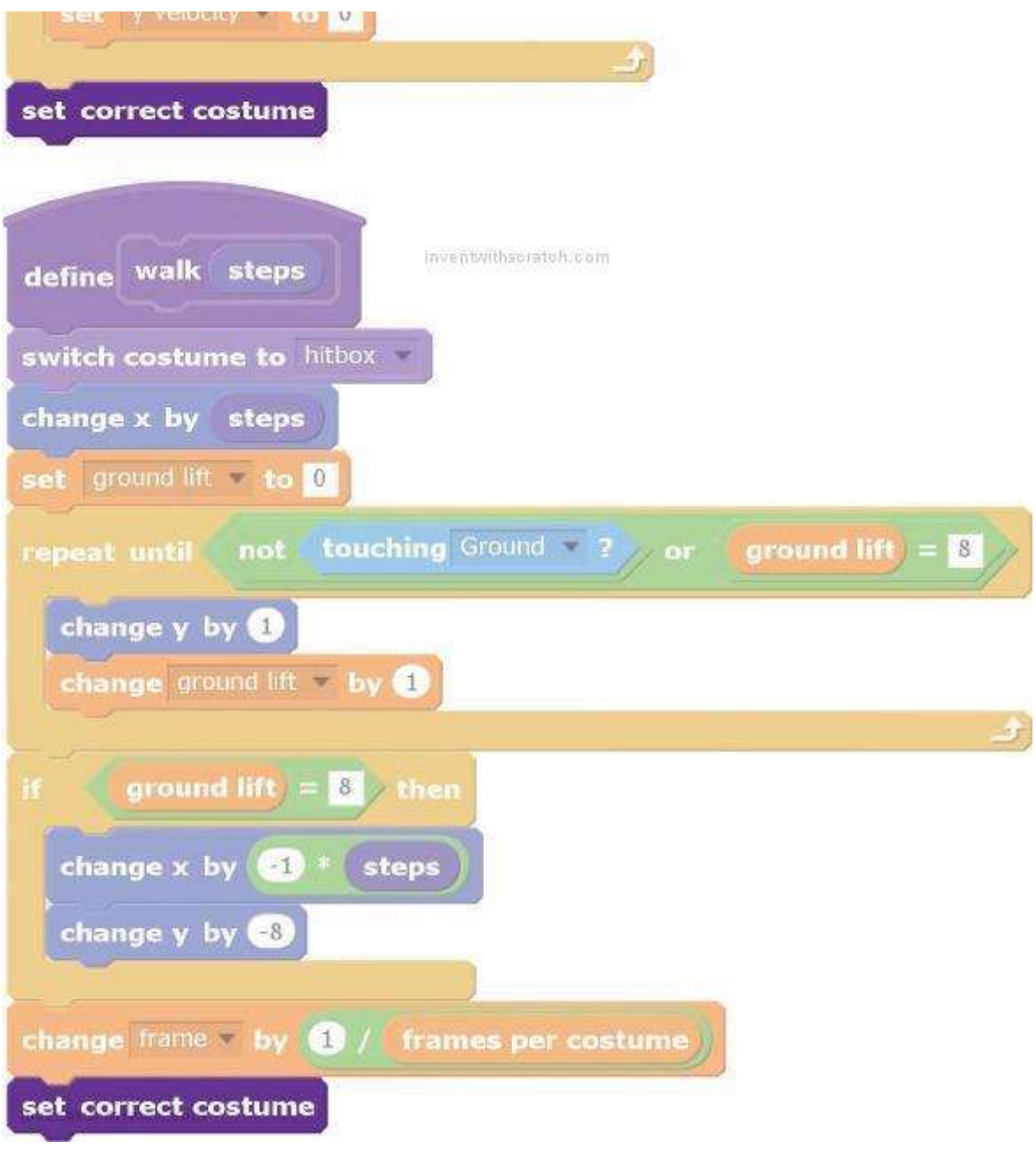
Floor is a programming term that means “round down.” Sometimes **frame** will be set to a number like 4.25 or 4.5, and so **4 + frame** would be 8.25 or 8.5, but we just want to round down to show costume 8.

Whew! That’s the most math you’ve seen in this book so far, but when it’s broken down, it becomes easier to understand.

The code in the **else** part of the **if then else** block handles what happens if in air is greater than or equal to 3. We check y velocity to see if the cat is falling (that is, if y velocity is less than or equal to 0) or jumping (that is, if y velocity is greater than 0) and switch to the correct costume. Finally, the **define set correct costume** code finishes.

Replace the **switch costume to costume1** blocks in the **define handle ground** and **define walk** blocks with the new **set correct costume** blocks. Also, add the **change frame by 1 / frames per costume** blocks so that the frame variable increases over time, as shown here.




set y velocity to 0
set correct costume
define walk steps
inventwithscratch.com
switch costume to hitbox
change x by steps
set ground lift to 0
repeat until not touching Ground ? or ground lift = 8
change y by 1
change ground lift by 1
if ground lift = 8 then
change x by -1 * steps
change y by -8
change frame by 1 / frames per costume
set correct costume

SAVE POINT

Click the green flag to test the code so far. Move the cat around the Stage, and make sure the walking animation displays correctly. Also, make sure the standing, jumping, and falling costumes are shown at the right times. Then click the red stop sign and save your program.

If you're lost, open *platformer7.sb* in the resources ZIP file and continue reading from this point.

CREATE THE LEVEL

The new walking animation makes the *Platformer* game look more appealing. Now let's change the plain white background into a real level. What's great about the code we've written for the Cat sprite to walk, jump, and fall is that it will work with a Ground sprite of any shape or color. So if we change the Ground sprite's costume (say, for different levels) we don't have to reprogram the Cat sprite!

13. Download and Add the Stage Backdrop

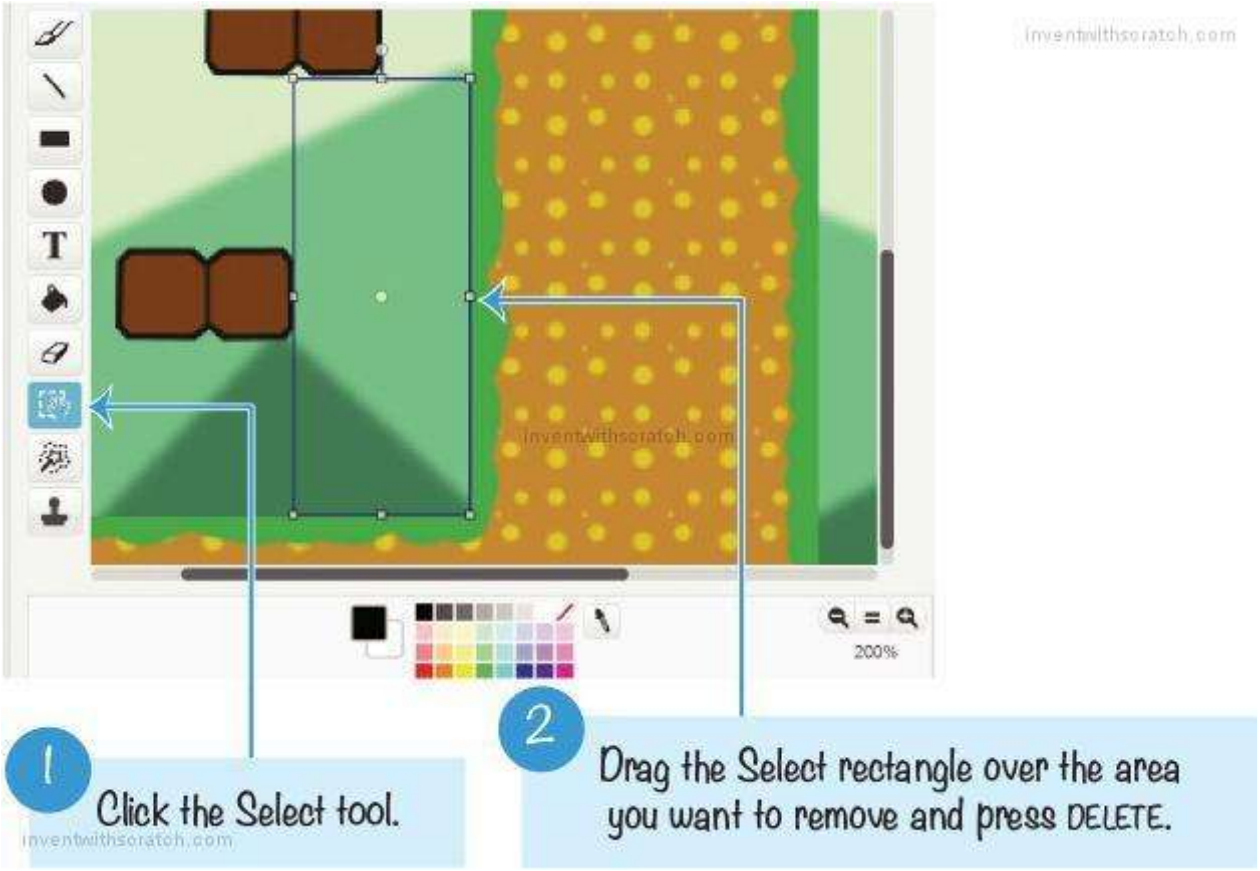
Click the Ground sprite's **Costumes** tab. Click the **Upload costume from file** button and select *PlatformerBackdrop.png*, which is in the resources ZIP file. After this costume has uploaded, you can delete the previous costume.

It's not enough to add the file *PlatformerBackdrop.png* as a costume for the Ground sprite. You must also upload it as a Stage backdrop. Click the **Upload backdrop from file** button next to New backdrop, and select *PlatformerBackdrop.png* to upload it. We need to upload the file to both places because we'll be erasing all the "background parts" from the Ground sprite in the next step. We only need the Ground sprite to mark which parts the Cat sprite can walk on. The backdrop will be the image that is displayed on the Stage.

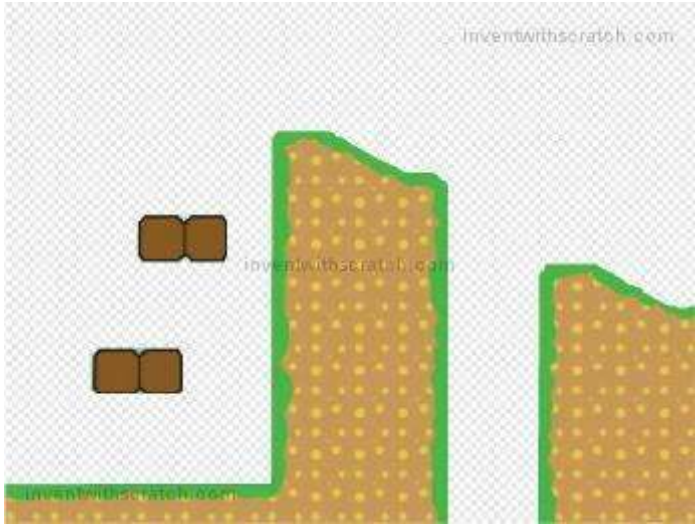
14. Create a Hitbox Costume for the Ground Sprite

The *Platformer* game code is based on the Cat sprite touching the Ground sprite. The Ground sprite's costume is a hitbox, so if the Ground sprite's costume is a full rectangle that takes up the entire Stage, it will be treated as though the entire Stage is solid ground. We need to remove the parts of the Ground sprite's costume that are part of the background and not platforms.

The easiest way to do this is to click the Select tool in the Paint Editor. Drag a Select rectangle over the part of the costume you want to delete. After selecting an area, press DELETE to remove that piece.



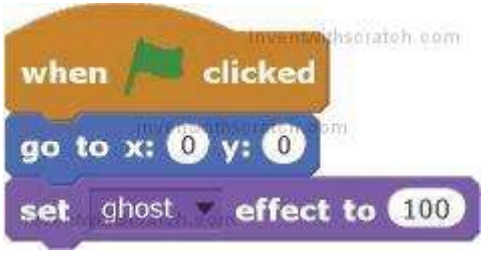
Use the Eraser tool to erase areas that aren't rectangular. If you make a mistake, click the Undo button at the top of the Paint Editor to undo the deletion. Keep removing the background parts of the costume until only the platform parts remain.



If you're having trouble creating this costume, you can use the premade *PlatformerBackdropHitbox.png* image in the resources ZIP file. The background parts of the image are already deleted, so you just need to click the **Upload costume from file** button on the Costumes tab to add it.

15. Add the Ground Sprite's Code

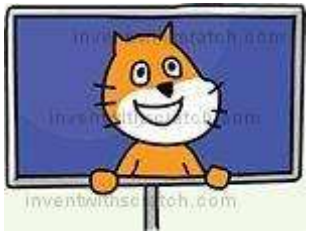
The Stage backdrop is used to set the appearance of the platforms and background. The Ground sprite is used to identify which parts are solid ground that the Cat sprite can walk on. Add the following code to the Ground sprite's Scripts Area:



The Ground sprite's costume needs to line up perfectly over the Stage's backdrop so that you can't tell it's there. Because the Stage backdrop and the Ground sprite's costume came from the same image file, you can do this by moving the Ground sprite to the coordinates (0, 0). Otherwise, the Ground sprite's costume won't line up perfectly over the backdrop.

The *drawing* of the Ground sprite's costume doesn't matter as much as the *shape* of the costume. As long as the Ground sprite is lying perfectly on top of the backdrop, we can set the ghost effect to 100, and the Ground costume and backdrop will line up. The backdrop shows what the level looks like, while the Ground sprite acts as its hitbox.

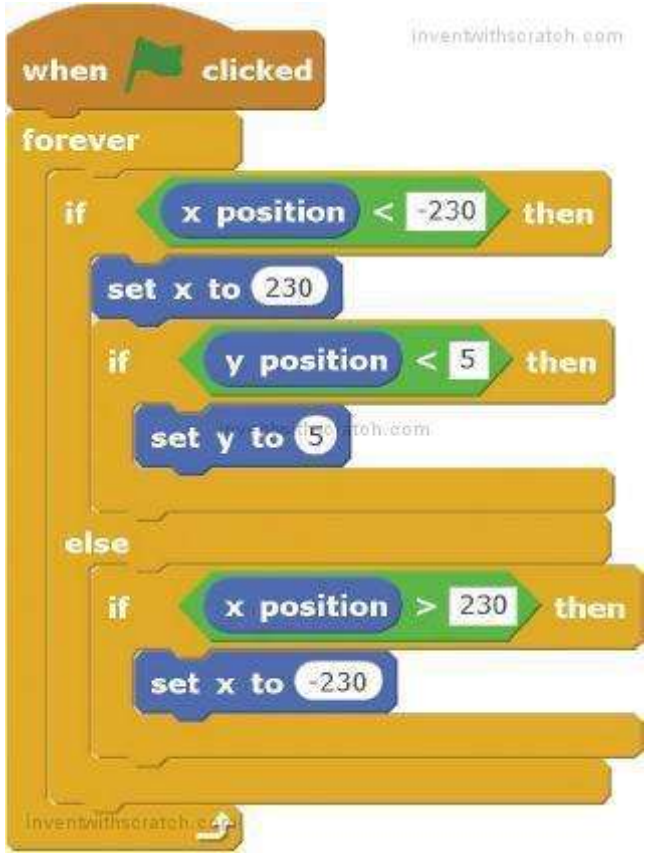
SAVE POINT



Click the green flag to test the code so far. Make sure you can move the cat all around the Stage. Then click the red stop sign and save your program.

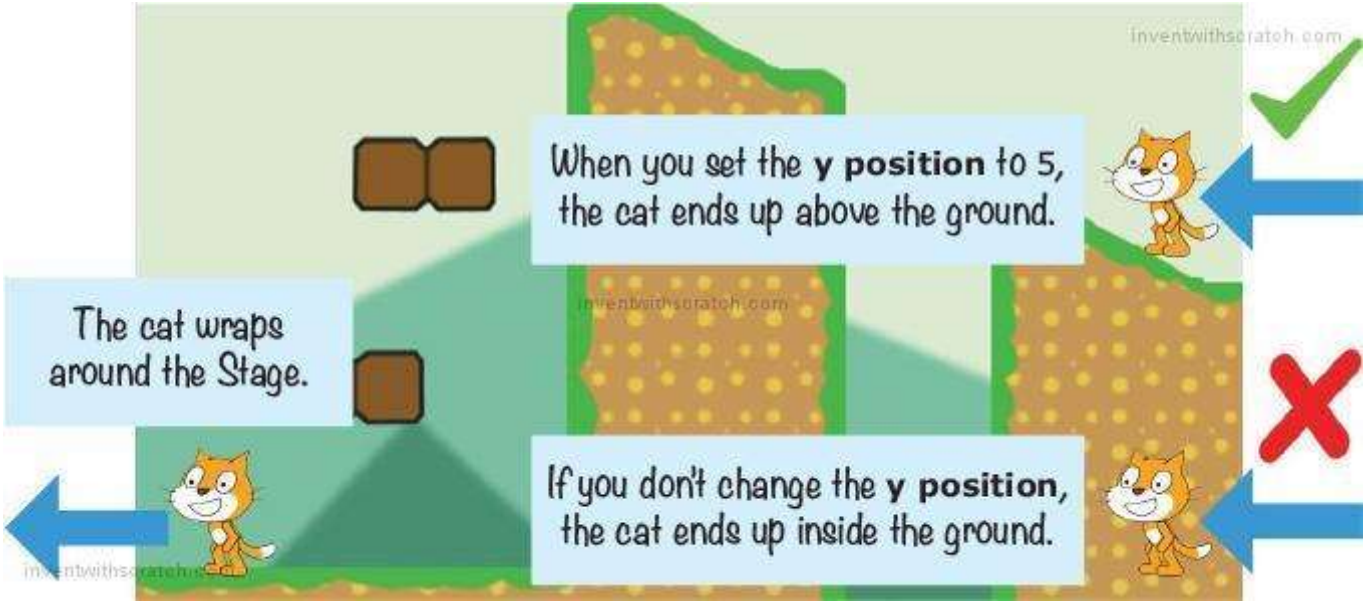
16. Add More Wrap-Around Code to the Cat Sprite

Notice that the level has a couple floating platforms and a hill with a pit in the middle. When the cat falls down the pit, it wraps around the Stage and reappears at the top. Let's add wrap-around code for the left and right edges of the Stage as well. Add the following script to the Cat sprite.



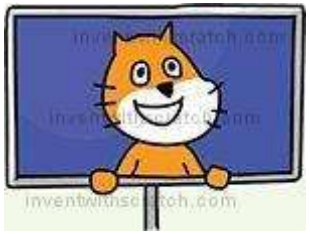
When the cat walks to the left edge of the Stage, its **x position** will be less than -230. In that case, we make it wrap around to the right edge by setting the **x position** to 230.

Also, if the cat is moving from the left side of the Stage, its **y position** will be less than 5. This will put it inside the ground when moved to the right edge of the Stage, so an **if then** block checks for this condition and sets the **y position** to 5.



The other **if then** block wraps the Cat to the left edge of the Stage if it's on the right edge (that is, its **x position** is greater than 230.)

SAVE POINT



Click the green flag to test the code so far. Make sure the cat can walk off the left edge of the Stage and wrap around to the right, and vice versa. Then click the red stop sign and save your program.

If you're lost, open *platformer8.sb* in the resources ZIP file and continue reading from this point.

ADD CRAB ENEMIES AND APPLES

The entire *Platformer* game setup is complete! The player can make the Cat sprite walk, jump, fall, and stand on platforms. The Cat sprite has some cool animations, and the backdrop looks like a real video game.

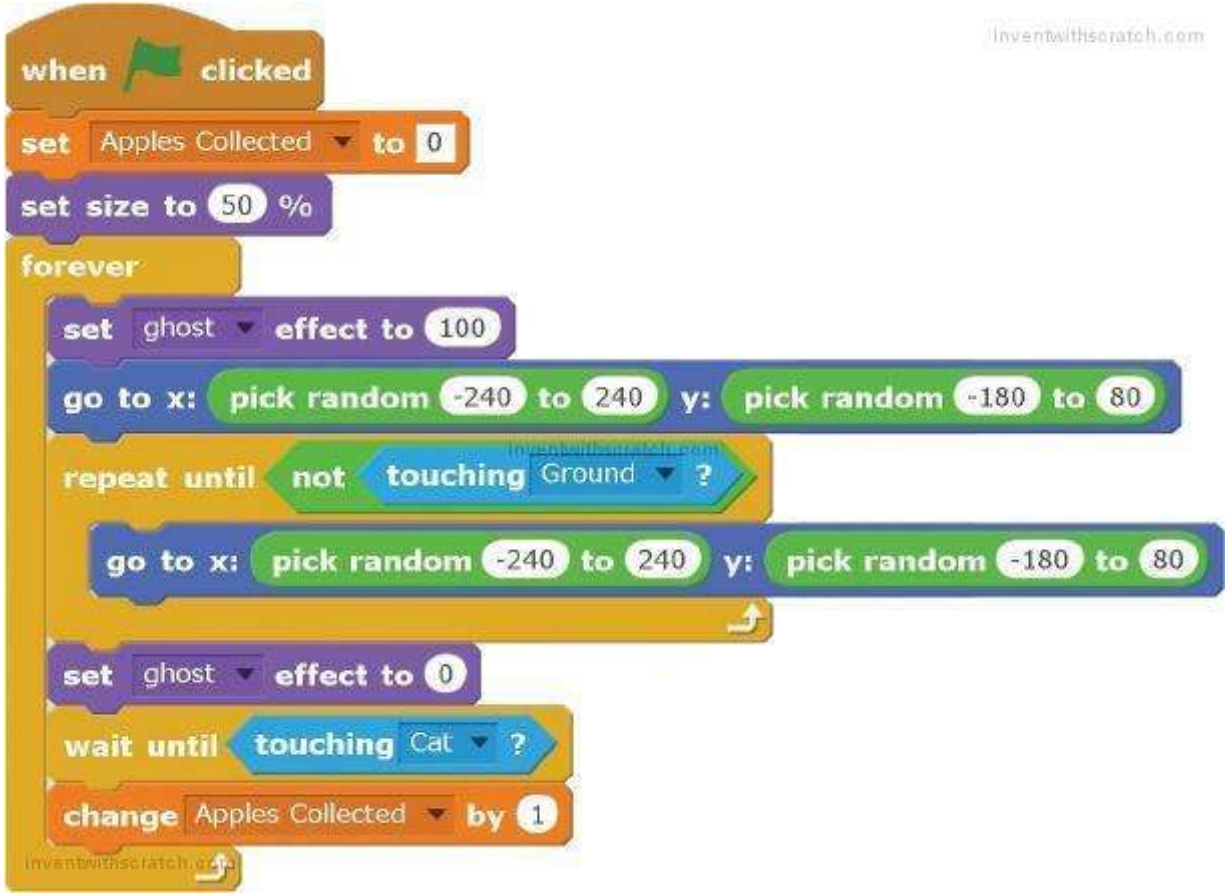
Now all we have to do is make a game using all the pieces we have. We'll add an apple that appears randomly around the Stage (similar to the *Snaaaaaake* game in Chapter 6) and add a few enemies that try to touch the Cat sprite and steal the apples.

17. Add the Apple Sprite and Code

Click the **Choose sprite from library** button and select the Apple sprite from the Sprite Library window that appears; then click **OK**.

As in previous games, we'll use a variable to track the game's score. Click the orange *Data* category, and then click the **Make a Variable** button to create a For all sprites variable named Apples Collected. This variable will keep track of the player's score.

In the Apple sprite's Scripts Area, add the following code.



At the start of the game, when the player clicks the green flag, the score in Apples Collected is set to 0. Also, because the Apple sprite is a bit too big, we set its size to 50 percent.

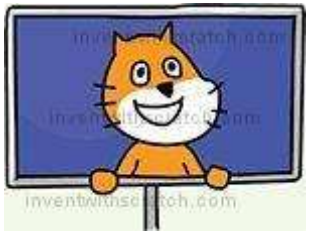
During the game, the Apple sprite needs to appear in random places on the Stage. We make the Apple sprite invisible using **set ghost effect to 100**. It then moves to a random place on the Stage.

But not just any random place on the Stage will do. We need to make sure the Apple sprite is not inside the Ground sprite, because it would be impossible for the player to get it. To prevent the Apple sprite from moving to somewhere inside the Ground sprite, the loop keeps trying new random places until the

Apple sprite is no longer touching the Ground sprite. The movement of the apple won't be visible to the player, because the ghost effect is still set to 100. When the Apple sprite finds a place that is not touching the Ground sprite, it becomes visible again with **set ghost effect to 0**.

Then the Apple sprite waits until the Cat sprite touches it. When that happens, the score in Apples Collected increases by 1, and the Apple sprite loops to find a new random place on the Stage.

SAVE POINT



Click the green flag to test the code so far. Make sure the Apple sprite never appears inside the ground. When the cat touches the apple, the score in Apples Collected should increase by 1, and the Apple sprite should move to a new random place. Click the red stop sign and save your program.

18. Create the Crab Sprite

The game wouldn't be very challenging if all the player had to do was jump around and collect apples. Let's add some enemies that the player must avoid.

Right-click the Cat sprite and select **duplicate**. The enemies will use the same code as the Cat sprite so that they can jump and fall on the platforms. (We'll remove the code that assigns the keyboard keys to control the sprite and replace it with code that moves the crabs around randomly.) Rename this duplicated sprite Crab. Then, in the Costumes tab, click the **Choose costume from library** button, select the crab-a costume, and click **OK**. Then open the Costume Library again and select the crab-b costume.

The Crab sprite will still have all the Cat sprite costumes (Stand, Fall, Walk1, and so on). Delete these Cat sprite costumes, including the hitbox costume. Create a new hitbox costume that is the right size for the crab. Here's what the hitbox costume should look like (the crab-a costume has been placed over it so you can see the relative size, but the costume is just the black rectangle).

19. Create the Enemy Artificial Intelligence

In games, *artificial intelligence (AI)* refers to the code that controls the enemies' movements and how they react to the player. In the platform game, the AI for the crabs is actually pretty *un*intelligent: the crabs will just move around randomly.

In the orange *Data* category, click the **Make a Variable** button and create a For this sprite only variable named movement. The movement variable will store a number representing the Crab sprite's movements:

1. Walk left
2. Walk right
3. Jump up straight

4. Jump to the left
5. Jump to the right
6. Stay still

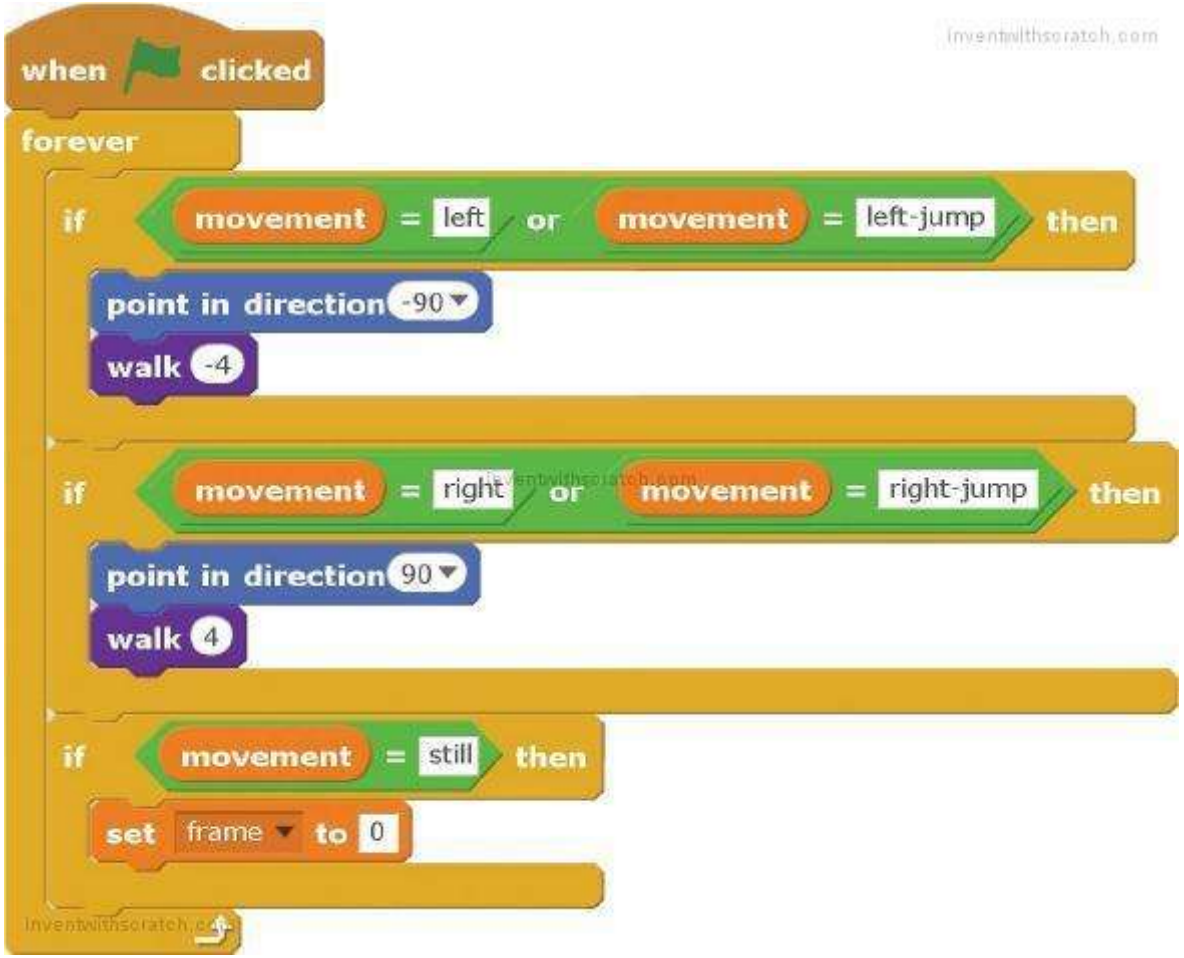
The Crab sprite’s movements will be randomly decided and will frequently change. Add the following code to the Crab sprite.

The Crab sprite will wait a random amount of time between 0.2 and 0.8 seconds before deciding on a new random move to make.

At first, the movement variable is set to a random number between 1 and 6 that decides which movement the crab will make.

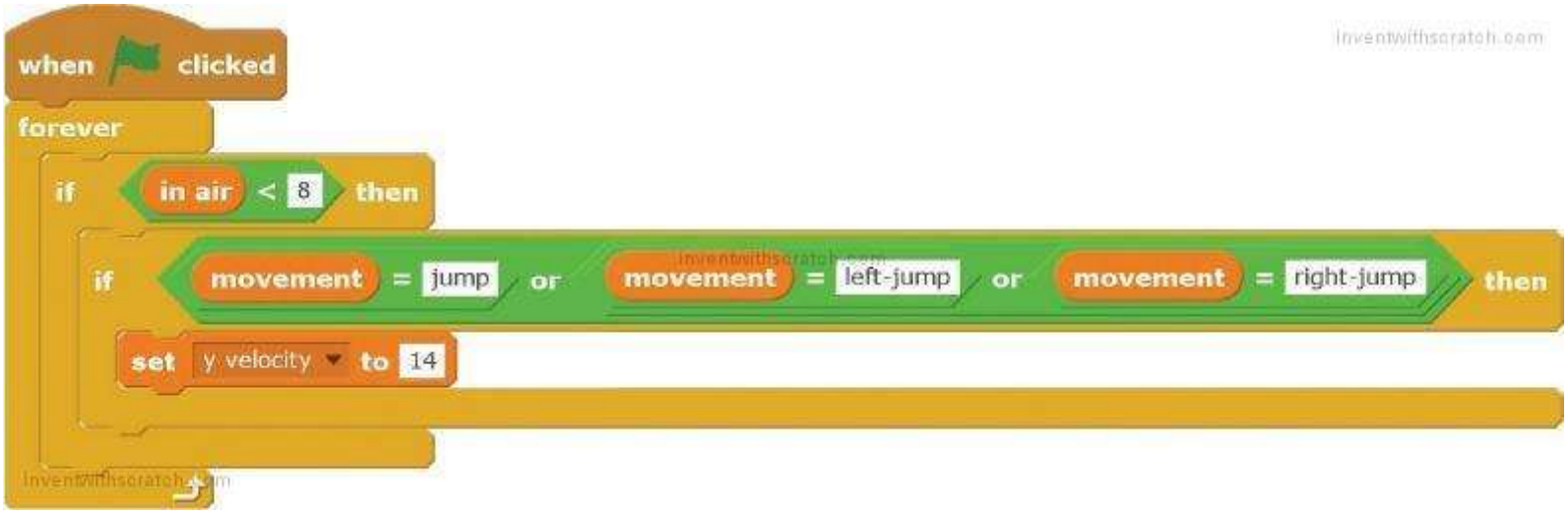
The rest of the Crab sprite's code defines these movements. Find any code from the Cat sprite code that used the **when key pressed** blocks and replace those blocks with blocks that check the movement variable. (If you ran the program right now, the keyboard keys would control the Cat *and* Crab sprites, because they have the same code!)

Modify the Crab sprite script that checks whether the player is pressing the A key or D key to match the following code.



As with the Cat sprite, this code lets the Crab sprite walk left and right. Change the values in the **walk** blocks to -4 and 4 to make the crab move slower than the player.

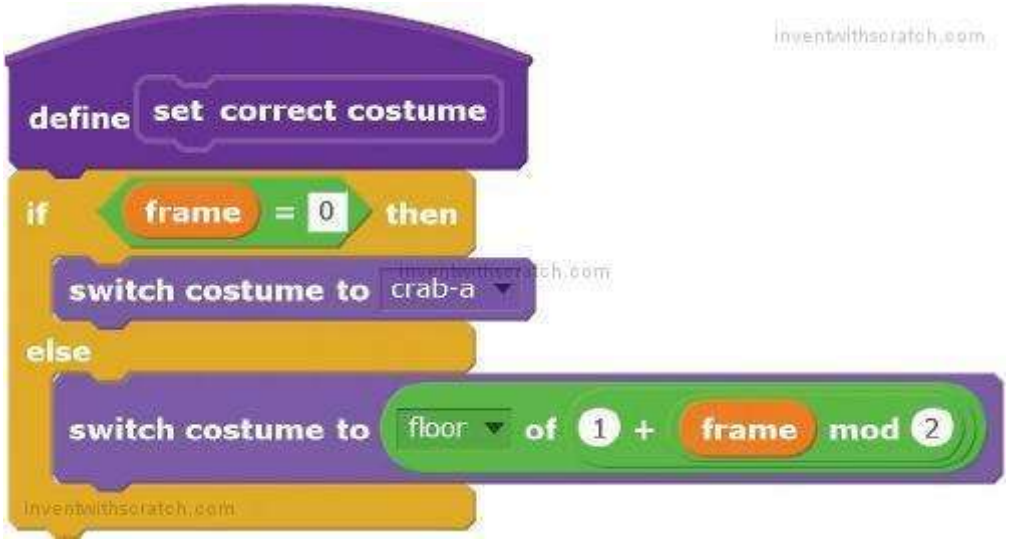
Then change the script that handles the player pressing the W key to jump to match the following code.



This code lets the Crab sprite jump up, left, and right.

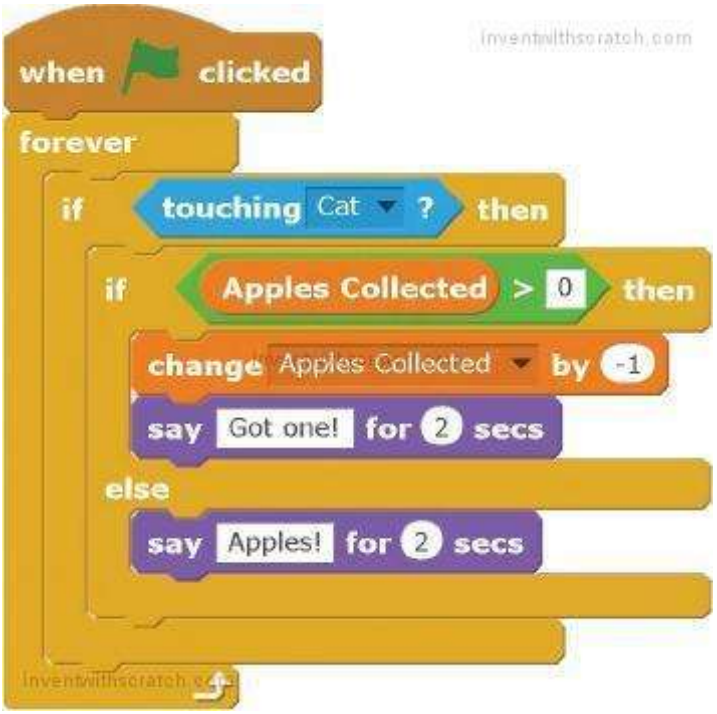
Now let's animate the crab's movements. The Crab sprite has only two costumes, crab-a and crab-b. We'll switch between these two costumes to make it look like the crab is walking. We can simplify the **set correct costume** block for the Crab sprite a bit.

Modify the **set correct costume** code to look like the following:



Notice that the numbers in the **floor of 1 + frame mod 2** blocks have also changed. The first costume is costume 1, and the Crab has only two costumes, so the numbers in these blocks have been changed to 1 and 2.

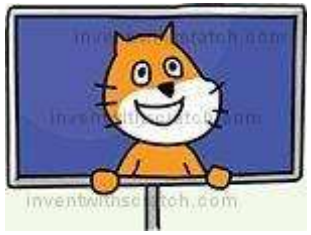
Finally, we need to create a new script so that the crabs can steal apples from the player. Add this code to the Crab sprite.



The Crab sprites subtract 1 from Apples Collected and say “Got one!” when they touch the player. If the player has 0 apples, the Crab sprites will say “Apples!” but will not subtract 1 from Apples Collected.

The game will be a bit more exciting with two crabs, so right-click the Crab sprite in the Sprite List and select **duplicate** from the menu.

SAVE POINT

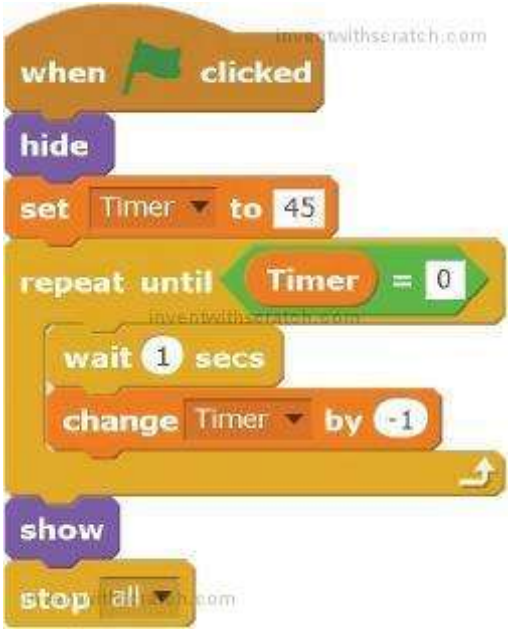


Click the green flag to test the code so far. Make sure the two crabs are jumping around. When they touch the cat, they should steal an apple and say “Got one!” If the cat doesn’t have any apples, the crabs should just say “Apples!” Then click the red stop sign and save your program.

20. Add the Time’s Up Sprite

We’re almost done! The last thing we need to add to the game is a timer. The player will be under pressure to grab apples as quickly as possible instead of playing it safe. Click the **Paint new sprite** button, and draw the text *Time’s Up* in the Paint Editor. Mine looks like this:

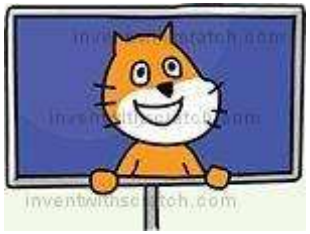
Rename the sprite Time's Up. Then create a For all sprites variable named Timer, and add the following code.



This code gives the player 45 seconds to collect as many apples as possible while trying to avoid the crabs who will steal one. When the Timer variable reaches 0, the Time's Up sprite will appear and the game will end.

Now the *Platformer* game is ready for final testing!

SAVE POINT



Click the green flag to test the code so far. Walk and jump around, collecting apples while trying to avoid the crabs. Make sure that when the timer reaches 0, the game ends. Then click the red stop sign and save your program.

The code for this program is too large to list the complete code in this book. However, you can view the completed code in the resources ZIP file—the filename is *platformer.sb2*.

SUMMARY

You did it! You're a Scratch master! The advanced *Platformer* game is the most elaborate and complex project in this book. You combined and used lots of different concepts to make this game, so it might help to read through this chapter a few more times.

In this chapter, you built a game that

- Uses a ground sprite that the player stands on
- Uses dark purple custom blocks with the Run without screen refresh option enabled
- Lets the player walk up and down slopes
- Has ceiling detection so the player bumps their head on low platforms
- Has detailed animations for walking, jumping, and falling
- Implements AI for enemies so they move around on their own

That wraps it up for this book, but don't let that stop you from continuing your programming adventure. You can always look through other Scratcher's programs to get more ideas. Find a game you like, and try to create it from *scratch*. (All my puns are intended.)

The great thing about Scratch is that it provides you with unlimited possibilities for the types of games you can make. You can create clones of popular classic games like *Pac-Man* or *Flappy Bird*. Or you can make unique games using your own designs. Good luck!

REVIEW QUESTIONS

Try to answer the following practice questions to test what you've learned. You probably won't know all the answers off the top of your head, but you can explore the Scratch editor to figure out the answers. (The answers are also online at *http://www.nostarch.com/scratchplayground/*.)

1. The dark purple custom blocks help you avoid duplicating code. Why is this a good thing?
2. How is a dark purple custom block's input like a variable?
3. Where can you use a dark purple custom block's input?
4. What does *modulo* mean in mathematics?
5. What does *floor* mean in programming?

WHERE TO GO FROM HERE

Ready for more? There are plenty of Scratch resources to keep things exciting.

- **Scratch Programming Playground Studio** (*http://www.inventwithscratch.com/studio/*) is a place to share your own games based on the ones in the book! You can check out other readers' projects while you're there.
- **The Scratch Forums** (*http://scratch.mit.edu/discuss/*) is a place for Scratchers to share ideas and ask and answer questions.
- **Learn to Program with Scratch** by Majed Marji (No Starch Press, 2014; *https://www.nostarch.com/learnscratch/*), is a book with more projects for learning computer science concepts in Scratch.
- **ScratchEd** (*http://scratched.gse.harvard.edu/*), is an online community created for teachers and other educators who use Scratch. Share your success stories, exchange Scratch resources, ask questions, and more.

There are many fun games and animations you can make with Scratch, but it does have some limitations. Your Scratch programs might not look like "real" games you play on a PC, game console, or smartphone.

So it's only natural that you might want to learn how to write code in a professional programming language. There are many languages to choose from, but I recommend Python or JavaScript. Python is the easiest language to learn (aside from Scratch) but it is still a language used by professional software developers. JavaScript is not quite as easy, but it's the language used for web apps that run in your browser.

If you want to learn Python, I recommend a book I wrote: *Invent Your Own Computer Games with Python*. This book is free to read online at *https://inventwithpython.com/*, or you can buy it at *https://www.nostarch.com/inventwithpython/*. If you want to learn JavaScript, I recommend Nick Morgan's *JavaScript for Kids* (No Starch Press, 2014; *https://www.nostarch.com/javascriptforkids/*). These books are great for the next step of your journey to become a master programmer!