Coding & Design with Scratch

Smiling Creek course handbook

11 lessons
Core projects and additional practice

A step-by-step block coding course with instructions, worked examples, and code screenshots. Work at your own pace and use the examples to build your own projects.

Contents

Lesson 1

Smiling Creek: Coding & Design with Scratch - Lesson 1

Lesson overview

Complete this lesson at your own pace.

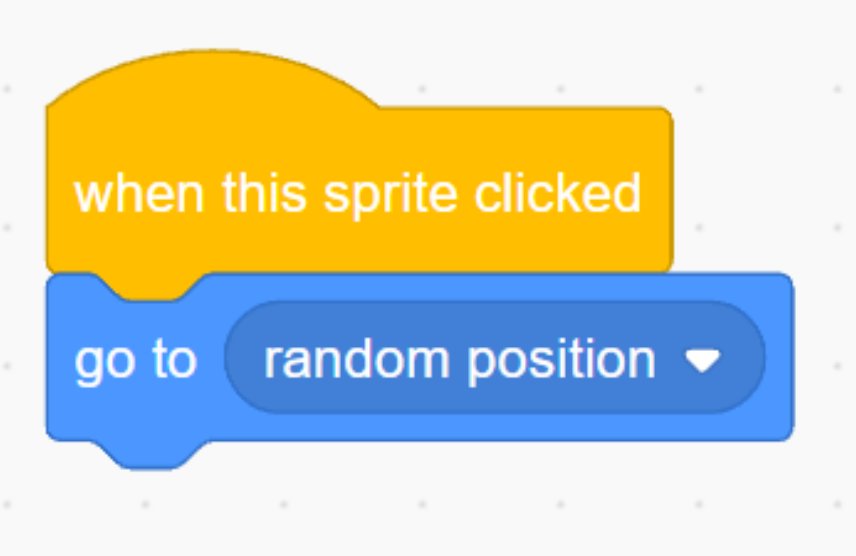
Find the complete code and project link at the end of this lesson.

Follow the steps below to build the project.

This lesson, we got more comfortable with Scratch's workspace, and introduced a few new concepts and ideas.

First, we started off by creating a new sprite, something that we could click on. I chose a balloon, but students could choose whichever sprite they wanted.

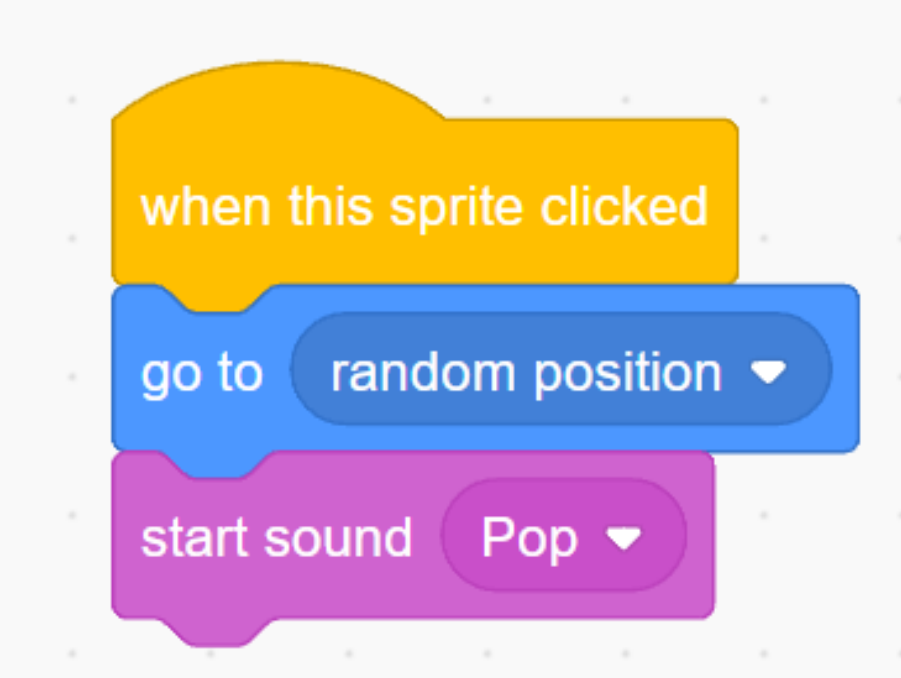
After this, we learned the "when this sprite clicked" block, and we combined this with the, "go to 'random position'" block to create a sprite that jumps around when it is clicked.



Remember that all **blue blocks** have to do with **motion**, and all **yellow blocks** have to do with **control**.

After this, we learned how to play different sounds in Scratch. **Sounds** are a key part in our coding, and we will explore different options for sound effects in later lessons.

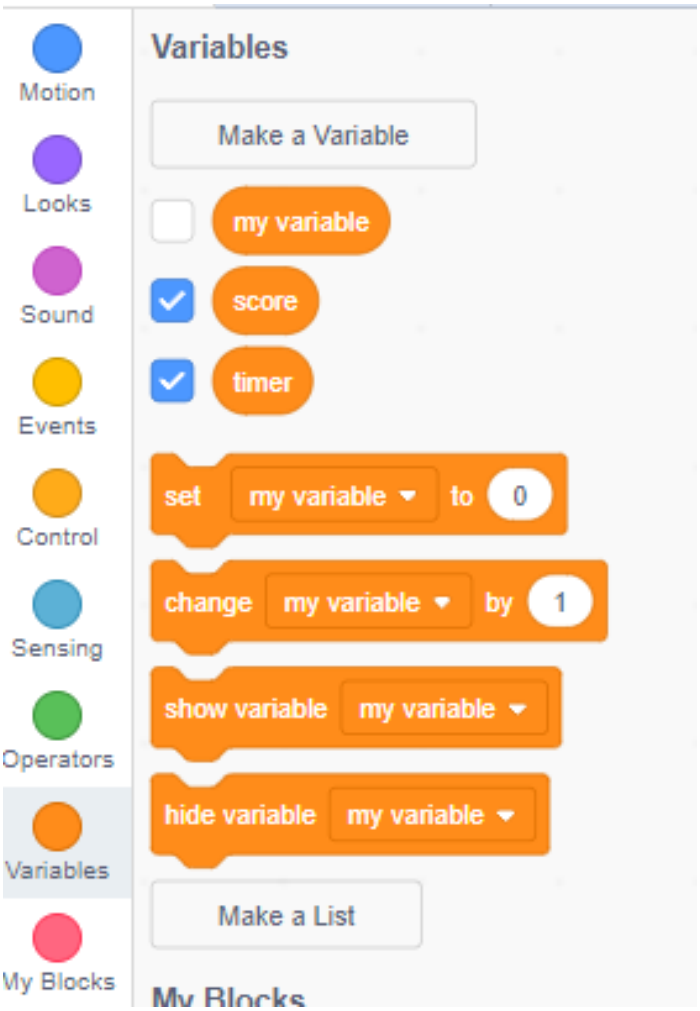
We used the "start sound" block after this code to create a sound queue for clicking our sprite.



Once we were done this, we introduced the most important concept we will learn in this coding course, variables.

Variables are used in every aspect of coding, and for this lesson, we learned how to create, identify, and control variables.

To do this, we went to the variables section of blocks, selected "**Make a variable**", and then entered the name of our variable.

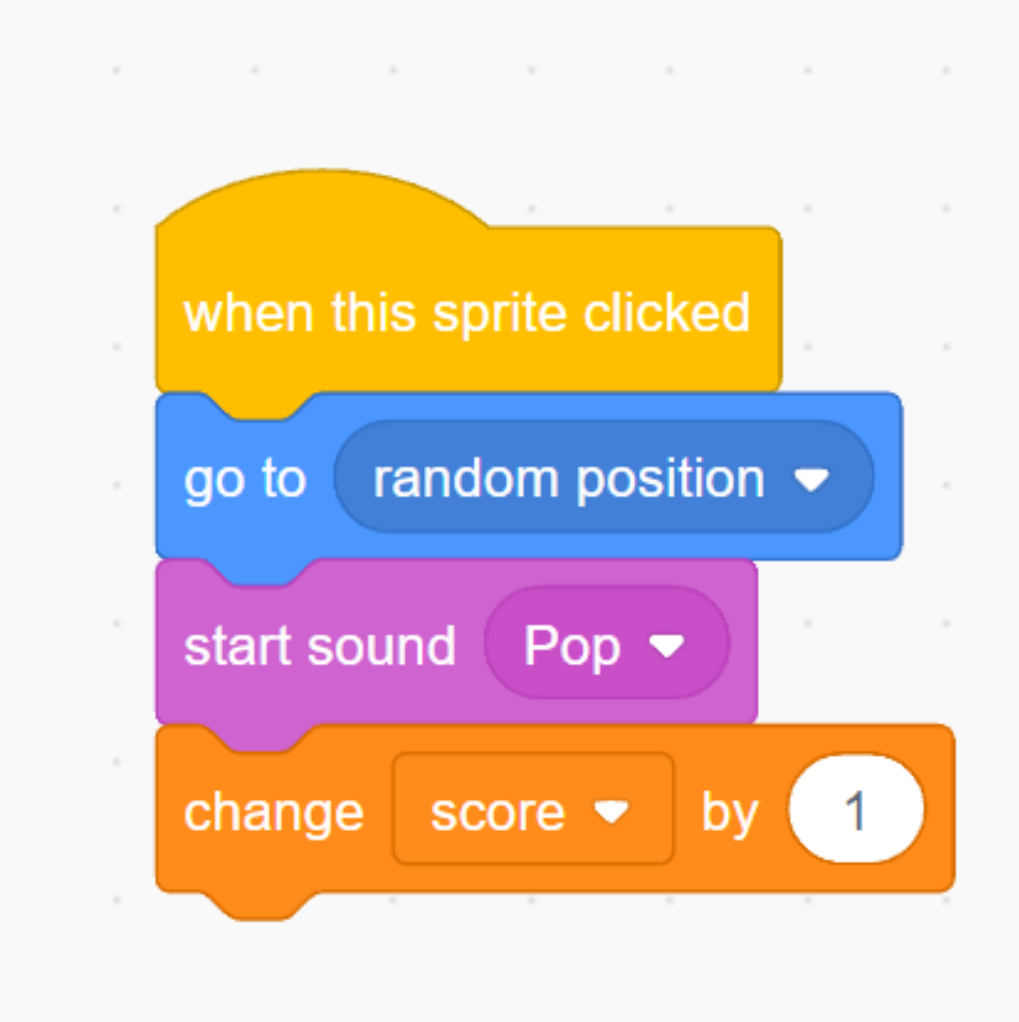


Variables allow us to keep track of different parts of the game, and also to control our game.

For this project, we created two variables: **score**, and **timer**.

First, we used **score** to keep track of how many times our sprite was clicked.

To do this, we selected the "change _____ by 1" block, selected the dropdown box, and chose score.



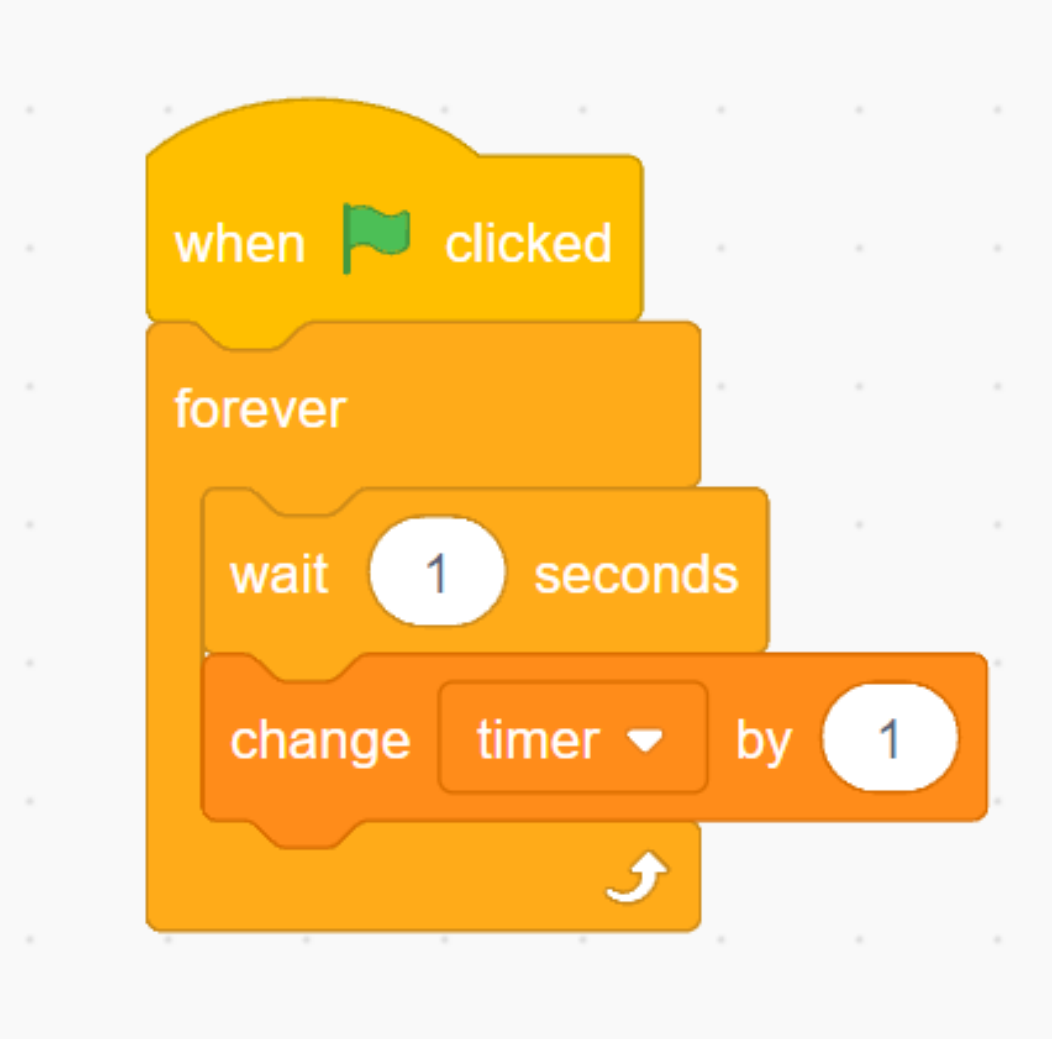
Now we can keep score of our clicks, but how do we make it a game?

To do this, we used our **timer** variable.

We made it so that when the flag is clicked, starting the game, we have a timer that is constantly going up.

To do this, we used a **forever loop**, which is a block that repeats whatever is inside it forever.

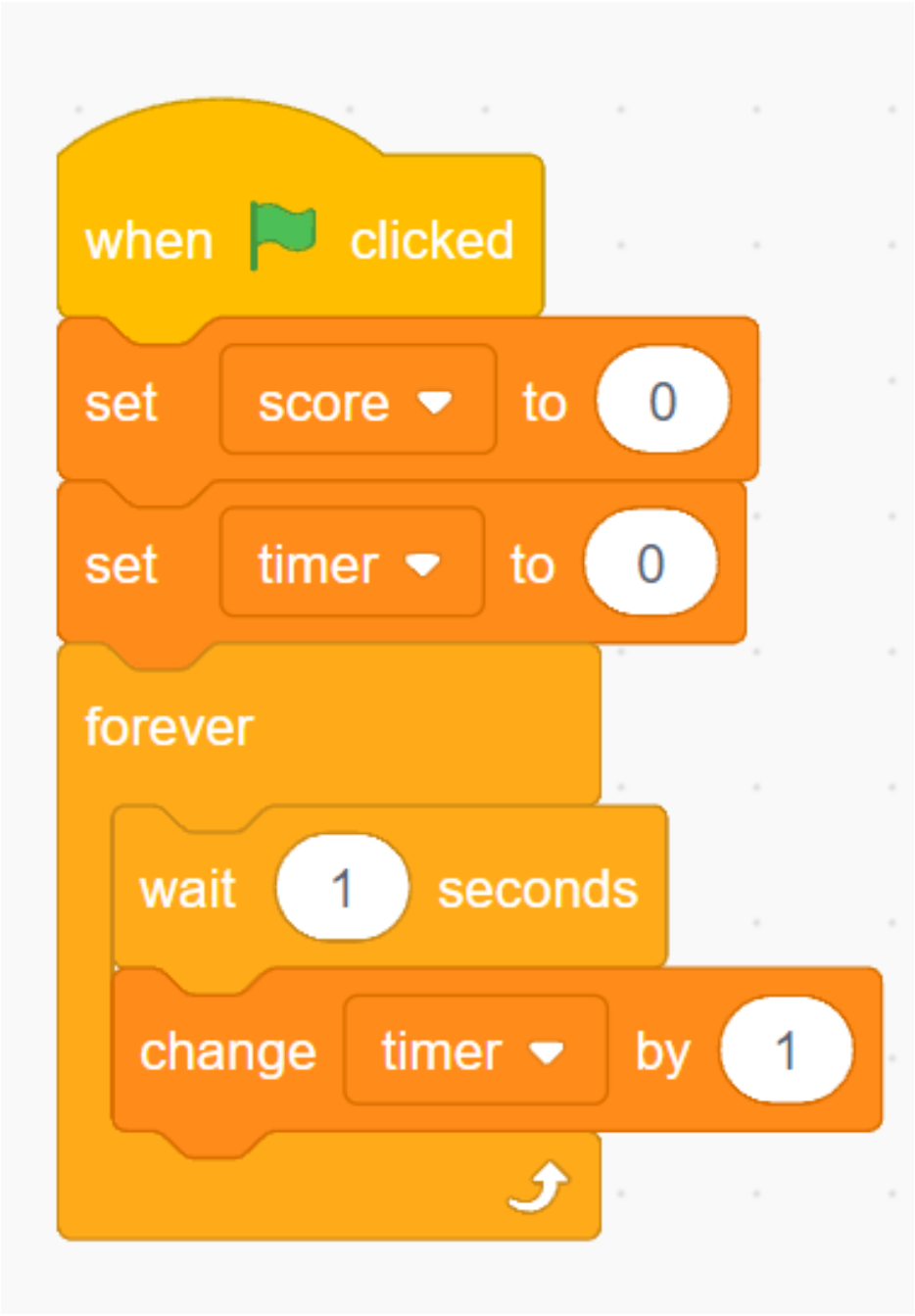
Inside, we placed a "wait 1 second" block, and then a "change _____ by 1" block, selecting timer in the blank.



This makes it so that forever, we wait one second, increase timer by one, and keep repeating. This lets us keep track of how long its been since we started the game.

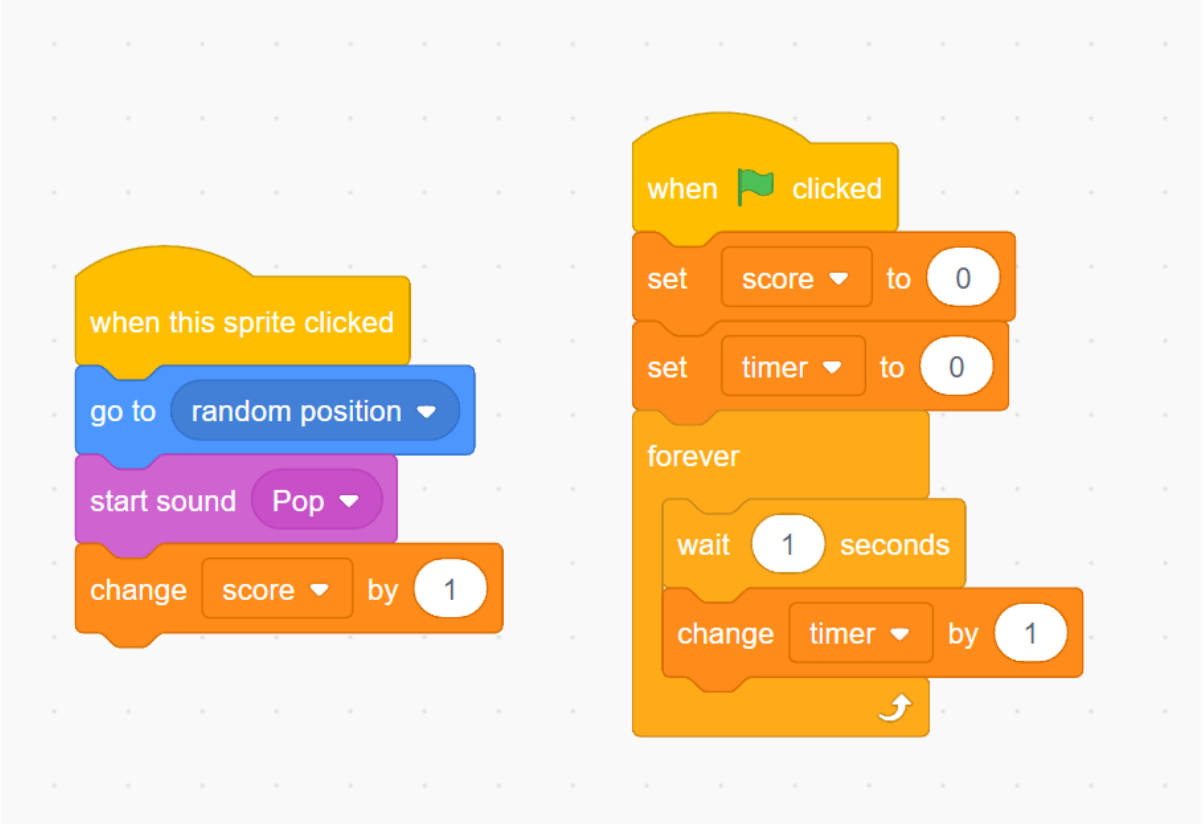
Then, to reset the timer and score when we start the game, we made it so that every time we start the game, before we start adding up our timer, we reset score and time to 0.

To do this, we used the "set _____ to" block, and filled it in so that we set our score and timer to 0.



At the end of the project, we had a game where we can click on a sprite, and see how many we can get in a certain amount of time.

The final code looks like this:



Link to project: https://scratch.mit.edu/projects/1068608619

To see code, follow the link above and select "See inside"

Save your project and test the code before continuing.

Try changing a value or sprite to make the project your own.

Continue at your own pace.

Lesson 2

Smiling Creek: Coding & Design with Scratch - Lesson 2

Lesson overview

Complete this lesson at your own pace.

Find the complete code and project link at the end of this lesson.

Follow the steps below to build the project.

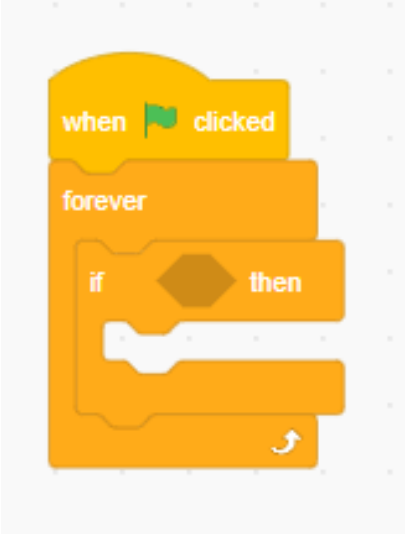
This lesson, we learned how to use multiple sprites to make a simple cat and mouse game. Also, we expanded our general coding knowledge, as well as learned how to use the sound menu in Scratch.

NOTE: In this review, we talk about both a computer mouse, as in, what you use to click and scroll, and also a mouse sprite in our game. We've tried to be as clear as possible which is which, but we are sorry in advance for any confusion.

To begin, we created two new sprites: the cat and mouse sprite.

We first had to program the mouse to be able to move towards where the player clicks.

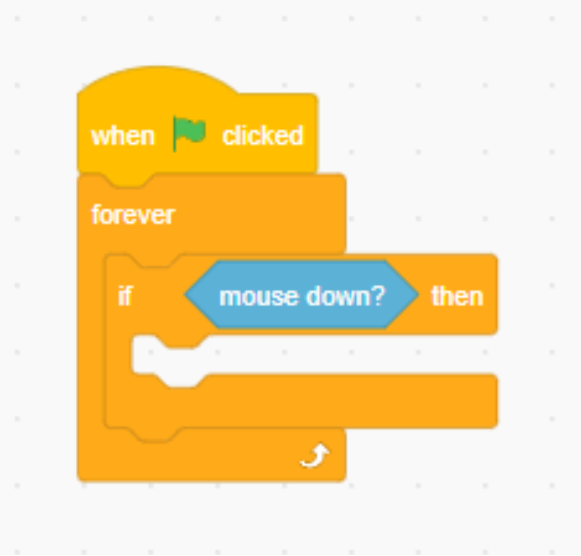
To do this, we used a forever loop in combination with an if loop to notice when the mouse is pressed.



We used a new type of block inside of the if loop, one the **light blue** blocks, called a **sensing** block.

This specific **sensing** block tells us when the mouse is pressed, something we need to know for our code.

Keep in mind that **light blue** blocks are **sensing** blocks, and **dark blue** blocks are used for **motion**.



(This code tells the mouse sprite that forever, it's going to keep checking if the computer mouse is pressed down, and then if it is down, then the mouse sprite will do something.)

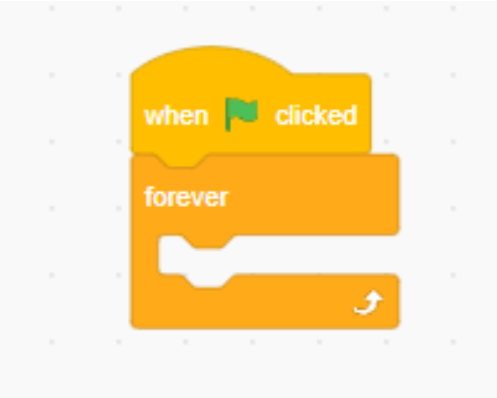
For us, we want this something to be the mouse moving towards where we clicked.

To do this, we use the point towards block and the glide 1 secs block to instruct our mouse to do these motions.

We also make sure that we are pointing towards the 'mouse-pointer', and that we are gliding towards 'mouse-pointer'.

Next, we want to allow the cat to chase the mouse.

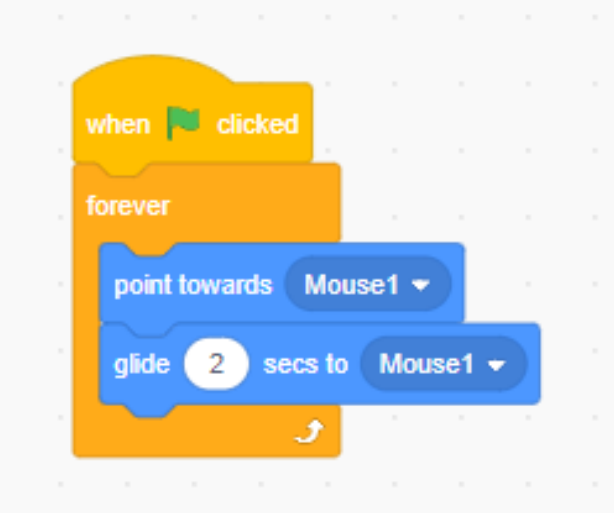
To do this, all we need is another forever loop underneath a when flag clicked block.



Inside of this, we want to keep telling the cat to glide for 1 second towards the mouse. For this, we can use the glide 1 secs block.

We want to select the '1' and change it to '2', so that our cat is slightly slower than our mouse to keep the game fair.

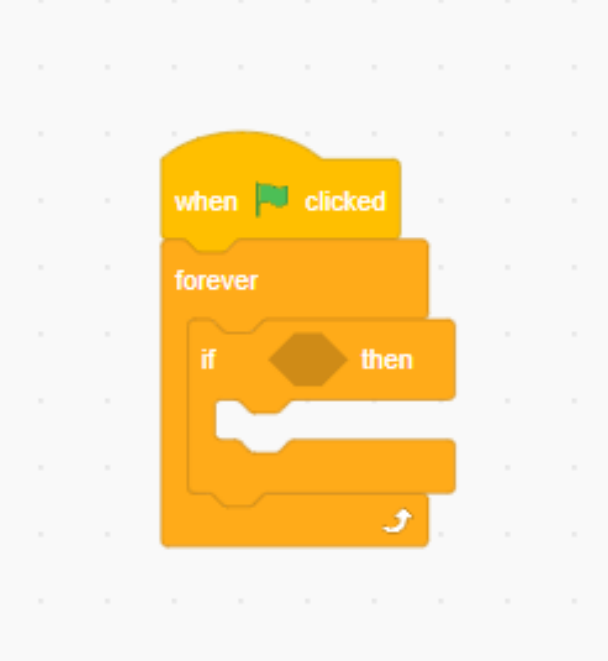
We also want to place the point towards block above this, so that our cat is oriented towards the mouse as it chases.



Now we have a cat that chases our mouse, but what happens when the mouse gets caught?

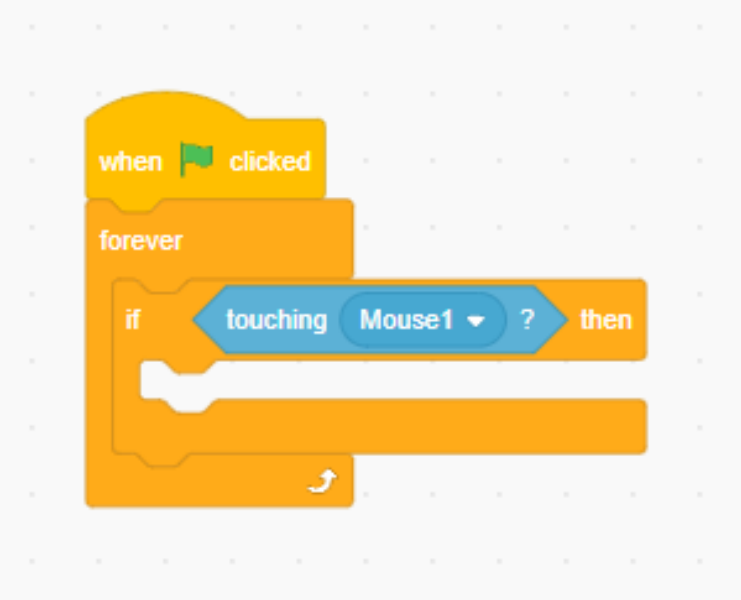
To create a reaction from the cat, we again need to use **sensing** blocks.

We will first use another when flag clicked, forever loop, and if loop to check if something is happening. In this case, something will be contact with the mouse.



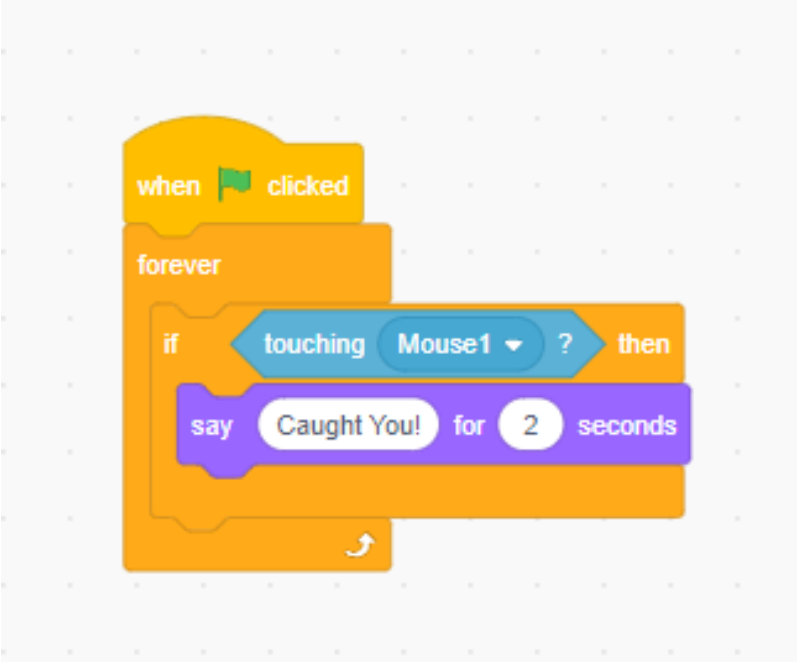
To check if the cat is in contact with the mouse, we need to go on the **sensing** tab on the left, and pull out the touching ___ block.

Then, drag this into the slot after if.



(Translated to English: forever, we'll keep checking if the cat is touching the mouse, and if we are, then we will do something)

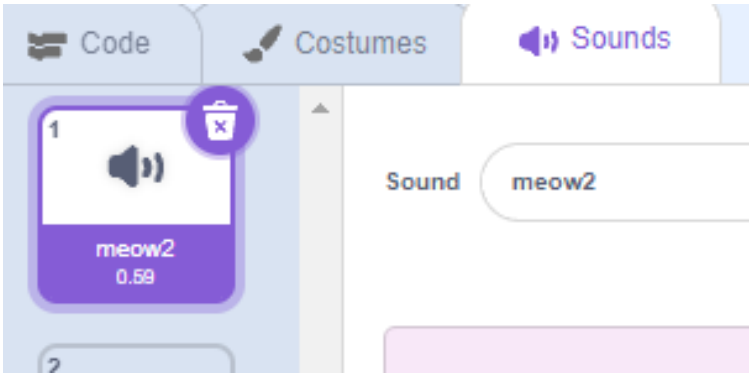
Now that we can check if the animals are touching, we want our cat to say something. To do this, we go to the looks tab on the left, and drag in the "Say 'Hello' for 2 seconds" block. Select 'Hello', and change it to, "Caught You!", or whatever other witty message you'd like the cat to say when it catches the mouse.



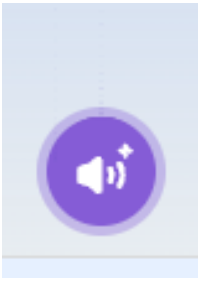
The last key learning from this lesson was how to use the "**Sounds**" tab in Scratch, this time in more depth.
Right now, when we drag in a "**Play Sound**" block, there's only one or two sounds available to play.

However, Scratch has thousands of different sound effects to choose from, and they can be accessed by:

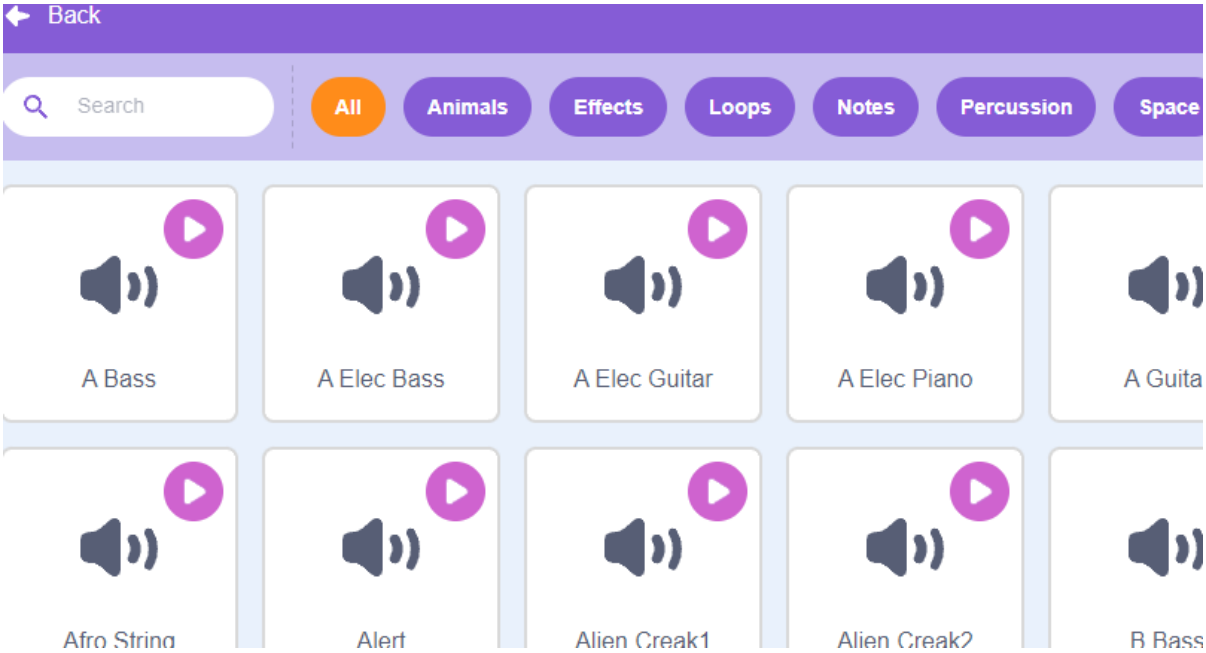
Click on the sounds window in the top left.



Click the circle on the bottom, reading "Choose a Sound"



Search whatever sound you'd like, and select the one you choose to be able to use it in the game.

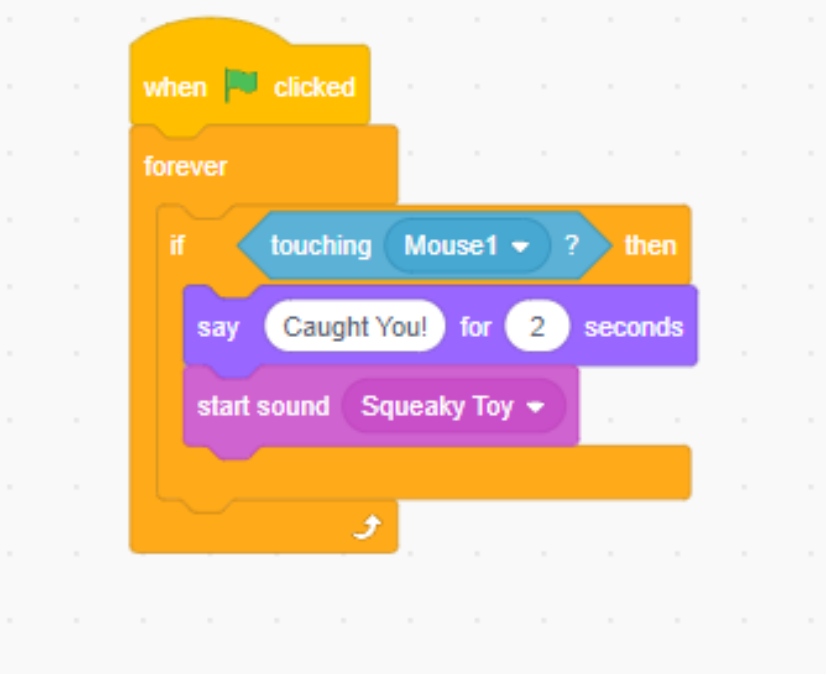


Now that we know how to do this, we select a sound to represent the cat catching the mouse.

For mine, I chose a squeak noise.

To add this to our game, just enter the **sound** blocks, (not to be confused with the sound tab we just used), and drag in the "Start Sound" block above the Say block.

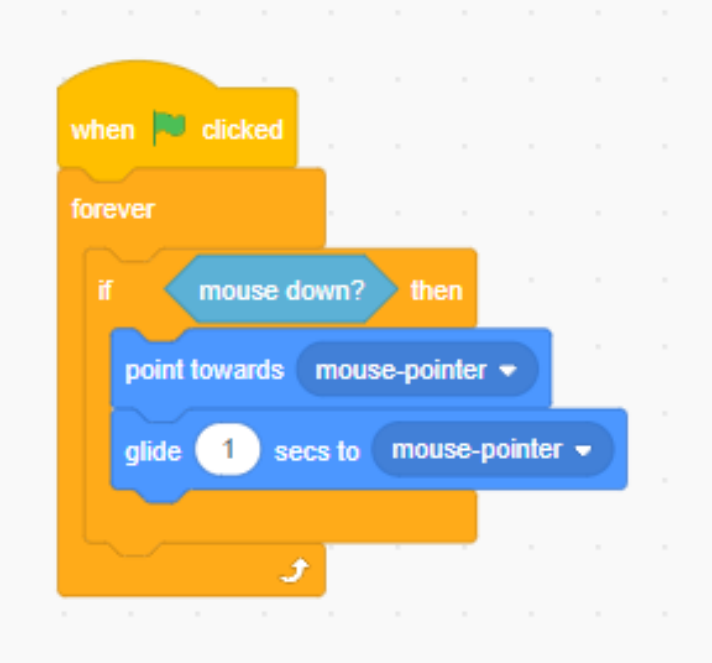
Then, select the box in the block and change it to whichever sound you chose.



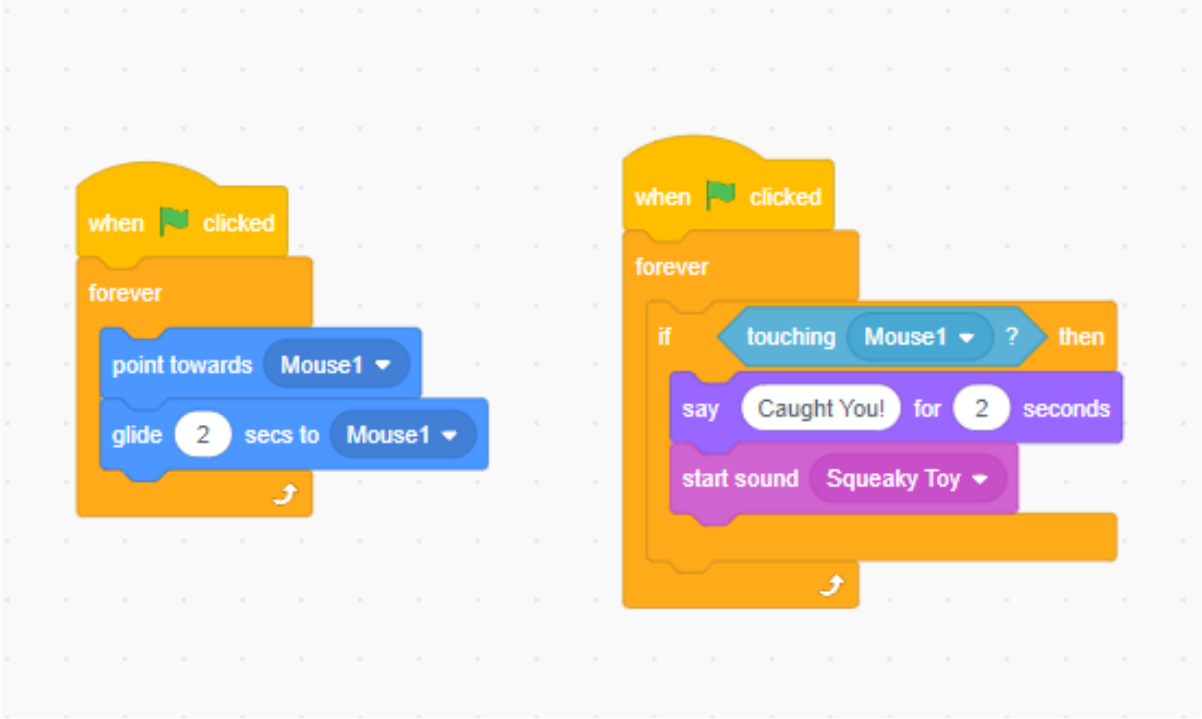
Now, our code is finished!

All complete code:

For the Mouse:



For the Cat:



Link to project: https://scratch.mit.edu/projects/1072460697

Save your project and test the code before continuing.

Continue at your own pace.

Lesson 3

Smiling Creek: Coding & Design with Scratch - Lesson 3

Lesson overview

Practise using variables to control a sprite.

Find the complete code and project link at the end of this lesson.

Follow the steps below to build the project.

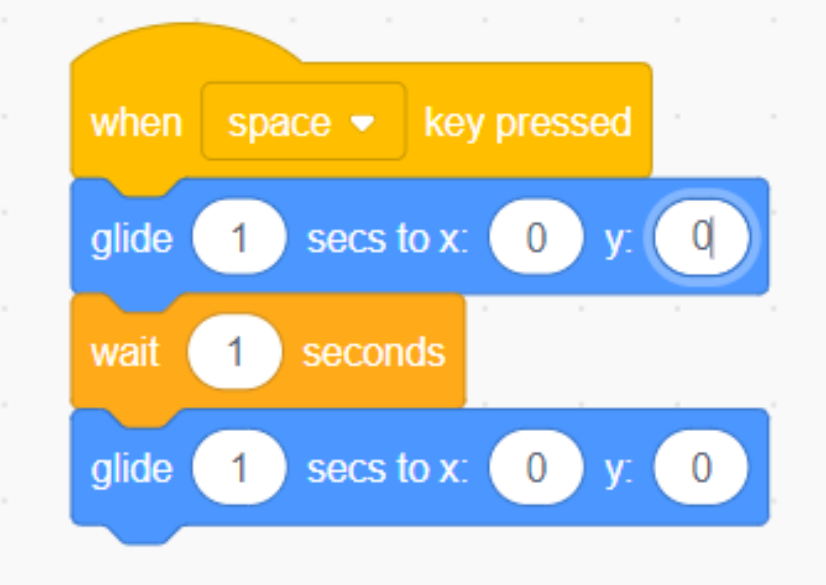
The main idea of this lesson was getting more comfortable with variables.

As previously mentioned, variables are incredibly important to programming.

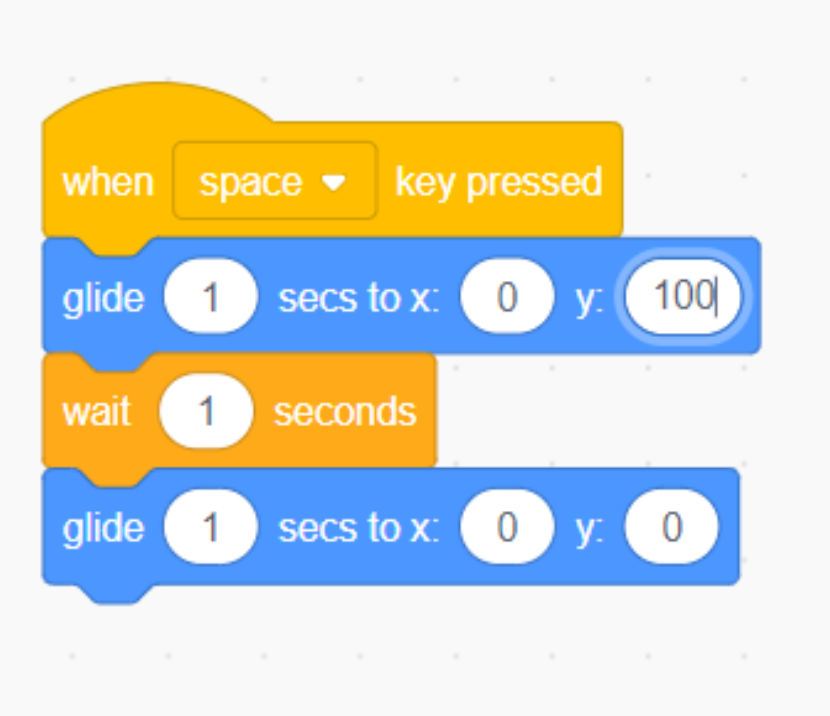
Today's project was less focused on exploring new ideas, and more on improving our understanding of current ones. We explored creating a jumping sprite, but more specifically, how we can control different aspects of this jump using **variables**. Putting variables in place of values is essential to programming in all fields, and if students ever begin to learn to program in other languages, they'll be surprised to learn that these other languages are made up of almost exclusively variables.

To begin, we created code that caused our sprite to glide up, pause for a second, and then glide back down to the starting position.

To do this, we used two **motion** "glide ____ secs to" blocks, and placed a "wait ___ seconds" block in between.



Next, we altered the values inside of these blocks to create the jumping effect.



(Reminder that **x** represents **horizontal** (left and right) movement, and **y** represents **vertical** (up and down) movement).

This jump works, and tampering with the values that control the amount of seconds will lead to a faster or slower jump and pause.

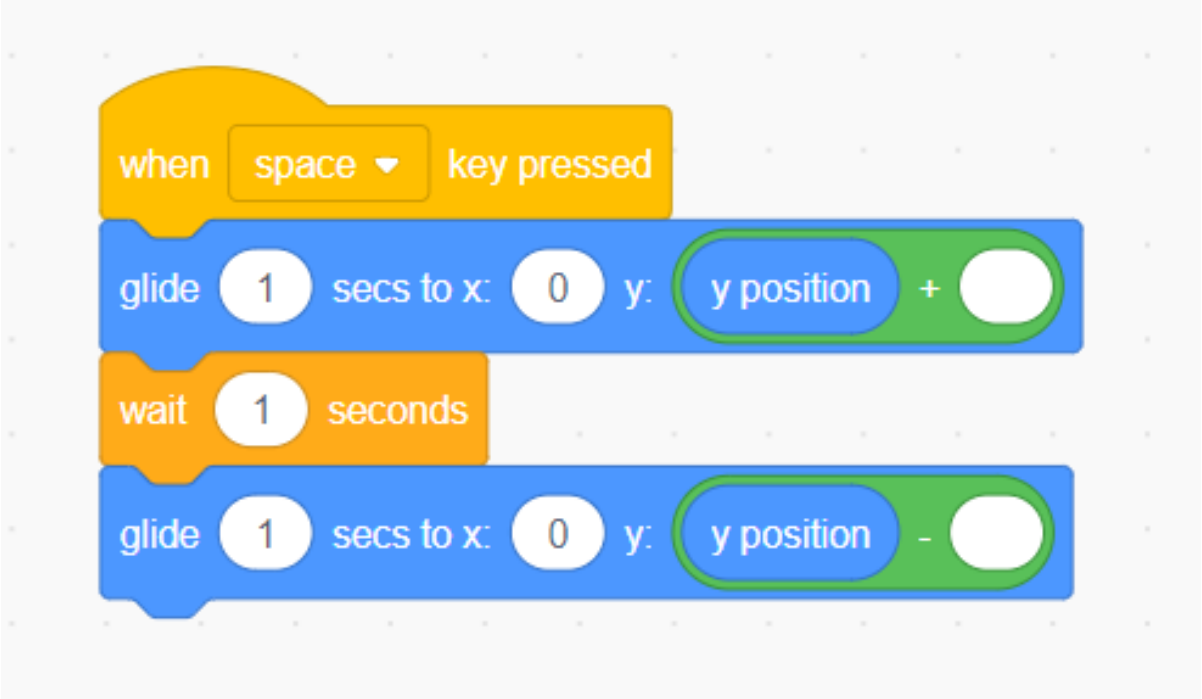
Unfortunately, this code has one disadvantage: it only lets the character jump in the middle of the screen. Since the location that the player is gliding to is a set location, we can't place the player anywhere else without ruining the jump.

To fix this, we need to use the player's y (vertical) position, and then let the player glide to the players current vertical position plus some jump-height variable that we choose, wait for a second, and then glide to the player's position minus this same jump-height variable.

We also need to use the player's current x (horizontal) position, to make sure that the jump doesn't shift left and right.

To achieve this, we'll first need to drag in two **motion** "y position" blocks. Next, a "(___ + ___)" green **operator** block, and then also a "(___ - ___)" green **operator** block.

Place the "y position" block in the first slot of the operator block. Your code should look like this:

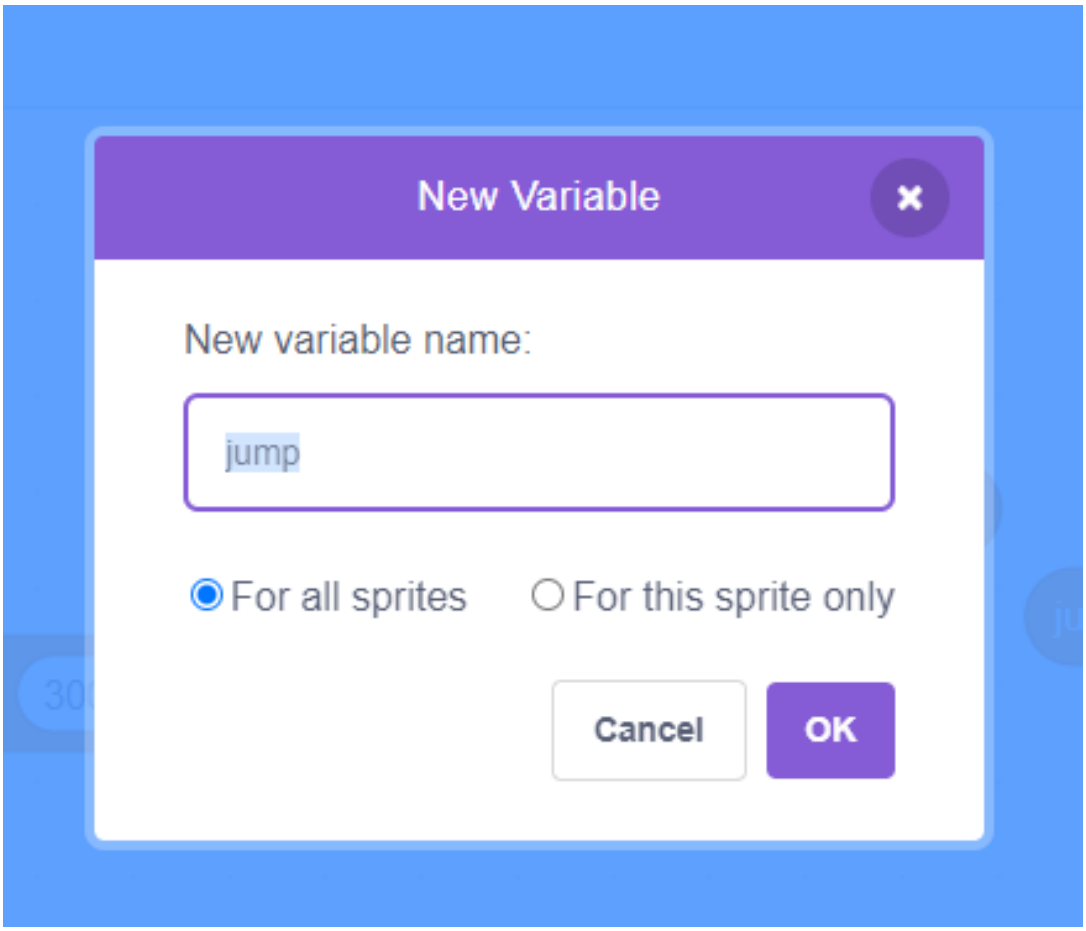


Next, to maintain the character's horizontal position when we jump, drag in two "x position" blocks, and set them in the "x:" slot on the motion blocks.

Finally, we learn how to use variables to set a value.

For recap, you create a variable by going over to the orange variables tab, and selecting "Make a variable".

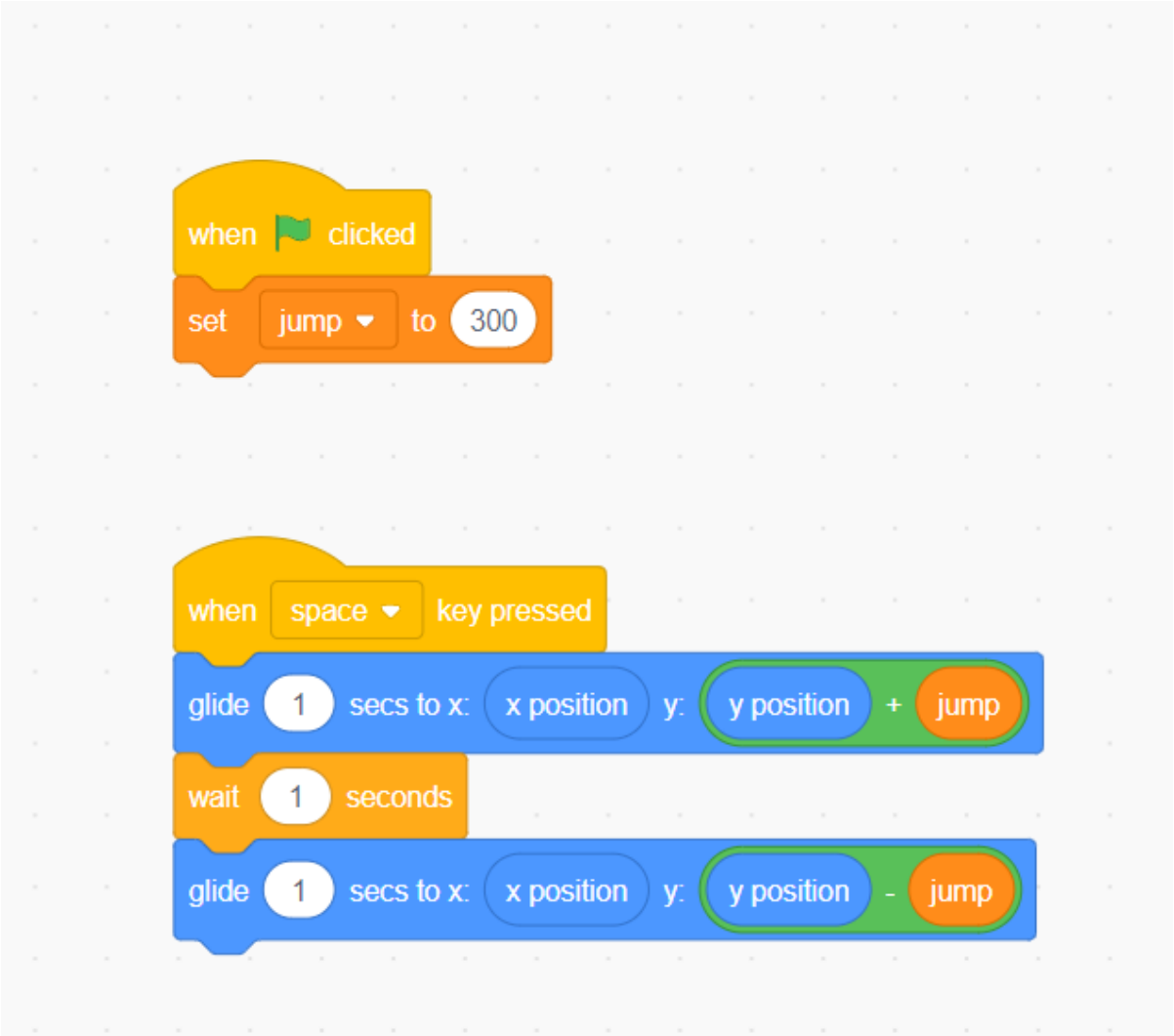
Then, simply enter the name of your variable and it will be created.



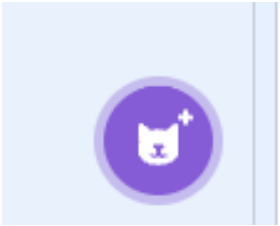
Once this variable is created, stay on the variables tab, and drag in the variable itself into the second slot of the green operator block.

The final step into creating a dynamic jumping character is to **set the jump variable's value**.

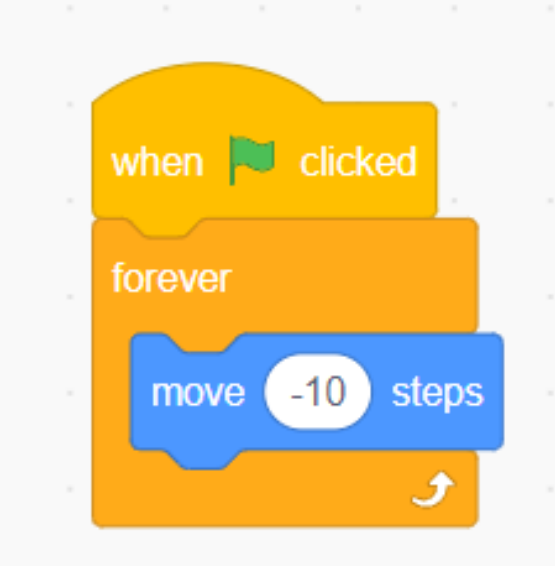
To do this, we simply drag in a when Flag clicked block from the events tab, and then a "set _____ to __" block underneath. Then, set the first blank to our jump variable, and then the second blank to whatever jump height value you prefer. Remember, to change the speed of the jump, you can change the number before "secs" on all 3 of the blocks either higher or lower.



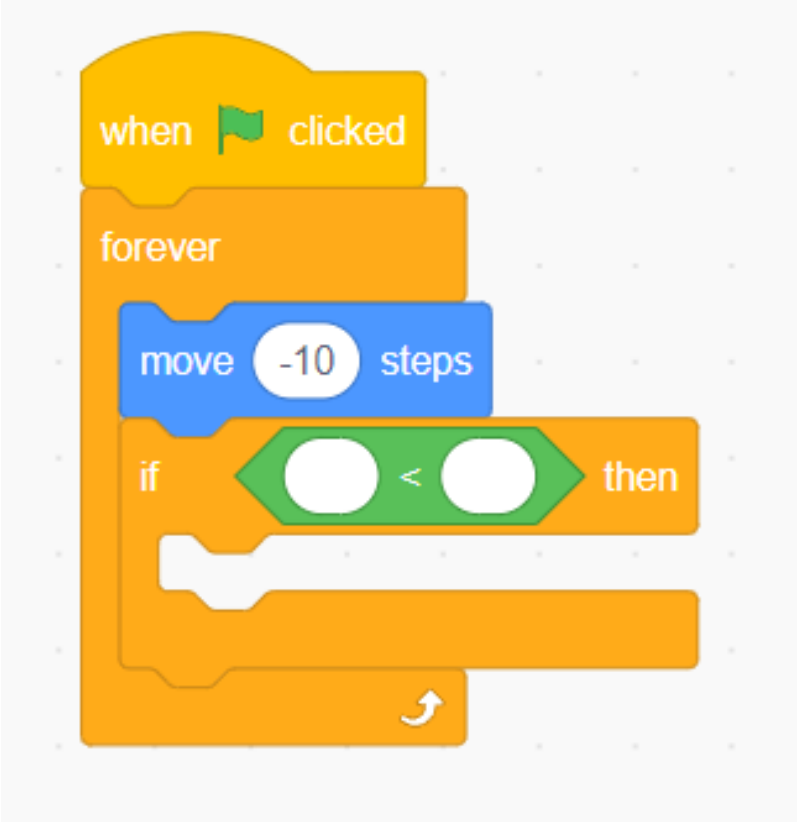
Finally, to give the character to have an obstacle to jump over, we created a new sprite by clicking the button in the bottom right.



Open the coding space of this sprite, and then drag in a "when flag clicked", and then a "forever" block with a "move __ steps" block underneath, and set the steps value to be -10. Changing the magnitude of this number will change the speed of the sprite.



Then, drag in an if block, and inside of the first box, drag in the green **operator** block with the lesser than symbol (students may know it as the mouth eating to the right).

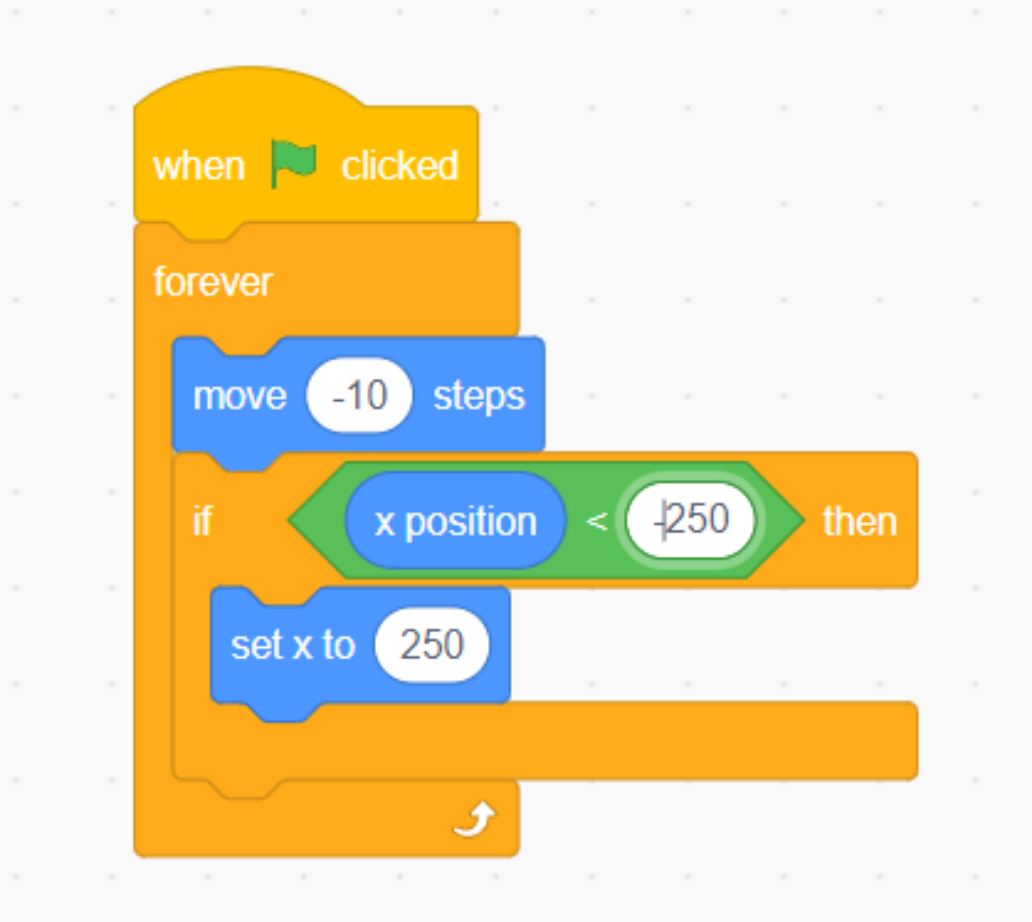


Next, drag in the motion "x position" block we worked with earlier. This will give us the horizontal position of this obstacle sprite. We're going to check to see if this position is a certain amount to the

left, at the end of the screen (around x = -250), and then if it is, we'll move it to about x = +250, the right-most part of the screen. This will ensure that it keeps looping through the scene.

Place the "x position" block into the first slot, and then type "-250" into the second.

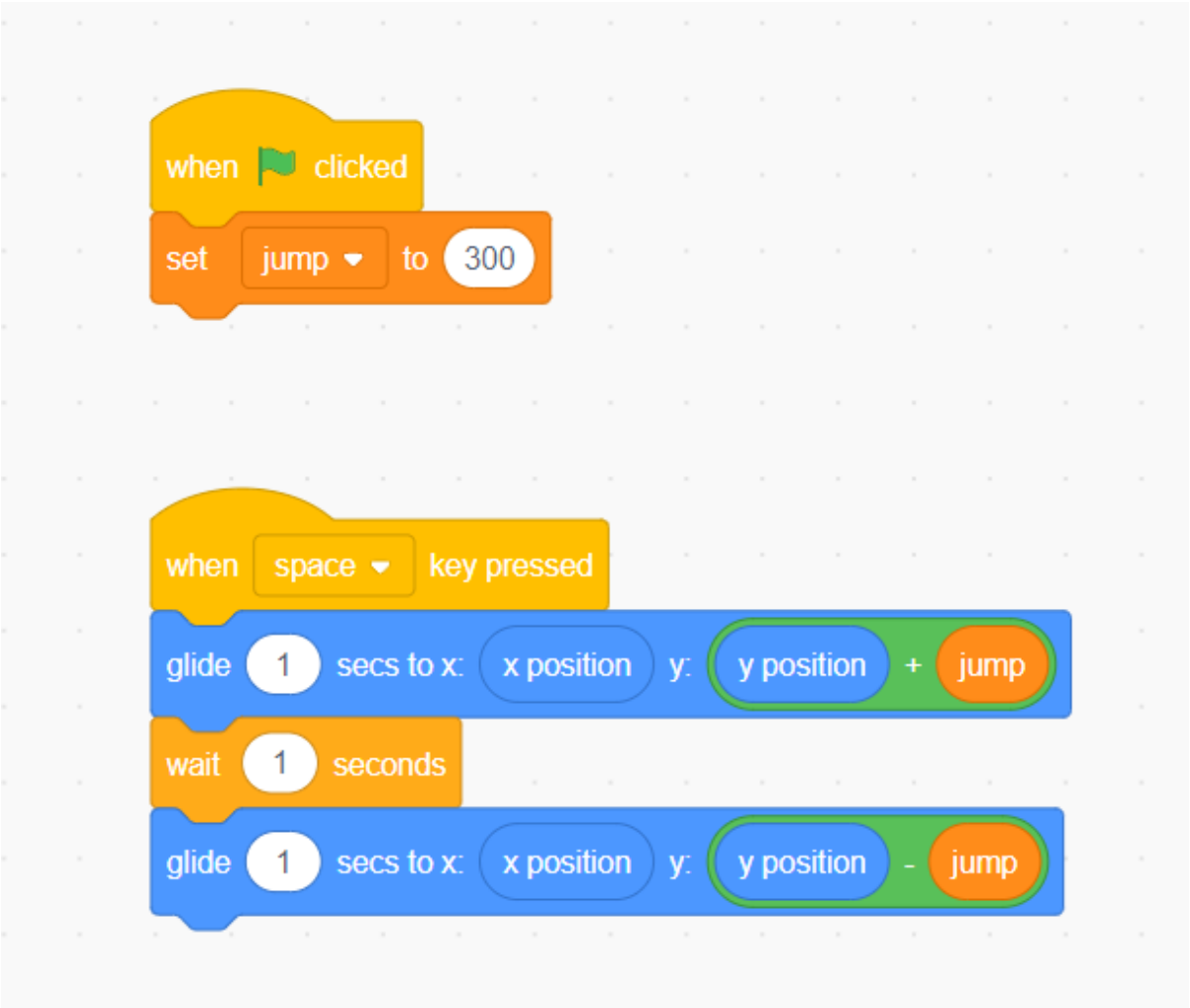
Underneath, drag in the "set x to ___" block, and then set that value to 250.



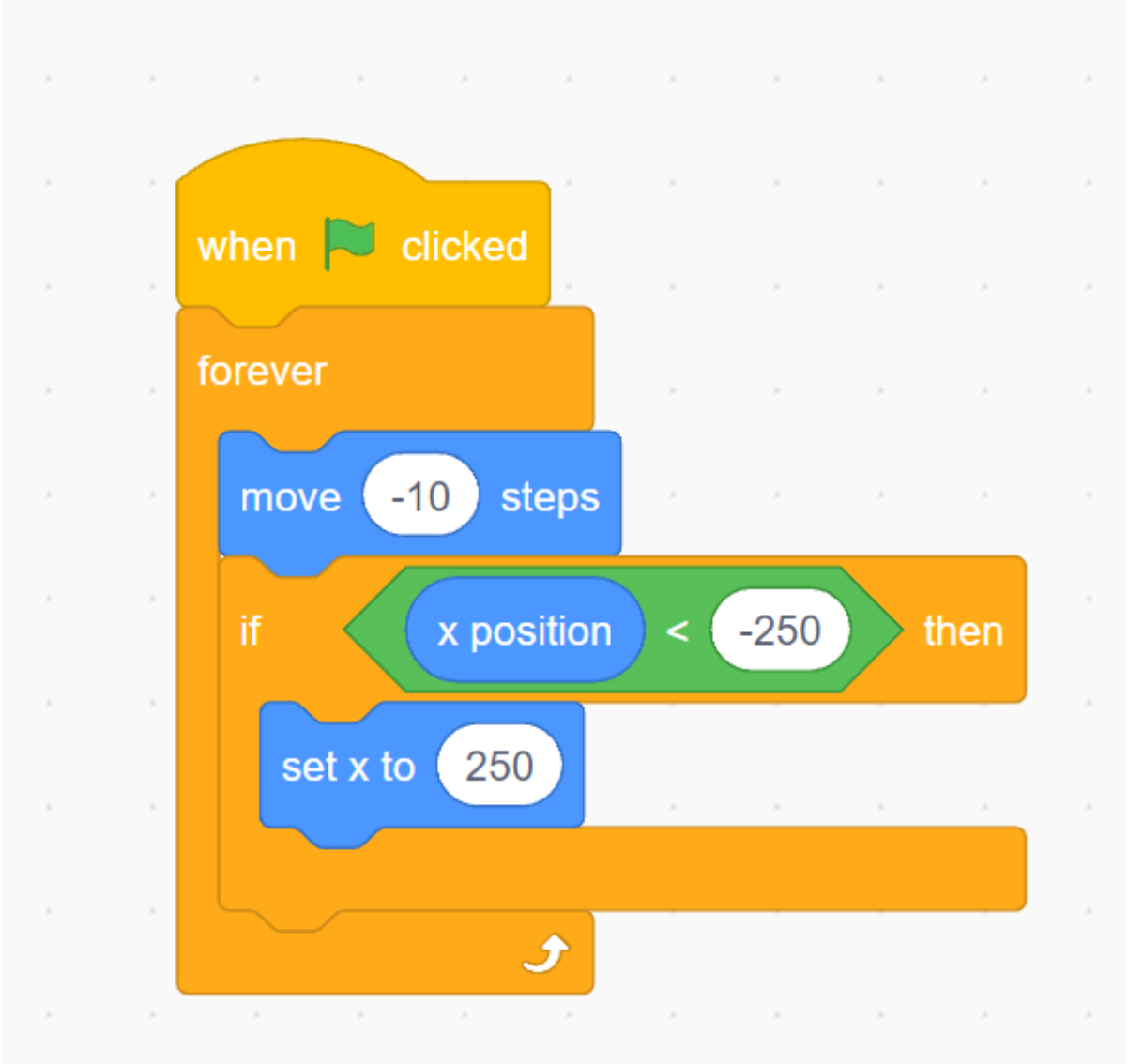
Once this is completed, we have an obstacle that will go back and forth across the screen.

That's it for the project! Here is all of the code:

For the first sprite (jumping sprite):



For the second sprite (Obstacle):



and the project link:

https://scratch.mit.edu/projects/1078127442

As a challenge, students are encouraged to try program a way to check when the two sprites have collided.

To explore the example code, open the project link above and select "See inside".

(HINT - use a block from the "sensing tab".

Save your project and test the code before continuing.

Lesson 4

Smiling Creek: Coding & Design with Scratch - Lesson 4

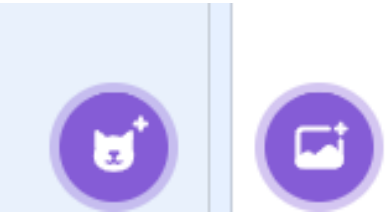
Lesson overview

Find the complete code and project link at the end of this lesson.

The main idea of this lesson was to explore alternative methods of movement, and also to understand and learn 'random' and how to implement it into our code.

We explored these concepts through the movement of our rocket, and also through the respawning of our meteor/planet at a random location on the top of the screen.

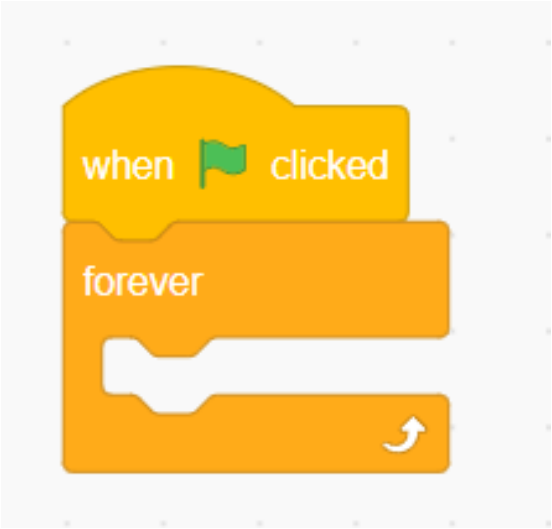
Lets first create a rocket sprite, and a space backdrop. If you've forgotten how, use the buttons below:



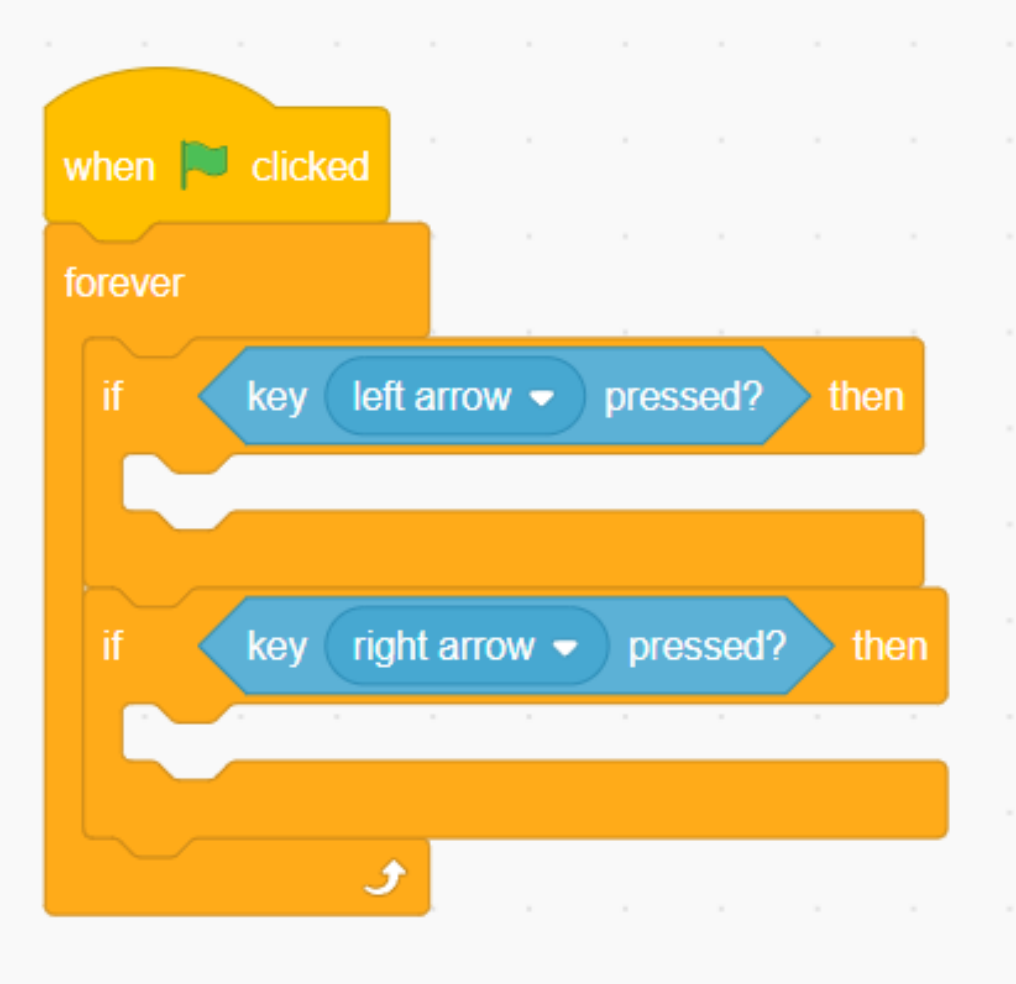
To start, we'll create some code that let our rocket move from side to side.

Normally, what we have done for movement is to use the "when ____ key pressed" block to move our character. However, this means that our character/rocket would move even when the flag isn't clicked, or in other words, it would move when the game hasn't even been started.

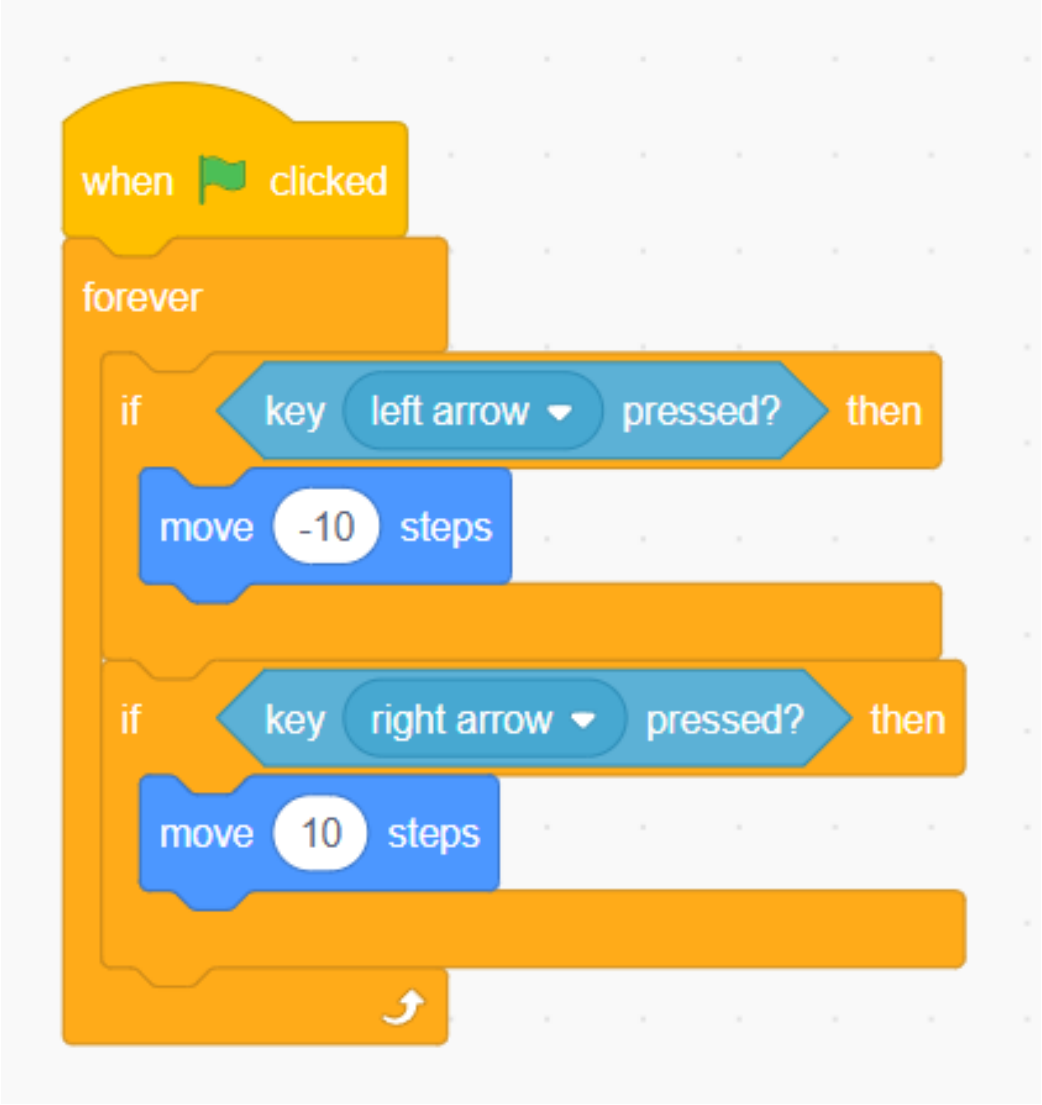
To fix this, we first get a **events** block "when flag clicked" block and place a **control** block forever loop underneath.



Next, we take two **control** tab "if" statements, and drag in the **sensing** block "key _____ pressed", setting the value in the blank space to be the right and left arrow keys.



Finally, we go to the motion tab and drag in two "move 10 steps" blocks into each. Set the one under "left arrow" to be equal to negative 10, allowing us to move to the left.

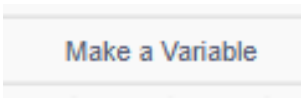


Now that our rocket is completed, its time to work on the meteor.

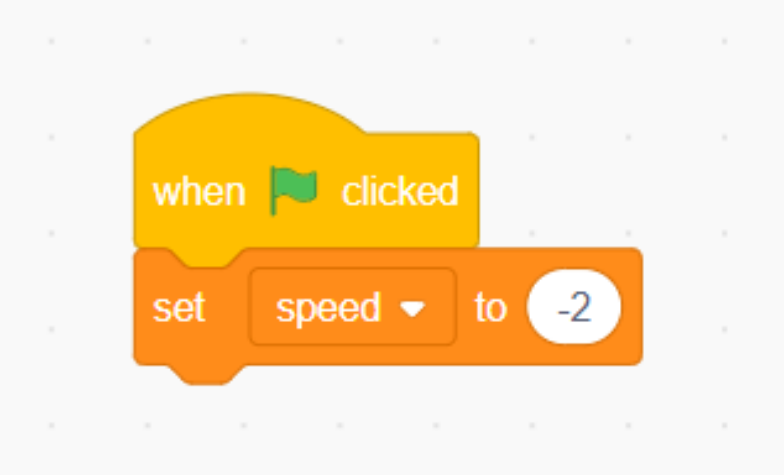
First, create a second sprite, preferably a meteor or planet or any sprite like that.

To begin, we want to create a variable called "speed". We will use this variable to control the speed of our meteor, allowing us to increase the pace of the game as it goes on.

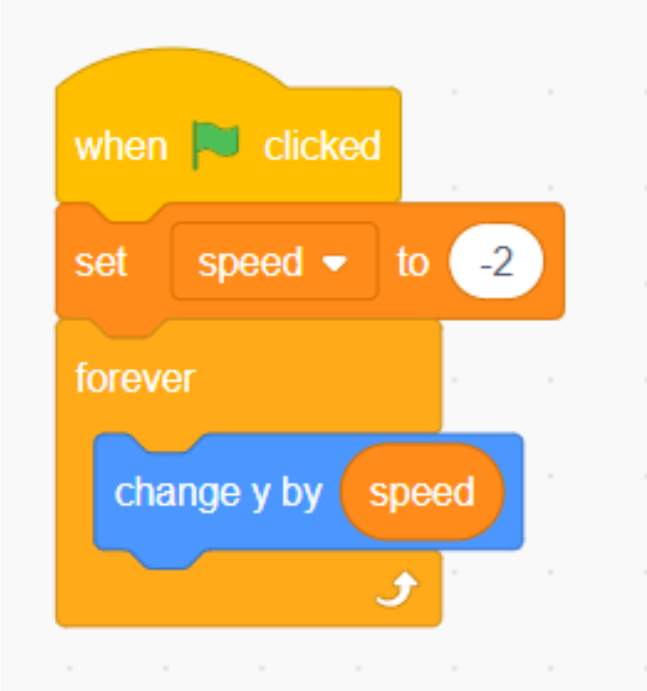
To do this, we go to the variables tab, and select the option "Make a Variable", and then type in speed and click OK.



After that, we'll drag in a "when flag pressed" block from the events tab, and then underneath, we'll drag in a "set _____ to" block from the variables tab. We want to fill out this block so that the variable in the middle slot is our speed variable, and the number is -2. This allows us to set our starting speed.

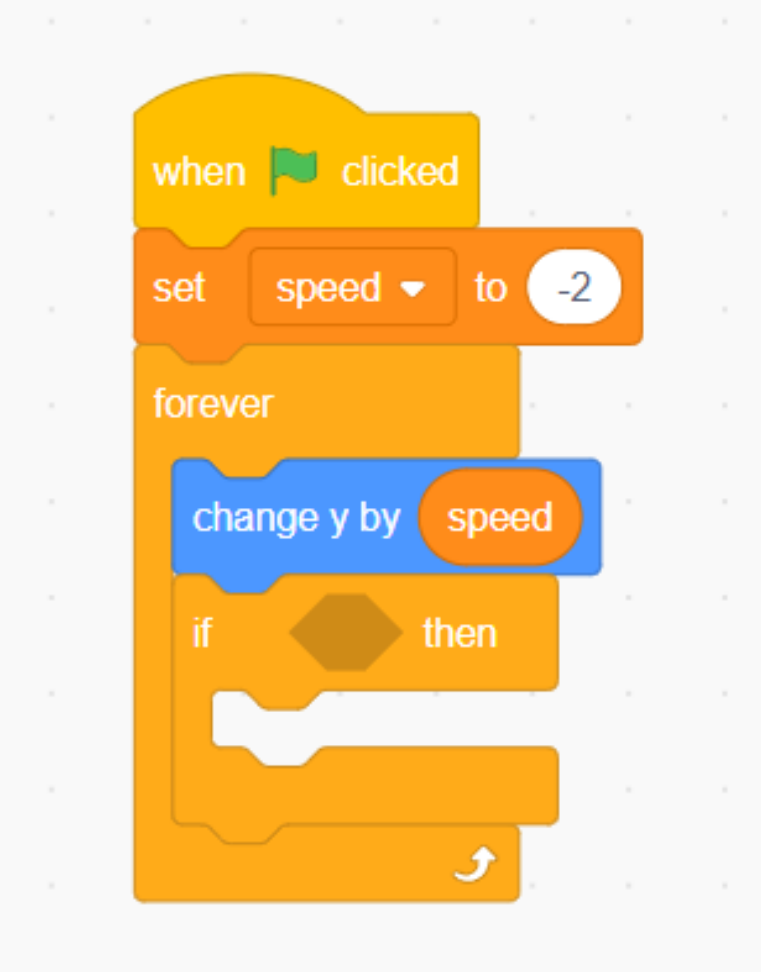


Once this is done, we want to drag in a forever loop again from **control**, and then drag in a motion "change y by ___" block. After this, we'll go over to the **variables** tab, and drag in our "speed" variable into this slot.



Next, to reset the position of this meteor once its reached the bottom of the screen, we're going to check when it's y position, or height, is equal to the bottom of the screen, and then we're going to place it at the top by setting the y position to the top of the screen, and then set the x position (horizontal) to somewhere random.

To do this, we'll need a **control** tab "if" block, and place it underneath the "change y by" block.

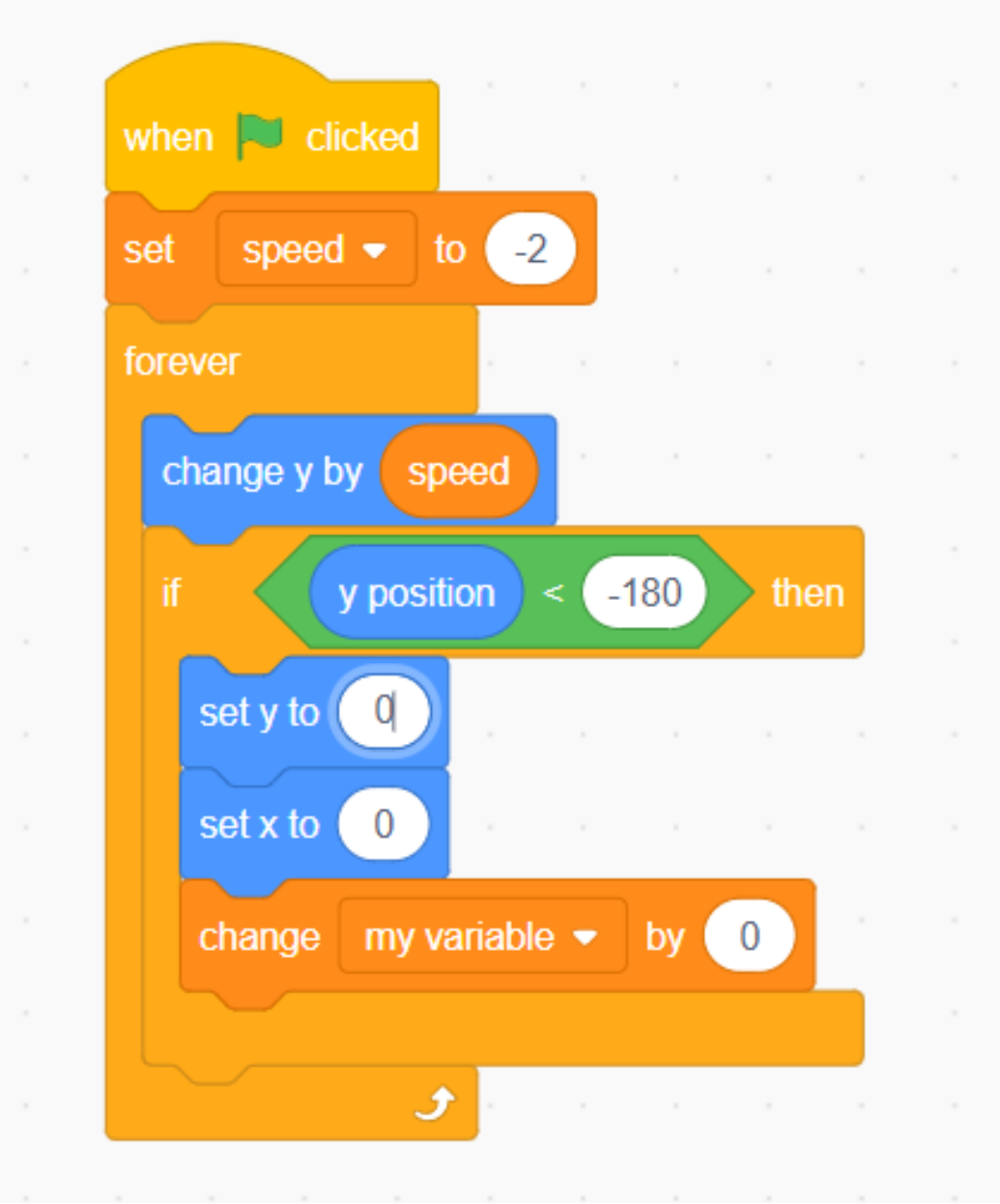


Then, go to the **operators** tab, and pull out a (_<_) block, where the greater number is on the right, or in other words the mouth is eating to the right. Place this block in the "if" block.

After this, go to the **motion** tab, and drag in a "y position" block to the first slot on the operator block. Then, set the number in the second slot to be -180.

Following this, inside the "if" block, drag in a "set y to __" block from the **motion** tab. Underneath this, drag in a "set x to __" block as well.

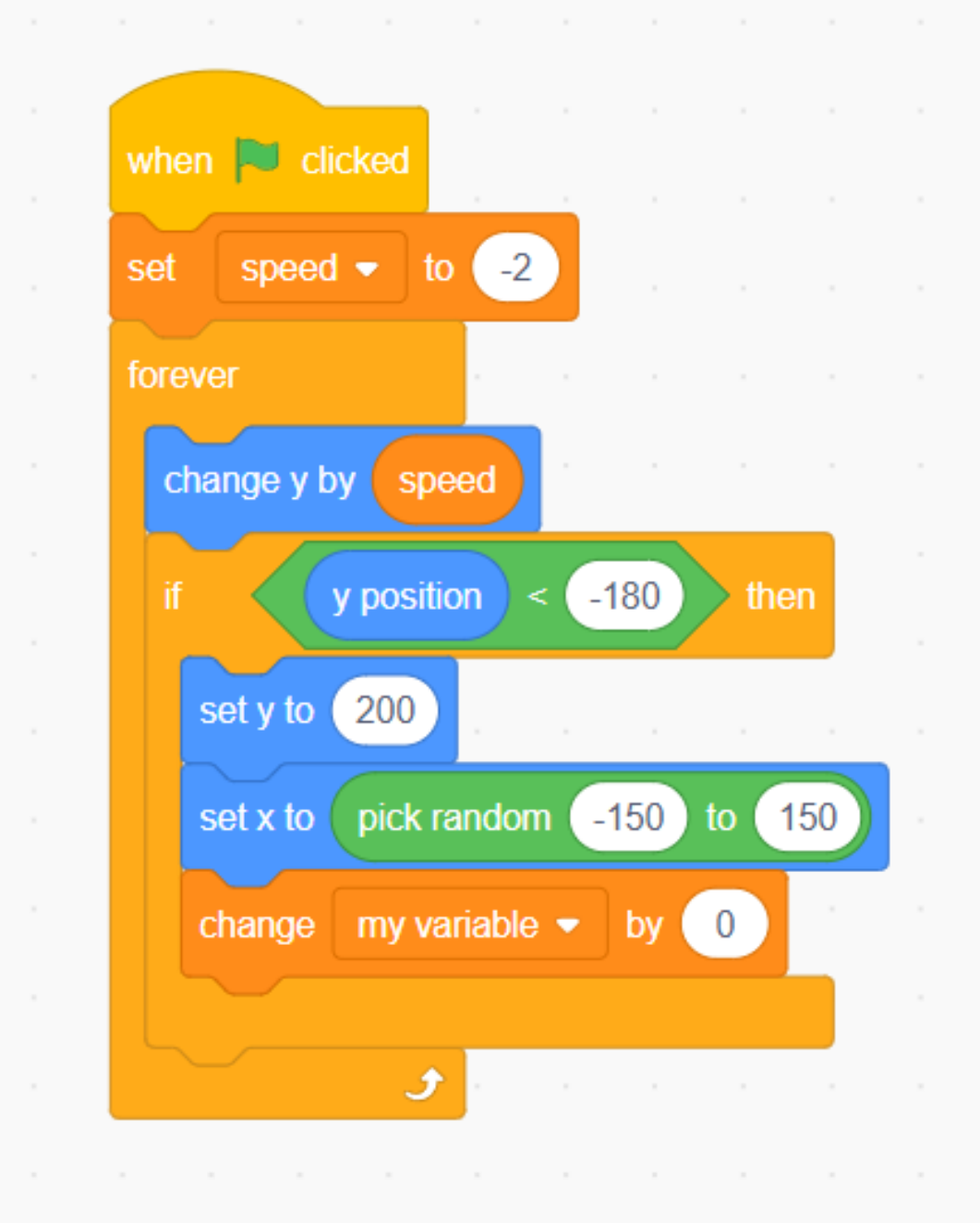
Finally, underneath this all, drag in a "change _____ by __" block from the **variables** tab. Your code should look like this:



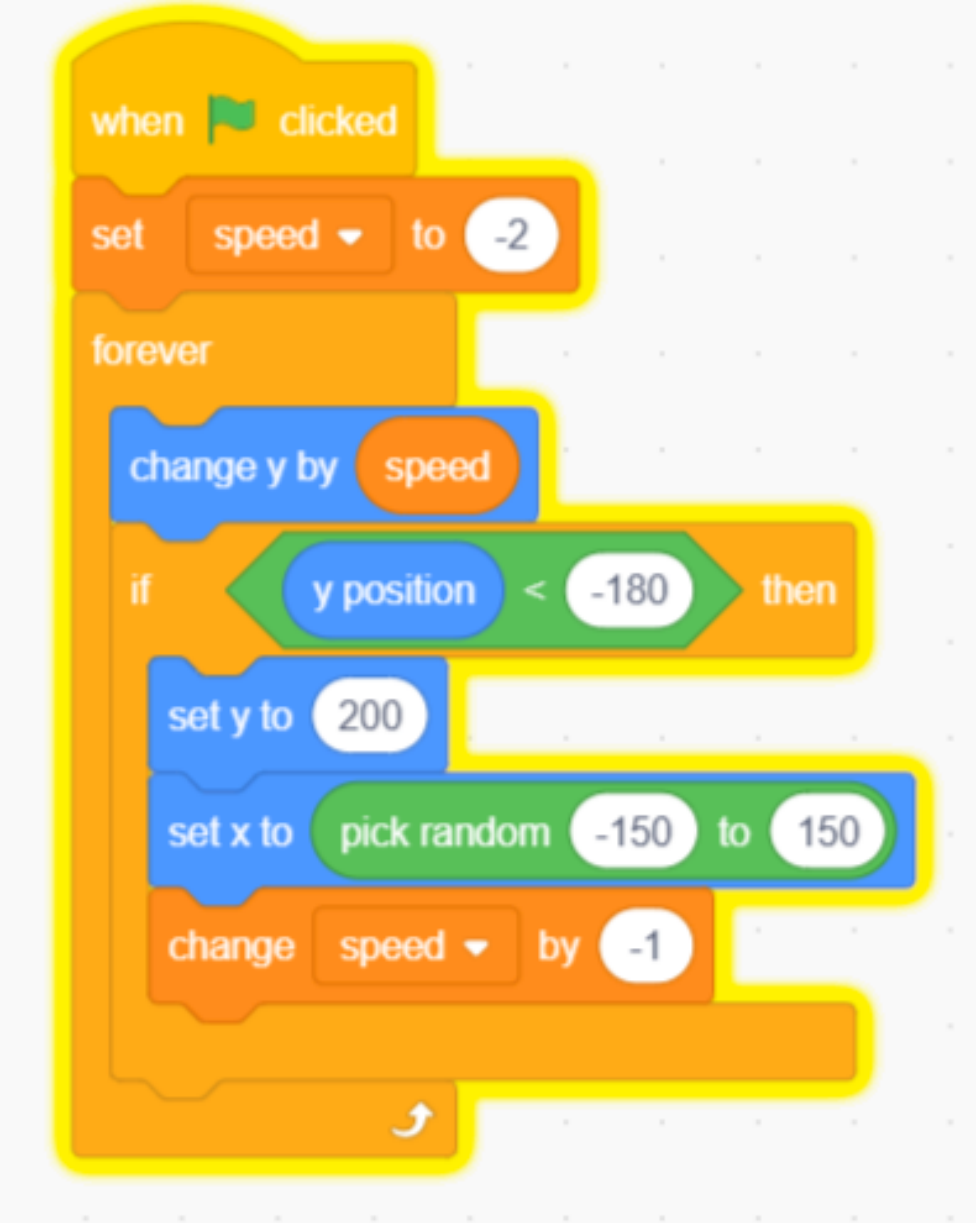
Now, set the number inside the "set y to" block to be 200.

Then, go to the operators tab, and drag in a "pick random ___ to ___" block. This will allows us to change the horizontal position of where the meteor falls on the screen.

Set the values in this block to be "-150" and "150".

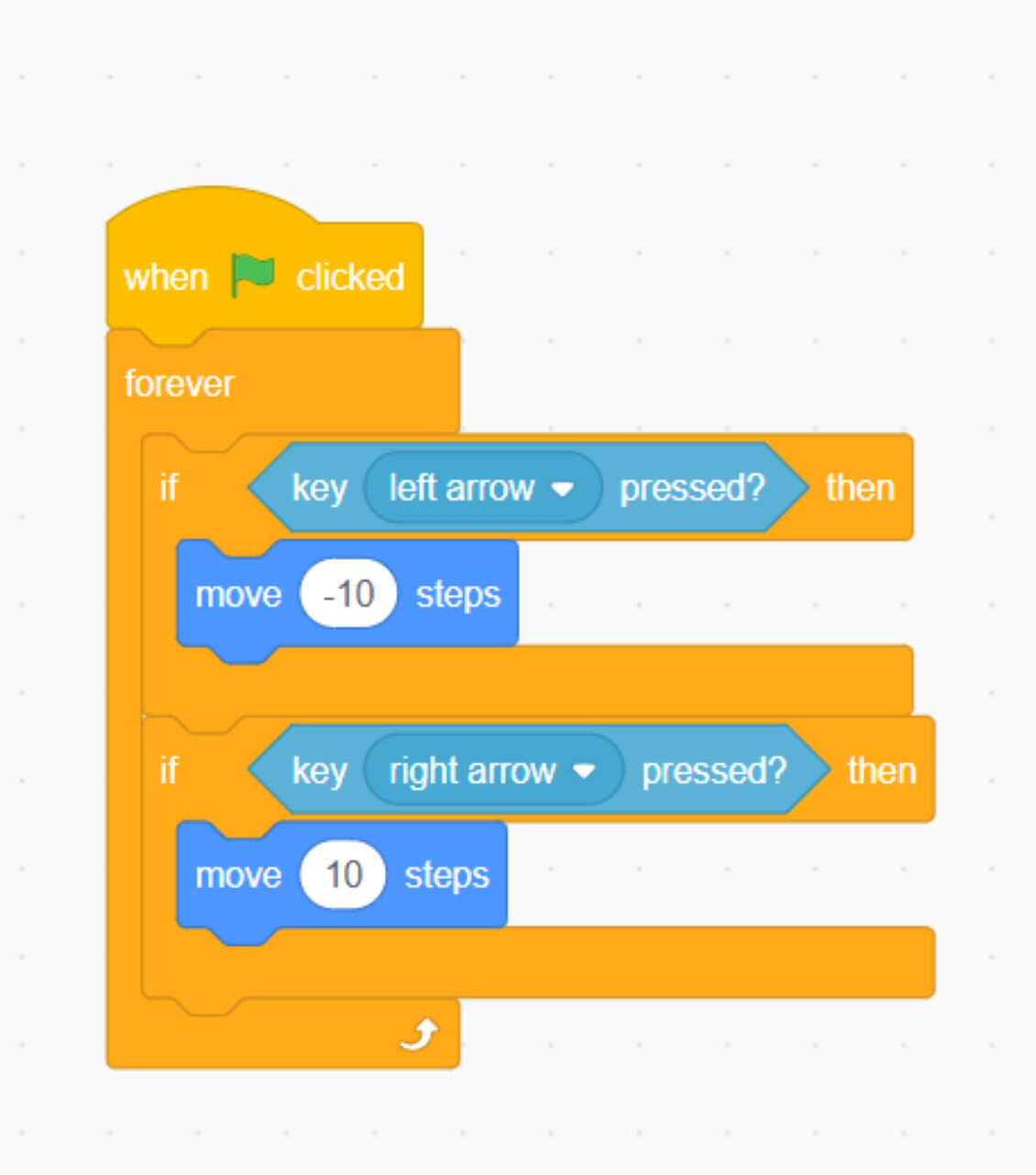


As one final feature, we must speed up the meteor as the game progresses. So, on the variables tab at the bottom, set the value in the "change ____ by" block to be our variable speed, and then set the value at the end to be -1. This will ensure that our speed increases every time we reset the planet to the top of the screen.

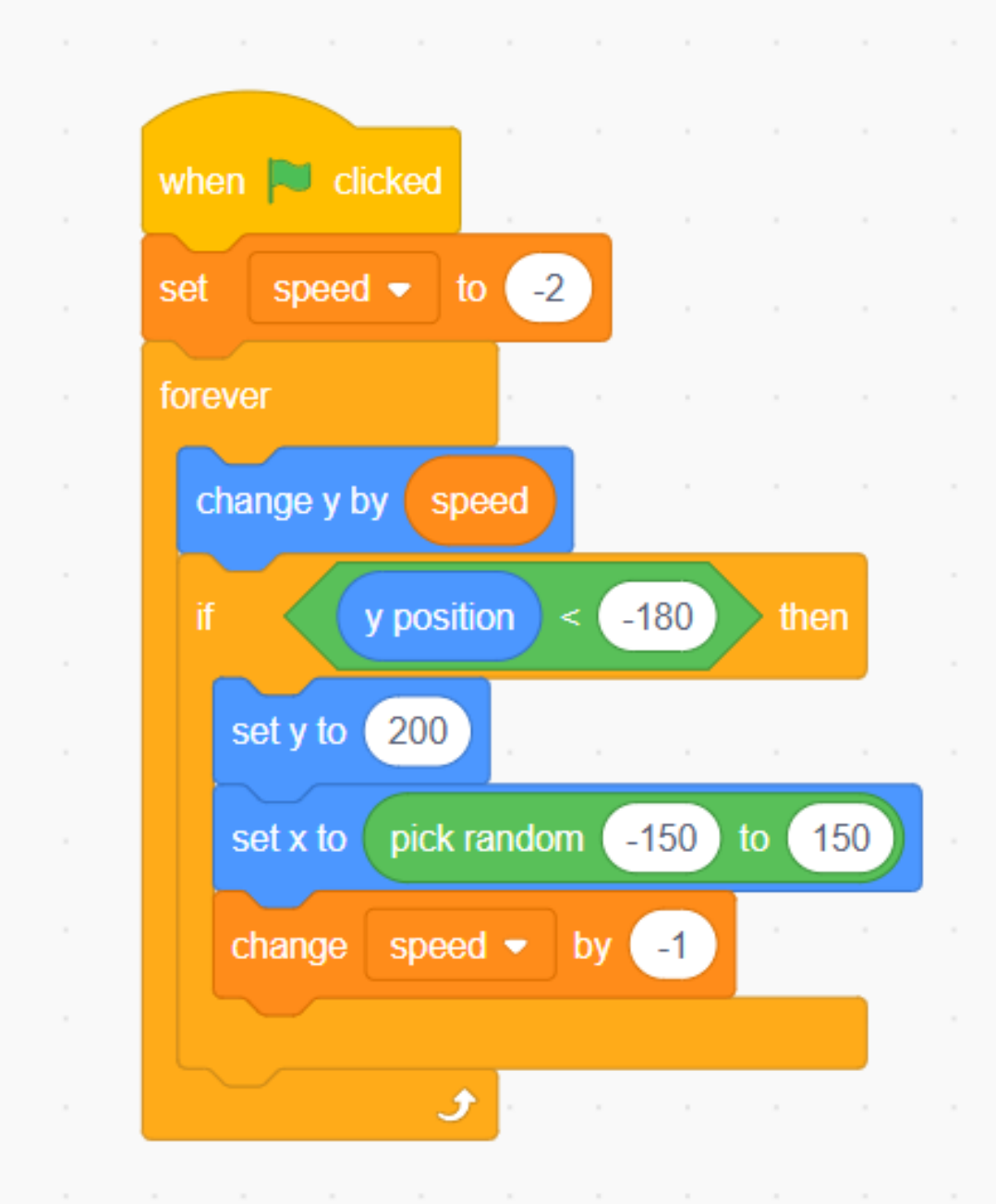


Here's the finished code from this lesson:

For the rocket:



For the planet/meteor:



Optional practice: try these two challenges to extend your project.

Try to detect when the meteor has hit the player, using a "touching" block from the **sensing** tab, and then stop the game with a "stop all" block from the **control** tab.

Try to create multiple meteors (you might need to change the value in the "change speed by" block).

Use the project link below to explore an example implementation.

https://scratch.mit.edu/projects/1084757736

Save your project and test the code before continuing.

Lesson 5

Smiling Creek: Coding & Design with Scratch - Lesson 5

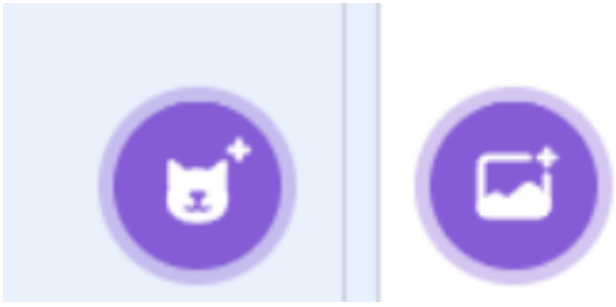
Lesson overview

In this lesson we learned a few new concepts in coding, and also a new type of block in Scratch.

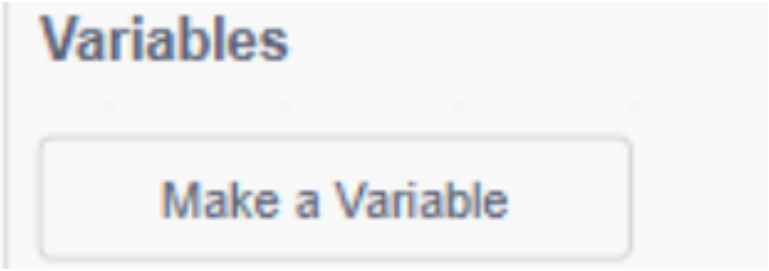
We learned how to create a **gravity** variable, and how to impose this onto our sprite so that it falls. On top of this, we also used this variable to make a more realistic jump for our sprite. Also, we learned about different sprite '**costumes**' in scratch, and how to switch between them to animate a character.

All of these concepts were used to create a 'Flappy Bird' style of game.

To begin, we created a 'Bat' sprite, and chose a 'Woods' backdrop. If you forgot how to create these, the buttons are below:



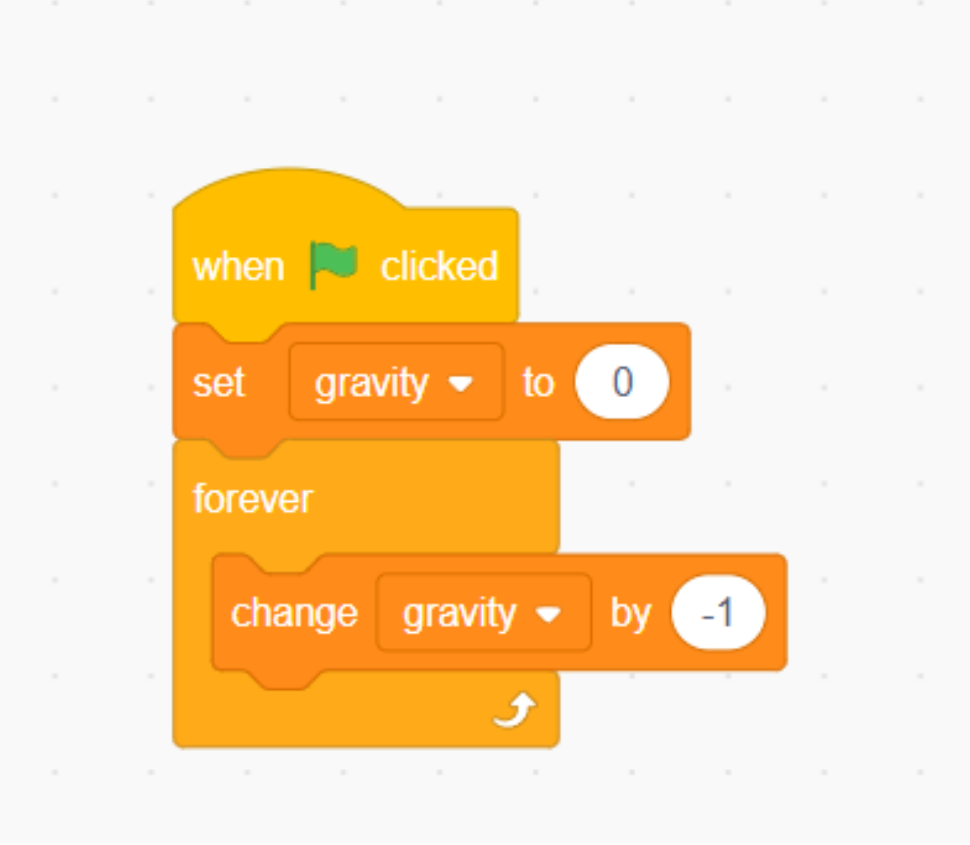
Once this was completed, we make sure we're in the Bat sprite's workspace, and then go to the orange **Variables** tab and select "Make a Variable". Name your variable "gravity", and then click "OK". This is the gravity variable that will affect the bat's movement.



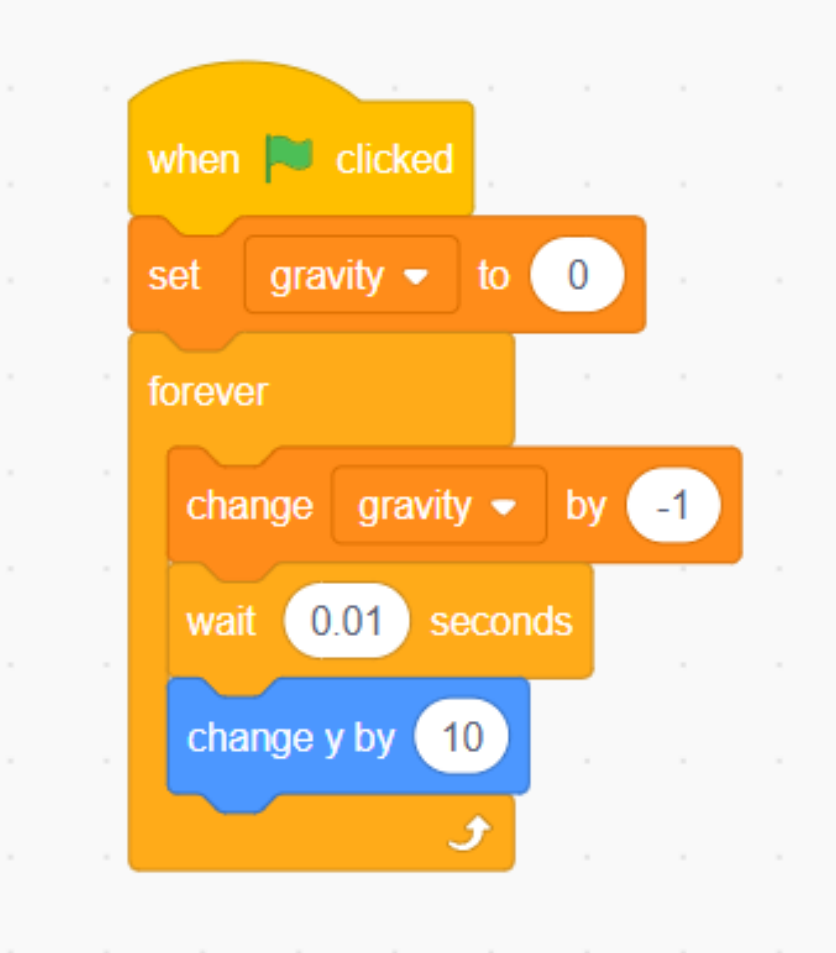
The way this gravity will work is as follows: every fraction of a second, the bat's y-position (up and down position) will be changed by whatever the 'gravity' variable is. If the variable is negative, the bat will go down, and if it is positive, the bat will go up. Also, if the gravity is-5, it will go down much faster than if it was-1. On top of this, the gravity variable will be constantly going down, and pressing the 'jump' key will reset this value to+10.

To translate this into code, we first need to drag in a "when flag clicked" button, and then from the orange **Variables** tab a "set _____ to 0" block. Make sure that the selected variable in the blank space is "gravity".

Next, drag in a "forever" block from the **Control** tab, and then inside of it drag in a "change _____ by __" from the orange **Variables** tab. Select "gravity" as the variable and "-1" as the value.



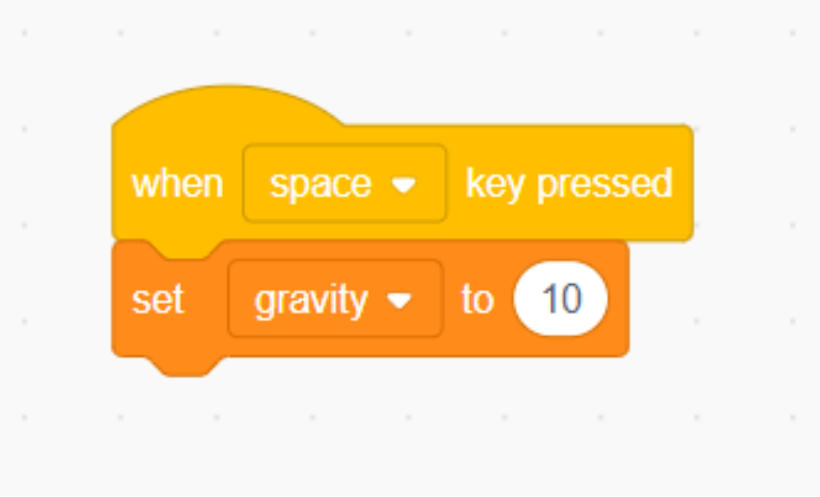
After this, drag in a "wait __ seconds" from the **Control** tab, and then a **Motion** "change y by ___" block. Set the value in the "wait __ seconds" block to be "0.01".



Next, drag in the "gravity" block from the orange **Variables** tab, and place it into the "change y by __" block.

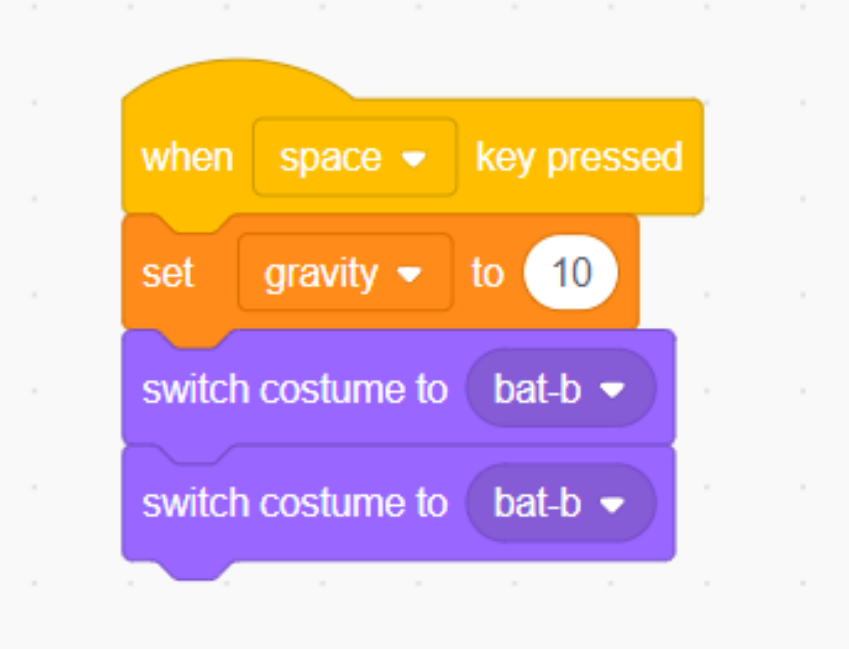
This chunk of code will make sure that at the beginning of the game, the gravity is set to 0, that the gravity value is constantly going down, and that every 0.01 seconds, we're affecting the y (vertical/up and down) position of the bat.

Next, to make the bat jump, all we need to do is drag in a "when space key pressed" block from the yellow **Events** tab, and then a "set gravity to ___" from the **Variables** tab, setting the number value to 10.

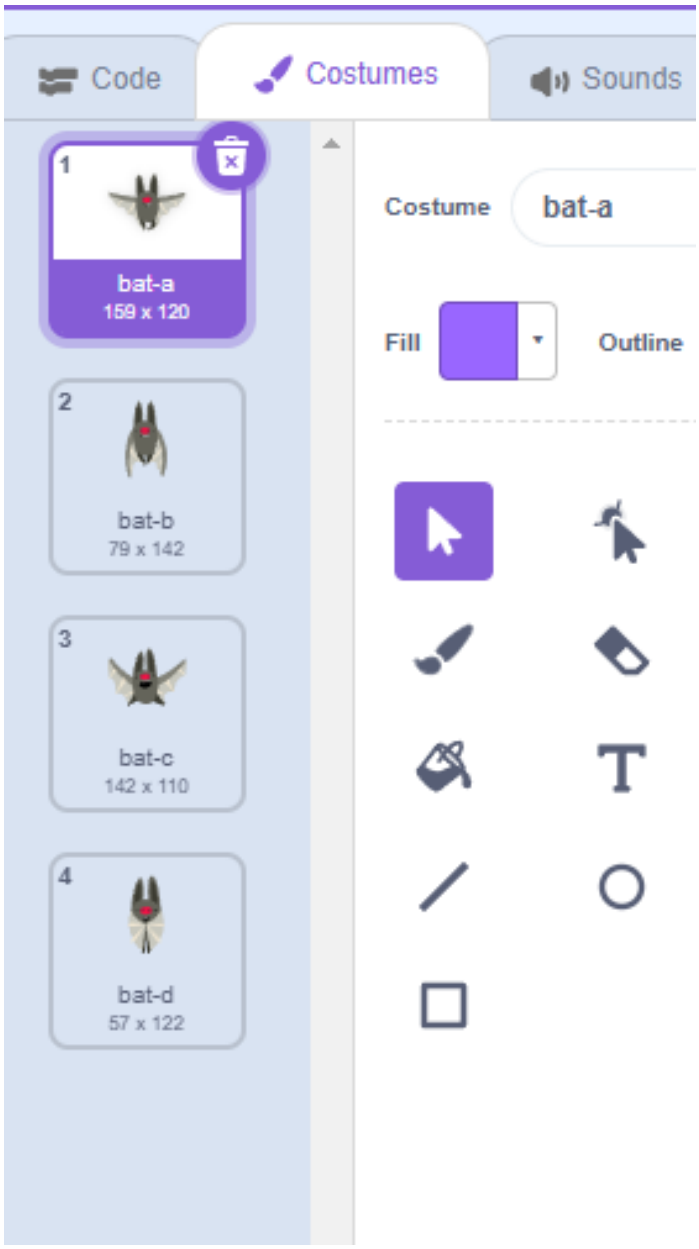


This will allow our bat to fall and jump. Make sure to click the flag button to begin the game!

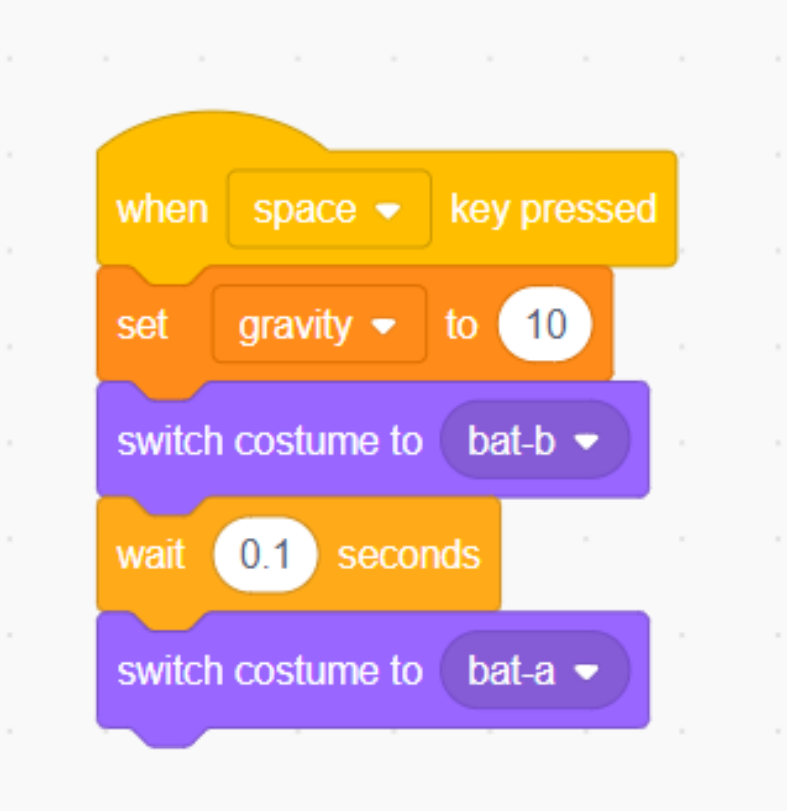
Next, to animate the bat, we can go to the purple **Looks** tab and drag in 2 "switch costume to" blocks at the bottom of the jump chunk of code. These will switch the appearance of the bat.



To view the different costumes of a sprite, simply go to the "Costumes" tab in the top right. If we view the Bat's costumes, we can see that bat-b has its wings down, and bat-a has its wings up. Combining these costumes will let us make a simple jump animation.



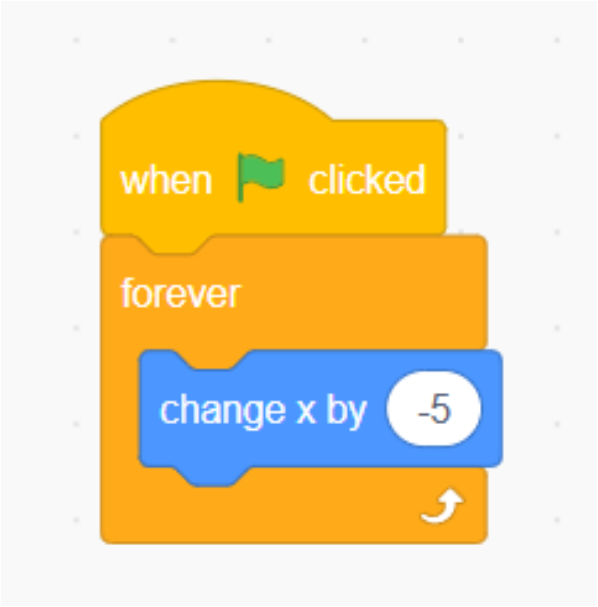
So, for the first "switch costume to" block, set the blank at the end to be "bat-b", and for the second block, "bat-a". Then, simply go to the orange **Control** tab, and drag in a "wait __ seconds" block, setting the value to be "0.1". This will add a gap before resetting the costume after we jump.



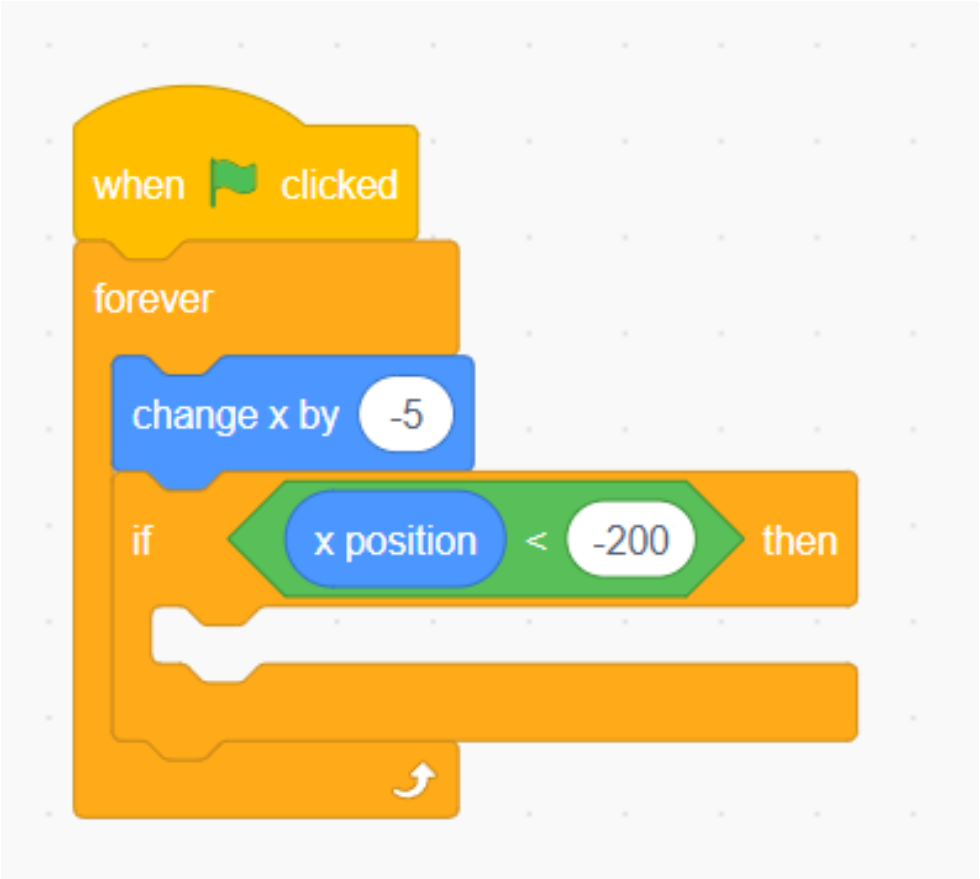
Once we have our working, jumping bat, its time to make some obstacles.

To do this, create a new sprite. It can be any sort of obstacle, but for my project, I chose the "Buildings" sprite.

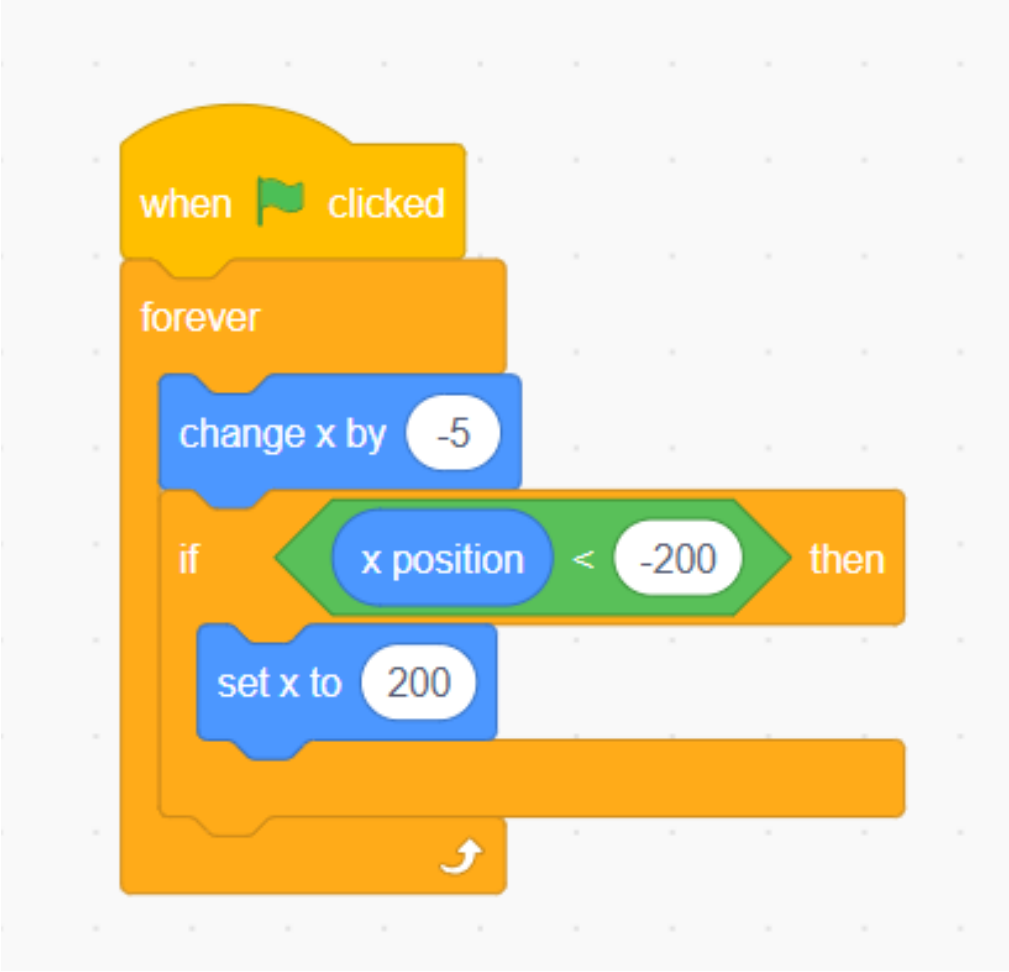
Once this sprite is created, drag in a "when flag clicked" and "forever" block inside of it. Then, go to **motion**, and drag in the "change x by __" block, setting the value to "-5". This will ensure that the building is constantly moving to the left.



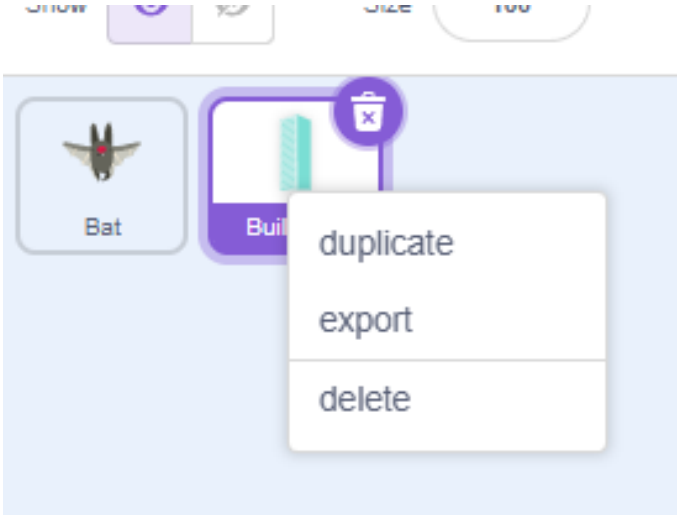
After this, drag in an "if" block from the orange **Control** tab, and then a (___ < ___) block from the green **Operators** tab. From the blue **Motion** tab, drag in the "x position" block into the first slot, and then type in "-200" to the second slot". This will detect when the building has reached the edge of the screen.



Then, from the blue **Motion** tab, drag in a "set x to __" block, and set the value to be 200.

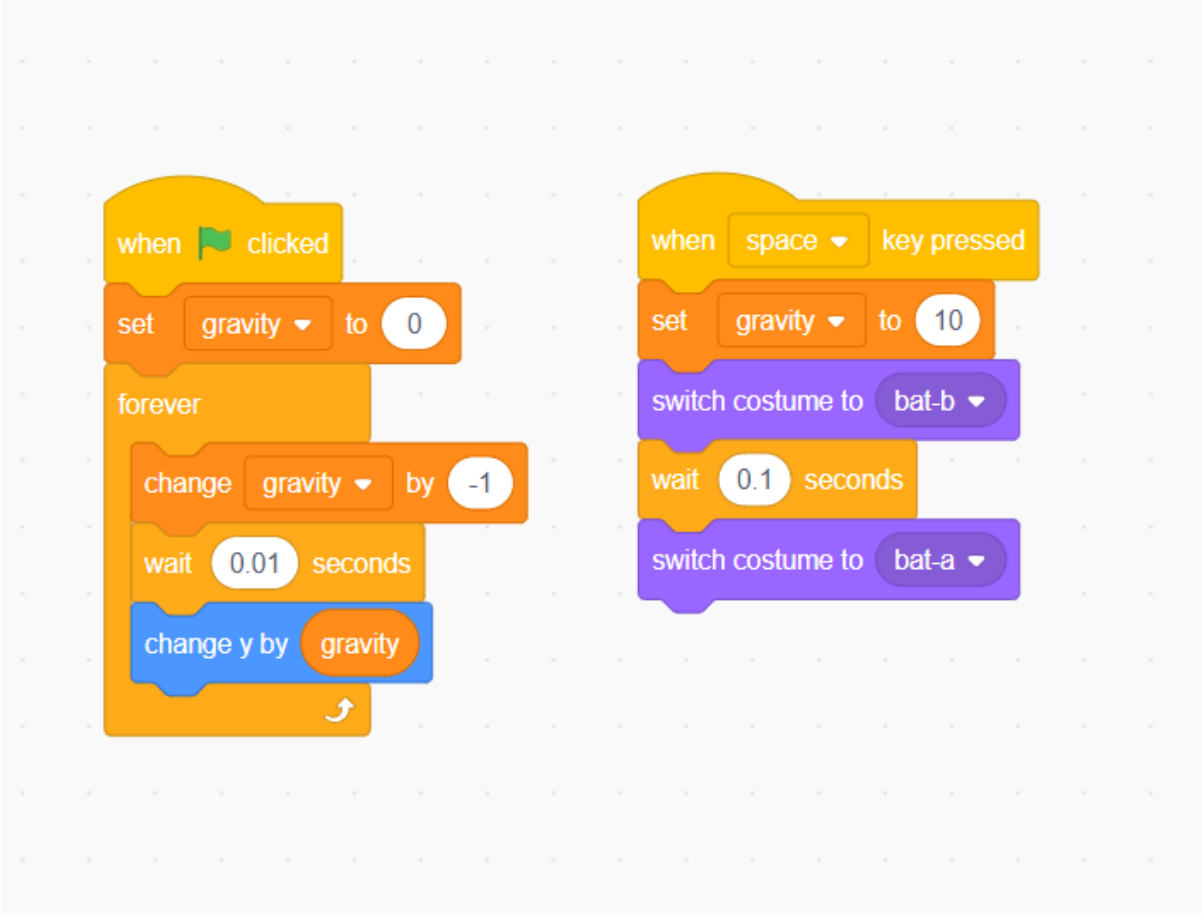


Once this is completed, right-click (or click with two fingers if you're on Mac) on the "Buildings" sprite, and then select "duplicate", and drag this new duplicated sprite to be above the other building. This will make a sort of corridor for the bat to jump through.

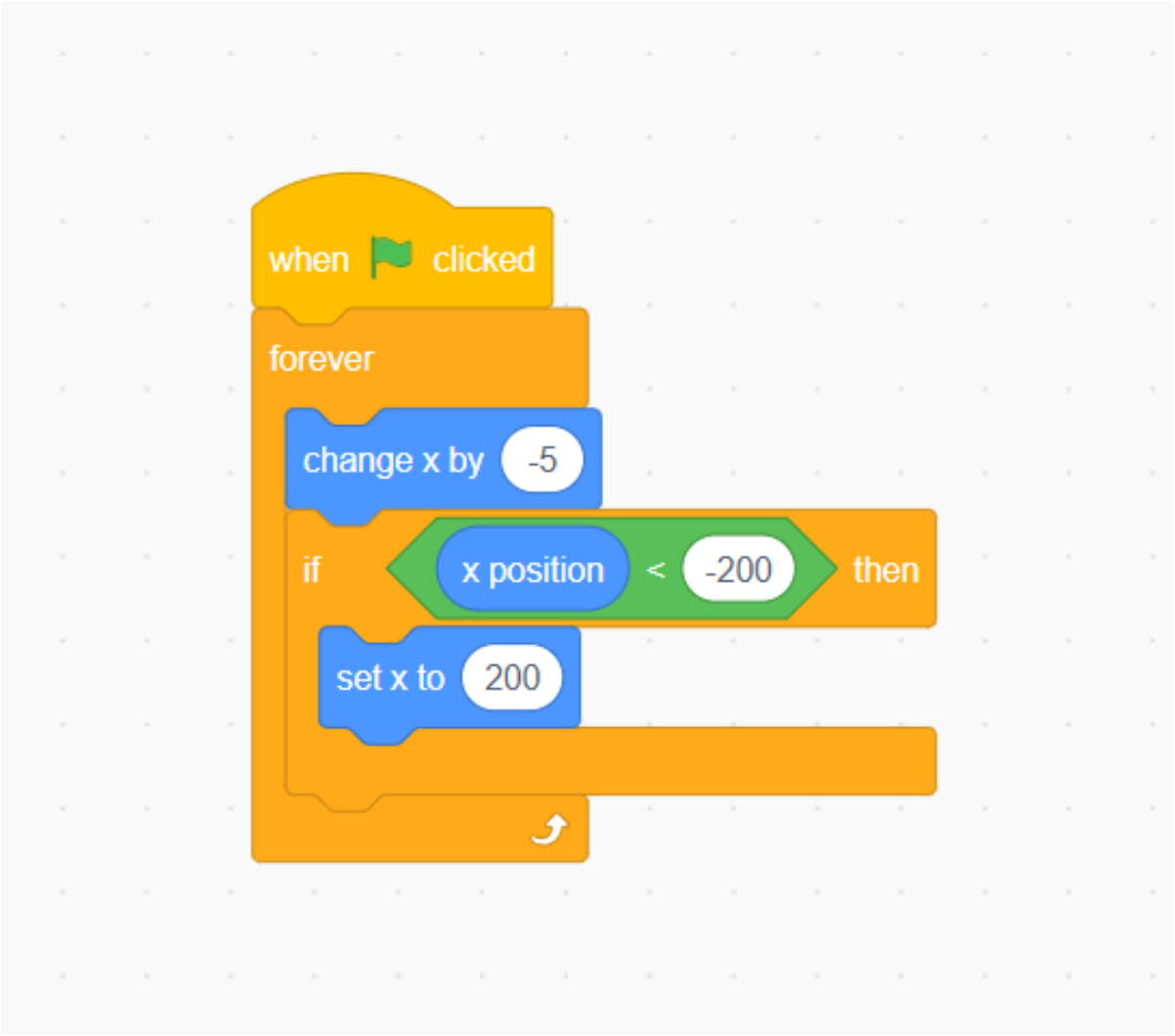


Here's the finished code:

For the Bat:



For the Buildings:



The project link is below :

https://scratch.mit.edu/projects/1088239591

Save your project and test the code before continuing.

Lesson 6

Smiling Creek: Coding & Design with Scratch - Lesson 6

Lesson overview

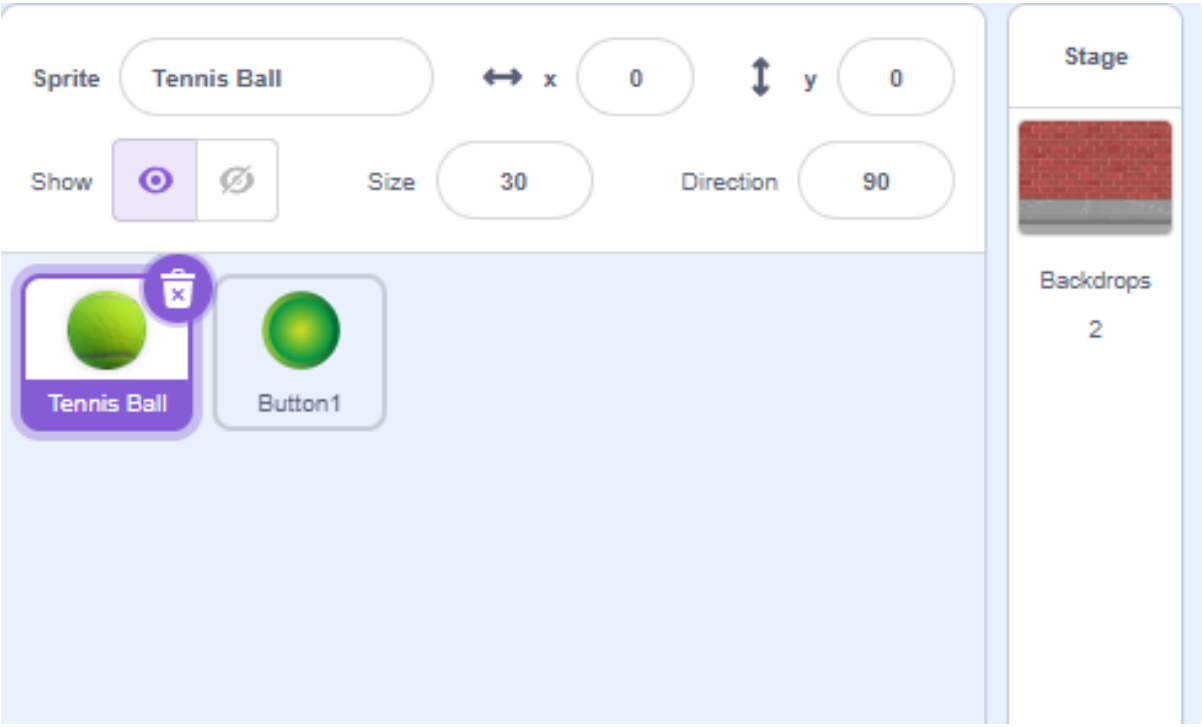
Work through this project at your own pace.

In this lesson, we explored the idea of using 3D effects in our games, and changing the **Size** of our sprites.

We were able to apply variables to the **Size** of our sprite, explored the idea of "random" a little further, and also implemented "if ___ then, else ___" blocks/statements into our code.

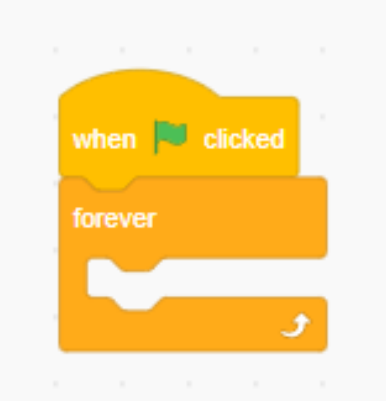
We used these concepts together to create a "Wall-Ball" type of game.

To begin, we created 2 sprites, called "Tennis Ball" and "Button1", and we set the backdrop to "Wall1".

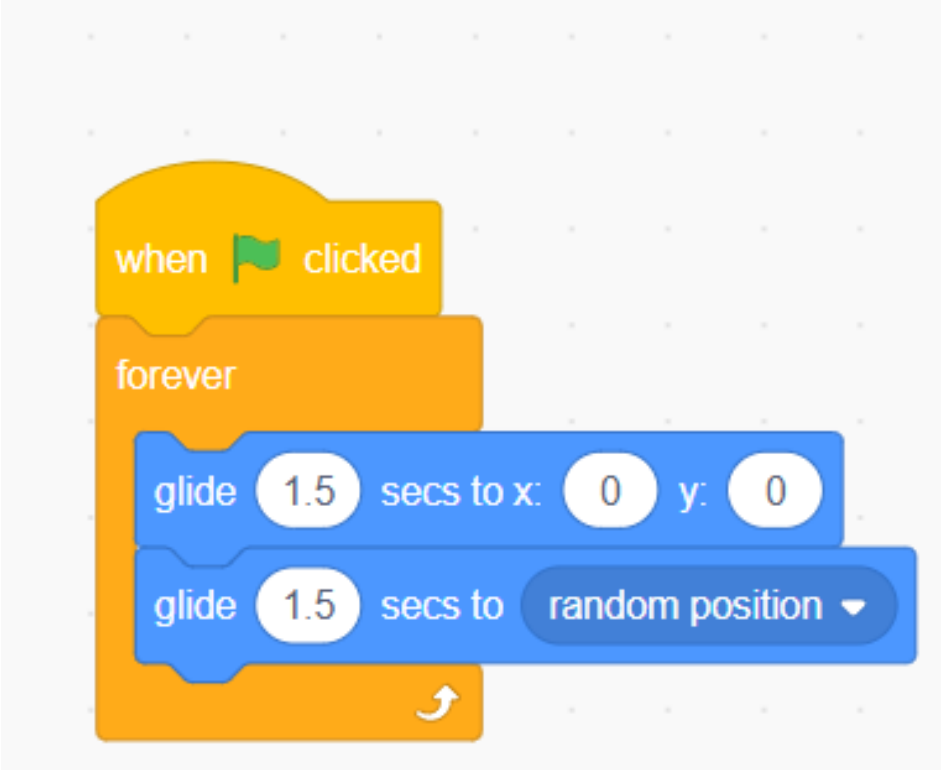


Once this was finished, we began our code inside of the "Tennis Ball" sprite.

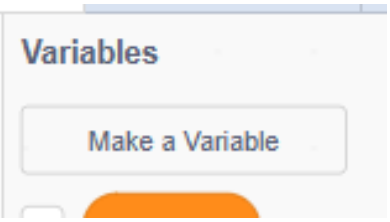
To start, as usual, we bring out an **Events** tab "When flag clicked" block and a **Control** tab "forever" block.



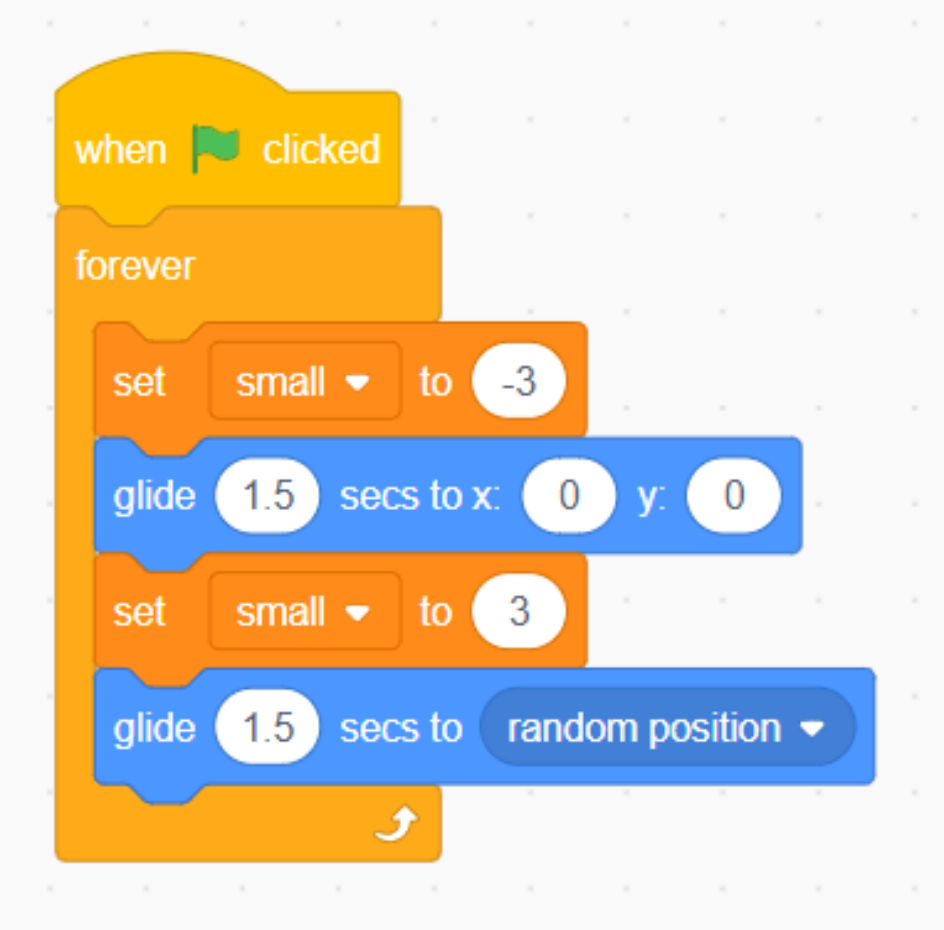
Inside of this, we go to the **Motion** tab, and drag in a "glide __ secs to x: __, y: __" block, and a "glide __ secs to random position". We set the 'secs' values in both blocks to be "1.5", and then for the first block, set the values to be "x: 0" and "y: 0"



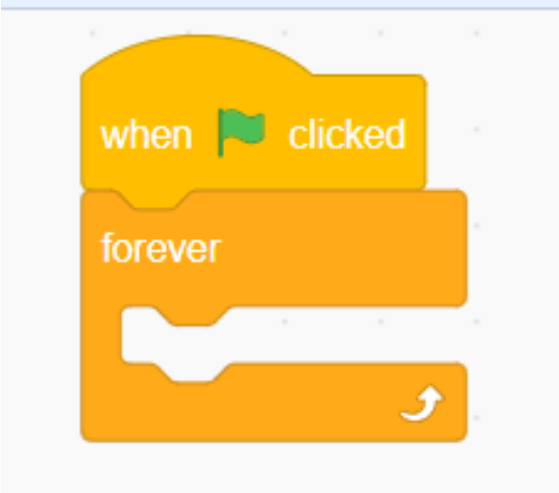
Once this is finished, we create a variable named "small" or "size" or anything like that. To create a variable, go over to the **Variables** tab and select "Make a variable". Then, type in the variable name. This variable is what we use to control the size-change of the Tennis Ball, meaning that whenever this variable is positive, the Ball will be growing, and whenever it is negative, the Ball will be shrinking.



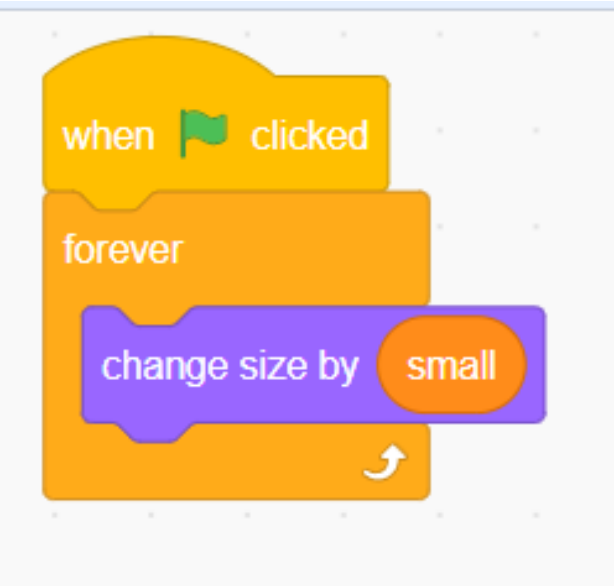
To implement this, go over to the **Variables** tab, and drag in 2 of the "set ___ to __" blocks. Place them before both of the "glide" blocks. Then, set the variable to be "small" or "size" or whatever you chose, and set the value in the first block to be -3, and the value in the second block to be+3.



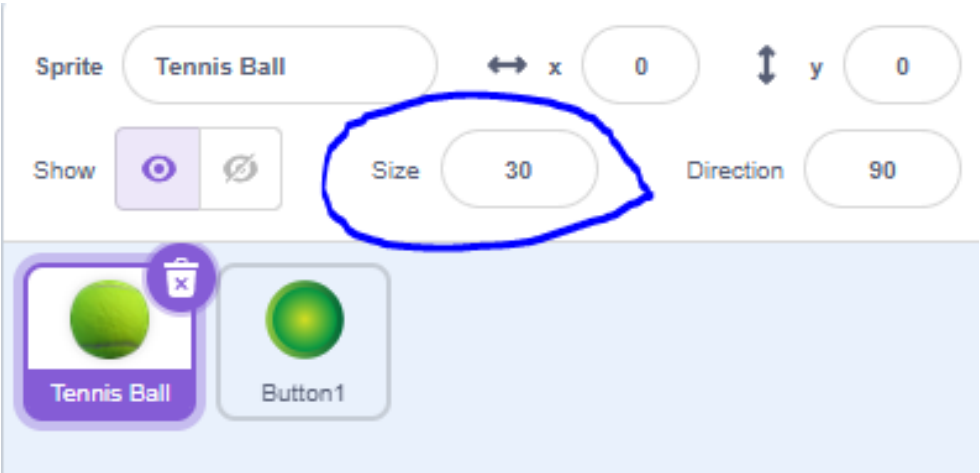
Then, to allow this variable to change the size of the Ball, drag in another "when flag clicked" with a "forever" block underneath.



Inside of this, go to the **Looks** tab, and drag in the "change size by ___" block. Then, go to the **Variables** tab, and drag your variable into the blank space.



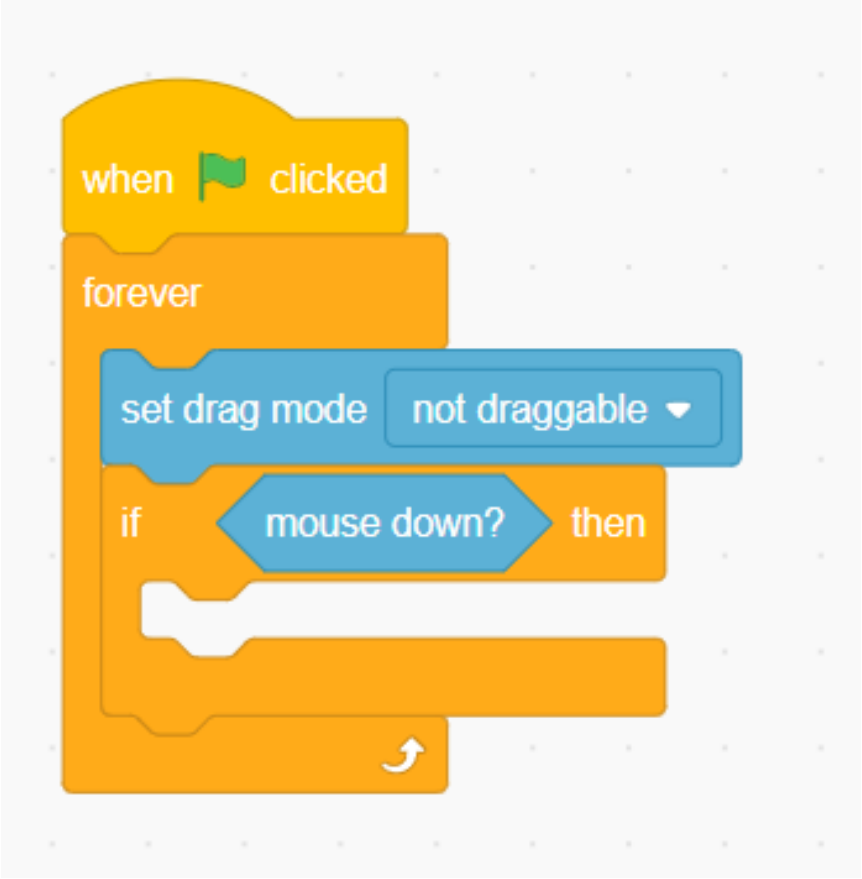
Once this is complete, the ball should be constantly growing and shrinking. Note that if the ball ever gets too big, you can always go into the "Size" aspect of the sprite and change it directly there.



Next, lets begin the button/paddle sprite.

As you could have guessed, we're starting with a "when flag clicked" and "forever" block underneath. However, inside, we want to place a "set drag mode ____" block from the **Sensing** tab. We need to make the value in this block "not draggable", and this will ensure that the sprite will function correctly when we move it in the game.

After this, we want to add an "if ___ then" block, and drag in a "mouse down?" block from **Sensing**. This will check if the mouse has been pressed (or the screen has been pressed on tablets).

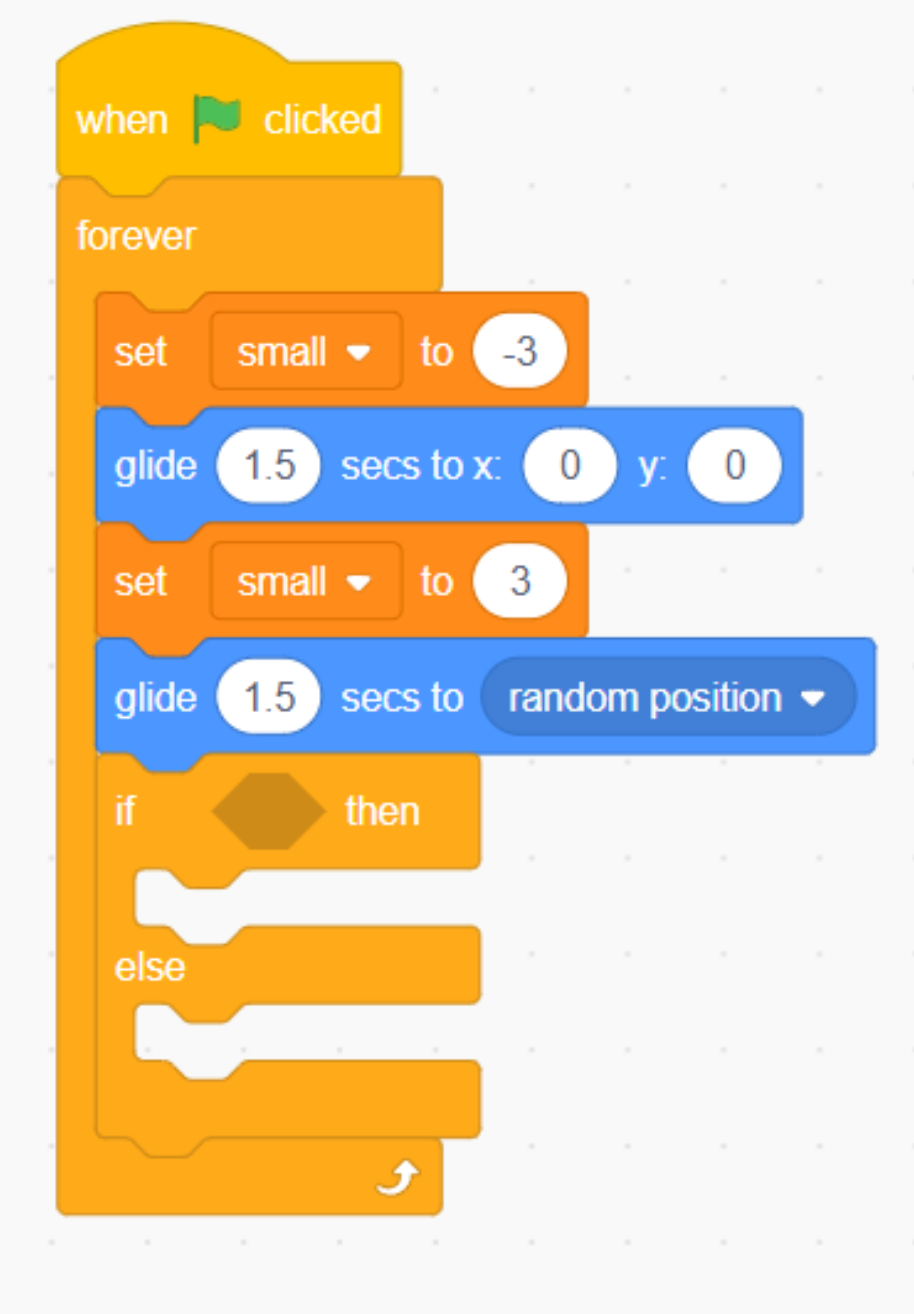


Inside, from the **Motion** tab, we'll place a "go to ___" block, and set the value to be "mouse-pointer". Now, we should have a working paddle.

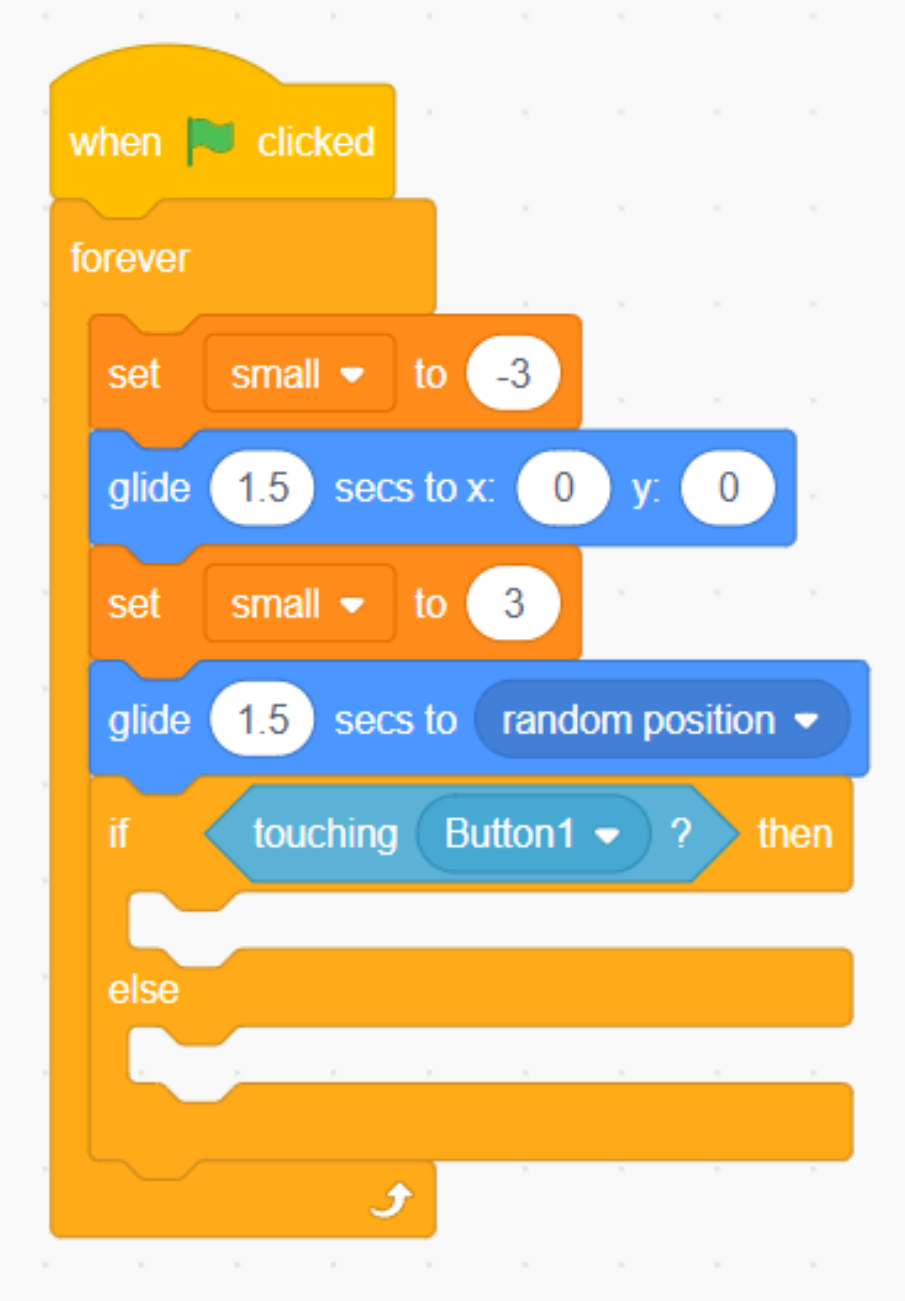
Lets move back over to the Tennis Ball code.

To detect if we've 'hit' the ball, we'll use a new block, from **Control**, the "if ___ then, else" block. This block will check if the first condition is satisfied, and if not, it will run the second part of its code. We've placed this block so that it runs only when the ball is at its peak size, or when it is 'hitting' the screen. Because of this, we'll make our first condition to check if our button is touching the ball, and this will increase our score, and if we aren't touching the ball at this time, we'll reset our score to 0.

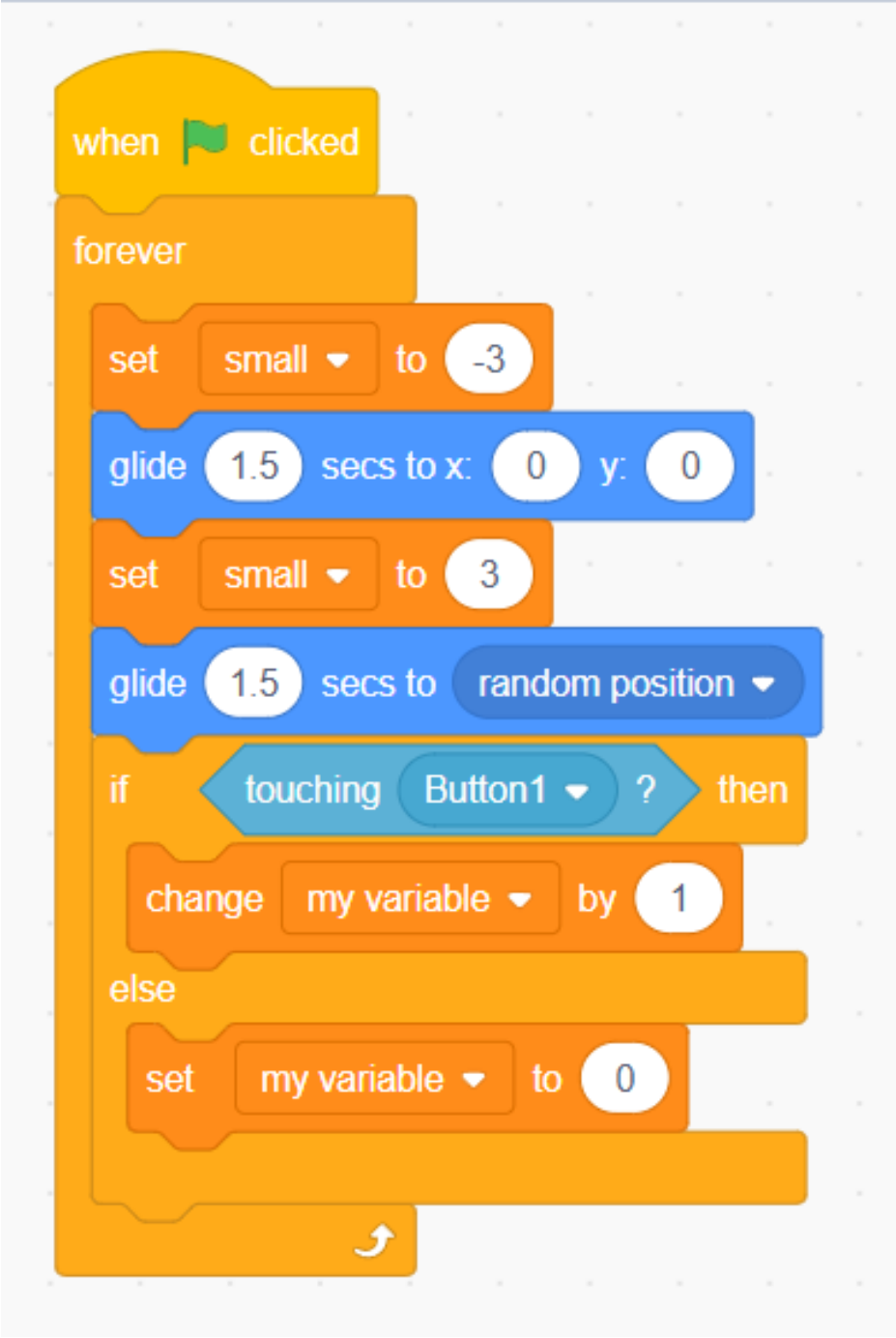
To do this, lets first drag in the **Control** "if ___ then, else" block.



Inside of the space after "if", lets drag in a "touching ___" block from the **Sensing** tab, and lets set the value inside to be the "Button1" sprite.

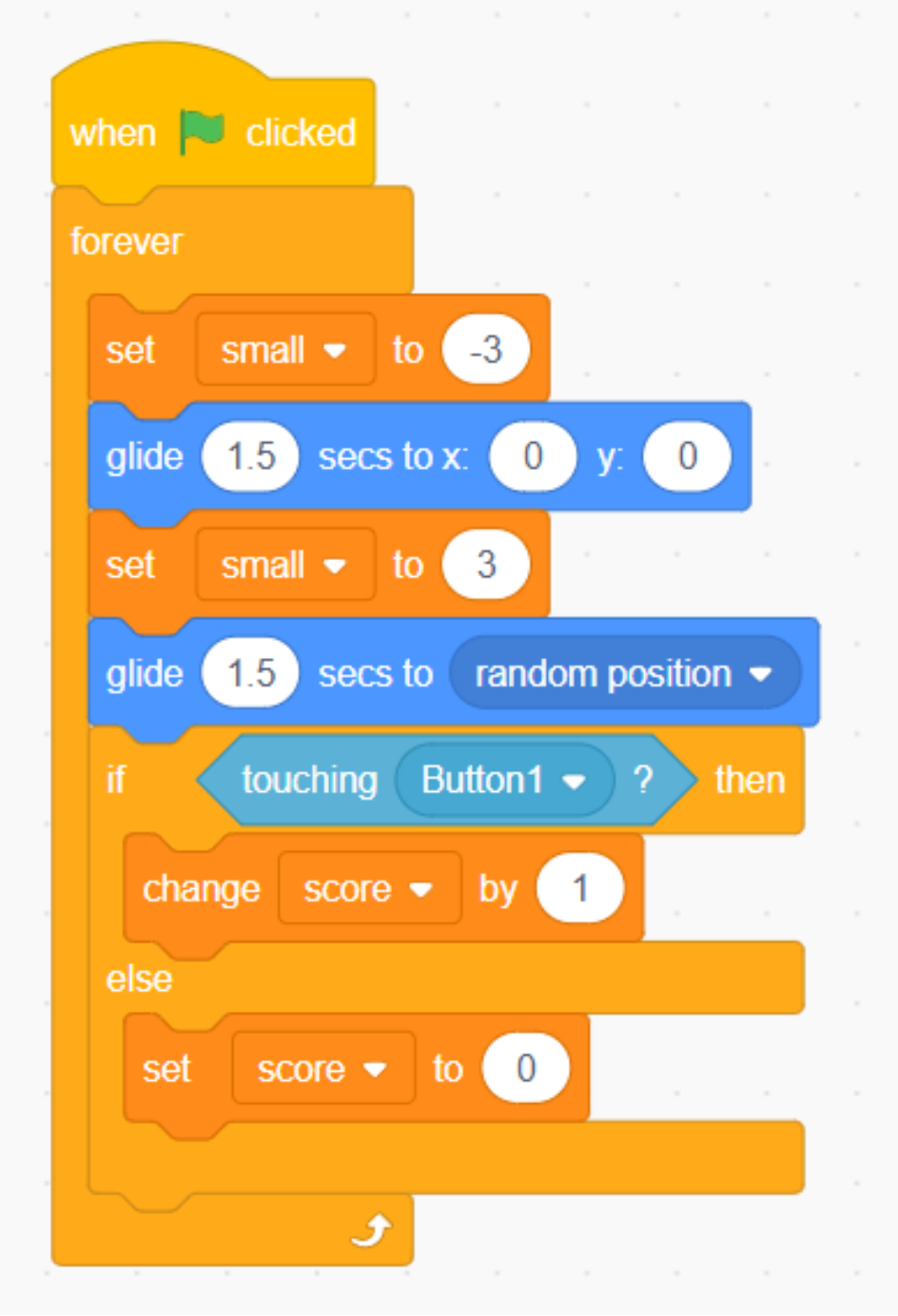


Then, go over to the **Variables** tab, and drag in a "change ___ by __" block into the first space, and then a "set ___ to __" block into the second space.

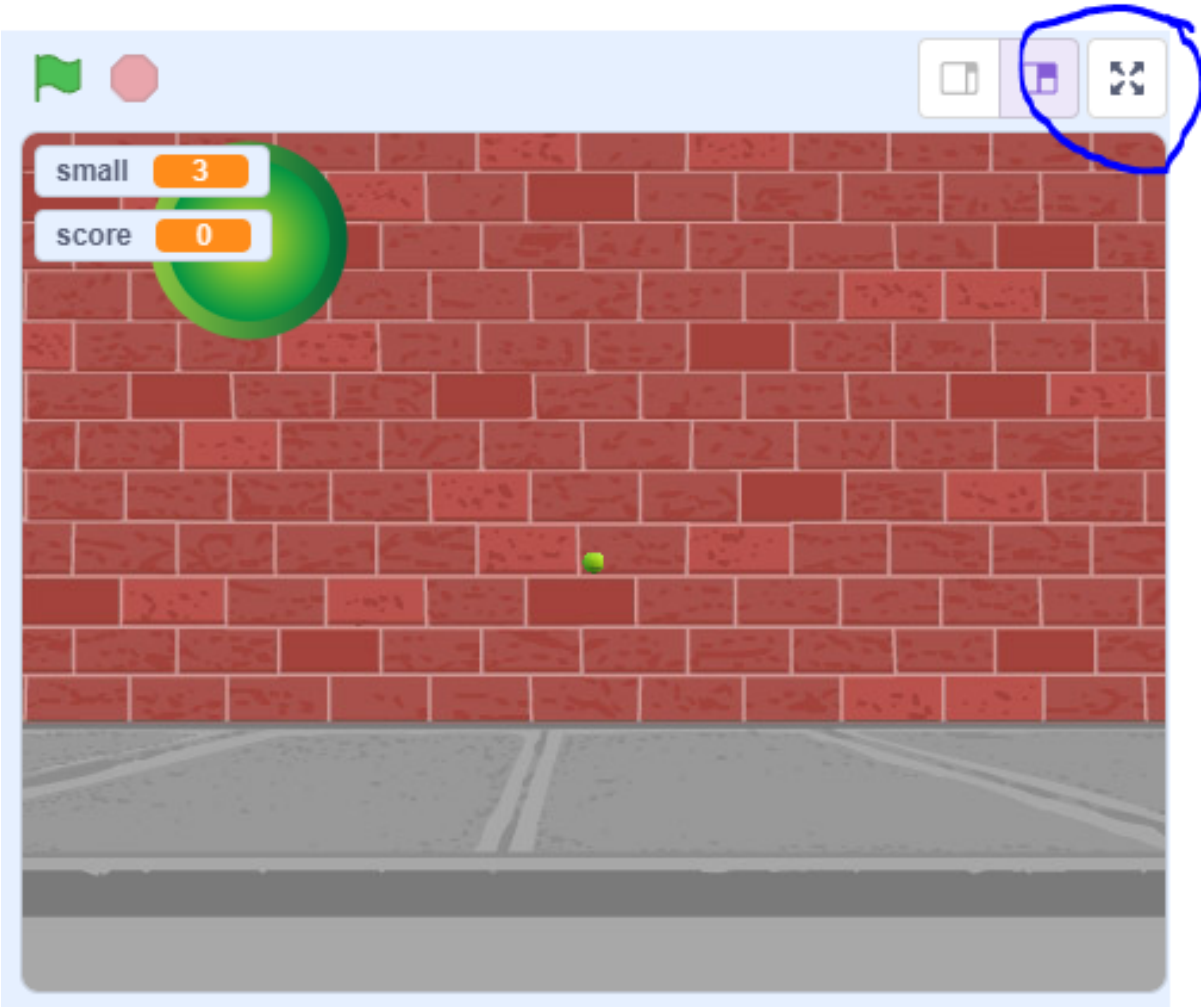


Finally, we need a "score" variable to change. So, while still in the **Variables** tab, select "Make a Variable" again, and name this one "score".

Then, set the _____'s of the variable blocks we just added to be "score", and set the values to be 1, and 0.

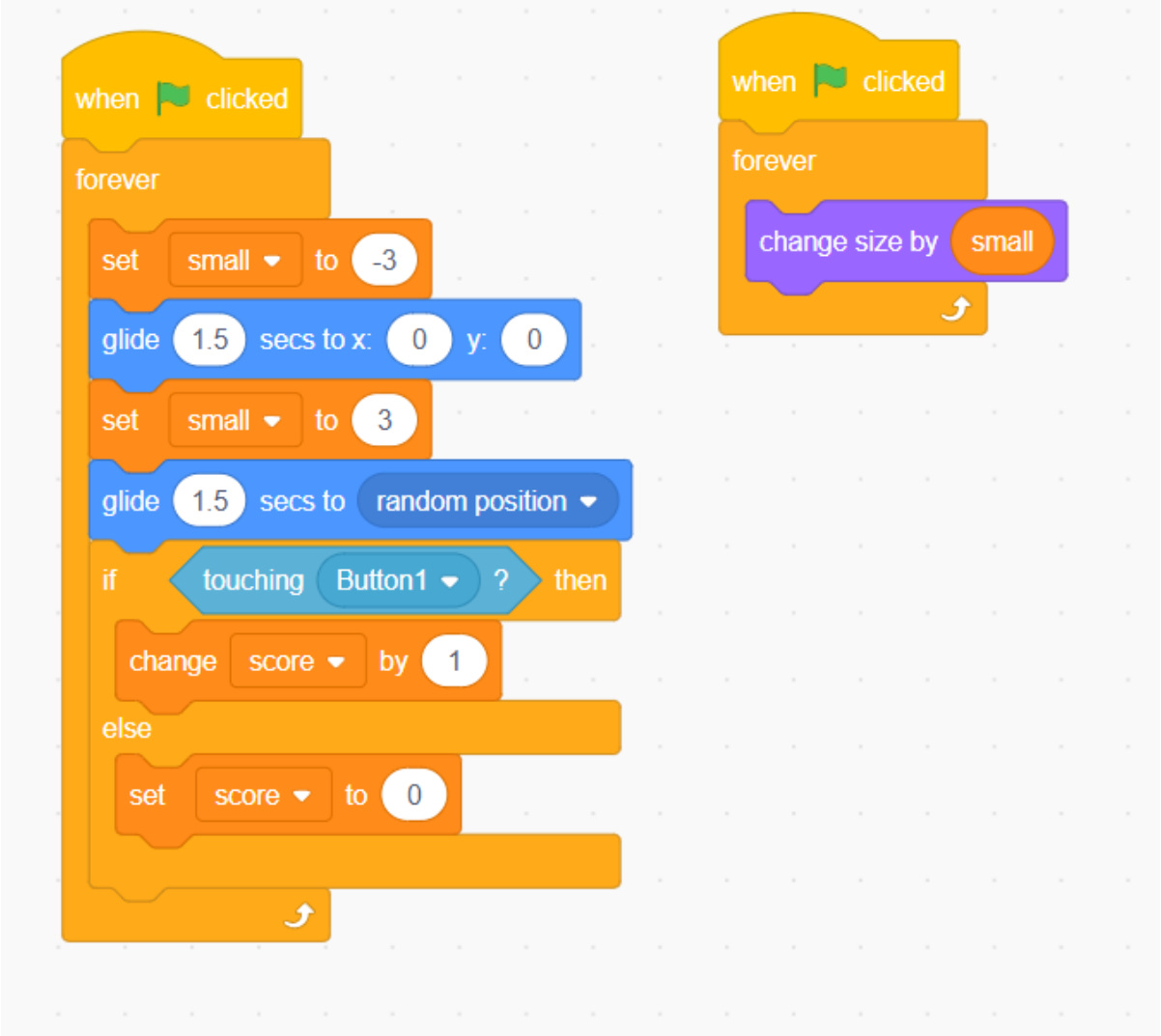


NOTE: Make sure to play the game in full screen, as the dragging gets messed up in the code-editor window. To do this, click the Fullscreen button:

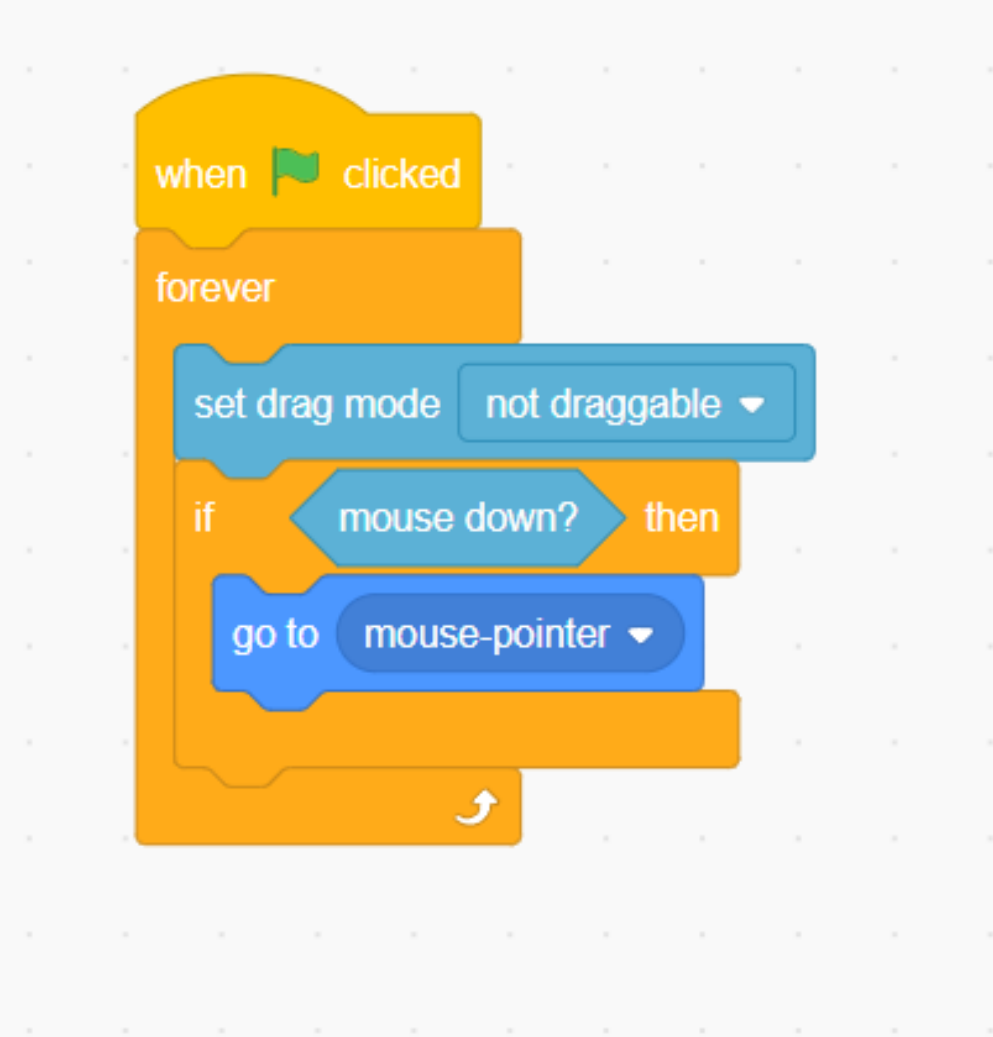


Here's the finished code:

For the Tennis Ball:



For the Button:



Here's the project link :

https://scratch.mit.edu/projects/1091438162

Save your project and test the code before continuing.

Lesson 7

Smiling Creek: Coding & Design with Scratch - Lesson 7

Lesson overview

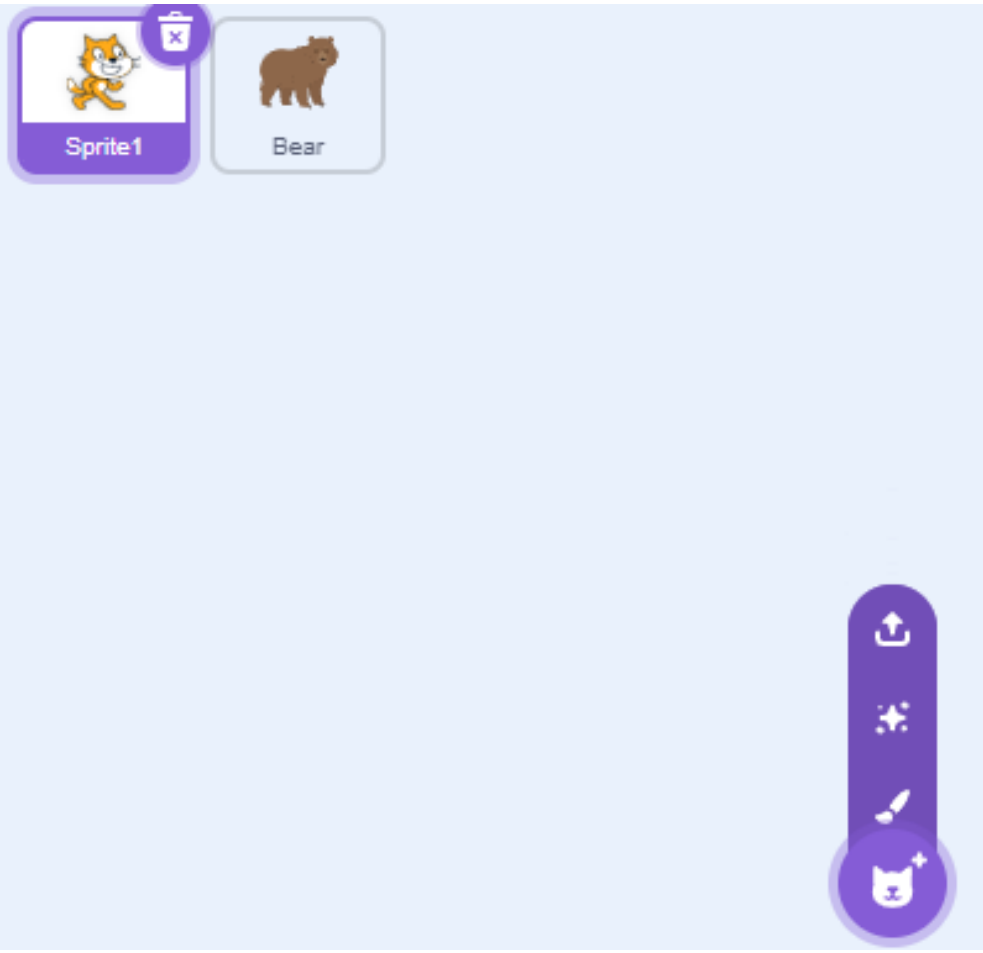
In this lesson, we followed a more theory-based lesson, and we learned different ideas of how to make a game-over screen in scratch.

We learned about the "show" and "hide" blocks, the use of a homemade Boolean variable in scratch, and also the "switch backdrop to" block and how to edit our own background.

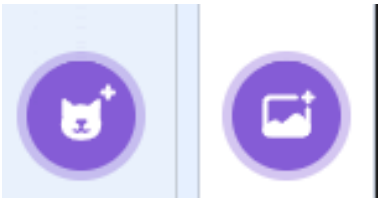
Mainly, this lesson was about learning the main ideas behind a game-over screen, and creating a template which can be used for other games we've created.

To begin, we needed to create a simple "game" where there was some way to lose. To do this, we created a sprite that follows the mouse-pointer, and then decided that "game over" will be triggered when we touch a second sprite in the corner of the screen.

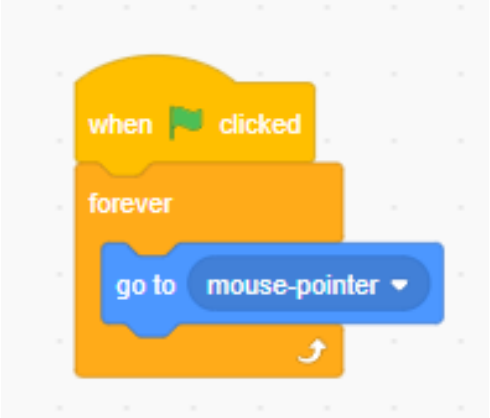
We first create two sprites, placing the second sprite in the corner of the screen.



Then, we need to go to backdrops, and select two backdrops: one for the game-over screen, and one for the regular screen. Don't worry, both backdrops won't show up on the right, but they'll be loaded invisibly for us to use later.

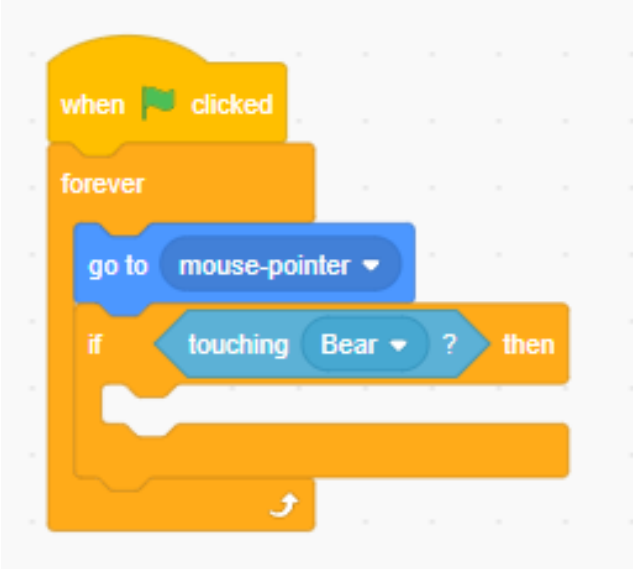


To create our test game, we first added a **Events** "when flag clicked" button, then a **Control** "forever" block, and finally a **Motion** "go to _____" block. We then set the blank in the motion block to be "mouse-pointer".



Then, we need to detect when we're touching the sprite in the corner.

To do this, we'll drag in a **Control** "if" block, and then inside of it drag in a **Sensing** "touching ___" block. Set the blank to be the secondary sprite. (NOTE: make sure that the secondary sprite isn't in the middle of the screen.)



Then, we need to set off a function that will do everything we need for a game over screen,

This will contain:

Changing the background,

Hiding both sprites

Ending the game

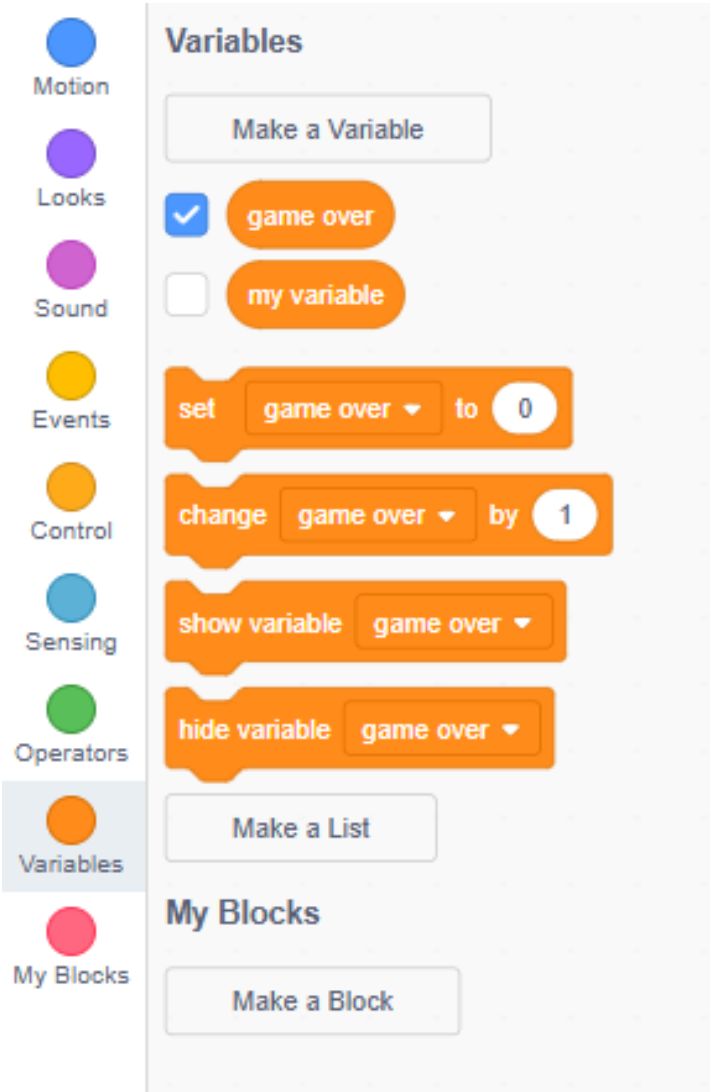
and whatever else someone might want to do at the end of a game.

In order to understand how a "game-over" screen works, we need to understand what a Boolean variable is.

In programming languages, a Boolean is a type of variable that can be one of two things, either true or false. For example, we could have a boolean variable called "hit", and make it true when it hits an object, and false when it moves away.

Scratch doesn't contain basic boolean variables, but we can make our own by creating a regular variable, and then either setting it to "1" or "0", with 1 being true and 0 being false.

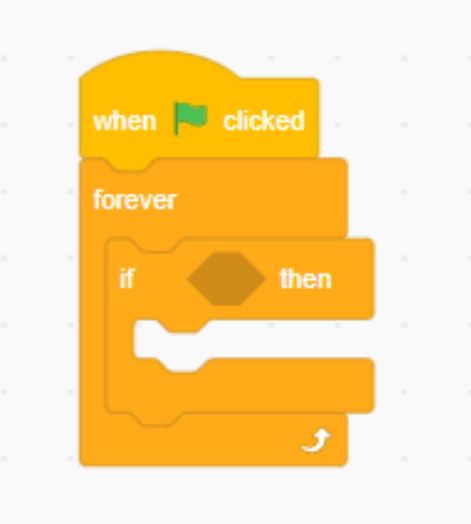
For our purposes, we'll create a variable named "game over", and then set it to "1" when we want our game to be finished, and then back to "0" when we start the game over again.



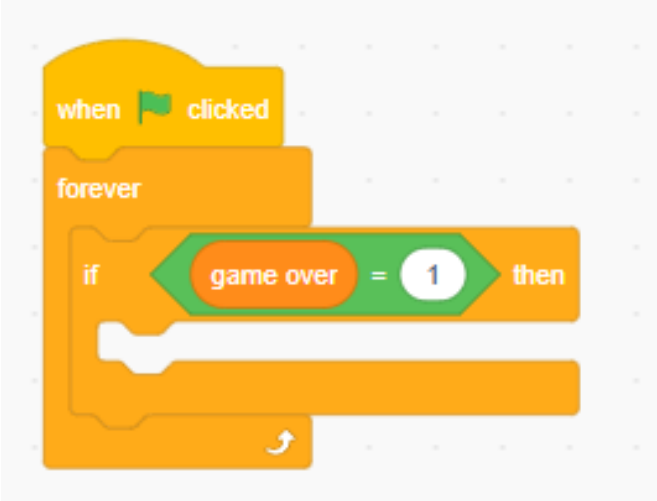
So, inside of our main blocks, after checking if we're touching the second sprite, go into the **Variables** tab and add "set ____ to __", changing the values to be "set 'game over' to '1'".

Then, we need a separate chunk of blocks which will check when game over is "1" or true, and then apply our changes.

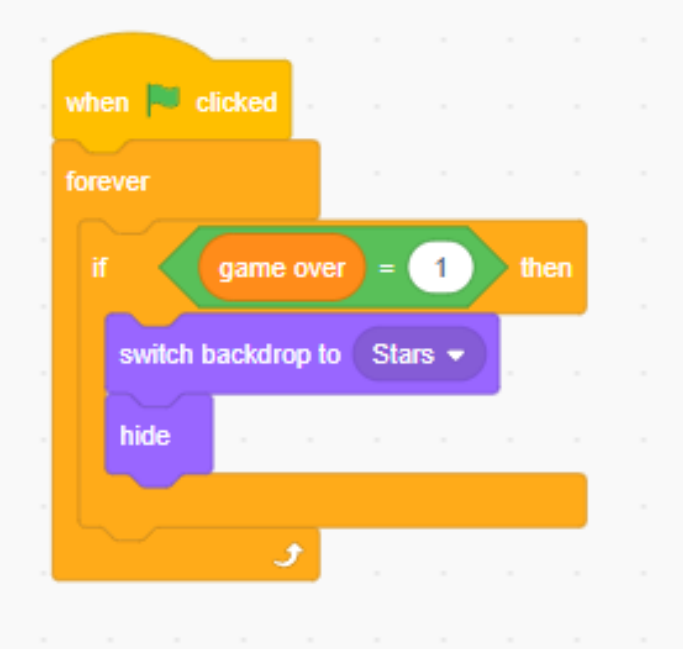
To do this, we'll drag in another **Events** "when flag clicked" block, **Control** "forever" block, and **Control** "if ___" block.



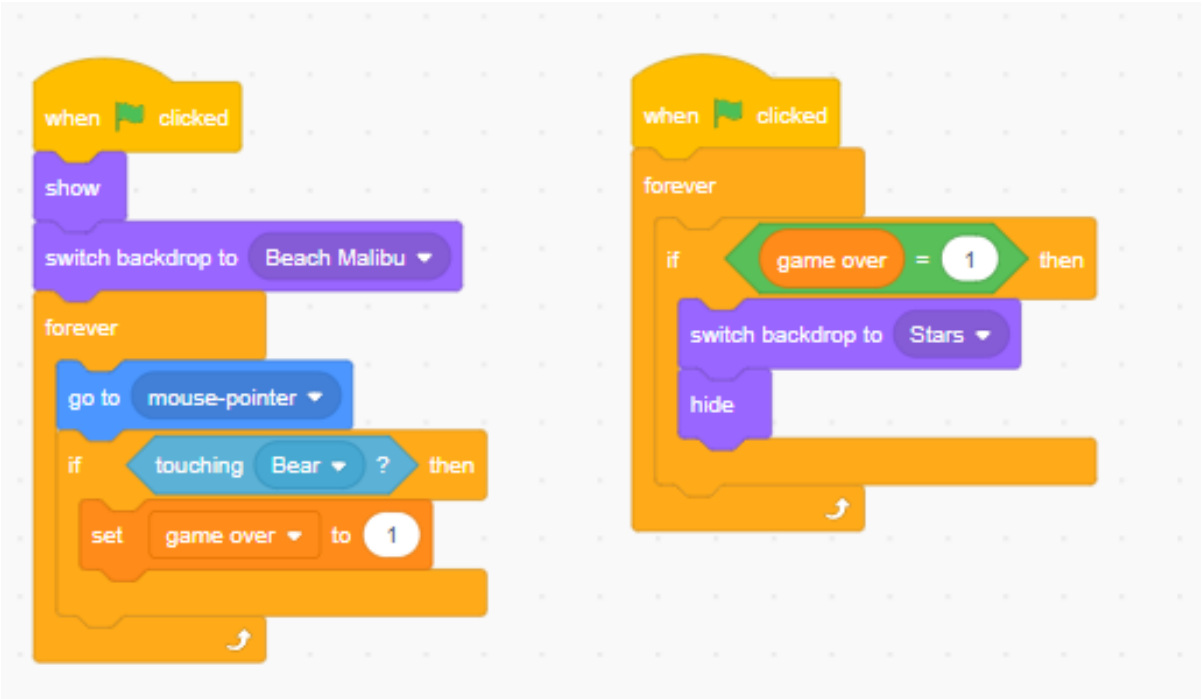
Inside of the "if ___ block", we go to the **Operators** tab, and drag in the "(__ = __)" block, then drag in our "game over" block from **Variables** and set the second value to be '1'. This code will check when the game is over.



Then, all we must do is add all of our changes, and then as a ground rule, make sure that these changes are undone at the very beginning of the next game. For example, we can use the "hide" block from the **Looks** tab to hide our character, and also the "switch backdrop to" block to switch the backdrop to our game-over backdrop.



But, to follow the ground rule, we must undo these changes at the beginning of the next game, and to do this, we can drag in the "show" block from **Looks** and the "switch backdrop to ___" block right after the "when flag clicked" block. Since they're outside of the "forever" block, these actions will only be completed once at the beginning of the game, letting us reset the game at the start. (NOTE: Make sure that there are two different backdrops in each of the "switch backdrop to" blocks.

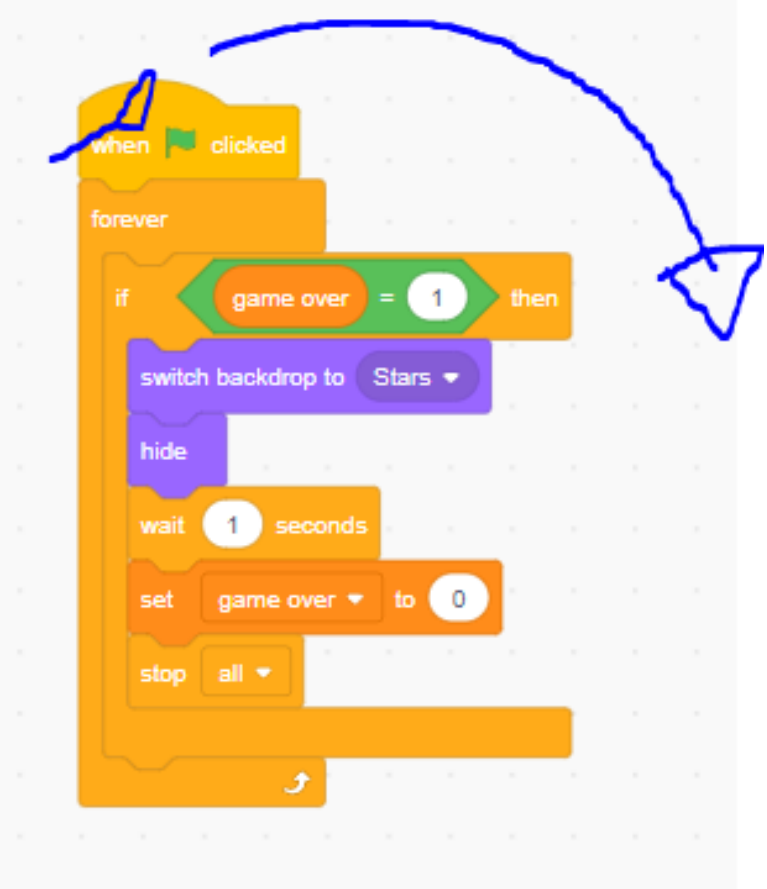


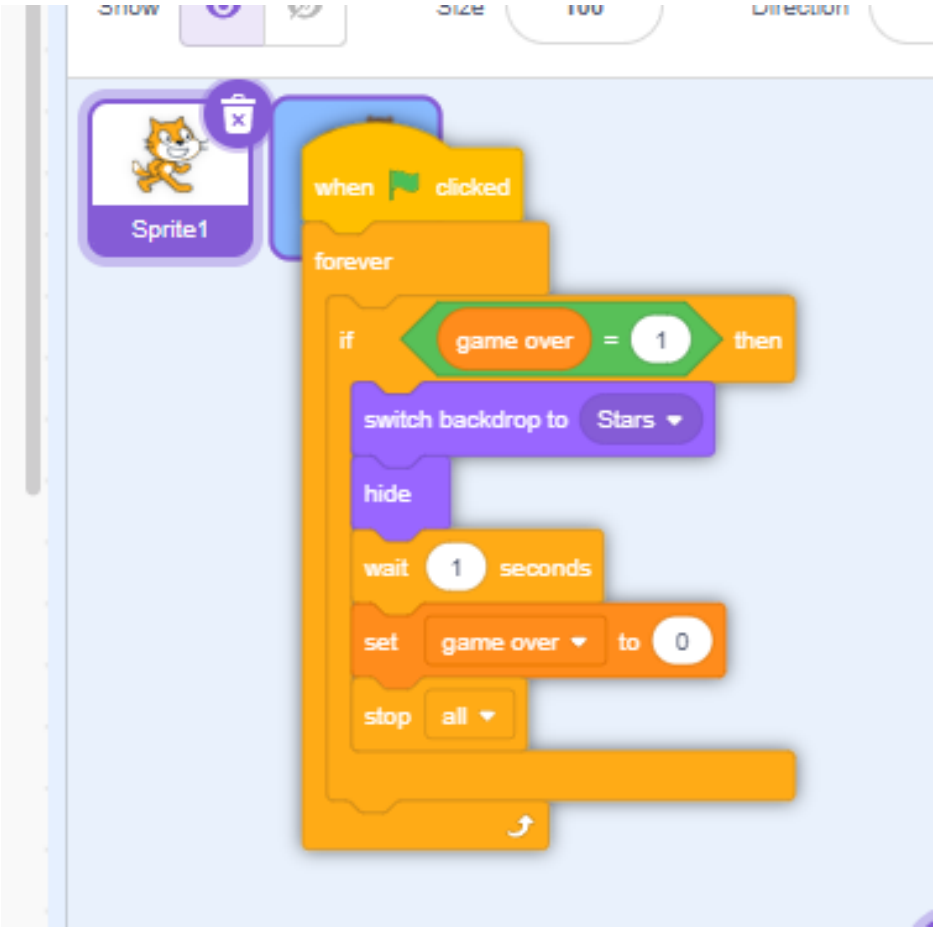
Then, we have to stop the game. We'll first make sure that all the sprites have time to react to "game over" being true, and to do this, we'll simply put a "wait 1 seconds" block from **Control** right underneath our "hide" block. Then, we'll reset our game-over variable to 0, using the **Variables** "set ____ to", setting the values to be "game over" and "0". This would mean that "game over" is no longer true, and ensures that the next game won't start by entering into the game-over screen.

Finally, to stop the game, we can add in a "stop all" block from the **Control** tab.

Its important to note that you could add more blocks to this "game over" chunk of blocks. If you had a "score" variable keeping track of how far the player is into the game, you could set it to 0 at this point, resetting it for next round.

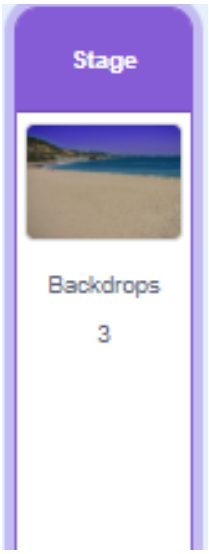
Also, if we want the secondary sprite to disappear as well, we can click on our whole "when flag clicked" chunk of blocks, and drag it into the sprite, which will duplicate the code over. This will make sure that the secondary sprite will disappear as well. Then, simply make sure we undo this at the beginning by adding a "show" block in front of the "forever" block.

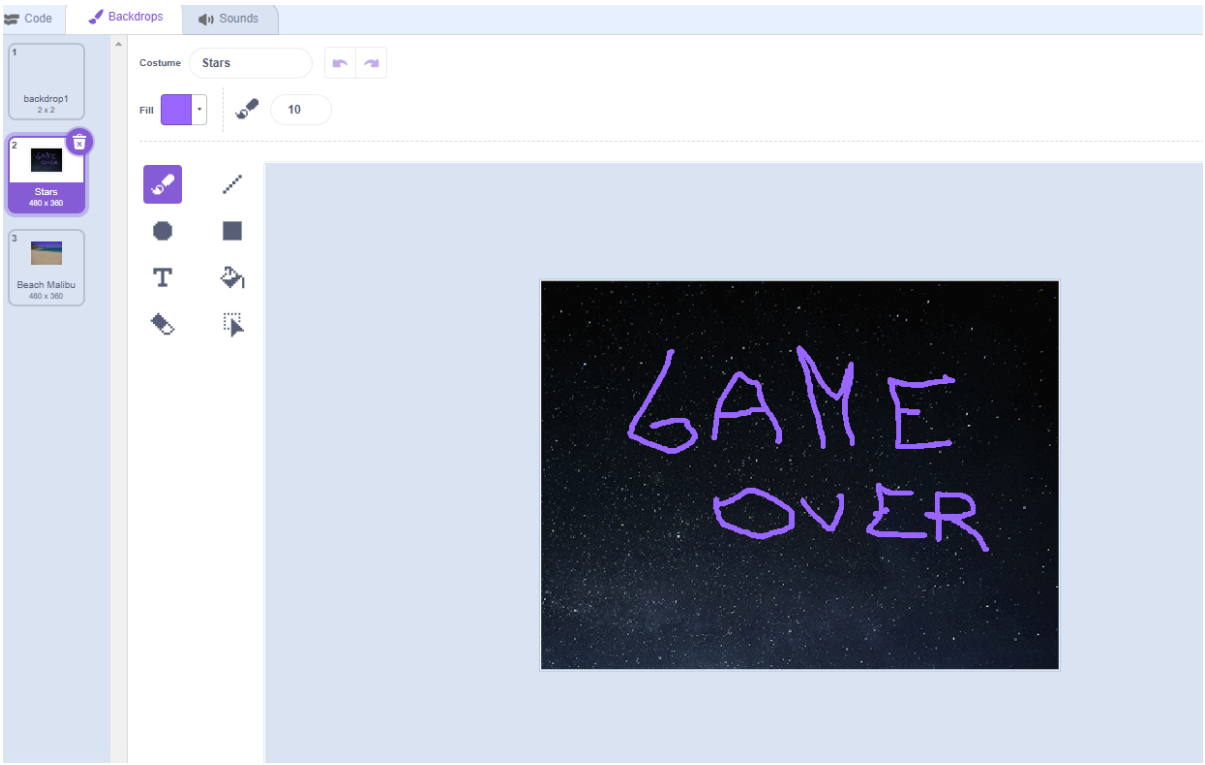

when clicked
forever
if game over = 1 then
switch backdrop to Stars
hide
wait 1 seconds
set game over to 0
stop all



For the complete code, use the project link at the end of this lesson.

As a final teaching for this lesson, we learned that we can edit backdrops the same way as sprites. To do this, simply select the backdrop, and then click "Backdrops" with the paintbrush tool. This will allow you to write on the backdrop. I wrote out "GAME OVER" on mine.

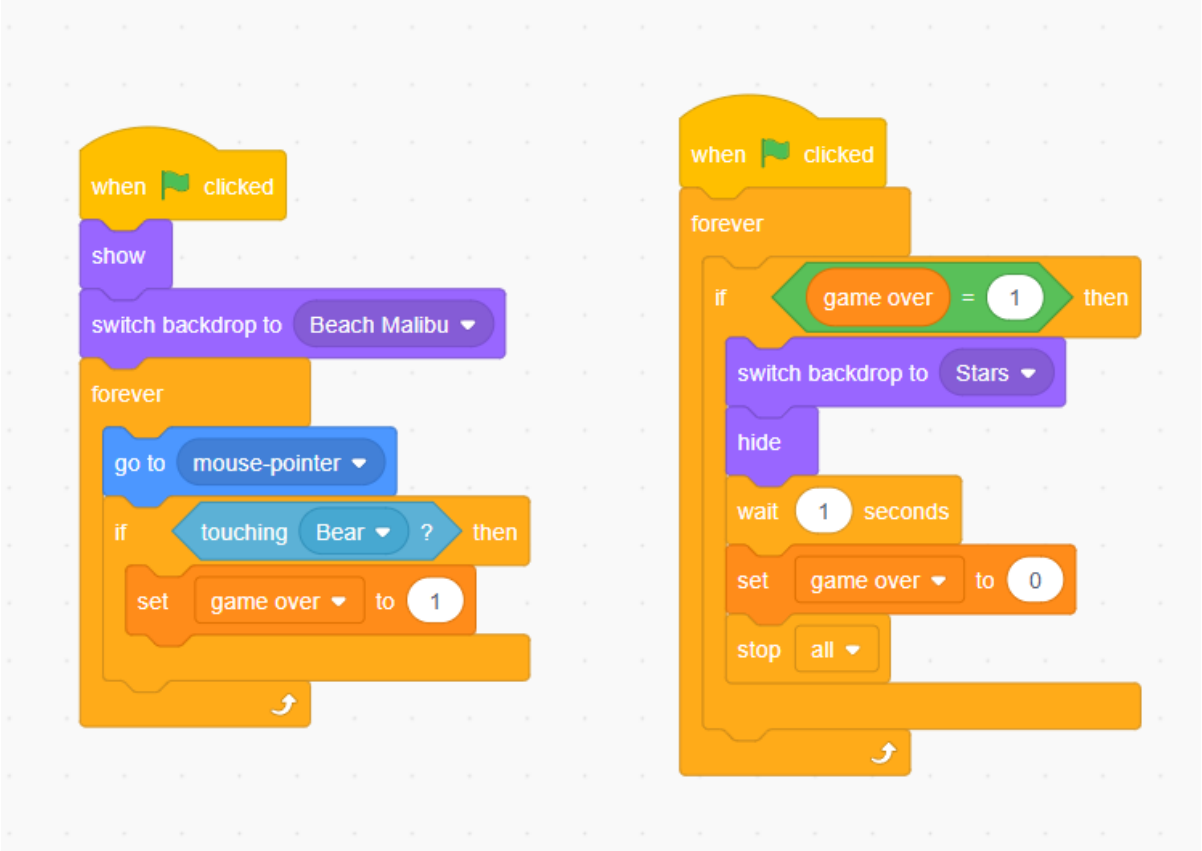




Try applying this code to another project. When restarting the game, reset the changes made by the game-over screen. If something does not work, compare your blocks with the examples and test one change at a time.

Heres this project's code:

Main Sprite:



Second Sprite:

For the complete code, use the project link below.

Project link:

https://scratch.mit.edu/projects/1098436588

Continue at your own pace.

Lesson 8

Smiling Creek: Coding & Design with Scratch - Lesson 8

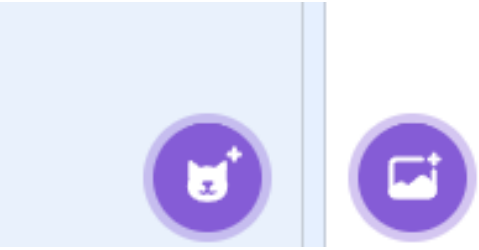
Lesson overview

In this lesson we created a replica of pong.

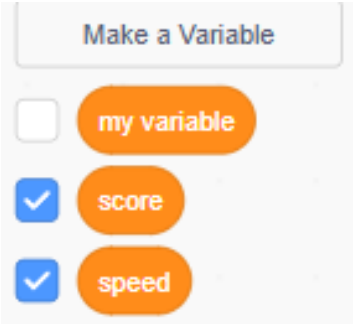
We practiced general block-building, and the skill of creating a game in a shorter period of time: this was definitely the longest lesson yet.

We combined ideas learned before, using lots of variables, and then also learning the "mouse y" button and concepts involving mouse position.

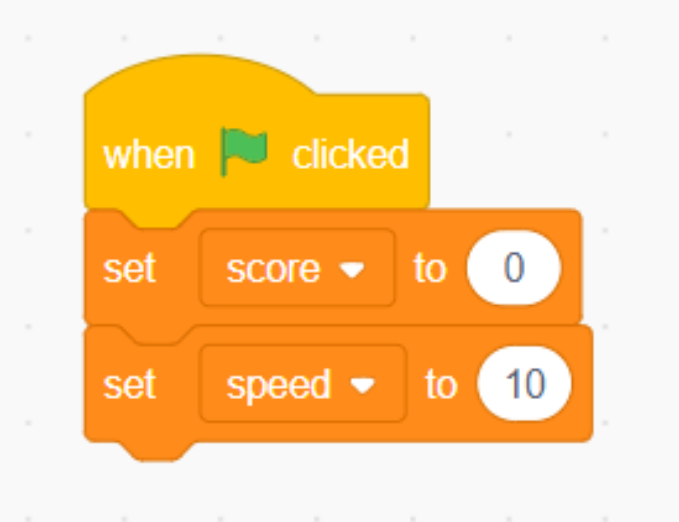
To begin, we chose a background, and then also a sprite to use as the ball. Students were allowed to pick an alternative sprite to be their pong 'paddle' if they wanted, but for the lesson, we simply used the default cat.



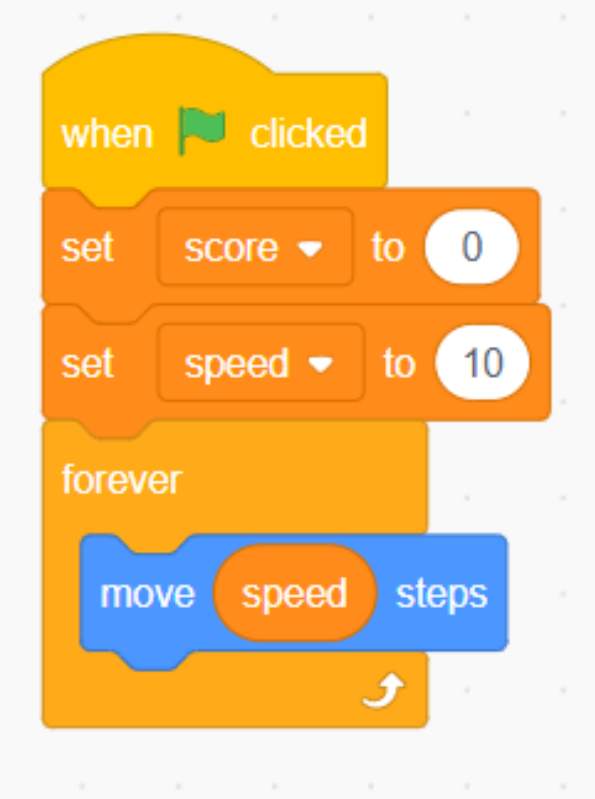
In this lesson we started by creating our two variables: score, and speed. These variables will control the speed of the ball and the score of the player.



To begin, we made sure we were on our Ball sprite, dragged in an **Events** "when flag clicked" block, and underneath, placed two **Variables** "set ____ to __" blocks. Then, we changed the blocks to read "set score to 0", and "set speed to 10". Throughout the game, our score and speed are going to go up, so these blocks will ensure that at the beginning of every game, the score and speed are reset.



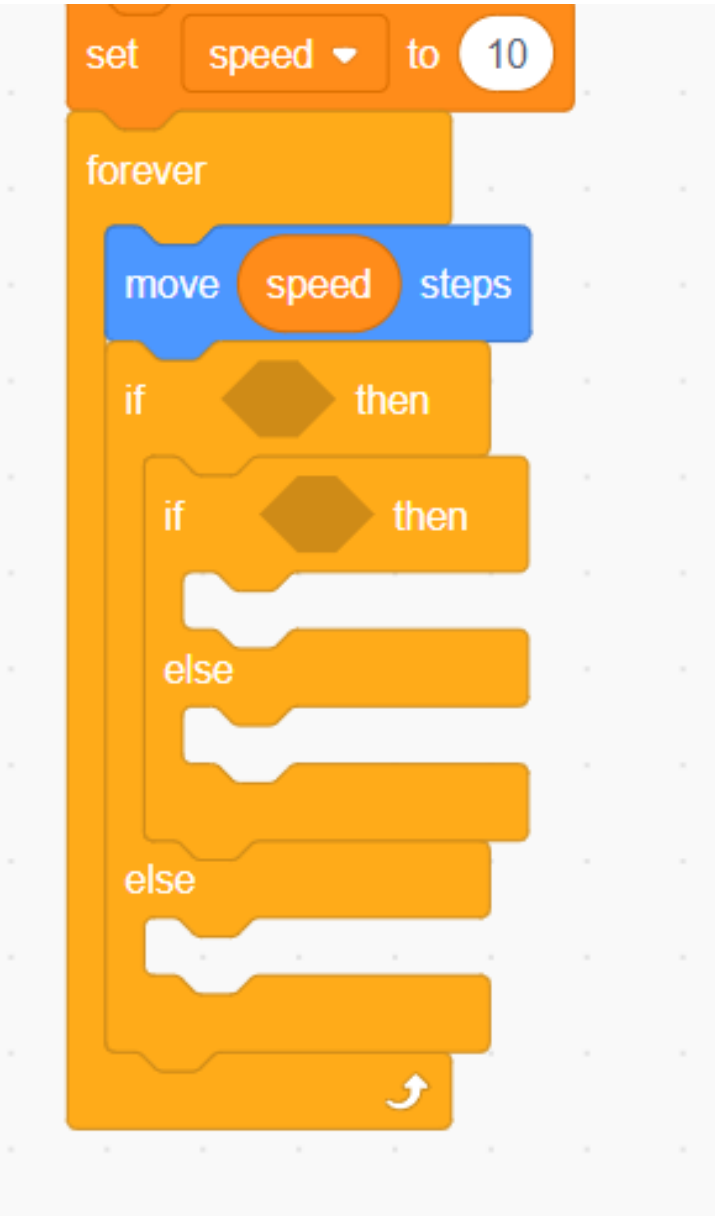
Next, we dragged in a **Control** "forever" block, and a **Motion** "move __ steps" block underneath, and dragged in our "speed" block from Variables into the slot. This will ensure that we can control the speed of the ball.



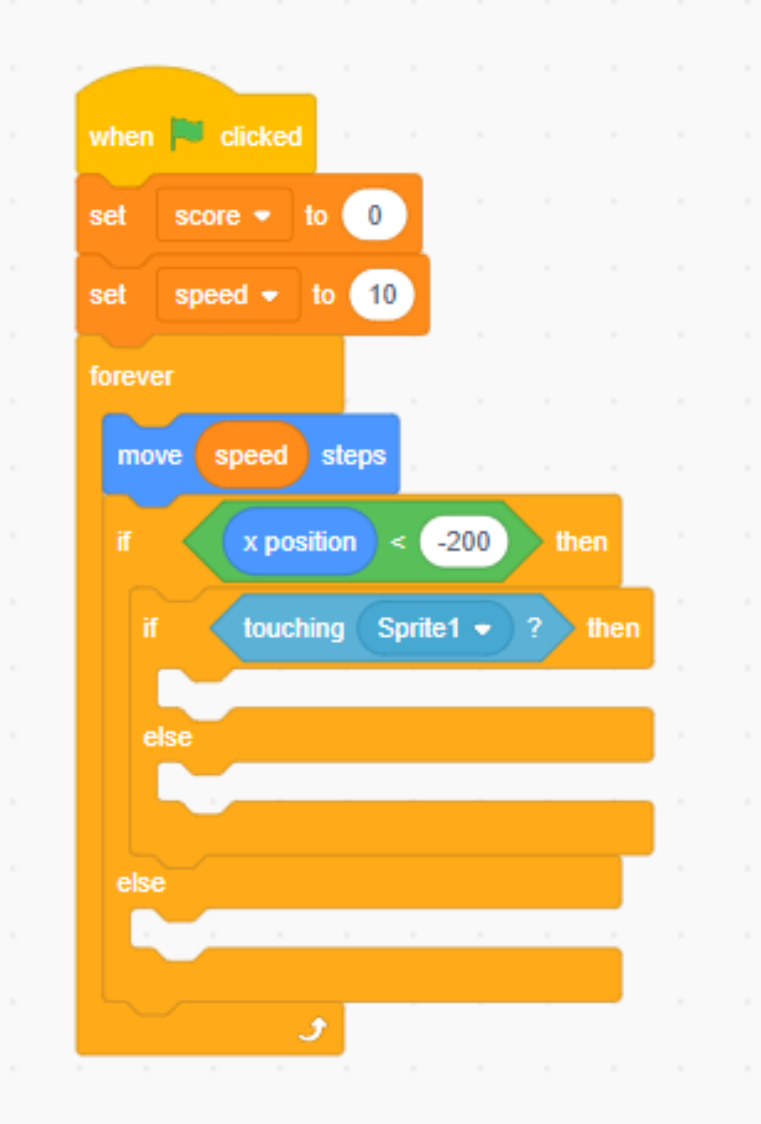
After that, we dragged in two **Control** "if ___ then, else" blocks. These will be used to check the following things:

If the ball is on the left edge, we'll check to see if its hitting the paddle sprite, and if so, we'll add a point, speed it up, and bounce it off. If not, we'll end the game.

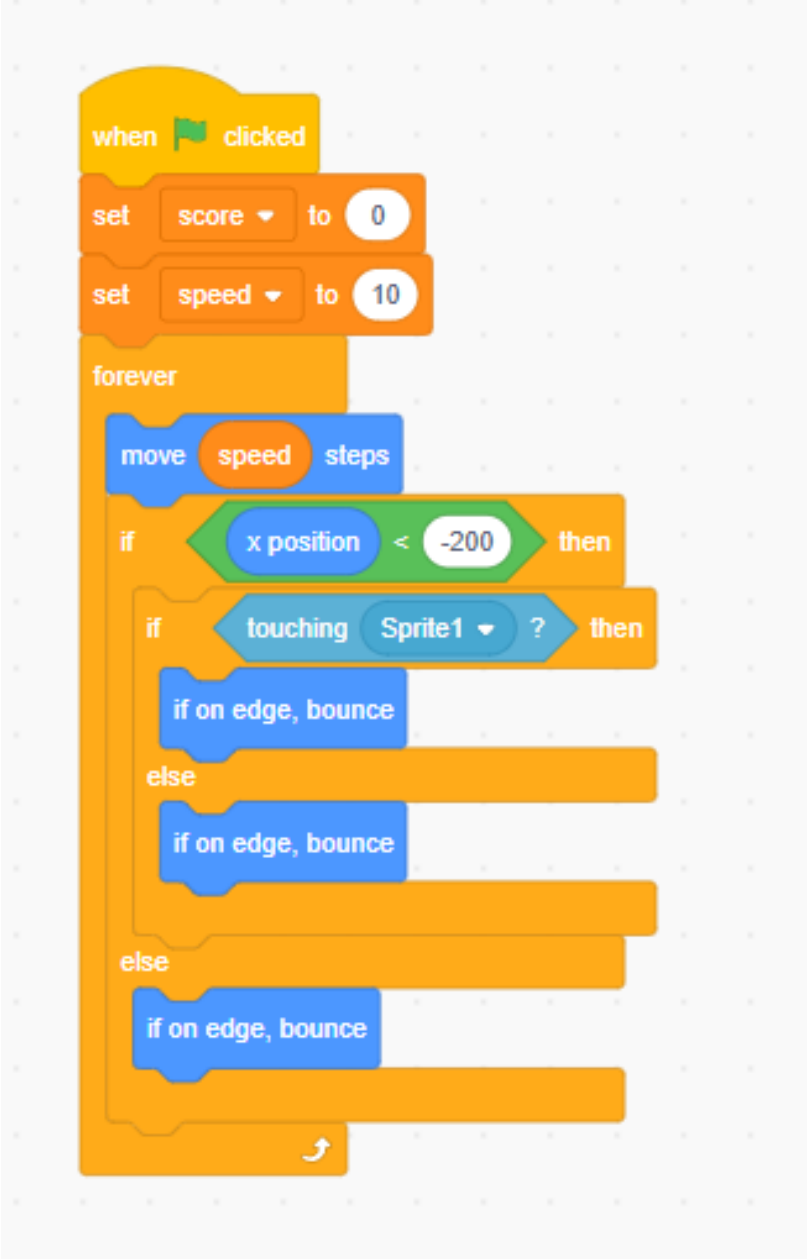
If the ball isn't on the left edge, we're going to just bounce it off anything it touches (the other walls).



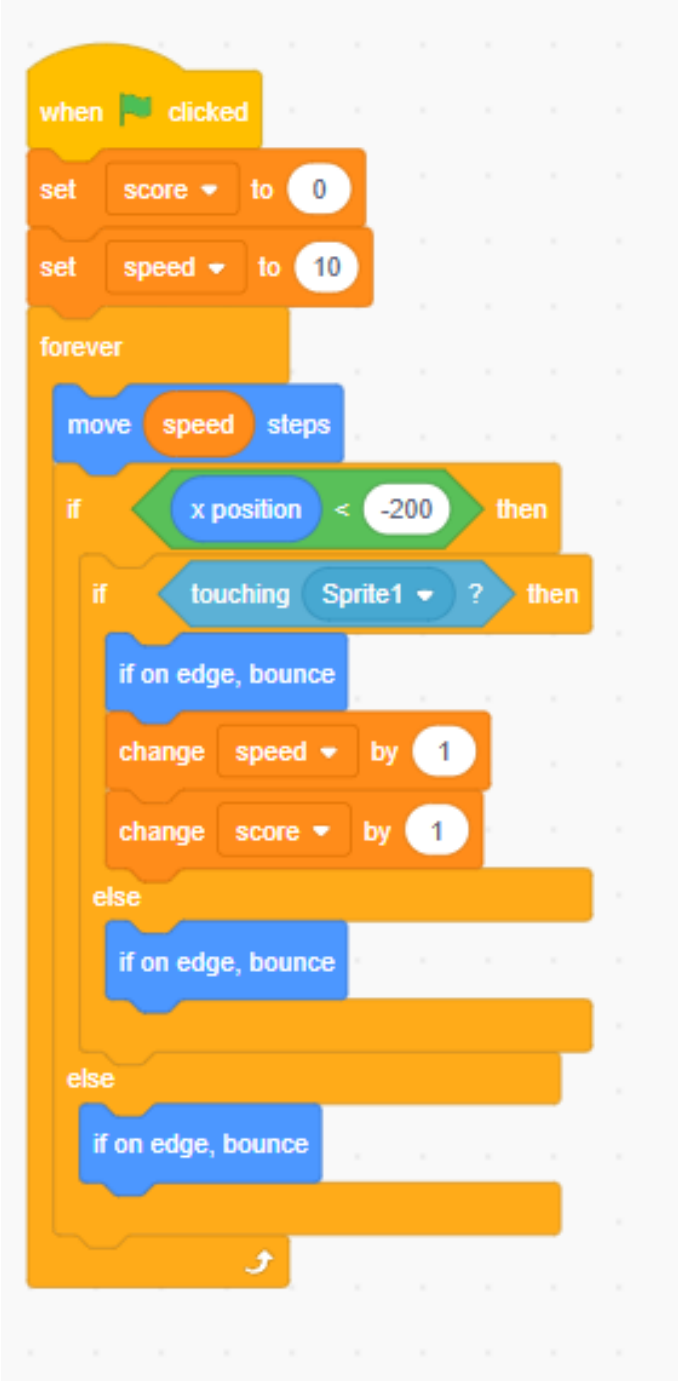
To build this, we'll first drag in the **Operators** "(___ < ___)" block to the first "if" block, and then drag in the "x position" block from **Motion**, and set the value to be -200. This will check if we're hitting the left wall. Then, drag in a **Sensing** "touching ___" block, and set the value to be Sprite1 (or the other sprite you chose for the ball to bounce off).



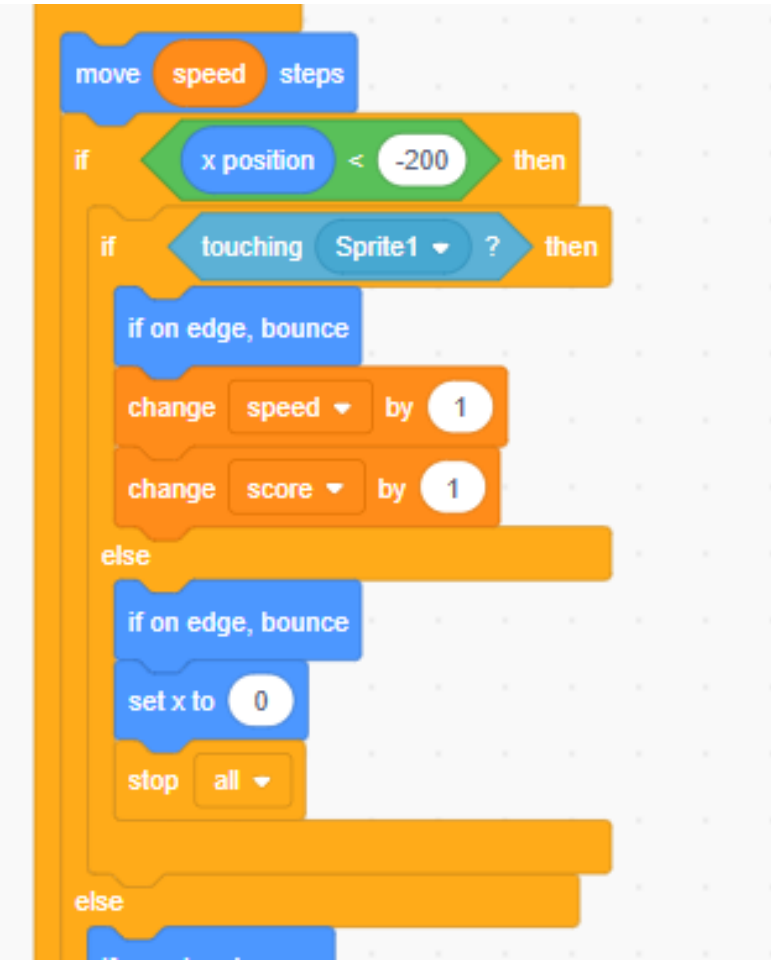
Then, lets drag in "if on edge, bounce" blocks into every open space. This will ensure that no matter what, the ball will bounce. This is important even when we lose the game because we want to make sure that the ball starts the next game facing right and not left.



Underneath the first "if on edge, bounce", go to the variables tab, and drag in 2 **Variables** "change ___ by __" blocks. Set the values to be "speed" and "score", and then the numbers to be 1 for both. This will speed up the ball and up the score when we successfully return the ball.



Underneath the next "if on edge, bounce", drag in a "set x to __" block from the **Motion** tab, and set the value to be 0. Then, underneath, drag in a "stop all" block from **Control**. This is the code that will run if we lose the game, so we'll reset rotation, set the ball back to the middle, and stop the game.

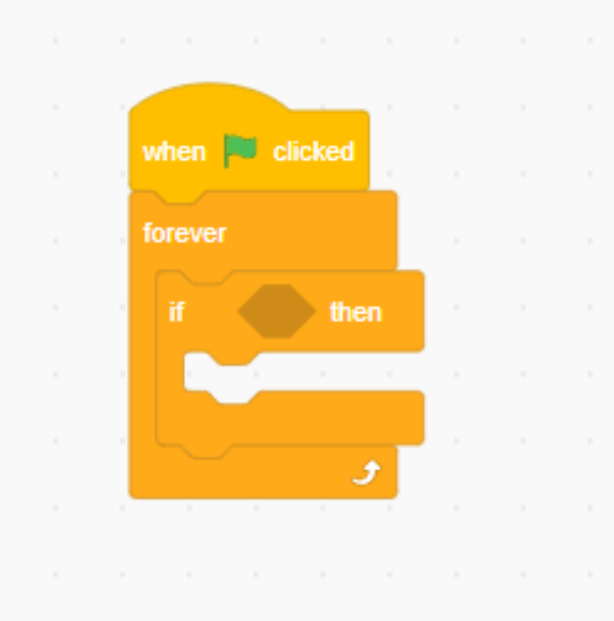


The code is now complete for the ball!

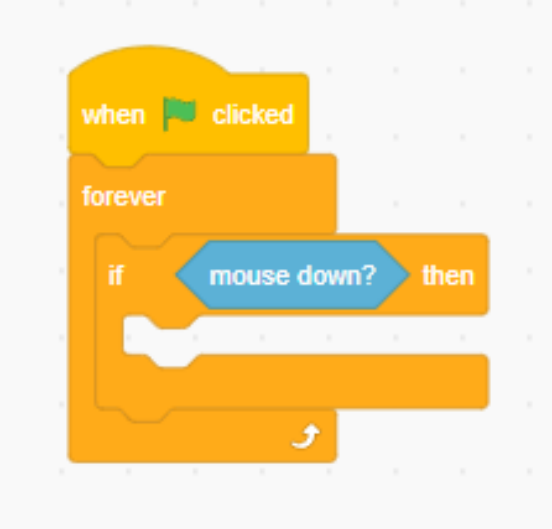
Finally, all we have to do is program the motion for the character.

Make sure you're on the character's code space now.

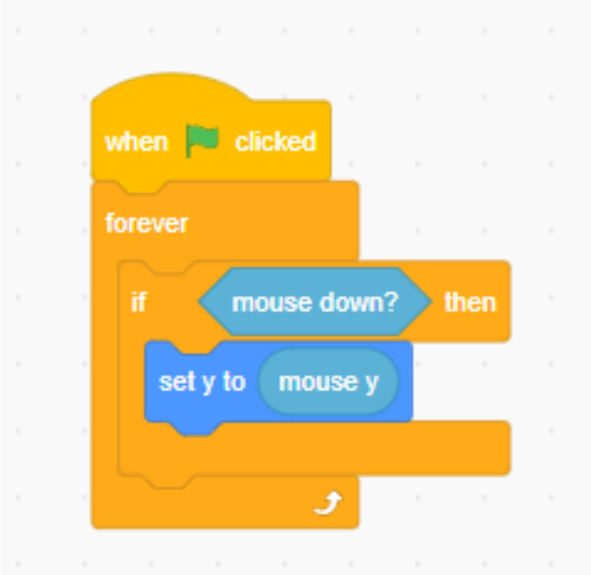
Drag in a **Events** "when flag clicked", underneath that, a **Control** "forever" block, and then underneath that, an "if" block. We're going to use this to set the vertical position of the Sprite to match the vertical position of the mouse.



Next, drag in a "mouse down ?" block from the **Sensing** tab.



Finally, drag in a "set y to" block from the **Motion** tab, and then drag in a "mouse y" block from the **Sensing** tab. This "mouse y" block will give us the mouse's vertical position.

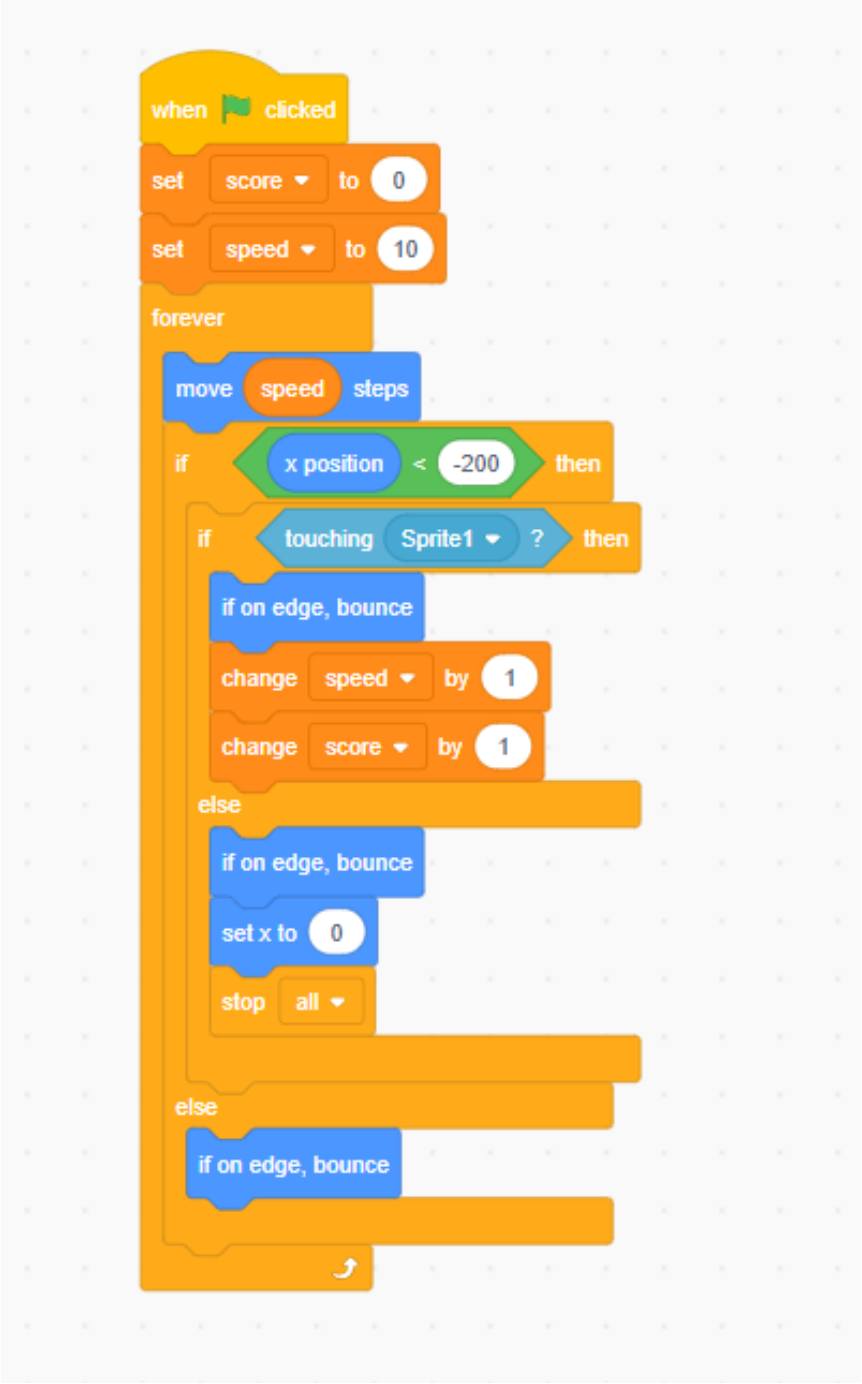


Now, the code is complete!

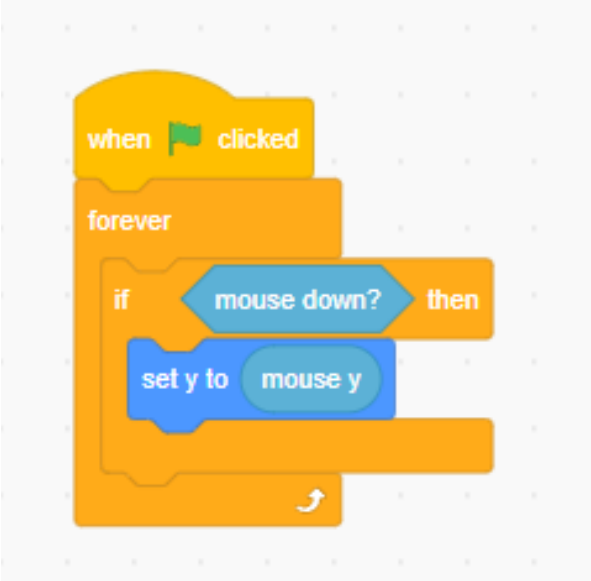
Note that if you're using the mouse, you'll have to enter full screen to play the game correctly, or leave the editor and just open the project link below.

All the code:

For the ball:



For the sprite:



Project link:

https://scratch.mit.edu/projects/1102007496

Save your project and test the code before continuing.

Lesson 9

Smiling Creek: Coding & Design with Scratch - Lesson 9

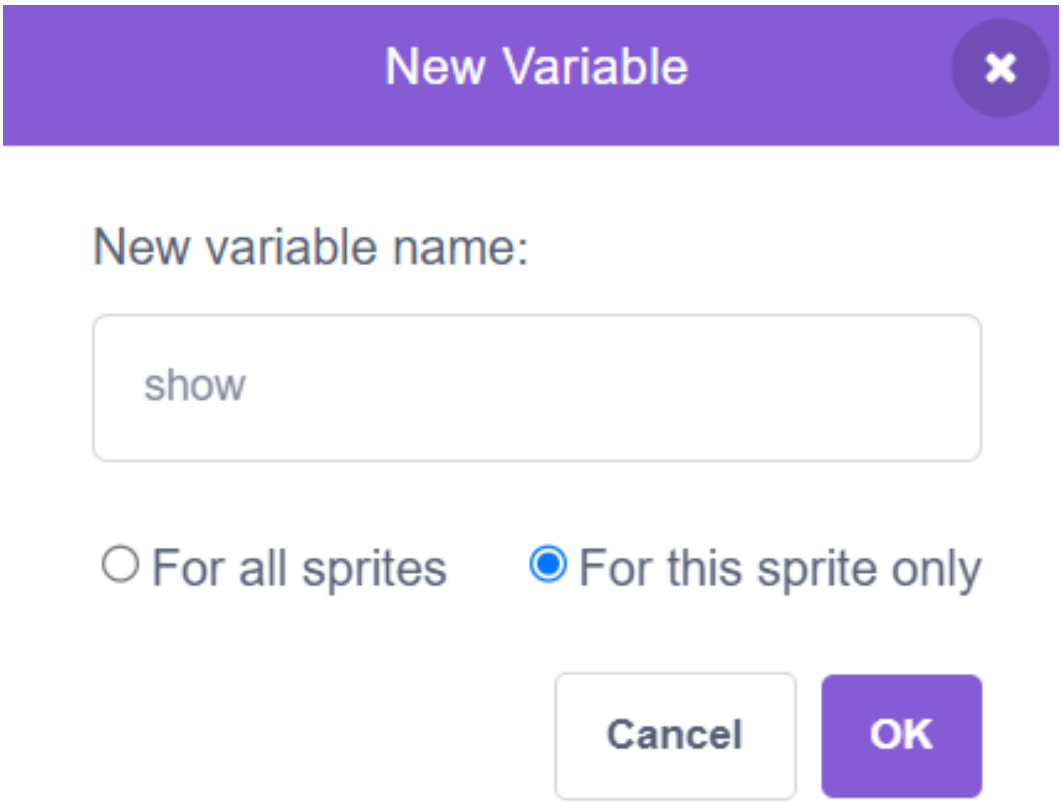
Lesson overview

In this lesson we created a whack-a-mole game in Scratch.

We practiced using the random function, as well as gaining a better understanding of the show and hide blocks. We also improved our understanding of using "Boolean" style of variables in scratch, meaning variables that are either 1 or 0. Finally, we learned the use of local variables, which are variables where you select "for this sprite only" when creating, and also learned coding in the backdrop code space.

To begin, we needed to create two variables: "Score", which will be universal, and "show", a variable which will be on each individual "mole", so that they can all pop up at different times.

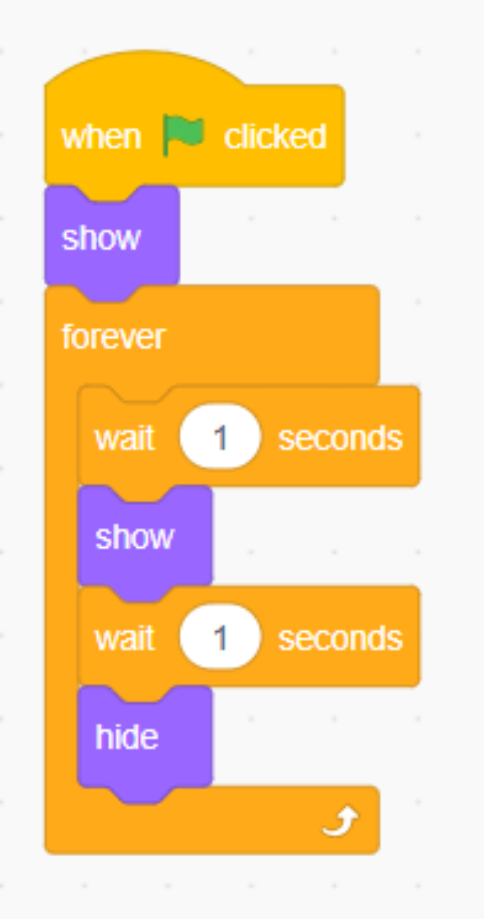
Make sure that when creating these variables, that for "score" you select "for all sprites", and for "show", you select "for this sprite only".



Next, we want to make sure we only have one sprite in the scene, and this sprite can be anything you want as your "whack-a-moles". Its also important to select a backdrop.

Then, open the sprite coding space. To begin, drag in a "when flag clicked" **Events** block, and then underneath, a **Control** forever block. Then, drag two "wait __ seconds" blocks.

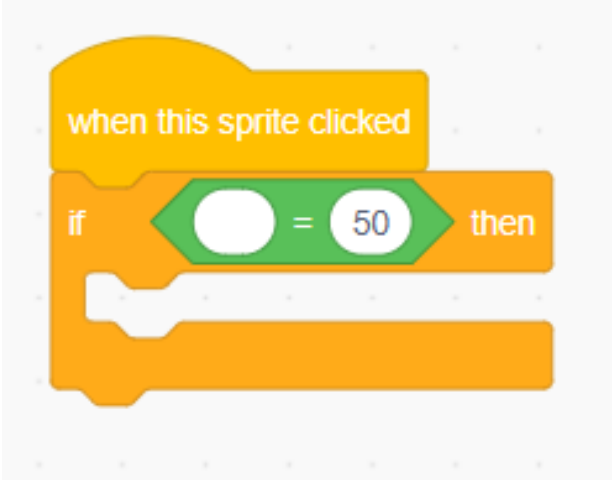
After that, go over to the **Looks** tab, and drag in a "show" block right after the flag, a "show" block right after the first "wait __ seconds", and then a "hide" block right after the second "wait __ seconds" block.



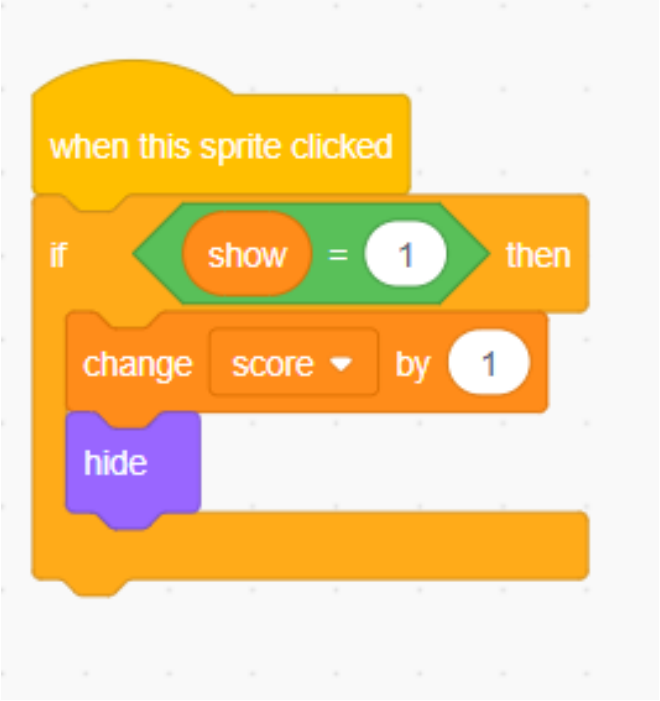
After this, go over to the **Variables** tab, and find the "set ___ to __" block. Drag in two of these, both after the "show" and "hide" blocks just dragged in. Make sure the variables are "show" and the values are "1", and "0".

Finally, go over to the **Sensing** tab, and drag in two "pick random __ to __" blocks, each into the "wait __ seconds" blocks. Then, set the values of the first block to be 1 and 5, and the second block to be 0.5 and 1. These random values will determine how long the "mole" pops up for and stays hiding for.

Next, creating a new chunk of code, drag in the **Events** "when this sprite clicked" button. Underneath, place a "if ___" block from the **Control** tab. After that, drag in an **Operators** tab block that looks like "(__=__)".

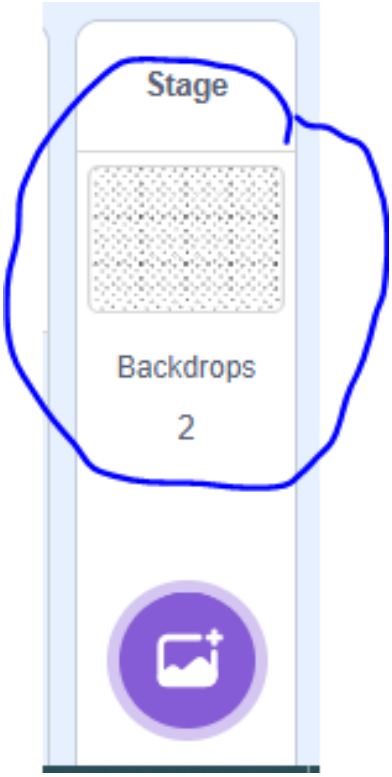


After this, go over to the **Variables** tab, and drag in "show" into the first space in the operator. In the second space, type "1". Underneath this, while still in the **Variables** tab, drag in a "change ___ by __" block, and set the values to "score" and "1". Finally, go over to the **Looks** tab, and drag in the "hide" block, placing it underneath this "change" block.

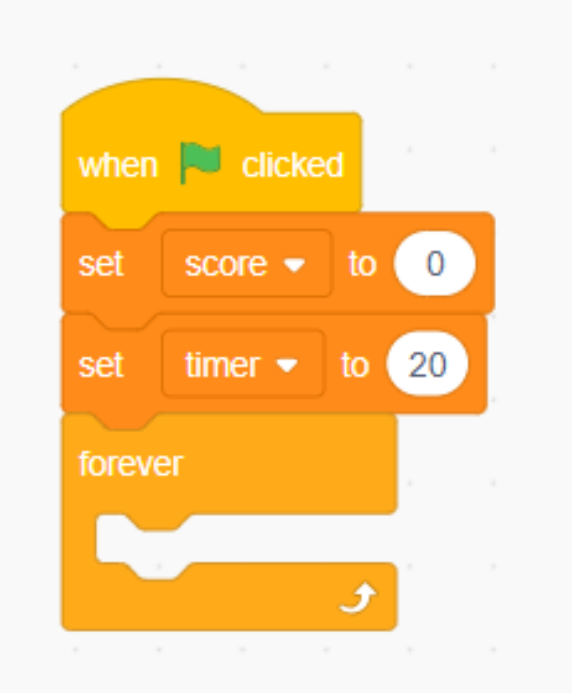


Your code for the mole is now completed!

Next, go to the bottom right of the screen, and select the "backdrop" itself. Not the purple button, but the backdrop itself. This will open up a new coding space, where we'll put a timer. Lets begin by going over to the Variables tab, and creating a variable named "timer", which will automatically be available for all sprites.

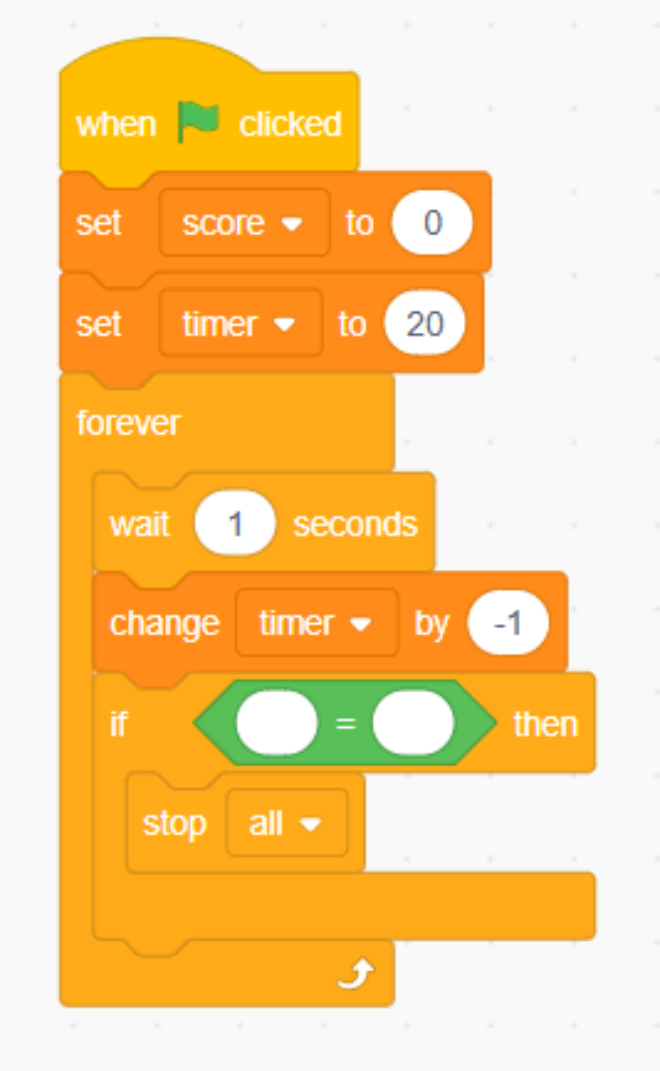


To begin, lets drag in a "when flag clicked", and then underneath, a "forever" block. Underneath the "flag clicked", go to **Variables**, and drag in two "set __ to __" blocks. These should be "score" to "0", and "timer" to "30". These will make sure that the timer resets at the beginning of the game, and that score starts at 0.



Then, drag in a **Control** "forever" block, and then underneath, a "wait __ seconds", and a "if ____ then" block. In between these last two, drag in a **Variables** "change __ by __" block, making the values "timer" and "-1".

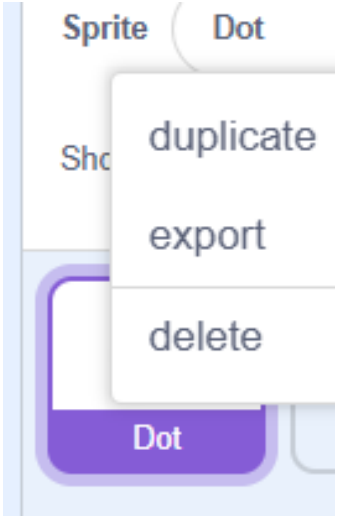
Then, head over to the **Operators** tab, and drag in the same block from earlier into the "if" space, the one that looks like "(__=__)". Underneath this block, go over to the **Control** tab, and drag in a "stop all" block.



Finally, drag in the "timer" variable into the first operator gap, and then type a "0" into the second space.

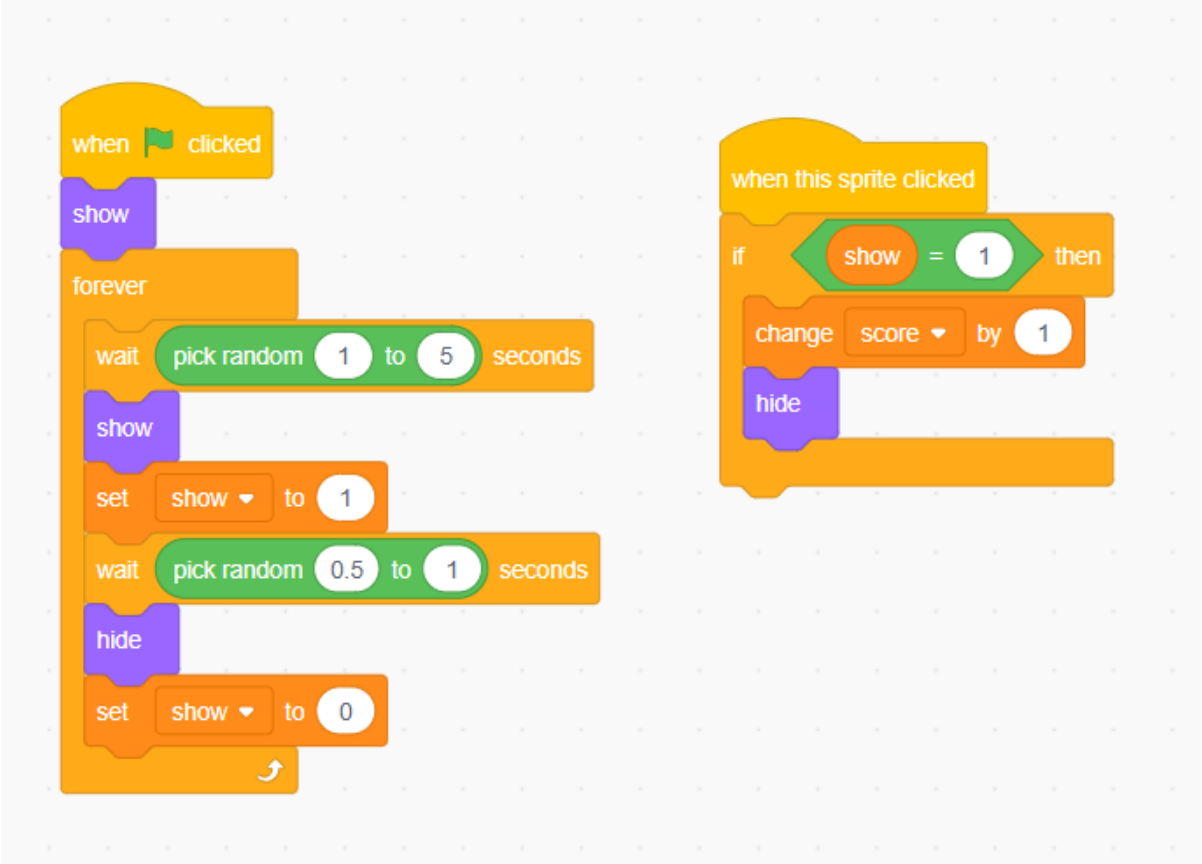
You're now done the code! But, there are a few more extra steps:

First, right click on the sprite, and select "duplicate". This will allow you to create multiple moles which you can space out in the scene.



If you're having trouble changing the position of the sprites, you can either click the flag button and stop button really quickly, and this will make it so that all sprites are visible, or you can change the position directly through the position tab in the sprite.

Code for "mole":



Code for backdrop:

Project Link:

https://scratch.mit.edu/projects/1105106779

Save your project and test the code before continuing.

Lesson 10

Smiling Creek: Coding & Design with Scratch - Lesson 10

Lesson overview

Complete this lesson at your own pace.

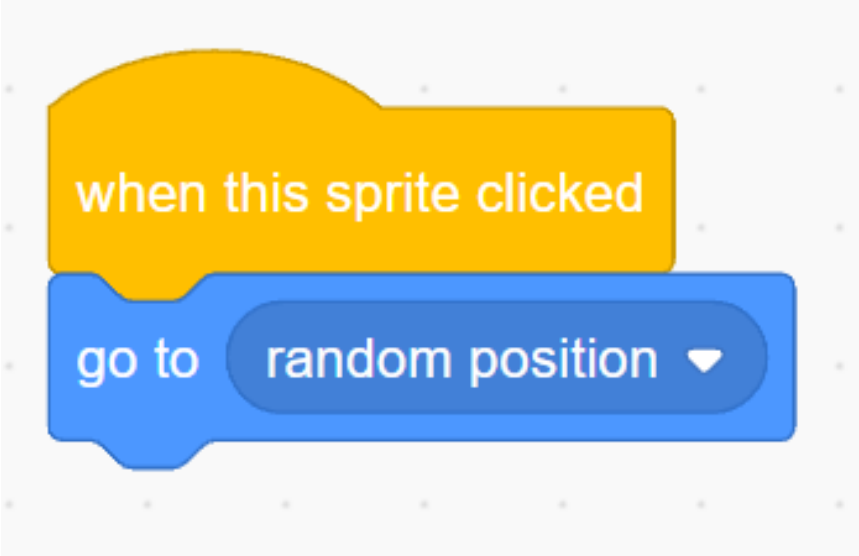
Find the complete code and project link at the end of this lesson.

Follow the steps below to build the project.

This lesson, we got more comfortable with Scratch's workspace, and introduced a few new concepts and ideas.

First, we started off by creating a new sprite, something that we could click on. I chose a star, but students could choose whichever sprite they wanted.

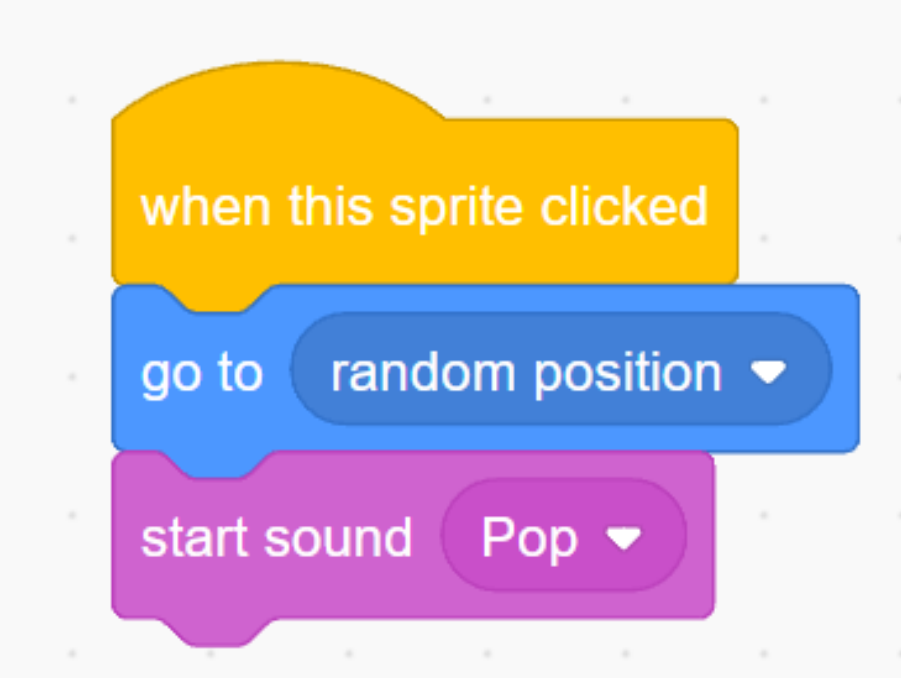
After this, we learned the "when this sprite clicked" block, and we combined this with the, "go to 'random position'" block to create a sprite that jumps around when it is clicked.



Remember that all **blue blocks** have to do with **motion**, and all **yellow blocks** have to do with **control**.

After this, we learned how to play different sounds in Scratch. **Sounds** are a key part in our coding, and we will explore different options for sound effects in later lessons.

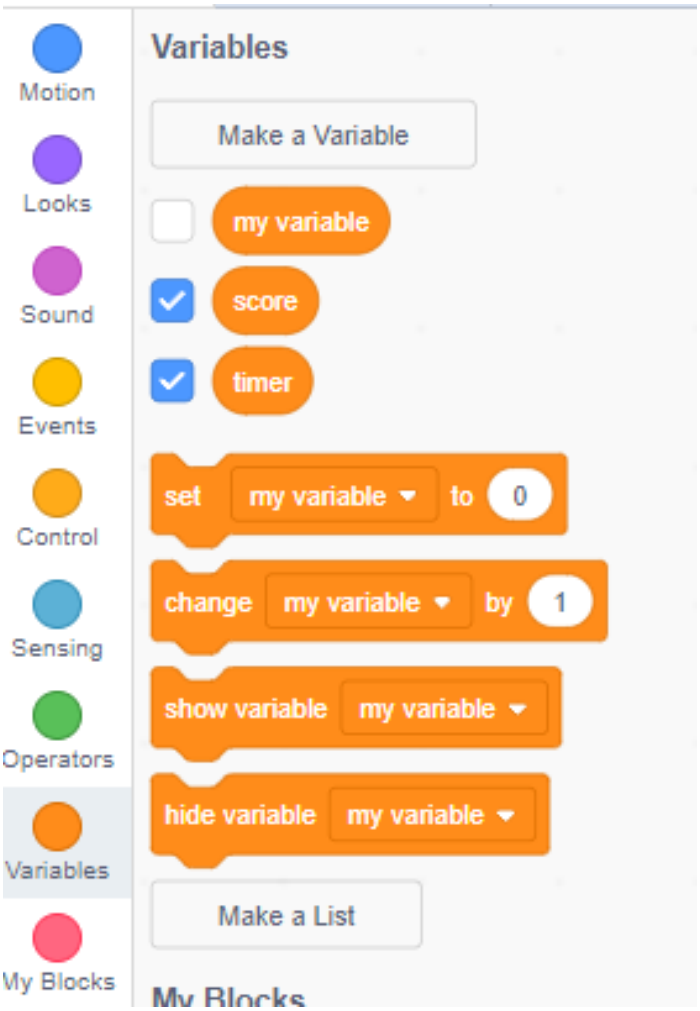
We used the "start sound" block after this code to create a sound queue for clicking our sprite.



Once we were done this, we introduced the most important concept we will learn in this coding course, variables.

Variables are used in every aspect of coding, and for this lesson, we learned how to create, identify, and control variables.

To do this, we went to the variables section of blocks, selected "**Make a variable**", and then entered the name of our variable.

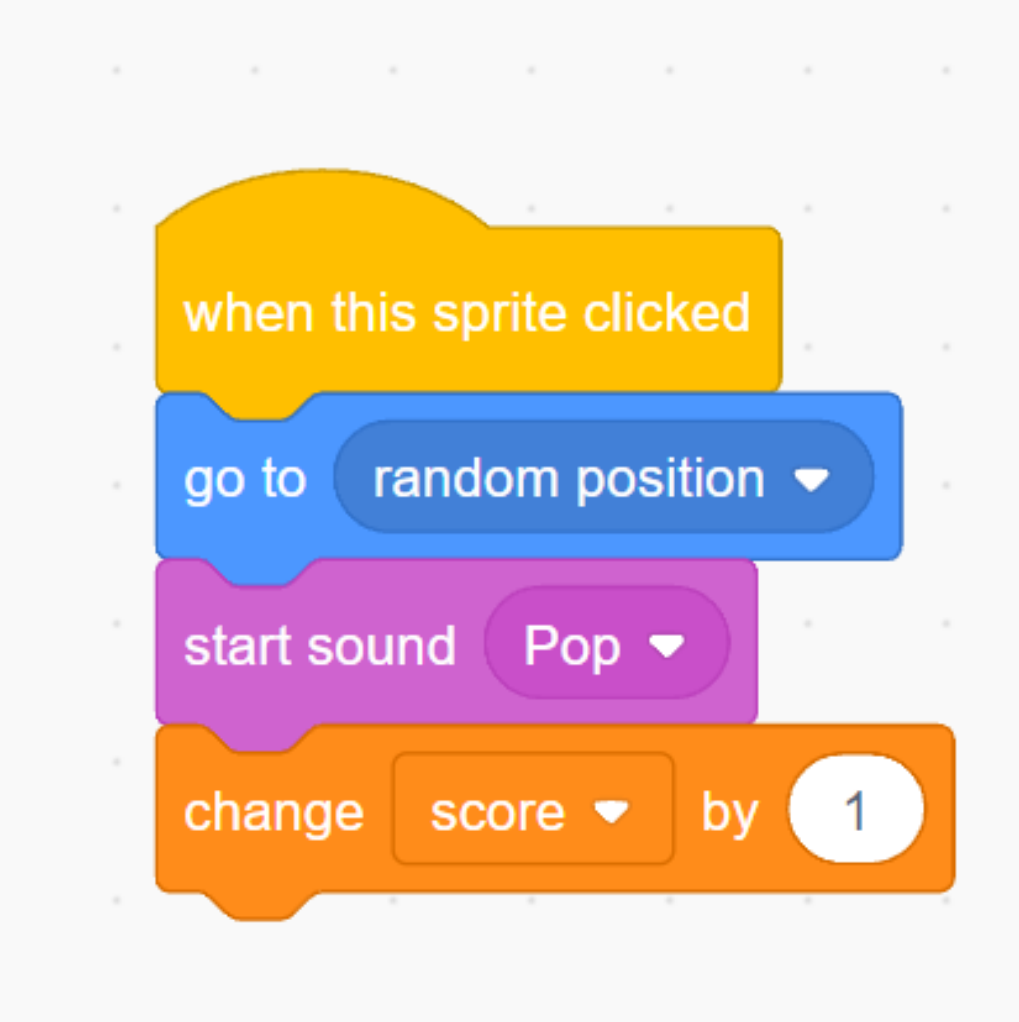


Variables allow us to keep track of different parts of the game, and also to control our game.

For this project, we created two variables: **score**, and **timer**.

First, we used **score** to keep track of how many times our sprite was clicked.

To do this, we selected the "change _____ by 1" block, selected the dropdown box, and chose score.



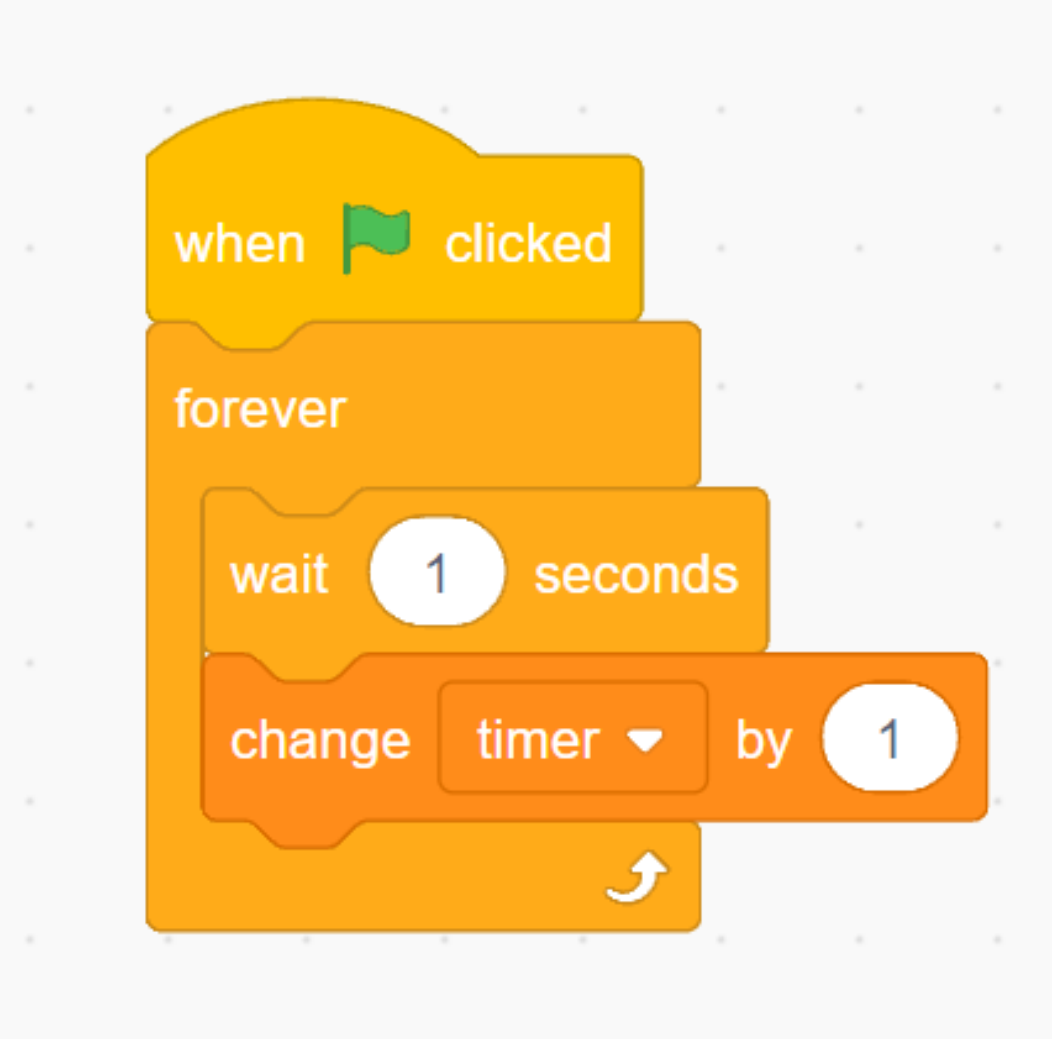
Now we can keep score of our clicks, but how do we make it a game?

To do this, we used our **timer** variable.

We made it so that when the flag is clicked, starting the game, we have a timer that is constantly going up.

To do this, we used a **forever loop**, which is a block that repeats whatever is inside it forever.

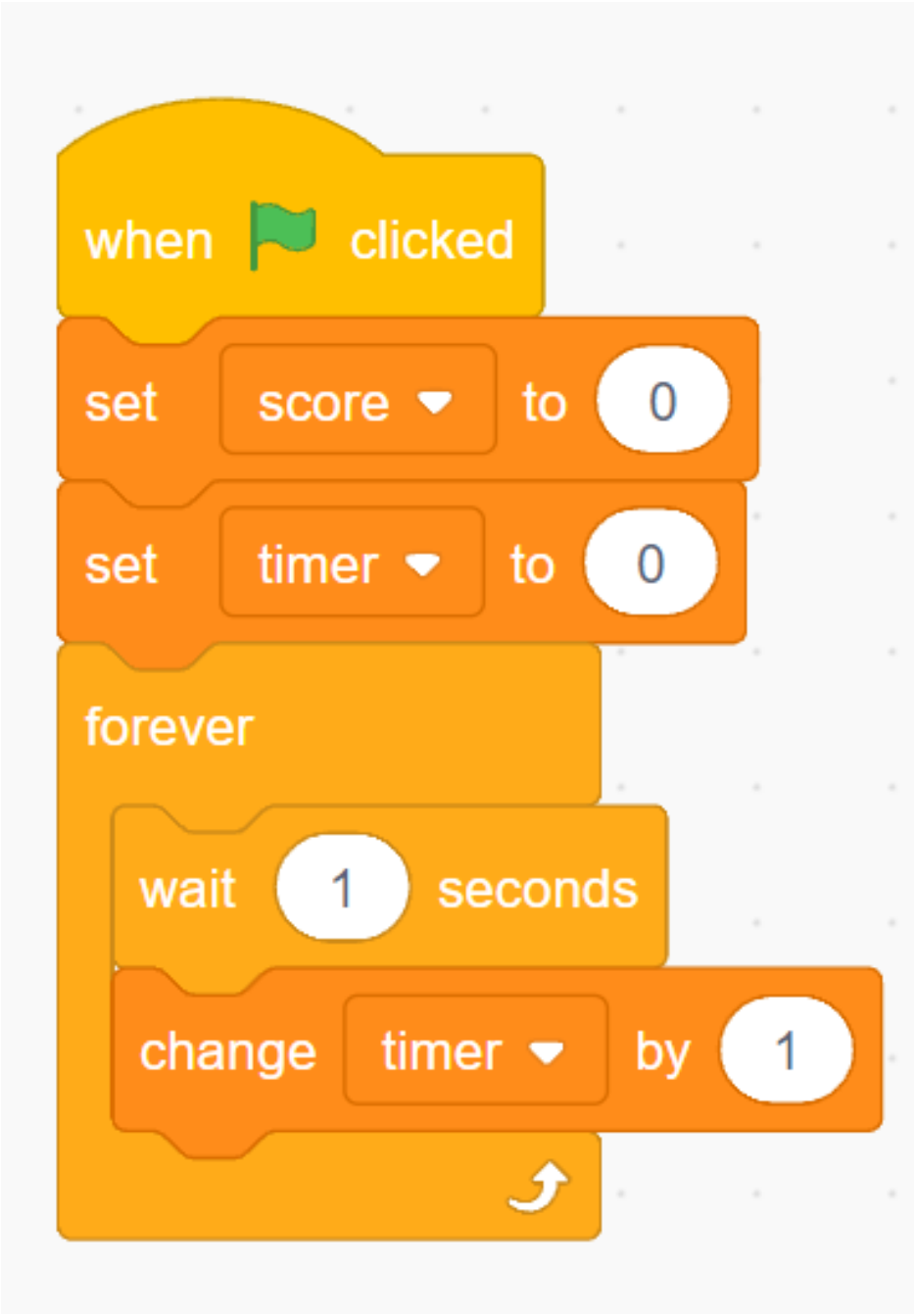
Inside, we placed a "wait 1 second" block, and then a "change _____ by 1" block, selecting timer in the blank.



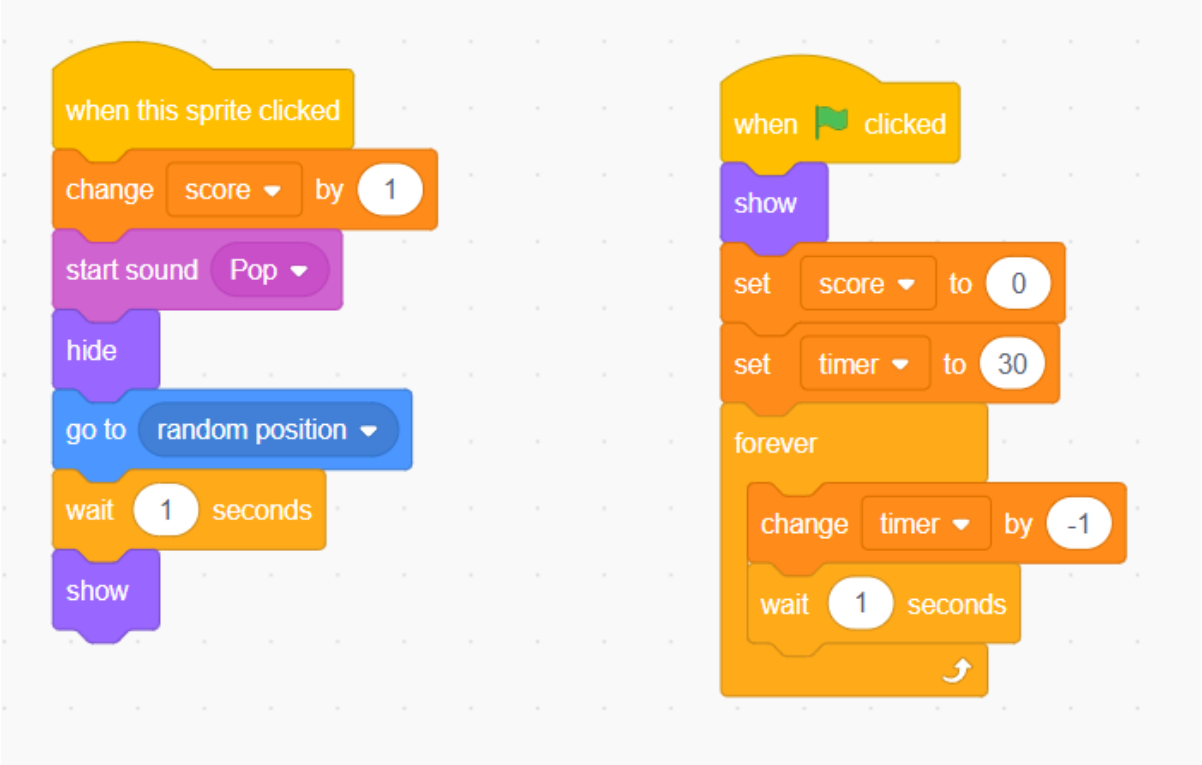
This makes it so that forever, we wait one second, increase timer by one, and keep repeating. This lets us keep track of how long its been since we started the game.

Then, to reset the timer and score when we start the game, we made it so that every time we start the game, before we start adding up our timer, we reset score and time to 0.

To do this, we used the "set _____ to" block, and filled it in so that we set our score and timer to 0.

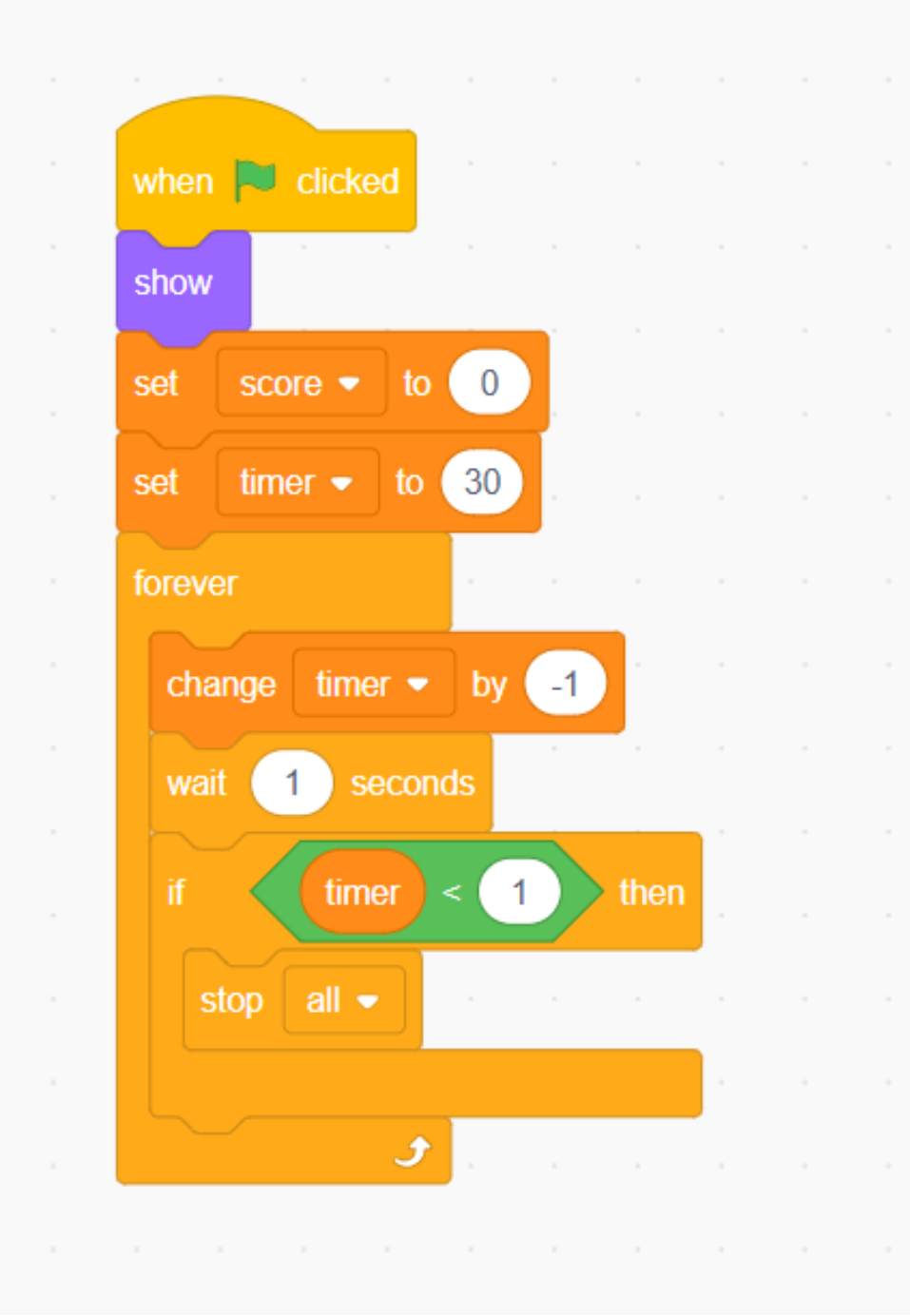


After this, we need to go back to our original chunk of code and drag in three blocks: 1 "hide" and 2 "show" blocks. We'll put the hide block right after the "start sound" block, and the "show" blocks at the end of the first chunk of code and at the beginning of the "when flag clicked" chunk of code. We'll also change the values for our "set timer to 30" block and "change timer by -1" block.

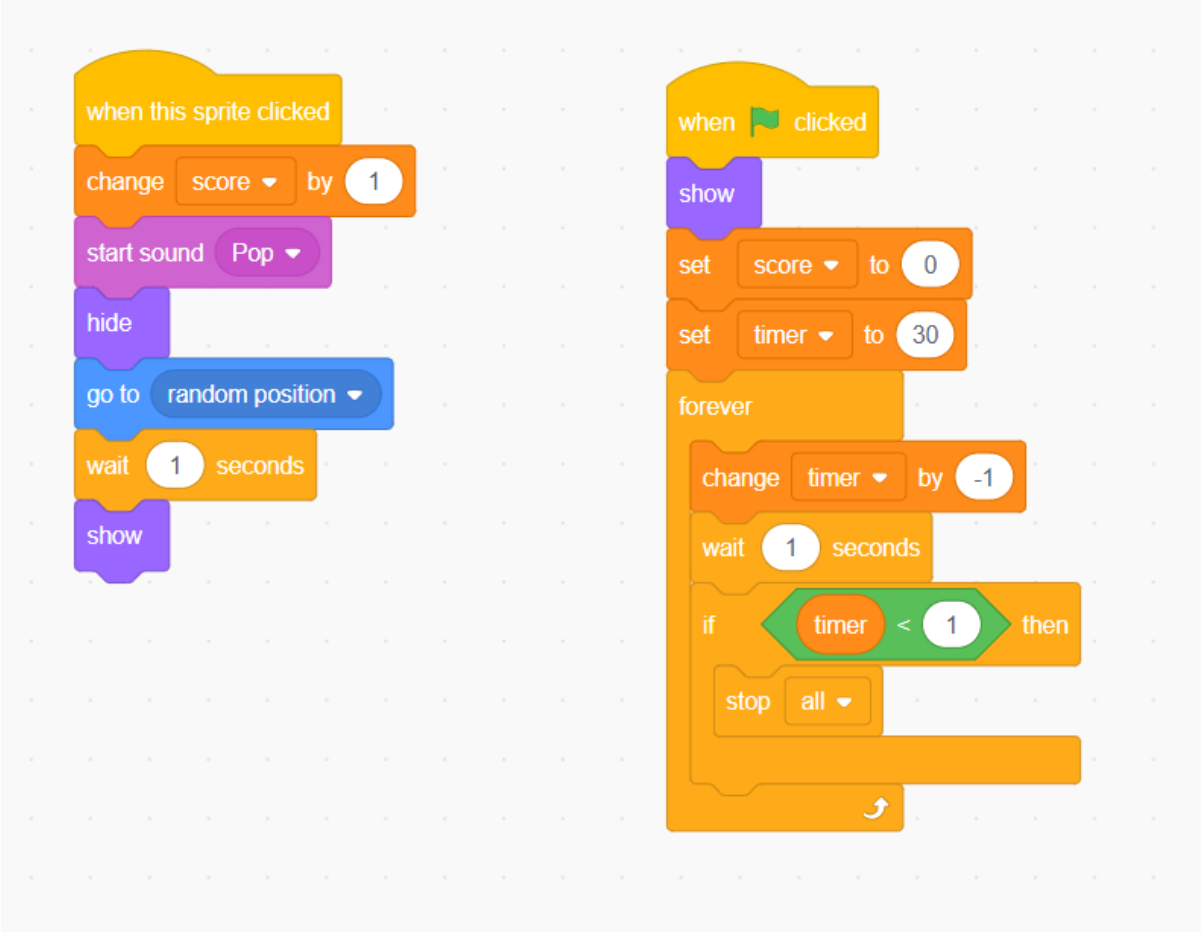


After this, we'll drag in an "if then" block from the **Control** tab, and inside, we'll place the "__ < __" block from the **Operators** tab. Then, into this operator block, we'll place the "timer" variable from the **Variables** tab, and set the second value to be 1. This code will check if the timer has run out.

As a final step, we'll drag in the "stop all" block from the **Control** tab.



Here's the final code:



After this, students are encouraged to duplicate the sprites (by right clicking and selecting 'duplicate' on the sprites), but in all but the original sprite, they must remove all blocks from the "when flag clicked" blocks except the "show" block. Not doing this results in the timer decreasing at double the rate if not even faster due to multiple sprites decreasing it every second.

Students are also encouraged to work on sprite visuals, and to go into the costumes tab and change colors or shapes.

Link to project: https://scratch.mit.edu/projects/1161818291

To see code, follow the link above and select "See inside"

Save your project and test the code before continuing.

Try changing a value or sprite to make the project your own.

Continue at your own pace.

Lesson 11

Smiling Creek: Coding & Design with Scratch - Lesson 11

Lesson overview

Complete this lesson at your own pace.

Optional design tool: Canva

To explore design alongside the coding projects, you can also set up Canva.
Follow the steps below and check that you can sign in on your device.
Keep your sign-in details private.

Go over to canva.com

Select where it says "Sign up" in the top right corner.

Follow the on-screen instructions to create an account, with help if needed.

Select any username.

If Canva asks you to verify your email, open the inbox for the address used to sign up and find the verification message.

Complete verification, then check that you can open Canva.

Return to the coding project below when you are ready.

Find the complete code and project link at the end of this lesson.

Follow the steps below to build the project.

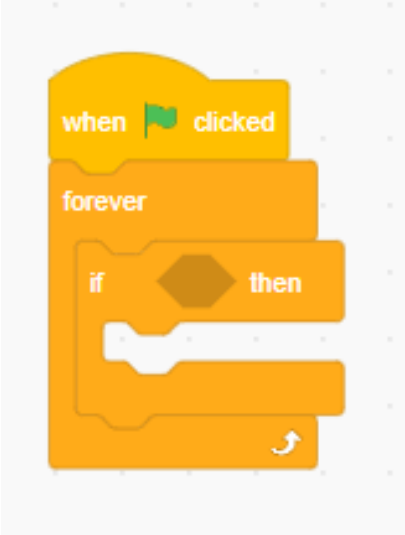
This lesson, we learned how to use multiple sprites to make a simple cat and mouse game. Also, we expanded our general coding knowledge, as well as learned how to use the sound menu in Scratch.

NOTE: In this review, we talk about both a computer mouse, as in, what you use to click and scroll, and also a mouse sprite in our game. We've tried to be as clear as possible which is which, but we are sorry in advance for any confusion.

To begin, we created two new sprites: the cat and mouse sprite.

We first had to program the mouse to be able to move towards where the player clicks.

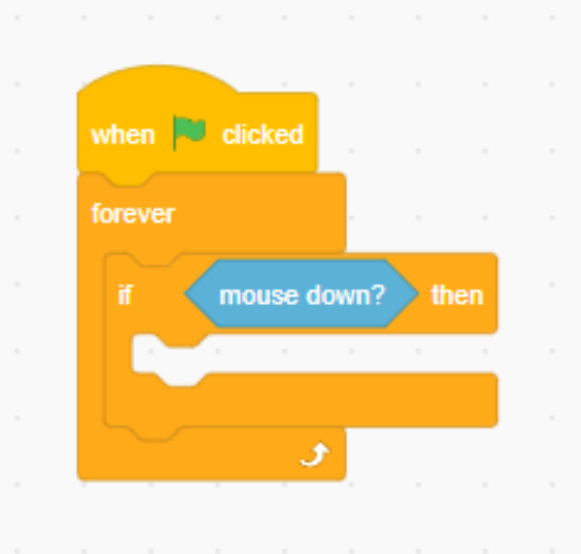
To do this, we used a forever loop in combination with an if loop to notice when the mouse is pressed.



We used a new type of block inside of the if loop, one the **light blue** blocks, called a **sensing** block.

This specific **sensing** block tells us when the mouse is pressed, something we need to know for our code.

Keep in mind that **light blue** blocks are **sensing** blocks, and **dark blue** blocks are used for **motion**.



(This code tells the mouse sprite that forever, it's going to keep checking if the computer mouse is pressed down, and then if it is down, then the mouse sprite will do something.)

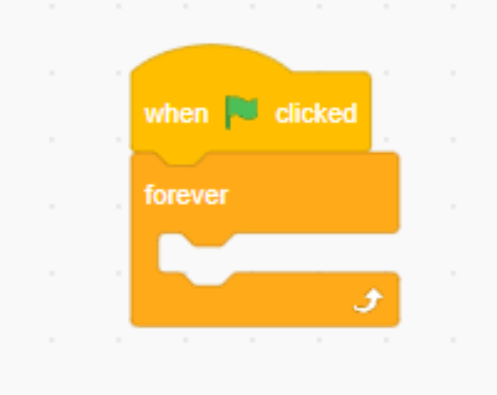
For us, we want this something to be the mouse moving towards where we clicked.

To do this, we use the point towards block and the glide 1 secs block to instruct our mouse to do these motions.

We also make sure that we are pointing towards the 'mouse-pointer', and that we are gliding towards 'mouse-pointer'.

Next, we want to allow the cat to chase the mouse.

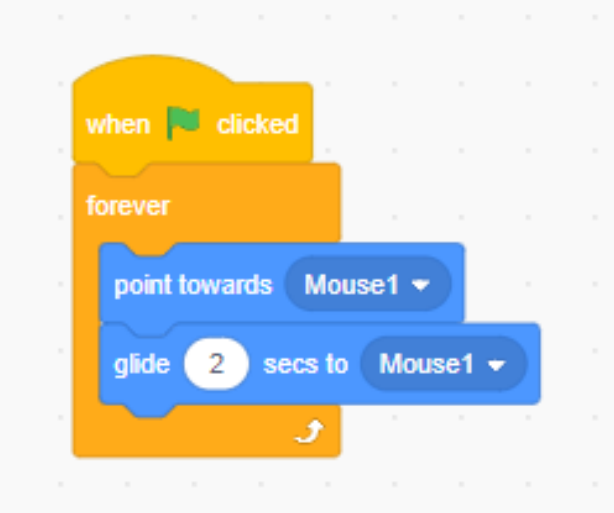
To do this, all we need is another forever loop underneath a when flag clicked block.



Inside of this, we want to keep telling the cat to glide for 1 second towards the mouse. For this, we can use the glide 1 secs block.

We want to select the '1' and change it to '2', so that our cat is slightly slower than our mouse to keep the game fair.

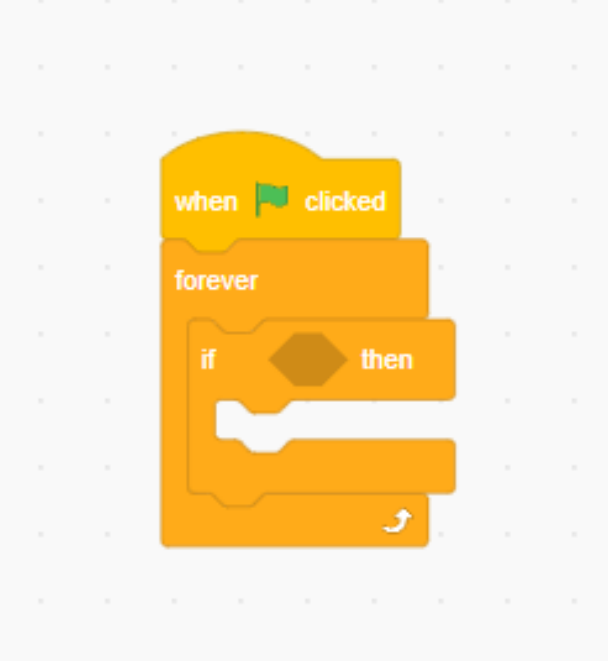
We also want to place the point towards block above this, so that our cat is oriented towards the mouse as it chases.



Now we have a cat that chases our mouse, but what happens when the mouse gets caught?

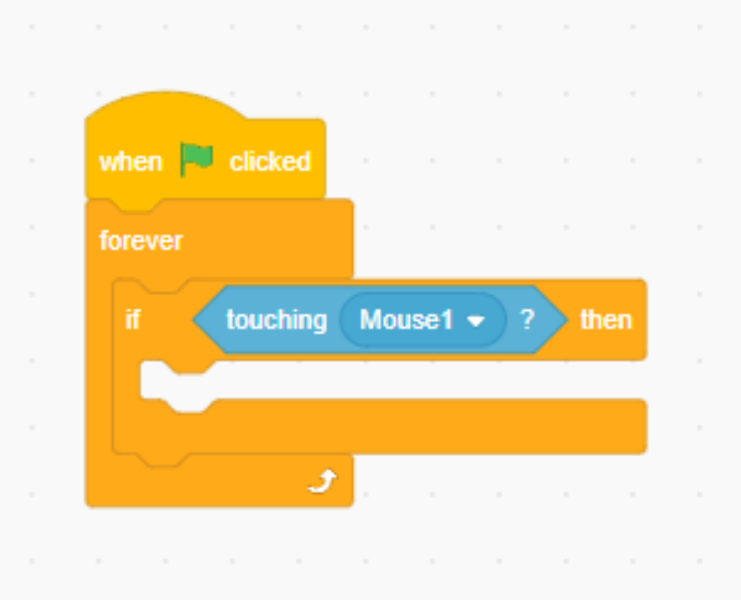
To create a reaction from the cat, we again need to use **sensing** blocks.

We will first use another when flag clicked, forever loop, and if loop to check if something is happening. In this case, something will be contact with the mouse.



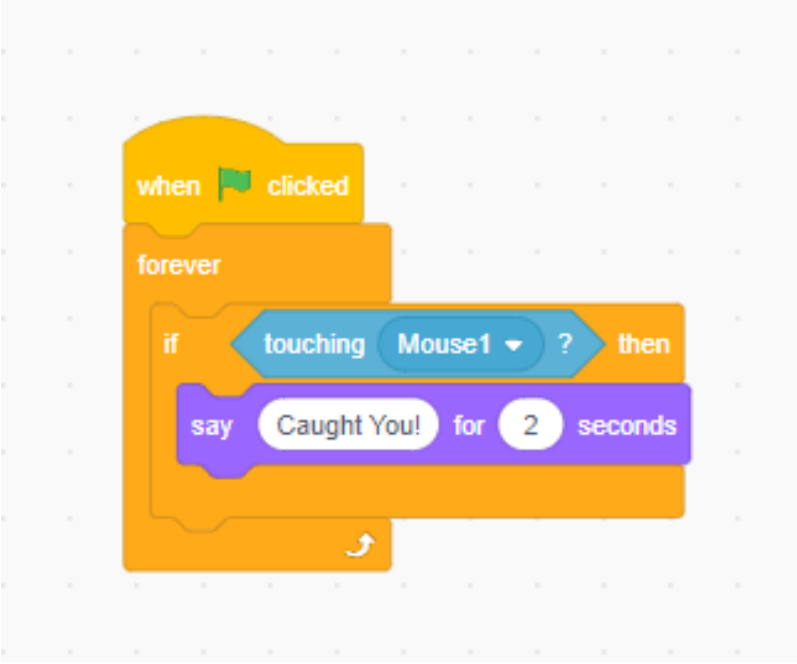
To check if the cat is in contact with the mouse, we need to go on the **sensing** tab on the left, and pull out the touching ___ block.

Then, drag this into the slot after if.



(Translated to English: forever, we'll keep checking if the cat is touching the mouse, and if we are, then we will do something)

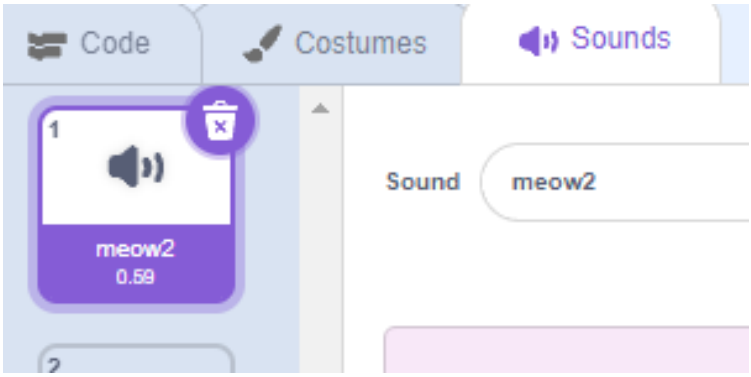
Now that we can check if the animals are touching, we want our cat to say something. To do this, we go to the looks tab on the left, and drag in the "Say 'Hello' for 2 seconds" block. Select 'Hello', and change it to, "Caught You!", or whatever other witty message you'd like the cat to say when it catches the mouse.



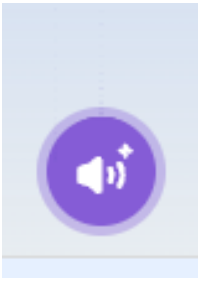
The last key learning from this lesson was how to use the "**Sounds**" tab in Scratch, this time in more depth.
Right now, when we drag in a "**Play Sound**" block, there's only one or two sounds available to play.

However, Scratch has thousands of different sound effects to choose from, and they can be accessed by:

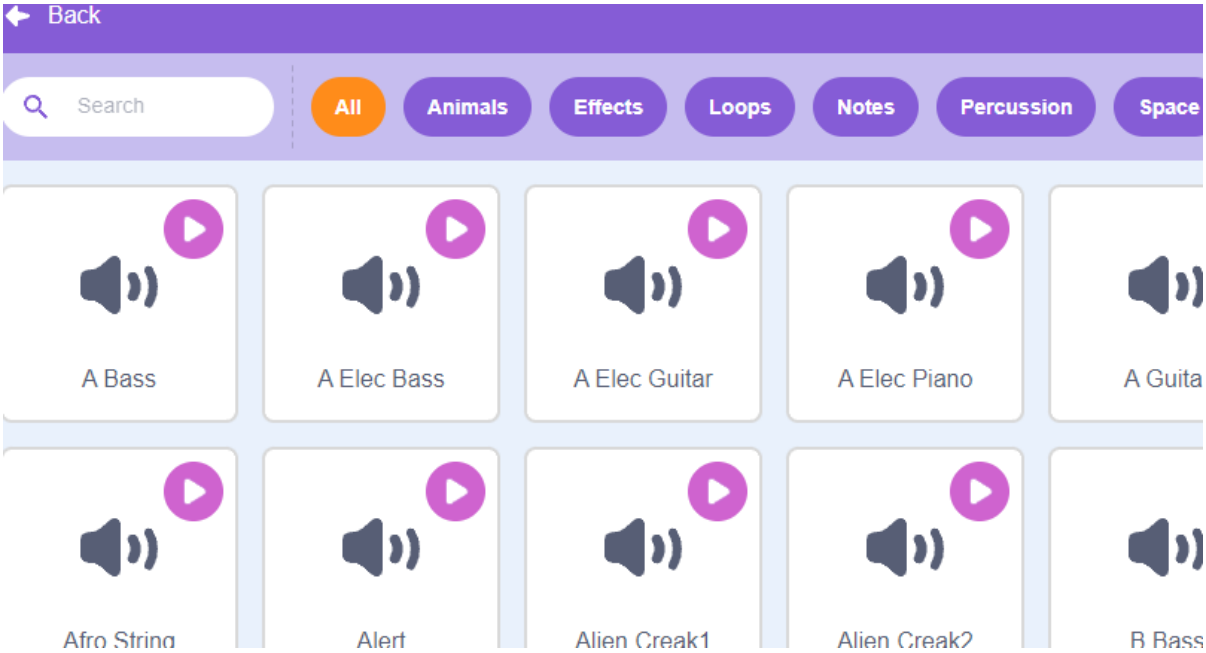
Click on the sounds window in the top left.



Click the circle on the bottom, reading "Choose a Sound"



Search whatever sound you'd like, and select the one you choose to be able to use it in the game.

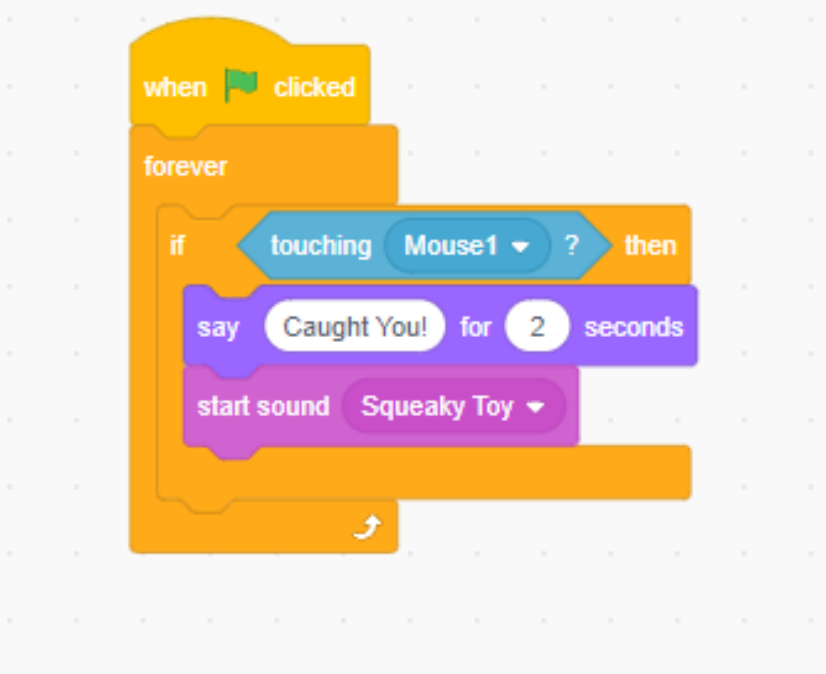


Now that we know how to do this, we select a sound to represent the cat catching the mouse.

For mine, I chose a squeak noise.

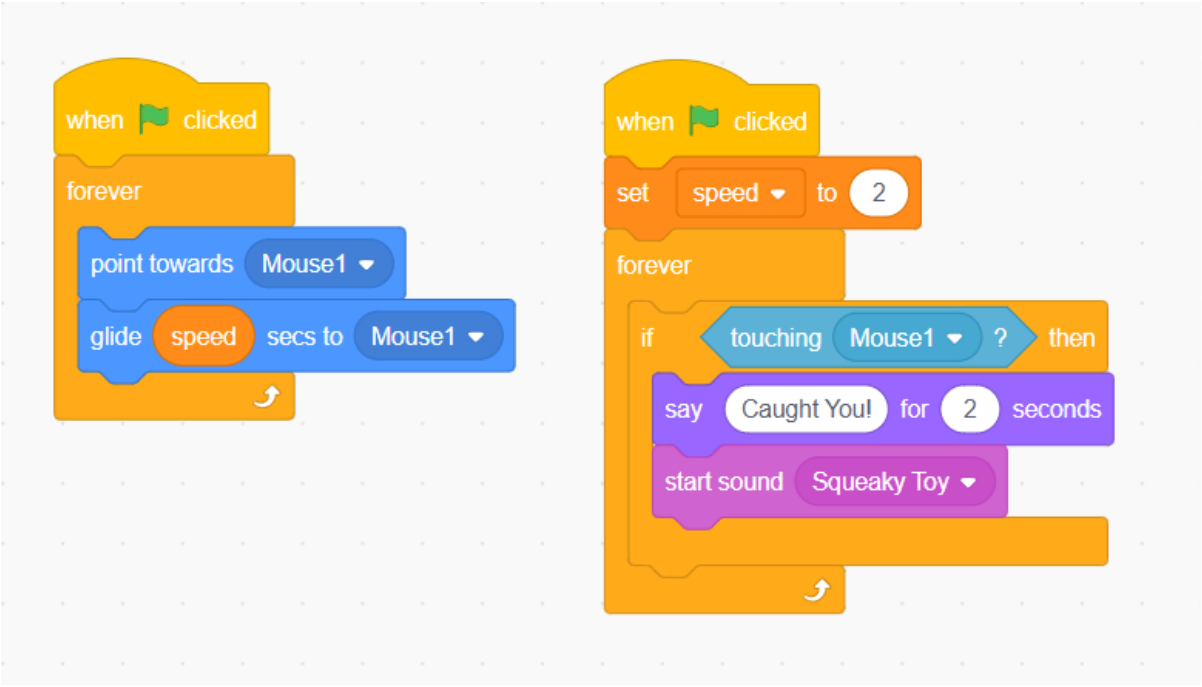
To add this to our game, just enter the **sound** blocks, (not to be confused with the sound tab we just used), and drag in the "Start Sound" block above the Say block.

Then, select the box in the block and change it to whichever sound you chose.

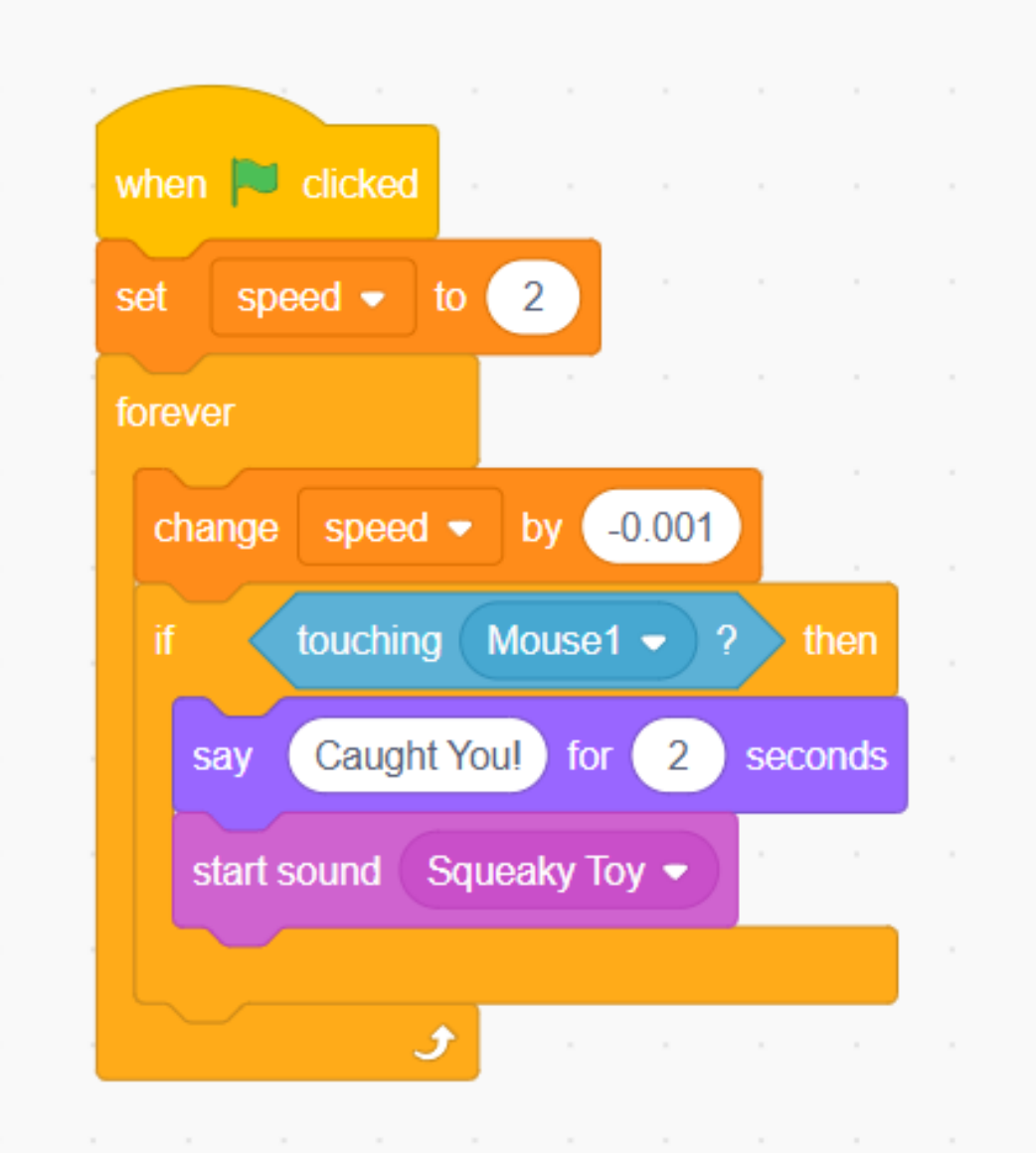


After this, we make a new Variable called speed. To make a variable, go over to the variables tab, select Make a Variable, and then type in speed and select ok.

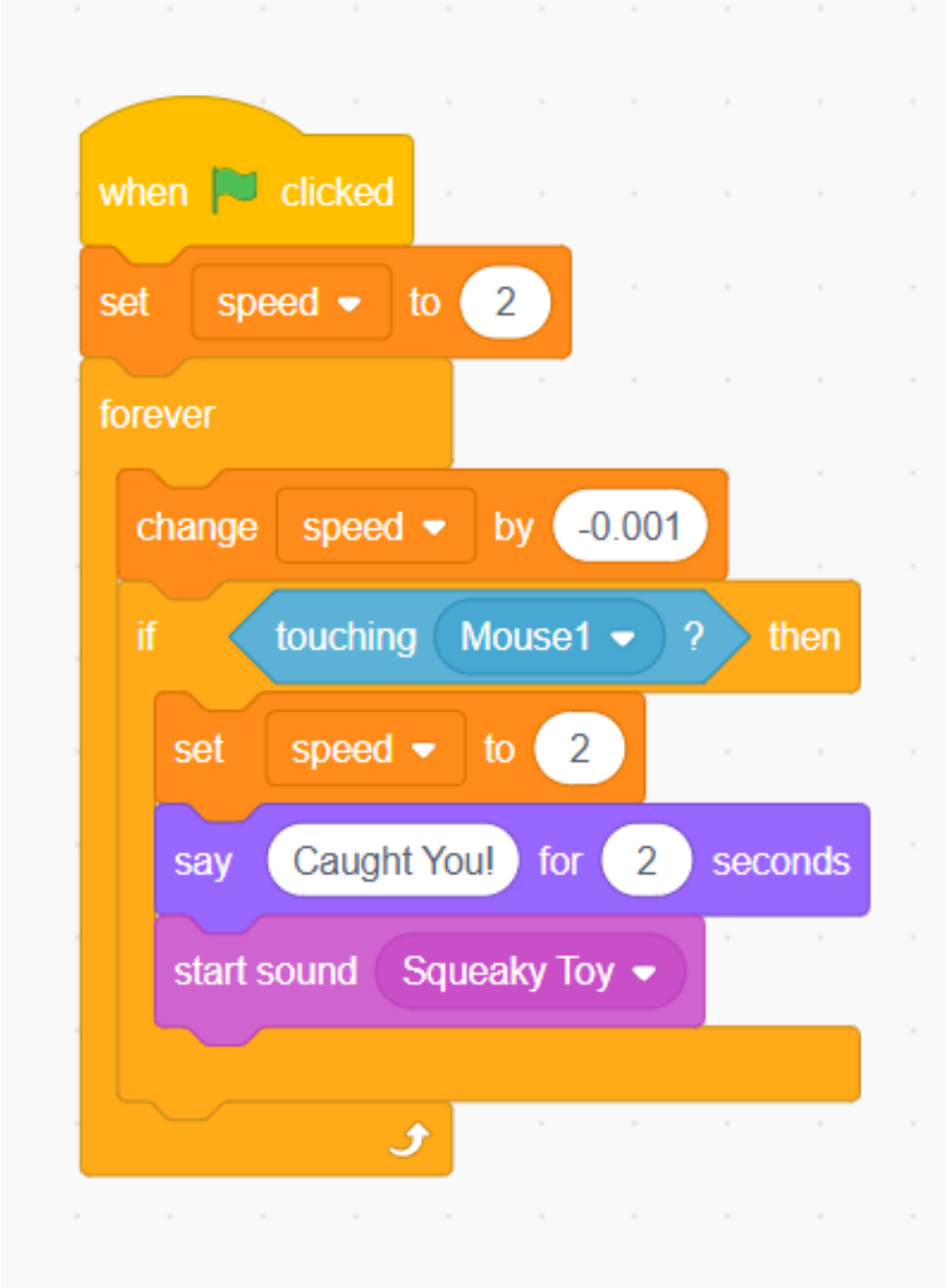
First, drag in the "speed" variable where it says "glide __ secs to Mouse1". Then, drag in a "set __ to" block from the Variables tab and drag it right after "when flag clicked". Make sure it says "set speed to 2".



Then, after the forever block, drag in the "change __ by __" block, and make it say "change speed by -0.001". This will ensure that the cat speeds up over time.



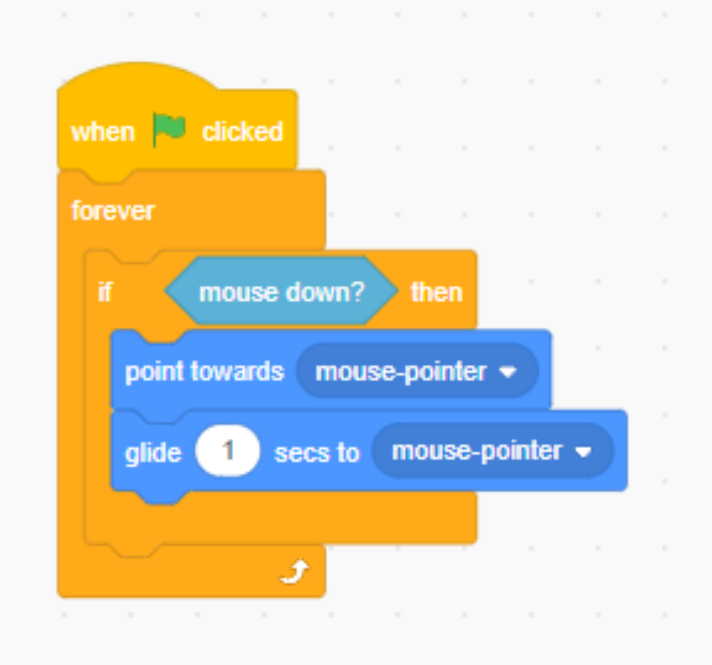
Finally, right above where it says "say Caught You!", drag in a "set __ to __" block, and make it say "set speed to 2". This will reset the game when the cat catches you.



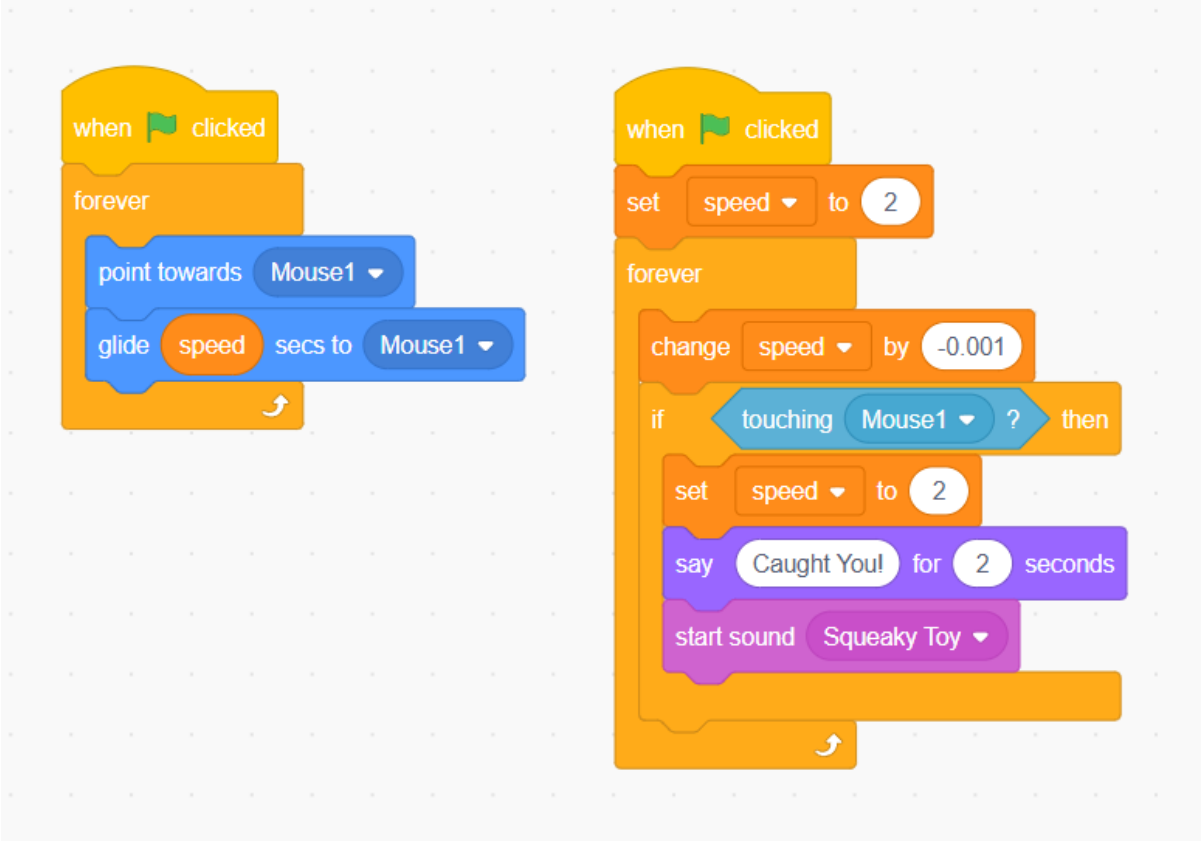
Now, our code is finished!

All complete code:

For the Mouse:



For the Cat:



Link to project: https://scratch.mit.edu/projects/1167454003

Save your project and test the code before continuing.

Course complete

Revisit a lesson to practise, or combine the ideas to create a project of your own.