

Name of Lesson Plan	Grade and Strand
Coding Guide	

Play an example of the completed program here: <https://scratch.mit.edu/projects/492469291/>

The goals of this activity:

1. Create clickable sprites that trigger an output when clicked.
2. Build a counter that accumulates one point for every time a sprite is clicked.
3. Build a score multiplier system that increases the number of points per click when certain score values are met.
4. Add “purchasable” items that autonomously increase or decrease the score on the counter (without clicks)
5. Decrease the score on the counter based on the “cost” value of purchases.

Here is what the final game does:

- The amount of carbon starts at 10,000
- The carbon increases by 1 every second
- Clicking the tree will decrease the carbon level down by 1.
- Clicking the tree will increase the amount of gold by 1
- Buying a factory adds an automatic increase in gold. Every purchase increases the rate of gold by one per second.
- Buying a factory also increases the carbon rate by one per second.
- Buying a green action reduces the carbon level by 100 and reduces the rate of carbon by one per second.

The goal of the game is in time to reduce the amount of carbon to zero, but as students will see, that is no easy task.

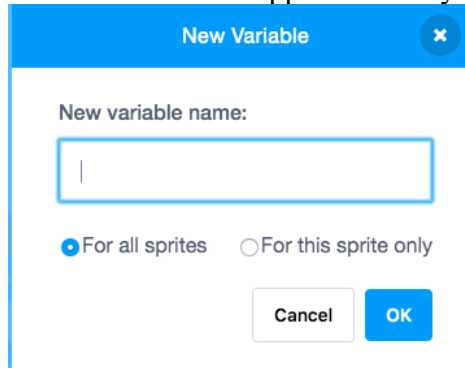
Create Variables:

Variables store information that can be used in the program. We will need four variables for this program:

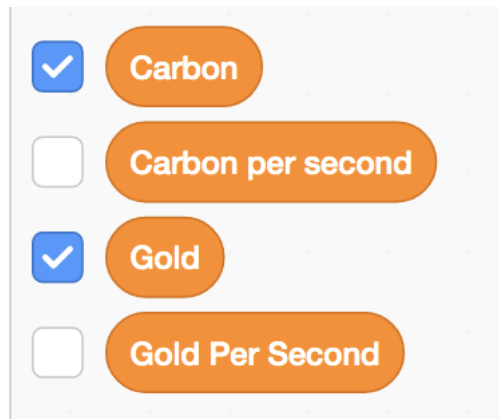
1. **Carbon Counter** - represents the total amount of carbon accumulated in the atmosphere during the game. The counter will be preset to 10,000 units and will increase autonomously by a fixed carbon per second variable (see below). This imitates industries and processes that continuously generate emissions. The counter is depleted by tree clicks (planting trees).
2. **Carbon per second** - stores information about upgrades (factories and green actions) that the player has purchased, which accumulate or deplete carbon automatically (without clicks) over time.
3. **Gold** – this counter represents the total amount of gold generated by tree clicks and produced autonomously by factories. Purchasing upgrades (factories and green actions) will cost gold and decrease the gold counter.

4. **Gold per Second** – stores information about autonomous gold generation rates per second (based on the number of factories purchased). For example, for each factory purchased, the gold per second rate will increase by 1 gold per second.

In the dark orange **Variables** menu, click the **Make a Variable** button to create a variable. Type the variable name into the **New variable name** field, and make sure that **for all sprites** is selected. This will make sure that the variable's stored information applies to every element of our game.



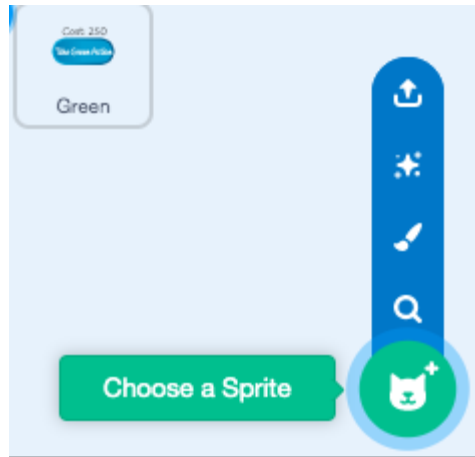
Whenever a new variable is created, a matching counter display appears on the stage. We only need the **Carbon Counter** and **Gold Counter** variables to be visible — the other two variables will be working behind the scenes. To make the other two variables invisible, uncheck the boxes next to those variables in the **Variables** menu.



Create Clicker Sprite

For the interactive element of the clicker game, we need a clickable sprite. The default sprite in Scratch is a cat — remove the cat sprite by selecting it in the sprite panel and clicking the trash can (delete) icon that appears.

To add a new sprite, open the Sprite library by clicking the **Choose a Sprite** button in the sprite panel.



Using the search bar in the Choose a Sprite library, search for “tree” and select the **Trees1** sprite.

You can drag the tree sprite into its desired position on the stage by clicking and dragging with your mouse.

Make the Base Clicker Program:

The Base Clicker program has one basic function: to add one point to the carbon counter score each time you click the sprite.

To make the clicker more visually interesting, we are also going to add code that locks the tree sprite in position while we play (so it doesn't move around while we're clicking), and so that the sprite give a visual output to confirm our clicks.

In the yellow **Events** menu, choose the **When green flag clicked** block and drag it onto the coding canvas. This will trigger the program to start when the tree sprite is clicked.

In the dark orange **Variables** menu, choose the **set [my variable] to 0** block and connect it directly below the **When green flag clicked** block. Using the drop-down list on this block, select the **Carbon Counter** variable. Change the value on this block to 10,000 This will reset the Carbon Counter to 10,000 every time we start the program.

In the blue **Movement** menu, choose the **go to x:0 y:0** block and connect it below the **set Carbon Counter to 0** block. Set the x and y values on this block to the position where you'd like to pin your tree sprite, where x is the horizontal position in pixels, and y is the vertical position in pixels (x: 0, y: 0 would be the centre of the stage). This will keep the sprite from moving around the screen while the player is clicking during gameplay. In the sample code, the tree is pinned to x: 0 and y: -85.

Now we'll program what happens when the tree sprite is clicked.

In the orange **Control** menu, select the **forever** loop block and connect it below the **go to x:0 y:-85** block. The rest of the program will be built *inside* of the forever loop and will be repeated indefinitely while the program is running.

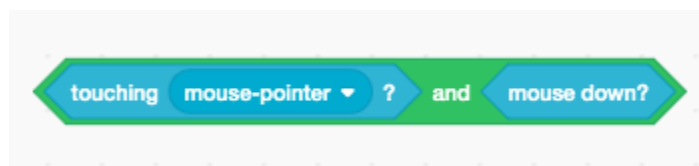
In the orange **Control** menu, select the **if [blank] then, else** conditional block and nest it inside the **forever** loop.



To describe the condition for the conditional statement, we'll need to use operator blocks and sensing blocks to define when the tree sprite is being clicked and released.

In the green **Operators** menu, select the **[blank] and [blank]** block and drag that onto the coding canvas.

In the light blue **Sensing** menu, select the **touching mouse-pointer?** block and drag it onto the first space of the green operator block. Still in the **Sensing** menu, select the **mouse down?** block and drag it onto the second blank space of the green operator block:



This defines when a player has moved the mouse cursor over the sprite and left-clicked the mouse. Place this combined block on the shaded hexagonal space on the **if [blank] then, else** conditional block. The shaded space will expand to fit this block.

Now we'll describe the output code for when the tree sprite is clicked and we'll place this code in the first "mouth" of the **if [blank] then, else** conditional block.

In the dark orange **Variables** menu, select the **change my variable by 1** block and drag it into the first "mouth" of the **if [blank] then, else** conditional block. Using the dropdown tool on this block, make sure that the **Gold** variable is selected. This means that every time the tree sprite is clicked, the **Gold Counter** score will increase by one point.

In the dark orange **Variables** menu, grab a second **change my variable by 1** block and drag it into place below the **Change Gold by 1** block. Using the dropdown tool on this block, make sure that the **Carbon Counter** variable is selected. Change the value from 1 to -1. This means that every time the tree sprite is clicked, the **Carbon** counter is decreased by one point (as if planting one tree reclaims 1 unit of carbon from the atmosphere).

In the purple **Looks** menu, select the **set size to 100%** block and place it below the **change Carbon Counter by -1** block. This will give a visual output to show that the sprite was successfully clicked.

If you test the program at this point, you'll be able to hold down the mouse button while clicking on the tree sprite, and the score will keep increasing until the mouse button is released. Because we want only one point added per mouse click (no matter how long the mouse button is held down), we'll have to add a control element.

In the orange **Control** menu, select the **wait until** block and place it below the **set size to 100%** block.

We want the program to wait until the mouse is released to continue. To define this, go into the green **Operators** menu and select the **[blank] = [blank]** block. In the light blue **Sensing** menu, select the **mouse down?** block and place it on the first blank space of the green operator block. In the second space, type the word "false".



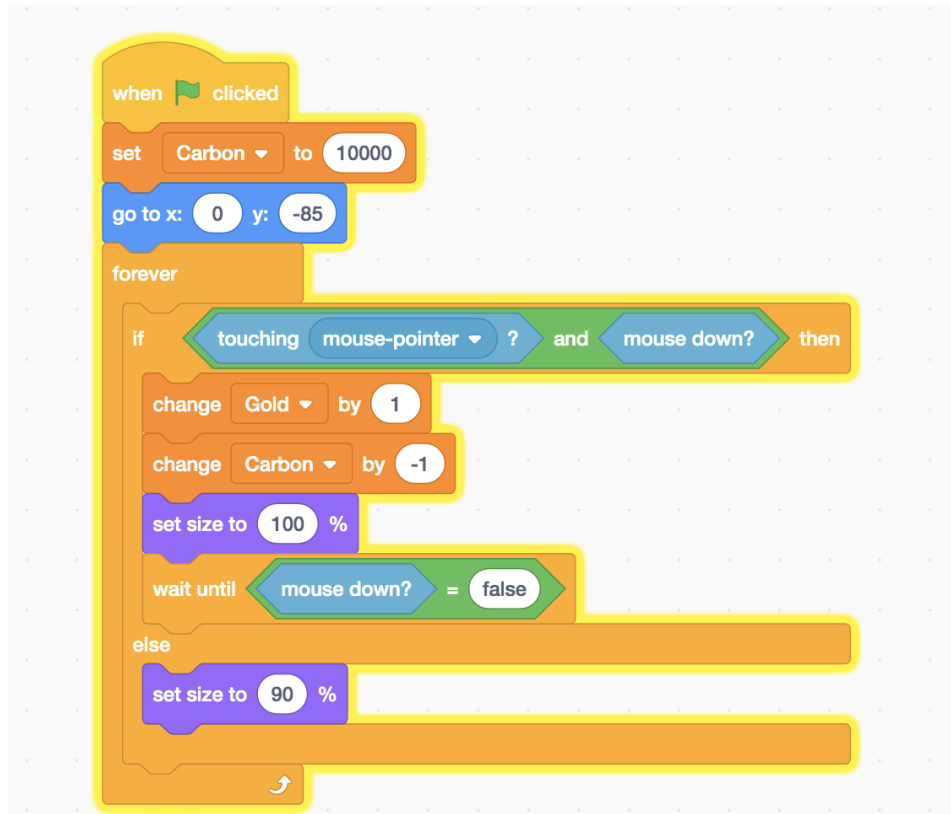
Place this combined block onto the shaded hexagon of the **wait until** block. Defined in this way, the program will wait until the mouse button is released (mouse down = false, whereas a mouse click would register as mouse down = true) to continue to the next step of the program.

Next, we need to program the output for when the tree sprite is not being clicked. Since the score does not change when the sprite is *not* being clicked, the only output is to change the size of the sprite (the other half of the appearance change code described in the first "mouth" of the conditional statement block).

In the purple **Looks** menu, select the **set size to 100%** block and place it into the second "mouth" (the **else** space) of the conditional statement block. Change the value of this block to 90%.

If you test your program now, the **Carbon** counter should start at 10,000 and decrease by 1 point every time the tree sprite is clicked. The **Gold** counter should start at 0 and increase by 1 point every time the tree sprite is clicked. When the tree sprite is clicked and released, the sprite should briefly change in size.

The completed clicker program should look like this:



Add “Upgrades” - Factories and Green Action

To add “factories” that increase **Carbon** emissions while increasing **Gold** production

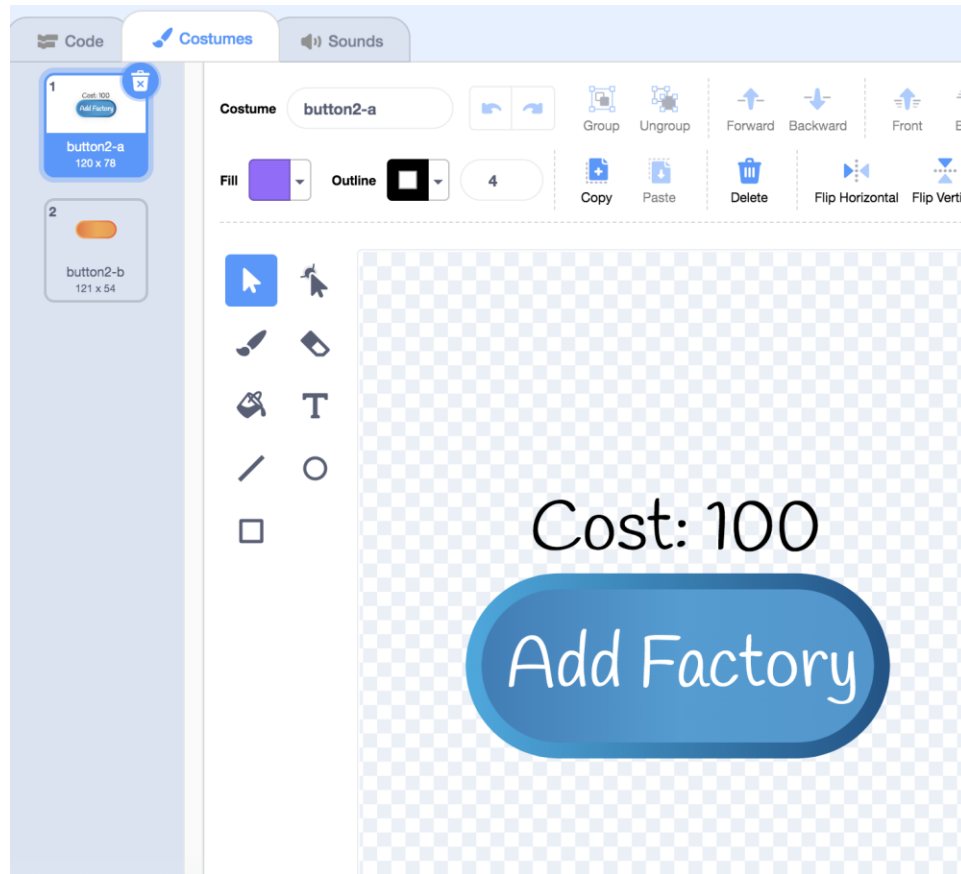
- Add a button sprite:

To add a new sprite, open the Sprite library by clicking the **Choose a Sprite** button in the sprite panel.

Using the search bar in the Choose a Sprite library, search for “button” and select your preferred button sprite.

To label your button, select the **Costumes** tab at the top-left of the screen.

In the Costumes tab, add text to your button using the Text box tool. Modify the colour of your text using the fill tool and drag the text into place. In the below example, we have used two text boxes: one to label the button, and a “price tag” cost label that sits above the button. The cost label communicates how many points a player will need to be able to activate the button.



Notice that there are two costumes for this sprite: one blue costume (button2-a) and one orange costume (button2-b). We will be using the second costume later in our code as a visual output to signal when the button has been clicked.

You can drag the button sprite into its desired position on the stage by clicking and dragging with your mouse.

- Program the button:

** Make sure that the button sprite is selected.

Return to the code tab.

In the yellow **Events** menu, select the **when this sprite is clicked** block to start a new program.

In the blue **Motion** menu, select the **go to x:0 y:0** block and connect it below the **when this sprite is clicked** block. Adjust the x and y values on this block to the position where you want to “glue” your button sprite. This will keep the block from being moved around the screen as players click it.

Now, we will program the button so that it changes costumes (from blue to orange) when it is clicked.

In the light orange **Control** menu, select the forever loop and connect it below the **go to x:0 y:0** block. The rest of this code will be built inside the forever loop.

We need to add a conditional statement to produce different outputs for when the button is being clicked versus when it is not being clicked.

In the orange **Control** menu, select the **forever** loop block and connect it below the **go to x:0 y:0** block. The rest of the program will be built *inside* of the forever loop and will be repeated indefinitely while the program is running.

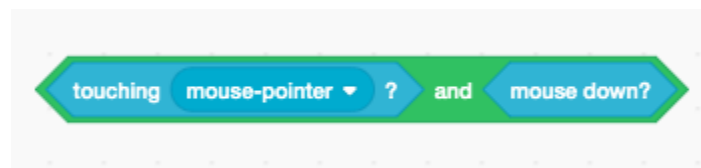
In the orange **Control** menu, select the **if [blank] then, else** conditional block and nest it inside the **forever** loop.



To describe the condition for the conditional statement, we'll need to use operator blocks and sensing blocks to define when the tree sprite is being clicked and released.

In the green **Operators** menu, select the **[blank] and [blank]** block and drag that onto the coding canvas.

In the light blue **Sensing** menu, select the **touching mouse-pointer?** block and drag it onto the first space of the green operator block. Still in the **Sensing** menu, select the **mouse down?** block and drag it onto the second blank space of the green operator block:



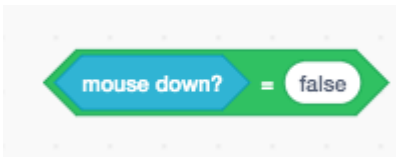
This defines when a player has moved the mouse cursor over the sprite and left-clicked the mouse. Place this combined block on the shaded hexagonal space on the **if [blank] then, else** conditional block. The shaded space will expand to fit this block.

Now we'll describe the output code for when the tree sprite is clicked and we'll place this code in the first "mouth" of the **if [blank] then, else** conditional block.

In the purple **Looks** menu, select the **switch costume to** block and place it inside the first mouth of the conditional block. Use the dropdown menu to select costume **button2-b**.

In the orange **Control** menu, select the **wait until** block and place it below the **switch costume to** block.

We want the program to wait until the mouse is released to continue. To define this, go into the green **Operators** menu and select the **[blank] = [blank]** block. In the light blue **Sensing** menu, select the **mouse down?** block and place it on the first blank space of the green operator block. In the second space, type the word “false”.

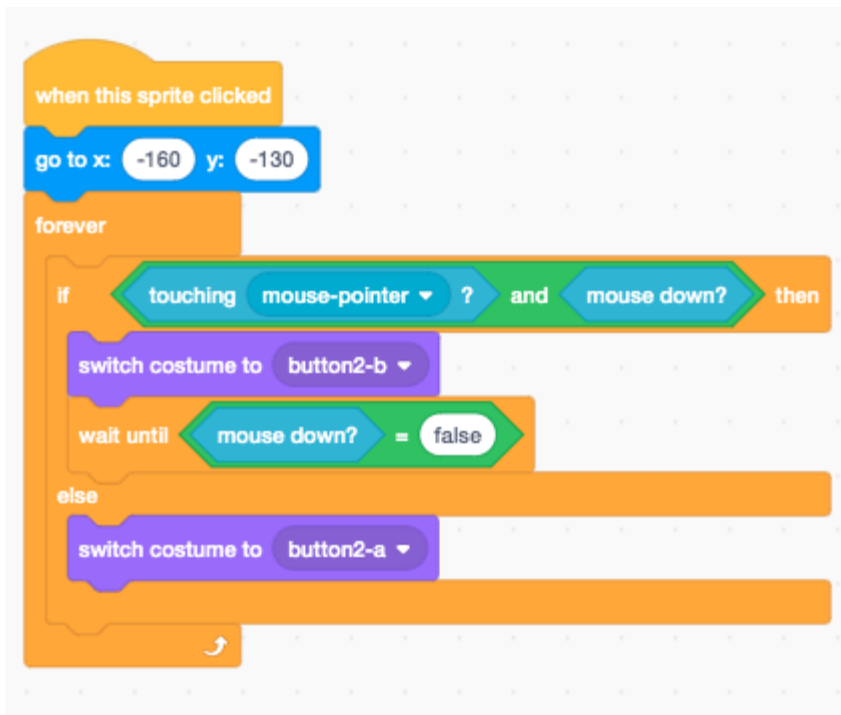


Place this combined block onto the shaded hexagon of the **wait until** block. Defined in this way, the program will wait until the mouse button is released (mouse down = false, whereas a mouse click would register as mouse down = true) to continue to the next step of the program.

Next, we need to program the output for when the button sprite is not being clicked. Since the score does not change when the sprite is *not* being clicked, the only output is to change the size of the sprite (the other half of the appearance change code described in the first “mouth” of the conditional statement block).

In the purple **Looks** menu, select the **switch costume to** block and place it inside the else “mouth” of the conditional statement. Use the dropdown menu to select costume **button2-a**.

The completed program should look like this:



- Program the button effects:

We want this program to achieve three things:

1. We want it to become active (useable by the player) **ONLY** when the player has enough **Gold** points to meet the cost of purchasing a factory.
2. When a factory is purchased, we want the **Carbon** score to autonomously increase by 1 carbon per second (produced automatically by the factory instead of by player clicks)
3. When a factory is purchased, we want the **Gold** score to autonomously increase by 1 carbon per second (produced automatically by the factory instead of by player clicks)
4. We want the player's **Gold** counter score to decrease by 100 points (the cost of purchasing a factory).

In the yellow **Events** menu, select the **when this sprite is clicked** block to start a new program.

In the light orange **Control** tab, choose the **if [blank] then** conditional block (with only one output "mouth"). Connect this block below the **when this sprite is clicked** block.

To describe our condition (if the Carbon Counter score is greater than 100 points), we will need to use a green operator block.

In the **Operators** menu, choose the **[blank] > [blank]** block and drag it onto the shaded hexagon on the **if [blank] then** block.

In the dark orange **Variables** menu, select the oval **Gold** block and drag it on top of the first blank space of the operator block. In the second blank field, change the value to 100.

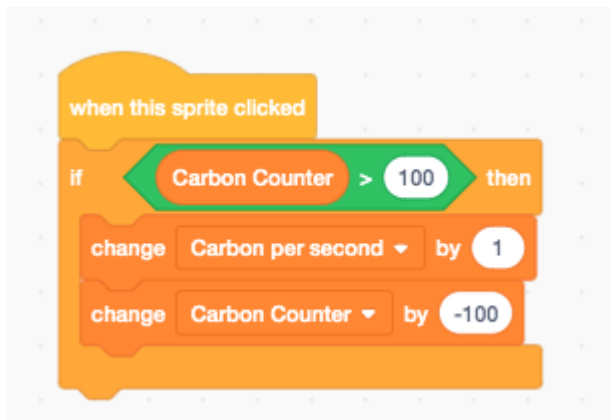
We can now program the output in the “mouth” of the conditional block.

In the dark orange **Variables** menu, choose the **change my variable by 1** block and place it inside the mouth of the conditional block. Using the dropdown menu, make sure that the variable is set to **Carbon per Second**.

In the dark orange **Variables** menu, choose the **change my variable by 1** block and place it inside the mouth of the conditional block. Using the dropdown menu, make sure that the variable is set to **Gold per second**.

In the dark orange **Variables** menu, choose the **change my variable by 1** block and place it inside the mouth of the conditional block. Using the dropdown menu, make sure that the variable is set to **Gold**. Change the value on this block to -100 .

The completed program should look like this.



Now we need to create a program that constantly updates the **Carbon per Second** variable and automatically adjusts the **Carbon** counter score as time passes.

In the yellow **Events** menu, choose the **When green flag clicked** block to start a new program.

In the dark orange **Variables** menu, choose the **set my variable to 0** block and connect it below the **when green flag clicked** block. Use the drop-down menu to select the **Carbon per second** variable. Change the value of this block to 1.

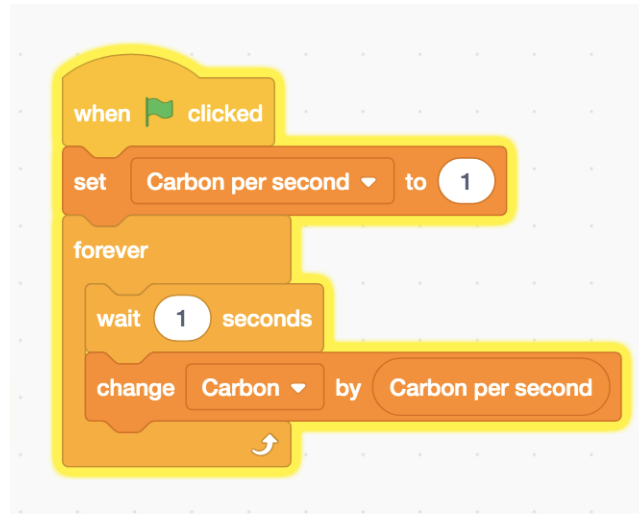
Now we need to program a loop that will increase the Carbon Counter score by the rate indicated by Carbon per second variable. This increase should occur once per second.

In the light orange **Control** menu, select the **forever loop** and connect it below the **set Carbon per second to 1** block. The rest of the program will be built inside the loop.

In the light orange **Control** menu, select the **wait 1 seconds** block and place it inside the **forever loop**.

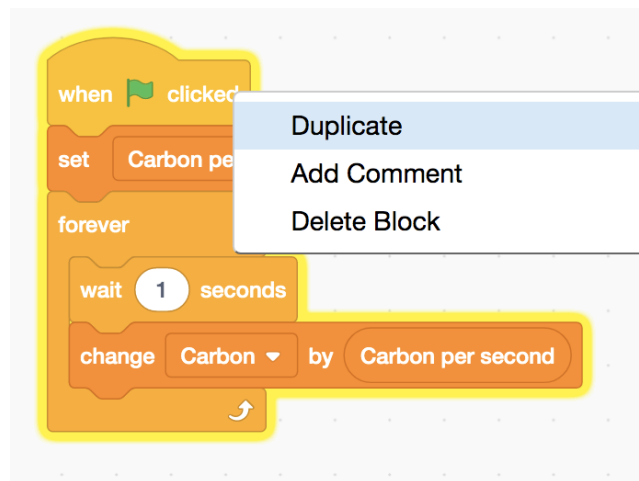
In the dark orange **Variables** menu, select the **change my variable by 1** block. Use the drop-down menu to select the **Carbon** Counter variable. Grab the oval **Carbon per second** variable block and drag it on top of the field on the block (replacing the 1).

The completed program should look like this:



Now we need to create a similar program that constantly updates the **Gold per second** variable and automatically adjusts the **Gold** counter score as time passes.

We can actually duplicate the existing code for updating the **Carbon** counter to use as a base for this program. Right click the top block of that program and select **Duplicate**.



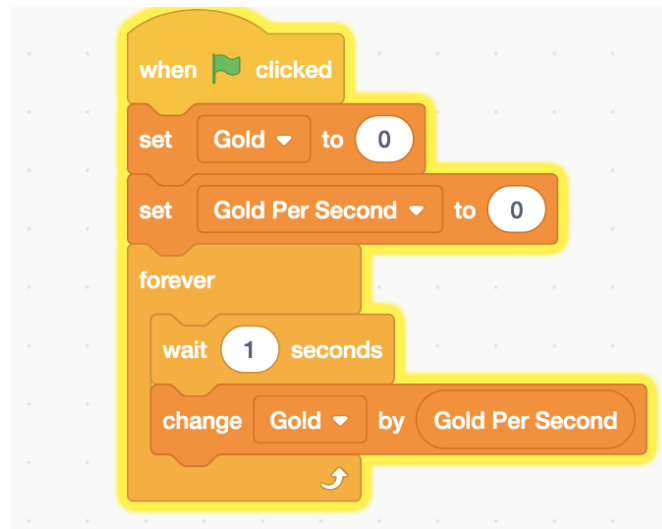
Using the dropdown list on the block, change the variable on the **set Carbon per second to 1** block to **Gold**. Change the value on this block from 1 to 0.

In the dark orange **Variables** menu, choose the **set my variable to 0** block and connect it below the **set Gold to 0** block. Use the dropdown list on this block to select the variable **Gold Per Second**. These

blocks combined mean that when we start the game without any upgrades, our **Gold** counter is set to 0 points and our **Gold per Second** rate is 0, so the only gold points that are generated are generated by clicking the tree sprite.

Change the variables on the **change Carbon by Carbon per second** block inside the forever loop. Use the dropdown list on the block to select **Gold**. In the dark orange **Variables** menu, choose the **Gold per Second** variable oval and drag it on top of **Carbon per second** on the block to replace it. The old **carbon per second** oval will “pop out” of the block and you can delete it completely by dragging it onto the block library.

The completed program should look like this:



To add “green actions” that decrease Carbon

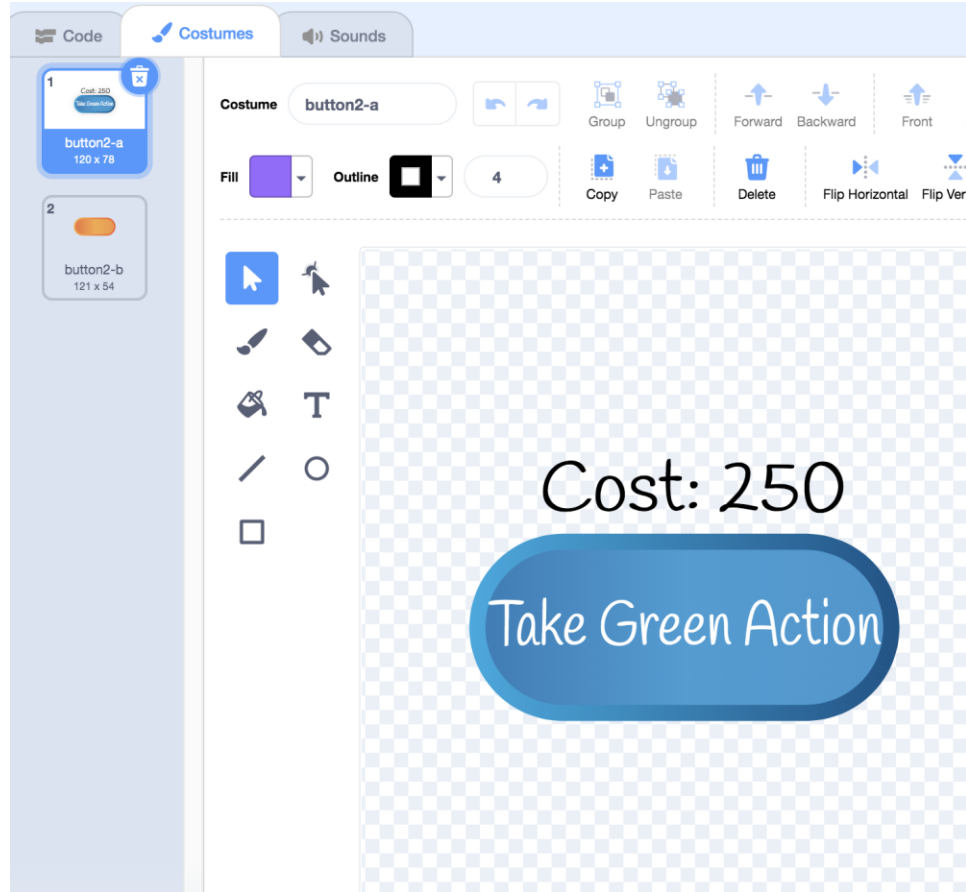
- Add a button sprite:

To add a new sprite, open the Sprite library by clicking the **Choose a Sprite** button in the sprite panel.

Using the search bar in the Choose a Sprite library, search for “button” and select your preferred button sprite.

To label your button, select the **Costumes** tab at the top-left of the screen.

In the Costumes tab, add text to your button using the Text box tool. Modify the colour of your text using the fill tool and drag the text into place. In the below example, we have used two text boxes: one to label the button, and a “price tag” cost label that sits above the button. The cost label communicates how many points a player will need to be able to activate the button.



You can drag the button sprite into its desired position on the stage by clicking and dragging with your mouse.

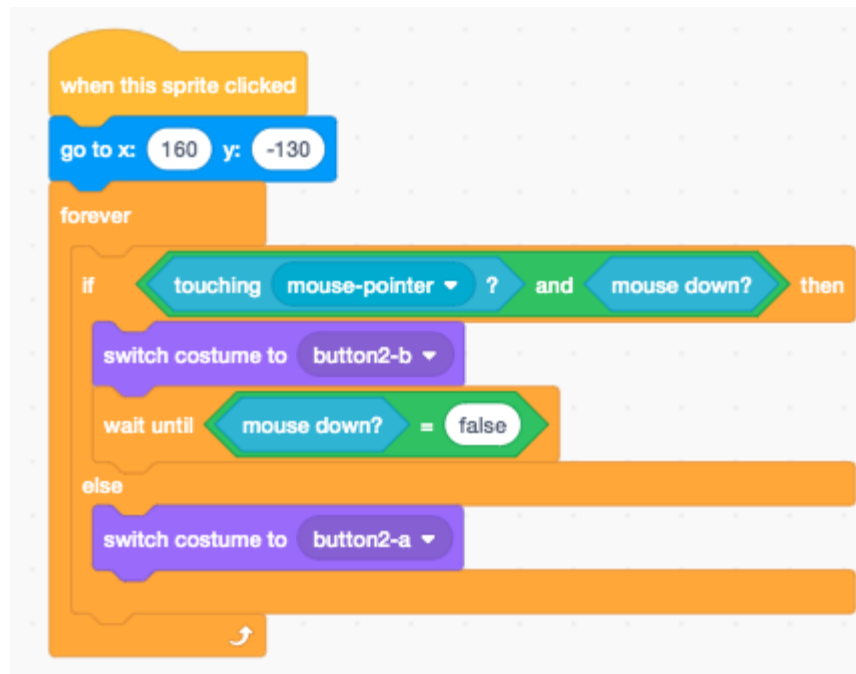
- Program the button appearance:

The code for this button is almost identical to the program we built for the Add Factory button. You can duplicate the existing code by clicking-and-dragging the entire program that you want to copy and dragging it on top of the new button sprite icon in the sprite panel.

If you select the Green Action Button sprite, the copied code should now appear in the coding canvas. The original code in the Add Factory Button sprite will remain.

Once you have copied the code to the Green Action Button sprite, the only change you need to make is to set the x-y position of the button.

The completed program should look like this:



- Program the button effects:

We want this program to achieve three things:

5. We want it to become active (useable by the player) **ONLY** when the player has enough **Gold** points to meet the cost of purchasing a green action.
6. When a green action is purchased, we want the **Carbon** score to autonomously decrease by 1 **carbon per second** (produced automatically by the green action instead of by player clicks)
7. We want the player's **Gold** score to decrease by 250 points (the cost of purchasing a green action).

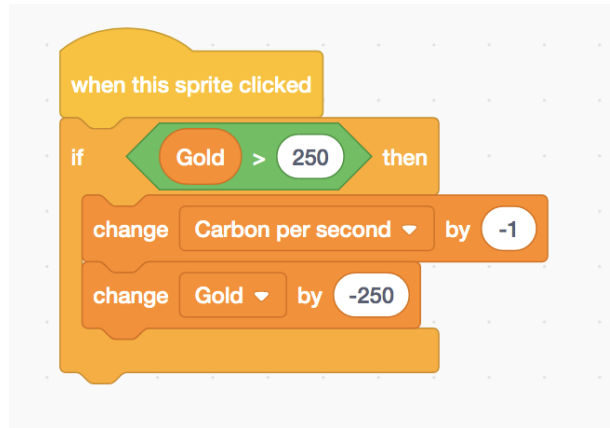
Since these effects are very similar to those for our Add Factory button, we can also duplicate the existing program from the Add Factory button for this new button.

Once the program has been copied over to the Green Action sprite, change the value on the green operator block to 250 (to reflect that the green action button should be clickable only when the Carbon Counter score is 250 points or higher).

Change the value on the **change carbon per second by** block to -1, since the green action should decrease the amount of carbon over time.

Change the value on the **change Gold by** block to -250 to reflect the cost of purchasing a green action.

The completed program should look like this:



Opportunities for Extension

Modify the appearance of your game.

In the playable example of the game (linked at the top of this document), you might notice that I've added a background and programmed it to change colours when the amount of carbon in the atmosphere reaches set values.

Add a Background:

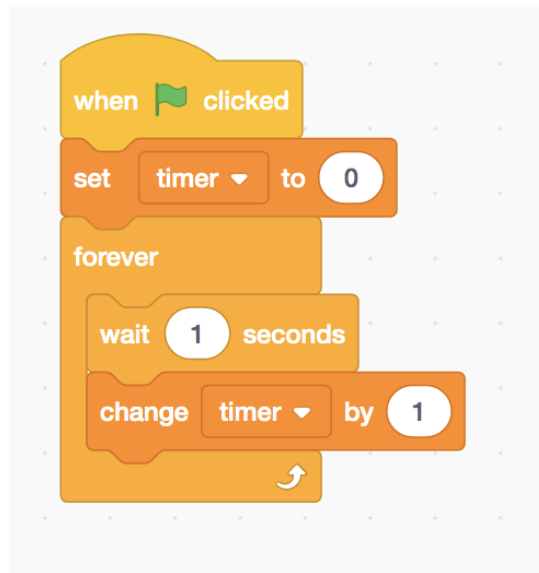
The background pane can be found to the right of the Sprite panel. To add a background, click the **Choose a Backdrop** button at the far bottom-right of the screen.

Choose the desired background from the backdrop library. In the sample good, we've selected the **Blue Sky** backdrop.

Add a Timer

By creating a timer variable, we can keep track of how many seconds it takes to reach zero carbon units (or, conversely, set a countdown clock).

A basic timer that counts seconds might look like this:



Encourage students to modify the values in the carbon counter, gold counter, carbon per second, and gold per second fields to change the outputs for different elements of the game.