Create An Interactive Game Using AI

Combine a visual programming language, artificial intelligence, and machine learning to program an interactive game using hand gestures.

Materials

- A computer with a camera
- Internet access

Get To Know The PoseBlocks Platform

In this activity, you're going to take the ideas behind the SignAll app and simplify them to create an interactive game that recognizes hand motions and reacts to them. You will move your hand around the screen as you try to touch randomly appearing objects. The camera on your computer will watch where your hand moves and its artificial intelligence will track and understand your movement.

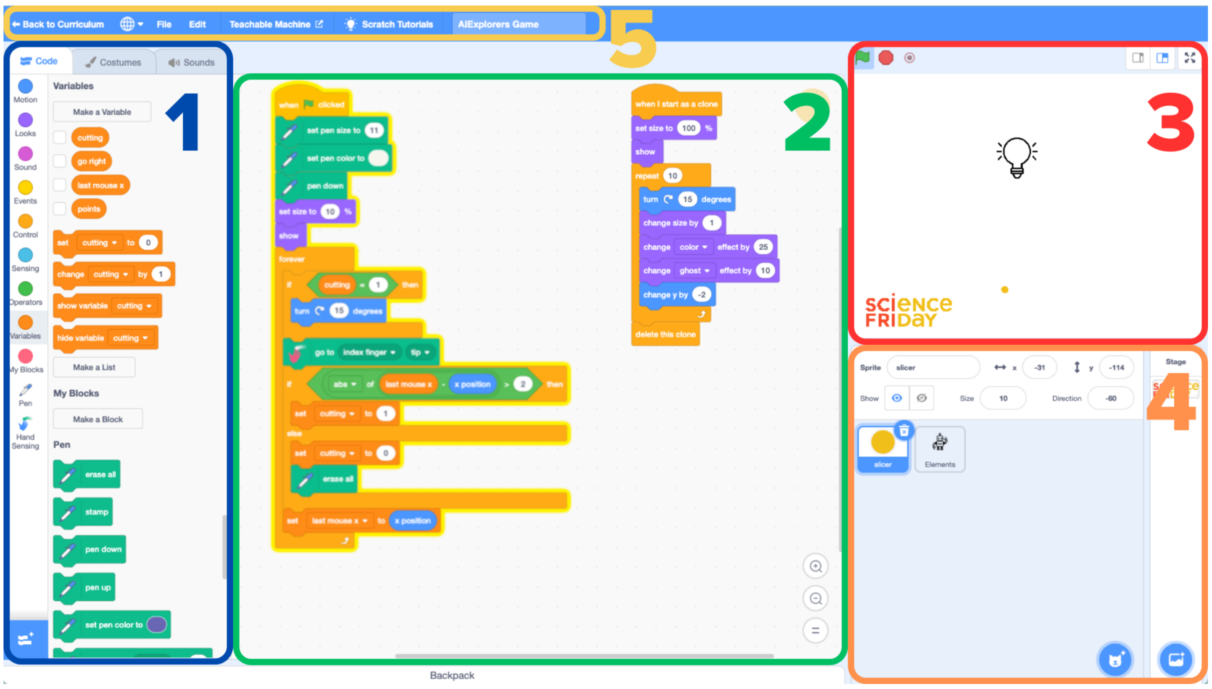
You may want to work with a partner or a group on this project so that you have friends to help brainstorm ideas, offer feedback, and debug your code. *You don't need any prior knowledge of AI or machine learning to do this activity.*

For this project, you'll be using the PoseBlocks Software by MIT. This free toolkit uses drag-and-drop blocks to create code, a lot like the popular Scratch coding language. The difference between them is that PoseBlocks has special blocks to use built-in body-tracking artificial intelligence.

To create your code, you will click on the slicer sprite and drag blocks into the **workspace**. The PoseBlock workspace has several sections. (If you're familiar with Scratch, this will seem very familiar.)

The area on the left has three tabs labeled **"Code," "Costumes," and "Sounds,"** each of which includes a library of items you can use. All the code blocks are in the code library, where they are grouped by function and color-coded. The costumes library allows you to add, change, or draw sprites. "**Sprite**" is the name for an object or character in PoseBlock. Each sprite can be given code to do something in your program. When a sprite has multiple forms, we call them costumes. For example, the elements sprite has nine costumes. Click on the "Costumes" tab to see them. (You can use any sprite you like. You can even draw your own.)

The sounds library allows you to add or record sounds. The **Code Area** in the middle is where you will drag your code blocks and click them together to make your game. On the top right is the **Stage**. This is where you can see the sprite perform the actions you code. The green flag starts the game. The red sign stops it. Below that is the **Sprite Pane**. This panel is where you can select sprites to code and adjust some settings for sprites. You can also select a custom Backdrop. Along the top is the **Toolbar**. That's where you can load sample code and give your project a name. Also note, you can click the "Globe" 🌐 to adjust the language used. There are over 70 available.



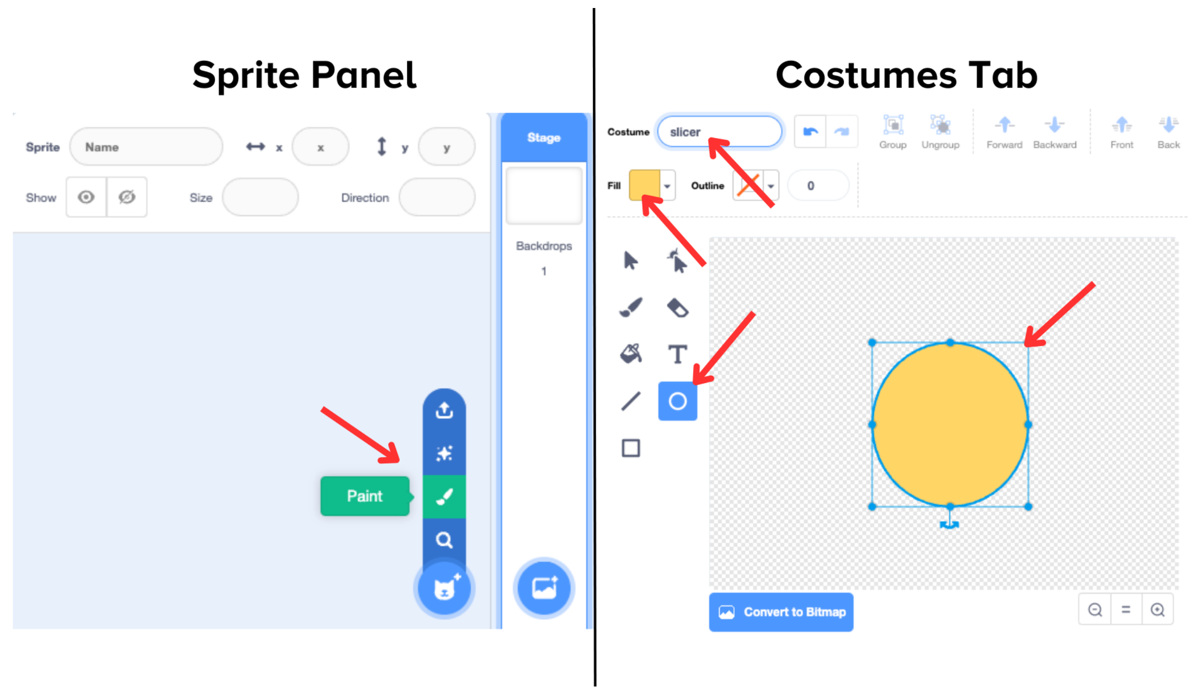
(1) The tabs for code blocks, costumes, and sounds. (2) The Code Area. (3) Stage. (4) Sprite Pane. (5) Toolbar. Created in Canva by Science Friday using a Scratch screen capture.

To get you started quickly on this project, a special directory of samples has been created. In the directory, you'll find three files: Game, Tutorial, and Game Graphics. The file called "Game Graphics" contains the sprites used in the game as PNG files so that you can edit them, upload just the ones you like, or use them in other creative ways. You can also download this file, which will upload all the sprites to PoseBlocks all at once.

Create And Code Your Game

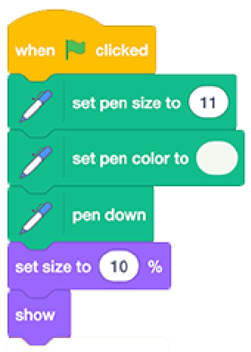
EMBED: https://youtu.be/M6sJ0H1PlYk, AI Explorers 🎮 👋 Interactive game Tutorial - MoonMakers
CAPTION: This tutorial video includes the most important details about how to assemble the game code. Source: MoonMakers, https://moonmakers.org/

You will start by coding the “slicer” sprite. Click on the blue cat icon at the bottom of the Sprite Panel, then pick the “paint” option. In the Costume window that opens, make a yellow dot (or whatever shape you like). Name it “slicer.” You can get rid of any other sprites by clicking on the “x” in the Sprite Panel. This dot will move around with your finger, so you can track your progress on the screen.



Create your slicer sprite by first selecting the paint option in the Sprite Panel and then drawing a dot in Costumes.

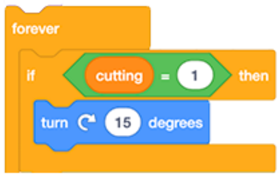
Coding The Slicer Sprite



```
when 🏳 clicked
set pen size to 11
set pen color to
pen down
set size to 10 %
show
```

For this game, you will use a special feature called a **[pen](#)**, which allows the sprite to draw shapes on the screen. To tell the pen what to do you need some code blocks. Find the “when green flag clicked” block. This will tell the game to start. Drag it from the code library into the code area. Next, you want to set the pen size to 11, make the color white, and activate it. Find the “set pen size,” “set pen color” and “pen down” blocks in the code library and drag them to the code area, too. Finally, add the “set size.” Set the size of the object to 10 percent and display it using the “show” block. This will also help you track the movement of your finger on the screen. Remember you can change any of the settings if you like. If you prefer a purple pen line, go for it!

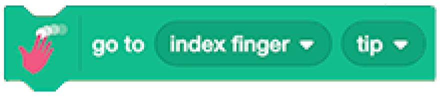
The next step is to create a forever loop and put a condition inside it that checks the value of the “cutting” variable. When your fingertip “touches” one of the random items on the screen, the slicer sprite “cuts” it, making it disappear. The cutting variable not only rotates the sprite by 15 degrees when it intersects with an object but also changes the figure of the sprite itself. This feature adds a visually engaging element to the gameplay experience, enhancing the aesthetics of the game.


```
forever
    if cutting = 1 then
        turn ↻ 15 degrees
```

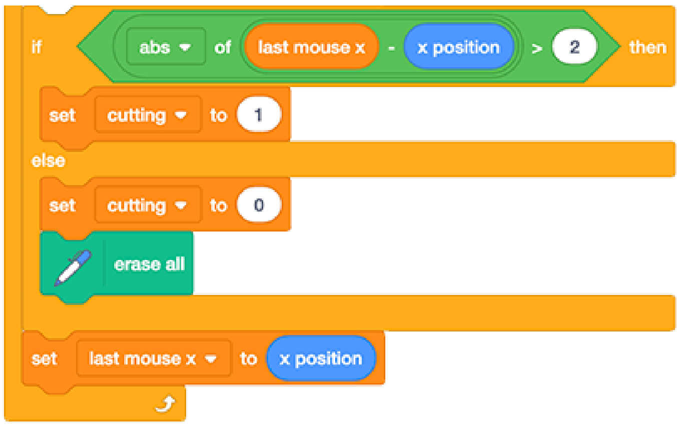
Since this is an action that you want to have happen over and over again, you need a special block. A **loop** is an instruction for the computer that repeats—in this case forever. Drag the “forever” block on the code area and add it to the blocks of your code. Anything inside that loop will continue to happen for as long as the game is running.

A **variable** is a value that can change depending on what information it receives. To create a variable, click on the “Variables” section of the code library and select “Make a Variable.” Make a variable called “cutting.” In this case, *if* the variable cutting is equal to 1, *then* the pen will turn by 15 degrees, as part of the movement of the game, and visual reaction of the cut, this when our fingers touch any of the elements on the screen, to make them disappear. Add an “If-then” block with a condition that uses an operator block to check the value of the cutting variable.

It’s time to activate the AI! Add the “go to” block to the end of your code and select “index finger” and “tip.” Your code will detect the location of your hand, specifically the tip of your index finger.


```
go to index finger ▾ tip ▾
```

Next, you have a condition that takes the **absolute value** of an **operation**. The operation tracks the **coordinates** of your fingertip on the screen. The "X" value represents your finger's horizontal position. In this operation, you will calculate the difference between the current—”X position”—and previous X position, labeled “last mouse X”—and make sure that difference is a positive number (abs). If the difference is bigger than 2, it means the object has moved a lot from its original position. The computer can make this calculation very, very quickly. Use the “if-then-else” block to help the computer make this decision. Note also that you will need to make the variable “last mouse X.”

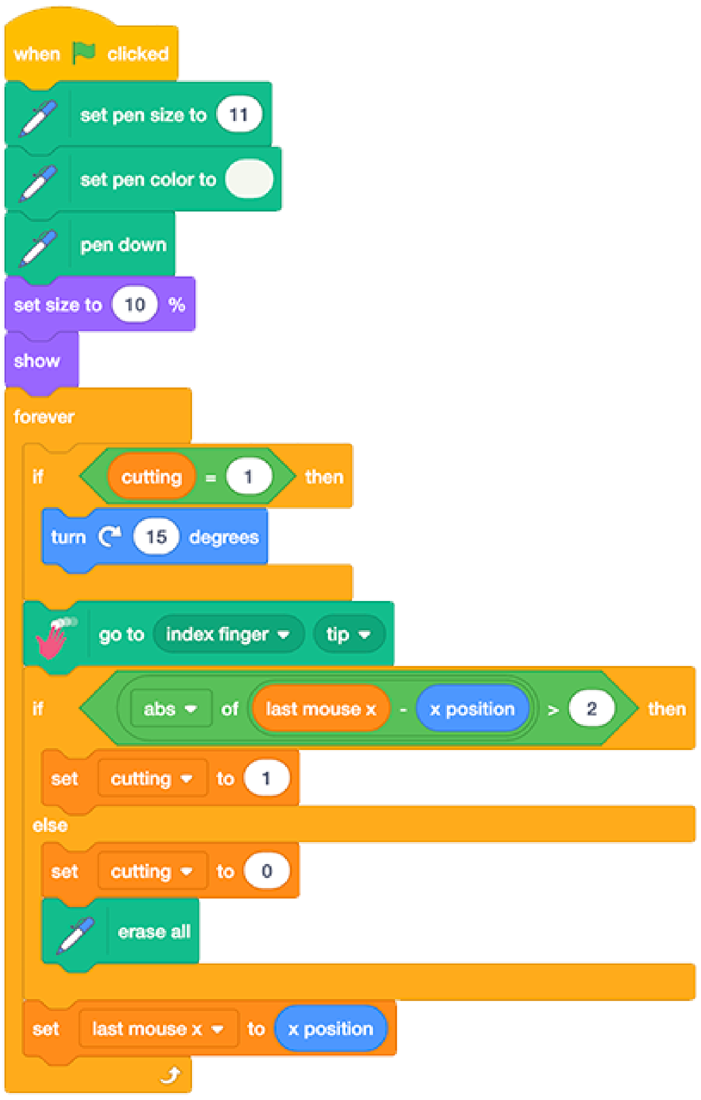


```
if abs of (last mouse x - x position) > 2 then
    set cutting to 1
else
    set cutting to 0
    erase all
set last mouse x to x position
```

If your hand is moving fast, *then* the variable “cutting” is set to 1, which will cause the pen to turn 15 degrees, so that the line is visibly drawn. *If* the difference between past and present pen positions is small, *then* "cutting" is set to 0. Since you are not moving, the pen does not need to turn and all its marks are erased. For the purpose of the game, when you aren’t moving your finger, the computer will assume you have touched the moving object successfully.

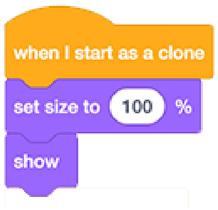
The last step is to set the “last mouse X” variable to match the X coordinates of the current position of your finger. Then the process loops and starts all over again.

After you’ve completed these steps, all the blocks should look like this:



```
when [flag] clicked
set pen size to 11
set pen color to
pen down
set size to 10 %
show
forever
    if cutting = 1 then
        turn ⟳ 15 degrees
    go to index finger ▾ tip ▾
    if abs ▾ of last mouse x - x position > 2 then
        set cutting ▾ to 1
    else
        set cutting ▾ to 0
        erase all
    set last mouse x ▾ to x position
```

Create A Slicer Clone

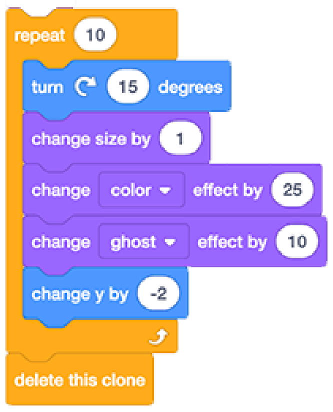


```
when I start as a clone
set size to 100 %
show
```

Now you'll address the second part of the slicer code. The goal here is to create a **[clone](#)**, or copy of the original sprite. The clone's job is to do something cool right after we perform a cutting action in the game—when your fingertip touches an object and makes it disappear. The slicer sprite will display a flashing animation, adding fun and more interaction to the game.

To define a clone, first set the size of the object to 100 percent, and display it.

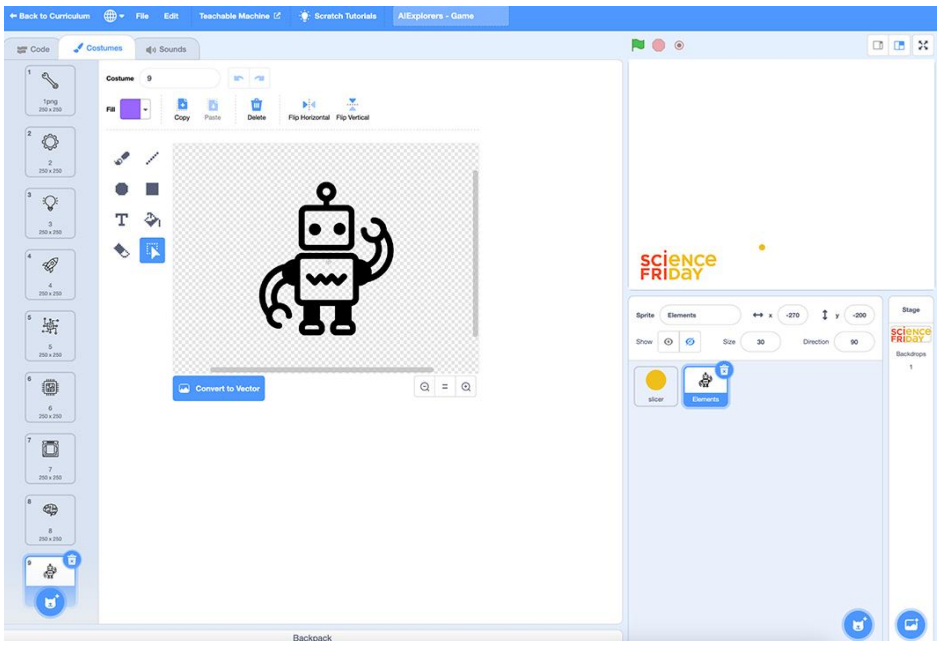
Then create a loop that will repeat 10 times. Set your code to rotate 15 degrees, resize by 1, and add a color effect and a fade effect. Change the position in Y to -2. This will create a colorful flashing effect when our fingertip touches one of the graphic elements in the game. Finally, add "delete the clone" when the animation is done. Play with the settings to create just the type of flashing animation you like!



```
repeat 10
    turn ↻ 15 degrees
    change size by 1
    change color ▾ effect by 25
    change ghost ▾ effect by 10
    change y by -2
delete this clone
```

Add Elements Sprites To The Game

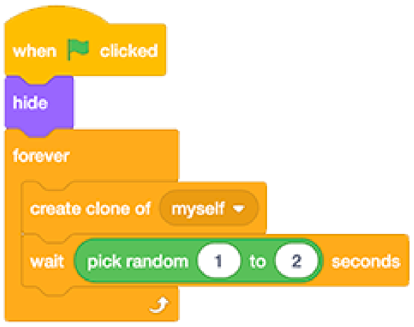
Now, you will code the "elements" sprites. These images will randomly appear and move across the screen in the game. You will need to "touch" them with your fingertip, which will trigger the slicer animation you made in the first step.



Notice that on the left side, there is a menu showing all the costumes for the elements sprite. Source: MoonMakers.

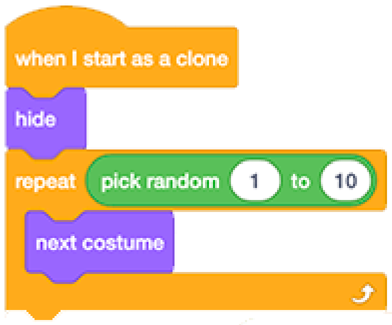
Once again you need to add a new sprite, just as you did before. It will be called "elements." However, this time, you want to have several different objects that can appear. Use the game graphics provided (see "[Get to Know the PoseBlock Platform](#)"), draw your own, or select sprites from the library. Click on the Costumes tab to see them. The most important thing is that every object must be a costume under the same “elements” sprite, as shown.

Click on the elements sprite to add a block to the Code workspace. When starting the game (clicking the green flag), hide the object and create a loop forever. Inside the loop, create a clone of “myself” by clicking on the downward arrow. Then add the wait block and set it for one to two seconds. This will generate a sprite at a random time between one and two seconds, so that you can’t guess when it will appear. This will continue for as long as the game is running.



```
when [flag] clicked
hide
forever
    create clone of (myself ▾)
    wait (pick random (1) to (2)) seconds
```

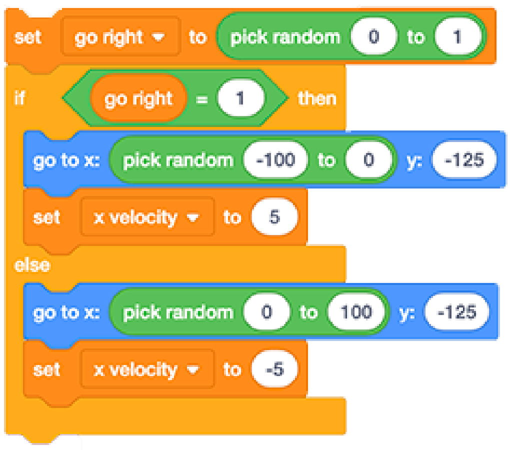
Once the clone is created, we want to select a random element sprite costume. Use the “When I start as a clone” block to begin the code. Then add a hidden block and a repeat loop. Add the “next costume” block to the loop and have it randomly pick a number from 1-10 using an operator block. This will randomize which elements sprite costume will pop up.



```
when I start as a clone
hide
repeat (pick random (1) to (10))
    next costume
```

Make The Elements Sprites Move

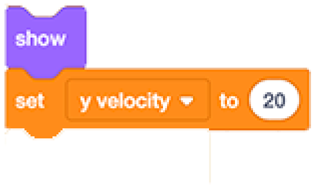
Now we need to give our elements something to do on screen. Add a variable and set it to "go right.” Use the same randomized code to set the variable to a random value between 1 and 0. Depending on the value, the elements sprite will move to the right or to the left on the screen.



```
set (go right ▾) to (pick random (0) to (1))
if <(go right) = (1)> then
    go to x: (pick random (-100) to (0)) y: (-125)
    set (x velocity ▾) to (5)
else
    go to x: (pick random (0) to (100)) y: (-125)
    set (x velocity ▾) to (-5)
```

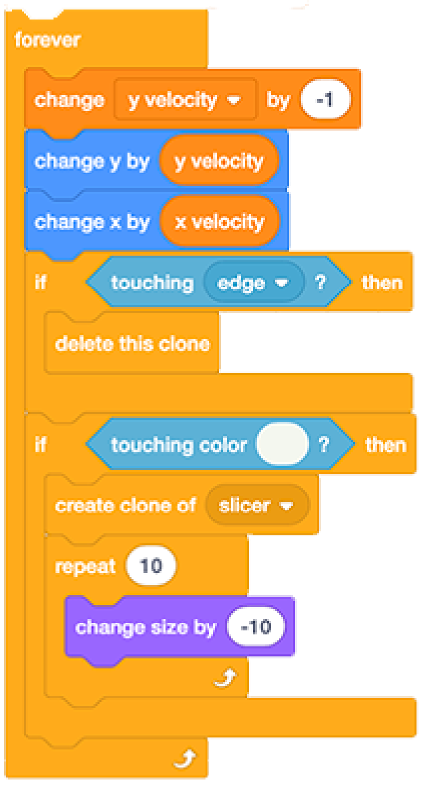
With the help of the “X velocity" variable, a positive value makes it move in one direction (right), while a negative value makes it move in the opposite (left). The value of the **velocity** determines the object's speed. Essentially, the velocity variable lets us control how the object moves within the game. Try changing the values to see how it affects the movement of the elements.

Now it's time for the element to show up on screen. Use the "show" block to do this. Set the velocity (speed) variable to 20 in the Y direction—which controls movement up and down—by creating a "Y velocity" variable.



```
show
set y velocity to 20
```

Now you need to code what happens to the elements sprite when it meets the slicer. Within a forever loop, change the Y velocity variable by -1. Change the position of Y by the variable "Y velocity," and do the same with the position in X. This will cause the element sprite to randomly pop up from the bottom of the Stage and then drop again. Then, create a condition that checks if the element sprite touches the border of the Stage. If it does, delete that clone.

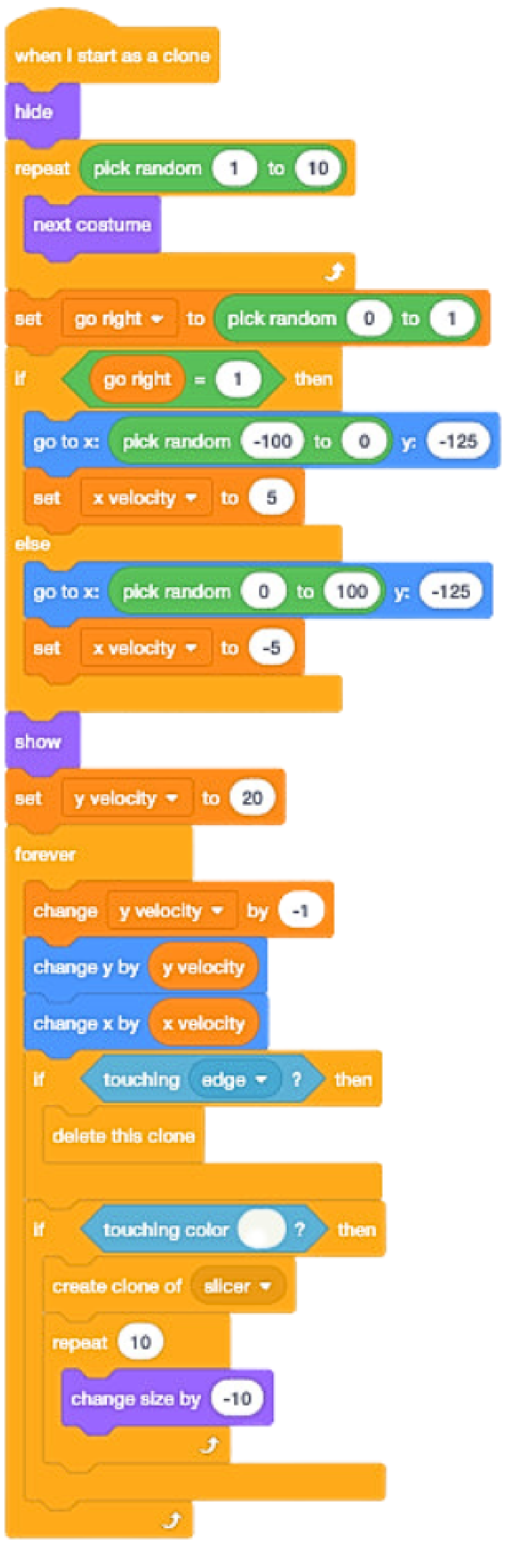


```
forever
    change y velocity by -1
    change y by (y velocity)
    change x by (x velocity)
    if <touching edge ?> then
        delete this clone
    if <touching color ( ) ?> then
        create clone of slicer
        repeat 10
            change size by -10
```

Next, create a condition so that when the element sprite is touched by your finger—the white colored pen of the slicer spite—a clone of the slicer is made. This will park the animation we coded previously. Finally, add a repeat loop and set the "change the size" block to -10. This will cause the element sprite to shrink until it's very tiny.

This is a good time to save your game. Unfortunately, PoseBlocks does not have a way to save and share your program online between sessions. Instead, you will need to save the file. In the toolbar, enter the name of your game in the white box. Then click on "File" and select "Save to your computer." To work on the game another time, just load the game file to PoseBlock in the same you did to upload other files as described above. Be sure to save after each session. You can send your game files to anyone, so they can play as well!

After you've completed these steps, all the blocks should look like this:



```
when I start as a clone
hide
repeat (pick random (1) to (10))
    next costume
set [go right ▾] to (pick random (0) to (1))
if <(go right) = (1)> then
    go to x: (pick random (-100) to (0)) y: (-125)
    set [x velocity ▾] to (5)
else
    go to x: (pick random (0) to (100)) y: (-125)
    set [x velocity ▾] to (-5)
show
set [y velocity ▾] to (20)
forever
    change [y velocity ▾] by (-1)
    change y by (y velocity)
    change x by (x velocity)
    if <touching (edge ▾) ?> then
        delete this clone
    if <touching color ( ) ?> then
        create clone of (slicer ▾)
        repeat (10)
            change size by (-10)
```