PLTW COMPUTER SCIENCE

Activity 1.1.4

Objects and Methods

Introduction

One part of a program often greatly affects another part of the program. Clicking a button in one part of the screen can make words in another part of the screen appear or change style. There are even examples of one program telling another program what to do. An example occurs when you drag words from a Word document into an Excel spreadsheet to copy them. How does one program or one part of a program tell another part of a program what to do?

Materials

- Computer with microphone and camera (built-in or external)
- Scratch™ 2 Offline Editor with source files or Scratch account and web page http://scratch.mit.edu/projects/22638588/

Resources

1.1.4 sourceFiles.zip

Procedure

One part of a program often tells another part of the program what to do. Programmers describe the "what to do" as a **method**. One part of a program "calls a method on an object." In Scratch the **objects** are the sprites, and you will be "**calling a method on the object**," or telling the sprite what to do.

In Scratch each sprite and the stage have their own script area. Every script belongs to one sprite or to the stage. A script can use the `Change costume` or `Move` blocks to affect the sprite's own costume or the sprite's own (x, y) location, but a script of one sprite can't change the location or costume of another sprite.

If you (the programmer) want one sprite to make another sprite move or change costumes, you use a message. Scratch messages are **broadcast** to all sprites. If several sprites all respond to a message, the program can be difficult to debug.

When we broadcast a message, it helps if we name the message following this convention:

- name the target sprite, and
- use a verb with empty parentheses.

These will be separated by the decimal point, which "points" the verb at the sprite. For example if you want a button sprite's script to tell the `face` sprite to `turnLeft`, broadcast the message `face.turnLeft()`. Even though all sprites will receive the broadcast (that's why they call it *broadcast*), anyone looking at the code will know that the sprite named `face` is supposed to be the recipient. Naming your Scratch broadcasts as **target.verb()** will make it easier for other programmers to understand your program. It will also make it easier for you to figure out what to change when a program isn't working.

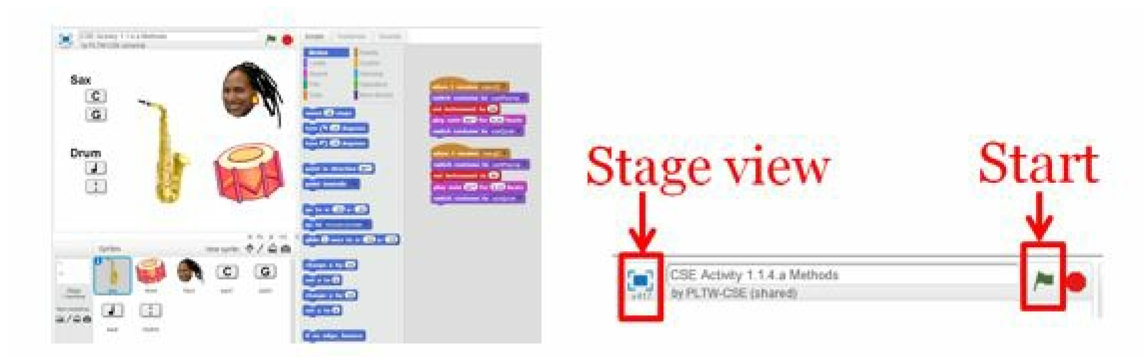
We are using the parentheses at the end of the message to make Scratch similar to other languages which often utilize the parentheses to send additional information.

1. Form pairs as instructed by your teacher. The professional greeting is an extremely important part of the professional skills included in this curriculum. No matter how/if you greeted your partner previously, take a moment and practice your introductory greeting with your partner. A professional greeting includes:
    - squaring your shoulders and knees with the person you are greeting and using good posture
    - making eye contact
    - smiling
    - using an enthusiastic, interested voice
    - saying something pleasant like "Nice to meet you," or "I am glad to be working with you," or "I look forward to working with you"

2. Open the Scratch project `1.1.4 Methods.sb2` or log in to your Scratch account and open the project http://scratch.mit.edu/projects/22638588/

3. Every project contains the stage and at least one sprite. How many sprites does this project contain?

4. Switch to **stage view** and click the **green flag**.



5. Click on each of the sprites. What happens in each case?

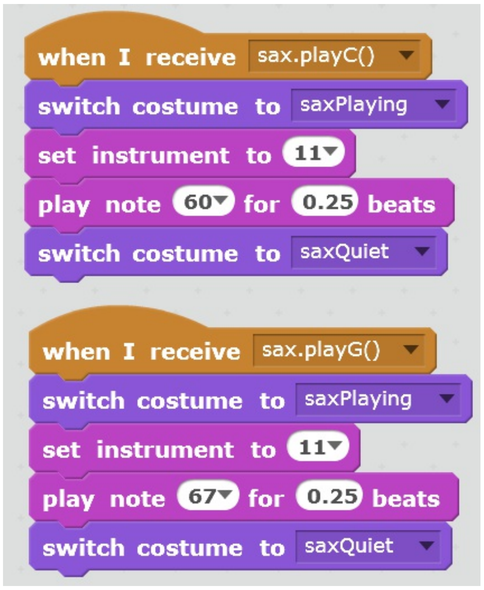
6. There is an important difference between an object doing something to itself and an object accepting commands from another object.

    - Which sprites make themselves move or change appearance when clicked?

    - Which sprites make another sprite move or change appearance when clicked?

7. The stage and each of the sprites has its own scripts areas. The `sax` sprite, for example, has two scripts, as shown at left below. Make a list of all the sprites and indicate how many scripts belong to each sprite. The list is started for you below on the right.

Scripts of `sax` sprite	List all sprites: `sax` has 2 scripts `drum` has ____ scripts



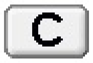
```
when I receive sax.playC()
switch costume to saxPlaying
set instrument to 11
play note 60 for 0.25 beats
switch costume to saxQuiet

when I receive sax.playG()
switch costume to saxPlaying
set instrument to 11
play note 67 for 0.25 beats
switch costume to saxQuiet
```

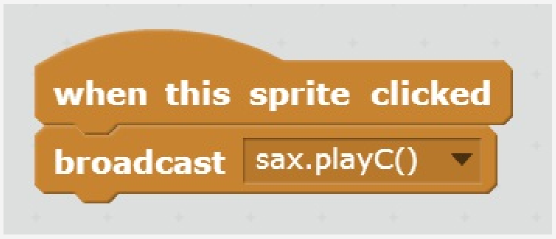
8. The sprites you listed in Step 6b all make another sprite perform an action by broadcasting a message. The message name follows the target.method() naming convention.

   For example, when the sprite `saxC` is clicked (that's one event), it creates another **event** by broadcasting the message `sax.playC`, as shown below on the left. This message event is **handled** by the `sax` sprite, and it has a script that receives the `sax.playC()` message, as shown below on the right.

   That handler script makes colored lines appear on the sax, plays note 60 (that's the musical note C) of instrument 11 (that's a synthesized saxophone) for 0.25 beats, and then returns to the costume without the colored lines.

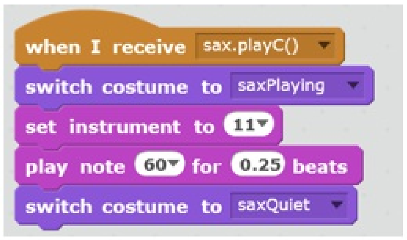
`saxC` sprite





```
when this sprite clicked
broadcast sax.playC()
```

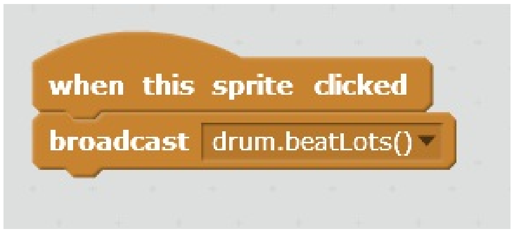
`sax` sprite



```
when I receive sax.playC()
switch costume to saxPlaying
set instrument to 11
play note 60 for 0.25 beats
switch costume to saxQuiet
```

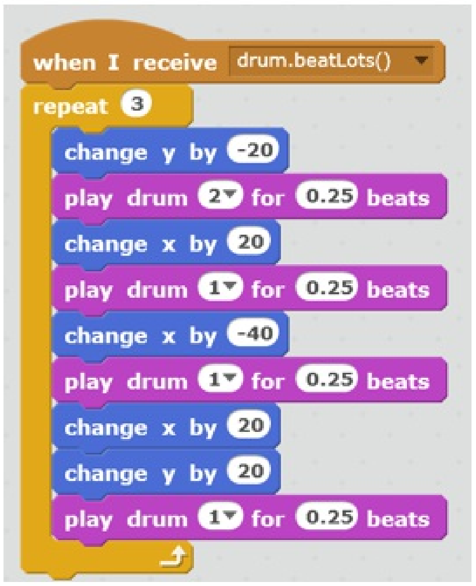
Following the example of the prose above, describe the two events and the two event handlers involved in responding to a click on the `rhythm` sprite.

`rhythm` sprite



```
when this sprite clicked
broadcast drum.beatLots()
```

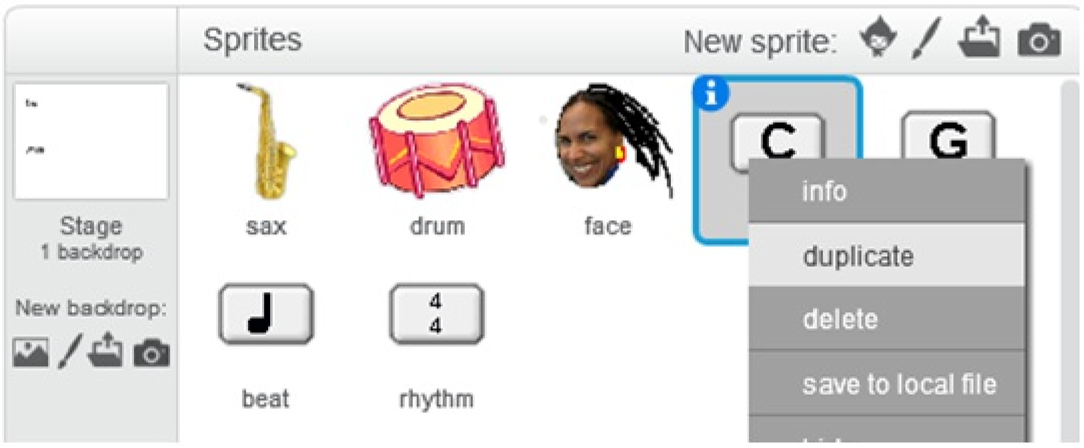
`drum` sprite



```
when I receive drum.beatLots()
repeat 3
    change y by -20
    play drum 2 for 0.25 beats
    change x by 20
    play drum 1 for 0.25 beats
    change x by -40
    play drum 1 for 0.25 beats
    change x by 20
    change y by 20
    play drum 1 for 0.25 beats
```

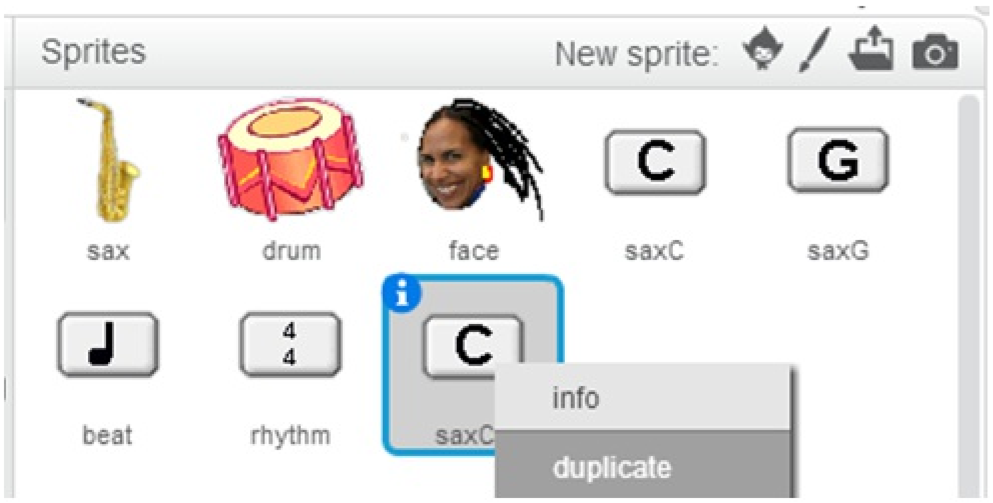
9. Identify another pair of scripts where one script creates a message event and another script handles the message event. Describe the pair.

10. Create a new sprite that looks like a button. Do this by right-clicking one of the existing button sprites and selecting **duplicate**.

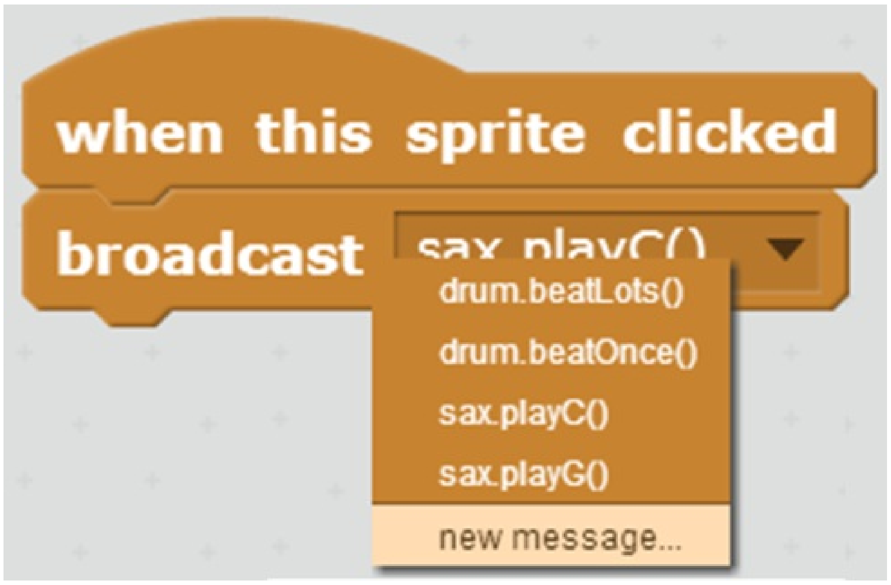


11. Brainstorm ideas for what you want the button to do. The action should be something that one of the three existing sprites will do. Describe your idea here. You will create a new feature of the Scratch project by making this button work properly.

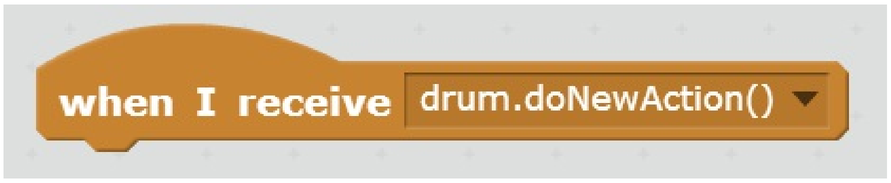
12. Right-click the new sprite in the sprite list and change the name of the sprite to describe what it does.



13. When you duplicated an existing sprite to make your new sprite, the new sprite inherited a copy of the existing sprite's scripts. You can see the script(s) in the script area. That script includes a broadcast block. Select the dropdown arrow for the broadcast message, and select **new**. Name the new broadcast message following the target.action() convention.



14. Create a new script in the target sprite's script area. Begin the script with the when I receive block.



15. As pair programmers, strategize, code, and test, iterating (which means repeating as necessary) until the new feature works as you want it to. Remember to switch driver/navigator roles frequently.

16. Save your Scratch project as directed by your teacher. Whatever the storage scheme, each partner should have access to the work completed.

17. Teams become much more effective when each team member takes time to reflect on the process. The team will discuss everyone's reflection and then reflect on the collaboration. What helped you work effectively together? What could you do to make the collaboration more effective? Discuss each of your thoughts on these questions. After reflecting individually and then discussing as a pair, write down your reflective thoughts individually.

Conclusion

1. Scratch broadcasts a message to all sprites. This is just how Scratch is set up as a programming language. We are using messages so that they are targeted to a particular sprite. In a large project with hundreds of sprites and scripts, explain why it might help to follow our convention that only one sprite will respond to a message and to follow our naming convention for messages.
2. Based on what you learned during this activity, explain what you think **events** and **handlers** are.
3. As computer programs get big and complicated, it is essential to manage the complexity. An important strategy for managing complexity is called encapsulation. With encapsulation some details of one part of a program are hidden from other parts of the program. In cars, for example, the details of the accelerator pedal's actions are encapsulated under the hood. As the user of the accelerator pedal, the driver only gets to press the pedal and is not encouraged to tinker with the internal mechanisms that make the pedal do what the engineer intended.

    - What are some details of the `playG()` method that are encapsulated by the handler?
    - A software developer can make it easier for other developers to tinker with the details of the program she is writing by exposing the details. We'll get into what this means later. For now, just consider: What details mentioned in part a might be useful to expose? In other words, what variations on `playG()` do you think would be convenient to allow?