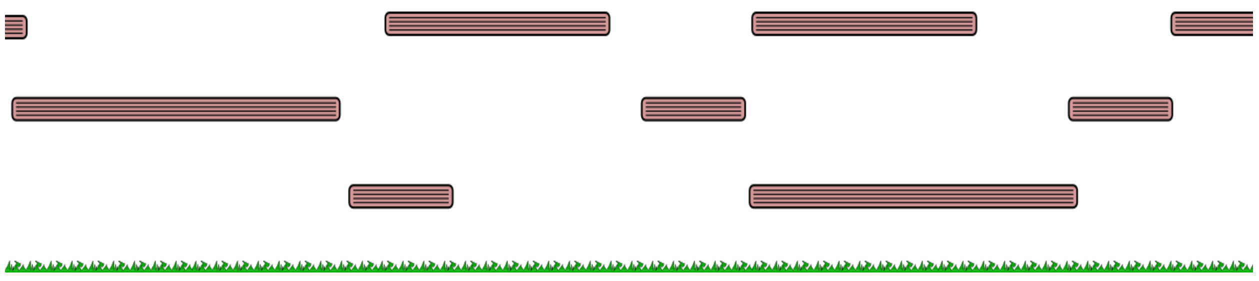
How to Make a Side-Scrolling Game

This is a tutorial on how to create a side-scrolling game in BYOB. In a side scrolling game, the background slides left and right behind a character as it progresses through the game. However, in BYOB backgrounds are fixed and cannot be moved, which creates a challenge. So while we think of this as a scrolling background we actually use sprites as the background, making sure they are in the bottom layer so that characters will appear on top of the background sprites. Sprites can be moved around and even placed partially off the stage, which is exactly what we need to create a side-scrolling game.

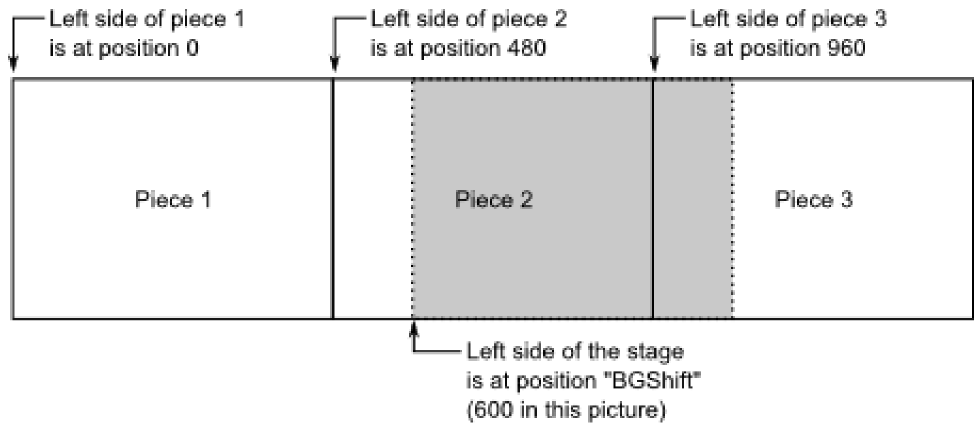
The description below shows how to make a basic background that can be scrolled left and right. If you are the kind of person that likes to work along with these write-ups in BYOB, don't. Just read the description and figure out how it works. When it comes to actually building this in BYOB there are a couple of things that make it easier, and are easier to describe after the basics are described, so just the "Basics" section and don't start trying things in BYOB until you understand all of that section and reach the "Build It" section.

Basics

The first step is to create the background. The stage is 480 pixels wide by 360 pixels high, so it is best to make a background that is 360 pixels high and some multiple of 480 pixels wide. You can use whatever drawing program you like, but make sure the dimensions of your finished background picture follow these guidelines. We'll make a background that is three stages wide, and so create the following 1440 pixel wide image:



Next, we divide this up into three pieces, each 480 pixels wide. When we load these in as sprite costumes, we'll set the "center" location of the sprite to be the lower left corner - no, that's not a "center" in any sense of the word, but it's a good reference point for what we're doing. Here's the idea: the stage is a "window" into the larger image, and the part that we can see contains at most two of the 480 pixel wide pieces. This picture shows the basics: our picture is divided into three pieces, and when viewed as one big picture each piece starts at a multiple of 480. We use a variable named "BGShift" to indicate the position of the stage, so in this picture valid values of BGShift are 0 through 960, inclusive.

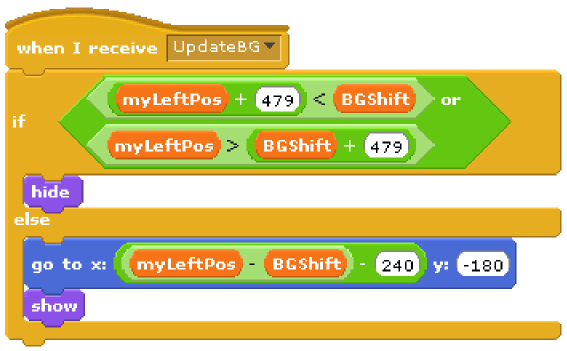


There are several ways to implement a side-scrolling background. The approach we describe is to have a script associated with each "piece" sprite that runs when it receives a signal and decides if and how to display itself. Lets use a sprite-local variable named "myLeftPos" to indicate the position of the leftmost column in the piece (so the value of "myLeftPos" for the first piece is 0, and the value of "myLeftPos" for the second piece is 480, etc.). Under what conditions is the piece off the stage and hence not seen? First, it is off the screen if the rightmost pixel in the piece is strictly to the left of BGShift (in other words, if myLeftPos+479 < BGShift). Second, it's off the screen if the leftmost pixel is greater than BGShift+479, the rightmost position of the stage (in other words, if myLeftPos > BGShift+479). Writing all of this out as a Boolean expression, a piece is not visible if

(myLeftPos+479 < BGShift) or (myLeftPos > BGShift+479).

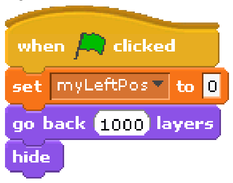
Using this Boolean expression, we can make each of the background pieces decide whether to display itself or not. If the expression is true, then the piece is invisible, so all it has to do is make sure it's not visible by using the "hide" block. What if it is visible? In that case we need to decide how to position the block, and make sure it is visible. Recall that we set the "center" of each piece to be the lower left hand corner of the piece, so we need to figure out what the coordinates of the lower left hand corner are, relative to the stage coordinates. The x coordinate of the lower left corner of the piece is offset by myLeftPos-BGShift from the stage lower-left corner, which is at position (-240,-180). Therefore, the coordinates of the lower-left corner of the piece are ((myLeftPos-BGShift)-240,-180), and we can use this in a "goto x y" block.

Let's put all these pieces together. When we receive a signal saying the background needs to be updated (we'll name this signal "UpdateBG"), we first test to see if the piece is visible. If it's not, we hide the piece; otherwise, we set the position as just described, and show the piece. Here's the entire script:



```
when I receive UpdateBG
if <<((myLeftPos) + (479)) < (BGShift)> or <(myLeftPos) > ((BGShift) + (479))>>
    hide
else
    go to x: (((myLeftPos) - (BGShift)) - (240)) y: (-180)
    show
```

We also need a script that will initialize each piece - we'll set it up so that when the program starts (when the green flag is clicked), it sets its "myLeftPos" variable moves back in layers as far as possible (moving back 1000 layers should be far enough), and hides itself. Here is the initialization script for each background piece:



```
when flag clicked
set myLeftPos to 0
go back 1000 layers
hide
```

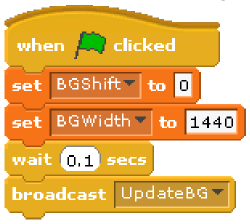
Hopefully you understand these scripts, which are for individual background pieces. Now it's time to put some of this together!

Build It

The first step in building this example program is to set up the three background sprite pieces. It's tempting to start by creating three sprites with the appropriate backgrounds and then build these scripts, but you can save yourself some work if you're careful about the order you do things in. So to start, use the "Choose new sprite from file" button in the sprites area to create one new sprite, and use your left-most background piece as the file to load. Next, go into the costumes for this sprite and hit "Edit" to bring up the costume editor. You need to change the "center" of the sprite, which is easier if you can see the whole thing, so zoom out in the "Paint Editor" (the magnifying glass with the minus sign) until you can see the whole thing and then click the plus sign next to "Set costume center" to bring up the center positioning cross-hairs - just grab that with the mouse and drag it all the way down to the lower left hand corner. When you've done that, hit "OK" and your costume is all set up. Finally, it's good to rename your script to keep it straight - name it something like "BG1" for the first background piece - and then build the two scripts that were described in the last section. Now you have a single background sprite, with the correct center location specified, and the scripts that allow it to be initialized and displayed properly.

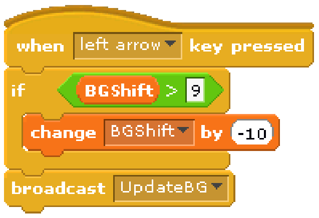
By doing that all for a single background sprite first, now you can just duplicate the sprite (right click on the sprite in the sprites area) to copy that full sprite, scripts and all. Do that twice. For each new sprite you'll need to go into costumes, import the appropriate costume and edit the costume to set the center point. You will also need to change the "myLeftPos" value in the initialization script so that it is the correct value for each background piece. At this point, if you used three background pieces, you should have three sprites, each with the display and initialization scripts, and each one with a different costume and “myLeftPos”. You should be able to set the variable “BGShift” and broadcast the “UpdateBG” signal to draw the background in any position that you would like. Next, let’s make a “driver” function.

Since this driver isn’t associated with any of the background sprites, we will put this code in the “Scripts” area of the background. If you are writing a game which is “driven” by a player’s character you could just as easily control the background display from that character’s scripts. The first thing we need to do is initialize variables - this is plural because we are going to create a new variable named “BGWidth” to indicate how wide the overall background is, which in our case is 1440 pixels (480*3). While we could hard-code this number into our scripts, having it available as a variable makes it easier to change the background - adding a fourth BG piece will just involve cloning a background piece, setting the background costume and “myLeftPos” appropriately, and changing the BGWidth to reflect the larger background. After we initialize BGShift and BGWidth, we would like to draw the initial background. We can do this by broadcasting “UpdateBG” so that each piece figures out how to display itself, but there is a small issue with timing here: there are variables that each background piece needs to initialize, and we don’t want to broadcast this message before they are ready. For that reason, we put in a “wait” block to allow the pieces time to initialize. This is actually not the best solution, since we don’t really know how long the piece initialization takes, but it’s good enough for this example (can you think of a better solution?). This is the resulting initialization script:

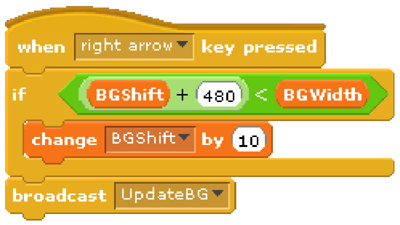


```
when [flag] clicked
set BGShift to 0
set BGWidth to 1440
wait 0.1 secs
broadcast UpdateBG
```

Finally, we want to scroll the background - we decrease the BGShift when the left arrow is pressed, and increase it when the right arrow is pressed. We want to make sure we stop when the edges of the background are reached, giving the following two scripts that respond to left and right arrow presses:



```
when left arrow key pressed
if BGShift > 9
    change BGShift by -10
broadcast UpdateBG
```



```
when right arrow key pressed
if BGShift + 480 < BGWidth
    change BGShift by 10
broadcast UpdateBG
```

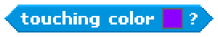
If you implement all of this, then you should have a simple program that scrolls this three-piece background left and right.

Some Enhancements to Consider

This is a very basic solution. It’s functional, and works pretty well, but if you were going to put some real time into this, there are a couple of things to consider improving:

- As explained above, synchronizing the initialization scripts is important, and it’s a little sloppy to put in a 0.1 second wait and just assume that this is enough. A more reliable solution would synchronize scripts so that you were guaranteed that everything was initialized before the first “broadcast UpdateBG” block was executed.

- The scrolling could be smoother. In particular, if you try this out and scroll right and left really fast, you might see flashing white spots between the background pieces. This happens because one piece is moved and redrawn before the other, and so there is a very brief time when there is a gap between the pieces. There are several ways to fix this, but probably the simplest is as follows: you can have the background pieces overlap by 10 pixels (the amount that pieces are moved) and ensure in the artwork that the 10 pixel overlap is consistent. By “consistent” I mean that if one piece moves before another you don’t get flashes of different colors - if you looked at it in slow motion it might look like the background “stretched out” a little when this happens, but when it happens fast you won’t notice this. Your eye will catch a very brief flash of a different color, but it will not detect a very brief stretching of the image.
- Reacting to the moving background: In a lot of games, a character will move along in front of the background, going over obstacles and that sort of thing. This can be challenging to program, and the best way to do it is to keep track of actual obstacle positions in a list that your program can use. While not as good a solution an alternative is to make sure your artwork outlines obstacles in a specific color - then you can detect when your character hits an obstacle by using the predicate that looks like this:

  ```
  touching color 🟪 ?
  ```

  There are other ways to deal with this, but this is probably the easiest to do quickly, and is particularly appropriate for going through a maze (where maze walls can use this color) or similar tasks.
- You can also “wrap around” the background for a continuous background - in other words, moving right from the rightmost piece brings the leftmost piece back in. For games in which you are exploring a repetitive area, and the game interest comes from other characters rather than the background, this can be very appropriate.

Doing This Yourself

If you want to experiment with these techniques, the three background pieces are available for download on the class web site. With those images, you should be able to pretty easily reconstruct the scripts described in this tutorial.