Scrolling Background

In this article, we will learn how to create a realistic scrolling background. This is useful for platform games such as Mario. I am going to take the program "scrolling-with-objects-3" from the book "Practice CS Concepts with Scratch" as the starting point and improve it further.

Concepts used in this program:

- Algorithms
- Arithmetic operators (abs)
- Conditions: YES/NO questions
- Conditionals (IF-Else)
- Costume-based animation
- Custom procedures
- Keyboard events – polling
- Random
- Relational operators (=, <, >)
- Simple looping (repeat, forever)
- Synchronization using broadcasting
- Variables - local/global scope
- Variables – numbers

Scrolling with objects-3

This original program (from the book mentioned above) has the following features:

- There are several background sprites, such as, buildings, trees, and clouds.
- There is a foreground sprite, such as, a person.
- When left/right arrow keys are pressed these sprites move horizontally, to create the impression that the person is walking, although the person stays at the same place just shuffling his/her feet.
- Some objects move slowly, some move faster, thus creating the impression of distance.

Click this link to see this original program: scrolling-bg-orig.sb2
https://scratch.mit.edu/projects/318882313/

This program has the following short-coming:

- As soon as a background sprite goes past one of the screen edges, it reappears at the opposite edge.

In this article we will fix this problem and have control on how far a sprite can go off the screen before it reappears.

Version 1:

Feature idea 1:

A background sprite should go past the screen edges and continue moving past the edges. It should snap to the other edge only after it exceeds some value which is well past the screen edge. For example: we may want a building to continue moving West until its X position is -600. Trying to move it beyond -600 would snap it to X=600.

Design:

Scratch does not allow a sprite to go completely past the screen edges. You can observe this by turning on the "x position" property and then moving the sprite along the X axis. The sprite gets stuck at some values. Running the "change x" command has no effect on the sprite when it is stuck.

For example, in our first version we will use a "tree" sprite as a background sprite. Turning on "x position" shows that this tree gets stuck at x=-308 and x=308. (We will record these Scratch limits in a private variable called "edgex" as discussed below.)

So, we need to come up with some other idea to avoid getting stuck like this.

Here is one idea: instead of looking at the "x position" property to determine a sprite's X coordinate, we will have our own variable called "myx". When the program begins, myx will be initialized to x-position. Any attempt to move the sprite using the "change x" command would only change the value of "myx". Unlike x-position, myx has no constraints: it can go as low or high as we want. For instance, we may decide to let myx go all the way down to -600 and up to 600.

How would this help in controlling the actual screen position of the sprite?

This is what we can do:

1. We will save the Scratch screen limits (e.g. 308 for the tree sprite) in a private variable called "edgex". Thus every background sprite will have its own "edgex".
2. We will save the limits of how far a sprite can travel out of the screen area in a variable called "snapx". For instance, for the tree we may decide to have snapx=600.
3. As long as –edgex < myx < edgex, the sprite will remain visible and its X coordinate will track the value of myx.
4. As soon as "myx" exceeds the min/max limits imposed by "edgex", we will hide the sprite, which will create the impression that the sprite has left the screen. The sprite will become visible as soon as myx comes back in range.
5. As soon as "myx" exceeds "snapx", we will snap the sprite to the other edge.

Here is the algorithm. Note that the sprite moves opposite to the arrow keys since it's a background sprite.

Main script:

```
Forever:
      If right arrow pressed
            Change myx by -5
            Position sprite
      End-if
      If left arrow pressed
            Change myx by 5
            Position sprite
      End-if
End-forever
```

"Position" algorithm:

```
If myx < -edgex OR myx > edgex
      Hide
Else
      Set x to myx
      Show
End-if
If myx < -snapx
      Set myx = snapx
End-if
If myx > snapx
      Set myx = -snapx
End-if
```

NOTE: Due to a bug in Scratch, when you "hide" a sprite, the "key press" event goes rather wild, i.e. the sprite gets a ton of "key press" events when though you press the key just slightly. To circumvent this bug, we will use the "ghost" effect to emulate hide/show.

Solution: scrolling-bg-1.sb2
https://scratch.mit.edu/projects/319447054/

Version 2:

Feature idea 2:

Add more background sprites and arrange to have a different background every time.

Design:

We will add more sprites to fill up the background. These sprites will have identical code, except, of course, each sprite will have its own "edgex" and "snapx". To make the program a bit more interesting, we will set these values randomly and pick the initial position of each sprite randomly, so that every time we run the program, the background sprites will be arranged in a different order. Finally, we will create a new custom block called "Position" that decides how

the sprite is to be positioned on the screen. Custom blocks are more responsive that broadcast scripts.

Feature idea 3:

Have a "speed" slider variable for the foreground sprite (walking person) using which you can create the illusion that the person can walk at different speeds.

Design:

This can be achieved using the "speed" variable as a multiplier for all background sprites. The main script will change as shown below:

Main script:

```
Forever:
      If right arrow pressed
            Change myx by -5 * speed
            Position sprite
      End-if
      If left arrow pressed
            Change myx by 5 * speed
            Position sprite
      End-if
End-forever
```

Solution: scrolling-bg-final.sb2
https://scratch.mit.edu/projects/319447401/

Author: Abhay B. Joshi (abjoshi@yahoo.com)
Last updated: 2 July 2019