John Hibbert / CSC402 / 11/13/2015

Final Project Report

Introduction:

My language, which for a lack of creativity I am calling "anim", is a very small language that takes ASCII images and creates a Windows Batch Files (.bat) animation based on it. One application of this would be to create a fancy 'splash screen' for a batch script. For instance, if you had a script that did some sort of business critical effect, you would have it introduce itself with a fancy opening screen to catch the user's attention. It is entirely aesthetic; there is not much practical purpose to this, except insofar as this removes the vast majority of the labor in creating these animations.

Implementation:

The implementation is fairly simple. We grab a few keywords from the language: at this point, they are only "type" and "method", as well as "speed" as an optional parameter for "method". We store those in the visitor class in variables. Then, we pattern match the series of strings that make up the ascii image, and store them in order in an array of strings. Then, we create a queue of those arrays of strings, with each successive item being the next step in the animation. In this process, we use switch statements to compare to the "type" and "method" strings to know what specifically to do to the original array of strings to make the new element in the queue into the next step in the animation.

For all of the types of animations, the basic steps are the same. We have a counter that starts at one and goes up each time through a while loop. In the while loop, there is a switch statement, which chooses based on the "type" string variable. That calls a submethod that is particular to that animation, and passes what it needs, which is usually a copy of the original image and the counter to know what image it is. Some also need to know the longest length among strings in the original. Each of those submethods will take the original and the counter number and iterate through the strings in that array and alter them so that becomes the current step in the process. Once that process returns a new array to the queue of arrays, we peek at the new addition to check some criteria where we can tell if the image has been fully displayed. If that is the case, we get ready to escape the while loop. Then we increment the counter and go back to the top of the while loop. Once that is done, we use another method to insert the escape characters that Windows Batch files need to properly display certain characters.

Once we have the queue of arrays of strings finished, we simply dequeue the first element, iterate through the array and concatenate its contents into a string we call 'code,' along with the small amounts of syntax needed that was not already added in. Mostly, this is piece of code that tells the program to wait, chosen by the 'method' field, and a command to clear the screen.

Some particulars:

There are eight types implemented so far:
Upwipe, Downwipe, Rightwipe, Leftwipe: These mimic a 'film wipe' where the image is still but more and more of it become visible with each step. The directions indicate which way the visible edge is 'travelling.'
Upslide, Downslide, Rightslide, Leftslide: These involve the image 'moving' across the screen until it is in place. The directions indicate which way the image is 'moving.'

John Hibbert / CSC402 / 11/13/2015

There are two methods implemented so far:
Ping: Between drawing an image and clearing the screen for the next image, this uses the ping command to tell the batch file to ping a non-existent and wait a specified number of milliseconds. If a speed is given, that is the number of milliseconds.
Funct: Between drawing an image and clearing the screen for the next image, the file calls a function that the batch file will create at the end of the file. All it does is count down from the specified number. If a speed is given, that is the number from which it will count down.

Windows Batch scripting does have commands for 'pause' and 'sleep', but both of them only work in entire seconds, which is way too slow for animations that could have more than 60 frames. That is why we jump through all these hoops.

The reason for having a choice of methods is that I noticed that the ping command would occasionally not wait as I expected it to. I believe it was because my wireless was not working at the time, and the program was 'smart' enough to not bother to try and ping when it had no connection. So, I have another method that works, but it is less exact because we are not working in milliseconds. Also, in my very late testing, the 'funct' ones would jump forward and stutter, even though the actual data was complete in the file. I could not discern why it was happening, though it appeared to be somewhat consistent. This is probably why everyone online uses ping when talking about these animations, since it seems to work as expected most of the time.

Challenges:

The first main challenge I faced was finding a way to build the data structure I wanted: a queue of arrays of strings. As it turns out, Queue is an interface, so I had to use something that implemented it. I eventually went with a LinkedList object, which has all the functionality I needed.

The next big challenge was to work out ways to coax the data into that data structure in the form that I wanted. Basically, I had to think of a simple algorithm that would match what was supposed to happen on screen for each type of animation I wanted to implement.

Some of them were simple, such as DownWipe: all you have to do is include the elements that are equal or less than counter and force the others to be blank.

Some of the others were quite challenging, especially considering how a string must print from left to right. So, to get it to slide in from the right, we have to include a huge number of spaces in front of the image. So, just concatenate 79-C spaces in front of the line, right? Well, the problem is that the line can't be over 79 characters or it will go onto the next line and mess up the image. So we also have to only include the substring that would be visible. But you also can't include a substring where the length is less than c, or you will have a StringIndexOutOfRange exception. It is a huge nuisance, especially when dealing with a length (which starts at 1) in a way that must be comparable to an array (which starts at 0). Hence, there are some places where I had to throw in a 1 to cause it to stop complaining.

Another challenge was caused by the fact that I couldn't directly test the animations inside the Linux machine. I just had to eyeball it, so the examples I used were small enough that it was possible to check with just a look at the terminal output. I also couldn't get shared folders to work, so I had to plug in a pen drive, tell the VM to grab it, drag the files, unmount it, unplug it, replug it in, let windows grab it, and then drag the files out. This lead me to wait to test in depth until late in the process, which caused a few bugs to be missed from the comparatively simple test files. Most of these related to mixing up the longest length and the index of the element with that length.

One more problem is that the first line of the command is still sometimes visible when it runs, but without knowing more about the the windows cmd, I can't get rid of that.

An unrelated challenge was determining how to know that I had done enough. The language is simple, but the blood and guts of it is pretty involved. I was originally just going to have the wipes, but then I thought of the slides, and figured that should have those eight and it would be a robust proof of concept.

Examples:

The outputs are too big to reproduce here, so I will simply comment on the examples I provided.

Test1 is a leftlside ping 25. It seems to work fine.

Test2 is a rightwipe funct that demonstrates the jumping problem with funct that I described above.

Test2a is the same thing as test2 with ping to demonstrate the difference. If you look inside them, they are almost identical (as far as the 'frames' go, at least), so the difference is just in the method.

Test3 is downslide funct, which doesn't seem to have the same problem as test2. I am unsure why, but I don't believe it is in the code, but something particular to batch files themselves.

Test4 is upslide ping. It shows that under the current implementation, the upslide starts from the bottom of the image and not the bottom of the screen. I suppose neither is necessarily preferable, but a future ability to specify would be good.

Conclusion:

I am pleased with what I managed to do here. I wish I was more familiar with Batch before doing this as I may have been able to suss out certain problems that were not coming from my Java. Still, the goal was to reduce the labor in this, and I feel I have done that.