Comp160
Project 1

Fall 2018

Project Overview

For your first programming project you'll be completing and adding to the car program as described in Section 3.6. If, at this point, you haven't read that section, then do so now. Exercises 39-42 and 44 work you through the baseline version of the program. You'll start this in lab. Once you've completed this version of the program you'll be required to add additional features. Each new feature opens up the possibility of a higher grade. Below you'll find descriptions of the additional features,the details of the grading system, and a calendar of important dates.

Version 0

The initial version of the program is what you end up with after completing exercises 39-43 and exercise 44. In it a car moves across the scene from left to right. Clicking the mouse in the scene warps the car back to the x coordinate at which the mouse was clicked.

Version 1

In version 0 the car disappears off the right side of the scene if you don't use the mouse click feature to warp the car back. Version 1 should make it so the car restarts at the left side of the scene after it goes off the right edge. It should also let the user restart the car at the beginning by pressing the R key. Be certain that pressing any other key has no effect on the program.

Version 2

Now let's add some ground into the scene. To make things interesting split the ground up into three regions: dirt, water, and asphalt. The regions can each be a third of the scene width or some other proportion you'd like. The must show up, in left to right order, as dirt, water, and then asphalt. Now, change the program so that the car moves 3 spaces per tick in the dirt, 1 space per tick in the water, and 5 spaces per tick on the asphalt.

Version 3

Let's add to the controls of the car animation. Make it so that pressing the left arrow moves the car to the start of the previous terrain area. For example, if the car is in the water, pressing left moves it to the dirt. If it's in the dirt, then left should wrap around and start the car in the asphalt. Pressing right does the same thing but moves the car to the next terrain area: dirt to water, water to asphalt, and asphalt to dirt. Don't stop there. Setup the controls so that pressing D warps the car to the dirt, W warps it to water, and A warps it to asphalt. Finally, when the car is in the dirt, draw some brown dirt like stuff behind it and when it's in the water, draw some water like stuff behind it.

Challenge

This last feature is optional. If you have the time and inclination, do it for an extra challenge and for fun. When the car passes off the right side of the scene and reappears on the left it warps. It's all or nothing. Try to make it so that as the front of the car goes off of the right side of the scene it begins to appears on the left side. This means that if the front half of the car is off of the right and side and all you can see on the right is the back half, then you should see the front half of the car coming out on the left. This should create the illusion that the scene is just an endless loop and make the animation that much slicker.

Grading

Your grade is a function of completeness and quality. The versions you complete determine the completeness of the program. Versions must be completed in the order listed above. Work should not be done on any version if you have not completed all the versions proceeding it. We are working on developing programs in an iterative fashion by incrementally building our way up to a final goal, namely version 3. All of your work is to be done in a single file. If one version forces you to change code you've already written, then change that code and update your tests and documentation accordingly. Do not comment out old definitions to leave a record of proof of your prior work. Notice that each version adds something new and doesn't break or remove previous program features. This is by design. It is important to learn to work on code in this fashion. So important that having bits and pieces of each version will be detrimental to your grade. Successfully completing a version of the program will determine the letter grade range for your project.

Version	Grade
0	D
1	C
2	B
3	A

Partial credit can be earned by completing some, but not all of the elements of the next version. For example, completing version 0 and one of the two features discussed in version 1 can raise your grade up as high as a C-. When applicable, you can also earn some partial credit for having signatures, purpose statements, headers, and tests for function related to the features of the next version. Remember that progress is not just measured by finished code but evidence of properly working the design recipe for whatever new feature you're working on. Again, progress should always develop along with the versioning given above. Don't skip steps.

Making progress through the versions is important but so is having well written, high quality code that is easy to read and follow. Your grade can slip if your work is buggy or sloppy. Every function you write should have full test coverage. Document each function with a signature and a purpose statement. Function and variable names should be descriptive and well chosen. Appropriate constants should be used. Your code should be properly indented and make use of white space and newlines to make it easier to read and to avoid line wrapping when printed. The code in the textbook should be your guide for what well written code should look like. Start new lines when they start newlines, use spaces where they use spaces, indent how they indent. Having poorly documented, untested, and sloppy code can result in your grade dipping down to a plus in the next lowest letter: a B could slip to a C+ and so forth. On the other hand, high quality documentation and beautiful code can bump you up from something like a B to B+ without even having to make progress on A level features.

Above all else, your code should run and do at least some part of what it's supposed to do. At its lowest level this could mean a bunch of functions with some tests that pass and some tests that fail such that the whole thing never really comes together. Failing tests means the program was able to execute it just didn't produce the desired outcome. Failing tests are bugs, not errors. Error are the things that prevent tests from even running. Failing tests are much, much better than tests that fail to run. You cannot begin to make a program better or make it do more things until it first does something. No matter what, turn in code that runs without error. Submissions that fail to run due to errors are unlikely to get a passing grade.

Important Dates

Date	What's Happening
Monday 10/1	Begin exercises 39-42 and 44
Tuesday 10/2	Begin exercises 39-42 and 44
Monday 10/8	Open Lab Time
Tuesday 10/9	Open Lab Time
Wednesday 10/10	Submit code via email and a printed hard copy by 5pm