CSC 591/481 Fall 20 Homework 5: Scripting New Games

Due: 11/18/20 at 11:45am

Overview

Your task for the fifth and final assignment is to implement a script management system, some scripted functionality, and a new game (or games) to demonstrate the resuability of your engine design. As with assignments 1–4, you will be using the SFML library to handle your graphics, and building ontop of the functionality you created in earlier assignment. Before the due date, please submit your code and a writeup of your results, including screenshots where appropriate. The assignment is an ***individual assignment***, you are to work alone. As always, you are expected to abide by the University's Academic Integrity Policy (`http://policies.ncsu.edu/policy/pol-11-35-01`), which includes providing appropriate attribution for all external sources of information consulted while working on this assignment.

There are 125 points available on this assignment. Students enrolled in 481 are required to complete Parts 1, 2 & 4 and the writeup, but may also choose to complete Part 3. Students enrolled in 481 earning more than 100 points by also completing Part 3 will receive a 100 on the assignment. Students enrolled in 591 are required to complete all parts of the assignment, and will receive a grade as the percentage of the 125 points they earn. All students are required to submit the writeup addressing the sections of the assignment they have completed.

Part 1: Scripting (30 pts)

For this part of your homework, you are to create a `ScriptManager` that will enable you to script the behavior of game objects as well as script event *handling* in your 2D platformer game. You are welcome to use the demonstration code posted on the course moodle page as a starting point. But, keep in mind the demonstration code is not a complete implementation that can be dropped in your engine without modification—it is a proof of concept only.

Using the `Duktape` library (`https://duktape.org/`), and `Dukglue` (`https://github.com/Aloshi/dukglue`) if you so choose, create a script manager that allows you to load and run arbitrary scripts. Using the script manager, at the minimum you must demonstrate:

- the capability to modify game objects during regular update cycles using scripts.
- the ability to *handle* events using scripts.
- the abililty to *raise* events using scripts.

One example of each will suffice for this part of the assignment. Duktape only supports `javascript`, so your scripts must be written in javascript in order for this to work.

The Duktape programmer's guide (`https://duktape.org/guide.html`) and source package have plenty of documentation and examples to illustrate the basic functionality needed to load and execute scripts. If you choose to use Dukglue as well, the `README.md` on the github page has excellent examples of usage that should compliment the minimal example provided on Moodle. That being said, the exact use cases you're being asked to implement are not covered precisely in any single example available from the Duktape or Dukglue developers—you will have some design decisions to make around how many `contexts` you want to use, whether you want to wrap those contexts in (static) objects or not, whether you pre-load scripts or load them on-demand, whether the scripts are stored in files or in-memory strings, *etc.* Like with all things in this class, each of these design decisions have some implications on performance and the rest of your design, so please be thoughtful.

Part 2: A Second Game (40pts)

At this point in the semester, you should have a very functional game engine. Congratulations! Now it's time to test its versatility. Your task for this assignment is to implement a second game using your engine code. You should make use of your engine's functionality to implement a second game that isn't a 2D platformer. One of the two following games are good options, but not a requirement:

- one level of a classic Bubble Shooter. If you are not familiar with the genre, play some of the free games on `http://bubbleshooter.net/` to get a feel for how they work.
- one level of the classic Space Invaders game. If you are not familiar with the game, simply search for "space invaders" on `http://www.youtube.com/` and play some of the videos that result.

You may make these games single or multiplayer. Regardless of the choice, it must be playable and it must use your client-server architecture, game object model, timelines, and event management system. You must have scripted functionality in your game as well. When implementing this second game, pay careful attention to how much reuse you get out of your core engine code. In particular, pay attention to how much functionality you can accomplish using scripting, rather than native code. To include in your writeup, do a diff between your code for the first game (the 2D platformer) and code for the second game. What percentage of the lines of code are different?

Bubble Shooter

If you choose to implement the Bubble Shooter, you should have a minimum of two colors of bubbles and there should be at least two layers of bubbles upon startup. There's no requirement to add additional layers during gameplay, but you should lower existing bubbles periodically as they do in a full bubble shooter implementation. To make the game logic easier, feel free to use two connected bubbles of the same color as your criteria for dropping bubbles, rather than the standard three. Your gun should rotate and fire according to your keyboard (or mouse) inputs. Also, make sure your game will end if bubbles touch the bottom of the screen.

Space Invaders

If you choose to implement the Space Invaders game, you should have a minimum of two rows of space ships upon startup. The ships should move across and down the screen as they do in the real game. Your gun turret should move and fire according to your keyboard or mouse inputs. You are not required to have defense structures typical of the first three levels on the full space invaders game.

Part 3 (591 required, 481 optional): A Third Game (25 Pts)

For this final part of the assignment you are to implement a third game. You may choose to implement the other recommended game from Part 2, or another game of your choosing. One possibility is the classic game Snake (`http://patorjk.com/games/snake/`). If you choose to implement a different game, it is probably wise to consult with the instructor prior to beginning work on it. The criteria for this part of the assignment are the same as Part 2, including the code diff for the writeup.

Part 4 (required for all): Reflection (30 pts)

In 2–3 pages in a separate document not part of your assignment writeup, please reflect on the overall design of your engine as it has evolved over the course of the semester. That is a minimum of 2 *FULL* single-spaced pages. In particular, please explain how successful (or unsuccessful) you were at designing for reuse. What systems did you have to change to get your second game to work? How did you change them? If you were going to start over again to design your game engine again, what would you do differently? What would you do the same? Why?

Writeup

Now that you have designed and implemented a script manager and a second game, write a 1-–2 page paper summarizing your design. That is a minimum of 1 *FULL* single-spaced page. It is **strongly** suggested that you do not limit yourself to only answering the questions posed in this assignment. Think creatively about what you have done. Why did you make the design decisions you made? What did they enable for this assignment? What will they enable in the future? The most successful writeups will contain evidence that you have thought deeply about these decisions and implementations and what they can produce and have gone beyond what is written in the assignment.

As an appendix to your paper, please include all relevant screenshots to support your discussion. The appendix does not count toward your 1—2 page requirement. Your points for the writeup will represent $\frac{1}{2}$ of the points allocation for each of Parts 1, 2, & 3, but not Part 4 (*i.e.*, 20 of the 40 points for the second game section will come from your writeup).

What to Submit

By 11:45am on 11/18/20, please upload to moodle a .zip file containing your code for all games (including scripts), .pdfs of your writeup and reflection document, and a README file with instructions on how to compile and run the various versions of your code.