CSC 481/591 Fall 20 Homework 3

Due: 10/07/20 by the start of class

Overview

Your task for this assignment is to add more functionality to your engine. You will submit your code and a writeup of your results, including screenshots where appropriate. This 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 the first 100 points (Sections 1–3), but may also choose to complete Section 4. Students enrolled in 481 earning scores above 100 will receive a 100 on the assignment (*i.e.*, extra points will not be given as extra credit). Students enrolled in 591 are required to complete all 125 points, and will receive a grade as the percentage of the 125 points they earn. All students are required to submit a writeup addressing the sections of the assignment they have completed—simply submitting code without a writeup discussing that code or including a section in the writeup to cover a part of the assignment for which there is no code submitted will result in zero points for that section of the assignment.

Part 1: Game Object Model (40pts)

For the first component of this assignment, you must design and implement a *runtime game object model* of your choosing. The design is completely up to you. You may opt to use one of the models we discussed in class, or come up with something on your own. The only requirements are that it be extensible enough that you can reasonably implement new types of objects that you aren't necessarily using at this point in time.

To demonstrate your model, you must implement two types of objects:

1. **Static platforms** (*i.e.*, rectangles) that have a position in the environment and a color. Demonstrate at least three different colors by creating at least three different platform objects.

2. **Moving platforms** that are implemented similarly to the static platforms, but also move around the screen in a patterned way. Demonstrate at least two different patterns of movement. **Note:** movement in response to key presses does not count for one type of movement.

This part of the assignment is largely about your design and your ability to realize it in code. The two types of objects required are not likely to stress the capabilities of your model, so make sure to give appropriate care to documenting your design in your writeup and in code comments so we can reasonably evaluate your design choices and implementations.

Part 2: Multithreaded, Networked Scene (20pts)

For this portion of the assignment, your task is to fully integrate your SFML cadebase with your multithreaded, networked client and server from Assignment 2. If you are enrolled in 591, this should be integrated with your code from Part 5 of Assignment 2. If you are enrolled in 481 you may choose to integrate with either Part 4 or Part 5 from Assignment 2. At the minimum, you must support the following functionality:

1. You must have a multithreaded server that reads data from and sends data to clients, all while accepting *an arbitrary number of incoming connections at arbitrary times*. Whether you send GameObjects, Strings, primitive types, or something else isn't critical provided the correct behavior results. The server must also gracefully handle a client disconnecting without affecting the operation of the remaining clients.

2. You must handle keyboard inputs that control the movement of a unique object on each client.

3. You must draw the entire scene on a minimum of four clients, including the movement of objects controlled by the inputs on the clients.

4. *You must utilize your runtime object model on both the client and the sever.*

In other words, you should start a server which will define the "scene," *i.e.*, maintain the game world object model including static and moving object locations. Then, you should start any number of clients (we will test with two, three, and four), where each client will create a player object (square, rectangle, oval, *etc.*) and handle keyboard inputs to move that object around the scene. Each of those movements should get transmitted to the server, where it will be re-sent to every other client and drawn on their scenes as well. In effect, you're laying the foundation for a multiplayer platformer game which you will continue to develop throughout the course of the semester.

The main differences between this part of the assignment and Part 3 of Assignment 2 are: 1) You have a requirement to handle an arbitrary number of connections that initiate at arbitrary times; 2) You have a requirement to handle graceful disconnects when clients stop; and 3) You have a requirement to implement all of this using the runtime game object model from Part 1 of this assignment. As a reminder, each client should run in a separate *process*, not a thread within a single process.

Part 3: A 2D Platformer (40pts)

Your task for this part of the assignment is to use your multithreaded server and client code, as well as your game object model to implement a multiplayer 2D platformer. At the minimum, you need to have the following game objects:

- **Characters** that are the depiction of the client-controlled players. Characters must respond to keyboard inputs to move left and right, as well as jump. Note: This will require some implementation of gravity.

- **Static platforms** that characters can land on.

- **Moving platforms** that function similarly to the static platforms, but move around the screen in a patterned way.

- **Spawn points** that are hidden (*i.e.*, not drawn or rendered to the screen) objects marking the location where characters start from. There may be multiple spawn points in your environment.

- A "**death zone**," which is a different type of hidden object that marks some boundary of your environment. When a character collides with the death zone, they should be teleported a spawn point of your choosing. There may be multiple death zones. You must use collision detection in order to implement the death zone.

- A "**side boundary**," which is another type of hidden object similar to a death zone, however when a character collides with a side boundary it should trigger a lateral translation of all objects in the

scene (some of which may transition from viewable on the screen to not, and *vice versa*; more on this below.)

When implementing these game objects, think in terms of the game object model you developed. Are you using inheritance to accomplish different behaviors? Components? Properties? What behaviors, properties, or components do you need to get these game objects working right?

In addition to these requirements, you must also implement "side scrolling", where the level of your 2D platformer expands beyond the viewable portion of the screen. Practically, this means you will need to have game objects with positions that may not be viewable at any given time.

In order to make sure your platformer works correctly, you will need to rely upon the collision detection functionality from your first assignment. Although not required, it may be very informative to implement additional functionality like scoring, a goal location, and/or a timer.

Part 4 (required for 591, optional for 481): Performance Comparison (25pts)

For this section of the homework you will implement a second "networking protocol" and do a performance comparison. Your aim for this part of the assignment is to characterize the performance of different approaches to client/server communications. You are to continue to use 0MQ as the library to facilitate communications; however, you must devise at least two different strategies for sending information between the clients and the server. Practically, this means either keeping the representation (*e.g.*, packed strings) the same and varying the type of information sent (position updates only, or entire object representations), varying the representation and keeping the information constant, or changing the ZMQ socket types substantively (*e.g.*, to make it peer to peer). The goal is to have the differences between your two approaches be different enough that there is a noticeable performance difference, but not so different as to fundamentally alter the "game play experience."

To compare your two methods, you must use a server with two, and in separate set of experiments three clients (although you may find it informative to do more). You will want to measure the performance in terms of runtime. Given that you're using a fixed number of clients (two and three), you should change the number of static and moving objects in the environment. The moving objects do not have to be controlled by keyboard inputs. You will want to measure how long it takes to complete some number of game loop iterations (say 10,000) under different conditions. How long does it take with 10 moving objects on each client? What about 100? Can you get to 500? How does this change if you're using two vs. three clients?

Note: It is not required that the code for this part of the assignment be integrated with your platformer game. You may choose to branch your codebase to develop a stand-alone application to collect your data and analyze performance, but it is required that you are using your runtime object model.

Writeup

Now that you have designed and implemented a number of engine components, write a 2-–3 page paper summarizing your design. That is a minimum of 2 *FULL* single-spaced pages. 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 2—3 page requirement. For students completing Part 4 of the assignment, any graphs used to present data should be part of the appendix. Your points for the writeup will

represent $\frac{1}{2}$ of the points allocation for each of the above sections (*i.e.*, 20 of the 40 points for the "Game Object Model" section will come from your writeup).

What to Submit

By the start of class on 10/07/20, please upload to moodle a .zip file containing your code, a README file with compilation instructions and execution instructions, and a .pdf of your writeup.