Section 4: Throughout this semester, my engine has gone through several phases of massive change. Between almost every assignment, I've had a major change to the project to meet the requirements. I started off the project with a skeleton that had very inflexible object designs. These objects only lived served through two of my assignments before I practically completely scrapped them.

Between assignments 1 and 2, I shifted from a single threaded design to a multi-threaded design. This required some close scrutiny as to where I needed mutex locks to ensure everything functioned as it should. I also had to invest a lot of time developing the time lines and making them usable by the engine to speed, slow down, or pause. I believe the timelines were implemented pretty well since throughout the project, I never had to overhaul any of the timelines from assignment 2 onwards. It took me a good amount of time to properly integrate server-client into my engine. Overall, I believe that this was done relatively well – only requiring some modifications in assignment 3 when I made my objects partially serializable to simplify information being sent over the network.

The shift from assignments 2 and 3 was the biggest shift I had to do on this project. In order to switch to a hybrid generic-strict component model, I had to throw away almost all of my code for objects (you can still see my old code in the OldFiles filter in visual studio). Here, I regret not changing my design more to allow more flexibility. I should've done something like creating an object that stores pointers from the engine that the game object would use and then fed that to the game object constructor. This way I wouldn't have had to spend years modifying my game objects whenever the time arises. The components I made were fairly decent since they improved the behavior of objects (and simplified calculations) – although I could've provided functions to my game objects to add components – that way I'd have more flexibility with object creations. As previously mentioned, I partially serialized my objects to allow for easier communication between the server and clients. I think this worked pretty well since it kept the

information being sent minimal. The thing I really could've improved here was giving a better unique identifier to objects that would provide better identification of objects on both the server and client.

Moving from assignment 3 to 4 took me a little less time than it did from 2 to 3. Refactoring my code to utilize events heavily took a good amount of mental power since it threw some of my calculations out of whack. During this time, I shifted all my object movement to be done by one component (other components may calculate movement and send it to this one indirectly through the game object). This simplified the code behind recording events – making it so that I only had to record "animation" events. My event implementation was mostly successful but proved a bit cumbersome whenever further changes are necessary since I defined the variant and event types in my event header (this makes it so that recompiling takes a while, especially since I have a lot of objects that read from it). Interestingly enough, after finishing my implantation of events, my collision calculations improved. I was unable to fully implement the replay speed adjustments – I'm not sure if it is a result of my event implementation or timeline implementation that was giving me trouble.

I'm not very proud to say this, but I had to change a good amount of my game object constructors to handle the shift to a second game. This was primarily because I didn't have a particularly good method of creating and destroying game objects (I had something like that for events, but overlooked game objects). One of the other things I regret is me making my constructors so convoluted – I should've created another object to store important engine pointers – or gone with another method to provide my objects with the pointers they need. If I restarted my project right now, I'd make my game objects fully serializable and simplified constructors. I think I could pretty much rip the code for my timelines and server-client stuff and put it into another project (with some minor adjustments).