CSC 481 Reflection

Sarah Willis

11/16/20

Over the course of the semester, I completed five homework assignments that involved continuously building upon a rudimentary game engine. The first homework involved getting accustomed to using SFML and developing some initial basic elements, such as creating windows and drawing to them, accounting for user input to move objects, and detecting collisions. With this first homework, I initially developed my game objects as individual classes, using inheritance to have the moving platform be a subclass of platform as well as to have collision be an interface implemented by all applicable objects (character and platforms). This design was natural to me due to my computer science education focus on Java where everything is an object. At this point, I imagined continuing to build upon my engine system with additional classes and inheritance hierarchies.

For the second homework, I was required to add in multithreading, timelines, and a client-server setup. I incorporated multithreading by separating out sections of my code that made sense to run in parallel, such as character movement, and created a timeline class with two subclasses as exemplified by the lecture pseudocode. For my client-server setup, I focused on a design combining the request-reply and pub-sub patterns to allow for certain instances such as client connections to involve two-way communication while other instances such as general updates from the server involved one-way communication. For the communication between client and server, I made use of messages encoded using ints and floats, parsing them out on either side in a defined manner. My client-server setup was admittedly not designed very well for reuse as it was very specific to the particular implementation I was completing and was not well modularized to allow for the utilization of particular sections and not others. At this point I was still utilizing my implementation of the objects and collision from the first homework.

The third homework involved the creation of a game object model and then integrating that model with my client-server setup as well as other existing code to create a 2D platformer game. After careful consideration following our lessons on different forms of game object models, I made the decision to turn away from my existing inheritance-focused implementation in order to go for a property-centric game object model, as I thought the idea of organizing objects into their properties would better modularize development and allow for greater reuse. Thus, I ended up getting rid of most of my code from the first homework, reimplementing my objects and collision by defining a number of properties that would in turn define my game objects. From the second homework, I reused my timelines, but I was unsuccessful at reintegrating my multithreading and had to significantly re-implement my client-server setup due to aforementioned issues with not designing well enough for reuse. Unfortunately, due to these issues as well as failing to give myself enough time on the assignment, I was not able to get my 2D platformer game fully functioning.

For the fourth homework, I was required to integrate an event management system as well as a replay system into my existing code. Due to these new requirements as well as the bugginess of my client-server setup, I made the decision to significantly change my setup, converting to a server-centric model with the server handling events and general game logic while the client handled only rendering objects with positional updates sent from the server and sending user input to the server. In doing this, I also worked to create an event management system and reimplemented my game logic utilizing this system. My event management system was generally made up of classes with the recommended type of representation, such as an EventHandler class with specific subclasses. I integrated my game object model into my event management system by having event handlers contain references to the map of all properties, allowing them to retrieve properties by their id and perform necessary functions upon them.

For my replay system, I focused on having the server record positional updates and then resend them to the client to render, removing the need to worry about saving and changing state over the course of a replay. My event management system reused my timeline implementation from the second homework and my game object model from the third homework was also reused, but many of the remaining elements from the past homeworks went unused or were reworked.

The final homework involved the integration of a script management system as well as the creation of a second game separate from the 2D platformer game developed over the course of the semester. For the integration of the script manager into the platformer game, I was able to reuse my existing implementation from the fourth homework, adding in a ScriptManager class as well as a script-specific event handler on top of it. For my second game, Snake, I was able to reuse my game object model, only needing to create a single new property to go along with several that I already had. I was also able to reuse my event management system, though the only event handler I was able to reuse was the one for positional updates, as the rest of the handlers had to be new to correspond to specific game logic for the new game. My timelines were thus also reused with my event management system as well as my new movement property, and I reused my scripting system from the first part of the homework to incorporate scripted functionality into the game, though I did not reuse the script-specific handler. I was able to largely reuse my client-server setup as well, though once again my failure to implement it in a more modular manner for proper reuse led me to a less adaptable client-server setup than I might have liked. I found myself having to try and fit messages aside from plain positional updates into the communication between the client and server regarding the positional updates in order to avoid having to further complicate my setup and create more messy code.

Overall, I would say that I was successful at designing for reuse in some areas but unsuccessful in others. My work in the first two homeworks ended up largely reworked because I had yet to step out of my comfort zone and consider more effective approaches that I would explore later in the course. However, my work in the later homeworks was more successfully reused. In particular, my game object model was successfully designed for reuse as I was able to reuse my property system to create new properties as needed and was also able to reuse some of the properties themselves, including in a different game. I think I did a good job with that system in abstracting out the different functions that a game object might have and implementing them to be reusable. I was also successful at reuse in designing my timelines, as I did not need to make any significant changes to them after first implementing them in the second homework. My event management system was also designed fairly well for reuse, though perhaps less so than the game object model due to the need to create all-new handlers, including a different handler for user input. This is likely where scripting could come into play to allow for the basic structure of handlers to be designed separately from their specific logic.

As for scripting itself, I think my scripting system worked reasonably well for reuse in my second game as I was able to reuse the ScriptManager without issue, though the system was certainly not as robust as it could have been. I found myself avoiding writing significant portions of code in the form of scripts due to the overhead required for registering methods and functions. I could have done a better job of integrating my scripting system more deeply into my engine in order to simplify its use and allow it to be readily reused in various situations. I also failed throughout the semester to design a client-server setup that could be readily and effectively reused. I became too focused on simply making something that worked in the immediate moment rather than taking the additional time to create a system that was more complex and could be better reused and adapted.

With that said, if I were going to start over and design my game engine again, I would be more thoughtful about how to implement a client-server architecture that could be readily reused. I would want to modularize the implementation of it, separating out different elements such as sending particular messages, creating sockets, etc. to allow for these elements to be reused without needing to rewrite code over and over. This would also involve abstracting away many of the details to help clean up the main methods and make the overall implementation less jumbled. I would also endeavor to create a far more robust scripting system, integrating it more deeply into my engine so as to make the incorporation of scripts into the game far easier. In particular this would involve integrating the scripting system with the event management system to allow events to be more easily raised and handled through scripts without needing to specifically register so many different methods and functions. I would also want to improve my implementation of my event handlers, utilizing the scripting system as mentioned previously to allow for greater reuse with them through separating specific game logic from the general logic of the handlers themselves.

In terms of what I would keep the same, I would go with a property-centric object model again as I found it to be a very adaptable representation of game objects. I also think separating out my game objects into their various properties made sense to me in spite of my natural inclinations toward particular classes and I liked the flexibility of this design, allowing me to have the client and server each contain only the properties appropriate for their function while still technically operating on the same set of game objects. I would also implement my event management system similarly because I think it was designed well for reuse with a general event class and event types represented as strings, a singleton event manager class, and a general event handler class with specific subclasses. As mentioned previously, however, I would want to improve the handlers to allow for greater reuse with the logic of different games, and I would also want to design the system better with performance in mind by minimizing the amount of redundant event raising and handling. I would also keep the same timeline implementation, as it proved general and adaptable enough for reuse.

In conclusion, I have learned a lot over the course of this semester about game engines themselves, the implementation and design of them, and in particular how important it is to keep reuse in mind while coding. The lessons I learned here will aid me in the future as I move forward to other classes and projects. In particular, I will aim to think more outside of the “inheritance box”, considering alternative designs such as the property-centric approach or component model. I will also aim to take the time to implement a system right the first time rather than costing myself time and effort by coding in a more rushed and messy manner as I tended to do with regard to my client-server setup. I will also harbor a greater appreciation and knowledge for the game engines that I utilize in the future.