CSC481 Homework 3

Casey Byrnes

10-30-2023

Design Decisions

For Homework 3, I first chose to use an object-centric game object model. I chose this model because most of my code was already close to this model already, but also because it makes it easy for me, as well as future users, to know exactly what functions each object has. My objects already contained a “is-a” relationship with the sf::RectangleShape, so it made sense to me to create a new object that “is-a” sf::RectangleShape and then make all my other objects use that same “is-a” relationship with my new object. Although there are benefits to a more property-centric model, for this game engine, those benefits did not outweigh the benefits of an object-centric model, namely ease of use/implementation. Since the scale of my game engine is currently small, having this model seemed sufficient, but if I used this engine more in the future and had enough growth to warrant it, I would consider using a mix of object-centric as well as property-centric.

To do this, I implemented a GameItem class that the interactable in-game objects can inherit from. To do this, I chose to have the GameItem inherit from the sf:: RectangleShape class since all of the objects that will inherit form this class are rectangles. Since these objects will be at places in the level that may need to shift left and right depending on a player running in those directions, I added functions for shiftLeft and shiftRight as well as a constructor. I then refactored my Platform and FloatingPlatform classes to inherit from the new GameItem class rather than the sf::RectangleShape class.

Next, I chose to implement a SpawnPoint class. I originally intended this to just be a simple point that would not be a rectangle, but it was required to inherit from my GameItem class. So, I just made the x and y sizes both zero and this allowed the SpawnPoint to inherit from my newly created GameItem class. This proved to be useful though because it can now inherit from not just the methods I have currently in the GameItem class, but any future ones I may choose to implement.

After that I implemented a new DeathZone class that also inherits from the GameItem class. Although it will not be displayed, it does have a size and position and needs to shift left and right when the player moves past the screen so inheriting from the GameItem class was very useful here.

I also implemented a SideBoundary class that can be used to detect when a player is running past the screen and the window needs to “shift” left or right. These objects do not shift however and they stay ion the same place on the screen no matter what is actually displayed in the window. To aide in knowing when to shift the screen, I created two additional int variables: origin and end. These are used to determine if the player is at the beginning of the level and the screen should not sift to the left anymore, or at the end and the screen should not shift to the right anymore.

To tie all of these objects together I made another new class, GameObject. I used this class to hold vectors for pointers to each of the objects that are created in the game. This makes it much easier to add more of each of the in-game objects and it’s also easier to loop through them as needed, such as collision checks. So now, rather than creating a new FloatingPlatform object, for example, the user will

create a single GameObject object and then call upon that object to make a new FloatingPlatform, which is added to the vector and the user can easily access it later when needed.

To use all of these in my game engine, I am creating a single GameObject object and then I am creating each of my objects from it. By creating the objects in the order that they appear in the game I am also able to easily measure the exact positions to places the next object by using the previous objects placement location and adding its width to that for the placement location of the next object. This also makes it very simple to calculate the end of the level so that the player can not go past that point in the level and reach a location that has no objects in it.

Unfortunately, I am still unable to get the networking issue sorted out from homework 2. My server and client are unable to connect to each other. I spent hours on this, including staying up until four in the morning and then running off of two hours of sleep. At the time of this writing, I am already using 1 of my late days and I still haven't made any real progress, so I am submitting without it. I cannot afford to lose more points using more late days, especially if I cannot get it working after more days!

If I could have gotten it working though I would not have needed to implement much to get it up to date for homework 3. I already was trying to use multithreading from homework 2, one thread to run the server and clients game loops, and one thread to handle communicating with each other. I also was attempting to handle graceful disconnects with the client sending a disconnect request, so I would have only needed to include a timeout if a message had not been received on the server end after a short period of time.

I was using an int to create ID's for each connected client so the server could handle up to the int max number of connections and I was already only handling inputs on each client that had focus. If they would connect properly, then they would be displaying the entire scene as well, but I would have needed to refactor a little to handle a wider level. For example, if one client was at the start of the level, that player would see the start, but if another player was at the end of the level then the end would be displayed to that player. The whole scene/level would be possible to be displayed though if the two players were near each other. The way I was trying to send the game objects (using a struct) would have accounted for this and each client would have just shifted the game window to view the items that were in the view of that client.

Each clients character would have also been easy to view on all other clients since each characters position would have been sent with each struct sent. With all of the platforms, death zones, spawn locations and characters in a struct, each client would only need to set the position for each of them (except for the client's own character) and then if the player walked into a side boundary, I would use an int to track how far they traveled from the start then shift all in-game objects by that amount. This would have required a small amount of refactoring, but it would have been fairly simple because of the way I designed it all on homework 2.

I am very disappointed that I was unable to get part 4 from homework 2 working because it is now affecting homework 3. I am worried that it will also affect homework 4 and 5 even, but nothing I do seems to work. If I wasn't taking any other classes then I could possibly spend even more time on this, but as it is, I am spending every waking moment working on homework for this class or another and I believe that it is better for me to just submit it without the best functionality and at least have something submitted.