CSC481 Homework 4

Casey Byrnes

11-15-2023

# Design Decisions

For Homework 4, I first created a new event class (called GameEvent) that contains a string type as well as a map of parameters, a priority (based off of the main/anchor Timeline) and a function to add parameters to the event map. I added a float into the constructor so that additional time could be added to the priority of an event. This allows for an event to be made with high priority but adding 0 to the priority, or less priority by adding a float to it so the event will be handled in a future iteration of the game loop. I then created a struct for the parameters using an enum and a union, so that each type of parameter can be access when added to the event.

Next, I created an EventManager class that is used to register/deregister events so that the events will only broadcast to the game objects that matter to each specific event. I also implemented a raise function which will add the events into a priority queue, using the priority of the events as the priority in the queue. This allows for events to be handled at a later game loop iteration, preventing infinite loops when one event creates another type of event. The last thing that the EventManager class does is handle events. I chose this method of handling the events, rather than a specific event class, because it made more sense to me to have the EventManager manage all of the events and simplify the user experience of handling them. I tried many different ways to incorporate a way to do this that would be automated, but everything I tried failed. I ended up implementing it through string checks to see if the event type matched the types that I created and if it did, then I would have the appropriate object hand the event. In the future I would like to simplify this though, especially with the idea of scripting in mind. It would make more sense to not have to access the code to create a new type of event so that it could all be done in a script.

After the event manager was implemented, I added a virtual void function to my GameItem class. Since all in game objects inherit from the GameItem class, all of the GameItem objects that are made will have access to the onEvent class, provided an implementation has been made for it. I chose to implement it in all of my game objects because the events that I am using incorporate different parts of all of the GameItem objects. This took a large amount of time and required a bit of refactoring to my existing code, but now all of the objects can handle the appropriate events when they need to.

Each object handles the events differently, but for the most part they simply perform the needed changes on the objects that are related. For example, the collision events need to make sure the player character doesn't intersect with the object it is colliding with, so it will move the character to the appropriate location so that it is just touching it. The one event where this was different was the death event. When the player hits a death zone, the PlayerDeath event is raised. When handling the PlayerDeath event, it creates a new event, PlayerSpawn. The PlayerSpawn event is what will actually move the player back to the spawn location. Having the death event make the spawn event would allow other developers the ability to customize their game in a different way if they chose. For example, if you wanted to implement lives into the game, you may choose to have the death event reset the game, rather than spawn the player back at a spawn point.

After all of these changes and additions, I started receiving different segmentation faults, and I was able to determine that it came from having a separate thread handle my inputs. To fix this, I once again refactored my code so that the user input checks were handled in the main game loop, and when input was received, an event was created to actually move the player character.

At this point, I had all of my events implemented, so I added a while loop near the end of the game loop that checks if there are any events that need to be handled during this iteration, and if so, the EventManager will handle them and then pop the event from the priority queue to remove it.

Once again, I struggled on this homework and especially the networking component side. I am still unable to get my client to connect with my server, even though I spent over 100 hours on this homework. During homework 2 I was able to successfully connect my client and server so I don't know why I am unable to still connect. Because of my struggles with the network since homework 2, I have not made any changes to the client or server and therefore I have no events in my client/server. For this reason, I have nothing to submit with homework 4.

I am again disappointed that I was unable to successfully complete this part, especially since it is worth so many points, but with the design that I currently have I think it would work without many changes if it was connecting. The server only handles the game's environment objects, so in this case, just the platforms. The way I have everything implemented would allow the server to send the latest updates for the moving platforms via events and then the clients would be able to display the platform. Since side scrolling is implemented, the x-axis location would need to be updated, but I have a variable that tracks this already so I would only need to add that variable to each platform's x location value and it would be adjusted correctly.

The other change that would need to take place would be the client sending it's updates to the server. I would choose to just send the location of the character itself, rather than the movement event itself, or I would create a new event that just updated the character location, rather than moving on input. I would add two more events for connecting and disconnecting, both to the server and the client models. This would allow for the server to receive a new connection event and create a character, then track that character through updates from the client. A disconnect event would be used to delete that character and handle a smooth disconnection.

I would also add another event for timeouts for a client. The server could create this event on every update from each client and add it to the priority queue, but with a priority lower so that it would be run in the future. Then if the client disconnected abruptly, the server would be able to remove that client and the character from it. If, however, that client stayed connected, on the next update it would remove that event from the queue, then add another one with a later priority. This would basically reset the "timer" for that client and it would not disconnect unless it actually timed out or send a disconnect event.

Based on how I would implement this, my approach would be neither Server-centric nor Distributed. I think that this would be better though because it allows the clients to each handle their own inputs and character movement, which would result in a more smooth experience for the players.

There would be no server lag with inputs so players would not notice any lag.  This approach could have some lag though in other ways. Other players showing up on the screen could potentially be less smooth because they are all being handled on their own client, sent to the server, then send to all of the other clients. I still think this would be fine though because of the game that I have implemented. The other players don’t collide and they can all just play for fun together. If it were a fighting game, however, then it could make a difference because characters could potentially be in slightly different locations on different clients’ screens. For most other games though that have a more “co-op” style gameplay, I think this approach would be best.