John Boezeman (jlboezem)
12/3/2015

Assignment 4 - Reflection

My engine design used a Property-centric Game Object Model with functionality tied to the Properties as well as data so the vast majority of my engine was easily swappable or reusable as very little of it was fixed. I decided on this model mostly as a challenge as I had not implemented this type of engine before but I also selected it because many object types in the types of games this engine was created for are quite similar with much shared functionality. For example, almost every object in the first and second game had a shape and position and needed to do some type of collision.

Because of this, it was simple to create Property classes for Shape and Position which handle these things for every owned GUID, permitting very simple object type creation as all that is required is supplying the basic information and a GUID to these singleton classes which then handle any number of objects registered to them. Even though I do think this design is interesting and allows very flexible and simple object management, it is more complicated and less well known in general, so I would not create an engine with this model for general use as the goal of an engine is to make reuse easy and having a less known/more complicated Game Object Model does not promote reuse. Of course, the internal Game Object Model does not have to be the same as the one shown to the user and can be completely different but it seems that the benefits of the Property-centric model are not worth the extra overhead required to implement it behind a more familiar model for general use.

These concerns aside, I was reasonably happy with my engine in general as it was trivial to implement a second, fairly different game. Additionally, it would have been very easy to implement either of the suggested games as both only really required a different level and a few small tweaks to functionality. For example, the Space Invaders game would require me to add the ability to shift down a row after reaching the end of a path but that would just require a little bit of code added to Pathed, perhaps adding an End Of Path action (or script) section. Next, I would need to add a keybind for shooting and add said bind to the PlayerProcessor for handling. After that, I would need to tweak Spawner to make the bullets work correctly. Finally, I would need to change the level creation code to make it create the Space Invaders Level. These three changes are all that would be required to implement Space Invaders from my engine starting from where it was at the end of the Platformer.

For my second game I had to do similar changes, including: keybinds, level creation, physics processing, collision processing, spawning, and damage/death. The keybinds and level creation changes are obvious and trivial, the physics and collision changes were to remove gravity and ignore shot & shot collision completely as this

provides a fair performance boost to not have to handle all the collision events such collision creates, spawning/damage/death changes were to make shots work correctly and disappear when appropriate while not destroying each other. For level creation, I made 8 Pathed Positional Shaped Drawable Spawners that have fixed paths, a position, a shape, are drawn, and spawn the shots the player must avoid. In addition to these, I made Damaging Shaped Positional boundaries that prevent the player or shots from leaving the viewable area. Overall, I made 72 changes (not lines but grouped changes), with my first project being 2,821 lines and my second 3,001 lines of code. The reason it is so simple to add the game is that the functionality and data for Properties have no ties to actual game objects and instead provide these for any object they have the GUID for, allowing extremely easy data and functionality specification for even novel objects.

I was also very happen in general with my Event System overall though I did have some issues with it, mostly due to issues with my network and event interaction. My general event architecture was an Event Manager that handled the events, Processors which handled the events they Subscribed to, and any class could raise events. The Event Manager presented an opportunity to use the relatively new Lambda utilities in Java to make simple and fast event handling and management with statements such as

```
processors.entrySet().stream().filter(entry ->
event.getType().matches(entry.getKey())).forEach(entry -> entry.getValue().forEach(p
-> handled.put(p, handled.get(p) == null ? p.handle(event) : true)));
```

and

```
events.removeIf(event -> filter.test(event));
```

to filter the Processors based on their provided Regular Expression indicating what they are interested in and invoking their handle method on the event if they are interested in it or remove any events like a given predicate, respectively. The Events were represented as Objects with a String for type and a Map for info to permit such easy filtering. Because of this design, it is trivial to implement new event types as you merely need to set the type String and info Map to the appropriate values for the new event. When implementing my new game, I did not need to add any new event types as all of the data I needed had already been implemented with collisions and keypress events.

If I were to start over, I would most likely create a more standard Game Object Model, with actual Objects rather than implicitly defining them with GUIDs as such models are more common and would make it easier for people to pick up my engine. Also, due to the commonality of said engines, it would be far easier for people to troubleshoot issues they may be having because there is a vast amount of information on such models. I would, however, use the same style of Event Management System

as it seemed to work well and cleanly in general though I would change how it interacts with the networking code. Speaking of networking code, I would definitely change my networking code as I was unhappy with it throughout the project. If I were to re-implement the Property-centric model, I would do it much the same though with more thought as to what properties I choose to implement. Additionally, I would make it much easier to get a full list of all the properties and their data. Finally, I would try to add more functionality to the Property class itself as it only has very basic functionality right now but seems like it could potentially hold a lot of the core methods for most properties though I was unable to implement it with the current iteration.