Joseph Hart

DIG 4778

Final Project

12/4/2022

The Article

I am making an Inventory program that will read a JSON file and produce a list of items and their amount on the screen. First in the console, and then in Unity.

I create a new solution in Visual Studio called *Final*. I download Newtonsoft NuGet package. I create two projects, the first is a console application called Final.Client, which is where the console API will be. Then I make a class library called Final.Commons, which is the code base and the portion that will be ported to Unity later.  
  
I make a folder called Content to store a JSON script and place in Final.Client. The JSON script will contain the “items” for populating the inventory that my program produces.

I begin in Common, creating my classes: Item, Inventory, Organizer, and App. Item has two fields for name and description, assigned by the JSON script later. Inventory has three fields: an item list called Storage which will be assigned by JSON later; then two dictionaries (ignored by JSON), one called ItemsByCount, and one called ItemsByName. The Inventory has a function for calibrating these dictionaries.

Now I wrote the JSON script. It’s simple, a list of Items and their descriptions. Duplicate instances of an item do not need a description; only the first instance of an item gets its description read. It is not as elegant as having a list of descriptions for the items to consult, but it works.

I make the Organizer class: the original objective was to take a list of Items, store any distinct items, tally their duplicates, and then produce the data as a new dictionary that can be assigned to the Inventory’s ItemsByCount. I solved this. But was perplexed how to *sort* the items into alphabetical order. I also wanted to provide the user with a command to sort the items by their tallies. Both additions were left out because I am busy and is beyond the scope of my proposal.

I finish Final.Commons with the App class. Here the Inventory is declared and the Organizer gets called. Both IO interfaces are declared and the Input function is subscribed to the Input event. The Input function parses a string in order to select an Item in the inventory and prints out the Item description. The dictionary was set to case in-sensitive comparison, so the user needn’t worry about casing.

I created two IO classes inside Client that are contracted to the IO interfaces in Common. This facilitates the Console class for the project. Next I created the Program class in Client. The App object is instantiated by deserializing the JSON script in the Content folder. I call App.Run() passing in the IO classes for arguments. The App continues to run while the boolean *running* returns true.

At this point I wanted to move the project into the GitHub. I admit I had some compiler troubles. I had to retrace my steps with Zork and remember to set the executable to the Client project. Then I updated the GitIgnore file to include all bin and obj files from the Final project.

Next up was putting the game in Unity. I create a Unity project called Final.Unity. I import the Final.Common.dll and make a canvas for displaying the contents visually. A 3x3 grid with icons and a single text box to convey the description of a selected item. I import the TMP essentials for this. Then I made three scripts: GameManager, InputUnity, and OutputUnity. The latter two are for Unity specific IO from Final.Common. The first is the Unity client API.

It’s simple, I needed to raise an event in Final.App once the Organized class finished producing the list. Then I Assign the data to an array of UI elements in Unity. I created four sprites: Sword, Phone, Bike, and Diffuse. The first three are specific for the example items, but the Diffuse is a fallback sprite for any and all items that lack a dedicated image. I would love to add a method in which an Item Icon can be added by the user/designer, but it is also beyond the scope of my proposal, and I am busy with other finals.

As it turns out, the Input script was kind of unnecessary because Unity has an event system for hovering over UI elements already. I still have the InputScript to satisfy the App.Run() arguments. The last thing I do is update the counters, first by turning them on if there is a value to be given.  
  
And that is that. A program in console and Unity that both use the Final.Common code base in order to display information that was provided in a JSON script!