1. Project Summary

Project Name
Unity music visualizer
Names
Ben Weiler and Sean Fitze

Final Video:
https://drive.google.com/file/d/1tIR-h94tjAc8y68zNHHtvYRte2XHuG_8/view?usp=sharing

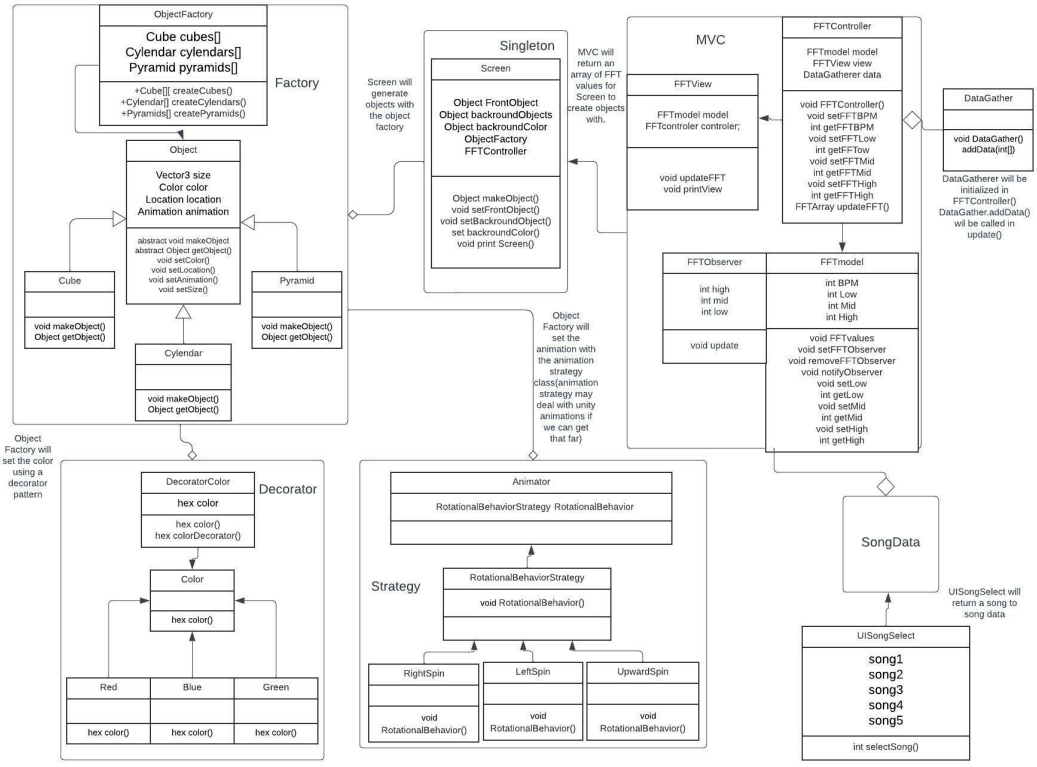
2. Final Product

Our final product is a combination of two repositories and deliverables. The first product that fully implements all the design patterns is a pipeline that takes 3-song FFT values (low, mid, and high) and converts them into a printed shape with features including 3d object type, color, and animation. The patterns used are an MVC for selecting a song and reading the FFT values from a CSV file, an observer used within the MVC, a state pattern that interprets the FFT values to tell the factory to create a certain shape, a factory pattern that creates a certain shape, a shape that implements inheritance for the 3d object kind, a strategy pattern for the animation and a decorator for the color. This is interactable for the user to select a song and see how a visualizer may create objects given a selected song.

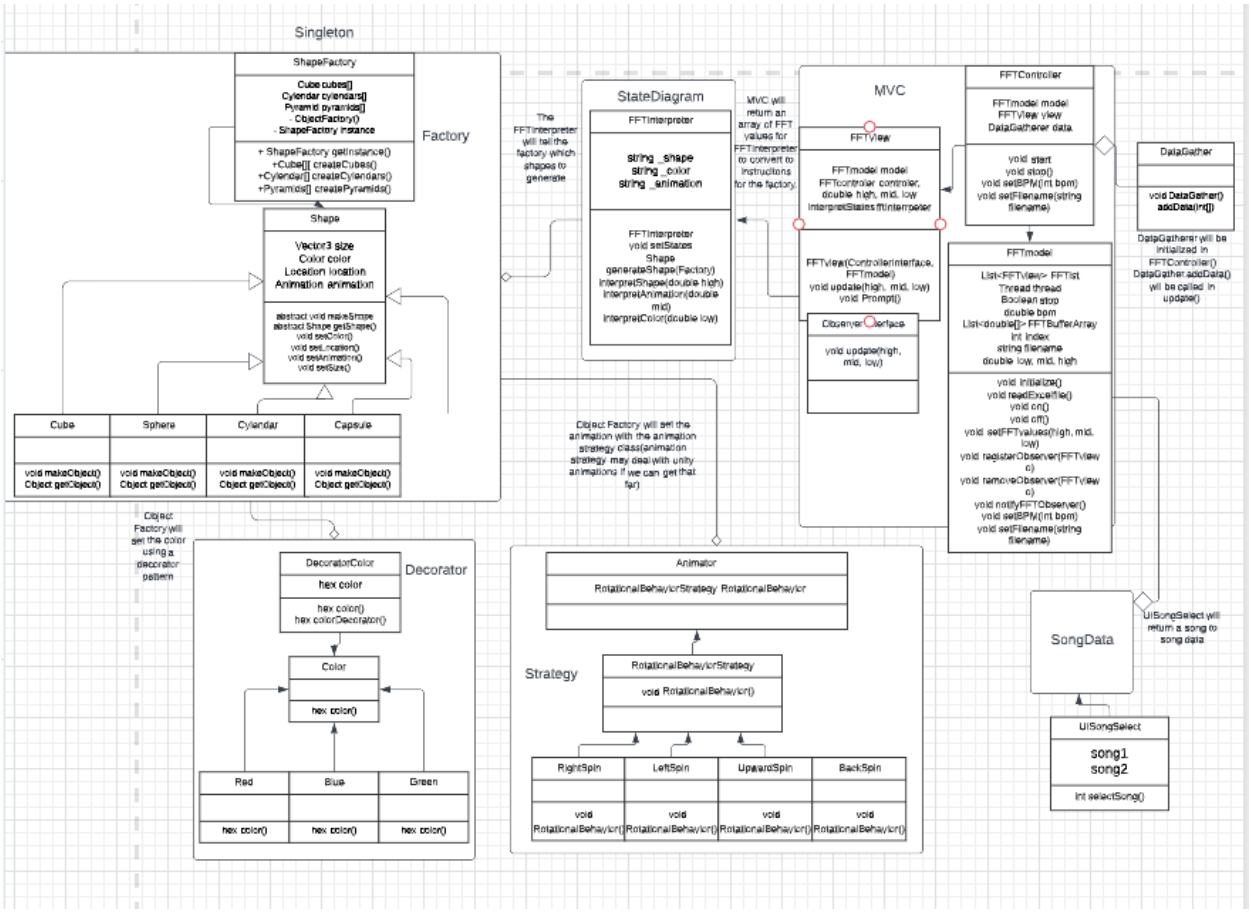
The other product is a unity scene that uses unity libraries to read FFT values from a selected song and generate shapes on the screen given its FFT values (high for the 3D object type, mid for initial rotation and animation, and low for color). These objects' locations are randomly assigned within a limited 3d vector space. This product still uses the factory, decorator, strategy, state pattern, and MVC to create objects on the scene with an extension of a singleton pattern within the factory. The initial plan was to have one object be changed with each beat, but we could not find a way to do this within unity so we just created a new object with each frame and updated the FFT value. While the scene is playing a camera is moving through the vector space at a speed dependent on the BPM of the song. Some other small changes include changing the FFT data type from float to double for the MVC and interpreter to read and use the values easily. We also removed a few 3D shape types like quad and plane due to their unappealing appearance in unity.

3. Final Class Diagram and Comparison Statement

Project 5 UML class diagram



Final project 7 UML class diagram



https://lucid.app/lucidchart/438c4a78-6b86-4956-9901-3d5a8df5c28f/edit?viewport_loc=-174%2C-107%2C3690%2C1702%2C0_0&invitationId=inv_79039047-aa96-4db2-bc23-e579c90a30e7

Changes -
MVC: Controller methods were simplified as it did not need all the functionality that was intended before. The idea of the controller was to stop and start the MVC after the view calls for it to do so.

A lot was added to the model as far as methods and attributes go. We found that the model had to do most of the heavy lifting as far as reading files and updating FFT values. We found that it was easier to preload the CSV files into a buffer array to keep track of the index of where we are in a song if the user decides to pause vs having to reopen the file every time. Before the model was mostly getting and setting data but found that it's real job was to just worry about knowing when to read in data and when to not. We used the observer method in the view to update the view data every time the model updates.

The view was given an interpretState object as well as high, mid, and low values. The change was made so that the view could keep track of its own data as well as send it to the State pattern to be given an animation, color, and shape.

Observer class was removed and turned only into an interface as it made it easier to understand what the view was taking when inheriting from the observer interface. The interface only shows the update method but does not provide implementation.

Factory -
ShapeFactory was turned into a singleton. When sending values from the view to the factory we wanted to make sure that only a single factory instance was being used and that we weren't building more. To be sure of this we made a singleton instance in the factory class and provided a get instance method that returned that instance. This method was used by the view every time the view updated its FFT values. It would grab the singleton instance and create more objects when needed.

4. Third-party code vs Original Code

This project did not require direct copying from other sources however we drew inspiration from the sources listed below:

For all patterns: https://www.tutorialspoint.com/design_pattern/index.htm

Animator: https://www.tutorialspoint.com/design_pattern/strategy_pattern.htm

State pattern(FFTInterpreter):

https://medium.com/net-core/how-to-manage-states-with-state-design-pattern-in-c-d4ca47ec6aa

Factory Pattern:

https://www.tutorialspoint.com/design_pattern/factory_pattern.htm

Opening .csv files in c#:

https://stackoverflow.com/questions/5282999/reading-csv-file-and-storing-values-into-an-array

https://stackoverflow.com/questions/15560011/how-to-read-a-csv-file-one-line-at-a-time-and-parse-out-keywords

Threading in model:

https://learn.microsoft.com/en-us/dotnet/api/system.threading.thread?view=net-7.0

Understanding FFT Values in Unity:

https://www.youtube.com/watch?v=0KqwmcQqg0s&ab_channel=PeerPlay

https://medium.com/giant-scam/algorithmic-beat-mapping-in-unity-real-time-audio-analysis-using-the-unity-api-6e9595823ce4

https://docs.unity3d.com/ScriptReference/AudioSource.GetSpectrumData.html

5. OOAD Process

1. We had to learn how to program in C#, which is very similar to Java and we are impressed with how well we could implement our learned patterns into C#.
2. Unity is challenging, especially if you have little prior experience. However, we were impressed with how well *some* patterns we designed for a text-based environment meshed with unity to create shapes and animate them.
3. Unity did not work well with the initial MVC pattern as the way scripts are added into unity are very object dependent. The MVS was coupled to itself in a way that made it hard to separate it into objects in Unity so I had to rethink how to implement it.
4. It was also difficult to find a framework to add view components to the FFTview. These would include any sort of window pop-ups that would be intractable. I assumed it would all add smoothly into Unity for figuring out UI but I was mistaken.
5. Another difficulty with transitioning to unity was the way we used the thread operations in the model. The model used this thread processing to continually read through the FFT values and using just the .NET framework we determined that reading speed by the user inputted bpm, this way we could actually see the thread working. With unity we have the power to read values every frame which is what we really want but we had to change the way we used time. Instead relying on an arbitrary bpm, we can adjust the timescale in game. With it we learned how to completely pause the animations in unity as well as restart the song.