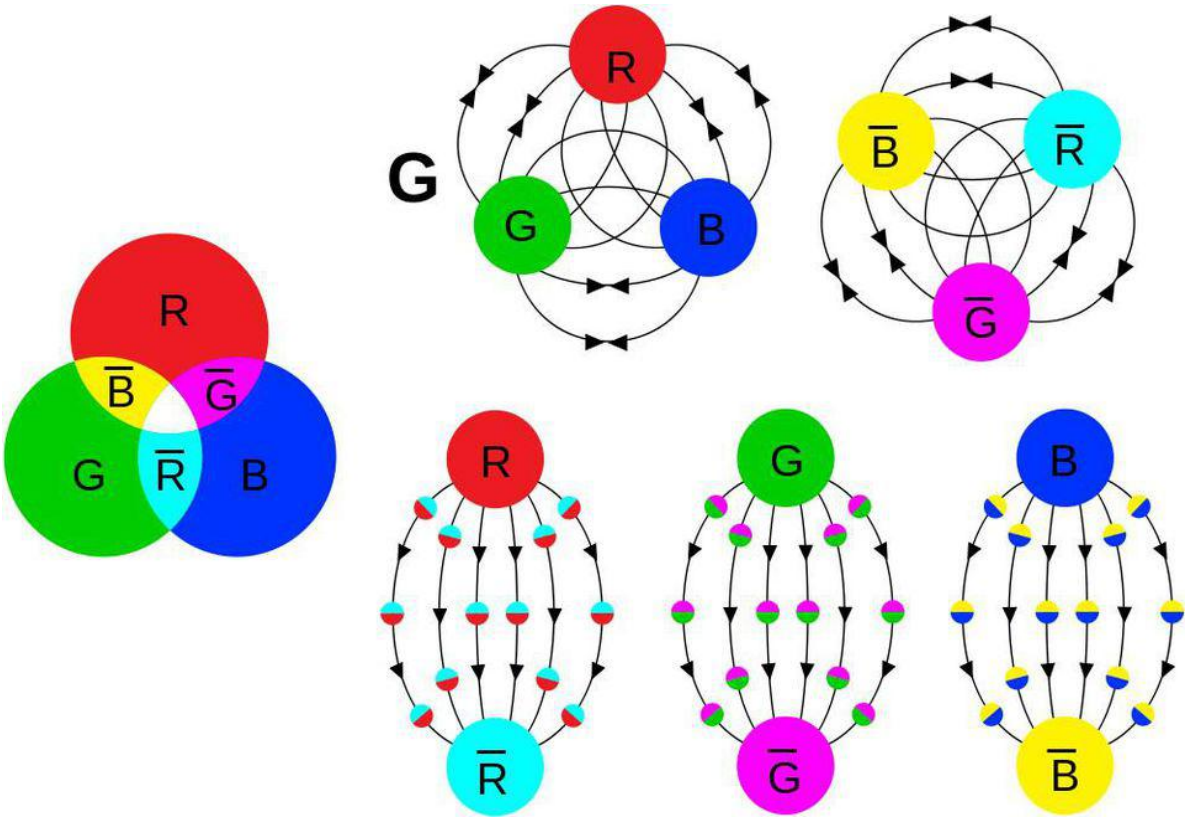
Description of Gluons and the Sea Quarks (non-valence)
[Engineering Breakdown : **Possible Approach** **: Potential Improvement**]

Gluons are the force carriers like the photon in electric charge. Each Gluon has two colors, as indicated on the Wikipedia page under Color Charge. All combinations of gluon colors should be represented. There are three colors and three anti-colors red, gluon, blue, and anti-red, anti-green, and anti-blue. You can use the same color representation as what you see on Wikipedia.



- **Blender Modeled Two-Color Layered Gluon Assets (8 variations in total) and export as.FBX. Using standard shader for compatibility, but could expand it further for aesthetics later.**
- **We will utilize the High Definition Render Pipeline (HDRP) for maximum fidelity and expandability. Switching from default is a pain, so we will have to start out right even if we don't need all the functionality on the get-go. HDRP supports most functions from the default and is massively more capable in many areas that could greatly benefit our project. Here is the comparison and General Information on HDRP.**
- **Use shader-graph offered in HDRP to reduce consumption and streamline development.**
- **Possible expansion and optimization opportunity with Scriptable Render Pipeline. (SRP)**

Each gluon can float around and interact with other gluons. They can do: nothing, attract or repel. When they make contact, they can bounce off of each other, changing from attracting to repelling, or they can pass right through each other. The fluctuations of the swarm of gluons are fluctuations in the population of the swarm, making an appearance of clouds. Gluons of the same color attract and of opposite color repel. Gluons with neither opposite nor same color don't attract or repel but can still interact by bouncing off each other.

- **The interaction of the individual gluons will be scripted in Unity. My default assumption is that we will have to attach a script to each of the gluons since they'll all have a probability of doing a few different things. So a general particle system that only manipulates a volume, pre-rendered animation sequences, and set motion path for animation could be out of the question. However, I got the Unity Technical Director on speed dial. So, if we run into any problems I can't resolve, I will give him a call.**
- **The basic scripting of the individual gluons should be relatively straightforward. For starters, we would have some variables for which of the eight states it is in. We will assign a number value for** "attracting, repelling, and do nothing" **in the script and simply call a randomize function on it, etc.**
- **Each gluon should have a rigid body and sphere colliders on it. We will experiment with how small the colliders can get before it seize to function. (This is merely my anticipation of it breaking, with HDRP this may not happen at all.) We will just utilize the conventional ontriggerenter(), oncollisionexit(), etc. A possible shortcut, enable istrigger() to disable all collisions temporarily for the** "do nothing" **outcome.**
- **A High level "Script Manager" has to be implemented so we could achieve maximum optimized efficiency from scripts. It is adapted from the MVC framework.**
- **A "Gluon Swarm Cloner" script should be dedicated to the instantiation of the gluon particles. My bet is that we want to give the script to an empty and instantiate all particles under an empty as children. Instantiation should be randomized as well. We will use this primary for setting up the initial scene of the environment, and we could let the "Sub-Swarm System" mentioned later in the document take care of the rest.**
- **A "Gluon Swarm Manager" script should be created to oversee the general population of the gluons, closely monitoring the population density and values of the swarm. It would also influence the movement and patterns of the swarm, as I will describe below.**
- **The movement of the gluons should consist of two parts:**

    - **The basic script on individual gluons should apply a randomized force in a randomized interval, towards a direction determined by the colors, by calling Quaternion.Euler(). We will test and experiment with the values as we proceed.**
    - **The swarm manager would control force fields and apply a "gust of wind" or "gravitational pull" to all particles. We can achieve this from many angles and methods, but the important thing is to make sure the representation is as natural as possible.**

The valence quarks are the three quarks that are already in the present build that you saw. They are the only stable ones. That means they are always there and moving around while the sea quarks and gluons fluctuate in and out of existence all around them. The gluons and sea quarks interact most strongly with the valance quarks. This is an important note because you will want to import them to work on this interaction.

- **Please send me all the available files from the previous build. This includes all assets, unity packages, and builds. I will salvage what I can while I'm not having high hopes of its quality. I will be prepared to make everything from scratch, and I have sources for additional assets if needed.**
- **Since valence quarks are described as stable, we will instantiate a set number of them, enough for the presentation. It will have a dedicated script similar to the gluon script, but we will make it more attractive for the gluons to come interact with it.**

Gluons and sea-quarks all have color, and so when they interact, they can change color, but the color rules are always followed. So, the gluon that is red and anti-blue can interact with a quark that is blue and have an exchange resulting in a quark that is red and gluon that becomes blue and anti-blue. These types of things we need to keep track of.

- **After studying this interaction a bit, this should be too difficult of a task. We would make sure to make variables to keep track of the states.**

The gluons should have a variety of lifetimes (from 0.1 to 5 seconds or so). Some should then disappear, and some should then split into sea-quark pairs, which are a quark and anti-quark. They should all be randomly spreading out and twisting around in all types of patterns except when a valance quark comes by. The three valence quarks move around while the gluons and sea quarks follow them as they pass. The valence quarks pull gluons and sea-quarks into existence as well. So the valance quarks are always surrounded by a thick cloud of gluons and sea quarks. There are far more gluons than sea quarks.

- **Make sure to write a suitable destroy() method for the gluons and randomize its lifetime and chances to be split into a quark and an anti-quark.**
- **When the split happens, we will call destroy() on the gluons in its place and then instantiate the two quark and anti-quark objects there and give it initial randomized parameters.**
- **We will have a "Sub-Swarm System" for valence quarks, and it will be attached to all of them to manage the effects of the pull and the instantiation of additional particles. This will be the natural way we generate particles after the initial start for a higher degree of simulation accuracy.**
- **We will definitely model the gluons, quarks, anti-quarks, and valence quarks slightly differently for differentiation in the presentation.**

Additional features mentioned:

"Another thing that is important to keep in mind is that we want to have the user be able to change a lot of the parameters, like gluon lifetime, percentage of splitting of gluons to sea-quarks, the percentage of gluons passing through each other or bouncing off of each other, the strength of the force between gluons and the strength of the attraction of gluons to the valence quarks and sea-quarks. The level of attraction of gluons to the sea quark is proportional to the sea-quark lifetime. The valance quarks last forever so the gluons are highly attracted to them."

- **The interactiveness of the simulation should be reasonably doable to implement. On a computer screen, we can just make a hud with the standard canvas in Unity. If we were to port it to VR, we will make a physical button so the user can interact with the controller, or go the hud route again. I will detail the implementation of each function below.**
- **To control the gluon's lifetime, my idea is to make the variable of the lifetime parameter public to the user. Connect that to either the hud(canvas) on screen or in the environment. Note that want we change here is the "range of randomization." Because of Random.Range(x, y) takes in two parameters for the bounds. A higher upper bound makes the lifetime longer, etc. We either make the user enter the number directly, or make a slider that is connected to the numbers with some presets. Or also, why not BOTH?**
- **In the same vein, I reckon everything else you mentioned can be achieved by doing similar things. These are just values that we used to only give a preset to. We just need**

to make it interactive for the user. The specific details are just tiny differences in the functions, so we will work with them as we implement them. For example, “The level of attraction of gluons to the sea quark is proportional to the sea-quark lifetime.**” Let’s check the sea quark’s lifetime while adjusting the attraction of the gluons, before setting the parameter, etc.**