Generating the visualization of the points:
-Currently, the Möbius CPITCH2 Model is generated in Python by a function that calculates the XYZ coordinates and plots them in an interactable graph.
-Depending on how complex the functions are, you should be able to program these same math functions in Unreal (either through a C++ script or even a blueprint) to output coordinates
-Following the coordinate output, you can input those coordinates into another function to spawn a point in Unreal's Level
-When choosing what the visualization should be, a primitive cube/box would allow the form to be viewed from any angle, but to better optimize your scene, you could use another form like a single plane mesh. Ultimately, this visualization should probably be customizable (and it may even be quite fun to let that point be changeable by the devs or users), because once all the points are plotted and the player is playing in VR, you'll get a sense of how much optimization you'll need to do - and the model that's visualizing these points will be a huge part of it
-We didn't talk about this in the meeting, but the concept of point clouds also may be useful here as a visualization option for the points, and I believe there are plugins that make them function in Unreal Engine.
-To connect the points you generate without spawning thousands of game objects, you may be able to use a line renderer which allows you to input functions and coordinates to draw lines

Interacting with the Point:
-Interacting with game objects in Unreal Engine is quite simple. The player can hit, passthrough, etc, with that object, which causes the collision event, which can then call certain functionalities such as playing a sound. However, be wary of this, because if each point has physics attached that allows it to be selected, and you have thousands of those points, the scene could become unsustainable for VR. Unfortunately, though, I'm not really sure of a reliable alternative to this that's any more optimal. Going forward, I'm going to generalize the process of interacting with the points by describing it as "choosing" the point
-From my understanding of your project, the coordinates generated from the CPITCH2 equations that will be used to visualize each point are also the coordinates input to generate the MIDI sounds. So I recommend the process of **Choose Point -> Determine Coordinates (**either by referencing it from the point itself, or the raycast interaction) **-> send coordinates to the sound generation system**
-To "choose" the point that will generate the sound, this is where we start asking the question of the control scheme. However, the basis will be the same - a sort of digital version of a laser pointer emanating from the player's VR controller (like a traditional laserpointer), headset (like a headlight, so it chooses points they're looking at approximately), or mouseclick (the laser pointer shooting into the 3D scene from the 2D screen coordinates of where the mouse clicks)
-This laser pointer is a ray cast, which is essentially a line that extends from the point of origin and can "hit" things. Essentially, the programmer will define the ray cast, how far it extends, and what to do once it hits something (play sound!). You will want it to look for point, and once it hits a point, it gathers that coordinate and inputs the coordinate data into the sound generator
-But which to use? Controller, headset, or mouse click?

Control Scheme: VR or Traditional 3D Interaction
-First off: These control schemes are not mutually exclusive, and it will likely be easier to prototype the project with traditional 3D interaction before bringing it into VR. As someone who wears glasses, I prefer prototyping in traditional 3D space as long as possible.
-Also, since the meat of your work will be related to generating the coordinates, their visualizations, and the sounds they spawn, you will be able to work in normal 3D space for a while and troubleshooting those

systems before adapting your systems to VR (which I do recommend because it's an *incredible* application of VR)
-The case for VR: VR is at its best when its representing 3D forms that would otherwise only be seen through a 2D lens (like a screen). Being able to step within the musical space will take the application from a 4th wall observation point to exactly what it is called: a musical *space.* You can use things like Unreal Engine’s Spatializer tool to let the student hear each sound in the space as it would be like in real life. They can be surrounded by the mobius strip, which will likely make selection easier, and more fantastical.
-However, VR is simply not as accessible to traditional 3D games, becaus it requires two functional hands and can cause motion sickness, so this is another reason to build for both.
-When building your games: Adding a player (or “agent” as Jacob referred to them) to a scene and giving them movement and a “camera” to see through is extremely easy, and most all game engines have presents/templates that allow you to do that very fast.
-To bring your normal 3D game into a VR game, you’ll want to create a new Unreal Engine project and choose the VR template, then “migrate” your level from your original project to the VR one. You could technically do it in one, but it’ll make adjusting build settings a bit annoying

Installation Options
-Design Lab 2 Longterm Exhibit. It can have two VR headsets (Meta Quest Pro), as well as a traditional game
-Visualization Studio Event for VR, with long-term storage on the Workshops Drive for at-will demonstrations
-MIDEN (VR Cave) Installation. The MIDEN can run unreal engine scenes, and while there are quirks and bugs that can occur, especially when unique plugins are included, and there are also even more optimization limitations, it would be an incredibly immersive experience
-Or all of the above! :D