Every script has necessary comment lines on top of variables and functions. Here is the basic documentation to explain the role of each script.

There are two scenes in the game: *IntroScene* and *TheMainScene*.

IntroSceneController.cs

This script contains the functions the buttons use in the *IntroScene* which includes the scene passing function and “How to Play” panel enable/Disable toggling.

CubeRotatorController.cs

This script rotates the cube in the *IntroScene* to rotate along three of its axes.

TheSwitchController.cs

This script changes the Ak.WWise switches within the game. WWise switches are parameters within WWise to trigger certain events. In this case, allowing different footsteps to be played. Through the array of switch names, the player can go through switches by pressing Q”.

FadeController.cs

This script allows to fade out of the sounds played by WWise. In an *IEnum* function (to be able to fit within frames) a function is called whenever a WWise sound needs to stop. The “*fadeOutDuration*” variable helps us calculate the “*theFadeSpeed*” variable by simply dividing the start and end volume preferences by the “*fadeOutDuration*”. Then, while the Current Volume is bigger than the end volume (which decreases by the factor of “*theFadeSpeed*” in each frame) the volume goes down. When it is out of the *while* loop, the volume is set to 0 just in case there is a fraction of volume left, then stop the WWise sound to let the WWise know that the. The event has ended (this is necessary for WWise operations) and then the Volume is back to maximum afterwards.

DronePositionController.cs

This script lets the “Drone1” fly randomly on top of the player. The user can set the boundaries, the specific area it travels in and the speed of movement from the Unity inspector. *Start()* sets a random Vector3 position for the Drone1 game object to go, and then in *Update()* the drone gradually goes to that position. Once it is close to that position, a new random position is set.

ButtonsController.cs

This script allows the player to hide “Drone1” or change between drones by pressing specific keys.

Both actions are based on toggling mechanisms.

MouseDrawLineController.cs

The heart of the game is this script. This seems complicated script does one main thing: start drawing and continue drawing if the mouse button is pressed, and stop drawing if not. The "LineRenderer" game object is used for this and Unity's default line renderer functions are called when necessary.

When the mouse button is pressed, the WWise sound starts to play. It has a callback function defined to it to have an async operation to control the position of the next line. I will explain this further in the next lines. When the button is released, the "*FadeController.cs*" is called to fade the sound out and stop it afterwards.

One other crucial thing this script does is calculate the mouse velocity by taking the previous point in the line, and the current one and dividing it by the time.deltaTime (frame-accurate time). This variable is then passed into two other routes: smoothing it with Linear Interpolation (a great way to smooth or control the change) and getting the magnitude, leaving it raw and getting the magnitude. The smoothed float value is passed into the WWise RTPC value control (Real-Time Parameter Control) which controls the harmonizing effect and the low pass of the WWise sound. So the faster the mouse is, the more the Wet Volume (%100 affected sound) is added to the mix with filtered output.

The unsmoothed value is passed into a function which changes the attributes of the material on the line. Since it has its own linear interpolation with different parameters, it needs to be raw.

The smoother velocity float is also used to see how many times it exceeds a velocity threshold, and during each drawing, if the value is exceeded, change the sound played by the next line to a more excited sound with arpeggios added to it.

To keep the line renderer object in front of the user and the tip of the mouse everywhere the player goes was a challenge. For that, a "*theSectionEnded*" flag is used. Each time the WWise Sound Event ends, the async callback function is setting the *theSectionEnded* to true. This then enables the line renderer to follow the mouse movement within an imaginary sphere around the player. So, the line always starts at the tip of the mouse within a fix distance to the player. When the mouse button is pressed, the line renderer position is locked.