This document will walk the major comments within the blueprint top-to-bottom, left-to-right, across the EngineerRoom\_BlueprintA Event Graph. I’ll denote if the comment block was owned by another programmer by placing “N/A” on the bullet point.

* Messages: N/A
* Wire Sorter: I did some simple math to find our wire minigame had 3^2 possibilities. The wire connected a vertical port to horizontal port, and there was three horizontal and three vertical ports. The game would generate a number between 0 and 8 inclusively to represent which configuration the user had to set the wire to. I then had to find what nth configuration the user had switched to by moving the vertical wire up or down, or horizontal wire left or right. It was fun finding the solution on paper and translating it to a blueprint code. It was also exciting using the modulo operation for a gameplay mechanic.
* Button Masher: This code evolved a lot over time. It was originally intended to be a single button to mash. Then it became a set pair. Then the designers finally settled on a mixed pair between a random left, and a random right button combo. Start button masher plays the animation for the player to expose the buttons, and picks the count to mash, and the buttons. We set the masher to ensure alternating presses made progress, pressing a single correct key multiple times wouldn’t count for more than one press, and pressing an incorrect key would add to the number of times you need to press. The tail end of this comment belonged to other programmers, and the God Mode additions to the code were made by another programmer towards the end of the project.
* Init
  + Setup Button Masher Input Arrays: these were the arrays of keys able to be used by the button masher.
  + Setup Camera: The engineer player has 4 distinct camera locations that can be set by the player. This sets the camera to the default position looking out the window.
  + Set Wire Array: The wire sorter used 9 separate models so the artists could make them all hang in fancy ways and appear to stretch to different locations. This array stores each of them.
  + Set Hacking Text: The hacking text is the text that indicates that the engineer character is “hacking.” This text is set to a blank string to start, and the text renderer is updated.
* Event Tick: N/A
* Endless Runner Update: N/A
* Move Pachinko Dropper Using Left Analog Stick: N/A
* Camera Movement: This is where we checked and set movement for the camera. The magic number of 1⅔ was decided as a multiple for the joystick values to indicate if the player was trying to move the camera, or not, after rounding the result. If a direction was found to move, it went to the various switch statements below to pick what the new position would be based on the current position. Then the destination location, rotation, and lerp timer are all set, and the camera lerps to its new position. This is a confusing way to do this movement, and looking back now I wish I would have refactored this code to make it more clean and readable.
* Update Gameplay Stage: This variable was decided to be used with the next block to help determine the player’s current objective. It checked states of certain entities that when present/destroyed indicated that a new objective was to be assigned. This should have been done within the respective entities, but time was short at this point of implementation so I came up with this band aid solution.
* Update Distance Text: Depending on the stage in the gameplay, this would track the distance between the player and their closest current objective. The distance unit number was then updated with a prefix, and postfix, set by our designers, before being set to display on the engineer player’s console.
* Endless Runner: N/A
* EMP: N/A
* EMP button event: N/A
* Boeing 747 code: N/A

All of the functions are owned by other programmers, save one exception. The wire sorter check was written by me. This did some simple math to determine how many moves the current wire configuration was from the correct configuration. This distance (between 0 and 4) determined how disrupted the other player’s screen was.