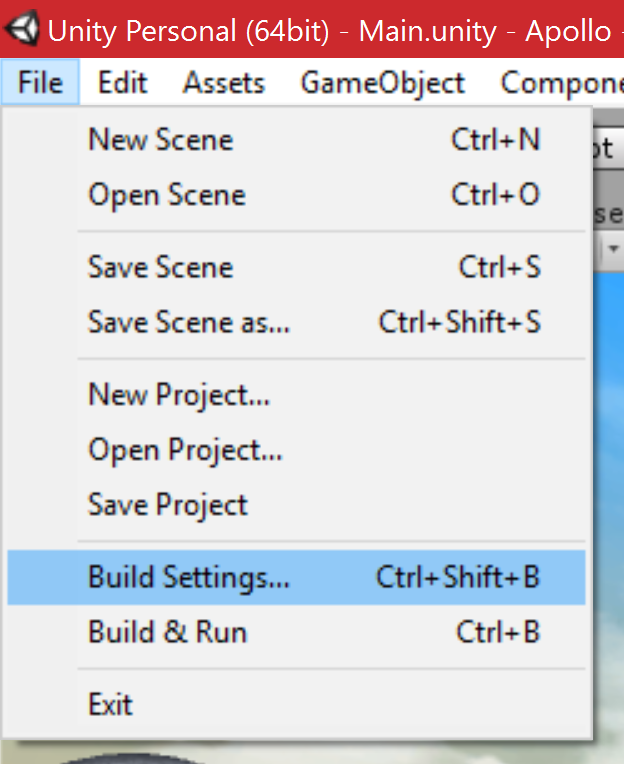
Challenge 3: Events and Pipeline Stages

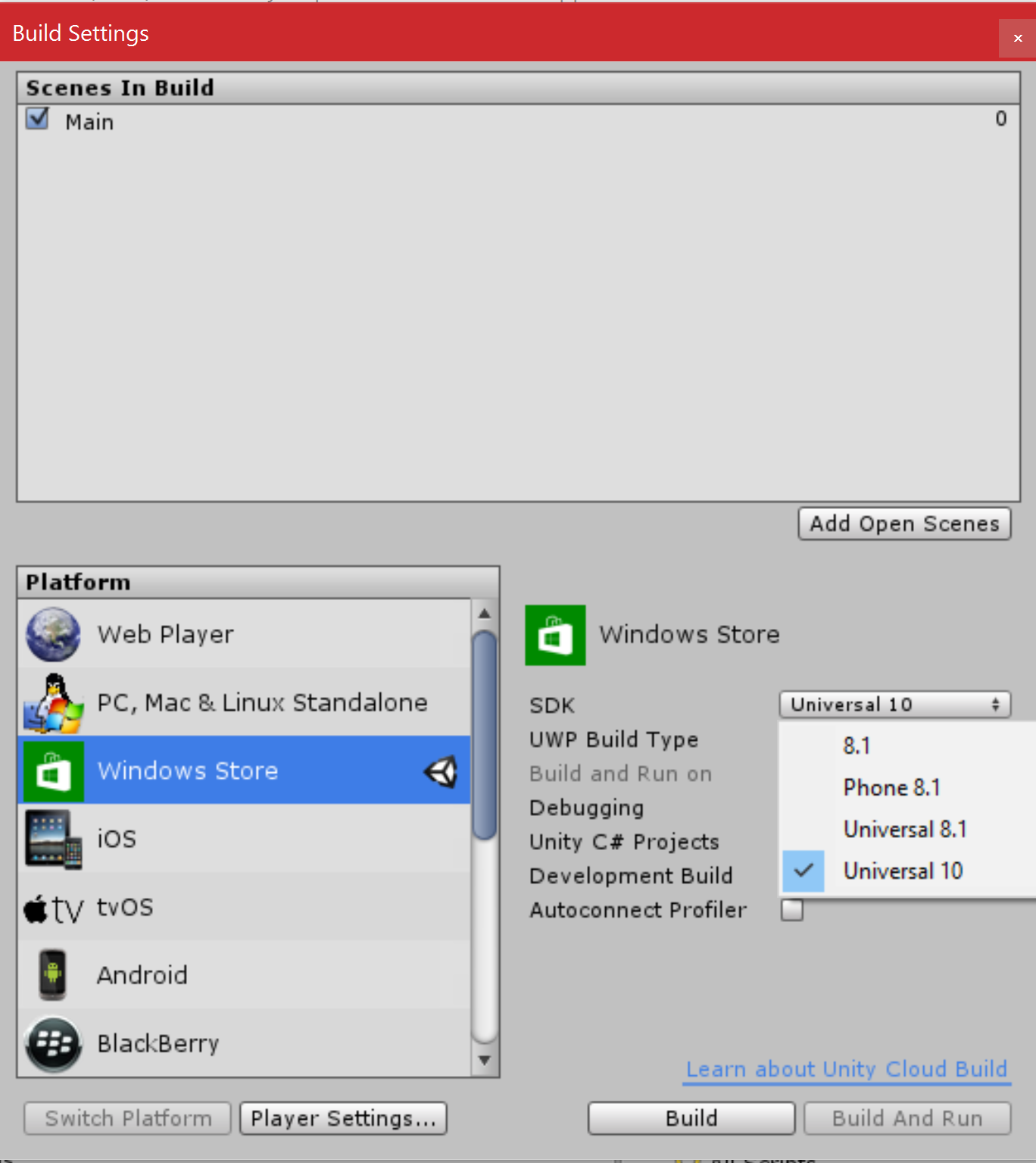
**Step 1: Export the UWP**

By this stage, we are already familiar with how the game works and how to export the game from Unity and attach the Graphics Diagnostic Tools. The aim here is to learn how the Event List works, and how you can use it.

Export the game from Unity

In Unity, click File->Build Settings  


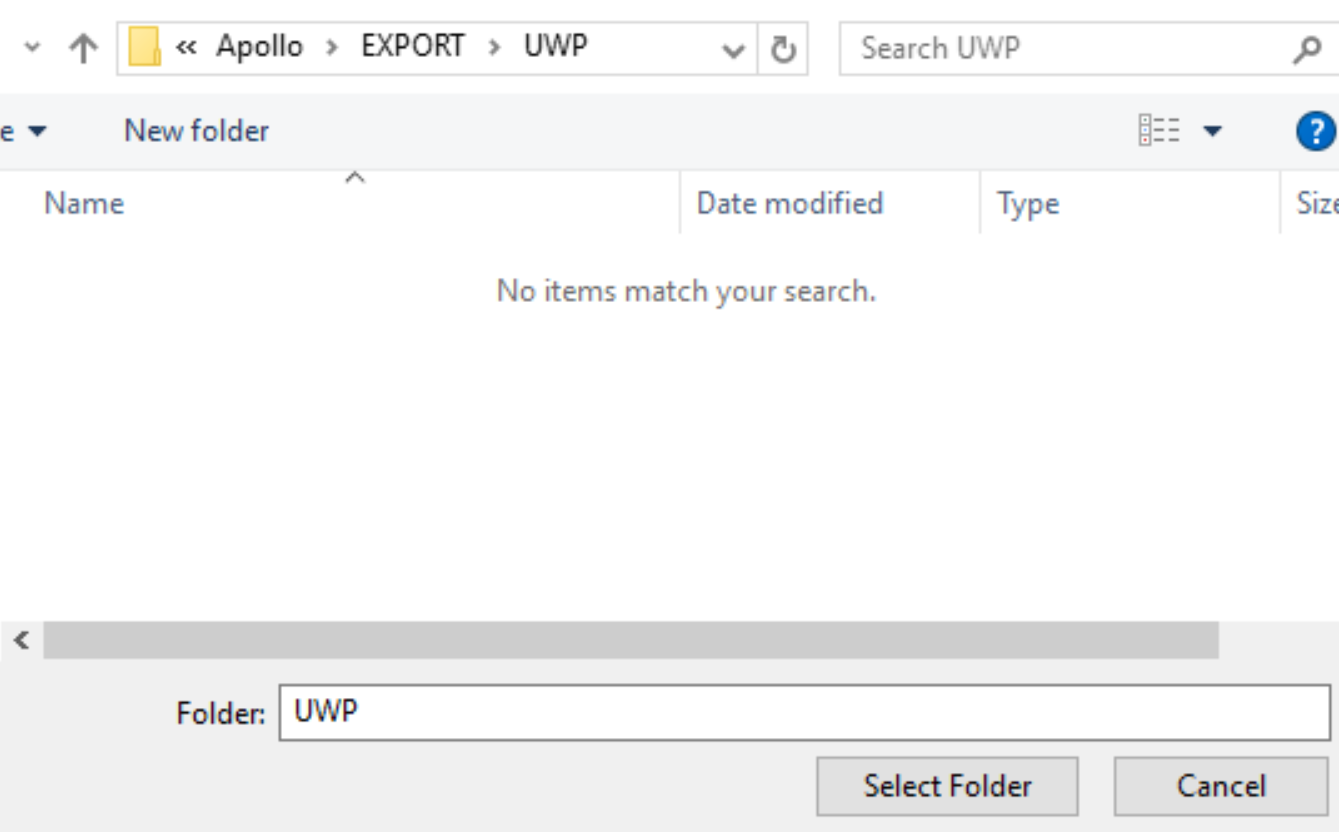
Ensure that Windows Store, and from the dropdown, Universal 10 is selected



Also, make sure the Unity C# Projects and the Development Build checkboxes is checked.

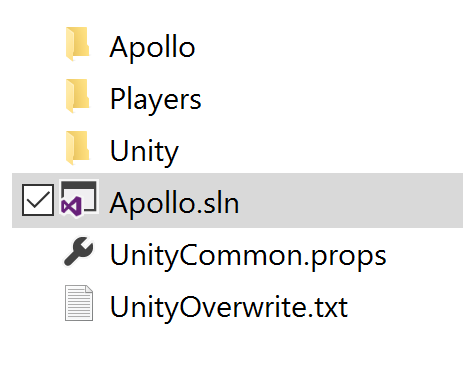


Click Build and navigate into EXPORT\UWP and click Select Folder to start the export.

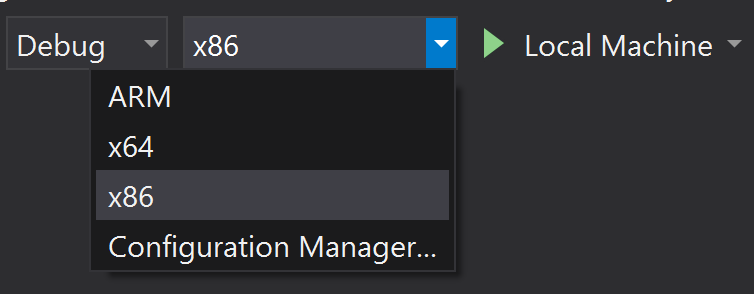


**Step 2: Opening the Visual Studio solution**

Once the export is done, the new folder should automatically open. Open the Apollo.sln.



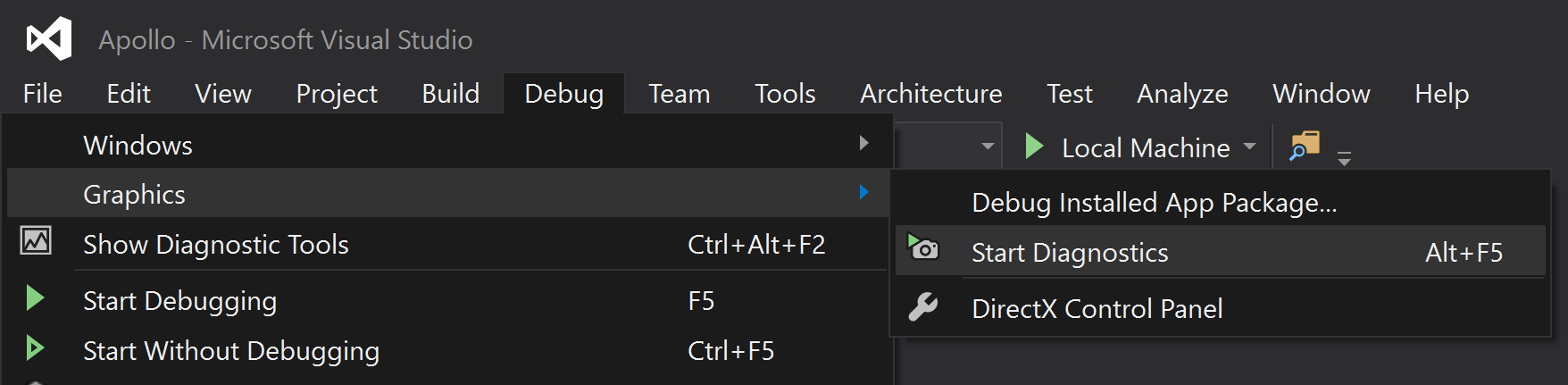
Once Visual Studio loads the project, change the build configuration to x86, and make sure that it is in Debug. Click the Local Machine button to build, deploy and run the solution and test that it works:

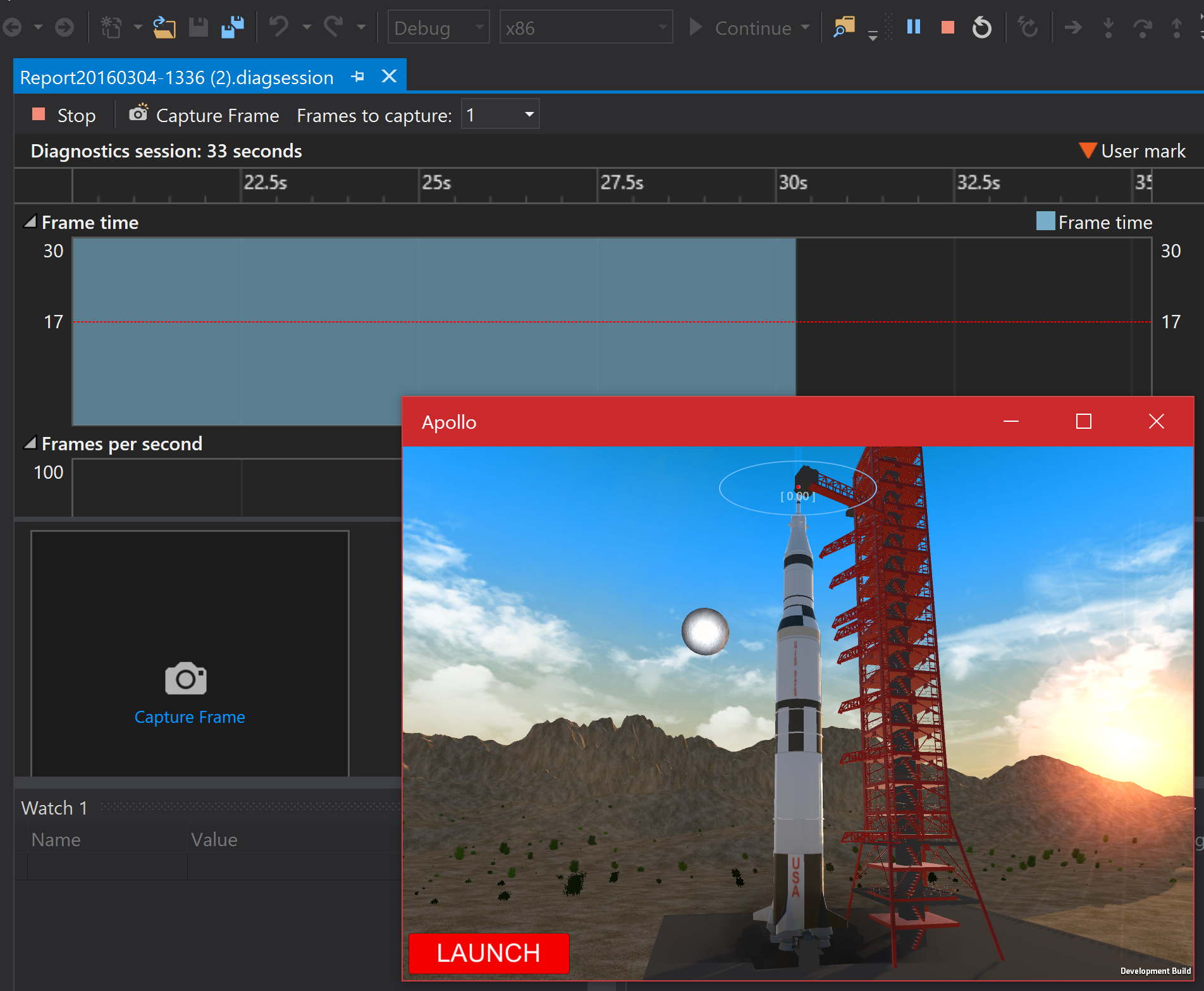


Note: Some build error messages might appear since this is the first time we are building the solution. This is normal as long as the build process is continuing.

**Step 3: Run in Graphics Debug mode**

Let’s run this with the Graphics Diagnostics by clicking Debug->Graphics->Start Diagnostics



When running the game now, you are presented with performance graphs, as well as a Capture Frame button:  


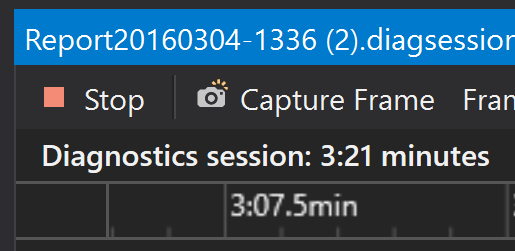
To capture a frame, you can click the Capture Frame button, or press Print screen on the keyboard when having the app in focus.

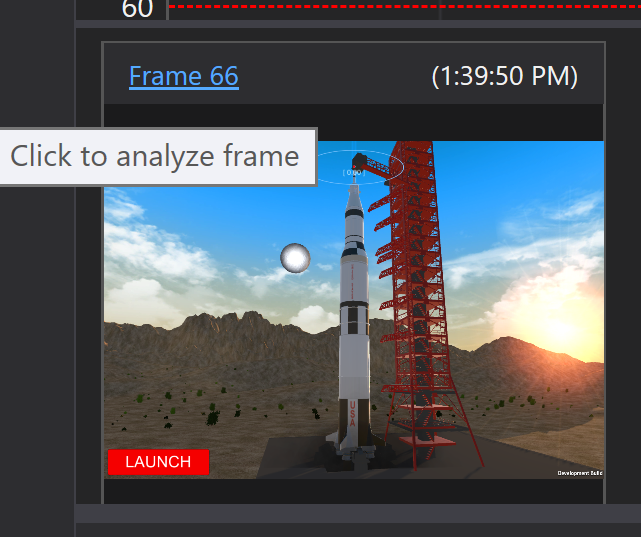
We are interested in finding out what’s happening with the moon. When the game starts, it’s stationary on the Launchpad, and the broken white moon is visible in the background.

When the moon is visible, capture a frame by clicking the button or using Print screen.

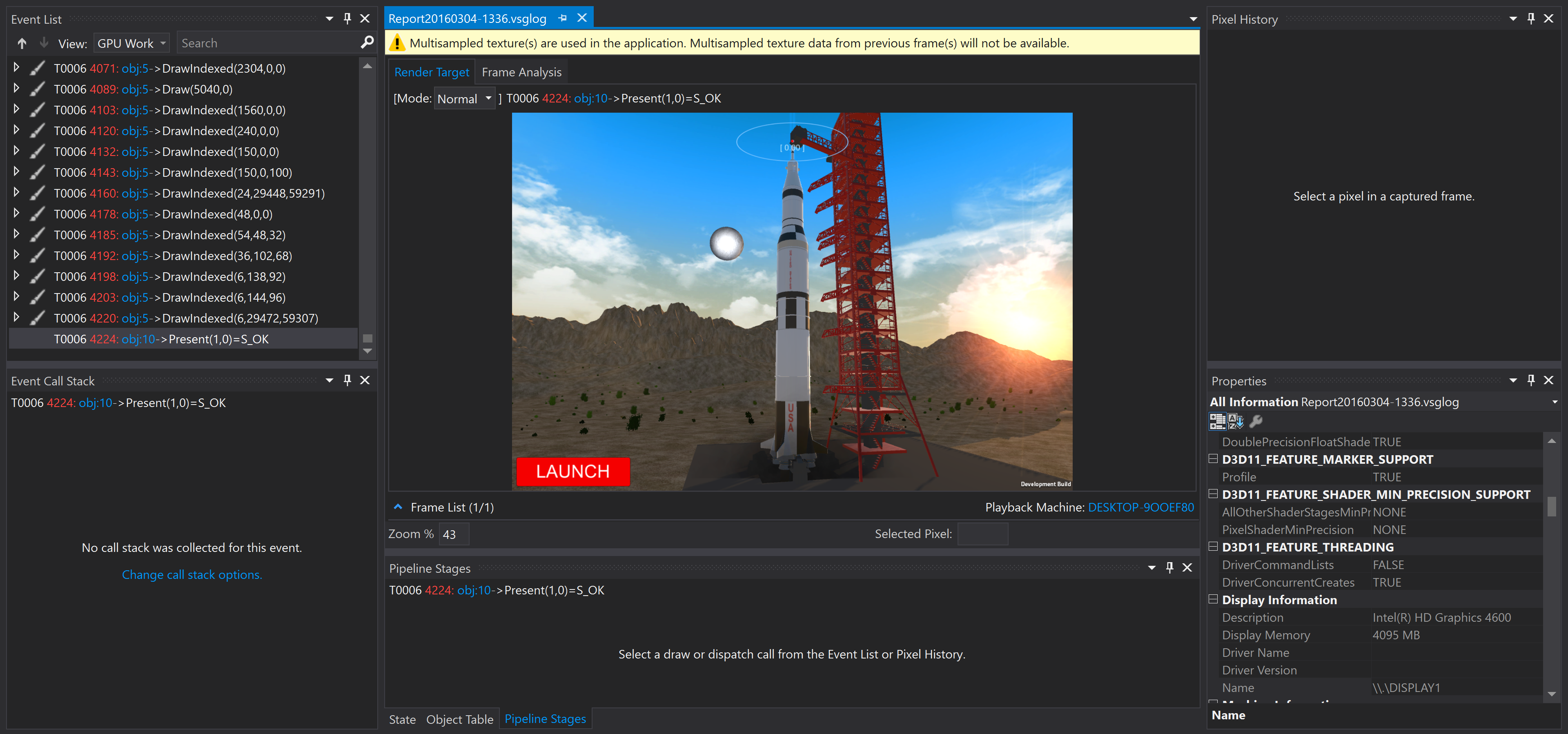
Then, when the frame is captured, you can see it visible where the capture frame button was.

Stop the session by clicking the Stop button:



Once the app stops, you can click the captured frame by clicking on the frame header:  


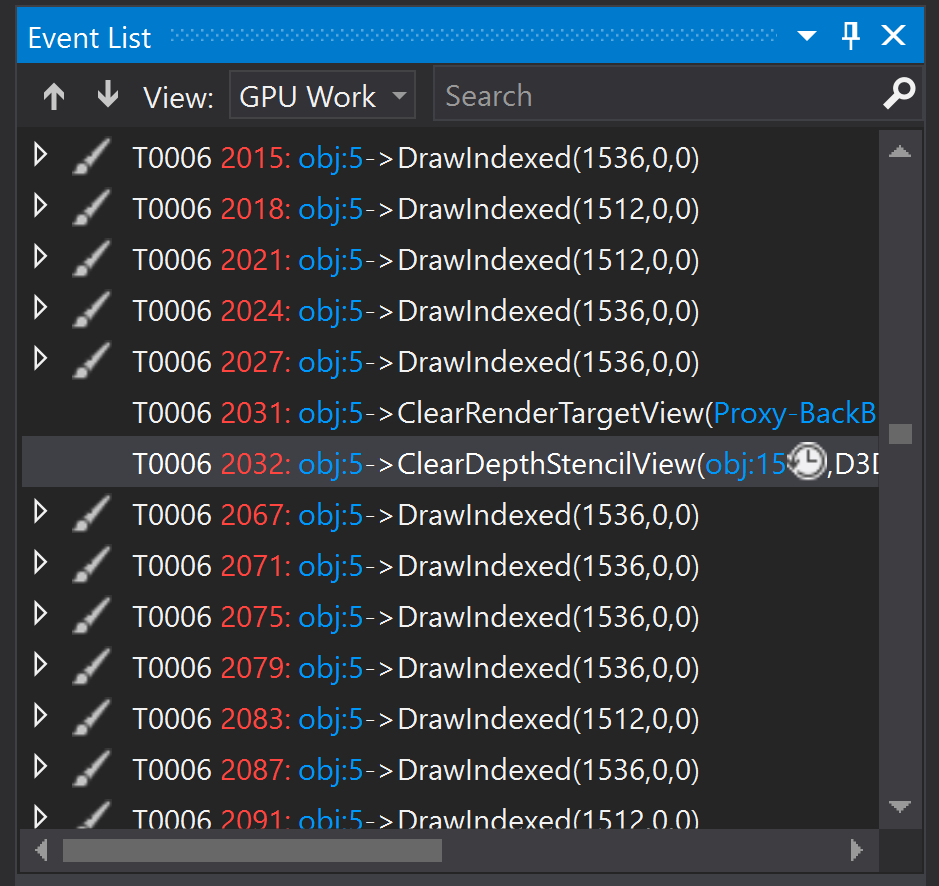
A new screen will open in visual studio, containing a snapshot of the system, processes, state of the app at that time the frame was captured.



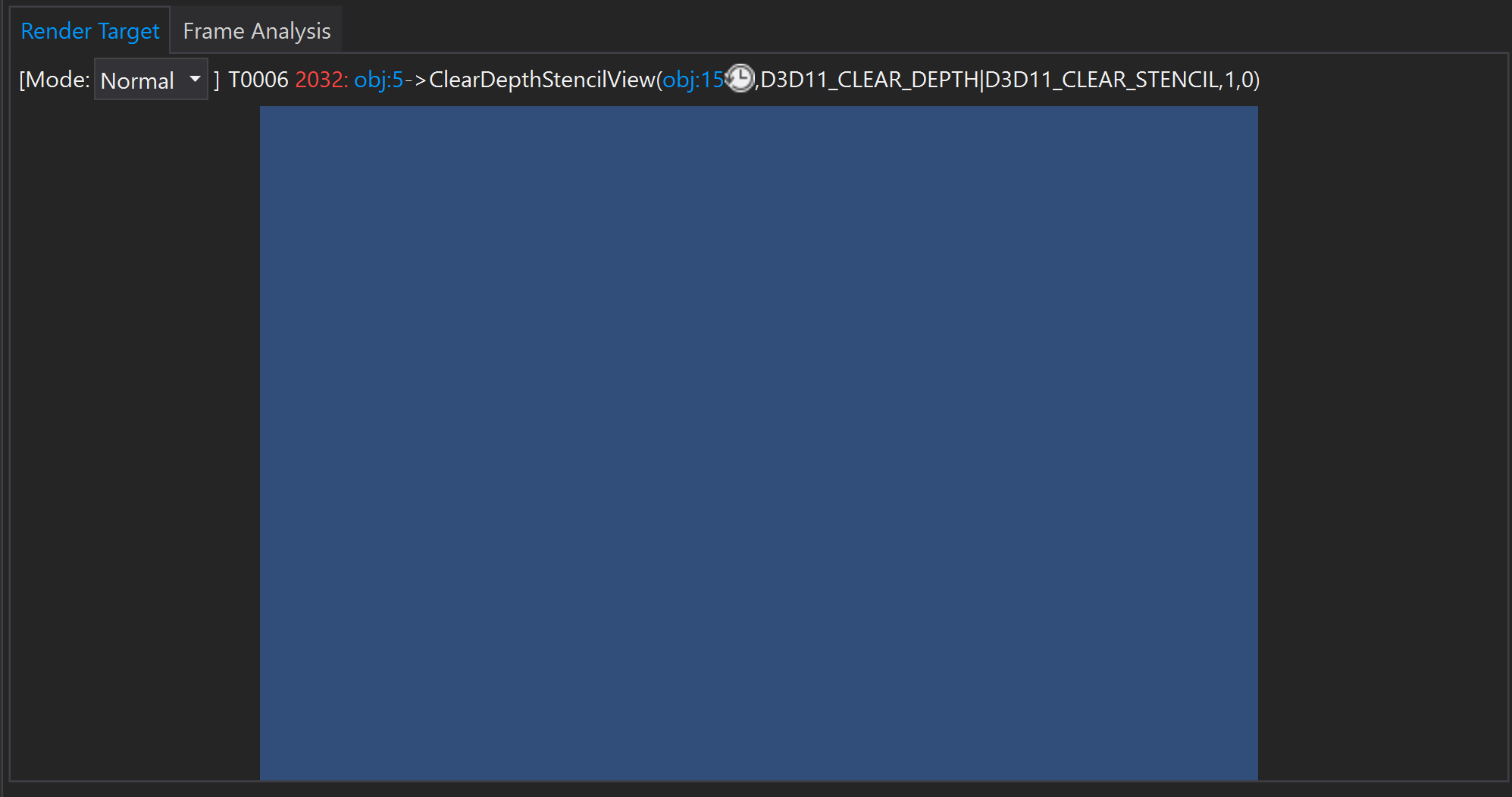
On the left side, you can see a long list of events. You can click these events to see how the frame looks at that event.

**Step 4: Finding the right clear event**

Let’s try this. Scroll up to a bit over the middle of the list view, and you will see an event that clears the screen to a given color:



Clicking this will show how the frame looks to the right of the list (in the preview):



If you follow the events down, clicking the one by one, you will see that events draw what part of the frame like the terrain, the launchpad, the rocket, the particles, the UI and so on.

If you do this one by one (don’t do this), you can see the frame being built up slowly, piece by piece.

**Step 5: Finding the draw calls that belong to our tower**

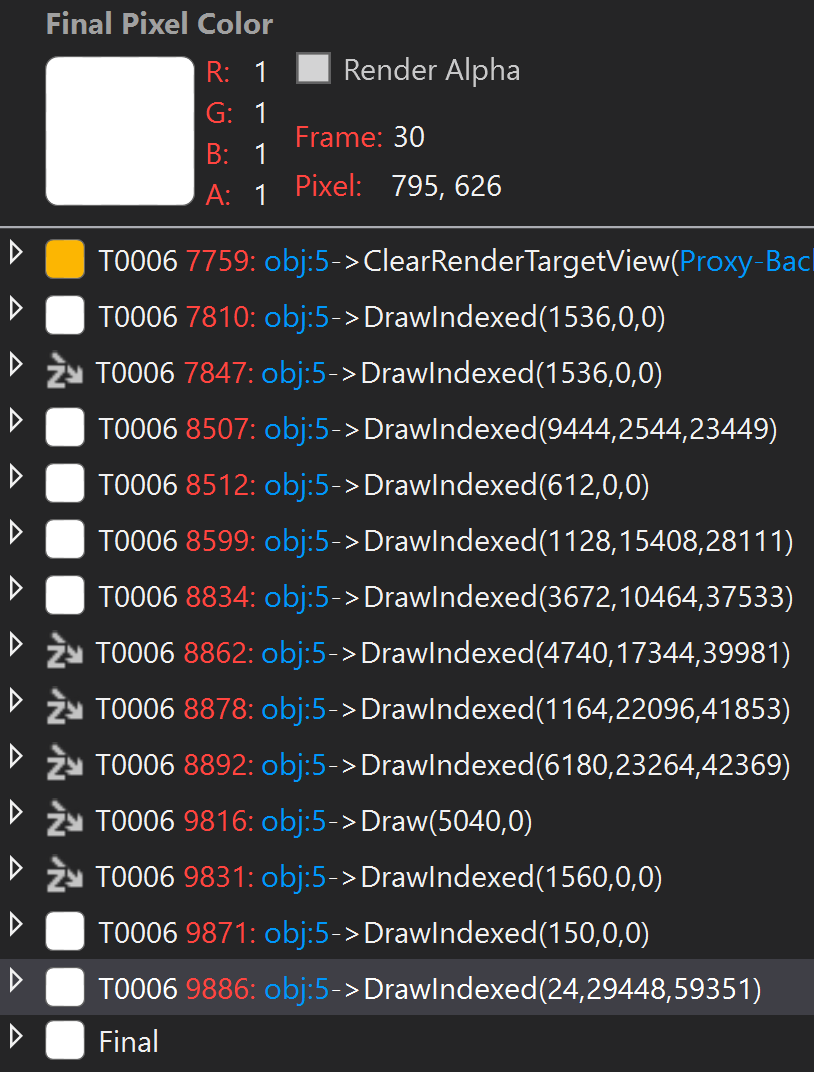
By clicking through the event view, you will first see that a lot of them belong to the terraing, and then another bunch that belong to the tower.

Another way of finding where about the tower draw events are, is by using the pixel history (as we used in Challenge 2)

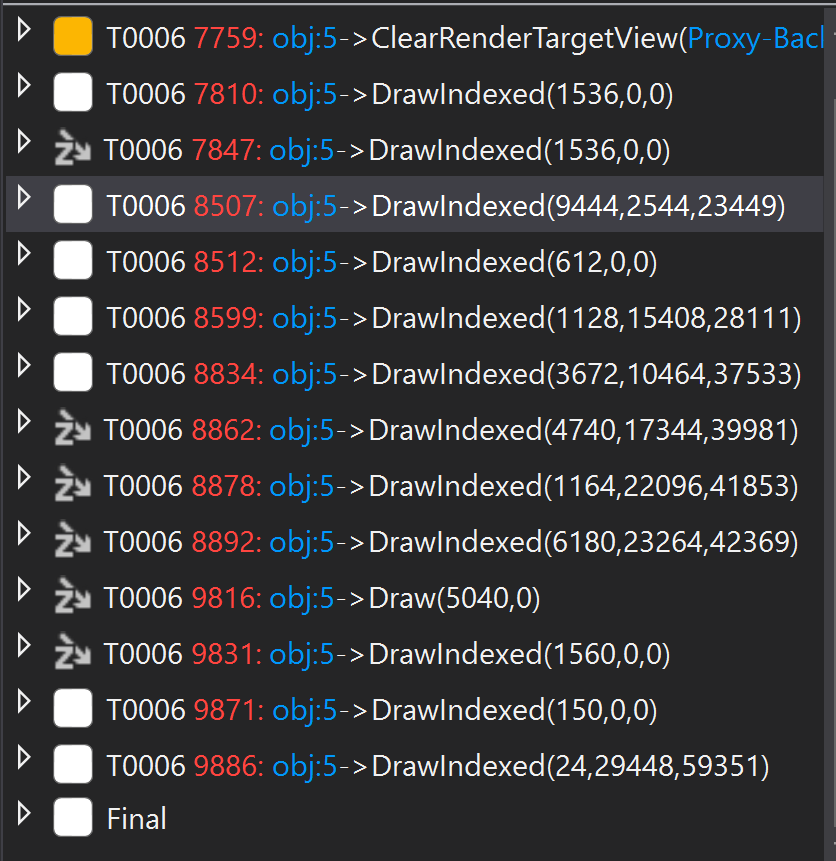
In the frame view, click on a pixel on the Launchpad tower to view the pixel history:



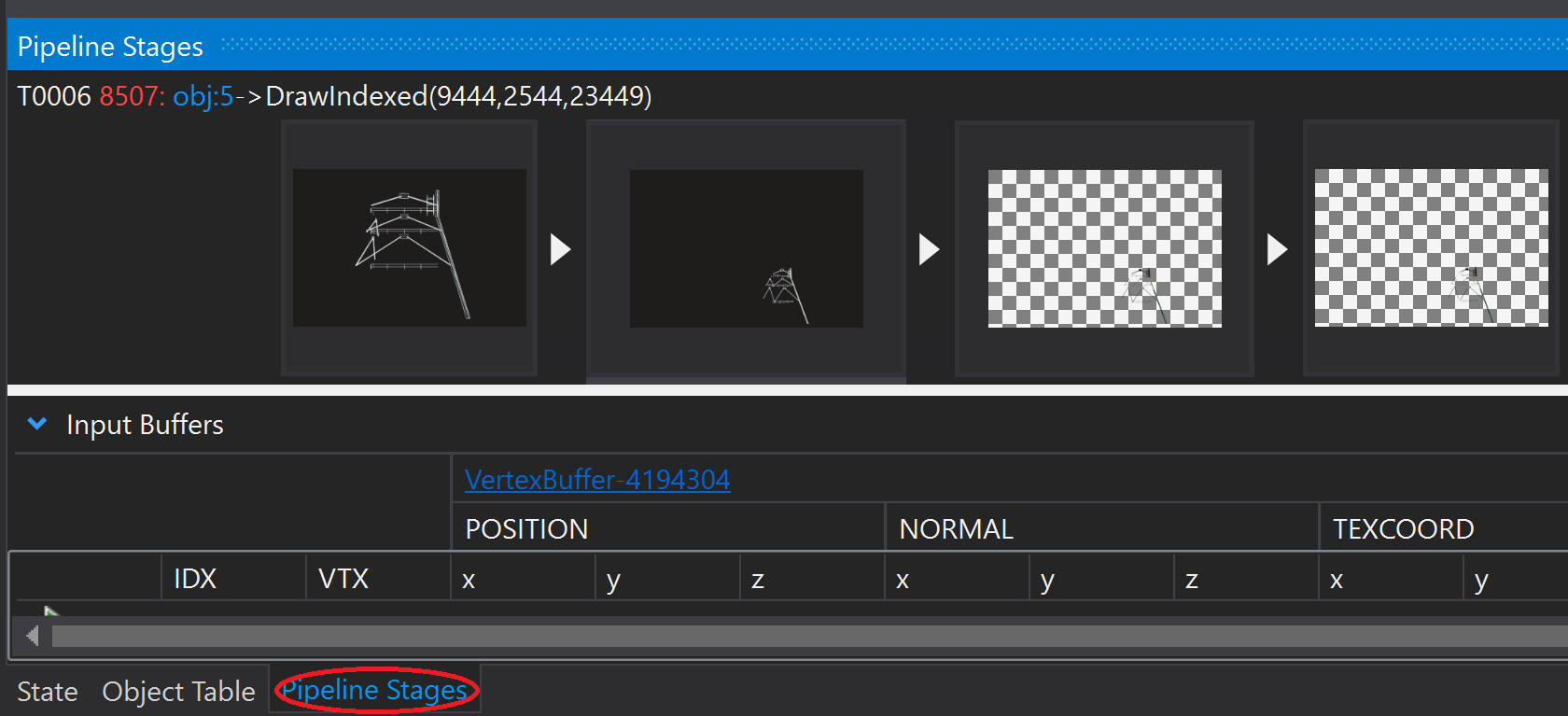
You can see that there are many events that are affecting the pixel:



Select one by one, starting from the top to find one that belongs to the tower:

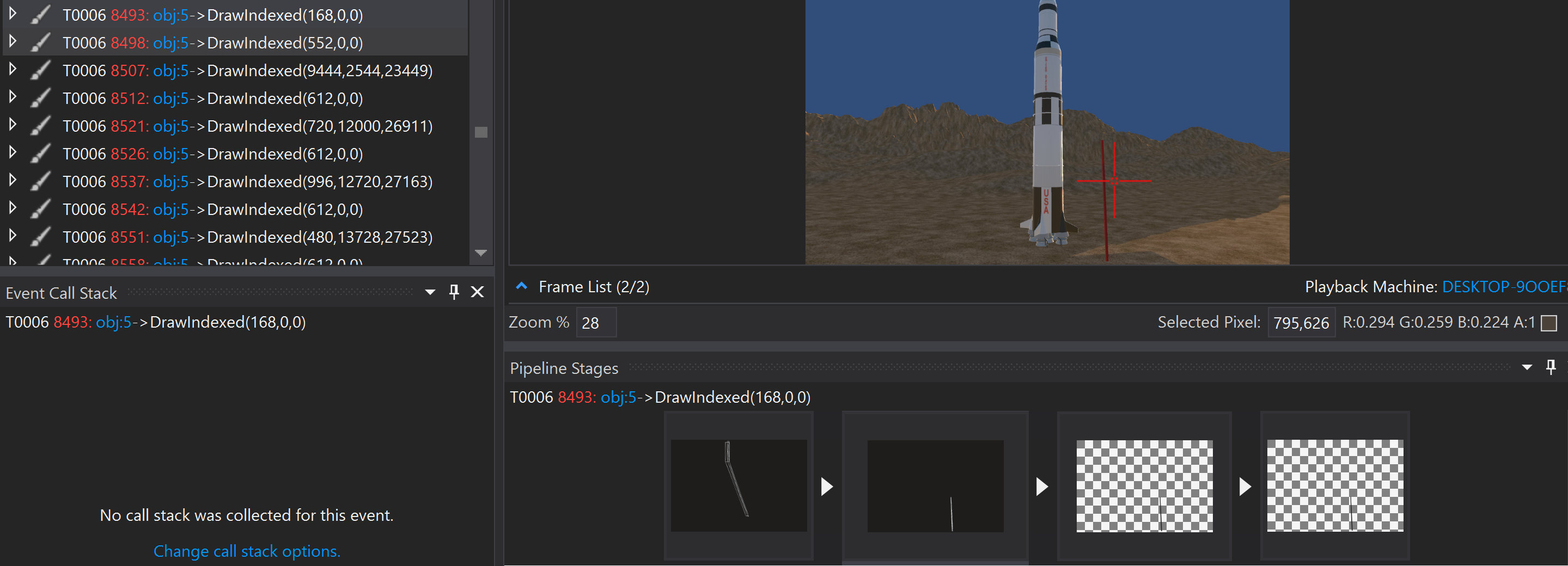


In my case, there are many that belong to the tower. You can identify this by seeing that a part of the tower has been drawn on the frame view, and if you select the Pipeline Stage tab, you can see the part that is currently being drawn:

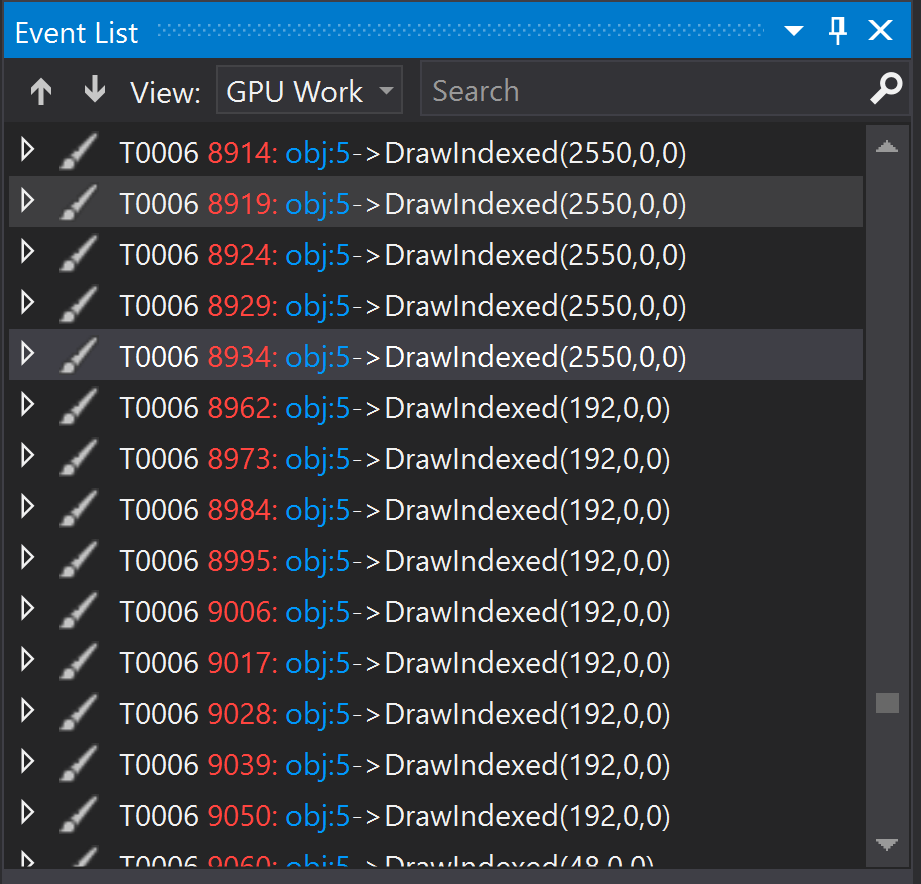


**Step 6: Finding all the draw calls for our tower**

Once you have found one part that belongs to the tower, move upwards to you find the first drawcall for the tower, and move down until the tower has been completely drawn.



The last part is found here:



As you can see, there is a lot of draw calls for making the tower. Sometimes this is ok and necessary, but in this case, we are drawing parts of the model multiple times, with the same shader and graphics pipeline stage.

**Step 7: Reducing the draw calls of our tower**

Go to Module4-GraphicsDiag\Source\Ex3\Launchpad Model, copy the model file, and replace the Launchpad model in Apollo\Assets\Models. This model has been changed by the designer, so the similar parts has been merged.

**Step 8: Test**

Export the UWP again, and capture a frame and look at the reduced amount of drawcalls.