Challenge 1: Profiling

**Step 1: Welcome!**

Open the Project in UNITY and play the game a few times by clicking on the PLAY button. You control the rocket by pressing A and D on the keyboard, and launch by pressing SPACE or click the Launch button on the user interface.

Above the rocket, there is a path indicator:  


Your job is to keep the red dot as centered as possible (so the offset reading below the marker stays as close to 0.00 as possible). There is an in game line that also will show where the path goes:



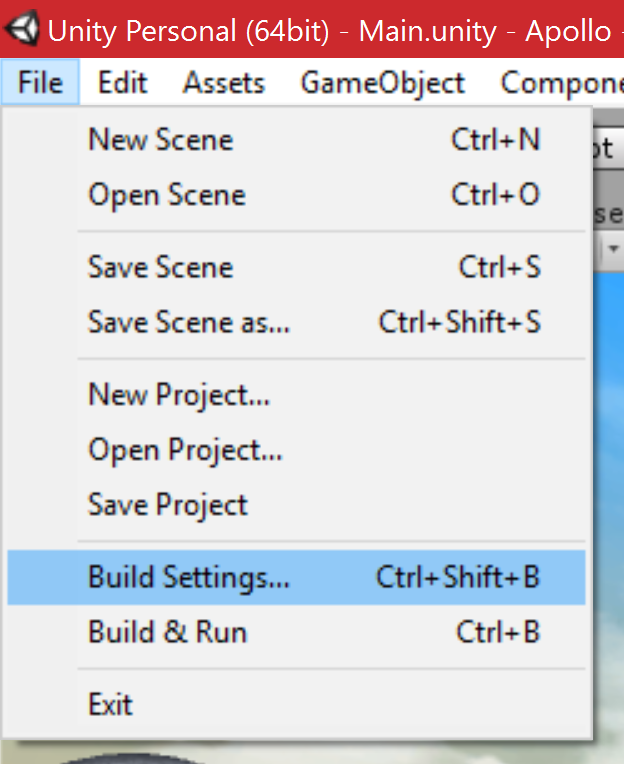
The launch conditions are windy, and the wind will try to push you off the path. Use A and D to slowly rotate the rocket back on track.

One round lasts for about 30 seconds, and you will be measured by how close you are to the path when you cross the finish line.

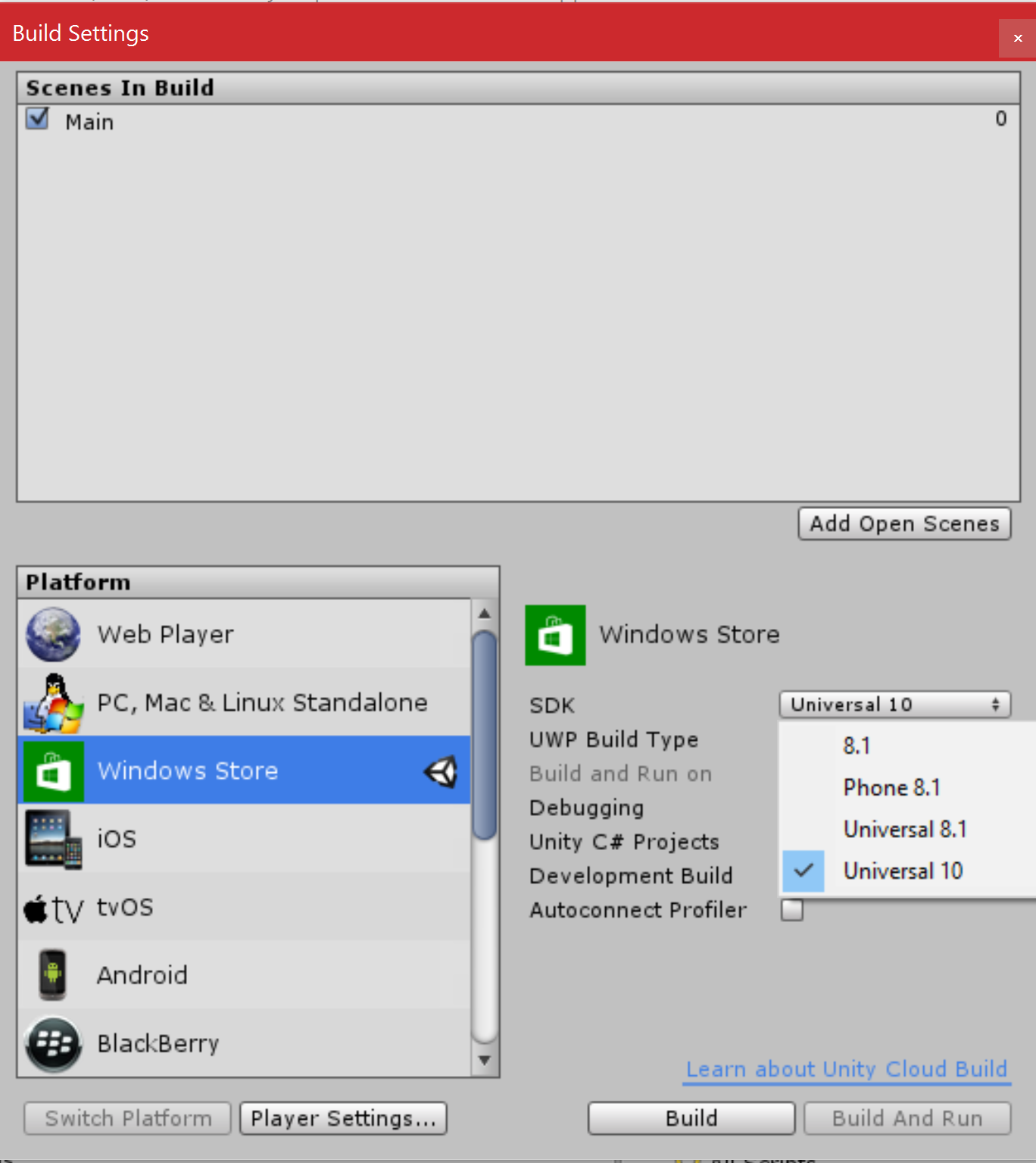


**Step 2: Exporting the game as a UWP**

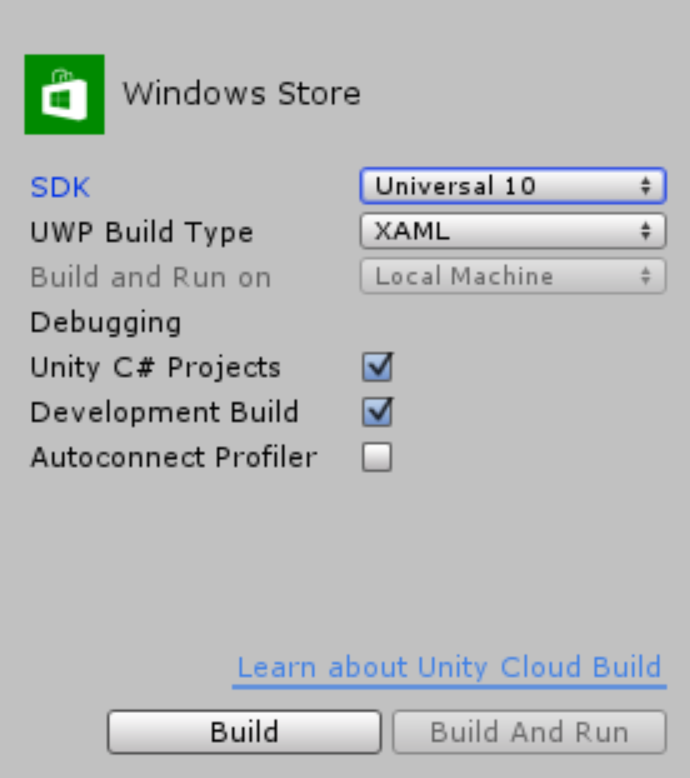
Now that we know how the game works, we want to run it through a performance analysis to see if we are having any performance issues.

In Unity, click File->Build Settings  


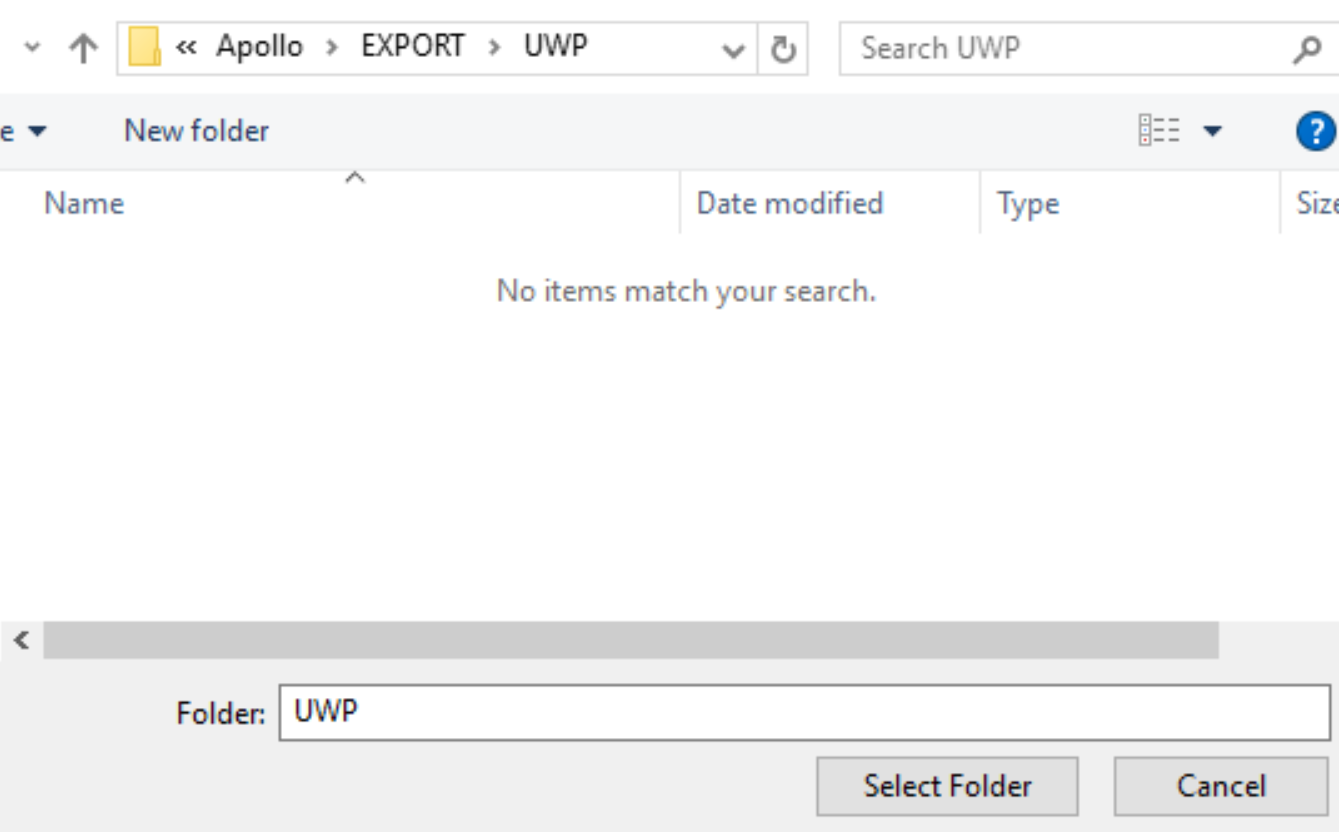
Select Windows Store, and from the dropdown, select Universal 10



Make sure you also tick the Unity C# Projects and the Development Build check-boxes.

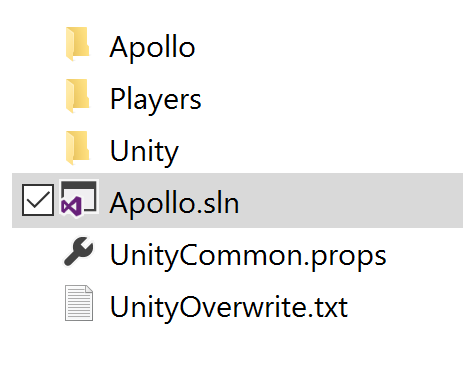


Next, click Build and create a subfolder in the rood project named EXPORT, and another one inside the new EXPORT folder called UWP and click Select Folder to start the export.

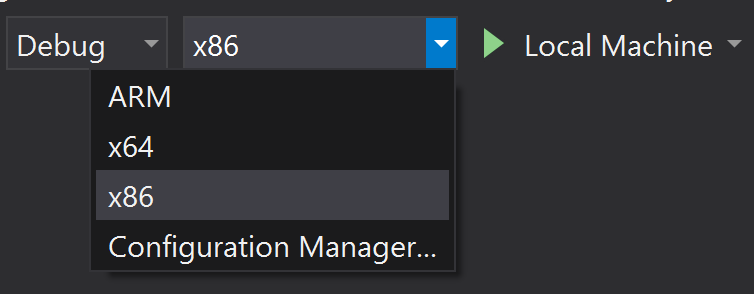


**Step 3: Opening the Visual Studio solution**

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:



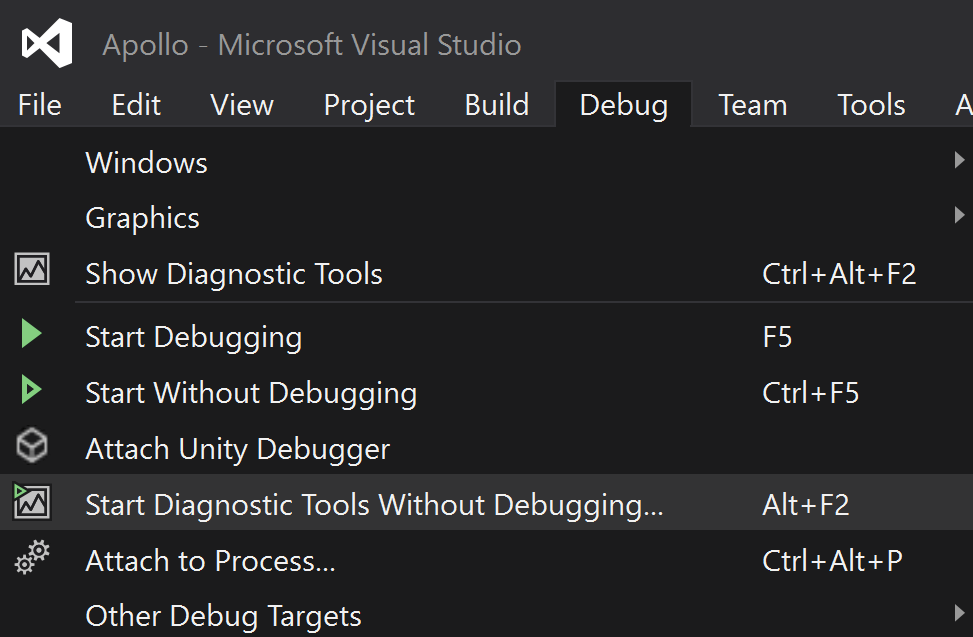
Note: Some build error messages might appear since this is the first time we are building the solution. This is normal as long as the build process is continuing.

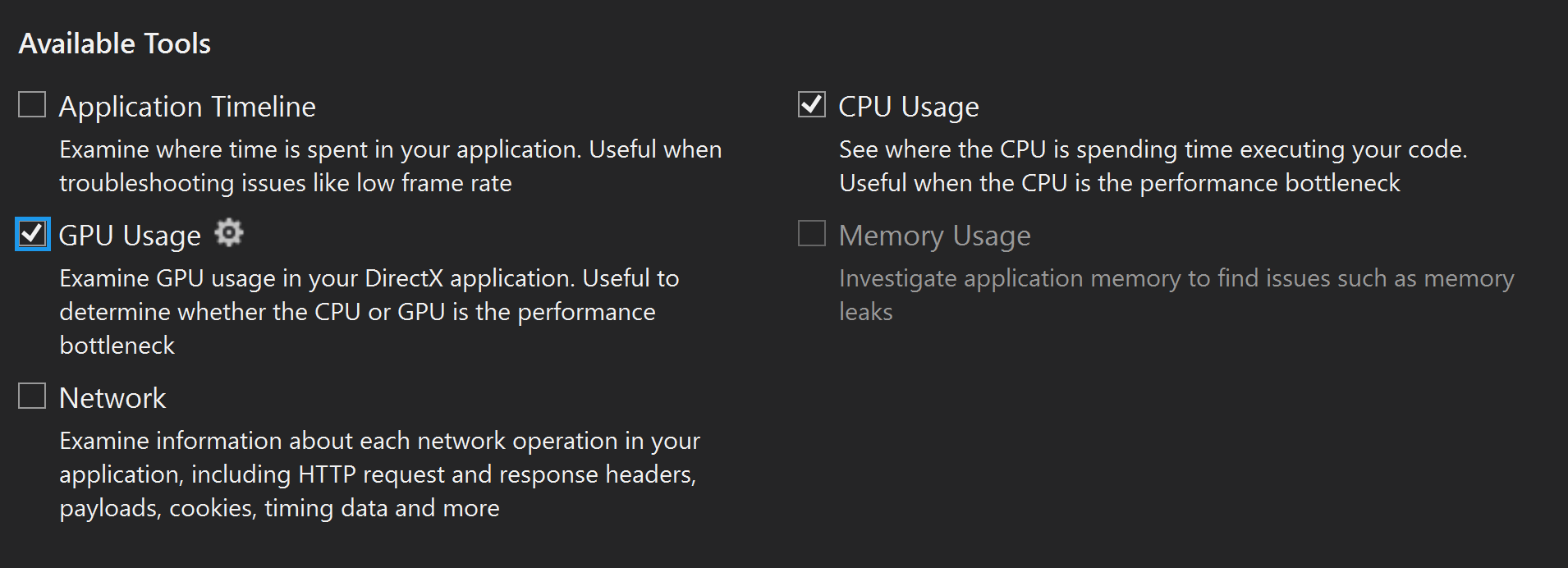
The game is a bit slow and laggy. A partial reason is that we are running under the debugger on a Debug build, but let’s see what happens behind the scenes.

Stop the game.

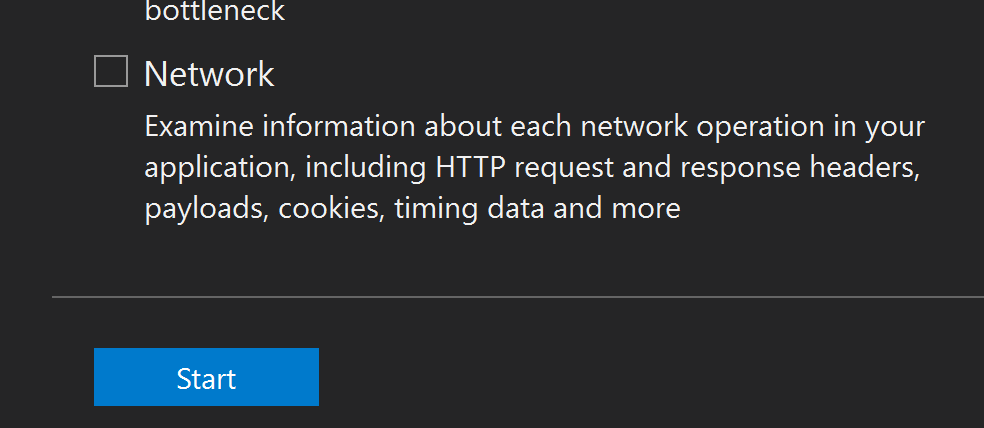
**Step 4: Running the Profiler**

Click Debug->Start Diagnostics Tools Without Debugging…

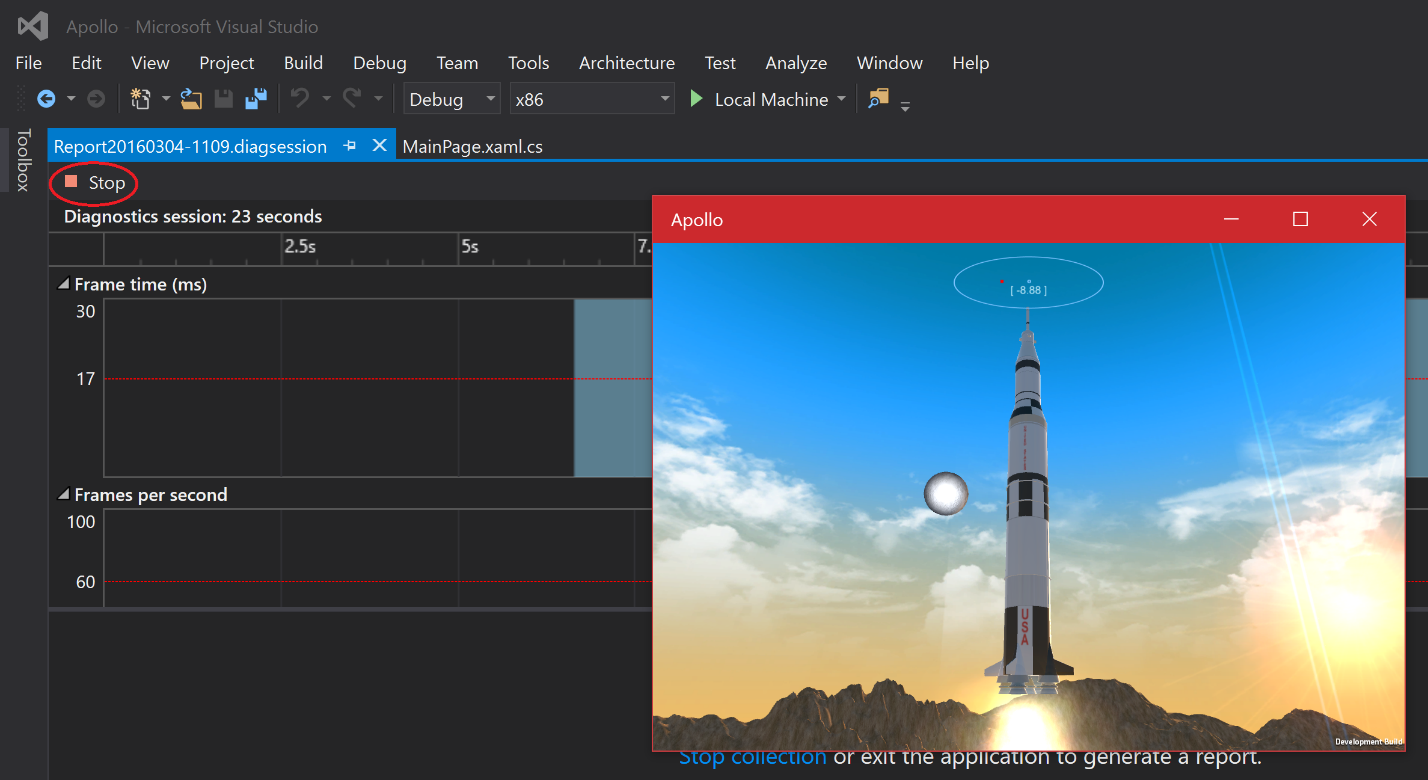


Run the game with CPU Usage and the GPU Usage Profiler enabled. If the game flickers a lot, stop the game and start it with the CPU Usage flag only.

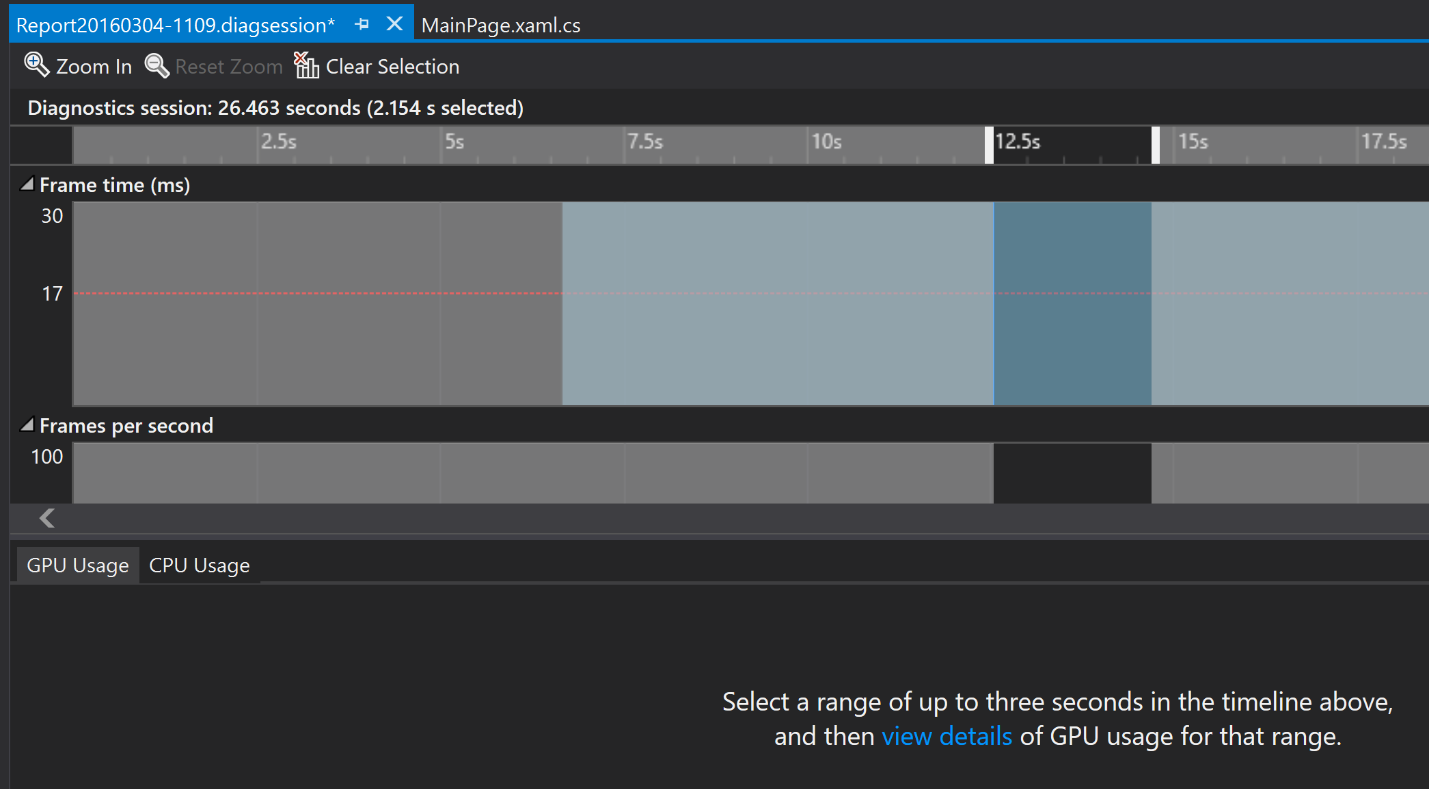
Next, click start to run the game with the profiler:



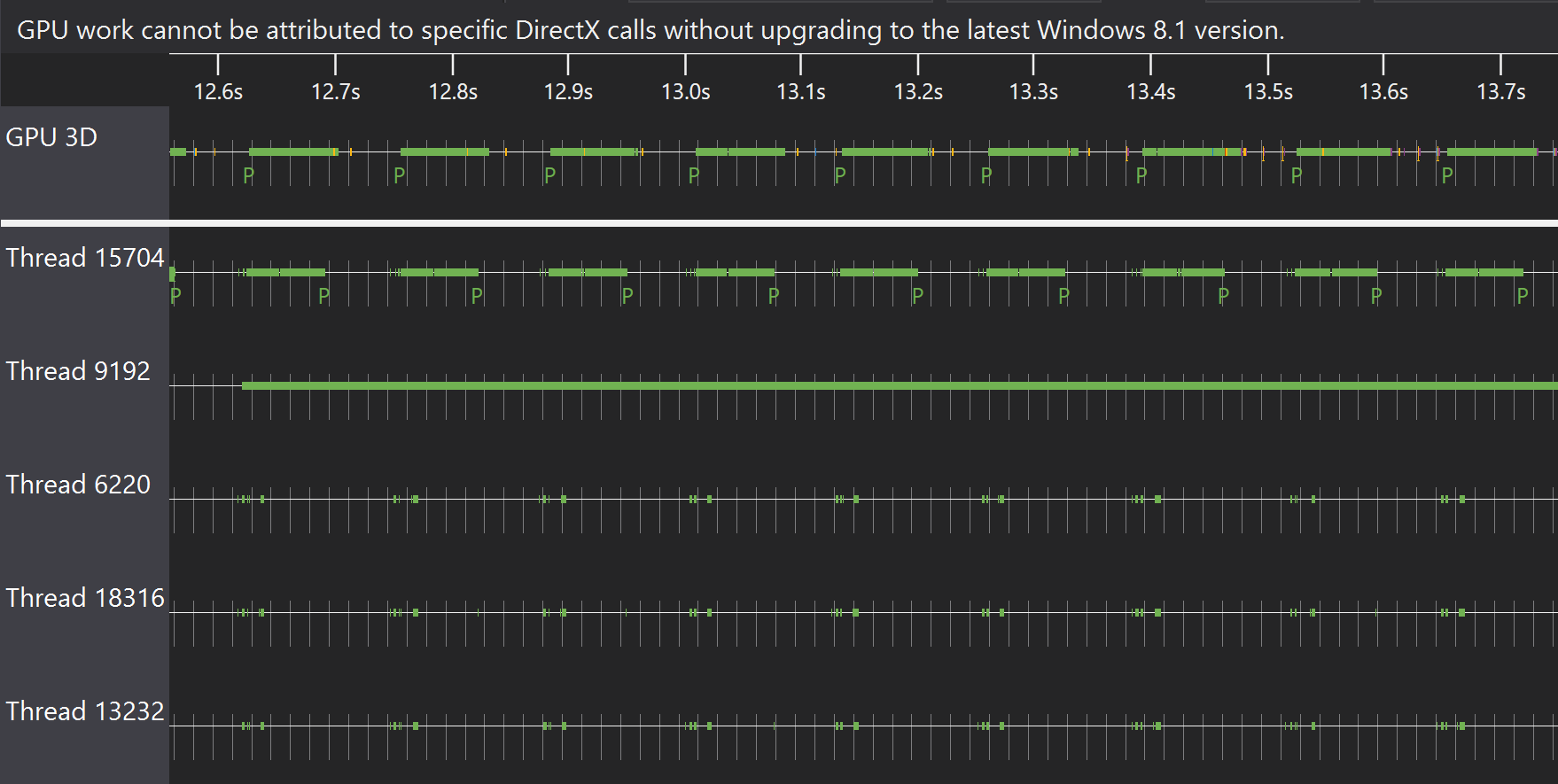
Do a launch and let the game run for a few seconds before clicking the STOP button.



The Graphics Diagnostics tools will gather some information.

**Optional: Use the GPU Profiler**  
If using the GPU Profiler: Once it is done, select up to a range of three seconds by clicking and dragging on the graph:  


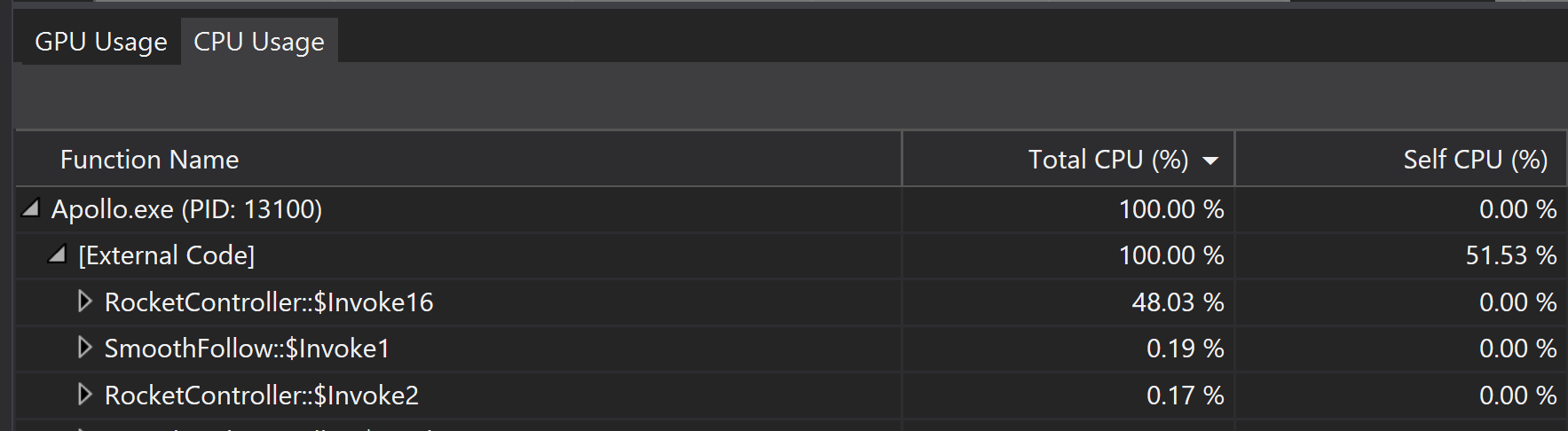
Once you have your range selected, click VIEW DETAILS to see the analysis of the selected interval.



See in the GPU profiler that it’s the CPU that’s having the load, GPU got nice chunks, indicating a drawcall.

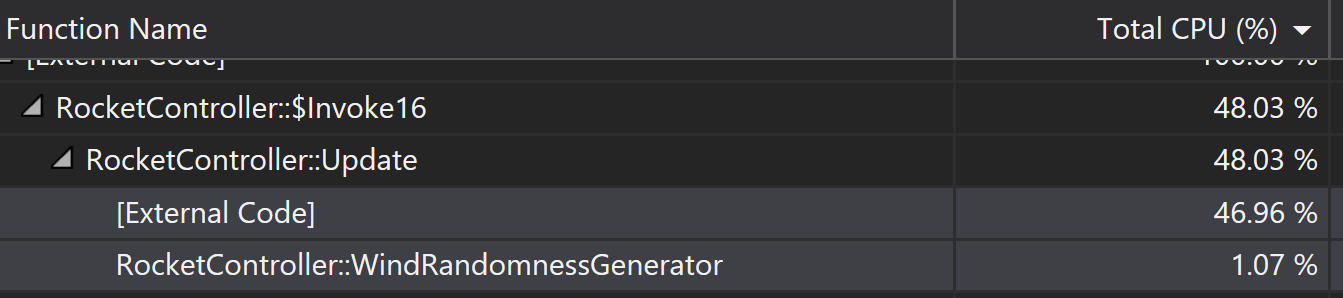
**Step 5: Checking the CPU Profiler**

Go to the CPU Usage tab by clicking the CPU Usage tab below the graphs.



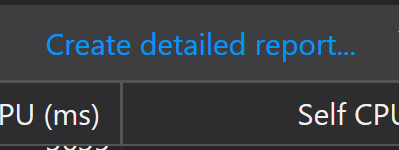
Check the CPU usage and notice that the RocketControllers Update function is using a lot of CPU usage, in my case 48.03%.

Open the RocketController::$Invoke16 tree to dive a bit deeper.

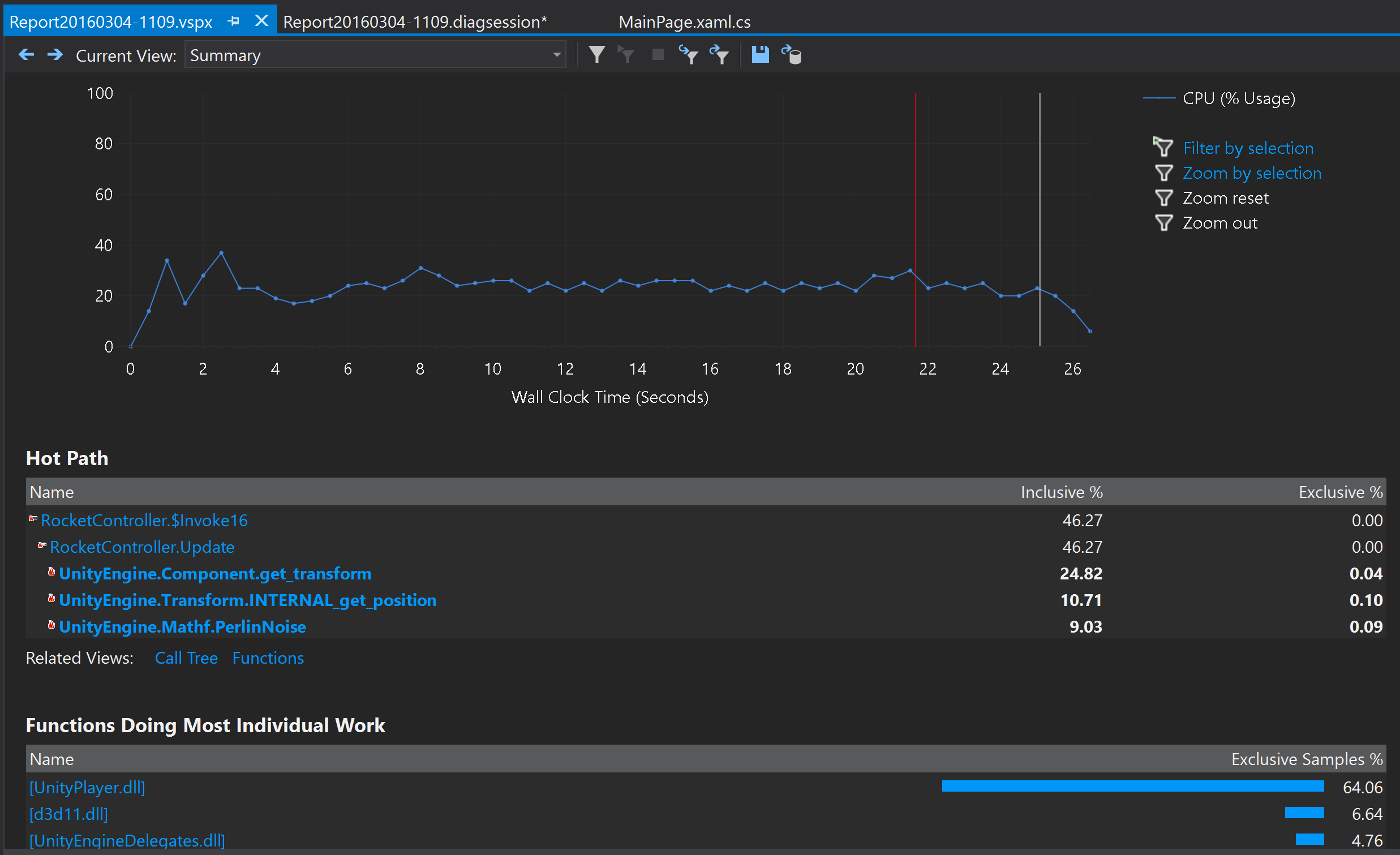


Going into it, we can see that it’s external code that’s spending most of it, and that external code might not have any debugging information. Also, notice the WindRandomnesGenerator function call here. It’s quite low on the CPU Usage, but I know it’s calling external code so this might be the source of error.

Let’s do some more digging by clicking the “Create detailed report…”:

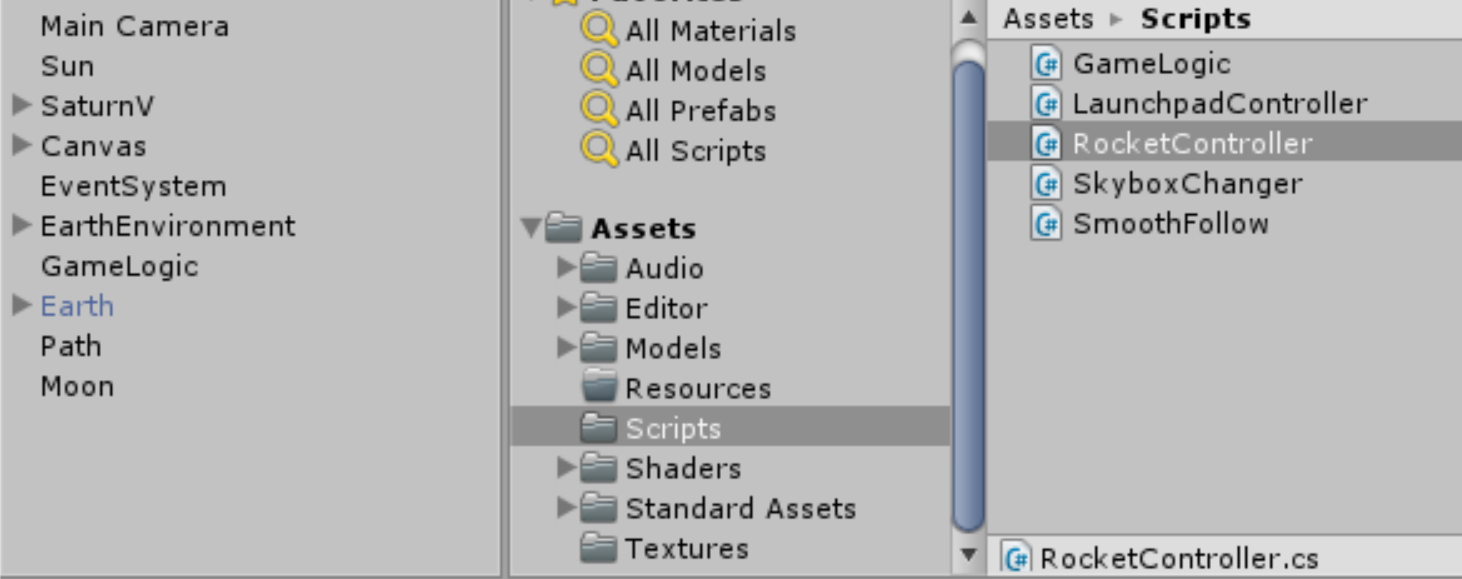


This will give some some more details around what we are doing, and you will see a similar screen to this:

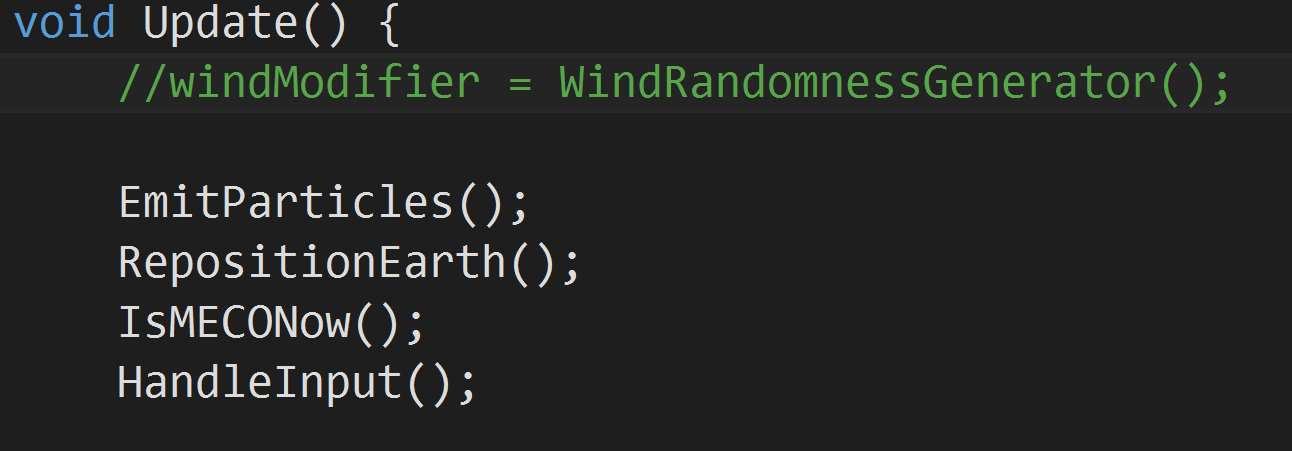


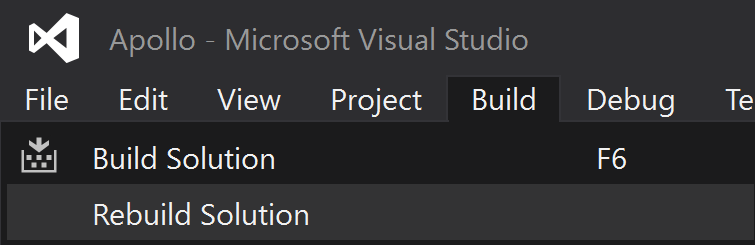
There is a lot of details on this screen, but for now, notice the Hot Path tree, and a call to Mathf.PerlinNoise. In my wind generator, I’m using multiple calls to the PerlinNoise function to generate a pseudo-random wind pattern using various octaves of perlin noise added together.

**Step 6: Checking the function**

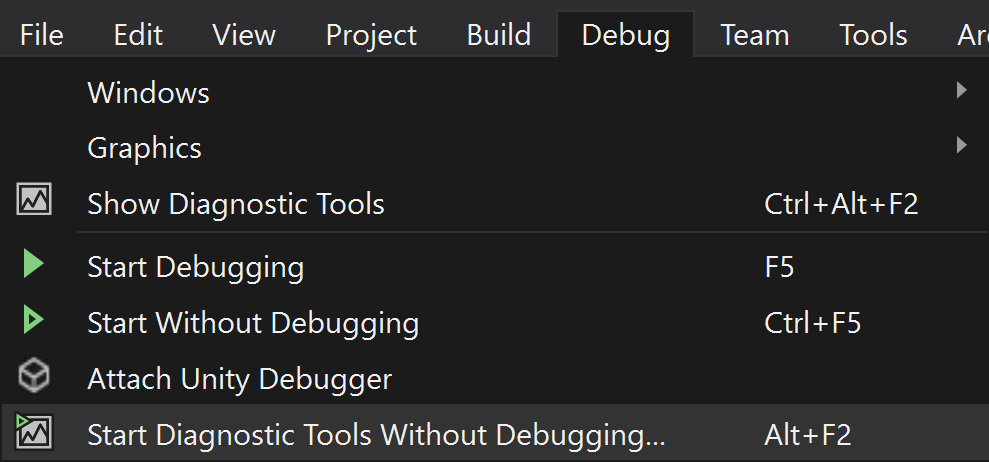
Going back to Unity, open the script for our RocketController:  
 By looking at the various functions in this short scripw, can see that most of the functions in the Update loops are pretty straight forward except for the WindRandomnessGenerator() function we identified above.

To ensure we are looking at the right performance issue, let’s first try to comment out the line of code that is generating the wind. Go to the Update() function in the RocketController script, and comment the first line of code:

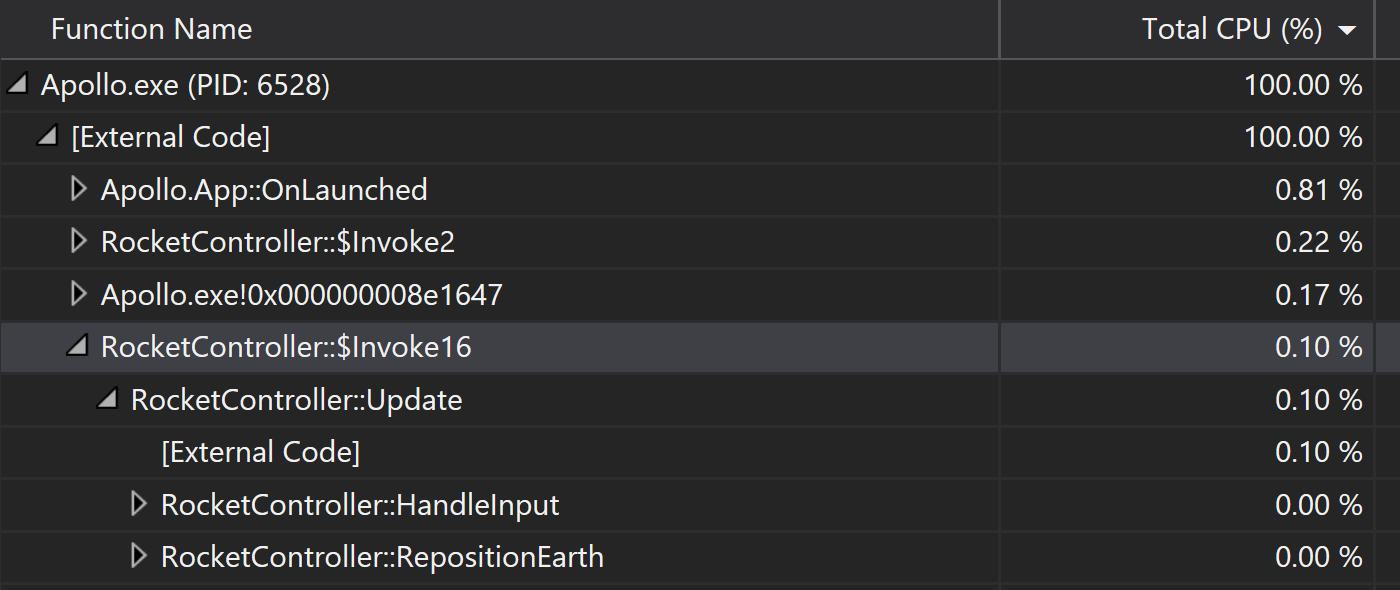


Now, save the change and do the export again from Unity. Once done, do a Rebuild all:  


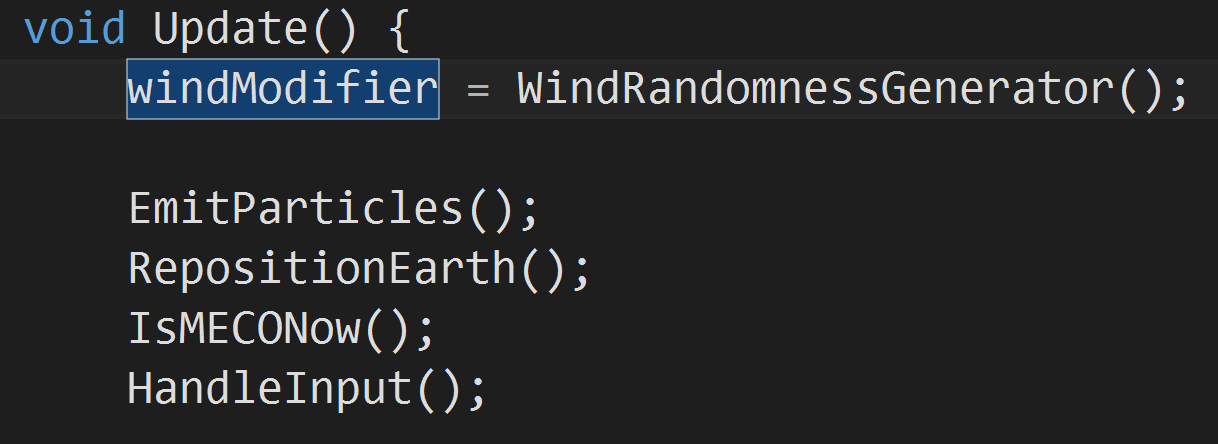
Run the game under CPU profiler like we did earlier:



Go the the CPU Usage tab and notice the changed values. The Update function is now at normal levels.



**Step 7: Fixing it**

Uncomment the call to our wind generator:  


It’s calculating way too many octaves on the Perlin Noise function, one per increase in view while in my intention, I just wanted to have a few octaves with a good range between each octave.

This must be changed. Change the jump in octaves to 20000, significantly reducing the number of calls.

The code should look like this:

float WindRandomnessGenerator()

{

float windRandomness = 0.0f;

int octaves = 0;

int lastOctaveDistance = 100000;

int increaseDistancePerOctave = 20000;

for (int i = 0; i < lastOctaveDistance; i += increaseDistancePerOctave)

{

octaves++;

windRandomness += Mathf.PerlinNoise(

transform.position.x / (lastOctaveDistance - i) + 1.0f,

transform.position.y / (lastOctaveDistance - i) + 1.0f);

}

return windRandomness / octaves;

}

Export the game again, and run it under the CPU profile like in Step 6.

The issue is now fixed, and we now got a better framerate!

However, we have more graphics issues to fix so let’s move on to the next challenge!