DM1595 - ProgUtv: Unity!

Work diary

1. C# observations:

While Python and JavaScript are dynamically typed and tend toward more flexible, easy to read, scripting-style coding, C# is statically typed and compiled ahead of execution. Yet C# shares familiar syntax features (braces {}, if/else, loops) with JavaScript and Java.

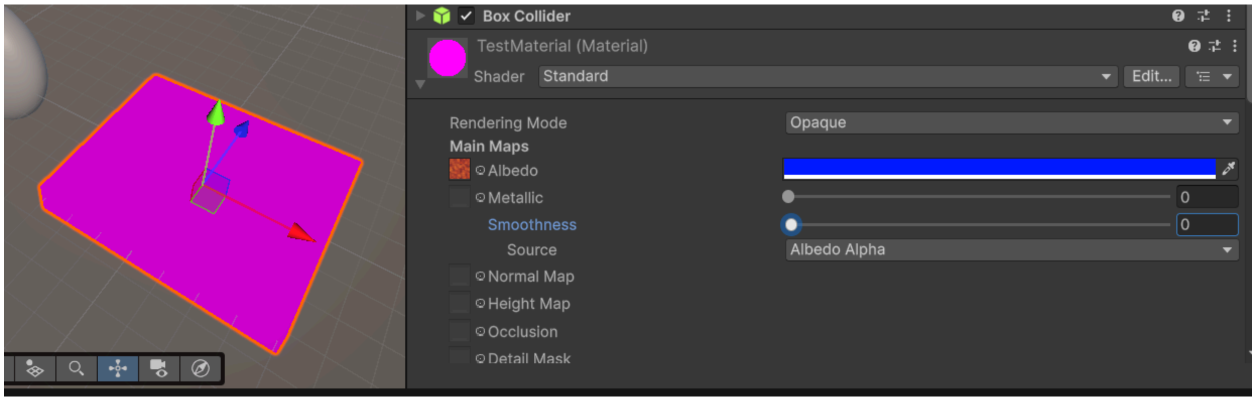
All programming languages I've come in contact with use similar control and data structures, but C# enforces stricter typing and integrates deeply with the .NET ecosystem and its class library. This means C# doesn't stand alone, .NET provides cross-platform support, interoperability with other .NET languages, and frameworks for web, desktop, and, of course, game development. Looks like a fun language!

2. Now, on to the work diary itself!

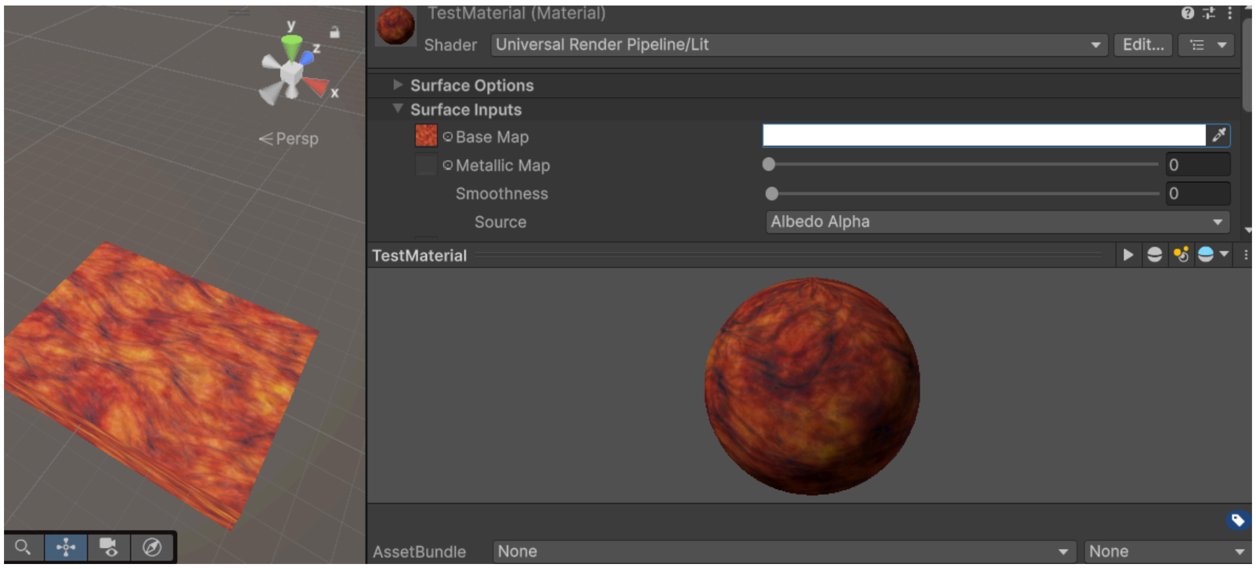
I have previously worked in Unreal engine, having taken a summer course in game development and 3D modeling. I hope my (brief) experience can be useful to create something fun :))

Module 1:

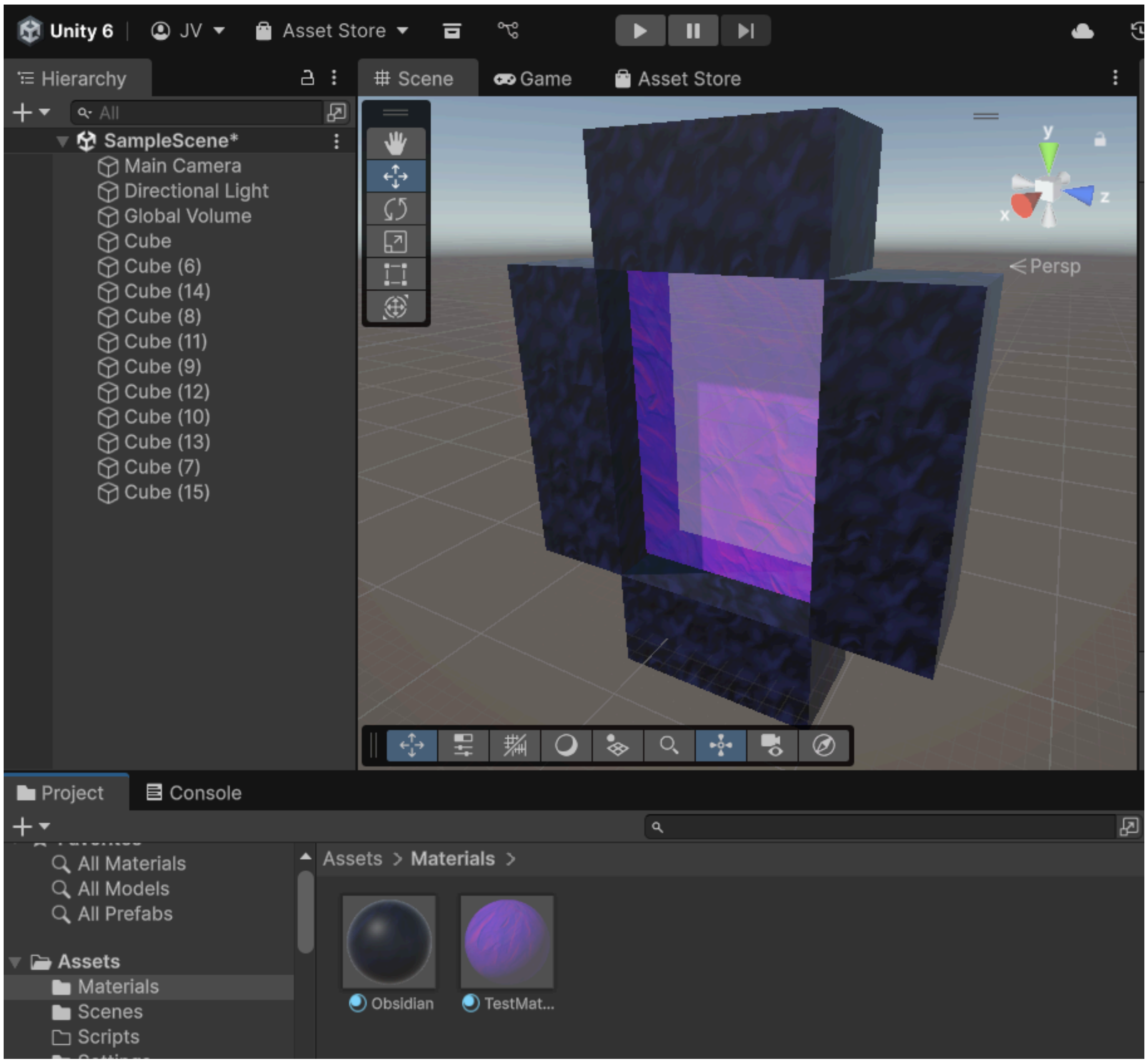
- "Push the play-button [...] move the camera around while in-game (how? 🤔)" Great question, I don't seem to manage to move it with WASD/arrow keys or the mouse...
    - Snooping around Unity forums, I found another noob with the same question as me, and a wise user by the name of *Tomo1098*, who replied: "*The scene camera can be moved around in the Editor; the game camera is entirely dependent on your game code.*" We won't be able to move the camera until we program it to move!
- The inspector window gives me (loads of) information for the selected material, and provides a way to change the values. This window lets me view/change things like the shader that is being used, as well as copy properties to other materials, create presets, etc...
- I got stuck on 'Modifying materials', step 4. I was able to change the color of the material after applying it to my GameObject, but I didn't find anything called 'Albedo'... I soon realized that the reference picture had a clue for me, the selected shader in the picture is 'standard' and the one I got by default is called 'universal render pipeline/lit'. Changing this to 'standard' gave me an Albedo text thingy, but changed the whole GameObject to a bright pink color, although the color I previously selected was still there.


Box Collider
TestMaterial (Material)
Shader Standard
Rendering Mode Opaque
Main Maps
Albedo
Metallic 0
Smoothness 0
Source Albedo Alpha
Normal Map
Height Map
Occlusion
Detail Mask

This was very weird, and 'the regular fix' (Edit>Rendering>Render Pipeline Converter to convert the material to URP) didn't work for me so I tried loads of different stuff around until I found a way to display my texture properly, although with the universal render pipeline, standard refused to work... Hopefully that will not be a problem moving on!


TestMaterial (Material)
Shader Universal Render Pipeline/Lit
Surface Options
Surface Inputs
Base Map
Metallic Map 0
Smoothness 0
Source Albedo Alpha
TestMaterial
AssetBundle None None

- It turns out I got something good out of my previous experimentation; I figured out how to tweak colors and transparency in materials, which led to me creating a (pretty accurate, in my opinion) nether portal for my tediousObjectScene!



In true Minecraft fashion, this masterpiece was built exclusively using cubes 🙂

For the ball game: I chose to create a Universal Render Pipeline (URP) scene, (as the document didn't specify which one to choose) and test if my materials act the same as before.

- Step 9: Moving the camera made me realize I can tweak camera position values in the inspector while play is pressed and the camera will move!! I feel so much smarter than when I started writing this document…

- I understand that making the ball a rigidBody will make it collide and act as a rigid body, but I'm left wondering why it can stay on the plane and roll off the plane when tilting the plane, if we haven't made the plane a rigidBody???????????

- The provided sphere control script didn’t work for me out-of-the-box, asking GitHub Copilot for help resulted in me know why that was the case (the provided code used the old way of controlling the player movement). In it’s own words: “your Unity project is using the new Input System package, but your code is still using the old UnityEngine.Input class” I asked it to update the code and after a few iterations (I’m not the brightest prompt ‘engineer’) I managed to make it work! It moves the sphere very slowly, but it works for now. I assume (haven’t tried) tweaking the force factor variable will change it, we’ll see if I feel like doing that later on.

- I felt like changing the force value pretty quickly! Testing the pickup script was waaay too tedious with a ball moving at 0.5 mm an hour. I managed to make it all work and discovered that changing the force value is NOT what makes the ball go faster, it’s changing the movement input increment! Fun insight :)) (ps. I made it possible to roll off the map, to add a little risk vs. reward type of situation)

- Regarding Instantiate: it is a Unity method that creates a new copy of a GameObject at runtime. It's like making a photocopy of an object - you end up with multiple instances of the same thing. If we were to use Instantiate in this case, every time the player touches the pickup, we’d create a NEW pickup object, the old pickup would still exist (unless explicitly destroyed), so after say, 100 pickups, there would be 100+ pickup objects in the scene - leading to memory bloat, performance issues, and all that bad stuff.

- Fun lab overall! It took A WHILE for me (not to toot my own horn, but as a somewhat experienced person I thought it would take a couple of hours at most) - I believe most students will have a hard time turning in this lab in time?? Would like to know the statistics week to week for this course :P