**Speedometer and Rolling Road Game Development**

So this is about the two games I worked on – the Speedometer and the Rolling Road – and I'm gonna explain everything I did, how I built them, and the stuff I got stuck on. I’m saying it the way I’d normally talk and write, how I actually think, how I type, and the sort of stuff I’d naturally focus on. I’ll go through what I learned, what tripped me up, and how I got it all sorted in the end. I’ll also talk about the little things that went wrong that might not seem like a big deal, but were actually the bits that made the process frustrating or just slowed me down unnecessarily.

**Speedometer Game**

So the first game was the speedometer one. It was meant to show the speed of a moving car in real time on the screen. I already knew that in Unity, to get any kind of physics-based movement, the Rigidbody component was involved. So I made sure the car had a Rigidbody. But here’s where the problem started: I couldn’t actually *see* any velocity field in the Unity Inspector. When you look at Rigidbody in Unity, all it shows you is mass, drag, angular drag, and angular velocity. But not velocity. So I thought maybe it wasn’t something you could use, or that it was hidden or maybe only appeared if the object was moving or something like that. That threw me off.

I looked into the scripting API and saw angularVelocity, but still not just “velocity.” So I didn’t even know it was possible at first. I said in our chat, like, how are you even supposed to know that .velocity exists if Unity doesn’t show it anywhere obvious? I wasn’t gonna just randomly guess it. It’s not written clearly, especially for beginners or anyone still getting comfortable with Rigidbody.

Eventually I learned that you *can* access rigidbody.velocity from code even if it’s not shown in the Inspector. You just have to know it’s there. Once I understood that, I used .magnitude to get the actual speed value, since velocity is a Vector3 and includes direction as well. magnitude gives you the length of that vector — basically how fast it's going, without worrying about which direction. So I used:

csharp

CopyEdit

float speedInMps = carRigidbody.velocity.magnitude;

float speedInKph = speedInMps \* 3.6f;

speedText.text = speedInKph.ToString("F1") + " km/h";

That line takes the Rigidbody’s velocity, gets the magnitude (speed), multiplies by 3.6 to convert from meters per second to kilometers per hour, then updates the UI. Simple once you know how, but I needed help figuring that out at first.

I also added a speedometer needle that rotated depending on how fast the car was going. I had to figure out how to use angles properly and clamp the needle’s rotation to match the max speed. I used Mathf.Lerp so that the rotation felt smooth and didn’t just snap to the new angle instantly.

**Rolling Road Game**

This one was different. The car didn’t actually move forward – it just sat in one place while the road moved backwards, which created the illusion that the car was driving. It was a different kind of setup. There was no Rigidbody needed for the car, which made things easier and cleaner, especially for controlling the visuals.

I used:

csharp

CopyEdit

transform.Translate(Vector3.back \* scrollSpeed \* Time.deltaTime);

to make each road piece move backwards. When a piece moved past a certain Z value, I teleported it back to the front to create a seamless loop. That way the road just keeps scrolling forever.

The part I struggled with was getting the timing and spacing of the road pieces right. Sometimes one would move too far before resetting and there’d be a visible gap, or the next piece would pop in too early and overlap. Took a few tweaks to line them up properly. Once that was sorted, the visual loop looked clean.

Later I added some obstacles like cones and tested out collisions using colliders. Since the car wasn’t moving, I didn’t use physics for it. The road and cones did all the moving, so the illusion still worked. Crashing into a cone was just a matter of detecting that collision.

**Stuff I Struggled With**

Easily the hardest part for me was the .velocity thing. It wasn’t about the code being difficult — it was about knowing it existed at all. Like I said earlier, Unity doesn’t show velocity in the Inspector, and the API doesn’t make it super obvious either. So I had no clue it was even a thing. If I didn’t ask for help in our chat, I probably wouldn’t have figured it out in time.

Another thing that really got on my nerves was the file upload situation. I built the project and zipped the files like I normally would, but when I went to upload it, the system rejected my .zip file. It only accepted stuff like .docx, .pdf, .txt, and that kind of thing. I tried uploading to OneDrive but that didn’t work either. It just wouldn’t accept the file.

So then I had to use Google Drive. I uploaded the zipped Unity projects there, copied the links, and pasted them into a Word document. That was my workaround. But even then, I had trouble saving the Word doc. There was no “Save As” button showing up, and when I did save it, I couldn’t find where the file had gone. Took me a few tries and help from our chat to finally figure it out and save the Word doc in the right place so I could submit it properly.

I also had that moment of doubt with .magnitude — I didn’t know at first if it was part of Vector3 or part of Rigidbody. That’s another thing that’s not super obvious unless you already know how vectors work in Unity.

**What I Learned**

* .velocity is a property of Rigidbody and gives you a Vector3. Even if it doesn’t show in the Inspector, you can still use it in code.
* .magnitude on a Vector3 gives the total speed, ignoring direction.
* Multiplying by 3.6f converts from meters per second to km/h.
* You need to use Time.deltaTime for consistent movement across frame rates.
* Mathf.Lerp is good for making movement or rotations smooth.
* Repeating environments like the road can be done by repositioning objects instead of instantiating new ones.
* If Unity builds can’t be uploaded as .zip files, use Google Drive, then paste the link into a Word doc and upload that instead.

**What I’d Do Next**

If I had more time or needed to improve the project, I’d probably add more interactivity to both games. For the speedometer one, I could make it more realistic with a rev counter, fuel gauge, maybe even engine sounds or gear changes. For the rolling road, adding obstacles that move, cars that you can overtake, or a score system would make it feel more like a proper game.

I’d also want to learn more about how Unity handles physics behind the scenes. Stuff like how .velocity and .angularVelocity are calculated, when you need to use FixedUpdate versus Update, and how to debug invisible properties more easily.

**Final Thoughts**

Both games worked in the end, but getting there took more effort than I expected, not because the core mechanics were too complex, but because there were just gaps in the info. Stuff like missing .velocity from the Inspector or weird assignment submission rules are the types of things that can catch you out even when you’re doing everything else right. Now I know what to look out for, I feel more confident doing similar projects in the future.