Game Development Journey Decisions

Character Controller vs Rigid Body:

Character Controller method isn’t responsive and will not take into account gravity so I will have to develop a completely new gravity system. One benefit to this method is much greater freedom with how our object behaves which is also the greatest downfall since you have to code every single part of that yourself and is unnecessary.

Rigid Body component doesn’t require a character controller. The one notable difference is the way physics operates with rigid bodies as it allows us to use a function called Fixed Update which I researched at the Unity API scripting Documentation. This function allows us to perform any physis simulating functions to be synced at the same interval. Furthermore, if more advanced physics dependent features need to be coded for the user then it is much more efficient to use the Rigid Body.

Debate on Character Controller vs Rigid Body - <https://medium.com/ironequal/unity-character-controller-vs-rigidbody-a1e243591483>

Update Function Unity API Scripting - <https://docs.unity3d.com/ScriptReference/MonoBehaviour.FixedUpdate.html>

Humble Object Pattern and Unit Testing:

Separate the logic used in MonoBehaviour framework. These MonoBehavious classes are sometimes too hard to instantiate since they are so coupled to their environment. We should use Monobehaviours as object just that are delegators for unity system calls to manage the logic.

This way we can unit test the actual logic of the program as we proceed

Humble Objects Video by Infallible Code - <https://www.youtube.com/watch?v=OecJvh8Zvc4>

Interfaces:

Start using interfaces where each interactable objects can implement and then the all the player class update method has to do is look for that specific object. This will avoid the code smell of classes are way too large which allows are code to be more scalable and easier to maintain. Now any class implementing this interface can be searched for without having to add a line of code for every single object.

Mathf.lerp:

Currently my animation in the game for when the camera changes isn’t smooth and kind of gives a sense of teleporting to the new camera point. I researched the unity scripting manual and some unity forums and discovered a very easy work around for this which is a function called Mathf.lerp which interpolates between the 2 transition points at a user given speed. Using this will be essential throughout development especially for unity built animations.

Mathf.Lerp function Unity API Scripting - <https://docs.unity3d.com/ScriptReference/Mathf.Lerp.html>

Photon vs Unity networking – network topology

Photon is more speed efficient –

For 2 clients to communicate with each other on a server in unity, a user message must always go back through the host client and can’t be directly sent through between clients.

If Client b wants to communicate with client c for example: Client B ▸ Relay Server ▸ Host A ▸ Relay Server ▸ Client C.

This takes 4 bounces to communicate this message. This is one downside to the client/architecture however this topology has many benefits as well. Photon however supports this same architecture as well as a peer-to-peer architecture as well.

If Client b wants to communicate with client c for example: Client B ▸ Relay Server ▸ Client C.

This has a total of 2 bounces vs 4 since the 2 clients can directly communicate with each other. In some cases, entirely bypassing the relay server as well and make the communication 1 bounce.

Greater documentation and tutorials:

Photon has also got great documentation and in depth tutorials to go with it.

https://www.raywenderlich.com/1142814-introduction-to-multiplayer-games-with-unity-and-photon

Multiplayer Architecture layout:

At the moment we have a player controller class which controls behaviour of the player user. Were going to also need a player manager class which will manage player variables like health etc and communicate this with other networked players.

**Room Manager**

(instantiates)

**Player Manager**

(instantiates)

**Player Controller**