Hello and welcome to my project, I am Coleman Caldwell and in this project I will be comparing the concepts we learned in PPL to the Unity game library which is built upon the C# framework. Unity is one of the most popular game engines in the world, used for animation and modeling, but the main thing is 2d and 3d games. Unity has a unique set of library features that ties very well to this class and I think there's a lot of crossover, even disregarding the standard C# library.

The topics i'll be covering in this presentation are on mutable state using a finite state machine and player health, higher order methods / callbacks both from the unity library and from C#, data types concrete and abstract also custom made, parsing, lazy evaluation, and evaluation of order from the built in functions.

A perfect example of mutable state in the context of unity can be demonstrated by what's known as a finite state machine. A finite state defines the state of an object (item, player, enemy) to whatever state they are currently in. The core concept is that this state will update depending on whatever is happening to the object, or in this case whether or not a player is moving or idle. In this script, I have a movement script for the player and a state idle or moving, and the current state which I will display. Every millisecond in this function, the state gets updated to the context of the player.

We see that the current state of the player is idle, but as soon as i start moving around it is moving, and when I stop it goes back to idle. This can be applied to any object in the game world.

A more universally known example of a mutable state in the context of unity is player health. Health is an entirely mutable state and is always changed within the context of the game. I have two basic functions that either take away or add to the player's health.

To give a basic example of how higher order methods are used in Unity, and through the C# language, I have a very simple function MoveCube which takes a Unity System.Action<GameObject> move. Takes a function as a parameter, and we just call that function inside the movecube function. In update, we call MoveCube with cube as the parameter, and the callback is updating the transform.position of the cube in the game space.

We see that in real time, the cube is actively getting updated and it is moving among the x-axis.

To iterate on this, we can also do this with multiple cubes and essentially use a map, or foreach in this case, to do the same thing with multiple objects. So we're doing the same logic but on multiple cubes, using a forEach to iterate through the list. We can see the list because we're in the developer space, over here we have our list of cubes and when I run this it will apply the transformation to every cube in our list. And you can see every cube gets transformed.

Another example of this with properties that are not user-accessible at all and with a little error checking is changing the color of the cubes upon runtime. And you can see through the code, the color of the cubes changed to magenta.

Talking about data types, another benefit of the Unity engine library rather than just the C# library is that it has a vast array of unique data types relative to the game world, whether that be game objects, colliders, masks, colors, transforms, materials, all to utilize with the game objects in the game space. But another thing that the C# library allows us to is to make our own custom abstract data types. So in this case, I created my own custom IIterator, which is an interface, but we can make it have any properties we want it to. Similar to a class.

Unity uses parsers for almost every aspect of it's engine, but the most overlooked part of a game engine in general is that Unity uses a JsonUtility parser to parse player data. Player data is stored in csv files. In this example here, the built in Json Utility is what we're using to parse player name and scores from the player file. Another example of this in the game space would be a parser to parse commands in an in-game user computer. The nice thing about Unity is you really have free reign in your game to do what you want with this. You could parse objects, object values, it all depends on the context of what you're making.

Unity also has built-in parsing of their game objects for example , or their materials, so you can find a game object by type or some other property. There is more than csv, json, xml, html etc.

Another important thing to note about the unity engine and how the scripting engine works, is you can very rarely have a free variable. Unity always compiles the scripts before you start the game. Even though C# supports this, Unity does not allow for undefined variables, you can instantiate new ones, but you can't have existing, undefined typed variables and assign them a type.

A good example of Lazy Evaluation in unity is the Unity library's Coroutine, in this case StartCoroutine. Essentially, coroutines allow for multithreading or awaiting of the same functions without interrupting the gamespace. So you can pause functions or await for a certain function to happen, without interrupting the gamespace. So in this example, I have a basic fibonacci sequence with a coroutine is waiting one second tom yield every new result. So we can see here that it's waiting one second for every new result of the fib sequence. Comparable to javascript await/async, can wait for one process to finish or multi thread.

Unity has a built-in, predetermined evaluation order for their engine to deal with physics, input events, animations, and other states and game logic that may be important to the game. But in a specific order to handle events as they should. Rendering, input logic, game logic, physics logic all in these games.

Evaluation order is also predetermined by these functions that you've been seeing in a lot of my other code. Awake, Start, Update, and FixedUpdate. Awake and Start are upon the initialization of the editor and the game world, Update is called once per frame, FixedUpdate is called once every 10ms. The general order of the events that unity handles the evaluation of is the initialization of the game, physics, input, animation or animation state, and the game logic.

And if we look in the editor and I collapse, normal update is getting called exponentially more than fixed update, awake gets called once, start gets called once, but you can see the disparity in the evaluation order and why this gets called in-order.

That was my project on Unity and how it relates to the core concepts we learned in PPL. I hope you enjoyed