This project parses the course material. In it, we use data oriented design through unity’s entity system. Through which we employ the Component-Based Architecture, dividing data, baking and logic into three distinct components. Structs are more performant for data (on stack, not heap) logic is more efficient due to jobs and the opportunity to employ the burst compiler which translates directly into machine code.

Since components only store data it can utilize multiple pointers to a single instance of that data. Allowing for efficient instancing of entities as opposed to game objects.

**C# Jobs:** Allows us to utilize every thread available to the CPU by delegating logic through it. This is opposed to unity’s default default main thread which can’t utilize the computational power available.

PlayerMoveSystem.cs uses a job to handle movement logic with burst compile (raw machine code).

The LifeTimeManagementSystem uses a job to decrement a timer float, once float reaches 0 it adds the IsDestroying tag to the entity, which is then used for query in another system to locate said entity and destroy it.

[Note, the destroying management system should be in its own script]

**Parsing Entities:** the query maintains a sorted list of entity components which allow for fast and efficient algorithms, for example:

**foreach ((IsDestroying tag, Entity entity) in SystemAPI.Query<IsDestroying>().WithEntityAccess())**

**{**

**ecb.DestroyEntity(entity);**

**}**

**state.Dependency.Complete();**

**ecb.Playback(state.EntityManager);**

**ecb.Dispose();**

**}**

In LifeTimeManagementSystem, using Query of SystemAPI to find all entities with the IsDestroying component; making it more performant than say gameobject.find, findbytag etc.

The code is what we did in class, as such I’ve tried explaining why it’s more performant.

I wanted to add the LifeTime component to the enemies and use the counter to determine the next wave but y’know how it goes…

The fireprojectile System is similar to the lifetime one, in that it parses for the tag and fires bullets should it be found.

**The process:** We created the PlayerAuthoring which holds several key components, such as velocities (player and bullet), prefab and destruction/identifier tags, these components hold the data we’re going to use/modify in systems. We then moved onto the PlayerInputSystem where we presume a single player and access it using a singleton to read the value of our inputs to set the respective component to the appropriate value.

In the PlayerMoveSystem we created our first job which is responsible for adjusting our position based on our input value.

In summary the PlayerAuthoring translates our gameobject (player) into an entity, as well as holding numerous components. The PlayerInput system adjusts the value of the input component and the PlayerMoveSystem takes that into actual logic for moving the player.

The bullet behaves in much the same way although with a constant Vector for direction. Through queries for the tag fireprojectiletag to determine when and where to shoot from, this tag was defined on the player and is adjusted in the PlayerInput system to make the keypress actually fire.

The spawner holds one massive component with tons of data, and an authoring script to set said data. In its system file, it uses time and an enemy counter to flip a Wave Bool to instance 10 enemies before a 5 second delay. I had a weird bug here where the enemycounter would have 1 left over, not sure why.

This is getting quite long but I hope I’ve illustrated my knowledge in DOTS. Thanks for reading!