Lab 1: Introduction to Unity

CS410/510: VR/AR Development

Ehsan Aryafar earyafar@pdx.edu

Sam Shippey sshippey@pdx.edu

High level goals for today

- Why do you care about Unity (despite frustrations)
  - Hands-on experience
- How does Unity work internally
  - Under the hood of game development
- How do you use Unity's editor
  - Creating a project
  - Putting things in the game world
  - Connecting to physics engine
  - Development loop
  - Where to get help outside of class
  - Setting up for Lab 2
    - Scripting, VR support, exporting projects

VR, AR, and games

- VR and AR are about more than games…
- … But gaming is the easiest way to get your feet wet with VR and AR topics.
- To that end, we need some way to make games.

Making Games

- What if I asked you to make a game
- Many questions to ask
  - What graphics should I used? 2D / 3D?
  - For what platform?
  - Is your game going to have multiple players (social element)?
  - Do you write native code and use OpenGL, or do you use a third-party game engine that does cross-platform development?

What is Unity?

- All-encompassing game engine
- Wide variety of deployment options
  - Supports 25 platforms and some of the best ones to monetize on: iOS, Android, Nintendo Switch, Steam, VR/AR etc.
- Modular piece of software capable of many tasks in “game” development and easy to expand.
- Supports coding in Javascript and C#
- Drag and drop interface
- Huge community of developers

Why Do You Care?

- Although VR/AR is more than games, it's easiest to think of them as games for now.
- Hides most of the nasty parts of VR/AR for us and lets us focus on content.

What's a game engine anyway?

- As a game developer you want to write a game not a game engine
  - Some languages have more game engine written in them than games, RUST is one example
- Game engines are pretty diverse, but tend to look similar in practice. They usually include:
  - Putting things on the screen (For some definition of "screen")
    - VR has two screens with fisheye effect
  - Some way to talk about objects (complicated with Unity)
  - Probably includes some way to move them around, probably looks something like physics on Earth (or at least this universe)
  - Asset loading pipeline
- Usually not designed for programmers, but for game designers. Getting used to this will make Unity less painful.
  - More like photoshop than a code editor, bugs are solved by ticking some box

Other Game Engines

- Unreal engine by Epic games
  - Optimized for first person shooter
  - Requires more programming (C++)
- Roblox Studio by Roblox
- Corona SDK
  - 2D game engine for beginners
- Game Maker Studio

Unity Internals

A short tour of Unity overall, before diving into a small demo.

- Entity component system/ECS
- Physics
- Rendering pipeline
- Assets
- Media
- Scripting

Unity Internals: ECS

- Entity, Component, System
- Alternative to OOP
- Still a way of handling state, but can be cleaner and easier to think about with regards to games.
- You can think of this kind of like a latency-optimized database: Systems make queries based on components, which can be implemented much faster.

ECS Attempts to Reduce Misuse of Cache

- In games, you commonly iterate over a set of objects multiple times per second, running methods on them every frame.
  - Your physics system might iterate over all objects that are subject to physics and call Object.Integrate(dt), updating their position, velocity, and acceleration. So traditionally you'd have your big object that contains all of its state, including those needed for physics, and you'd call the integrate function on every object that needs to be updated. In each object's Integrate() method, you access the object's position, velocity, and acceleration member variables.
- When you access position, it's pulled into a cache line along with nearby member variables. Some of those nearby member variables will be useful (velocity and the acceleration), while others will not be.
  - This is a huge waste of the cache and in an age where the performance bottleneck is the time it takes for data to get from main memory to the CPU's memory, it's a big deal.

Unity Internals: ECS

- 3 Parts:
- Entity
  - “Things” in the world
  - Actually totally blank, carries very little to no information
  - This might seem counterintuitive but makes a lot of sense

Unity Internals: ECS

- 3 Parts:
- Entity
  - “Things” in the world
  - Actually totally blank, carries very little to no information
    - These exist to attach components to them
  - In Unity, all of the game objects are component-based. You start with a blank object that has only the default required Transform component, and you add more components to give the object functionality.
  - It turns out it’s mostly not necessary to consider every element of a single object as often as it’s necessary to consider every object with some given element.

Unity Internals: ECS

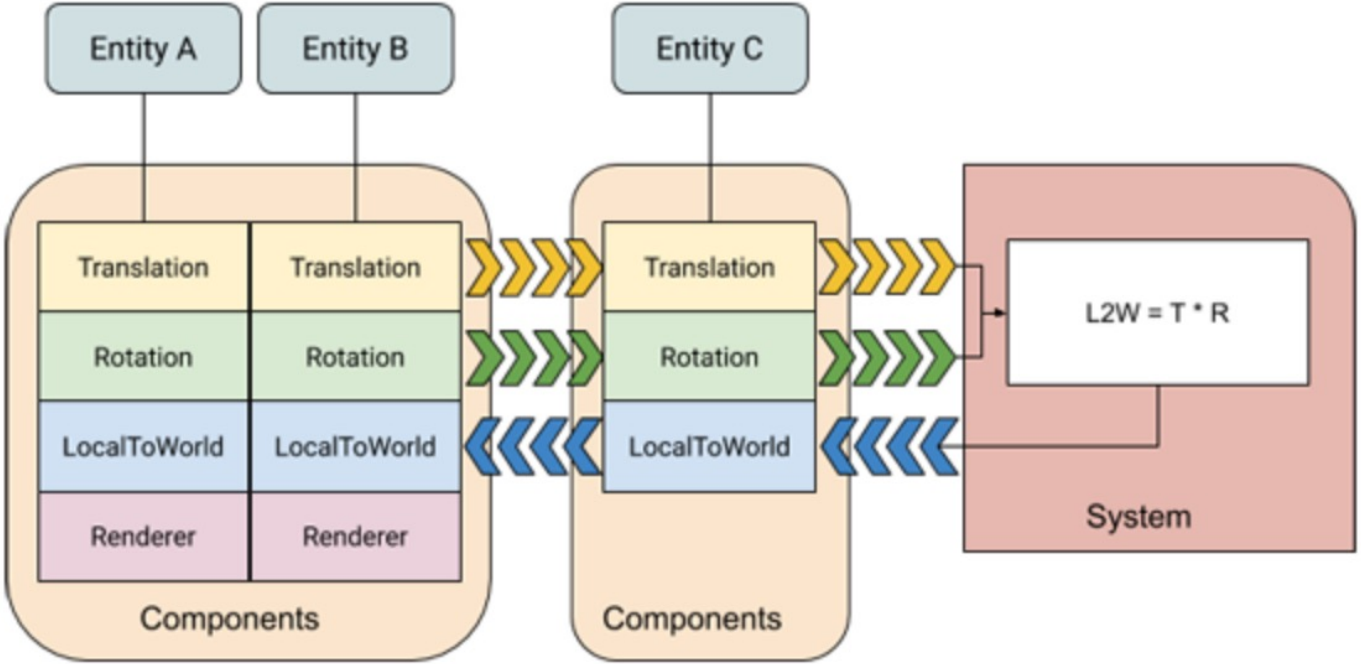
- 3 Parts:
- Entity
- Component
  - “Elements” bolted to our entities.
  - A component is a struct of data

Unity Internals: ECS

- 3 Parts:
- Entity
- Component
- System
  - Analogous to public functions- But aren't tied to any particular object.
  - Actions that take place and mutate state
  - A system can make queries over entities with certain components, which might "return" something like a list of specific components (not entities).

Unity Internals: ECS

- Entity is simply an ID. It does not contain anything. Instead the ID is used as an index into an array of components. An array is contiguous in memory which lends itself well to being the data structure of choice.
- The physics system might have a list of all entities that have a Transform, RigidBody, and Gravity component, and use the entity's ID as an index into the Transform array, into the RigidBody array, and into the Gravity array.



- In this diagram, a system reads Translation and Rotation components, multiplies them and then updates the corresponding LocalToWorld components (L2W = T*R).

Unity Internals: ECS

- Rejoice: You don’t need to actually write most of those.
- Unity handles most of the interesting systems and components.
  - Physics
  - Media
  - Etc

Unity Internals: ECS

- Unity thinks of games as a list of “Scenes”
- Scenes contain trees of GameObjects
  - GameObjects are Unity word for entities
  - They have a few components by default: Transform (position, rotation, scale), tags, and layers/layermasks.
    - Transforms are three dimensional vectors
  - The children’s (of an object) position and rotation transforms are computed with respect to their parent’s. If my position transform is (2,2,2), and my parent’s is (1,1,1), my position will be (3,3,3). The same applies to rotation.

Unity Internals: ECS

- Tags: Exactly what it sounds like. These are strings attached to a GameObject.
  - For example “Player” tags for player controlled GameObjects vs “Enemy” tags
  - Help identify GameObjects for scripting purposes (FindWithTag(str))
- Layer: There are at maximum 32 of them.
  - Define which GameObjects can interact with different features and one another.
    - For example, determine which sound to play when player touches an object.

Unity Internals: ECS

- GameObjects are tagged with Components
  - Components are practically anything
  - The most important for us to think about right now are physics and scripts
  - Transform: Exposed directly as 9 numbers, 3 for each of position, orientation, scale.

Unity Internals: Physics

- We need to change the transforms of GameObjects as a function of time.
- Physics is a really complicated way of making changes to transforms.
- Rigidbodies
  - What the physics engine sees
- Colliders
  - How they interact with each other

Rigidbodies

- A rigidbody is a component that the physics engine interacts with. Add it to a GameObject to tell the physics engine it needs to change this thing.
- Properties
  - Mass: when two objects collide, what gets pushed, what's stationary in collisions.
  - Is Kinematic: If this checkbox is set, the rigidbody won't respond to the physics engine (will be ignored by engine). Under full control of scripting and animation.

Importantly- Kinematics MUST use Collision detection continuous speculative

Rigidbodies

- collisionDetectionMode/Collision Detection: Determines how the physics engine should check for collisions.
- Interpolation: Due to difference between physics and rendering timesteps/frame rate. Enable on main character only.
- Constraints: If these are set, the physics system won't touch these properties of the GameObject's transform.

Colliders

- Colliders
  - Primitive colliders: Sphere, box, capsule
    - Try to use these as much as possible over anything else
  - Compound colliders
    - Multiple cheap to use colliders
- Mesh colliders
  - Avoid these if you can, but can be very good for detailed objects in a scene
  - VERY computationally expensive
- Static colliders
  - Non-rigidbody collider that doesn't move, good for things like floors, walls
  - You can add colliders to a GameObject without a RigidBody component

A quick note about time

- Frames are rendered as fast as we can render them or as slow as we need to render them.
  - Frame rate on the platform can be quite variable
- Physics updates should not be tied to frame rate.
- More on this when we talk about scripting, but remember that physics updates are different from frame updates and are handled separately so as not to tie physics to framerate.

The Rendering Pipeline

- The rendering pipeline is what gets all the game objects, assets, and eventually turn them into frames that goes onto the screen
- Not going into too much detail
- Generally you want to use the default pipeline unless you have a reason not to
- Can vastly change the feel of a game (pretty massively).
  - You can attach to different cameras
  - Two modes: standard and high definition

Assets

- The part of the game most of us don’t want to make (most of us are not artist)
- The asset store: The solution to that
  - Unity provides many free asset to import into your game
- Packages on the asset store can be pretty large, so be careful if you’re on a slow connection
  - Maybe a few GB
- Don’t be an “asset flipper”

Other Media

- Audio: Just think of them as more assets
  - Mixer component involves many sounds, use AudioSource.Play/PlayOneShot in scripts
    - No need to know differences unless you want to use them
    - You load sound onto mixer and mixer is what plays them
- Video: VideoPlayer Component, more assets
  - Can take any arbitrary source of video
  - Course project: feed security camera source into the VR world

Introduction to Scripting

- Do not fit nicely into ECS model
- Scripts are components that cause actions to happen, rather than merely store information.
- You create scripts directly within Unity. You can create a new script from the Create menu at the top left of the Project panel or by selecting Assets > Create > C# Script from the main menu.
- A script makes it connection to Unity by implementing a class, as a kind of the blueprint for creating a new component that can be attached to a GameObject.
  - **Script class name and file name should be the same.**

In scripting, we use an OOP to create something that replaces OOP!

What are scripts anyway?

- When you create a script initially it would have two default functions
- Start() and Update() (and other Updates)
  - Update runs every frame, handles frame update for the GameObject. Might include movement, triggering actions in respond to user
  - Start is used for initialization and will be called by Unity before gameplay begins.

```
using UnityEngine;
using System.Collections;

public class MainPlayer : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

    }
}
```

What are scripts anyway?

- Start() and Update() (and other Updates)
- Inspector integration (panel on the right side of Unity UI)
- GetComponent<T>()
  - Gets a Component, of type T, attached to the GameObject that this script is running on. Can use this on RigidBodies to move them.
- GameObject.Find("Name"), FindWithTag("Tag") (single thing returned), FindGameObjectsWithTag("Tag") (list returned)
- Reacting to events
  - For example when collision happens: OnCollisionEnter, OnCollisionStay and OnCollisionExit
- Instantiate() and Destroy()
  - Over the lifetime of a game, you would create and destroy many instances of the same GameObject

Learning a group of magic words, similar to learning a new library!

Time

Problem:

- Update() by default is once per frame- But frame rate is variable! This can be unreliable in some situations
  - Computer may not be able to keep up with display frame rate
  - For example, physics events happen asynchronously, so what should we do for the physics engine?

Solution:

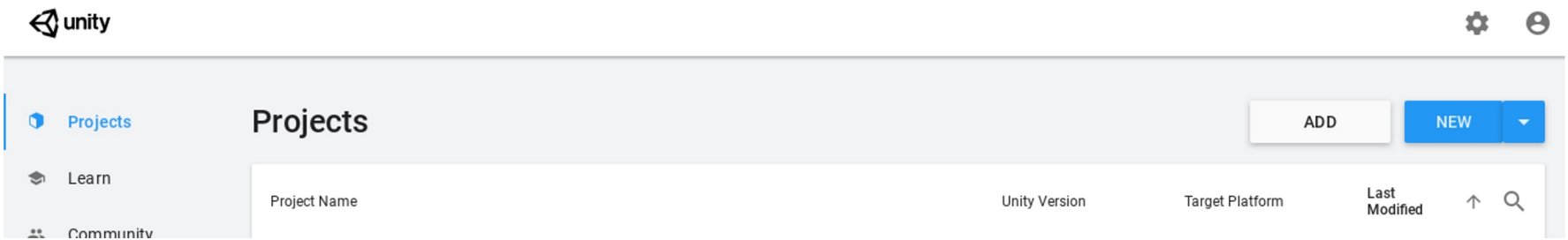
- Use Time.deltaTime to figure out time since last frame in Update(), update based on that.
- Use FixedUpdate() instead of update. We will do this in the demo.
  - The default value of FixedUpdate is 0.02 sec. But you can alter it in script or
    - Go to Edit > Settings > Time > Fixed Timestep and set it there.

Demo!

- Last time, we installed Unity. Today we'll use it.

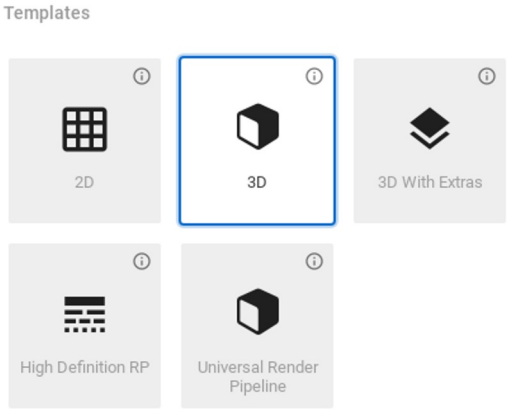
Creating a new project

- Select NEW on the side, under the Projects tab in UnityHub



Creating a new project

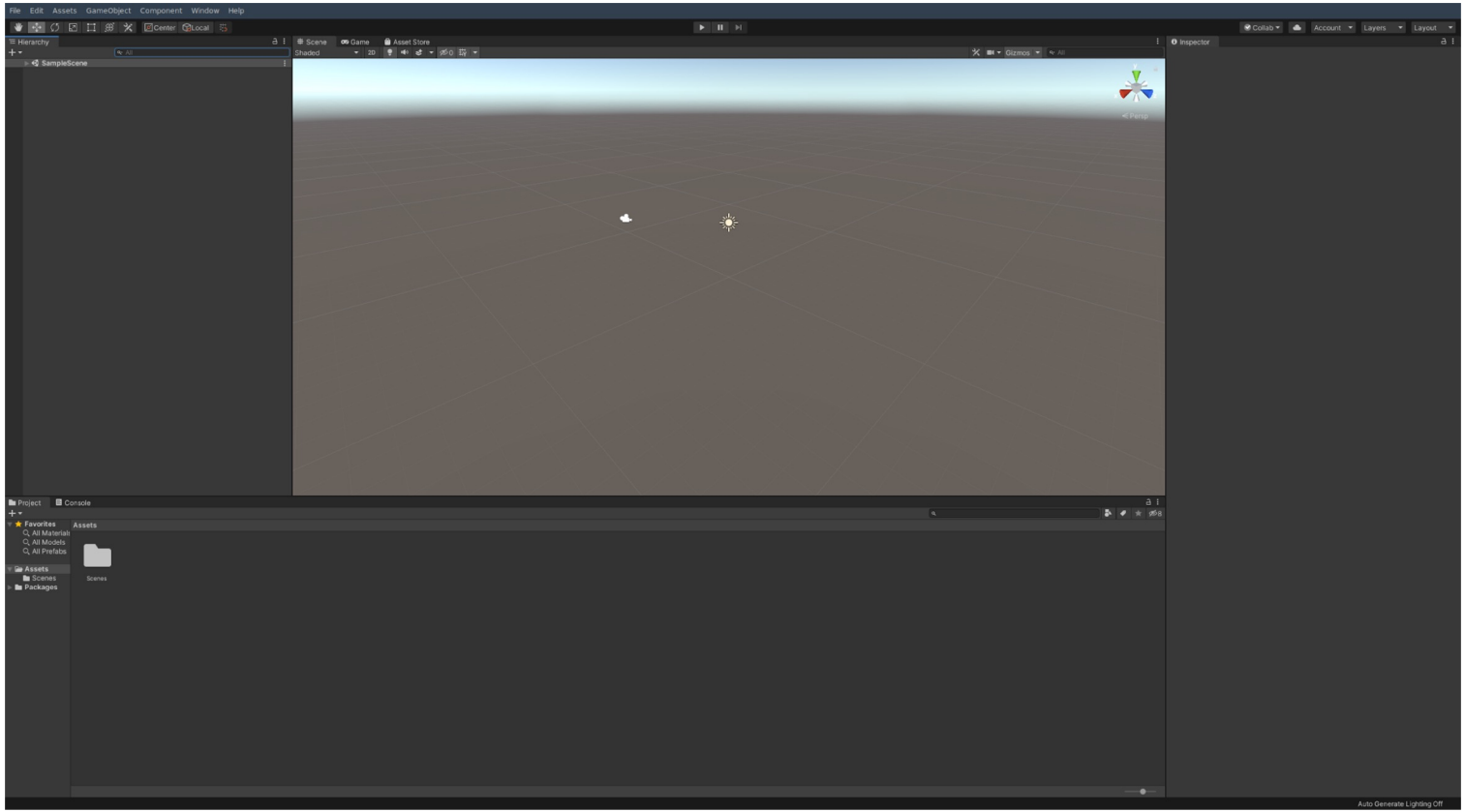
- Select NEW on the side, under the Projects tab in UnityHub
- Select 3D from the Templates, then name it something like “LabDemo1”



Creating a new project

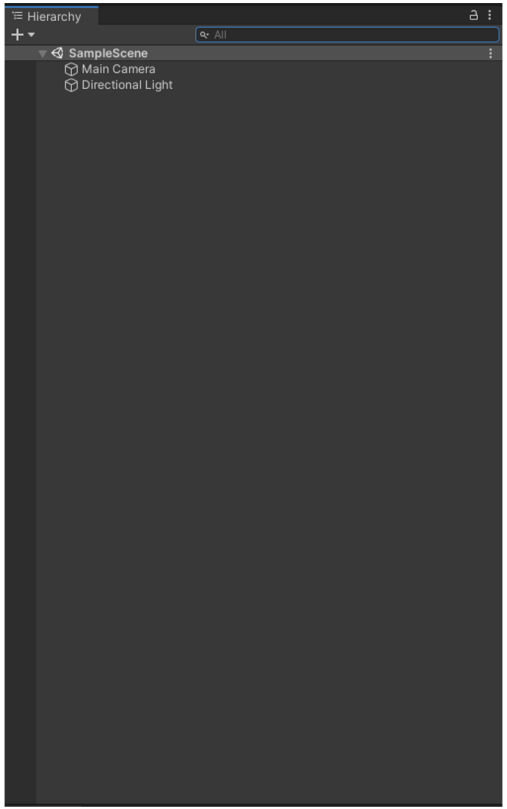
- Select NEW on the side, under the Projects tab in UnityHub
- Select 3D from the Templates, then name it something like “LabDemo1”
- Wait for it to import packages

Using Unity


File Edit Assets GameObject Component Window Help
Hierarchy
SampleScene
Scene
Game
Asset Store
Inspector
Project
Console
Favorites
Assets
Scenes
Packages
Auto Generate Lighting Off

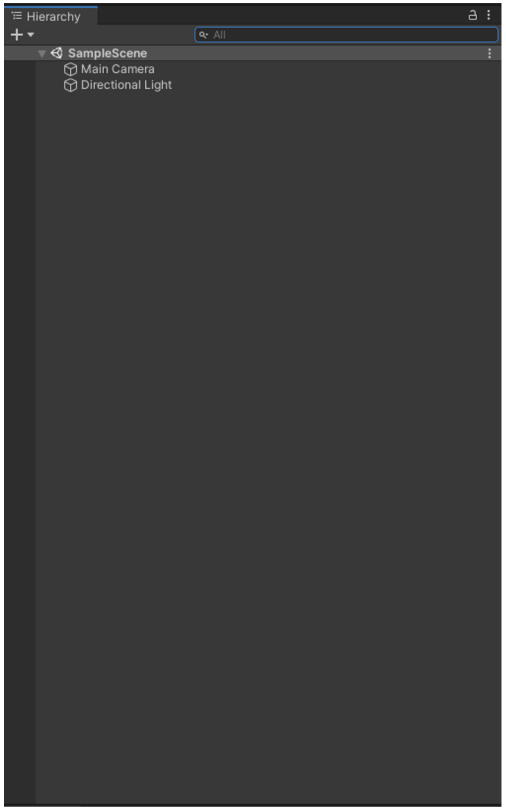
GameObjects

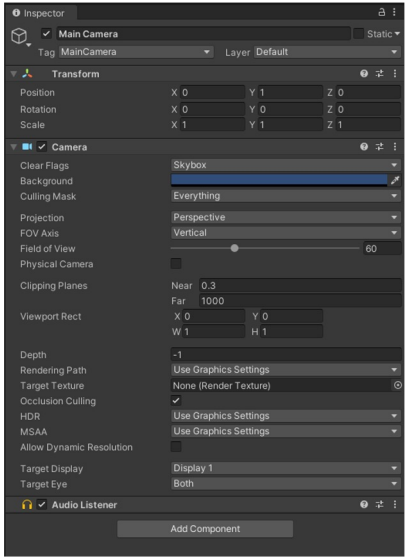
- Information about what GameObjects are in the scene and what their parents are
- Currently, we have a camera (“Main Camera”) and a light (“Directional Light”)



GameObjects

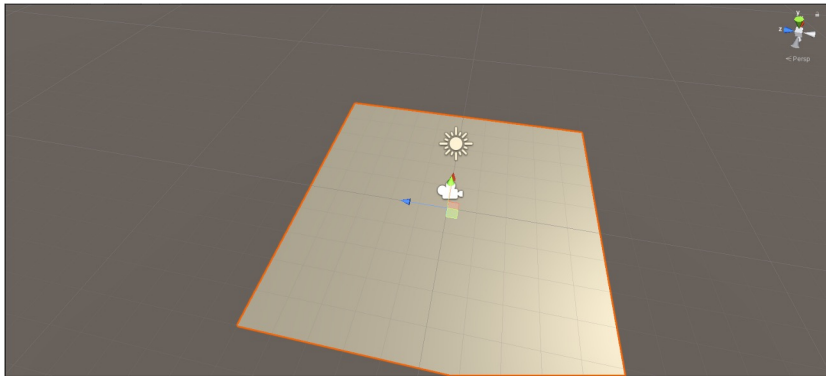
- Click the camera, and note the inspector on the right hand side of the screen.
- Under its transform, set its position to (0,1,0). It should change positions in





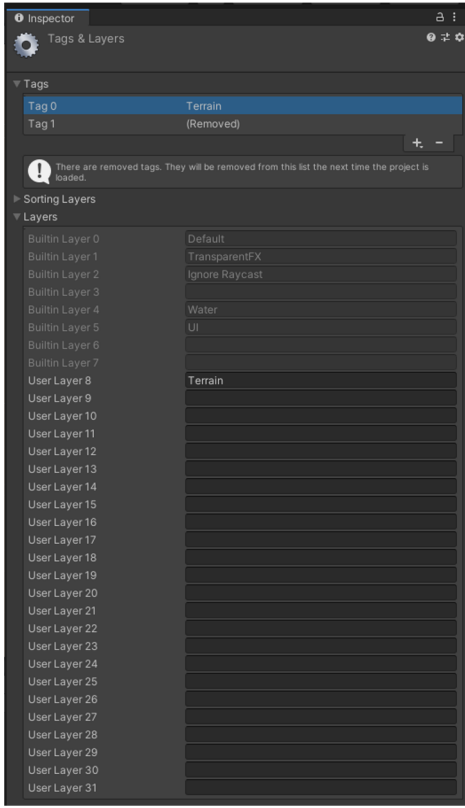
Floor

- Let’s add a floor
- On the top bar, go to GameObject -> 3D Object -> Plane
- Click it, set its position transform to (0,0,0)



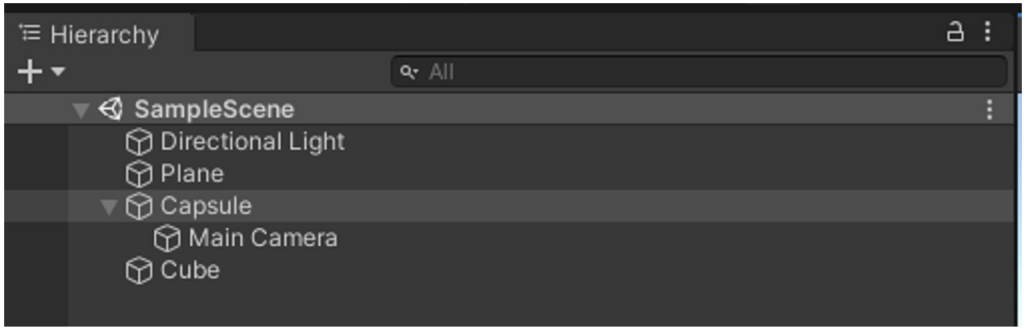
Floor

- Let’s add a floor
- In the inspector, look for “Layer” and “Tag”
- Go to “Add Layer” and “Add Tag”. Create a “Terrain” layer and tag.
- Add these to the plane.
- We’ll look at this more in scripting and compare the two.



The player and some objects

- GameObject -> 3D Object -> Capsule
- GameObject -> 3D Object -> Cube
- Set the camera as a child of our capsule
  - Whatever we do to do capsule would happen to the camera
  - Since transforms are computed relative to our parent, we now have a viable first or third person camera.



First Person vs Third Person Camera

Physics

- Select the capsule in the hierarchy. Right click it and rename it to “Player”.
- Go to the inspector on the right and click “Add component”. Search for “Rigidbody” and add it.
- In the rigidbody settings, set:
  - Use Gravity: On
  - Interpolate: Interpolate
  - Collision Detection: Continuous
  - Constraints -> Freeze Rotation: X, Y, and Z on

What happens when things go wrong, where to get help

- Accept that bugs are going to happen.
- Look on the forums for help, but know that suggestions may or may not be of good quality or be up to date.
- Ask in Canvas - Sometimes it’s easiest to just throw solutions at a problem and see which ones work.

Next Lab

- Scripting
- VR support options
- How we turn this in