Intro to Game Development (Part 2)

unity

UNREAL ENGINE

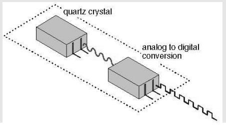
A1 Questions

- Notes:
  - We're looking for ways to streamline the XCode process (e.g. not needing to manually change signing team every time)
    - (the "automatic signing team" in Unity isn't your name, but a long hex code)
  - Git repo should not include build, Library, or Logs folders (although this isn't part of the grade)
  - Don't use UE4 on iOS until we figure out how to deal with weird Apple Dev problems
  - If having VM trouble, let us know
- A2 will be much less stressful
  - Only needs Blender (no platform-specific instructions)
  - Will assign it Thursday & have until next Thursday
- If you don't have headset, let us know and we'll get it outside class

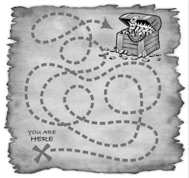
RECAP

(Most) (Modern) Game Engines

- Essentially realtime systems
  - **clock** (in this case tied to framerate) handles continuous logic
  - **Synchronization** between subsystems (physics, listeners, collision detectors, ray-tracing, etc.)
  - Tries to be as synchronous as possible (e.g. each frame lasts the same amount of time)
  - As opposed to solely reacting to input through **callbacks**, like in most mobile apps, text-based games, etc. which have no continuous logic beyond an actual clock
- Game engines are mostly **fully-featured APIs** containing many sub-APIs
  - Physics, collision logic, rendering, materials, optimized data structures, AI, audio etc.
- **Playground** to experiment with abstract CS concepts like clocks, state machines, data structures, black boxes, etc.



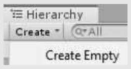
Example Application for the Class

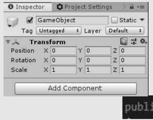


- Treasure Hunt
  - Seems to be a good example with connection to relevant areas of XR
  - Audio, locomotion, logic, haptics, IK, animation, etc.
  - First-person character, XR or non-XR...call them TreasureHunter
    - Can walk around and turn head
    - Can bump into objects in the environment but not go through them
    - Can grab collectible treasure with hands
    - Can hear
  - Treasure
    - 3D mesh that you hide in the VE to be found
    - Has some point value corresponding to how hard it is to find
    - Finding some amount of points wins the game
  - Main menu
    - Game start
    - Settings
    - High score
  - Win/Lose Screen

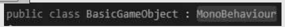
"Object" in a Game Engine

- Atomic "Object" with a 3D transform
  - 3D equivalent of basic Java Object…. Everything in the VE can be simplified down to this



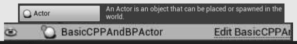


- **Unity: GameObject** (GO)

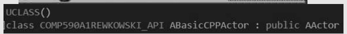


```
public class BasicGameObject : MonoBehaviour
```

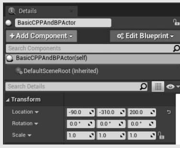
  - Most common term for this (and makes sense!)
  - List of components described by "**MonoBehaviors**"...which are also EACH the GO's type (multi-inheritance)
  - Name is one of few ways Unity follows industry standard & Unreal doesn't



An Actor is an object that can be placed or spawned in the world.



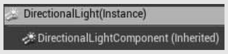
```
UCLASS()
class COMP590A1REWKOWSKI_API ABasicCPPActor : public AActor
```

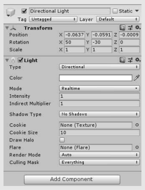


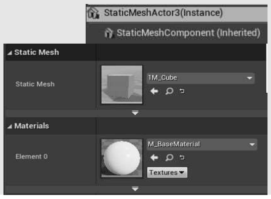
- **UE4: Actor & ActorComponent**
  - **Actor**: UE4's version of GameObject… should always have 3D root
  - **ActorComponent**: Similar to Actor except always part of Actor...meaningless by itself...
    - E.g. Mesh gives info about physical structure but needs 3D transform to be in game… so a mesh by itself is not too useful and is a COMPONENT of the Actor with transform
    - Might have a relative transform… but could be purely logical like most Unity (Colliders are exception)
  - Based on idea that everything in scene has some action...even if just a static obstacle

Quick Examples of Actor & ActorComponent

- (before we actual get into that part...just to have a high-level understanding)
- **StaticMeshActor** (mesh placeable in environment) is **Actor** with **StaticMeshComponent** (component defined by StaticMesh)
- **DirectionalLight** is **Actor** with **DirectionalLightComponent**
- In Unity, can add DirectionalLightComponent to something, but it can't **implicitly** define the GO as a light (there can never be a DirectionalLight "object" as far as Unity is concerned… just a GO with DirectionalLight behavior)

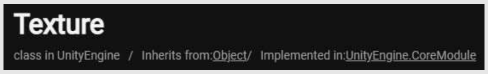

DirectionalLight(Instance)
DirectionalLightComponent (Inherited)

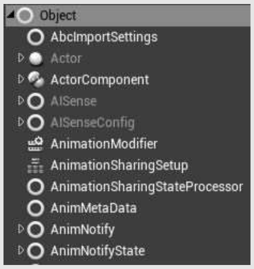

☑ Directional Light ☐ Static
Tag Untagged Layer Default
Transform
Position X -0.0637 Y -0.0591 Z -0.0009
Rotation X 50 Y -30 Z 0
Scale X 1 Y 1 Z 1
☑ Light
Type Directional
Color
Mode Realtime
Intensity 1
Indirect Multiplier 1
Shadow Type No Shadows
Cookie None (Texture)
Cookie Size 10
Draw Halo ☐
Flare None (Flare)
Render Mode Auto
Culling Mask Everything


StaticMeshActor3(Instance)
StaticMeshComponent (Inherited)
Static Mesh
Static Mesh 1M_Cube
Materials
Element 0 M_BaseMaterial

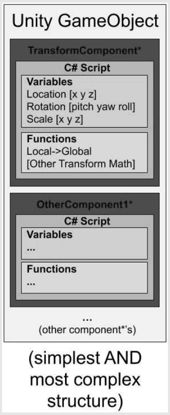
Non-GameObject Objects

- Things without physical representation in VE (Transform has no meaning)
  - Settings, save files, state machines, textures, shaders, etc.
  - Except for Actor & ActorComponent

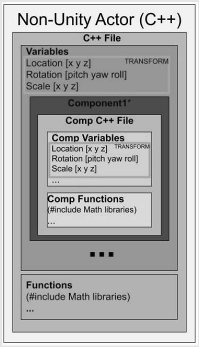

Texture
class in UnityEngine / Inherits from:Object/ Implemented in:UnityEngine.CoreModule


Object
AbcImportSettings
Actor
ActorComponent
AISense
AISenseConfig
AnimationModifier
AnimationSharingSetup
AnimationSharingStateProcessor
AnimMetaData
AnimNotify
AnimNotifyState

Unity vs Others

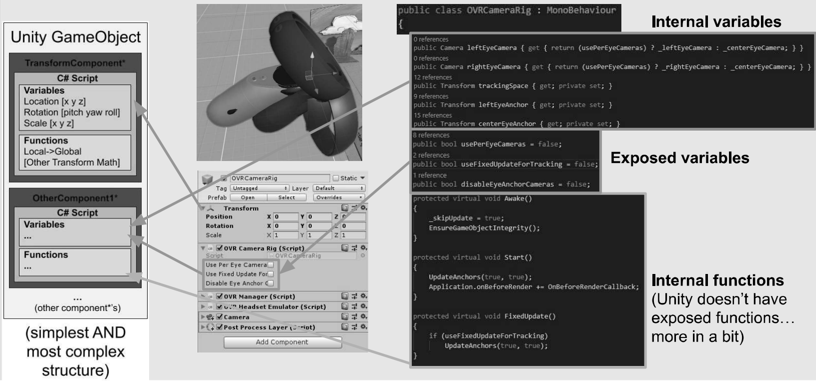

Unity GameObject
TransformComponent*
C# Script
Variables
Location [x y z]
Rotation [pitch yaw roll]
Scale [x y z]
Functions
Local->Global
[Other Transform Math]
OtherComponent1*
C# Script
Variables
...
Functions
...
...
(other component*'s)

(simplest AND most complex structure)


Non-Unity Actor (C++)
C++ File
Variables
Location [x y z] TRANSFORM
Rotation [pitch yaw roll]
Scale [x y z]
Component1*
Comp C++ File
Comp Variables
Location [x y z] TRANSFORM
Rotation [pitch yaw roll]
Scale [x y z]
...
Comp Functions
(#include Math libraries)
...
■ ■ ■
Functions
(#include Math libraries)
...

Structure of Unity's GameObjects

Similar to **strategy pattern** & core of why Unity's so "easy"...but also cause of unscalability


Unity GameObject
TransformComponent*
C# Script
Variables
Location [x y z]
Rotation [pitch yaw roll]
Scale [x y z]
Functions
Local->Global
[Other Transform Math]
OtherComponent1*
C# Script
Variables
...
Functions
...
...
(other component*'s)

(simplest AND most complex structure)

☑ OVRCameraRig ☐ Static
Tag Untagged Layer Default
Prefab Open Select Overrides
Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
☑ OVR Camera Rig (Script)
Script OVRCameraRig
Use Per Eye Camera ☐
Use Fixed Update For ☐
Disable Eye Anchor C ☐
☑ OVR Manager (Script)
☑ OVR Headset Emulator (Script)
☑ Camera
☑ Post Process Layer (Script)

```
public class OVRCameraRig : MonoBehaviour
{
```

Internal variables

```
0 references
public Camera leftEyeCamera { get { return (usePerEyeCameras) ? _leftEyeCamera : _centerEyeCamera; } }
0 references
public Camera rightEyeCamera { get { return (usePerEyeCameras) ? _rightEyeCamera : _centerEyeCamera; } }
12 references
public Transform trackingSpace { get; private set; }
9 references
public Transform leftEyeAnchor { get; private set; }
15 references
public Transform centerEyeAnchor { get; private set; }
```

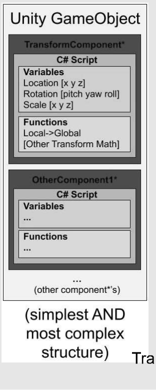
Exposed variables

```
8 references
public bool usePerEyeCameras = false;
2 references
public bool useFixedUpdateForTracking = false;
1 reference
public bool disableEyeAnchorCameras = false;
```

Internal functions (Unity doesn't have exposed functions… more in a bit)

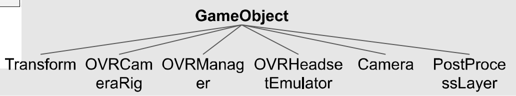
```
protected virtual void Awake()
{
    _skipUpdate = true;
    EnsureGameObjectIntegrity();
}

protected virtual void Start()
{
    UpdateAnchors(true, true);
    Application.onBeforeRender += OnBeforeRenderCallback;
}

protected virtual void FixedUpdate()
{
    if (useFixedUpdateForTracking)
        UpdateAnchors(true, true);
}
```

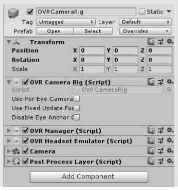
"Finding" Other Components


Unity GameObject
TransformComponent*
C# Script
Variables
Location [x y z]
Rotation [pitch yaw roll]
Scale [x y z]
Functions
Local->Global
[Other Transform Math]
OtherComponent1*
C# Script
Variables
...
Functions
...
...
(other component*'s)

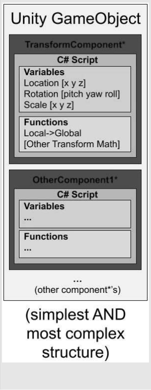
(simplest AND most complex structure)

- Components can find other components by getting their GameObject and asking it to GetComponent<Type> or GetComponents<Type> to return pointers
- This GameObject can be casted to any individual component
  - So OVRCameraRig (just display name in scene or of prefab) IS-A GameObject, Transform, OVRCameraRig, OVRManager, OVRHeadsetEmulator, Camera, PostProcessLayer. Outer GO can be grabbed with typedGO.gameObject
- But can't be casted to a class that already knows ALL of the components (aka definition)...GetComponents returns Type[]
  - No subtype of GameObject that defines the Components
  - So you as the dev must know structure beforehand...
  - While Unity doesn't.... It figures it out at runtime
  - GO is really just a collection of pointers to components




☑ OVRCameraRig ☐ Static
Tag Untagged Layer Default
Prefab Open Select Overrides
Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
☑ OVR Camera Rig (Script)
Script OVRCameraRig
Use Per Eye Camera ☐
Use Fixed Update For ☐
Disable Eye Anchor C ☐
☑ OVR Manager (Script)
☑ OVR Headset Emulator (Script)
☑ Camera
☑ Post Process Layer (Script)

What do you notice that's different from OO design?

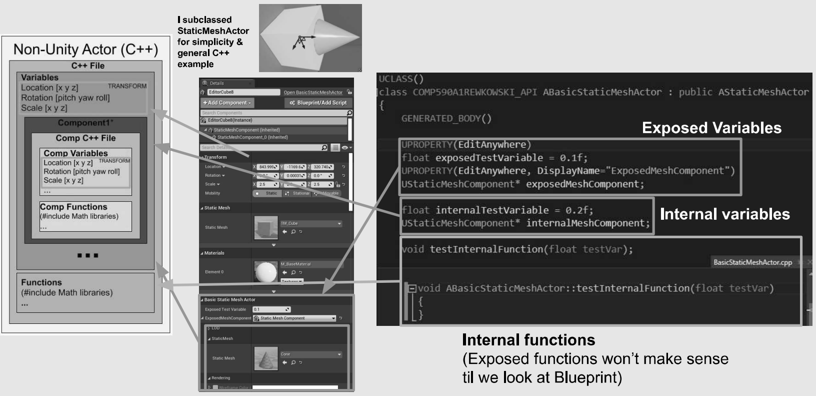

Unity GameObject
TransformComponent*
C# Script
Variables
Location [x y z]
Rotation [pitch yaw roll]
Scale [x y z]
Functions
Local->Global
[Other Transform Math]
OtherComponent1*
C# Script
Variables
...
Functions
...
...
(other component*'s)

(simplest AND most complex structure)

- There is no dev-editable script linking the Components
  - Outer GameObject cannot be subclassed… not defined any more specifically internally (can't make components member variables)
- So is there any complete definition of "what" this GO is?
  - No! Can create template through prefab but no way to autocomplete anything between scripts without the dev linking them manually (manually defining bunch of GetComponents AND checking for null/count…. Or linking through editor)
- Positives?
  - **Might allow more flexibility**
  - Might allow GameObject to be more barebones…
    - UE4 Actor contains a lot of data by default (multiplayer, rendering, etc.)…. Can use up RAM quickly
  - Might be simpler to formalize than UE4… it's a tree where components (nodes) can only access each other by going ALL the way back to root and then down to other component
  - Way simpler to understand. The structure never gets more complex

Basic Structure of Other Engine GameObjects & Actors

Object-oriented….not much different than other OO programs you're familiar with



I subclassed StaticMeshActor for simplicity & general C++ example

```
UCLASS()
class COMP590A1REWKOWSKI_API ABasicStaticMeshActor : public AStaticMeshActor
{
    GENERATED_BODY()

    UPROPERTY(EditAnywhere)
    float exposedTestVariable = 0.1f;
    UPROPERTY(EditAnywhere, DisplayName="ExposedMeshComponent")
    UStaticMeshComponent* exposedMeshComponent;

    float internalTestVariable = 0.2f;
    UStaticMeshComponent* internalMeshComponent;

    void testInternalFunction(float testVar);

void ABasicStaticMeshActor::testInternalFunction(float testVar)
{
}
```

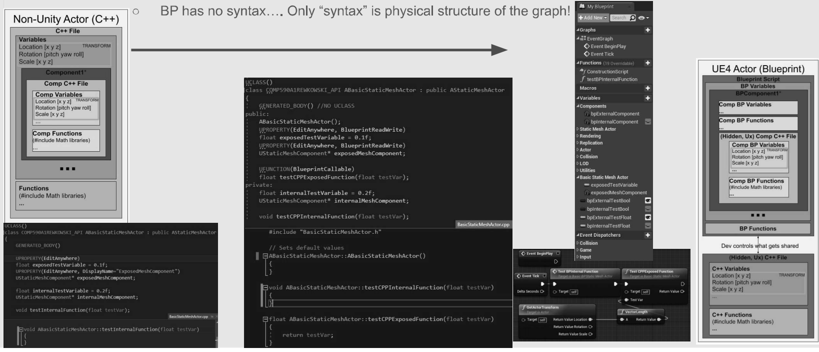
Exposed Variables

Internal variables

Internal functions
(Exposed functions won't make sense
til we look at Blueprint)

Towards Blueprints & High-Level Programming...

- Works alongside C++. UE4 Actor allows vars/functions to be shared
  - C++ can subclass BP and vice versa!
    - (generally don't subclass BP IN C++ b/c defeats the purpose of abstracting the C++)
  - BP has no syntax…. Only "syntax" is physical structure of the graph!

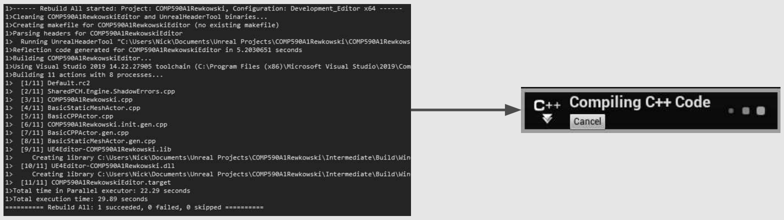


```
UCLASS()
class COMP590A1REWKOWSKI_API ABasicStaticMeshActor : public AStaticMeshActor
{
    GENERATED_BODY()

    UPROPERTY(EditAnywhere)
    float exposedTestVariable = 0.1f;
    UPROPERTY(EditAnywhere, DisplayName="ExposedMeshComponent")
    UStaticMeshComponent* exposedMeshComponent;

    float internalTestVariable = 0.2f;
    UStaticMeshComponent* internalMeshComponent;

    void testInternalFunction(float testVar);

void ABasicStaticMeshActor::testInternalFunction(float testVar)
{
}
```

```
UCLASS()
class COMP590A1REWKOWSKI_API ABasicStaticMeshActor : public AStaticMeshActor
{
    GENERATED_BODY() //NO UCLASS
public:
    ABasicStaticMeshActor();
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    float exposedTestVariable = 0.1f;
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    UStaticMeshComponent* exposedMeshComponent;

    UFUNCTION(BlueprintCallable)
    float testCPPExposedFunction(float testVar);
private:
    float internalTestVariable = 0.2f;
    UStaticMeshComponent* internalMeshComponent;

    void testCPPInternalFunction(float testVar);

    #include "BasicStaticMeshActor.h"

    // Sets default values
ABasicStaticMeshActor::ABasicStaticMeshActor()
{
}

void ABasicStaticMeshActor::testCPPInternalFunction(float testVar)
{
}

float ABasicStaticMeshActor::testCPPExposedFunction(float testVar)
{
    return testVar;
```

No More of This!

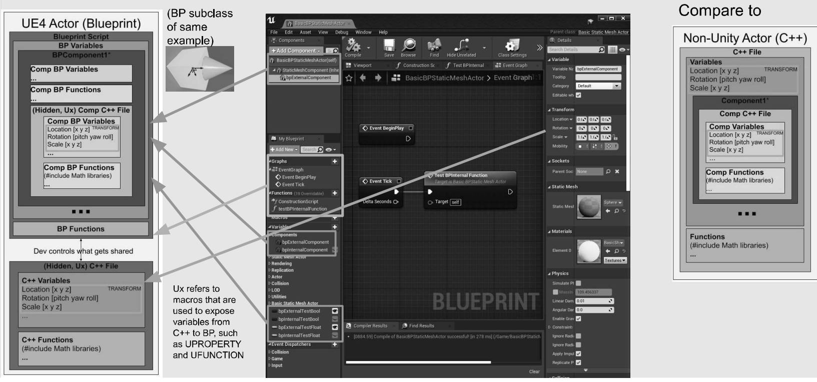
Either not deal with CPP at all or streamline compilation



```
1>------ Rebuild All started: Project: COMP590A1Rewkowski, Configuration: Development_Editor x64 ------
1>Cleaning COMP590A1RewkowskiEditor and UnrealHeaderTool binaries...
1>Creating makefile for COMP590A1RewkowskiEditor (no existing makefile)
1>Parsing headers for COMP590A1RewkowskiEditor
1>  Running UnrealHeaderTool "C:\Users\Nick\Documents\Unreal Projects\COMP590A1Rewkowski\COMP590A1Rewkows
1>Reflection code generated for COMP590A1RewkowskiEditor in 5.2030651 seconds
1>Building COMP590A1RewkowskiEditor...
1>Using Visual Studio 2019 14.22.27905 toolchain (C:\Program Files (x86)\Microsoft Visual Studio\2019\Com
1>Building 11 actions with 8 processes...
1>  [1/11] Default.rc2
1>  [2/11] SharedPCH.Engine.ShadowErrors.cpp
1>  [3/11] COMP590A1Rewkowski.cpp
1>  [4/11] BasicStaticMeshActor.cpp
1>  [5/11] BasicCPPActor.cpp
1>  [6/11] COMP590A1Rewkowski.init.gen.cpp
1>  [7/11] BasicCPPActor.gen.cpp
1>  [8/11] BasicStaticMeshActor.gen.cpp
1>  [9/11] UE4Editor-COMP590A1Rewkowski.lib
1>     Creating library C:\Users\Nick\Documents\Unreal Projects\COMP590A1Rewkowski\Intermediate\Build\Win
1>  [10/11] UE4Editor-COMP590A1Rewkowski.dll
1>     Creating library C:\Users\Nick\Documents\Unreal Projects\COMP590A1Rewkowski\Intermediate\Build\Win
1>  [11/11] COMP590A1RewkowskiEditor.target
1>Total time in Parallel executor: 22.29 seconds
1>Total execution time: 29.89 seconds
========== Rebuild All: 1 succeeded, 0 failed, 0 skipped ==========
```

Towards Blueprints & High-Level Programming...

- Simplest BP abstracts ALL C++.... never need to touch Visual Studio!
  - Simply adds BP Variables + Functions on same layer as C++. BP organized structurally



(BP subclass of same example)

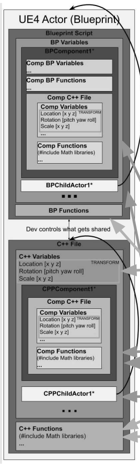
Ux refers to macros that are used to expose variables from C++ to BP, such as UPROPERTY and UFUNCTION

Compare to

Non-Unity Actor (C++)

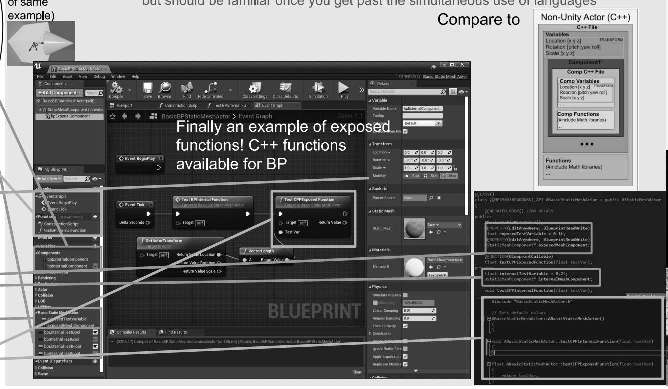
Towards Blueprints & High-Level Programming

- Mixing & matching C++ & BP can create complex OO structures… strange at first but should be familiar once you get past the simultaneous use of languages



(BP subclass of same example)

Compare to



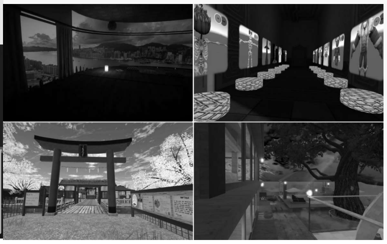
Finally an example of exposed functions! C++ functions available for BP

```
UCLASS()
class COMP590A1REWKOWSKI_API ABasicStaticMeshActor : public AStaticMeshActor
{
    GENERATED_BODY() //NO UCLASS
public:
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    float exposedTestVariable = 0.1f;
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    UStaticMeshComponent* exposedMeshComponent;
    UFUNCTION(BlueprintCallable)
    float testCPPExposedFunction(float testVar);
private:
    float internalTestVariable = 0.2f;
    UStaticMeshComponent* internalMeshComponent;

    void testCPPInternalFunction(float testVar);

    #include "BasicStaticMeshActor.h"

    // Sets default values
ABasicStaticMeshActor::ABasicStaticMeshActor()
{
}

void ABasicStaticMeshActor::testCPPInternalFunction(float testVar)
{
}

float ABasicStaticMeshActor::testCPPExposedFunction(float testVar)
{
    return testVar;
}
```

```
[0206.17] Compile of BasicBPStaticMeshActor successful! [in 235 ms] (/Game/BasicBPStaticMesh
```

Social XR: VRChat & Metaverse

- (things Unity is better at in addition to large-scale, procedural AR)
- Balance between optimization and runtime versatility
- **VRChat/Second Life**: user-created worlds linked by Hub, avatars, framework
- **Metaverse**: user-created or company-created worlds, linked by user "identity"
- Very difficult to support user-generated content in traditional game engines; requires mixing & matching components not known before runtime
- Unity GameObject more flexible; designed to add/remove components during runtime



UE4 Treasure Setup

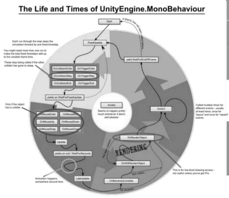
Intro to Game Development:

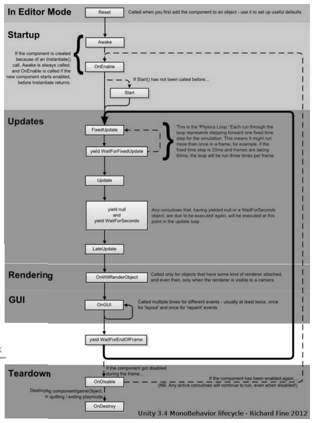
Game Engine Editors, Events, & Raycasting

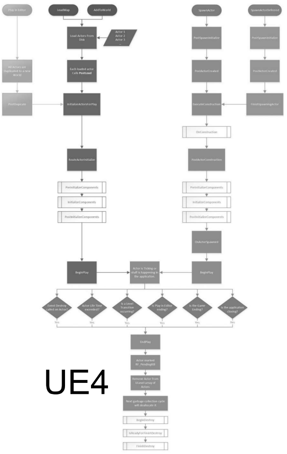
Events

- Engine usually calls things in certain sequence
- Called “lifecycle” of GO

Unity




In Editor Mode
Reset
Called when you first add the component to an object - use it to set up useful defaults
Startup
Awake
If the component is created because of an Instantiate() call, Awake is always called, and OnEnable is called if the new component starts enabled, before Instantiate returns.
OnEnable
If Start() has not been called before...
Start
Updates
FixedUpdate
yield WaitForFixedUpdate
This is the 'Physics Loop.' Each run through the loop represents stepping forward one fixed time step for the simulation. This means it might run more than once in a frame; for example, if the fixed time step is 20ms and frames are taking 60ms, the loop will be run three times per frame.
Update
yield null and yield WaitForSeconds
Any coroutines that, having yielded null or a WaitForSeconds object, are due to be executed again, will be executed at this point in the update loop.
LateUpdate
Rendering
OnWillRenderObject
Called only for objects that have some kind of renderer attached, and even then, only when the renderer is visible to a camera.
GUI
OnGUI
Called multiple times for different events - usually at least twice, once for 'layout' and once for 'repaint' events.
yield WaitForEndOfFrame
Teardown
If the component got disabled during the frame...
OnDisable
If the component has been enabled again...
(NB. Any active coroutines will continue to run, even when disabled!)
Destroying component/gameObject, or quitting / exiting playmode
OnDestroy
Unity 3.4 MonoBehavior lifecycle - Richard Fine 2012



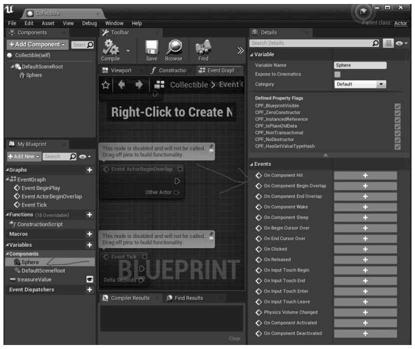
UE4

Important Events/Entry Points

- **BeginPlay/Start**: called when GO component first starts running in VE (usually when game starts playing) (Unity also has **Awake** for the entire GO)
- **Tick/Update**: called beginning of every frame
- **OnCollision[Enter/Exit]/Hit**: called when 2 physics objects collide
- **On[Trigger/Overlap][Enter/Exit]**: called when 2 triggers overlap
- (in UE4) **EndPlay**: called when level ends (Unity has similar **OnDestroy**…. But no callback for level end specifically). Used to transfer data between levels (Unity programmers usually use Singletons which is not ideal :/)
- (in UE4) **Constructor**: Used to pre-process stuff, pre-attach, etc.
  - Unity MonoBehaviors have a constructor that can be used to initialize components (e.g. if I have a component that relies on RigidBody physics, then make sure a RigidBody is also attached to the GO), but b/c of Unity’s engine passes, you’re discouraged from doing anything that involves OTHER GOs
- (in UE4, no Unity equivalent) **Level Blueprints**: Used to define how Actors relate to each other without modifying the Actor code (e.g. specify current camera)

In Blueprint

- You can see events specific to a component without going into the API

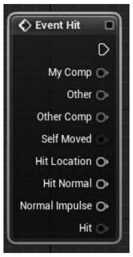


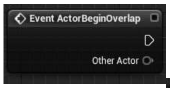
Collisions and Triggers

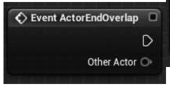
- **Collisions/Hits**: physical, Newtonian (e.g. objects bump into each other, stopping momentum)
- **Triggers/Overlaps**: some mesh zone that calls the function when some other mesh enters it (e.g. trigger something when room is entered)

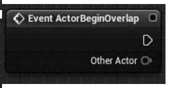
The Functions

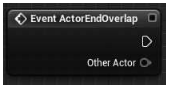
- (I'll show BP nodes since black box make explaining easier)
- **OnCollisionEnter/Event Hit**: called when objects hit each other
- **OnCollisionExit**: called when they stop hitting each other
  - UE4 doesn't distinguish between them b/c UE4 assumes, like in real physics, that a real physics collision can't cause them to overlap. Collision is single event
  - For things resting on each other, this still doesn't really make much sense (rarely describe "resting" position with single contact point….more on this in physics chapter).
- **OnTriggerEnter/[Actor/Component]BeginOverlap**: called when something enters a trigger zone
- **OnTriggerExit/[Actor/Component]EndOverlap**: called when it exits





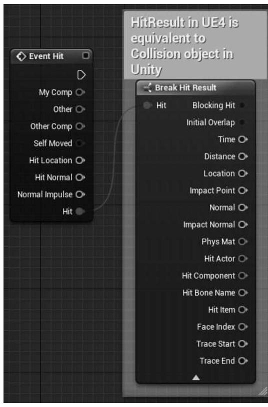






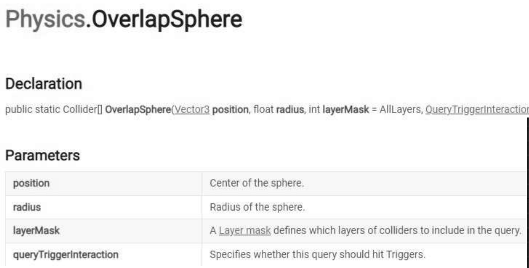
Important Difference between Trigger & Collision

- **Collisions** have collision info (impact force, impact location, impact normal, etc.)
- **Trigger/Overlap** events only have info about what they triggered
- Common source of confusion: Collider vs Collision.
  - **Collider** in Unity lets you get ref to triggered GO by Collider.gameObject. Used for Triggers. "Collider" is just collision box component
  - **Collision** object contains Collider AND physics info (used for collisions)
  - Also, Unity only handles collision per-component. UE4 can handle per-Actor or per-component



Collisions for Multiple Objects

- UE4: "GetOverlappingActors/Components"
- Unity has Physics.OverlapSphere



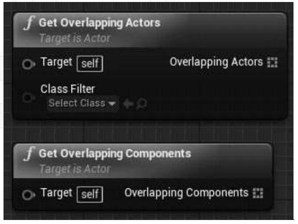
Physics.OverlapSphere

Declaration

public static Collider[] **OverlapSphere**(Vector3 **position**, float **radius**, int **layerMask** = AllLayers, QueryTriggerInteraction **queryTriggerInteraction** = QueryTriggerInteraction.UseGlobal);

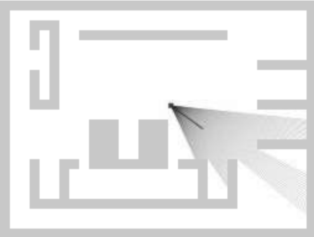
Parameters

position	Center of the sphere.
radius	Radius of the sphere.
layerMask	A Layer mask defines which layers of colliders to include in the query.
queryTriggerInteraction	Specifies whether this query should hit Triggers.



Ray-Casting: Key to Game Dev

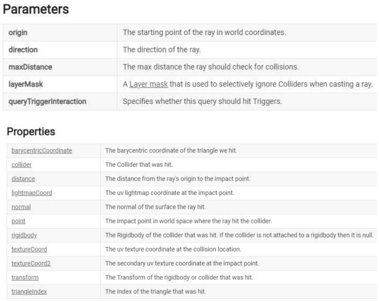
- Shoot a ray from a position in some direction and figure out what it hit
- Solves tons our problems. One of most common technique for understanding the VE
- We have raycasting, spherecasting, conecasting, etc.
- Like Collisions/Hits, raycasting gives some physics info (at which angle did ray hit? Where did it hit? etc.)
- Really useful for XR b/c it allows flexibility
- Allows for simple projection between 3D and 2D
- UE4 calls it "line-tracing"
- Simplified, controlled ray-tracing



The Tracing Function

UE4 & Unity differ slightly: Unity asks for ray length, UE4 asks for stop point. Return mostly same info

Unity



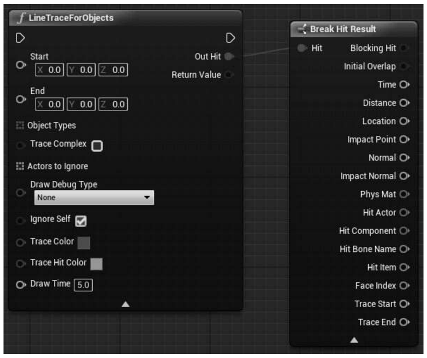
Parameters

origin	The starting point of the ray in world coordinates.
direction	The direction of the ray.
maxDistance	The max distance the ray should check for collisions.
layerMask	A Layer mask that is used to selectively ignore Colliders when casting a ray.
queryTriggerInteraction	Specifies whether this query should hit Triggers.

Properties

barycentricCoordinate	The barycentric coordinate of the triangle we hit.
collider	The Collider that was hit.
distance	The distance from the ray's origin to the impact point.
lightmapCoord	The uv lightmap coordinate at the impact point.
normal	The normal of the surface the ray hit.
point	The impact point in world space where the ray hit the collider.
rigidbody	The Rigidbody of the collider that was hit. If the collider is not attached to a rigidbody then it is null.
textureCoord	The uv texture coordinate at the collision location.
textureCoord2	The secondary uv texture coordinate at the impact point.
transform	The Transform of the rigidbody or collider that was hit.
triangleIndex	The index of the triangle that was hit.

UE4



Quick Exercise

- With these functions in mind (collision/trigger, raycast), how can we grab collectibles without hardcoding? There are multiple ways!
- OnTriggerEnter between a collision box on collectible and attached to controller (automatic pick-up or press button)
  - If InTriggerZone && ButtonPressed, then collect
- When user clicks button, spherecast or raycast to where they're pointing
  - On ButtonPressed, raycast and collect thing that it hits if it's a collectible

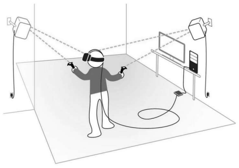
Demo (Implementing Collecting with TreasureHunter)

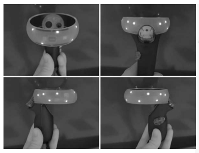
Intro to Game Development:

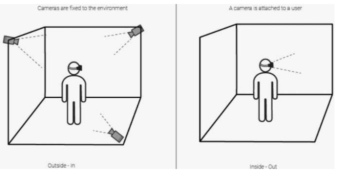
XR

XR Camera

- XR: special case of game dev with “camera”/head motion controlled by **tracking**
  - **Outside-in**: Vive lighthouses, Oculus Sensor
  - **Inside-out**: HMD-mounted cameras, visual odometry & sensor fusion to detect motion
    - Most modern MR does this
- Usually specify an “origin” for XR camera to control spawn point wrt VE
  - **VR**: calibrated by VR system (usually center of tracking space on floor)
  - **AR**: usually wherever the HMD is when the app starts
  - Without GameObject for origin, camera would always be wrt VE origin… then entire VE would need to shift to user to be in the right place






Cameras are fixed to the environment
A camera is attached to a user
Outside - in
Inside - Out

Thursday: Realtime Programming with Oculus Quest