INTRODUCTION TO UNREAL

CS378

DR SARAH ABRAHAM
GETTING STARTED IN UE4...

- Can be a bit intimidating to bypass Blueprints!
  - Lots of code functionality
  - Large API with varying levels of documentation
  - Easy to do it wrong

- Good starting points for documentation:

- But in practice you’re mostly going to rely on your search engine of choice...
**SCENES AND ACTORS**

- Game worlds and levels are similar to a movie: there is a scene, and actors within that scene.
- Scene is composed of actors (all objects in the scene are a type of actor).

**Actors in this scene:**
- Skybox
- Building
- Lights
- Plant
- Furniture
- Ground
WORKING WITH ACTORS

- Base class of all gameplay objects that can be placed in the world
  - Can be spawned into the world
  - Can contain components which determine actor’s behavior
- Handles memory management to spawn and destroy the actor object
- As an example, here are the virtual functions called on load:
  - `PostLoad` -> `OnComponentCreated` ->
    `PreRegisterAllComponents` -> `RegisterComponent` ->
    `PostRegisterAllComponents` -> `PostActorCreated` ->
    `UserConstructionScript` -> `OnConstruction` ->
    `PreInitializeComponents` -> `Activate` ->
    `InitializeComponent` -> `PostInitializeInitializeComponents` ->
    `BeginPlay`
ACTOR CLASSES

- Over 240 derived classes of AActor
- Many different types of functionality depending on the situation
  - You certainly won’t need to use all of them but may be useful!
- Some common ones:
  - APawn
    - Physical representation of actors that can be possessed by a player or AI
  - AController
    - Non-physical actors that can possess pawns and control actions
  - ATriggerBase
    - Actors that can generate collision events
**ACTOR COMPONENTS**

- Actors have components that implement much of their behavior and functionality.

- UAActorComponent is base class but do not have transforms (i.e. scale, rotate, translate).

- Scene Components have transforms but not necessarily a geometric representation.

- Primitive Components are SceneComponents with a geometric representation.

- UAActorComponents can be registered to receive frame updates.
  - Not very performant so only register when necessary and unregister when no longer necessary.
**UO Objects**

- Base class of all objects in Unreal Engine
- Not required to use but provides useful functionality
- Includes functionality for:
  - Garbage collection
  - Reflection
  - Serialization
  - Reference updating
  - etc...
 UE4 has quite a few code standards you should aim to follow

- Extremely helpful on large, constantly changing teams
- Still helpful on smaller, stable teams for readability

Full guide here: https://docs.unrealengine.com/en-US/Programming/Development/CodingStandard/index.html but we will discuss a lot of this later...

- Prefix U inherits from UObject; Prefix A inherits from AActor; Prefix S inherits from SWidget; Prefix I are abstract interfaces; Prefix E are Enums, Prefix F is for structs and most other classes
UOBJECT LIFE CYCLE

- All UObjects and sub-classes are garbage collected
  - Upon creation, UE4 adds object to its internal object list
  - Create using creation methods
- Caveat: **never use new!!**
- Create a strong reference using UPROPERTY macro or can manually flag
- Can call Destroy or DestroyComponent on actors and components
  - Will mark the object for destruction and null the UPROPERTY pointer upon destruction
What are macros?

- Lines of code that are expanded by the preprocessor and substituted in during compilation
- Can be “object-like” (no arguments) or “function-like” (with arguments)

Used for abstracting frequently used code or definitions

Used for creating **meta-object systems** in large, complex frameworks
MACROS AND SPECIFIERS IN UNREAL

- UE4 heavily uses macros to control engine and editor functionality
  - UPROPERTY creates strong references to objects, exposes property to the editor, and allows property to be recognized by reflection
  - UFUNCTION allows function to be recognized by reflection
- **Specifiers** inform how object or function should be used:
  
  ```
  UPROPERTY(Replicated, EditAnywhere, BlueprintReadWrite, Category = “Character”)
   
  float health;
  
  UFUNCTION(BlueprintCallable, Category = “Character”)
  
  void takeDamage();
  ```
Several different ways to create objects in UE4 -- none of which involve calling `new`!

All UObjects (whether actors or components) should use their default creation methods:

- `FooObject* f1 = NewObject<FooObject>();`
- `World->SpawnActor<FooActor>(FVector::ZeroVector, FRotator::ZeroRotator);`
- `UComponent* FooComponent = CreateDefaultSubobject<FooComponent>(TEXT("ComponentName"));` // Only use in object constructor
Because of this compilation process, you must be cognizant of the macros and includes associated with generated code

- i.e. do no randomly start deleting pre-generated code!

- `#include “MyObject.generated.h”`
  - Must be last include in header of MyObject

- `UCLASS` specifies class is a `UObject` and should have reflection data

- `GENERATED_BODY()` placed at start of the class declaration
  - UE4 will populate this with all necessary boilerplate code for this type
ULEVEL

- Level object that contains list of actors (lights, volumes, mesh instances, etc), geometry (BSP) information, and a World it is associated with

- Multiple levels can be loaded and unloaded in a World to stream assets

- An ALevelScriptActor exists within a level and executes level-wide logic on actor instances
  - Access that via code or Blueprint to deal with level-wide behaviors
ACTORS' GAMEPLAY LOOP

- OnConstruction(const FTransform & Transform) called when actor is placed in editor or spawned at runtime
- BeginPlay() called when play begins for this actor
- Destroy(bool bNetForce, bool bShouldModifyLevel) called to initiate destruction of the instance
- Tick(float DeltaSeconds) called every frame on this actor
  - Avoid this at all costs!
  - How?
EVENTS AND DELEGATES

- Events (or timers/delegates) should be used over tick whenever possible
  - ...It should pretty much always be possible...
- Many Blueprint events provided for common use-cases
- Can implement/call events in either C++ or Blueprints
  - Must use function specifiers to override in C++
- Can use delegates for native C++ code (will cover those later)
USING BLUEPRINT EVENTS
WHAT ARE SOME TYPES OF GAME EVENTS?
SOME BLUEPRINT EVENTS...
INPUT EVENTS

- Set input mappings via configuration files
  - Axis (continuous) or action (press and release)
- Input Components link mappings to game actions within the controller or pawn
- Input event can be called from C++ or Blueprint
Collision and Overlap Events

- Can set actors to ignore, overlap or block other object types in the scene.
  - Overlap will generate events but not result in a physical collision.
  - Block will result in a physical collision and generate events if flagged.
WHAT ABOUT THINGS THAT AREN’T SPAWNED IN?

- Many “physical” things are spawned into a game level
- What sort of things are not spawned into a game level but are helpful to have/track?
GAME STATES

- Often we want to know something about the state of the game
  - How many people are playing?
  - Who is winning?
  - What are the rules?

- GameMode, GameState and PlayerState provide information about the current state and how to play
GAME MODE

- Game modes define the rules of the game and exist only on the server
  - Number of players/spectators present and allowed
  - How players enter and are spawn/respawned in the game
  - Pause-handling
  - Level-transitions and cinematic mode handling
- Two base classes to choose from
  - AGameModeBase for simplified handling
  - AGameMode includes extra support for multiplayer and legacy systems
- Consider: why does UE4 have the game modes setup it does?
GAME STATE

- Game states allow clients to monitor the state of the game and are replicated to all clients
  - Built around networked multiplayer but useful for local multiplayer/single-player as well
- Tracks game-wide properties such as:
  - List of connected players
  - Team scores
  - Missions completed
PLAYER STATE

- Player states are created for each player contain information about the player such as name, score, health, etc.
  - Built around networked player but useful in local multiplayer/single-player games as well.
- Replicated to all clients and contains network information (such as ping) about the player.
HOMEWORK BEFORE NEXT CLASS...

- Makes sure you have completed Assignment 0 (creating an Epic account and downloading Unreal Engine 4.27/4.25) to the machine you will be working on for the rest of the semester
  - This will take a while and require a decent Internet connection so give yourself enough time!
- Next class will be Lab 1, where you will familiarize yourself with UE4
  - I will be streaming via Twitch so you can work from home if you’d like to work on a desktop for your development