Unreal Engine Class Demo Guide

Oct 28, 2026 · @FabulaStudio

Four demos cover the student's request: build and rig a character with MetaHuman and Control Rig, drive an enemy with AI Perception and State Tree, command a Total War style unit with one squad brain and formation slots, and build menus in UMG. Everything below is Blueprint only, targets Unreal Engine 5.6 or newer, and fits in a Third Person template project.

Before class

One Third Person template project, prepared the night before, carries all four demos. Create it in Unreal Engine 5.6 or newer with Starter Content on, then do the setup below so no demo stalls on a plugin restart.

Item	What to do	Why
Project	Games, Third Person, Blueprint, Starter Content on	Gives you a rigged mannequin (Manny and Quinn), a playable level and an animation blueprint to copy from
Plugins	Enable MetaHuman, Control Rig, IK Rig, StateTree, GameplayStateTree, Common UI, Mass Entity and Mass AI, then restart	Each one needs an editor restart, which eats class time
NavMesh	Drop a Nav Mesh Bounds Volume over the whole level, press P to show the green walkable area	Every AI demo depends on it
Enemy mesh	Duplicate the Third Person character blueprint and name it BP_Enemy, tint its material red	A visibly different body makes the AI demo readable from the back of the room
Spare content	Download the Game Animation Sample project and a free Fab soldier or zombie pack	Backup assets if a live download fails

Item	What to do	Why
Epic account	Sign in to Epic in the editor and confirm MetaHuman Creator opens	MetaHuman requires an Epic login and internet

Keep one finished copy of each demo in a second project so you can open the answer if a live build breaks.

Creating and rigging 3D characters

The fastest demo is a MetaHuman built inside the editor, because it arrives rigged, skinned and ready to animate in minutes. Modelling and rigging a character by hand belongs in Blender, so the Unreal side of the lesson is about importing, retargeting and adding controls.

Three roads to a rigged character

Road	Tools	Time in class	Best for
Build a human in editor	MetaHuman Creator (in the engine since 5.6), Mesh to MetaHuman	15 min	Showing what "rigged" means with zero modelling
Import a custom model	Blender with Rigify or Auto-Rig Pro, FBX export, IK Rig and IK Retargeter	20 min	Students who already sculpt or model
Buy or download	Fab marketplace, Mixamo	5 min	Enemies, crowds, placeholder soldiers

Live demo flow (about 15 minutes)

1. Open MetaHuman Creator from the Content Browser, pick a preset body, change the face, hair and clothing, then click Assemble to drop it into the project.
2. Open the skeletal mesh and show the bone hierarchy. Ask the room: what happens if the knee bone did not exist? That is the whole point of rigging.
3. Open the character's Control Rig asset. Grab the hand control and drag it so the arm follows. Explain controls as the puppet strings, bones as the puppet.

4. Create a Control Rig from the Mannequin instead, with Modular Rig. Drag the Spine, Arm and Leg modules onto the skeleton and show that a rig can be built from parts in two minutes.
5. Right-click the MetaHuman skeletal mesh and choose Retarget Animations. Pick Manny as the source, choose the Third Person animations, and export. The new character now runs and jumps with no hand animation.
6. Replace the mesh in the Third Person character blueprint, set the new animation blueprint, and press Play.

Importing a Blender character (if a student brings one)

1. In Blender, rig with Rigify or a plain armature, apply all transforms, scale at 1 unit = 1 cm, and export FBX with Only Deform Bones and Leaf Bones off.
2. Import into Unreal with Skeletal Mesh and Import Animations on, no skeleton selected, so Unreal creates one.
3. Create an IK Rig for the new skeleton. Define retarget chains for spine, head, each arm and each leg, then set the retarget root to the pelvis. Unreal 5.4 and later guesses the chains with Auto Create Chains.
4. Create an IK Retargeter from Manny's IK Rig to the new rig, fix the retarget pose if the arms are in A pose instead of T pose, and export the animations.

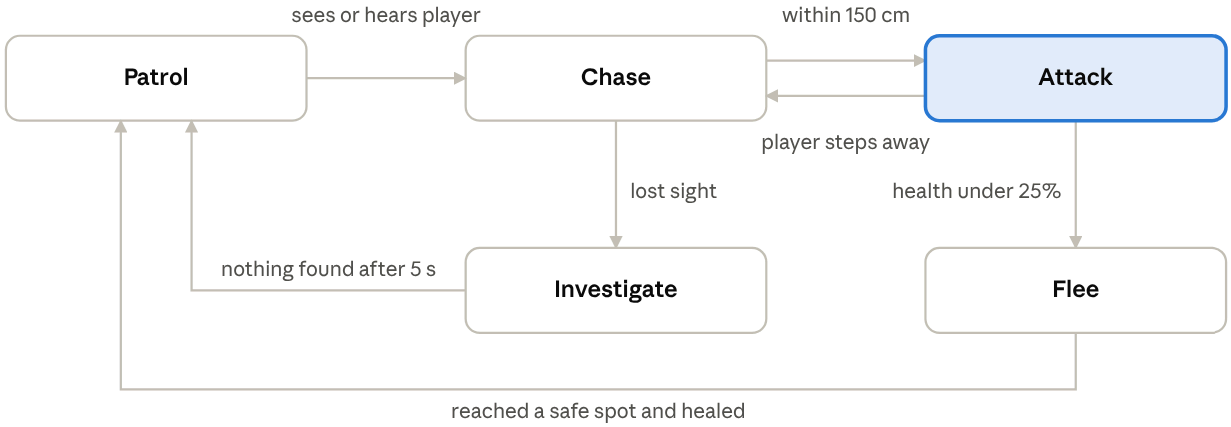
What to say about Skeletal Mesh Editing Tools

Unreal can paint skin weights, add bones and fix a bad import inside the Skeletal Mesh Editor. Show the weight-paint brush once so students know it exists. Full rigging still belongs in Blender.

Semi-smart enemy AI

A semi-smart enemy is three Unreal pieces working together: a NavMesh so it can walk, an AI Perception component so it can see and hear, and a State Tree that decides what to do with what it senses. All of it is Blueprint and builds live in about 30 minutes.

Perception events move the enemy between five states



enemy AI states · 5 states, 7 transitions

The accent box is where the player takes damage. Every other state exists to get the enemy there or to keep it alive. Add a state by adding a box, which is also how you add one in the State Tree editor.

The pieces

Piece	Where it lives	Job
Nav Mesh Bounds Volume	The level	Marks walkable ground so Move To has a path
BP_EnemyController (AI Controller)	Possesses BP_Enemy	Holds perception and the State Tree component
AI Perception component	On the controller	Sight, Hearing and Damage senses fire events
State Tree asset	Content Browser, schema StateTree AI Component	States, tasks and the transitions between them
Environment Query System (EQS)	Optional query asset	Finds cover or a safe point for Flee

Build steps

1. Open BP_Enemy. Set AI Controller Class to a new blueprint BP_EnemyController and Auto Possess AI to Placed in World or Spawned.

2. In BP_EnemyController add an AIPerception component. Add a Sight sense with radius 1500, lose-sight radius 2000, peripheral angle 70, and tick Detect Neutrals, because Blueprint actors count as neutral by default. Add a Hearing sense with range 2500.
3. On the component's On Target Perception Updated event, branch on Successfully Sensed. If true, store the actor in a TargetActor variable. If false, clear TargetActor and store the stimulus location in LastKnownLocation.
4. Add a StateTreeAI component to the controller and create a State Tree asset with the StateTree AI Component schema. Call Start Logic on Begin Play.
5. Add five states that match the diagram. Each state gets one task: Patrol uses Move To with Get Random Reachable Point In Radius 1000 then a Delay of 2 seconds; Chase uses Move To on TargetActor with acceptance radius 150; Attack plays an attack montage, calls Apply Damage, then delays 1 second; Investigate moves to LastKnownLocation and delays 5 seconds; Flee moves to the result of an EQS query or to a tagged SafeSpot actor.
6. Add transitions as conditions on each state, reading the controller's variables through property binding: TargetActor valid moves to Chase, distance to TargetActor below 150 moves to Attack, TargetActor cleared moves to Investigate, Health below 25 percent moves to Flee, and task completion in Investigate or Flee moves back to Patrol.
7. Press Play. Hold the apostrophe key for the Gameplay Debugger to watch the perception cones and the active state. Press P to show the NavMesh if the enemy stands still.

Making it feel smart in five extra minutes

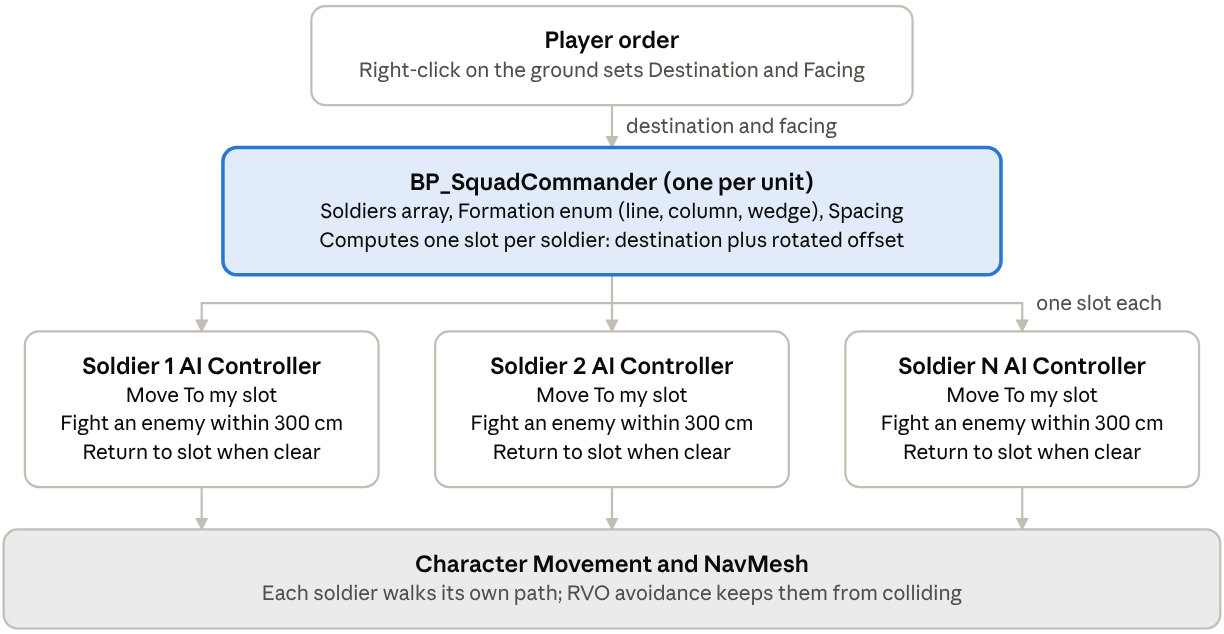
- Call Make Noise in the player's sprint input so running draws the enemy from behind walls.
- Add the Damage sense, so hitting an enemy that cannot see you still turns it around.
- Replace the random patrol with a Spline component and move between its points.
- Give the Attack state a random cooldown between 0.8 and 1.6 seconds so a group does not swing in unison.
- Spawn three enemies. The same tree runs on each with no extra code, which sets up the next section.

Behavior Trees with a Blackboard do the same job and have far more YouTube tutorials. Teach State Tree because it is Epic's current default, and tell students the Behavior Tree videos map one to one: a Blackboard key is a State Tree parameter, a Task is a Task, a Decorator is a transition condition.

Many soldiers, one commander

A Total War unit is one brain and many bodies. The commander owns the order and the formation math, and each soldier runs a tiny AI that walks to its assigned slot and fights whatever comes close. The linked video is a Total War: Attila mod (1212 AD), and every block of spearmen in it works this way.

One commander computes the slots; every soldier only walks to its own



squad architecture · 1 commander, N soldiers

Nothing in a soldier knows about the other soldiers. That is what makes it scale: adding a soldier is one more entry in the array, and changing formation is one function on the commander.

Formation math

The commander hands soldier number i a slot. Compute the offset in the formation's own space, then rotate it by the unit's Facing and add the Destination.

Formation	Row of soldier i	Column of soldier i	Offset (X back, Y right)
Line, c soldiers per row	i divided by c, rounded down	i modulo c	X = minus row times Spacing, Y = (column minus (c minus 1) over 2) times Spacing

Formation	Row of soldier i	Column of soldier i	Offset (X back, Y right)
Column	i divided by 2	i modulo 2	Same formula with c = 2
Wedge	i divided by 2	alternates left, right	X = minus row times Spacing, Y = plus or minus row times Spacing over 2

Build steps (about 40 minutes live)

1. Duplicate BP_Enemy as BP_Soldier. Add variables Commander (object reference to BP_SquadCommander), SlotIndex (integer) and TargetSlot (vector).
2. Create BP_SquadCommander as a plain Actor. Add variables Soldiers (array of BP_Soldier), Formation (an enum with Line, Column, Wedge), Spacing (float, 150), SoldiersPerRow (integer, 8), Destination (vector) and Facing (rotator).
3. Add a pure function GetSlotLocation(Index) that implements the table above. Use Rotate Vector with Facing, then add Destination.
4. Add a function IssueMoveOrder(NewDestination, NewFacing). Store both, then For Each Loop over Soldiers: set TargetSlot to GetSlotLocation(Index), get the soldier's AI Controller and call Move To Location with acceptance radius 50.
5. On Begin Play, loop UnitSize times: Spawn Actor From Class BP_Soldier at GetSlotLocation(Index), set its Commander and SlotIndex, and add it to Soldiers.
6. In the player controller, on right mouse button: Get Hit Result Under Cursor By Channel, build Facing with Find Look At Rotation from the unit's center to the hit point, and call IssueMoveOrder on the selected commander. Use the Top Down template camera or raise the Third Person camera so students can see the whole block move.
7. Give BP_Soldier a shorter State Tree than the enemy: Follow Slot (Move To TargetSlot), Engage (an enemy is within 300 cm, so chase and attack it) and Return (no enemy, go back to TargetSlot). The perception setup from the previous section carries over unchanged.
8. Reparent BP_EnemyController to DetourCrowdAIController, or tick Use RVO Avoidance on the Character Movement component, so sixty soldiers do not pile into one another at a doorway.
9. Make the commander itself semi-smart with a second State Tree on the commander actor: Hold, Advance toward the nearest enemy commander, Charge when within 800 cm, Retreat when fewer than 30 percent of Soldiers are alive. Students now have unit-level and soldier-level AI from the same tool.

How far Blueprint scales

Full Character actors with skeletal meshes run comfortably at a few dozen per unit and a few hundred in a scene on a classroom laptop. Past that, turn on the Animation Budget Allocator plugin and lower Animation Blueprint update rates by distance. Thousands of soldiers, as in the video, need the Mass Entity framework that drives the City Sample crowds, which is a C++ topic. Say so in class so students aim for 50 to 200 soldiers, which already looks like Total War from a tactical camera.

Interactable menus

Menus in Unreal are Widget Blueprints built in UMG, the Unreal Motion Graphics editor. A main menu, a pause menu and a settings screen take about 25 minutes live, and the same three ideas cover every menu a student will build: a widget holds the layout, a button event does the work, and the input mode decides whether the mouse talks to the menu or the game.

The three menus and what each teaches

Menu	Opened from	Teaches
Main menu	Its own empty level, MainMenu, set as the Game Default Map	Create Widget, Add to Viewport, Set Input Mode UI Only, Open Level
Pause menu	Escape key in the gameplay level	Set Game Paused, Set Input Mode Game and UI, Remove from Parent, toggling
Settings	A button on either menu	Sliders and combo boxes bound to Game User Settings, Apply Settings, Save

Build steps

1. Create a Widget Blueprint WBP_MainMenu. Drop in a Canvas Panel, then a Vertical Box anchored to the center. Add a Text Block for the title and three Buttons, each with a Text Block child: Play, Settings, Quit. Set the buttons to 300 by 60 and give them a hover tint so the room can see focus change.
2. Select the Play button, scroll to Events in the Details panel, click the plus beside On Clicked. In the graph, call Open Level by Name with the gameplay level's name. Quit calls Quit Game. Settings calls Create Widget on WBP_Settings and Add to Viewport,

then Remove from Parent on self.

3. Create a blueprint PC_Menu from Player Controller. On Begin Play: Create Widget WBP_MainMenu, Add to Viewport, Set Input Mode UI Only, Set Show Mouse Cursor true. Create a Game Mode blueprint GM_Menu that uses PC_Menu and an empty Default Pawn. In the MainMenu level's World Settings pick GM_Menu, and in Project Settings set Game Default Map to MainMenu.
4. Create WBP_Pause with Resume, Settings and Main Menu buttons. In the gameplay Player Controller, add an Enhanced Input action IA_Pause bound to Escape. On Triggered: if the pause widget is not valid, create it, Add to Viewport, Set Game Paused true, Set Input Mode Game and UI, show the cursor. Resume reverses all four and clears the reference. On IA_Pause set Execute when Paused to true or the Escape key dies the moment the game pauses.
5. Create WBP_Settings. Add a Slider for master volume, a Combo Box (String) for resolution, and a Check Box for fullscreen. On Value Changed of the slider, call Set Sound Mix Class Override or set a Sound Class volume. On the combo box, call Get Game User Settings, Set Screen Resolution, Apply Settings, then Save Settings. Fill the combo box on Construct from Get Supported Fullscreen Resolutions.
6. Press Play from the MainMenu level with Standalone Game selected so the Game Default Map and resolution changes behave as they will in a build.

Habits worth teaching on day one

- Name widgets with the WBP_ prefix and tick Is Variable only for the ones the graph touches.
- Use a Widget Switcher to flip between Main, Settings and Credits panels inside one widget instead of spawning three.
- Put reusable buttons in their own widget WBP_MenuButton with an exposed Text variable and a custom On Pressed Event Dispatcher. One style change then updates every menu.
- Set Input Mode back to Game Only every time a menu closes, or the character stops moving and students think the game broke.
- For gamepad and keyboard navigation, enable the Common UI plugin and derive the root widget from Common Activatable Widget. Show it, do not build on it in a first lesson.

In-world interactable menus

A menu that floats over a vending machine or a door panel is the same widget placed in a Widget Component on the actor, with Space set to World and Receive Hardware Input on. Point a Widget Interaction component from the character's camera at it, and clicks pass through to the buttons. This is a five-minute extension once the pause menu exists.

Suggested demo order

Run the four demos in the order below, because each one reuses what the previous one built: the rigged character becomes the enemy, the enemy becomes the soldier, and the pause menu wraps the finished scene. The whole run is about two and a half hours, so split it across two class sessions at the line after enemy AI.

Block	Minutes	Ends with	Student task after
MetaHuman and Control Rig	20	A new character running with retargeted animations	Build your own MetaHuman, retarget the jump
Blender import and IK Retargeter	15	A student's model moving with Manny's animations	Rig a simple model in Blender, bring it in
Enemy AI	35	An enemy that patrols, chases, attacks, investigates and flees	Add a hearing-only stealth enemy
Break between sessions			
Squad commander	40	A block of 24 soldiers that marches, turns and fights on right-click	Add a Wedge formation and a second unit
Menus	25	Main menu, pause menu and settings working in Standalone	Build a credits panel in a Widget Switcher
Questions and failures	15		

Two things to say out loud at the start. First, the unit demo is the enemy demo copied and given a boss, so nothing in it is new. Second, every system shown has a C++ path that goes further, and Blueprint is the right place to learn the ideas.

Sources

All pages are Epic's official documentation, current for Unreal Engine 5.8 as of 2026-10-28. Hand these to students as the reading list; each page has a Blueprint quick-start near the top.

Topic	Page
Characters	MetaHuman documentation
Characters	Control Rig
Characters	IK Rig animation retargeting
Characters	Skeletal Mesh Editor
Characters	Game Animation Sample Project
Enemy AI	Navigation System
Enemy AI	AI Perception
Enemy AI	State Tree
Enemy AI	Behavior Trees
Squads at scale	Mass Entity
Menus	Creating user interfaces with UMG and Slate
Menus	Common UI plugin
Reference video	Total War: Attila, 1212 AD mod siege (3v3), the student's example of multi-soldier units