CS354P

DR SARAH ABRAHAM

OVERVIEW: ANIMATIONS

ANIMATION PRINCIPLES

- Disney animators defined the “12 basic principles of animation” in 1981
  - Widely used and discussed in the broader context of animation

- Squash and stretch
- Anticipation
- Staging
- Straight ahead action and pose to pose
- Follow through and overlapping action
- Slow in and out
- Arc
- Secondary action
- Timing
- Exaggeration
- Solid drawing
- Appeal

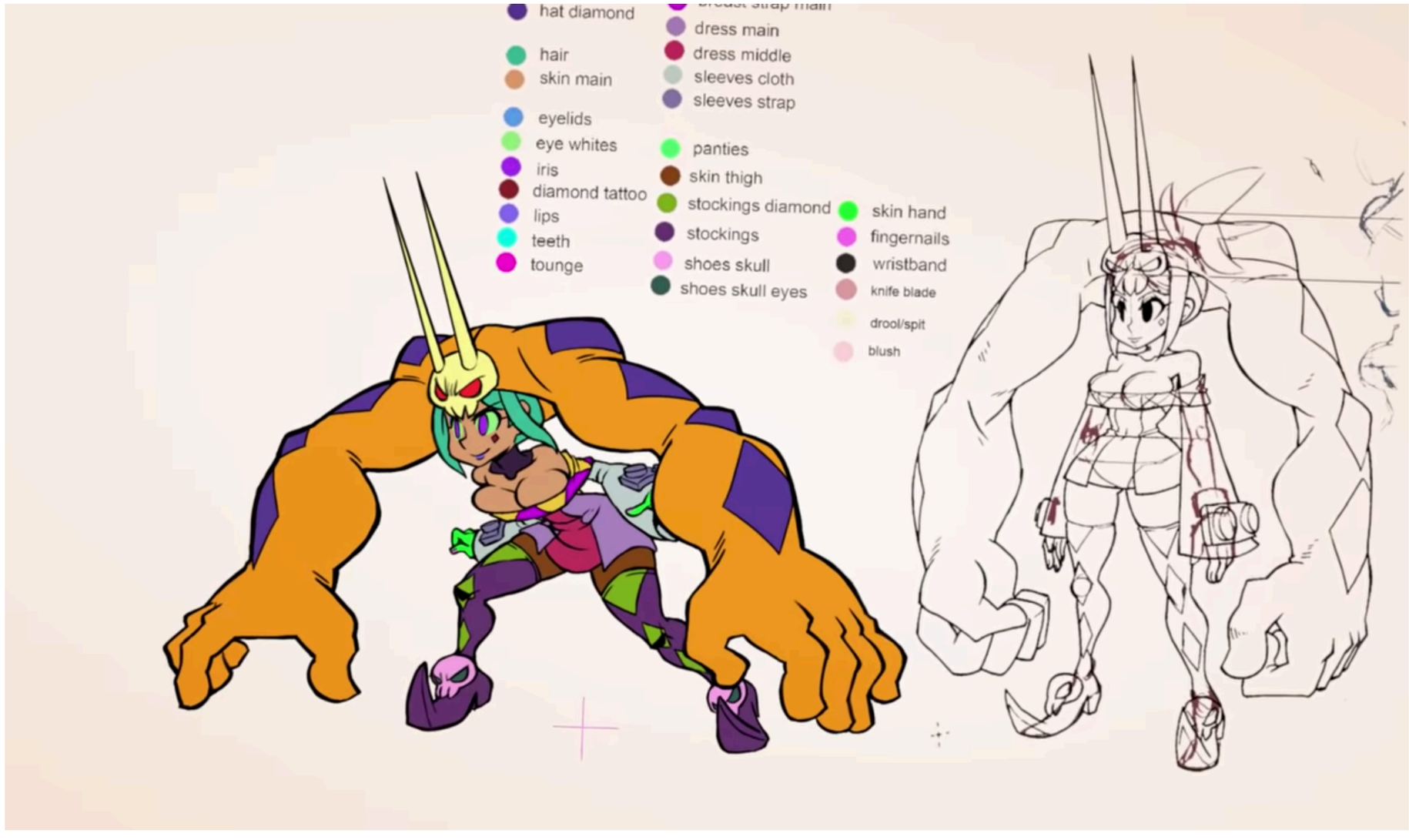
ANIMATION CONSIDERATIONS

- Artists are concerned with incorporating animation principles into their work to create impactful animations
  - Convey information to players
  - Work within the frame-timings required in an interactive system
- Programmers are concerned with animation efficiency and pipeline
  - Support animators via tools that allow efficient creation, import, and connection between animations
  - Ensure animations are packed efficiently into memory and work even in lossy environments (e.g. networking)

2D ANIMATION PROCESS

- Lead animators draw **key frames** in “pose-to-pose” animations
  - Captures dynamic actions and camera position of the shot
- Determine number of frames needed between keyframes
- More junior animators draw **in-between frames** to interpolate between shots based on the timeline
  - Process known as **in-betweening** or **tweening**

EXAMPLE: SKULLGIRLS PIPELINE



https://www.youtube.com/watch?v=5VkDXBsIXso

3D ANIMATION PROCESS

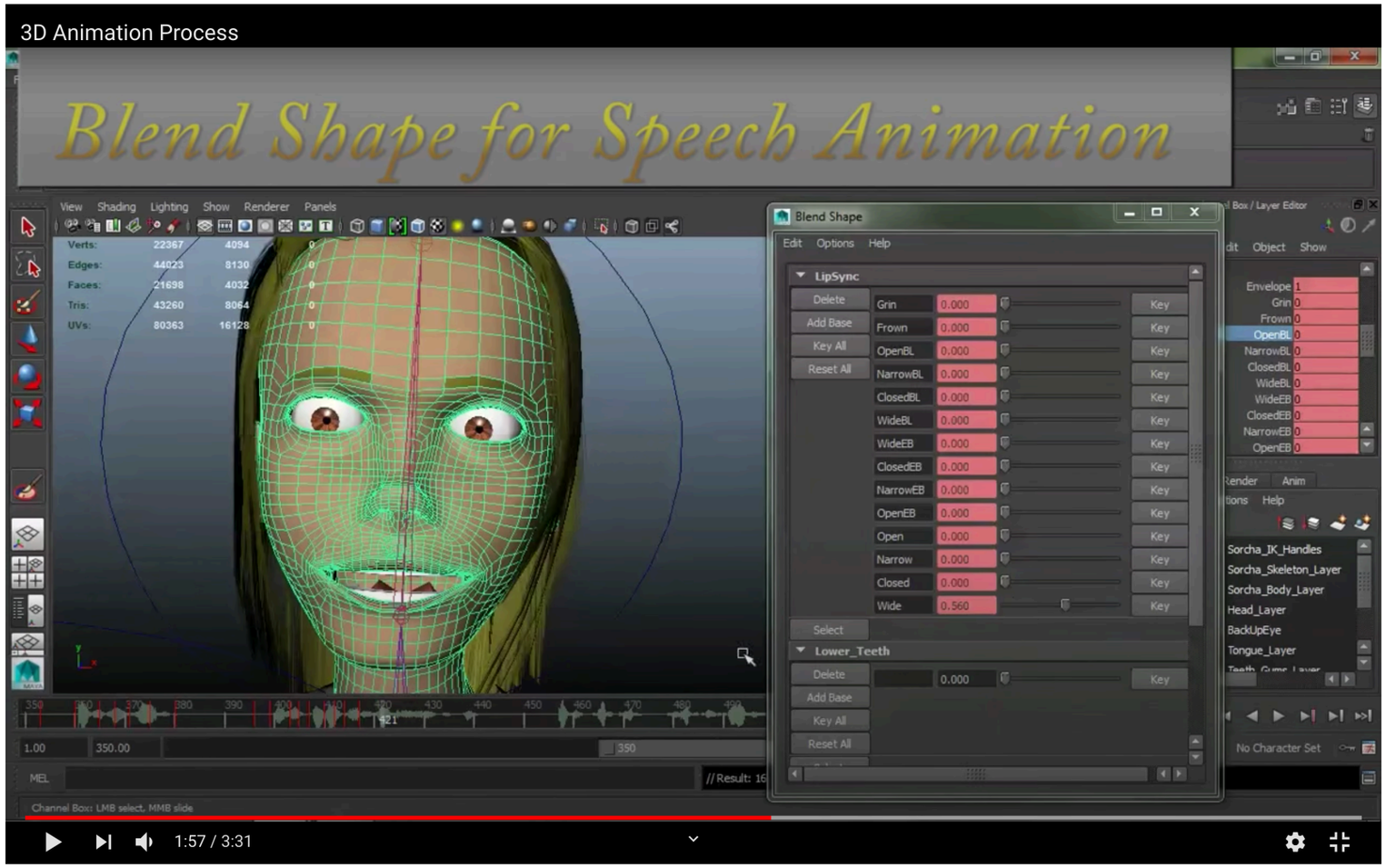
- Modeler creates 3D model composed of polygons
- Rigger connects these polygons to an underlying **skeleton** or **rig** that controls their movement
- Animator adds in **animation controls** to better constrain and control movements
- Animator creates key-framed poses and in-betweening is handled by the computer based on **splines**

EXAMPLE: PROGRESSION SHOT REEL

https://www.youtube.com/watch?v=I_nK6rozuLc

3D ANIMATION: FROM MODELING TO SCENE

- 3 minute video demonstrating the process
  - https://www.youtube.com/watch?v=tGDAeCSPGCk

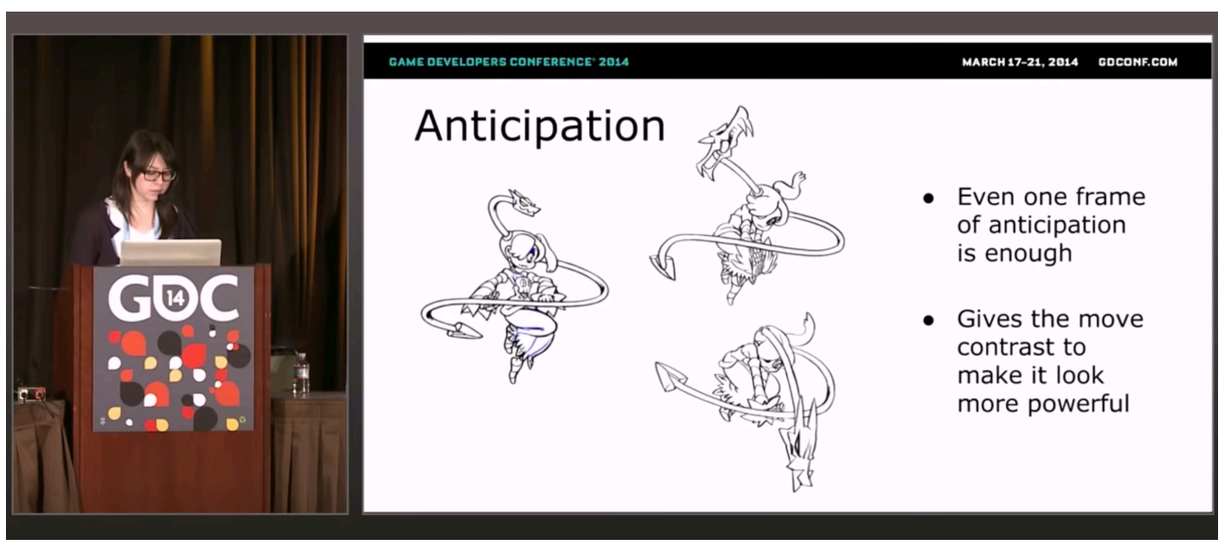


OTHER TYPES OF ANIMATIONS

- Driven animations
- Blended animations
- Motion capture animations
- Dynamic animations

SOME GDC TALKS ON ANIMATING…

- Making Fluid and Powerful Animations for Skullgirls
  - https://www.youtube.com/watch?v=Mw0h9WmBlsw
- Guilty Gear Xrd's Art Style: X Factor Between 2D and 3D
  - https://www.youtube.com/watch?v=yhGjCzxJV3E



PIPELINE INTO ENGINE

- Animating in a game engine requires:
  - A model
  - A skeleton
  - Animations
- UE5 can retarget models and animations to a skeleton, so possible to reuse skeletons between similar-shaped models
- Animations contain spline and skeleton position data based on keyframes and timings
  - Can be modified somewhat in engine but generally want these to be as close to the final product as possible pre-engine

ANIMATION STATE MACHINES

- Character animations are highly state dependent in games
  - Associated with character state, such as current actions and context (health, weapon equipped, etc)
- Animations often blended during state transitions to create smoother changes
- Use of **animation blueprints** simplifies this process in Unreal
  - Allows animator to work directly in the system without much direct assistance from a programmer

ANIMATION BLUEPRINTS

- Works on skeletal meshes
- Includes an EventGraph and an AnimGraph
- EventGraph is the usual Blueprint graph
  - Contains animation-related events to better control the animations
  - `Event Blueprint Update Animation` called every frame
- AnimGraph evaluates the final animation pose to output
  - Contains state machine functionality to determine the working pose
  - Contains blend functionality to combine multiple animations

CONNECTING ANIMATIONS TO GAME STATE

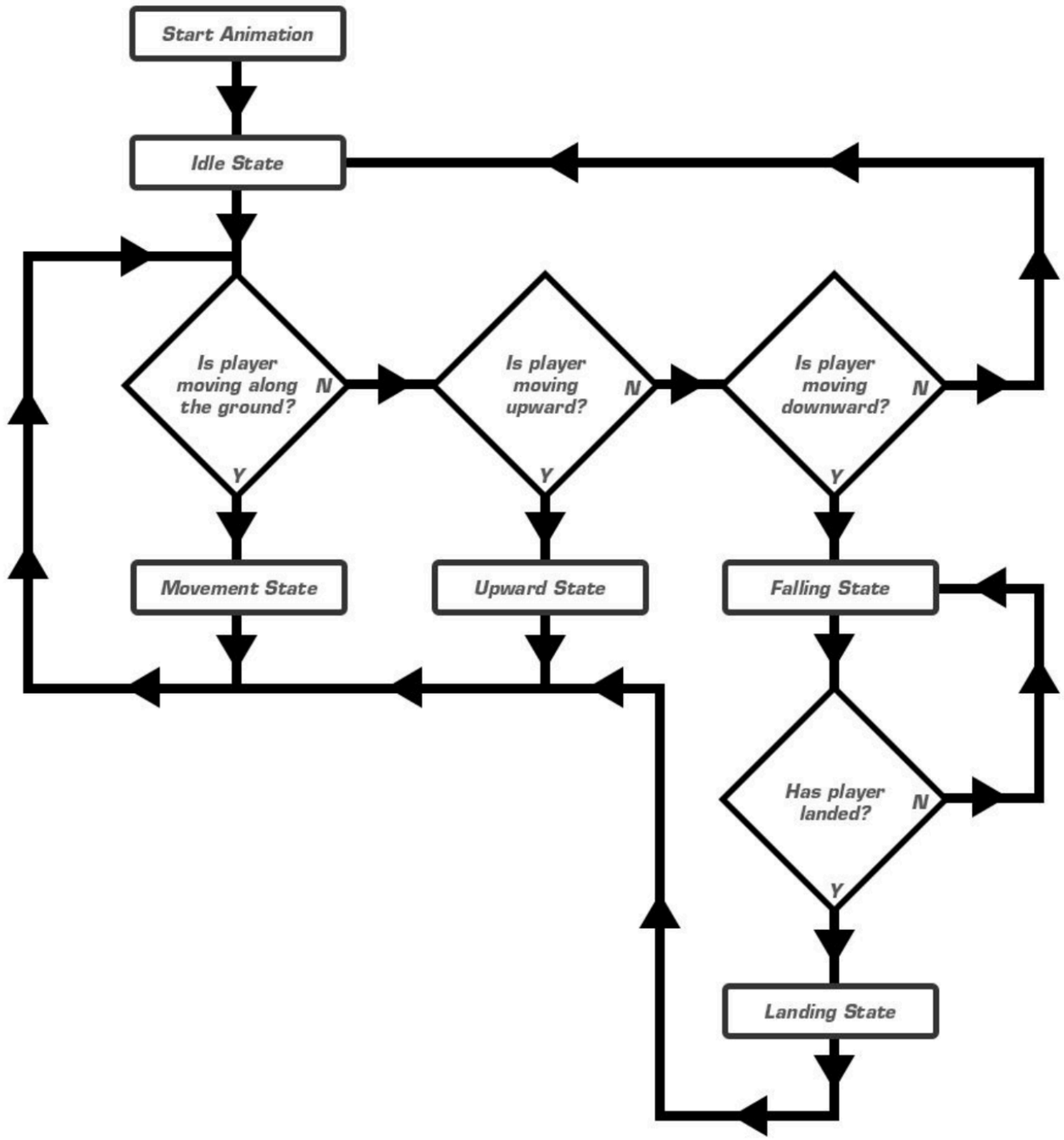
- Game state must be setup outside of the animation system by the game developers
  - No built in state machine functionality outside of animations
    - Animations should **react** to changes in state rather than *driving* them
- May need to create variables within the EventGraph for use in the state transitions
- Should be done through BP
  - C++ should only be used to inform system about game and character state

ANIMATION STATE MACHINES

- Same principle as any state machine
  - Object exists in exactly one state at any time
  - Object can transition from one state to another based on transition rules
- UE5 supports hierarchical animation states
  - A state machine can be nested within another state machine

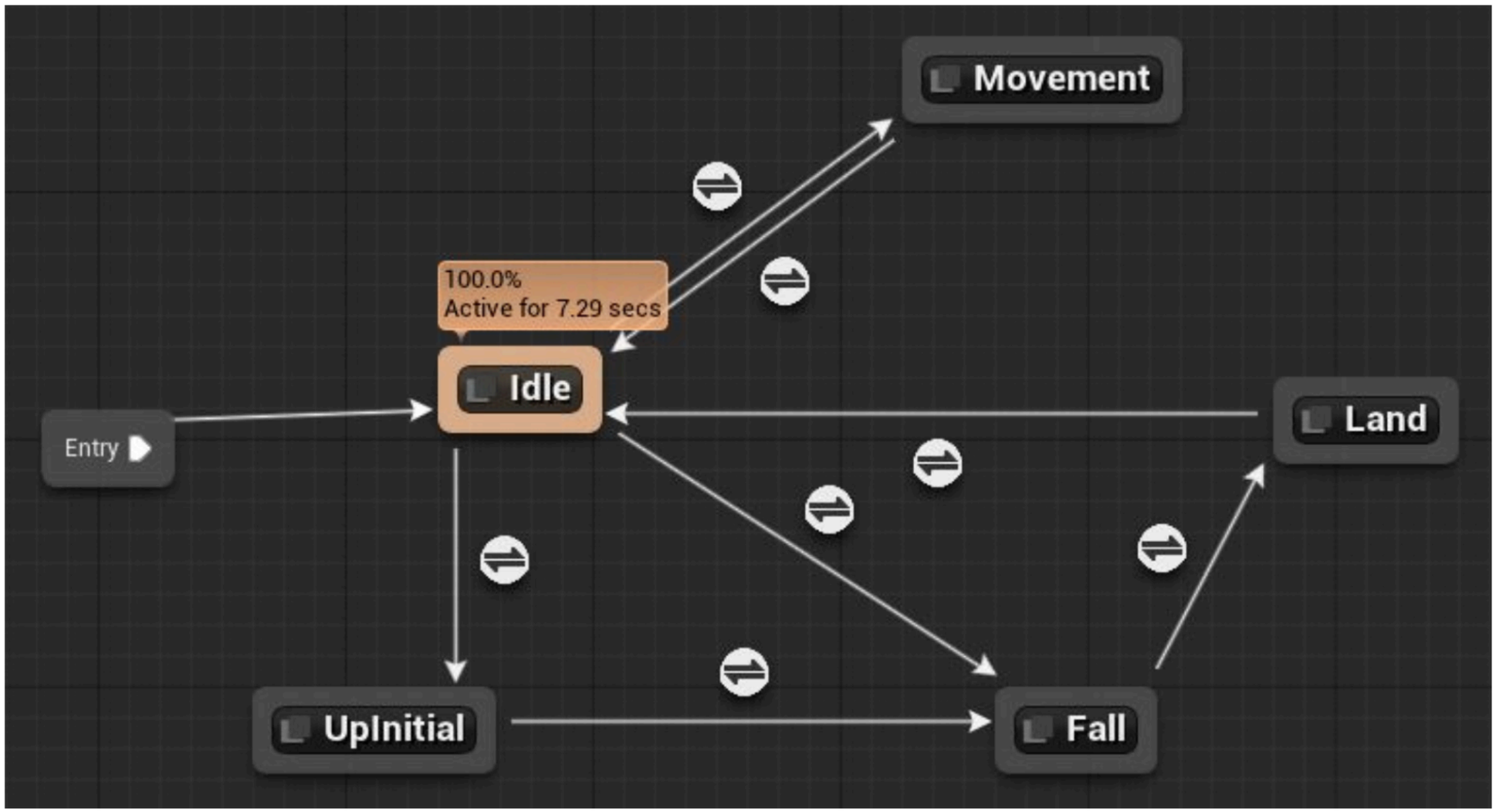
UNREAL ANIMATION STATE MACHINES

- A flow chart of a very simple state machine



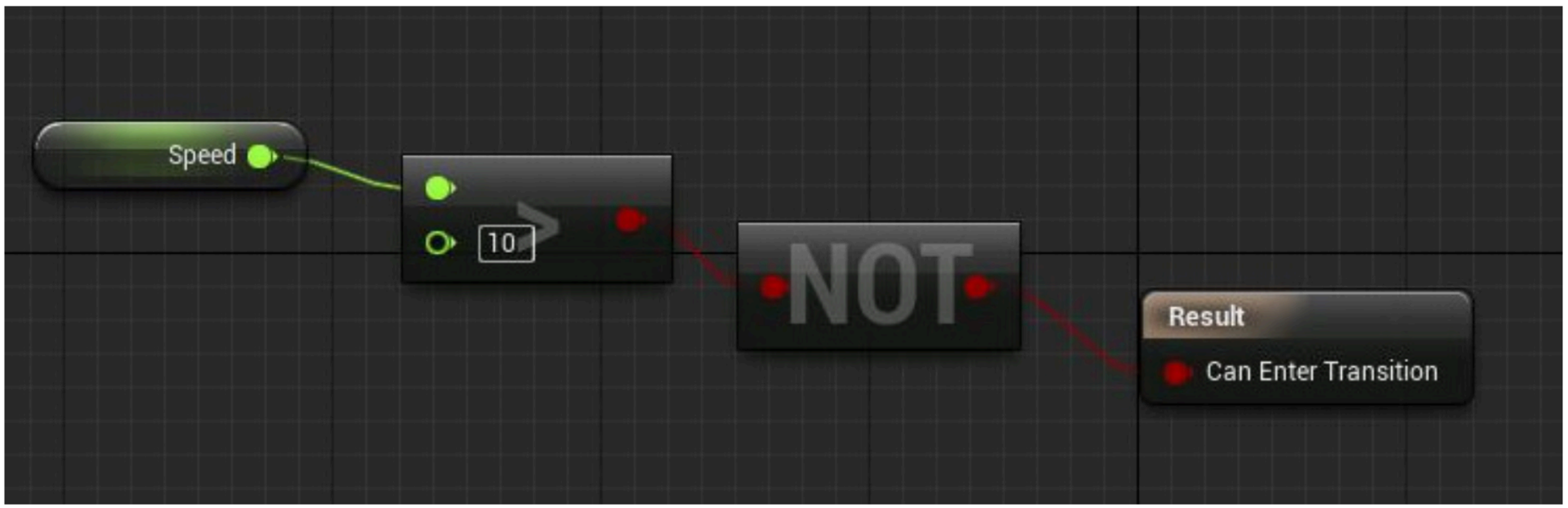
UNREAL ANIMATION STATE MACHINES

- The states and transitions in AnimGraph



TRANSITIONS

- Result is a boolean that specifies if a transition to a new state is available based on given rules
- Transition is directional and only applies from the given state



Enter a transition if the speed is not greater than 10

THINKING ABOUT GAME STATE MACHINES MORE GENERALLY

- Enums and if-else statements are sufficient in small cases but become difficult to debug in large systems
  - Note: enums still useful as handles for designers and artists to reason about the system
- Can build more complex functionality into a FSM manager
  - Use of **interfaces** to define states and transitions
  - Use of **stacks** to track previous states if it's necessary to return to those

FORWARD AND INVERSE KINEMATICS

- Kinematics is an area related to robotics and animation
  - Forward Kinematics (FK) solves the problem by manipulating the joints directly (e.g. rotate your shoulder, extend your elbow, turn your wrist)
  - Inverse Kinematics (IK) solves the inverse problem (e.g. what is the position and rotation required of my shoulder, elbow, and wrist to put my hand on this door knob?)

INVERSE KINEMATICS

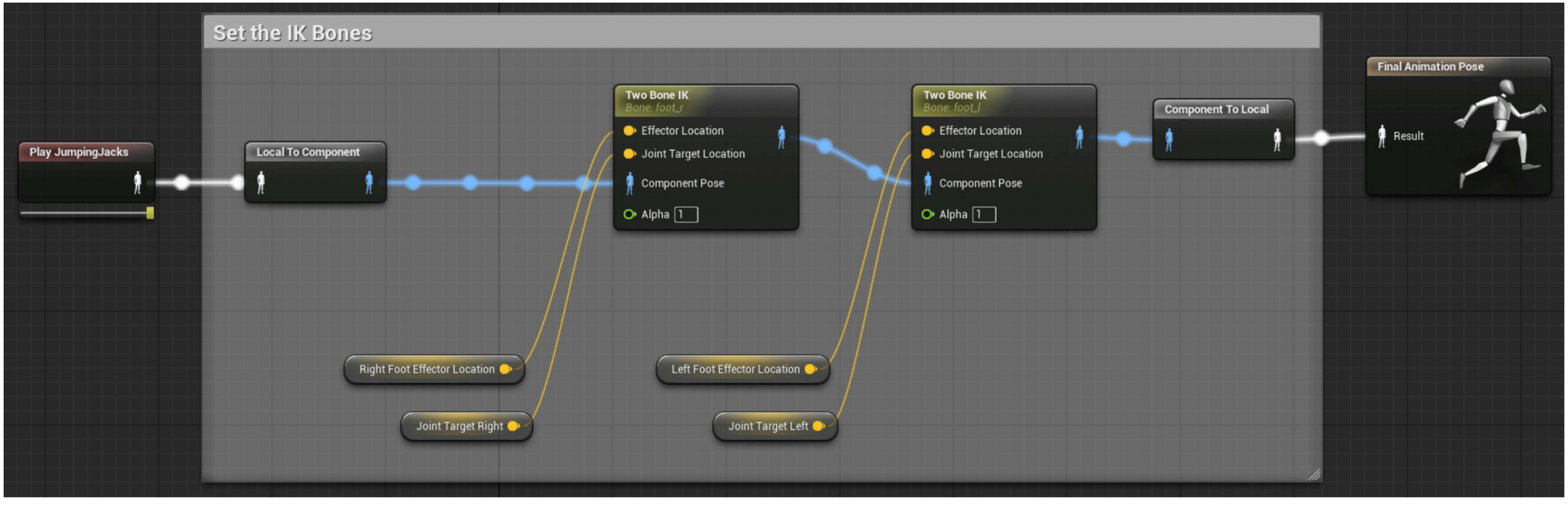
- Inverse Kinematics is preferred handling for animations
  - Easier to reason about from an animation perspective
- Inter-related joints form a **kinematic chain** (e.g. a hand, elbow, and shoulder are all related)
- Work from the end of the kinematic chain (called the **effector**) and update the joint angles and positions of the associated joints (e.g. the elbow and shoulder) to determine the final pose

UE5 AND IK

- UE5 supports IK solving in animations
  - Allows actors to more correctly interact with uneven terrain/ climbing mechanics etc
- Requires:
  - Use of traces in Actor setup to detect where to position IK effector
  - Additional variables in AnimBlueprint to connect to the IK effector position in the Actor
  - Use of 2-Bone IK nodes with the AnimBlueprint to solve for the updated pose

2-BONE IK NODE

- Local space is model's local coordinate system
- Component space is coordinate system relative to rig's root bone
- Trace determines effector locations used in 2-Bone IK



ANIMATING WITHOUT POSES

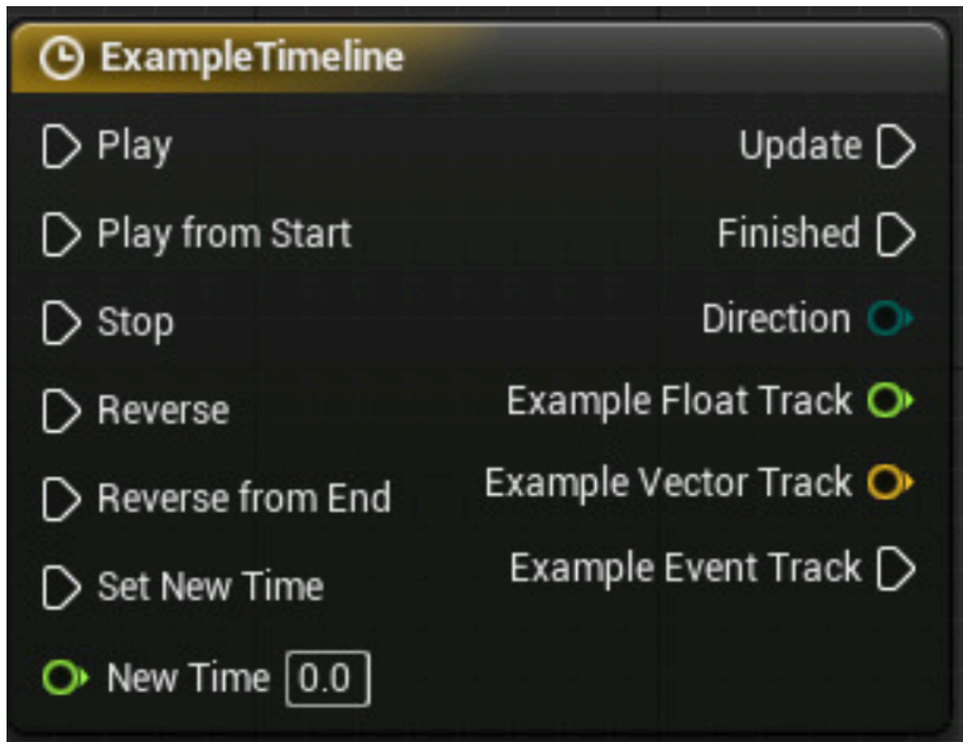
- Can create animations without animator-created poses
- Use of interpolation functions to determine a path/rate of change
- Useful on objects that require some animation but do not have the complexity of a character model
  - Opening/closing doors
  - Moving platforms
  - Flickering lights
  - etc...

UE5 TIMELINES

- **Timeline nodes** provide a basic framework for interpolating values within BP
  - Can be done via C++ but it makes working with the curves more difficult
- Exec node determines how timeline will play and what to do upon Timeline completion
- Curve determines interpolation for outgoing values
- Outgoing values passed into other nodes to change state (e.g. alpha, position, rotation, etc)

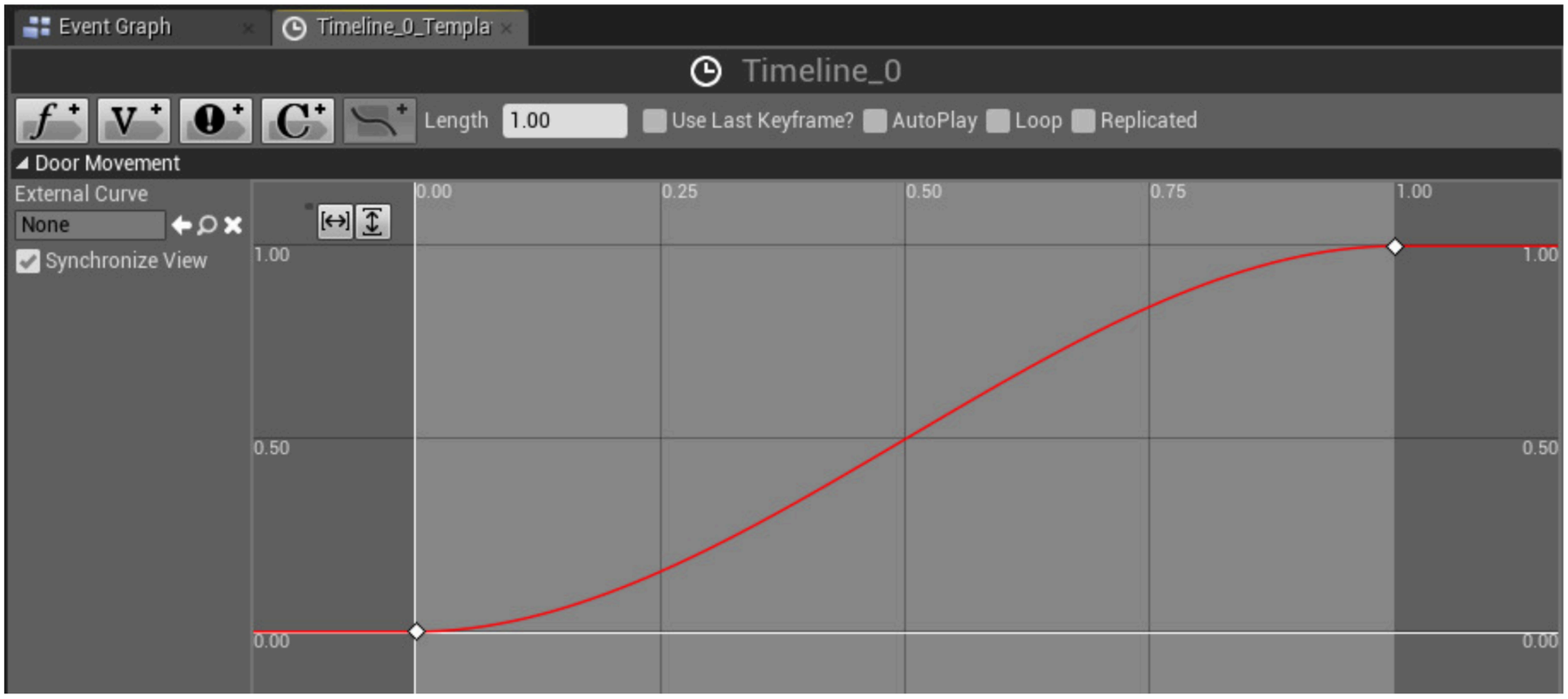
TIMELINE NODE

- Designer can determine how and when the timeline plays within the Event Graph
- Calls to Update happen based on Tick
- Call to Finished happens at end of Timeline
- Outgoing parameters are modified based on the curve



TIMELINE SETUP

- Curve controls interpolation of parameter values
- Other Timeline info can be set as well

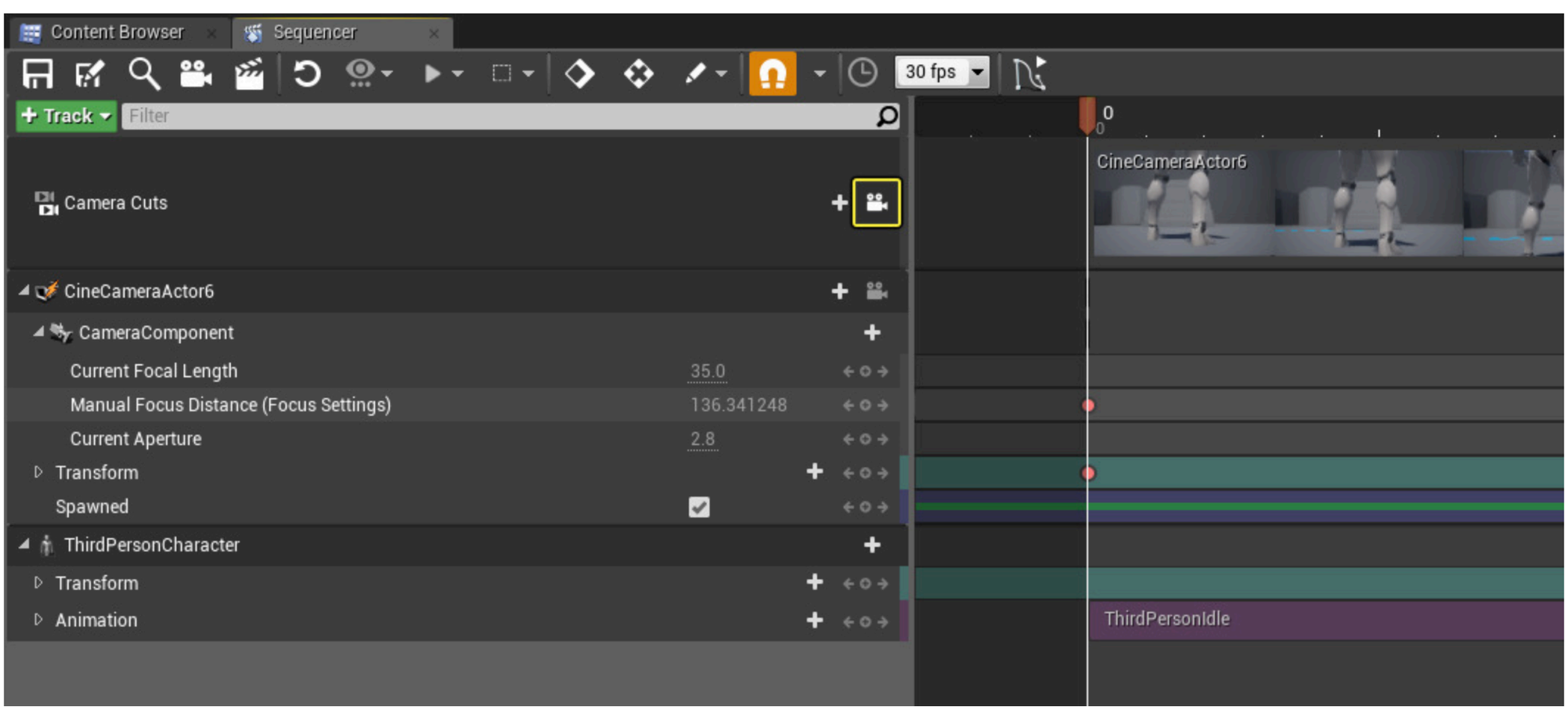


UE5 SEQUENCER

- Sequencer is the primary tool for creating in-game cinematics (i.e. scripted in-game cutscenes)
- Player may or may not be controllable during these sequences
- Useful in situations where there are complex animation interactions that are specific to a particular level or dramatic moment
  - Dialogue sequences
  - Scripted fight sequences
  - Scripted NPC behavior (i.e. we want specific behaviors not controlled by the AI system)

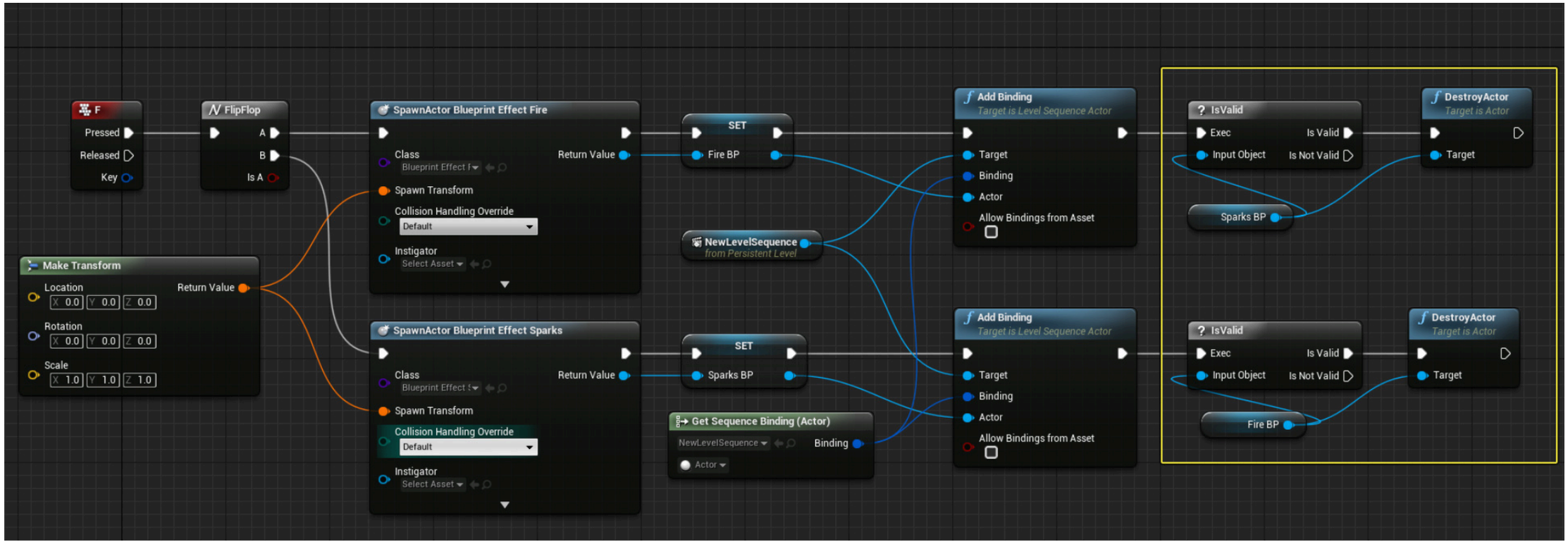
SEQUENCER EDITOR

- The sequencer editor is its own ecosystem
  - Fairly complex to master
  - Very similar to tools like Final Cut or Premiere



SEQUENCER AND BLUEPRINTS

- Cinematics can mostly be placed in levels by designers via Blueprint
- May want an underlying C++ system to simplify this process in practice though...



FURTHER READING

- Animation Blueprints [https://docs.unrealengine.com/en-US/Engine/Animation/AnimBlueprints/index.html]
- Sequencer Quick Start [https://docs.unrealengine.com/en-US/Engine/Sequencer/QuickStart/index.html]
- Sequencer and Blueprints [https://docs.unrealengine.com/en-US/Engine/Sequencer/HowTo/AnimateDynamicObjects/index.html]