Process for Implementing

Rigging

- Double check scaling and transform values of groups and joints. Everything should be 0'd out and units should conform to Unreal standards
- 2 sets of rigs
    - Deform Rig: joints weighted to the character
        - This rig ends up in Unreal, and has the animations baked down to these joints.
    - Control Rig: not weighted to the character, but has a full hierarchy with control curves, IK/FK, etc.
        - This rig does not go in engine, but helps for making animations within the animation software.
- The deform rig will be constrained to follow the control rig so the animator can move the controls and still affect the mesh/deform rig. Because the control rig has a proper hierarchy and constrains the deform joints, animation shouldn't notice a difference.
- When exporting the character, select the geometry then the deform rig and export as FBX into Unreal. For animations, only select the deform joints and export.
    - Export early and often! Make sure things are transferring properly!!
    - I heard for people rigging in Blender that it's important that the deform rig is named "armature" for Unreal, not sure if this is accurate for Maya as well but worth noting
- Scaling: Scaling in animation software doesn't transfer properly into a game engine. For ideal, proper scaling, <u>the joints on the deform rig should have a flat hierarchy</u> (ie, every joint **only** parented to the root, no joints parented to each other).
    - Scaling may not work per axis as it does in Maya, but uniform scaling should work fine
    - (Note: Previous attempts at this had the hierarchy working, but scaling wasn't being transferred properly in engine. May need more settings to be tweaked)
- Possibly see about joints for holding on to or throwing things in game
    - Unreal has "sockets" that can go under a joint, and other actors in the game can be attached or detached from those sockets. If the player holds something in their hand, uses a weapon and throws it, or has to have

collision in a certain, animated space, these things might be within sockets. Double check if we need any extra joints for these.

Animation

There are 3 types of animations to be aware of - looping, transition, and hero animations.

- Looping animations are what they sound like - animations that loop
  - Things like walk cycles, idle, or anything that you can stay in for an undetermined amount of time (such as guarding), are considered looping animations.
- Transition animations are animations that can play between looping or hero animations.
  - For example, there may be an animation when the player presses the guard button before going into the guard loop, or one for when they release before they can move again. These points between other animations would be considered transitions.
  - May not be used quite as much but can always help with immersion or some gameplay actions.
- Hero animations are what I will be calling the animations that affect gameplay the most. These would be things like dodging, guarding, attacking, throwing, etc.
  - These hero animations need much closer guidance from the other departments to judge how long things should last or when things should happen. ***Animators should check with either PM's, designers, programmers, or myself to judge appropriate timings, transitions, and frame ranges. This goes for the other animations as well, but especially notable for these!***

<u>With that explanation out of the way, here are some tips for making animations seamless and responsive for a game engine:</u>

Consider if root motion is necessary for the animation, or if it should even be used in the game at all

- Root Motion is a setting that can be turned on or off in Unreal per animation. Root motion means the animation drives the movement of the character, instead of just the values in programming. Typically used for hero animations for more complex movements, though is harder to tweak since movement is driven by animation, not just by values in game.

    - Knowing this, animations should be approved for root motion if we decide to use it. Otherwise, if we want programming to handle all movement related functionality, we should opt against root motion.
    - Note: It is possible to have values affect movement alongside root motion, but it is up to leads, designers, and programmers to say whether it should go this route, or if animations should even drive the movement at all.
- If an animation is approved for/should have root motion, animators can and should move the player around in their animation software to make the player move with them.
    - May need more guidance for how and when things move, as these animated movements will directly affect the player in game.
- If the animation should not use root motion (a good 90% of them, or 100% if we decide not to use it at all), animators should keep their animations focused to the origin point. Non-hero animations should not move the character forward or to the side in 3d space, aside from maybe a lean.
    - If the animation does move away from their original position without root motion enabled, the model will be off-center from where the game registers the player to be, making the player think attacks are or aren't hitting when they should be. Or, animations could "jump" back to where the player is when the animation finishes.

Animations should be separated by actions

- Every animation, between idle, walk, punch, block etc. should be separate animations, or otherwise separated in the animation software or FBX export. ***Transition animations should always be their own animations, separated from the animations they are transitioning in and out of!*** They will be hooked up in Unreal to play at the proper times.
    - It is possible to have multiple animations on the same file and split them up, however to make things a little easier, each of these should just be their own separate files in Unreal.

Animation tips: Always keep gameplay, transitions, and loops in mind

- Most, if not all animations should have the same or similar start and end poses to make them seamless.
    - Looping animations should have basically the same start and end pose, while transition animations should have their start and end poses close to the animations they're transitioning into and out of.

    - Hero animations that may not really loop or transition should try to be similar and start/end close to the idle pose, although these can be a lot more lenient, or even nonexistent to keep with the feel of the game.

- For hero/transition animations, keep anticipation frames short and sweet to sell the responsive feel of the game
    - When the player presses the attack button, they should feel the effect of that as close to the button press as possible. Try to let the animation cut straight to the point.
    - It's even possible to cut the anticipation frames altogether and go straight into the impact! Useful for hitstuns and faster attacks
    - Unreal also has its own way of blending animations together, so poses don't have to be 1 for 1 with the previous animation.

In Unreal

Riggers/animators should export their animations into the engine themselves, and make sure everything transfers properly

- Typically, when doing animations, the deform rig + mesh should be imported first, then the animations after so they can reference the rig.
- The rig in engine should only have the deform joints. No groups, constraints, or control joints should be in engine.
- Animations should be baked to the deform rig on export, and should load in Unreal as separate animation assets.
- Rigging (and animation) should also test that animations and scaling settings are imported and working properly

Everything afterwards would be mostly a job for me (Gabriel) to handle as the technical animator, in conjunction with other departments

Animations would go through the animation blueprint (ABP) and be set up with transitions

- Transitions in Unreal’s ABP system (not to be confused with transition animations!) are essentially like “if statements” that check if certain conditions are true or false in order to move between animations, such as if a player’s velocity is above a certain threshold, if they are in a falling state, or if any other custom booleans/variables are true or false.
    - These can get pretty complex if we want them to. You could have a different attack animation play based on whether you are in the air or not, as a simple example.

Animation montages will be made from certain animations to do more specific things with

- Montages will typically be made for hero animations, but can be made for any animation
- They can be played and called at any time in game and overwrite whatever animation is currently playing before transitioning back
- Currently, montages are called by event dispatchers on the GM1 character at the time a certain action happens, and keeps animation programming separate from gameplay programming

Animation Notifies will then be made and put onto montages and other animation assets

- Animation notifies are essentially blueprint functions that play certain effects at specified frames in an animation.
    - Because these are attached to character animations, you can access the actor that's playing the animation and call functions on that specific actor/player
    - Useful for turning on and off collisions, start and end points of an attack, or calling other audio/visual assets like sound effects or VFX
    - Animation notifies are beautiful for getting things to play at certain points in an animation and I love them for that
- Animation Notifies come in 2 forms:
    - Anim Notify - these trigger once, right when the animation hits that frame
    - Anim Notify State - these trigger multiple times - once at the start, multiple times on tick (once every frame) for however long they are set for, and once at the end.
        - All of these are overridable functions. It is possible to only use the tick function, or only the start and end, if you choose. You don't have to call and use all three of them.
- Some examples you could use them for would be like rotating the player to a point over time, turning on a trail or changing a shader/texture, having a hitbox collision be active midway through, making the player invincible for a time, or even adding SFX for footsteps every time the foot hits the ground in the animation.

Some extra things that Unreal can do with animations:

- Animations have their own blend in/out time variables, where animations will ease into or out of certain animations over a specified time, or can be set to 0 for an instant change. Good for smoothing out animations that might have different poses
- As mentioned before, the rig in Unreal can have "sockets" attached to any joints and can have virtually any asset locked to certain points on the character. If we want the player to hold or wear something, or for certain gameplay things like VFX or collisions to attach themselves to, they can use sockets to make things more consistent with the character's actions.
- Animations can use blendspaces for more advanced things, such as different walking animations depending on the speed the player is going or which direction they are walking

- Unreal has “slots” for their animations that can be used to have animations play on certain parts of the body. For example, if a player can walk while blocking, we could merge the guard animation with the walk cycle so that only the top half is guarding, while the legs are still walking like normal. Or maybe the head could look in the direction of an object, but the rest of the body is still free to move around, walk, and attack.