Whiskey Buisness/ Ragdoll

Architecture/Design Document

Table of Contents

Contents

Change History

Version: 0.1

Modifier: Gabriel Hernandez

Date: 28 / 02 / 2025

Description of Change: Module Design Document started.

1Introduction

This document delineates the architectural framework and design specifics of the ragdoll physics mechanics integrated within the AQuetzalMultiplayerCharacter class. This class is pivotal not only for managing player inputs and conventional movement but also for orchestrating complex ragdoll physics.

The ragdoll system is designed to respond to various gameplay dynamics, such as combat impacts and environmental hazards, by simulating a lifeless body that reacts according to the laws of physics.

For developers, the documentation outlines the technical implementation and integration points, enabling them to understand and contribute to the system efficiently.

Project managers will benefit from insights into the system's modularity and performance considerations, facilitating effective task assignments and resource planning.

Maintenance programmers are provided with the necessary information to ensure the system remains robust, efficient, and easy to extend or modify in response to future requirements.

2 Design Goals

The design of the ragdoll mechanics class is driven by several key objectives aimed at enhancing gameplay and ensuring technical robustness:

- **Realism and Immersion**: The primary goal is to provide a highly realistic response of the character’s body to impacts and forces within the game environment, contributing to a deeper sense of immersion for players. The implementation must convincingly handle the transition from controlled character movement to passive ragdoll states triggered by gameplay events such as explosions, falls, or being knocked out.

- **Modularity and Reusability**: The ragdoll system is designed to be modular, allowing for easy integration and reuse across various character classes and types within the game.
- **Ease of Maintenance and Extension**: The system is structured to be easily maintainable and extensible by other developers.

3 System Behavior

The ragdoll system within the AQuetzalMultiplayerCharacter class is intricately designed to simulate realistic physical behavior of the character's body in response to various in-game stimuli. This section outlines the functional behavior of the ragdoll mechanics from activation to deactivation, detailing the transitions and interactions at play.

Activation:

The ragdoll effect is triggered under specific circumstances such as severe impacts from enemy attacks, environmental hazards, or fatal falls. The transition from an animated state to ragdoll physics involves several steps:

- **Detection of Impact**: The system first detects a significant impact or health threshold breach.
- **Disabling Animations**: Upon triggering, animated control over the character's body is disabled, and the skeletal mesh's animation blueprint transitions to a passive state.
- **Enabling Physics Simulation**: The character's skeletal mesh then enables physics simulation.

Physics Simulation:

- **Collision Handling**: As the character hits the ground or other objects, collision responses are calculated to produce natural body movements and interactions based on the materials and surfaces involved.
- **Continuous Update**: The physics simulation continuously updates the position and orientation of each bone in the skeletal mesh

Recovery:

- **Condition Check**: The system periodically checks whether conditions for recovery are met, such as the absence of ongoing harmful impacts or player input signaling an attempt to stand up.
- **Deactivation of Physics**: Once recovery conditions are satisfied, physics simulation on the skeletal mesh is disabled, and control is handed back to the game's animation system.

- **Animation Blending**: The character transitions out of the ragdoll state through a blending process that smoothly integrates pre-defined "get up" animations. These animations are chosen based on the character’s final ragdoll position (e.g., lying face up or face down).

System Integration:

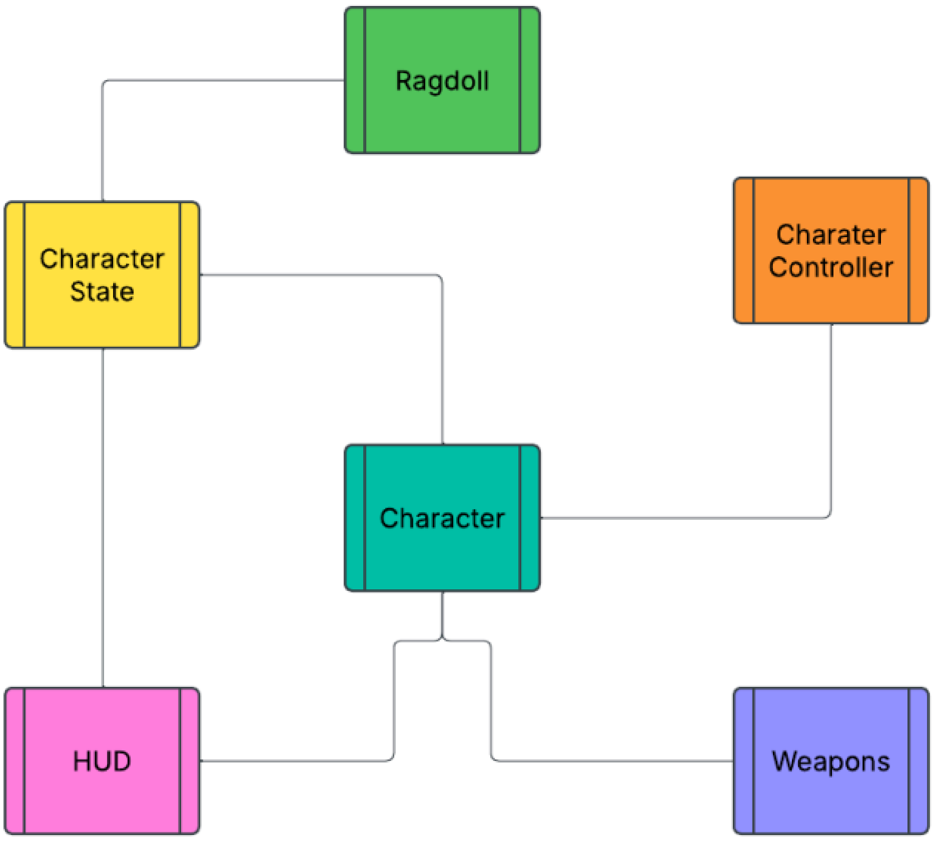
- **Feedback Loops**: Throughout the ragdoll state, the system provides feedback to other game systems, such as the health management and player control systems, ensuring that gameplay mechanics like damage calculation and movement control are accurately maintained in accordance with the character's state.
- **Event Handling**: The system is capable of handling multiple triggers and transitions between different states, allowing for complex sequences of interactions, such as being knocked down repeatedly or transitioning between different types of impacts.

Error Handling and Constraints:

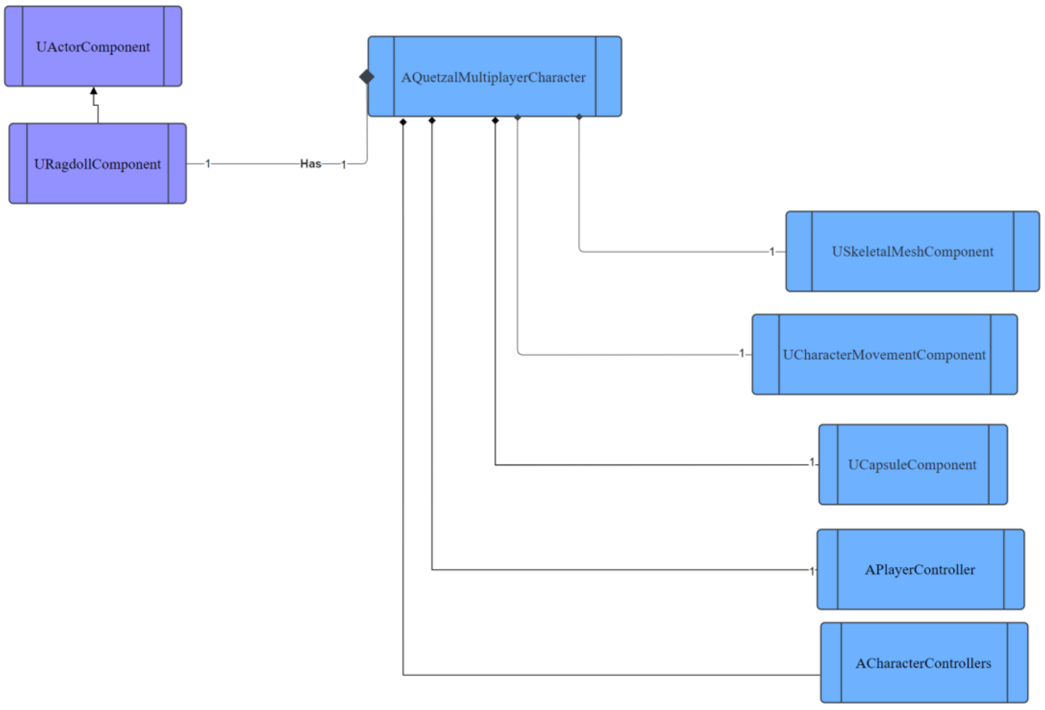
- **Stability Measures**: To maintain stability and performance, the system implements constraints on joint movements and collision responses, preventing unnatural bending or breaking of the character model.
- **Error Recovery**: In cases where the simulation encounters errors, such as unexpected changes or loss of collision data, the system has mechanisms to reset the character’s position or adjust parameters to regain stability.

4 Logical View

4.1 High-Level Design



4.2 Mid-Level Design of the Ragdoll



High-Level Design of varying systems in the game.

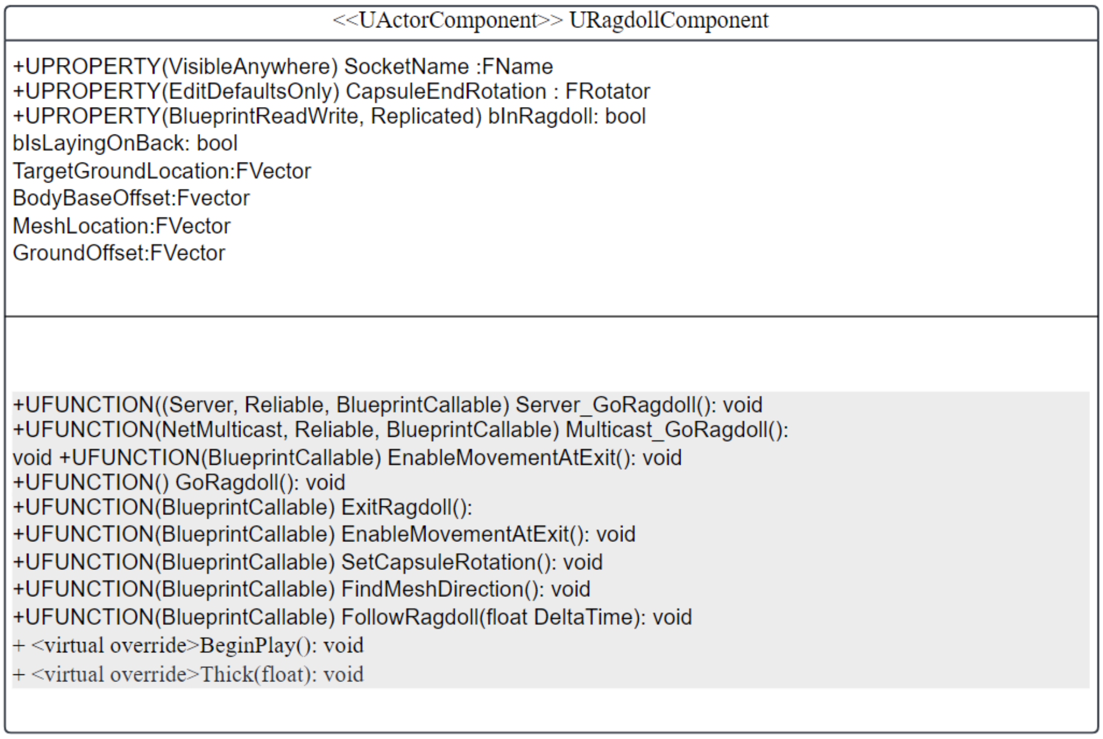
The Character Controller will handle the input from the user that will control the player. The HUD will keep the players informed of the current health and score of the players using values from Character and Character State.

Weapons will interact with the player by activating equipped weapons or receiving damage from weapons.

Depending on the Character State the player can enter a Ragdoll state.

At the center is the Character interacting with all systems in some varying ways.

4.3 Detailed Class Design of the Ragdoll



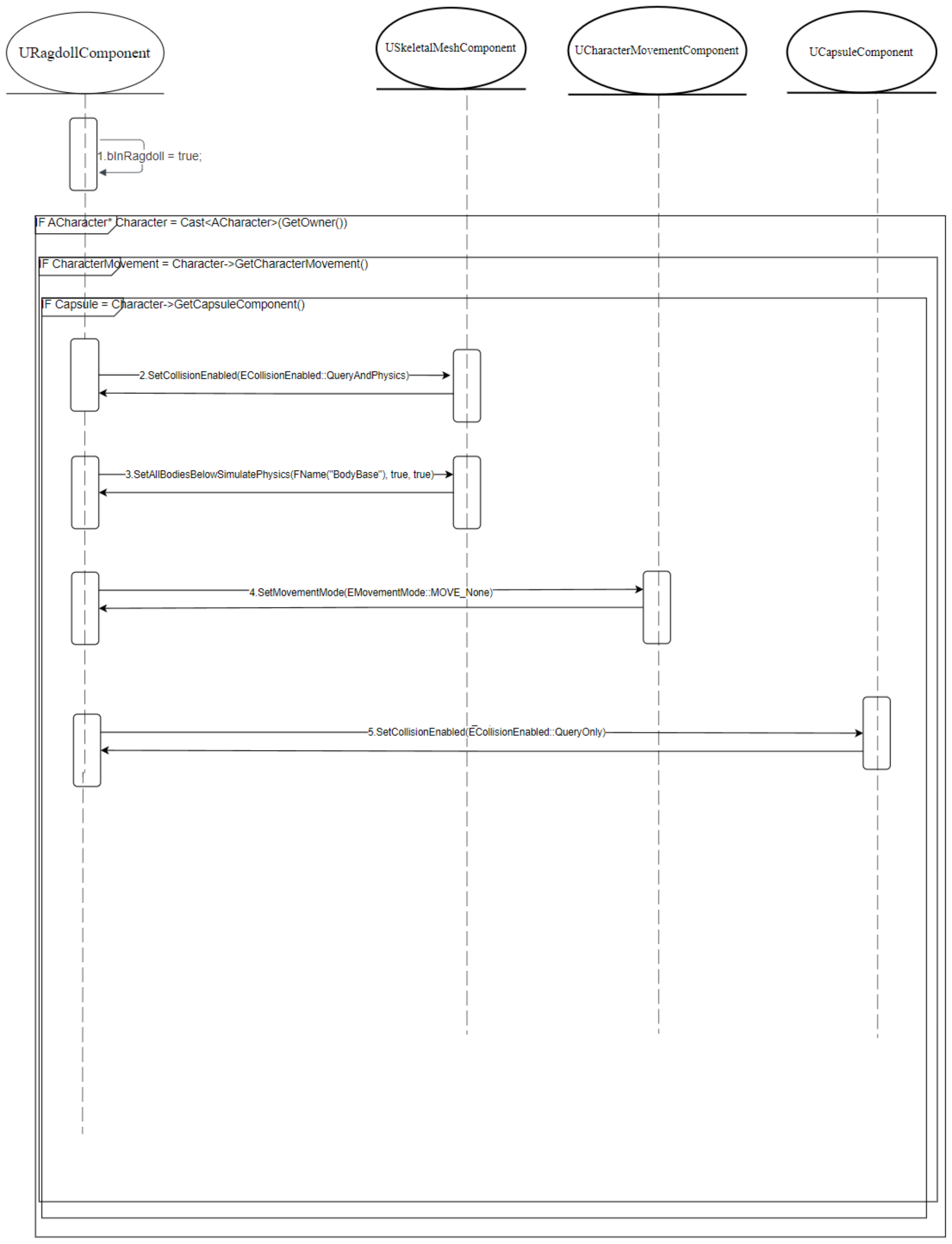
```
<<UActorComponent>> URagdollComponent

+UPROPERTY(VisibleAnywhere) SocketName :FName
+UPROPERTY(EditDefaultsOnly) CapsuleEndRotation : FRotator
+UPROPERTY(BlueprintReadWrite, Replicated) bInRagdoll: bool
bIsLayingOnBack: bool
TargetGroundLocation:FVector
BodyBaseOffset:Fvector
MeshLocation:FVector
GroundOffset:FVector

+UFUNCTION((Server, Reliable, BlueprintCallable) Server_GoRagdoll(): void
+UFUNCTION(NetMulticast, Reliable, BlueprintCallable) Multicast_GoRagdoll():
void +UFUNCTION(BlueprintCallable) EnableMovementAtExit(): void
+UFUNCTION() GoRagdoll(): void
+UFUNCTION(BlueprintCallable) ExitRagdoll():
+UFUNCTION(BlueprintCallable) EnableMovementAtExit(): void
+UFUNCTION(BlueprintCallable) SetCapsuleRotation(): void
+UFUNCTION(BlueprintCallable) FindMeshDirection(): void
+UFUNCTION(BlueprintCallable) FollowRagdoll(float DeltaTime): void
+ <virtual override>BeginPlay(): void
+ <virtual override>Thick(float): void
```

5 Process View of the Ragdoll

StartRagdoll


URagdollComponent
USkeletalMeshComponent
UCharacterMovementComponent
UCapsuleComponent
1.bInRagdoll = true;
IF ACharacter* Character = Cast<ACharacter>(GetOwner())
IF CharacterMovement = Character->GetCharacterMovement()
IF Capsule = Character->GetCapsuleComponent()
2.SetCollisionEnabled(ECollisionEnabled::QueryAndPhysics)
3.SetAllBodiesBelowSimulatePhysics(FName("BodyBase"), true, true)
4.SetMovementMode(EMovementMode::MOVE_None)
5.SetCollisionEnabled(ECollisionEnabled::QueryOnly)

The StartRagdollCPP() method transitions from a controlled state to a ragdoll state where physics take over, simulating realistic responses to impacts or falls.

Process Steps:

1. **Disable Character Movement**:

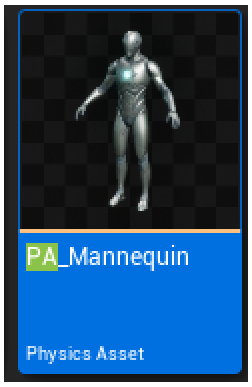
    - Movement is halted to prevent influence from ongoing player inputs, with the movement mode set to MOVE_None.

2. **Disable Capsule Collision**:
    - The capsule collider is disabled to avoid interference with the ragdoll physics, allowing the skeletal mesh to react freely to environmental collisions.
3. **Configure Skeletal Mesh for Physics**:
    - The skeletal mesh is detached and prepared for physics interactions:
        - Set as a ECC_PhysicsBody to participate in physics simulations.
        - Physics and collision are enabled to interact with the game world.
        - Physics simulation is activated from a specific bone downwards, fully controlling the mesh's movement by physics.
4. **Reattach Skeletal Mesh**:
    - After setup, the skeletal mesh is reattached to the capsule component to maintain its association with the character's main component.
5. **Update State Variable**:
    - The state variable isRagdollingCPP is set to true, marking the character's transition into the ragdoll state.

6 Set Up Guide

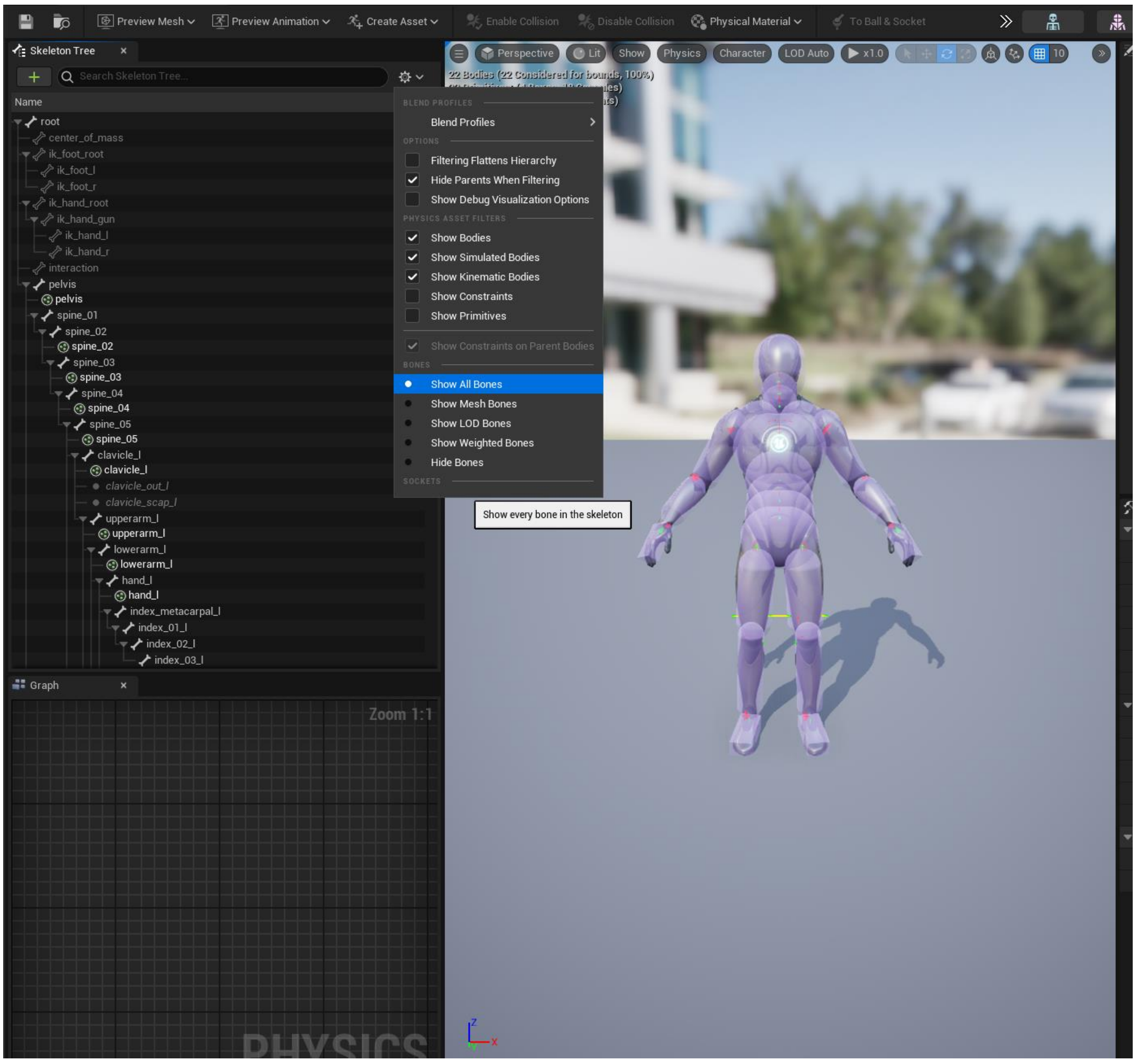
Physical Asset SetUP

Locate the Physical Asset of the character



Double click it and get into the Physics view

Change the settings to Show all bones so you can see the root and the skeletal mesh attached to the PA



BLEND PROFILES
Blend Profiles

OPTIONS
- [ ] Filtering Flattens Hierarchy
- [x] Hide Parents When Filtering
- [ ] Show Debug Visualization Options

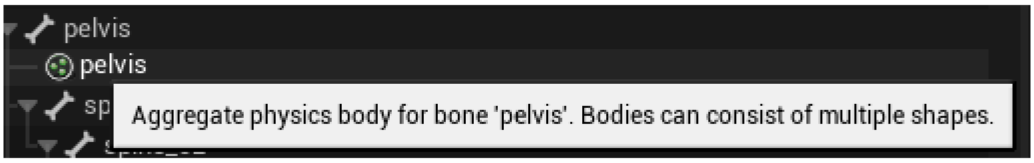
PHYSICS ASSET FILTERS
- [x] Show Bodies
- [x] Show Simulated Bodies
- [x] Show Kinematic Bodies
- [ ] Show Constraints
- [ ] Show Primitives
- [x] Show Constraints on Parent Bodies

BONES
- ● Show All Bones
- Show Mesh Bones
- Show LOD Bones
- Show Weighted Bones
- Hide Bones

SOCKETS

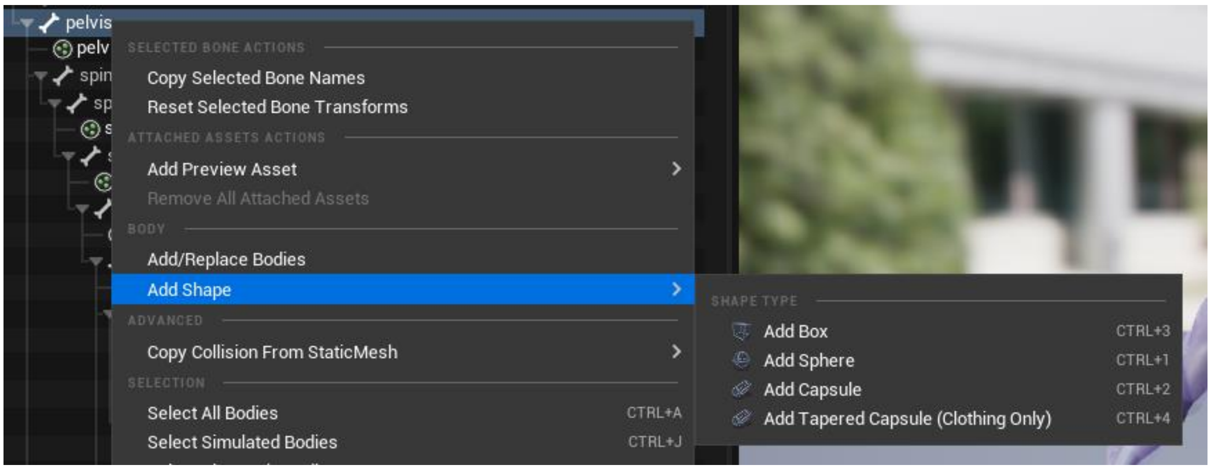
The skeleton tree is divided into

Bones(From the skeletal mesh), Shapes(the physical assets) and constraints (connections between the PA/articulations)



Aggregate physics body for bone 'pelvis'. Bodies can consist of multiple shapes.

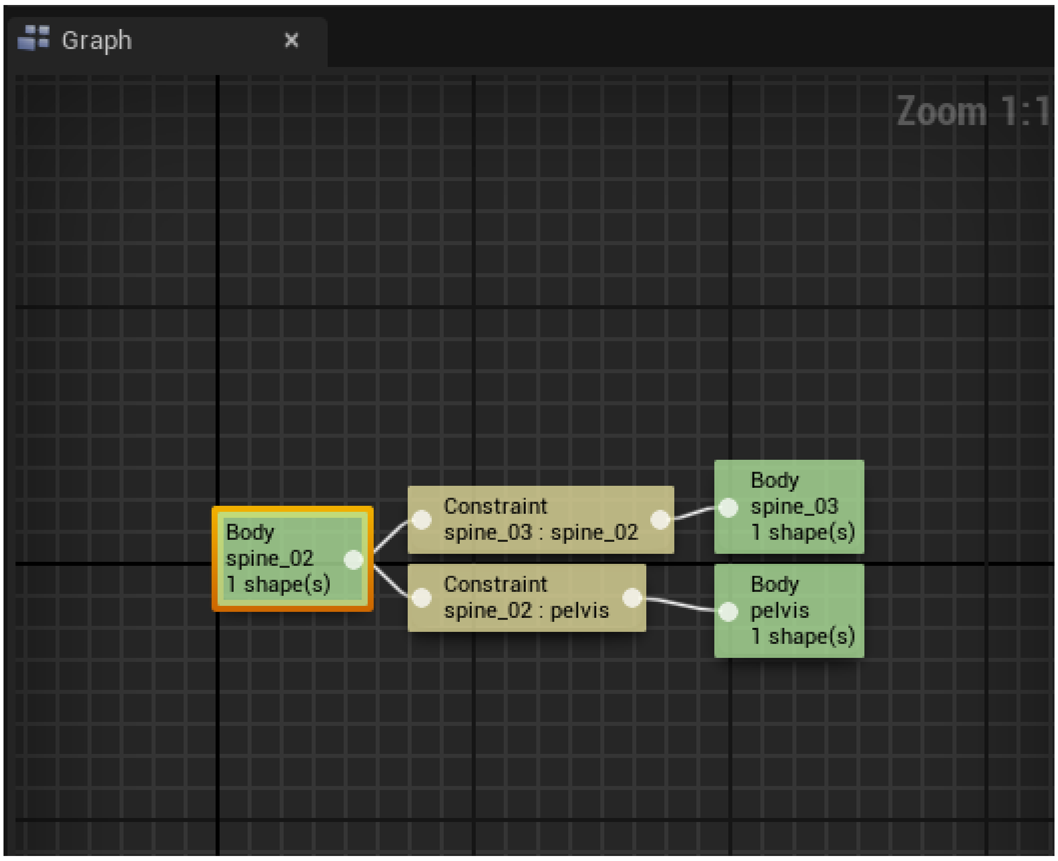
For all the bones you think they need a PA Right click the bone and add a shape



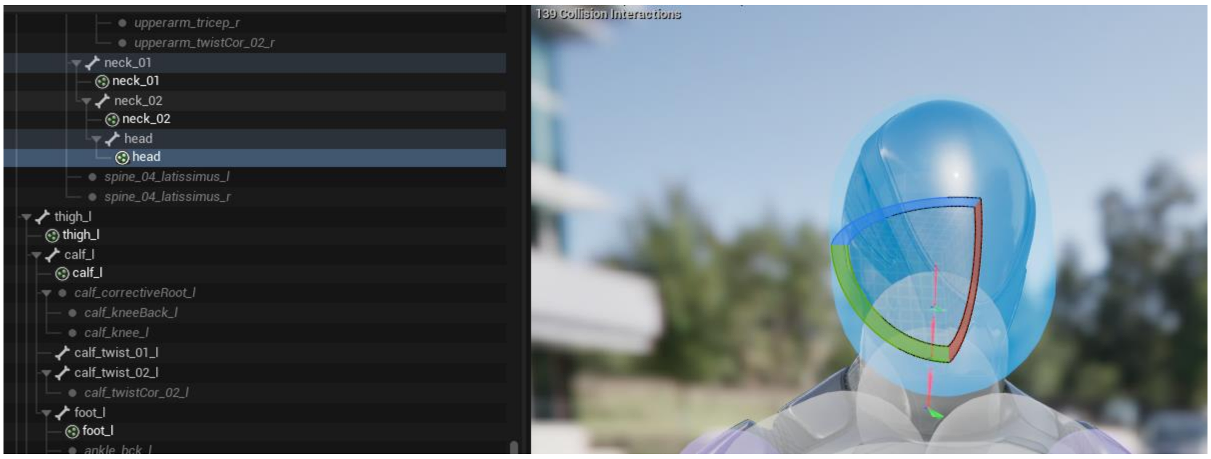
You need to edit the Size and rotaton of each PA,

You can see those bones and constraints

Also you can see them in the physics graph

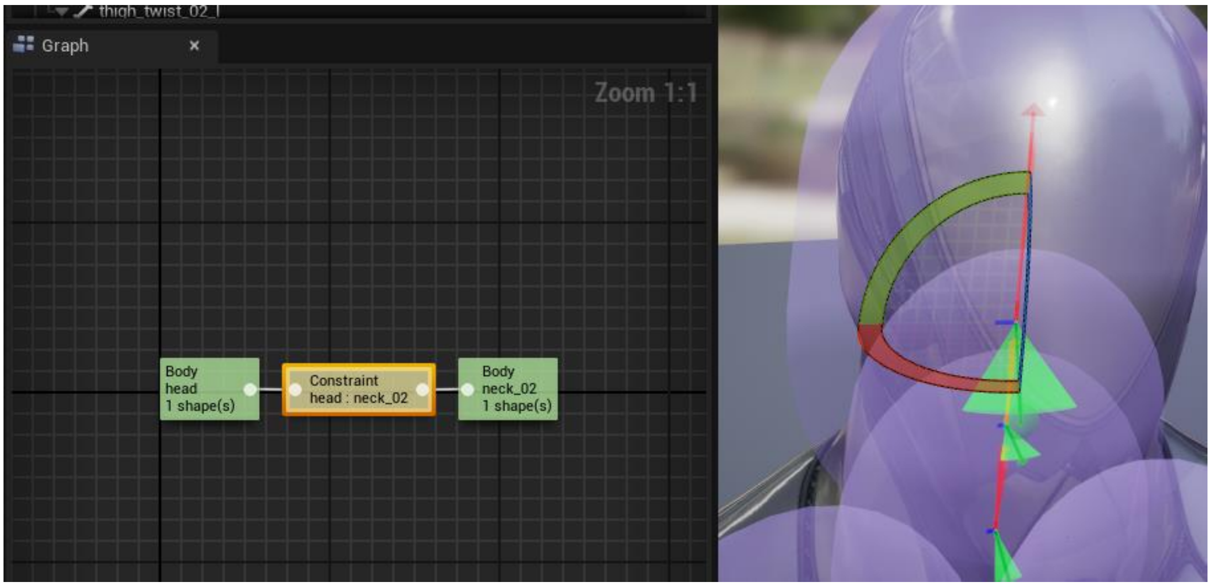


Select Bodies: Click on different colliders in the viewport to select and view corresponding bones.

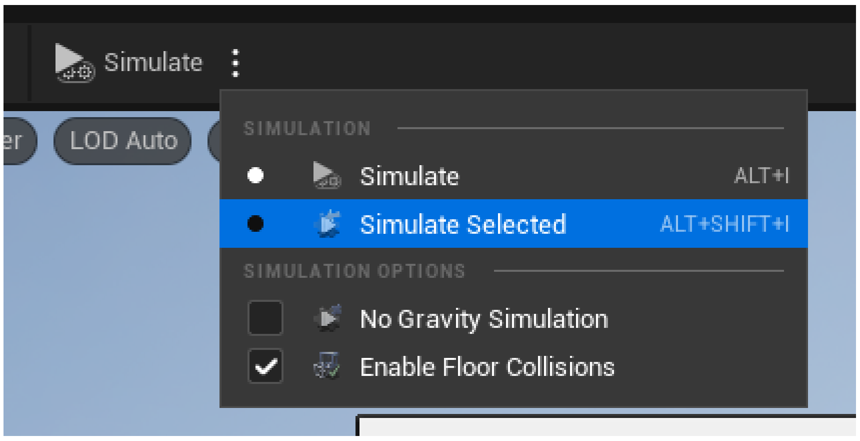


Select Constraint: click on the corresponding node or the arrows that show the constraints

Constraints limit how far each bone can rotate or twist.



Test Simulation: Use the Simulate button to see how the character collapses under physics.

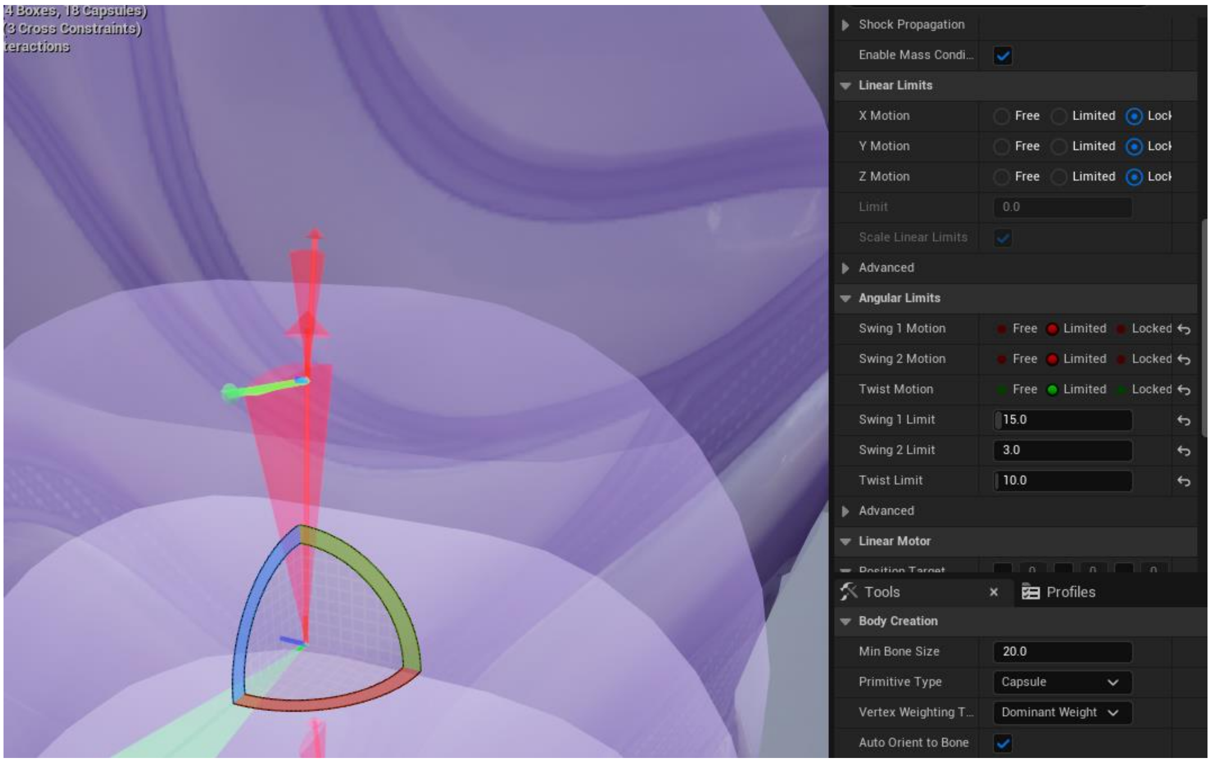


You can simulate all the body or just a group of them

Physics Setup

Constraints are important because they determine how each body part will react with the others.

For that we need to edit the Angular Limits



Shock Propagation	
Enable Mass Condi...	✓
Linear Limits	
X Motion	Free Limited [✓ Lock]
Y Motion	Free Limited [✓ Lock]
Z Motion	Free Limited [✓ Lock]
Limit	0.0
Scale Linear Limits	✓
Advanced	
Angular Limits	
Swing 1 Motion	Free [✓ Limited] Locked
Swing 2 Motion	Free [✓ Limited] Locked
Twist Motion	Free [✓ Limited] Locked
Swing 1 Limit	15.0
Swing 2 Limit	3.0
Twist Limit	10.0
Advanced	
Linear Motor	

Tools × Profiles

Body Creation	
Min Bone Size	20.0
Primitive Type	Capsule
Vertex Weighting T...	Dominant Weight
Auto Orient to Bone	✓

We have Swing 1 Motion, Swing 2 motion and Twist.

Swing 1 and Swing 2: These limits specifically manage the arm's movement within vertical and horizontal planes.

Swing 1 handles the forward and backward motions, akin to swinging the arm towards or away from the body.

Swing 2 controls the lateral movements, allowing the arm to swing side to side across the body. Setting these parameters accurately ensures that the arm’s movements remain within natural human limits, thus preventing any awkward or exaggerated swings.

Twist:
This limit is focused on the rotation’s movement around the arm's longitudinal axis. It regulates how much the arm can twist.

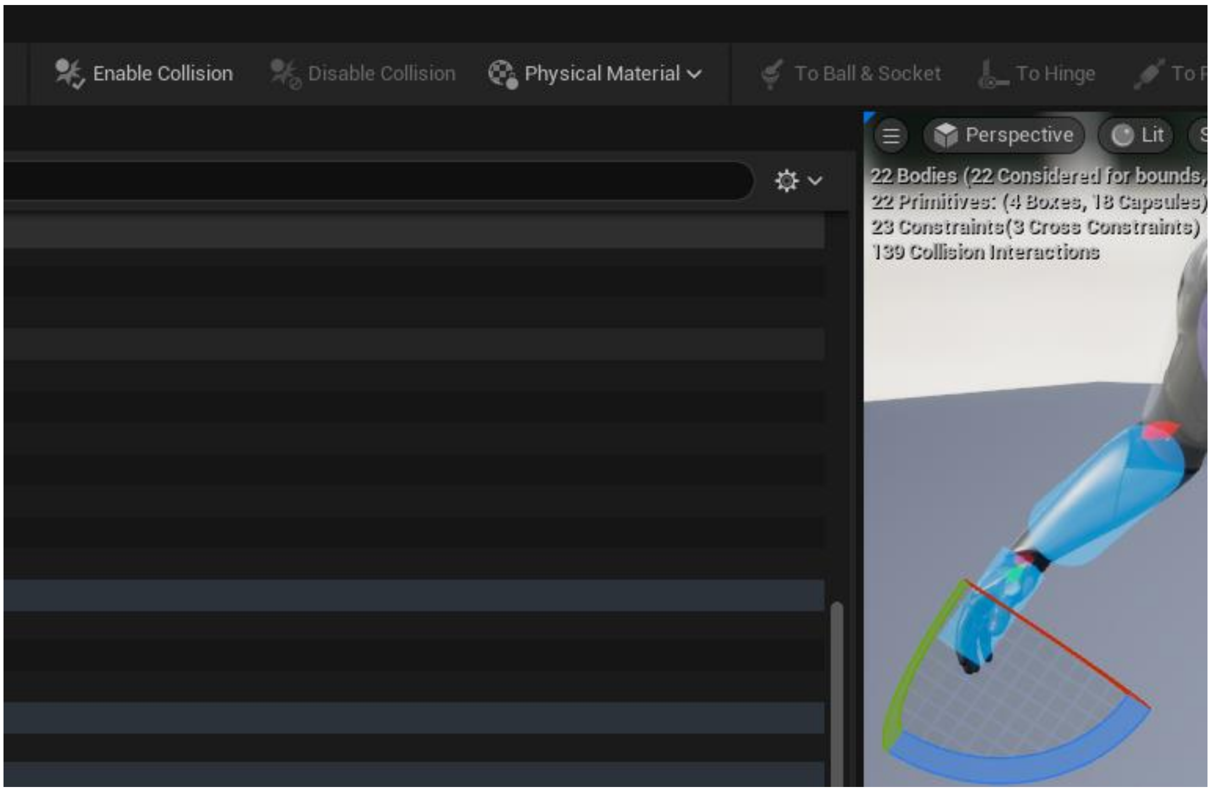
Linear Limits control movement along an axis, allowing settings for free, limited, or locked motion.

Angular Limits manage joint rotation via Twist and Swing parameters.

Drive Settings apply spring-like forces to maintain alignment of physics bodies.

Before starting to edit the Constraints we need to make sure that exist a collision Between others

You need to select 2 shapes and enable collision to them



22 Bodies (22 Considered for bounds,
22 Primitives: (4 Boxes, 18 Capsules)
23 Constraints(3 Cross Constraints)
139 Collision Interactions

Not all of them need a collision enable between.

Its important to have a Root bone, is crucial in ragdoll physics because it retains velocity information when the character's capsule may not move, serving as a reliable reference for movement and animation transitions in Unreal Engine, where actor velocity isn't accessible during physics simulations.

Constraints Setup

Use the constraints of the PA Mannequin that its already edited to work properly