GAMEPLAY MECHANICS DOCUMENT
HERCULES UNTOLD
LEVEL: TAKING THE PATH

Table of contents.

1. Introduction.....
2. Core Gameplay Systems
   - 2.1 Hercules Character & Input
   - 2.2 Combat System
   - 2.3 Dodge & Weapon Equip
   - 2.4 Healing & Stamina
3. AI Systems
   - 3.1 Enemy AI
   - 3.2 Animal AI
4. Items & Interactable
   - 4.1 Weapons
   - 4.2 Treasure & Currency
   - 4.3 XP Orbs
5. Attributes & UI Feedback
   - 5.1 Attribute Component
   - 5.2 Interfaces (Hit & Pickup)
   - 5.3 HUD and Overlay
6. Environment Systems
   - 6.1 Rock Spawner & Hazards
   - 6.2 Breakable Objects
7. Animation Systems
   - 7.1 AnimInstance
   - 7.2 Combo AnimNotify
   - 7.3 Enemy Animations
8. Game Mode & HUD
9. Levels
10. External tools & Other Unreal Engine Processes
11. Critical Review
12. Gameplay Systems & Evaluation Summary.

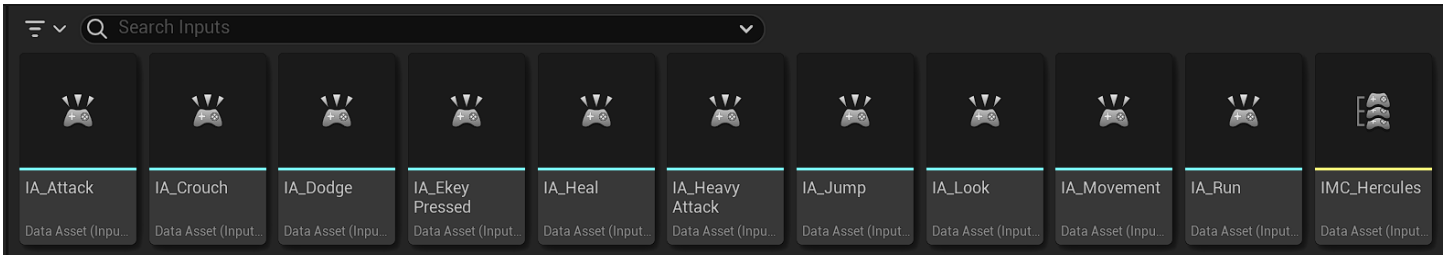
1. Introduction

Hercules Untold is a mythological action-adventure game where players control Hercules through a richly designed world filled with enemies, puzzles (in future), ambient life, and ancient mysteries. This document outlines the complete gameplay mechanic systems built for the project, developed using Unreal Engine 5.4 with C++ and Blueprint integration. From sword combat and stamina-based movement to destructible environments, dynamic enemy AI, and a modular UI system, this document provides a full breakdown of how the game works internally with the goal of enabling maintainability, future feature integration. It contains combat and stamina-based movement to destructible environments, dynamic enemy AI, and a modular UI system, this document provides a full breakdown of how the game works internally with the goal of enabling maintainability, future feature integration, and team scalability.

2. Core Gameplay Systems

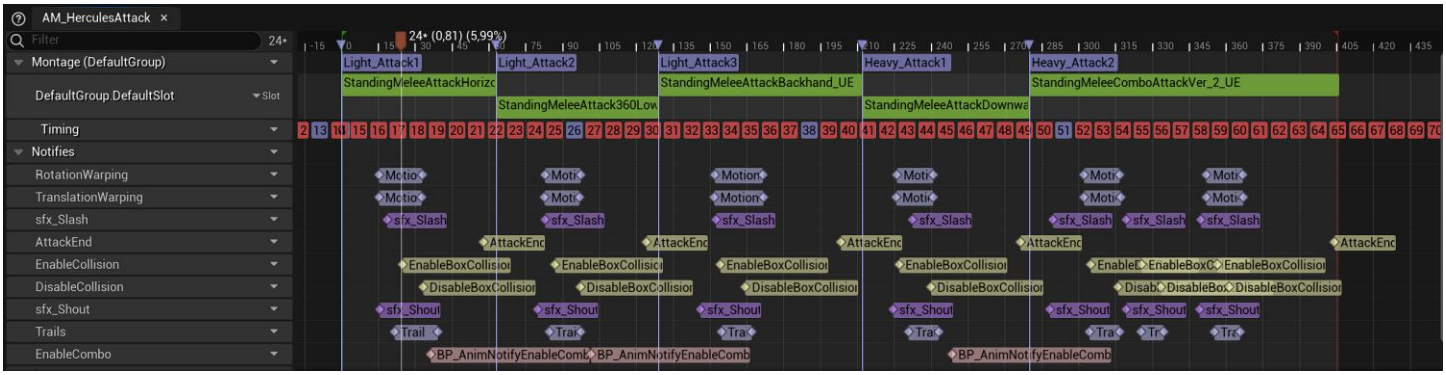
2.1 Hercules Character & Input

The core player class AHerculesCharacter handles character movement, interaction, combat input, and game logic. The class has it owns functions, or it inherits function form the ABaseCharacter class (base Class for Player and Enemies) override them or not. Input is bound via the Enhanced Input System and mapped to actions such as:
Movement: Forward (W) / Backward (S), Left (A) / Right (D)
Camera Look: Mouse strafe
Actions: Run (shift + movement), Jump (Spacebar), Run Attack (swift + Left mouse button), Light Attack (Left mouse button), Heavy Attack (Right mouse Button), Dodge (Middle mouse Button) Dodge Away (swift + Middle mouse Button), Equip/unequip Weapon (E), Healing (1)
Movement respects current action and stamina states and for applying all the necessary values CharacterMovementComponent used.

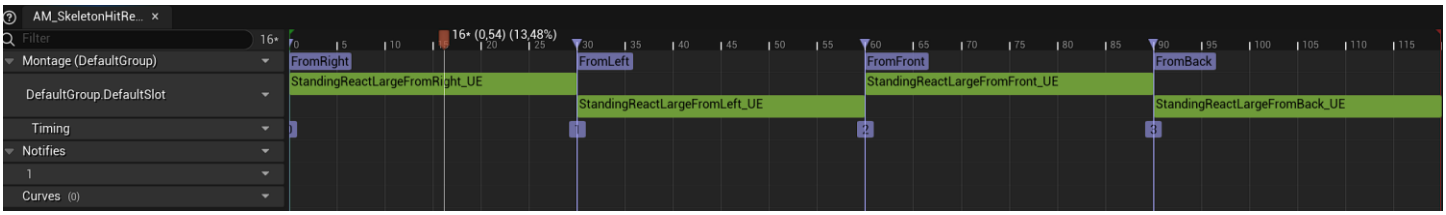


2.2 Combat System

The player use states for the implement the combat logic, can use two kinds of combo attacks and a Special attack (Run Attack). The Light combo or the Heavy combo. To accomplish this a Dynamic combo system was created, the player must trigger the next combo in certain amount of time 0.5sec at the end of the current attack (Light or Heavy). Each combo has a multiplier that multiplies the amount of damage and the stamina consuming. EnableCombo() functions contain the logic that can enable the combo which is trigger from UAnimNotifyEnableCombo placed in the animation montage.



For dealing damage both player and enemies are using the BoxTrace() logic, with result filtered through IHitInterface. Player and Enemies inherit from the AWeapons class the logic of box trace and the weapons are attached to their skeleton mesh on the socket that created RightHandSocket (on the right hand). Also, a shield is attached in the left hand creating a socket ShiledSocket. Now the shield is used only for visual purposes but in future logic will be applied for blocking attacks. For visual and sound effects trails and sounds added. For creating more realistic combat and clever enemies motion wrapping was added and a hit reaction that calculates from the box trace the vector of the hit and trigger the appropriate reaction from the montage.



CHARACTER STATES

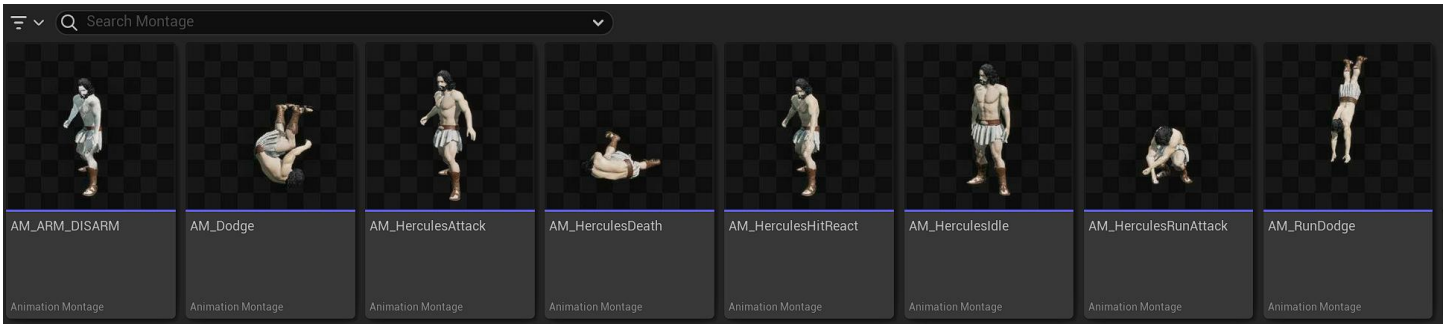
```
UENUM(BlueprintType)

// creat enum for character if it is attacking or not
enum class EActionState : uint8
{
	EAS_Unoccupied UMETA(DisplayName = "Unoccupied"),
	EAS_Walking UMETA(DisplayName = "Walking"),
	EAS_Running UMETA(DisplayName = "Running"),
	EAS_HitReaction UMETA(DisplayName = "HitReaction"),
	EAS_Attacking UMETA(DisplayName = "Attacking"),
	EAS_EquippingWeapon UMETA(DisplayName = "Equipping Weapon"),
	EAS_Dodge UMETA(DisplayName = "Dodge"),
	EAS_Dead UMETA(DisplayName = "Dead")

};
```

2.3 Dodge & Weapon (UN) equip

The dodge speed varies depending on whether Hercules is running or walking. For each situation a different montage is used. Both dodge actions cost stamina and are disable if the amount of the stamina that is need it is not available. The equip and unequip change the animation state and switches sockets from RightHandSocket to SpineSocket so have the visual representation but has crucial functionality for changing the state and the input context so player can be in combat state and use the attack buttons.



Equip State

```
UENUM(BlueprintType)
// create ENUM for the character state  Holding or not a weapon assign in 8 bit intiger
```

```
enum class ECharacterState : uint8
{
	ECS_Unequipped UMETA(DisplayName = "Unequipped"),
	ECS_EquippedOneHandedWeapon UMETA(DisplayName = "Equipped One-Handed Weapon"),
	ECS_EquippedTwoHandedWeapon UMETA(DisplayName = "Equipped Two-Handed Weapon")
};
```

2.4 Healing & Stamina

Healing is triggered with key (1) and heals a fixed amount. A cooldown image and material created so it can be used in the HUD for representation, fades and fills using a dynamic instance material. Stamina regenerates over time RegenStamina and is capped at MaxStamina

3. AI Systems

3.1 Enemy AI

The AEnemy class has its own state machine and functionality. Also inherits functions from ABaseCharacter and some of the functions are override. It has the ability for patrolling in random points or in certain target waypoints that are applied in an array. For chasing the player using PawnSensingComponent and MoveToLocation. The attacking based on the CombatRadius and AttackCooldown, for changing the states times also used starting and resetting them as the logic needs. Also, a logic of LoseInterest is applied so when the player is far away the enemy will return to the patrol state. For avoiding collisions with other actors I used RVOAvoidence. On death enemies play a death animation which is randomly chosen from the death posses and then the AI logic is stopped and after a certain amount of time the enemy is destroyed from the scene. As a reward drops XPOrbs which can be used for level up (in Future).

```
UENUM(BlueprintType)
// create an ENum for Enemy
enum class EEnemyState : uint8
{
	EES_NoState UMETA(DisplayName = "NoState"),
	EES_Dead UMETA(DisplayName = "Dead"),
	EES_Idle UMETA(DisplayName = "Idle"),
	EES_Patrolling UMETA(DisplayName = "Patrolling"),
	EES_Chasing UMETA(DisplayName = "Chasing"),
	EES_Attacking UMETA(DisplayName = "Attacking"),
	EES_Engaged UMETA(DisplayName = "Engaged")

};
```

3.1 Animal AI

AAnimals define wildlife AI and can be found in certain locations on the map. They have simple behavior and their own state. Some of the animals also used as an enemy (male Deer and Wolf). They can wander randomly with in WanderRadius, they can be in an idle state MinWait-MaxWait in seconds, and they can flee FleeRadius if the player gets close enough. For avoiding collisions with other actors, I used RVOAvoidence.

```
UENUM(BlueprintType)
enum class EAnimalState : uint8
{
	EAS_Idle UMETA(DisplayName = "Idle"),
	EAS_Wandering UMETA(DisplayName = "Wandering"),
	EAS_Fleeing UMETA(DisplayName = "Fleeing")
};
```

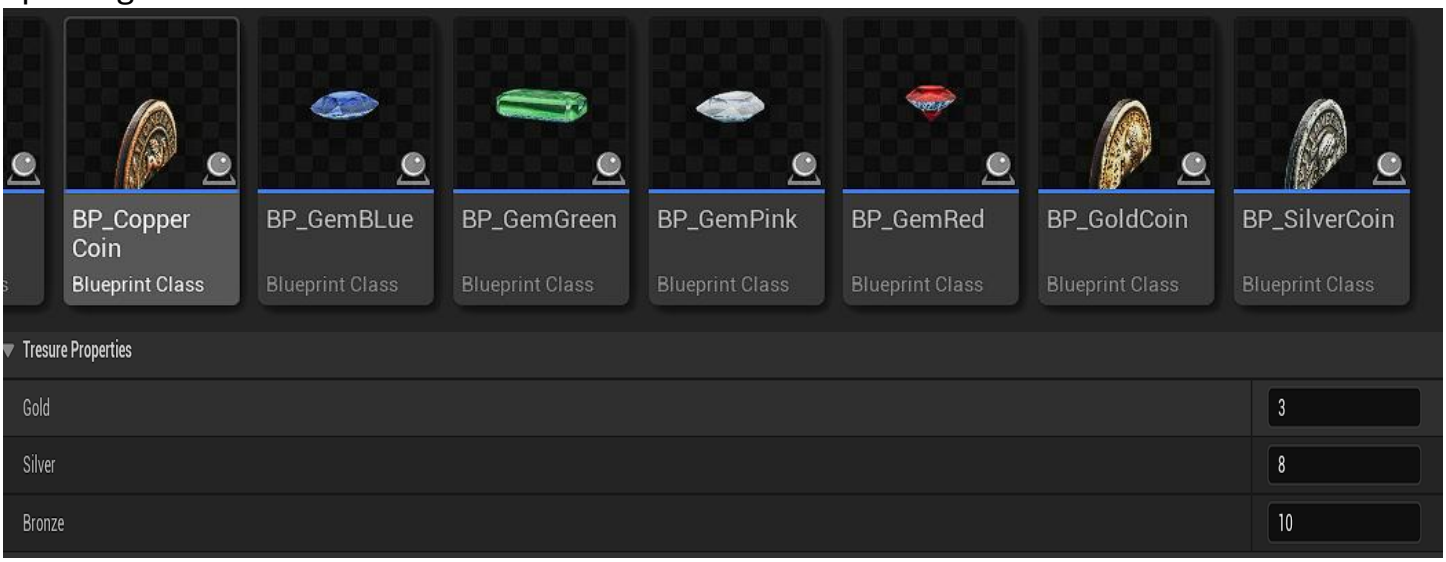
4. Items & Interactable

4.1 Weapons

AWeapons derived from the AItems class. It is equipped with proximity and input as already described in the combat mode. The collisions are disable by default and are only enable during attacking state. Supports the BoxTrace() and use the BoxTraceStart and BoxTraceEnd to define the area. Calls the GetHit() from IHitInterface and applies damage to the appropriate ownership. Support the enemy and player logic and with the flag bUseOwnerDamage avoiding damaging his ownership and their same class. That was accomplished with Fname tags. All the weapons have the ability to pick up from the player if it is desired.

4.2 Treasure & Currency

ATreasure is responsible for the currency and it is also derived from the AItems class. Future logic to buy and sell items. It uses the IPickUpInterrface and is connected also with the UAttributeComponent and the HUD. Coins use an auto conversion logic 100 bronze =1 silver, 100 silver = 1 gold and handled from the IPickUpInterrface. From the editor we can set the amount of the values in bronze silver and cold and the visual of it. After collecting the treasure the Hud is updating the visuals.



4.2 XP Orbs

The AXPOrbs has the same logic as the previous. It also derives from AItems class and uses the IPickUpInterrface. Their drop every time that the player kills an enemy and is updated when Hercules overlaps with them. The value changes are updated from the HUD (Future Functionality Level UP).

5. Attributes & UI Feedback

5.1 UAttributeComponent

This class manages all of the player and enemy attribute systems and is central to gameplay balance and interaction logic. It tracks and updates vital gameplay stats and supports direct communication with HUD. This modular structure makes it easy to update the UI and gameplay logic without tight coupling between systems.

- Manages Health, Stamina, and XP Orbs
- Handles currency logic including Bronze, Silver, and Gold with automatic conversion
- UseStamina() to reduce stamina on actions
- ReceiveDamage() to apply incoming damage
- HealInstant() for quick recovery
- HealOverTime() to apply timed heal.
- Broadcasts change OnHealthChanged() delegate, enabling real-time HUD synchronization

5.2 Interfaces

Interfaces are used to facilitate decoupled communication between systems. This allows for flexible, modular behavior without requiring direct class references. The use of interfaces ensures the system is extensible and reusable for any new item, enemy, or interactive actor can plug into existing mechanics with minimal effort.
Two core interfaces are implemented:

- **IHitInterface**
  Enables any actor to react when struck. Used in:
  - Enemies (to play hit reactions, take damage)
  - Breakable objects (to trigger destruction and loot drops)
- **IPickupinterface**
  Used by Hercules to interact with collectable items. Functions include:
  - AddXPorbs(), AddBronze(), AddSilver(), and AddGold()
    These allow pickups like AXPorbs and ATreasure to communicate with the player's attribute system.

6. Environment Systems

6.1 Rock Spawner

The ARockSpawner class is responsible for dynamically spawning falling rock hazards when the player enters a predefined trigger zone:

- Utilizes a UBoxComponent to detect Hercules' overlap and begin spawning logic.
- Rocks (ARock actors) are spawned at random intervals based on configured MinSpawnInterval and MaxSpawnInterval values.
- Each rock falls with gravity and, upon impact, can damage the player using built-in collision detection.
- Optional features include playing a camera shake and a spawn sound effect.
- Rocks destroy themselves automatically either after a configured LifeSpan or immediately upon impact.

This mechanic adds an environmental challenge and encourages spatial awareness and timing.

6.2 Breakable Objects

ABreakableActor introduces destructible environmental elements that enhance interactivity and exploration:

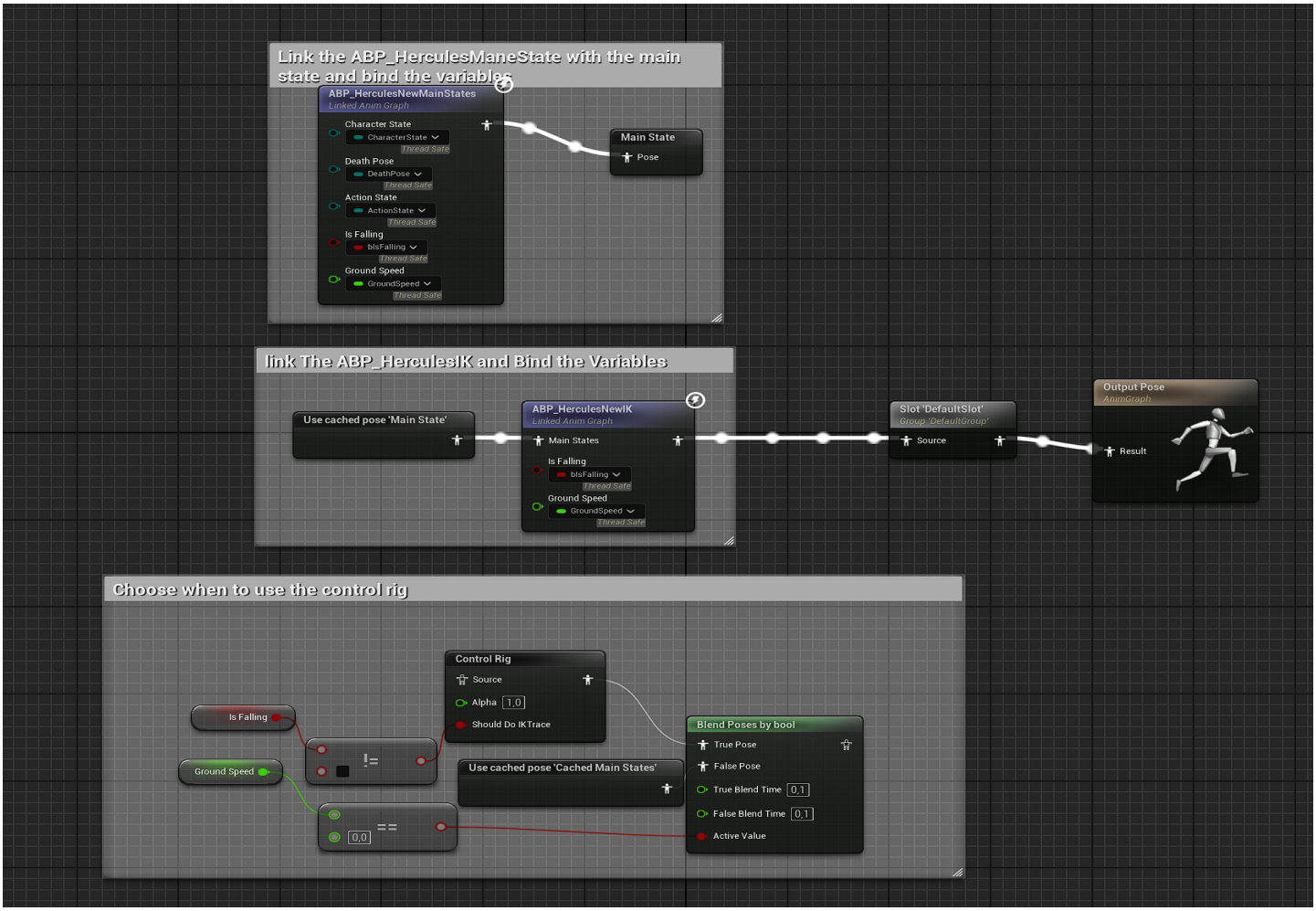
- Implements the IHitInterface so that rocks or weapons can trigger its destruction.
- Uses a UGeometryCollectionComponent to simulate a realistic break effect (fracturing upon impact).
- When destroyed, a random ATreasure actor is spawned from a configurable array (TArray<TSubclassOf<ATreasure>>).
- A UCapsuleComponent is used for collision, allowing the object to block navigation while being unaffected by Chaos physics calculations.
- The object can only be destroyed once, using a bBroken flag to prevent multiple activations.

7. Animation System

7.1 Hercules AnimInstance

UHerculesAnimInstance binds animation logic to character state:

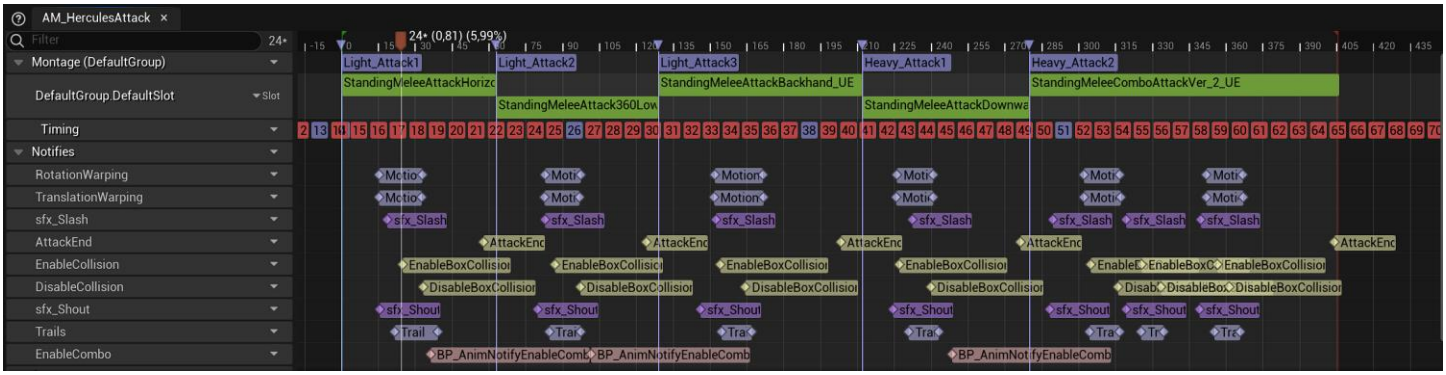
- Tracks GroundSpeed, bIsFalling, bIsRunning
- Retrieves CharacterState, ActionState, and DeathPose from Hercules
- Used in blendspaces and state machines



7.2 Combo AnimNotify

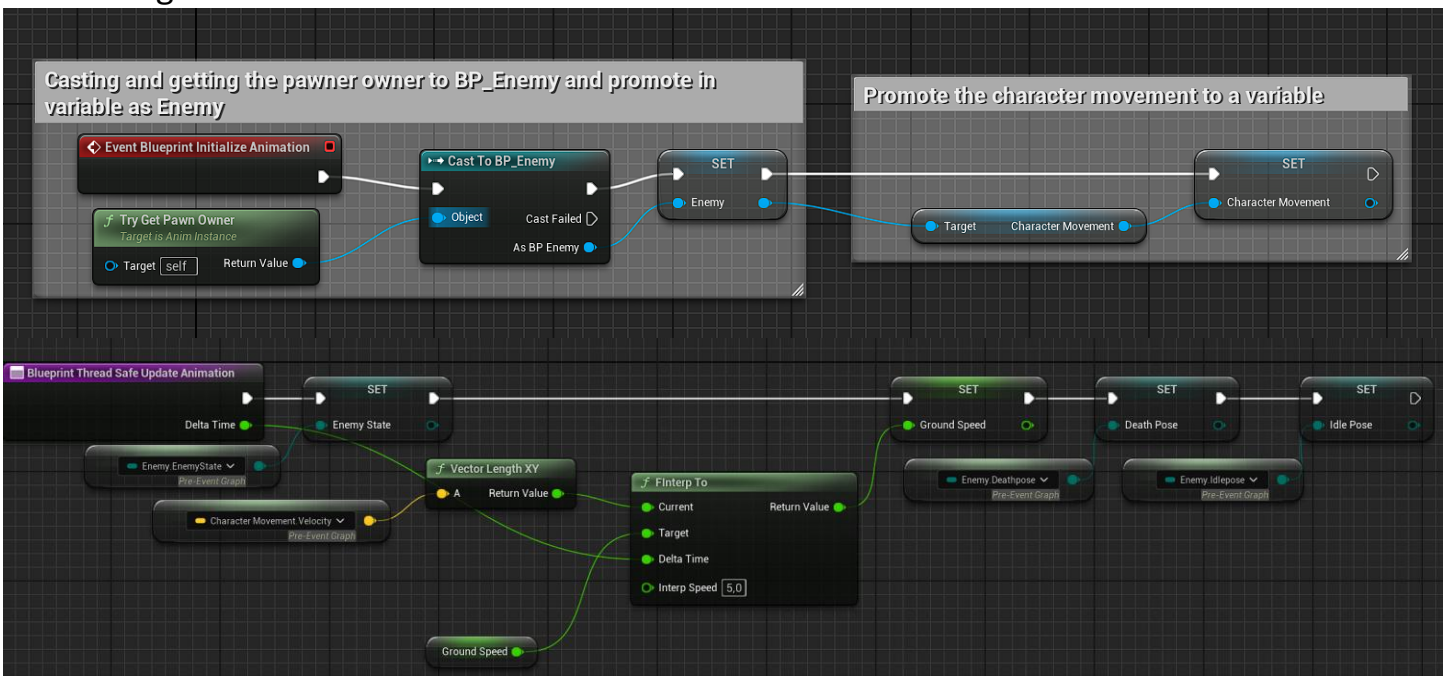
UAnimNotifyEnableCombo enables combo chaining by calling EnableCombo() in AHerculesCharacter

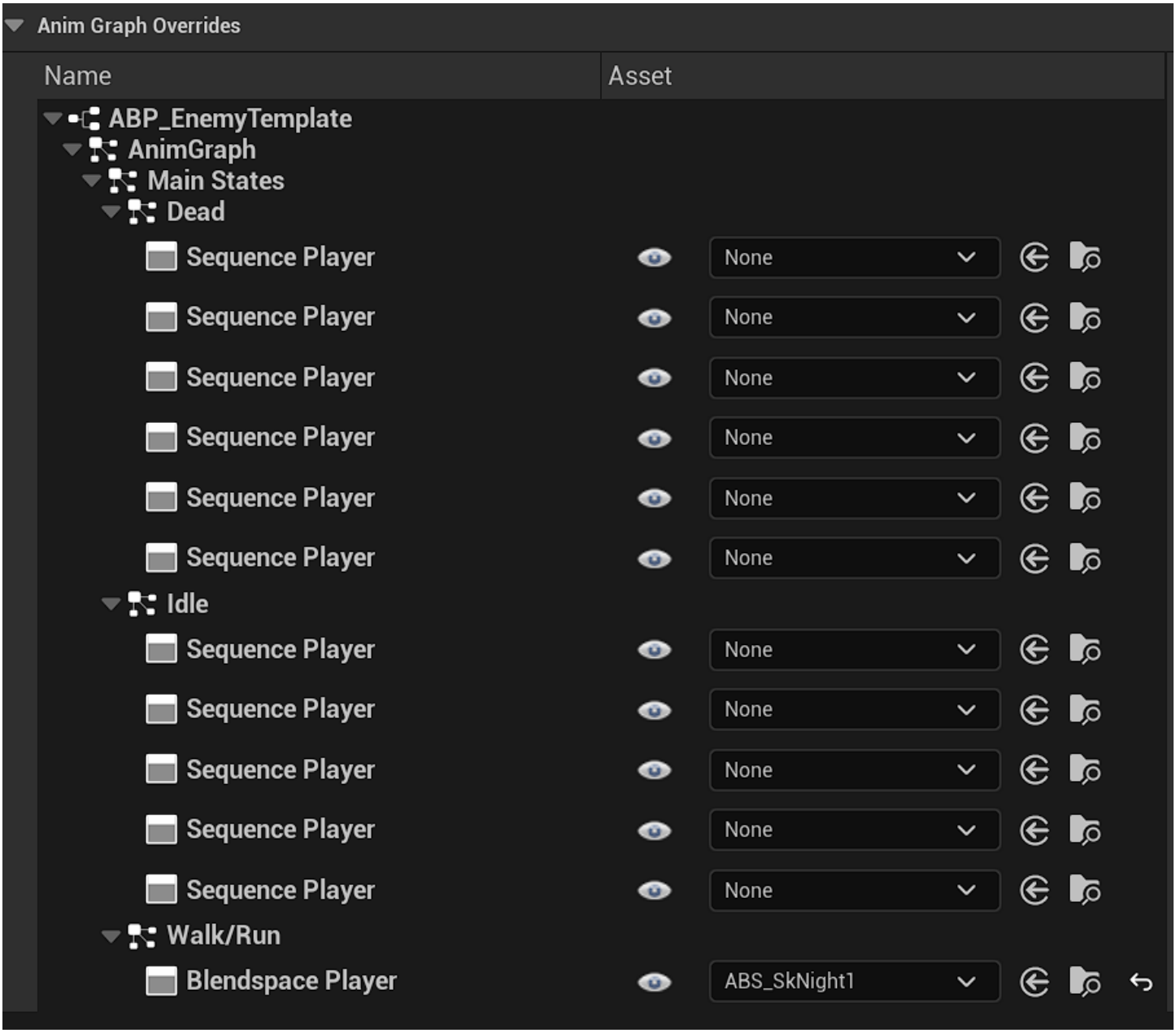
- Placed manually in animation montages (e.g. LightAttack_1)
- Enables chaining only at correct animation frame



7.3 Enemy Animations

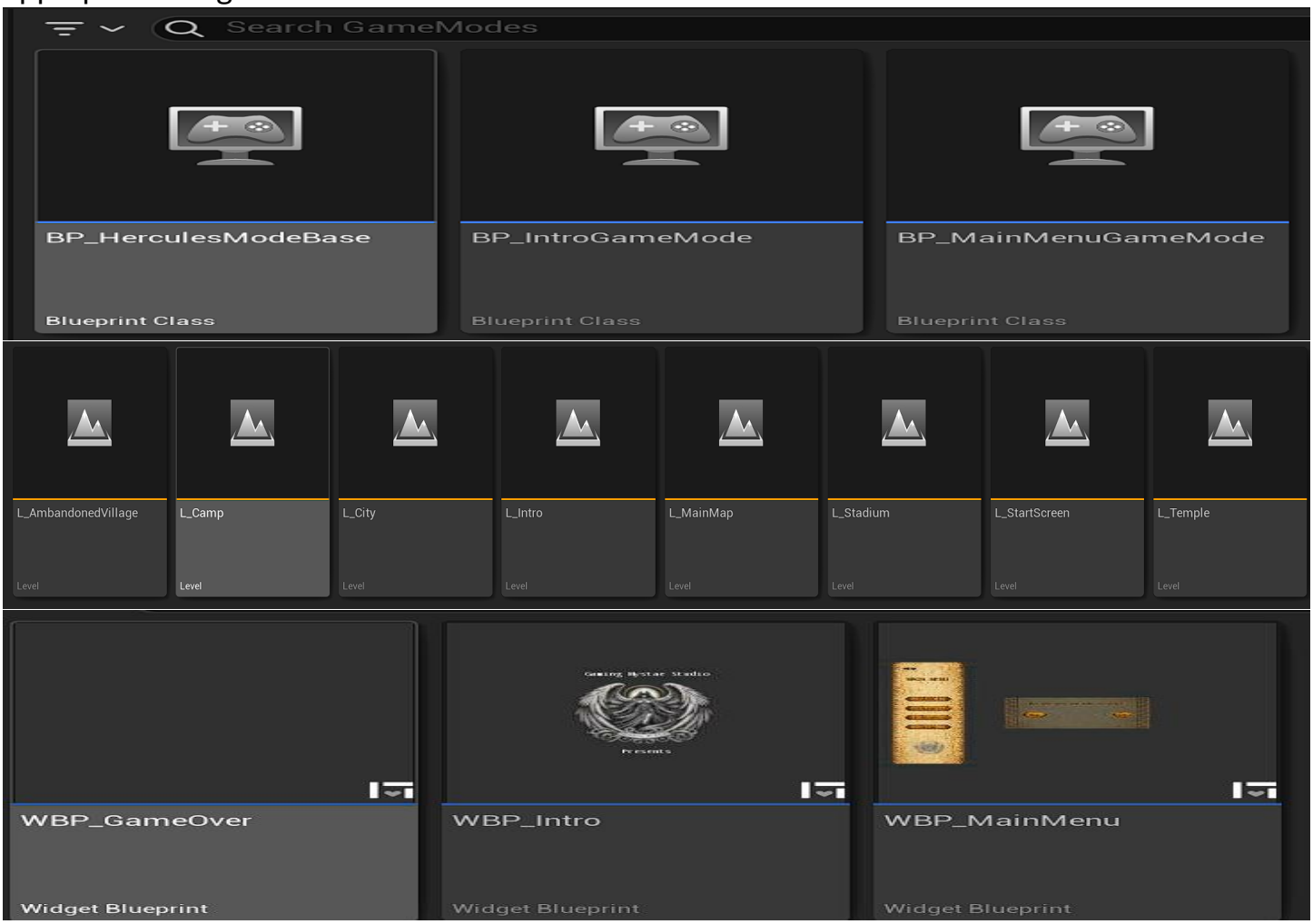
Avoiding multiply animations blueprints for the different enemies a main template animation blueprint was created and all the enemies derived the logic from this animation the variables and the montages.




Anim Graph Overrides
Name
Asset
ABP_EnemyTemplate
AnimGraph
Main States
Dead
Sequence Player
None
Sequence Player
None
Sequence Player
None
Sequence Player
None
Sequence Player
None
Sequence Player
None
Idle
Sequence Player
None
Sequence Player
None
Sequence Player
None
Sequence Player
None
Sequence Player
None
Walk/Run
Blendspace Player
ABS_SkNight1

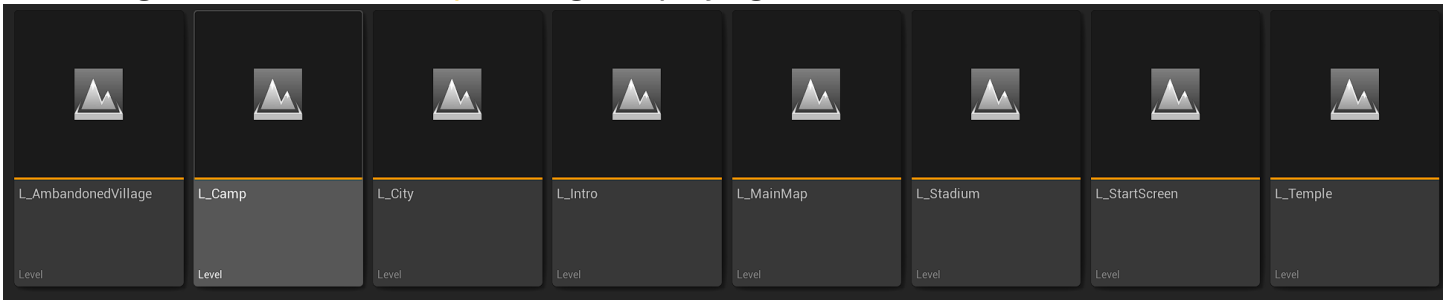
8. Game Mode & HUD

There are three different Game Modes. The HerculesModeBase the IntroGameMode and the MainMenuGameMode. Each one is connected to the appropriate level and loading the appropriate widgets.



9. Levels

Three levels have been created. The L_Intro which it is the loading screen with video that is the story line of the game and with loading screen. The L_StartScreen which is an animated screen with the widget that player can see the credits, to adjust the settings to mute the sound to quit or start the game. The L_MainMap is the game playing level.



10. External tools & Other Unreal Engine Processes

Outside Unreal Engine, several third-party tools and creative workflows were used to support asset generation, design, and optimization:

Audio Production

- **Audacity**: Used for editing sound effects and adjusting timing.
- **Freesound.org**: A source of royalty-free sound effects.
- **Fab.com**: For acquiring ambient tracks and battle SFX.

3D Mesh Creation & Retargeting

- **Blender**: Used for sculpting environment meshes and modifying static meshes.
- **Meshy.com** / **Fab.com**: To source and optimize models for the environment and props.
- **ArtStation.com**, **RigModels.com**: For creative inspiration and model references.
- **Adobe Substance 3D:** For creating the materials of the meshes.

Character Design

- **MB-Lab** and **Daz3D**: Used to generate the base Hercules character mesh.
- **Blender**: For adjusting proportions, re-topologizing, and mesh clean-up.

Animation Systems

- **Mixamo.com**: Provided retargeted animations for attacking, idle, and dodge behaviors.
- **AccuRig**: Used to create custom rigs on scanned models.
- **Blender**: For modifying animation sequences.

Image & UI Creation

- **GIMP**: For designing UI elements like the HUD icons.
- **ChatGPT**: Used for documentation assistance, naming conventions, and system breakdowns.

Unreal Engine inside.

- Chaos Cloth system for creating the movement Flag
- Fracture system mode.
- Modelling System mode.
- Rig system.

- Brush Editing mode.
- Landscape mode.
- Foliage mode.
- Particle system.
- Niagara system.
- Metasounds system.
- Attenuation system.

11. Critical Review

Due to my initial excitement and relative inexperience in game development, I underestimated both the time constraints and the overall scope of the project I intended to create. As a result, the game was not completed within the planned timeframe. The greatest challenge I encountered was designing characters with authentic ancient Greek clothing (a process that proved to be particularly time-consuming and technically demanding). Additionally, several planned systems such as inventory, quest management, cinematic cutscenes, and in-game shopping were not implemented.

Throughout the development of Hercules Untold, my primary goal was to build a modular and extensible action-adventure framework with engaging player mechanics, interactive environments, and a cinematic feel. The project integrates systems such as combo-based combat, layered enemy AI, and real-time environmental hazards.

One of the most successful features is the combo system, which chains attacks based on stamina and is closely tied to the animation system via custom notifies. This demonstrates a solid understanding of animation timelines and gameplay synchronization. The enemy AI is multi-layered (including patrol, detection, pursuit, attack, and death reactions) along with loot drops and UI updates.

Another strong design element is the use of interfaces (such as IHitInterface and IPickUpInterrface), which significantly enhanced code reusability and decoupled class dependencies. This allowed objects, enemies, and interactive elements to communicate with the Hercules character through abstract commands, making future systems like shops, quests, and puzzle triggers easier to implement.

From a visual and feedback standpoint, the HUD provides immediate updates for health, stamina, currency, XP, and healing cooldowns. The dynamic material used for cooldowns adds a modern and polished aesthetic to the UI.

While the core mechanics function effectively, their integration at the level design stage could benefit from additional puzzle-based or exploration challenges — for example, locked areas requiring specific skills. The current design lays a strong foundation for an action RPG but lacks unique interactive obstacles. Furthermore, although the existing AI system performs well, further abstraction through behaviour trees or state machines could improve scalability for larger projects.

Structurally, the project is well-documented and supports both future development and academic evaluation. The integration between C++ and Blueprint interfaces has been consistently smooth, demonstrating a thoughtful balance between performance and flexibility.

12. Gameplay Systems & Evaluation Summary.

Character Mechanics

Mechanic	Complexity	Notes
Light & heavy combo attacks	Complex	Timing-sensitive, stamina-based chaining
Dodging with stamina	Medium	Includes stamina cost and iFrames
Weapon equip/disarm system	Medium	Dynamic weapon state switching
Healing with cooldown & UI feedback	Medium	Cooldown visualized with dynamic material
Interaction with breakables & pickups	Medium	Connected to custom interfaces
Full input mapping (Enhanced Input)	Above baseline	Modular and scalable input system

Gameplay Mechanics Implemented

Feature	Complexity	Notes
Rock spawner hazard	Medium–High	Trigger-based real-time hazard
Breakable actors with loot	Medium	Dynamic loot drops with spawn variation
Currency system	Medium	Dynamic conversion logic implemented
Enemy AI (patrol, chase, attack, loot drop) with RVO	Complex	Layered logic with behavior switching
Animal AI with RVO and state transitions	Medium	Includes passive/reactive states
Interaction system with interfaces	Complex	Abstracted communication (e.g., IitInterface)

Gameplay Interaction Systems

Interaction	Complexity	Notes
XP from defeated enemies	Simple	Adds basic RPG progression
Combo unlock from AnimNotify	Medium	Animations synced with logic
Weapon pickup & equip	Medium	Includes switching and stat effects
Rock spawner trigger logic	Medium	Real-time challenge generation
Breakables with treasure spawns	Complex	Randomized loot using spawn points