Too Many Eyes / *Locomotion*

System Design Document

# Changes

## Version 0.1

Editor: Alexander Bowerman

01/11/2022

* Document Created.
* Introduction, Design Goals, and Behaviour descriptions added.

## Version 0.2

Editor: Alexander Bowerman

02/11/2022

* High Level Design, Mid Level Design UML added.
* High Level Design, Mid Level Design descriptions added.
* Case View Steps & Screenshots added.

## Version 0.3

Editor: Alexander Bowerman

03/11/2022

* Logical View UML added.
* Added Process View for MoveForward, MoveRight, Jump.

## Version 0.4

Editor: Alexander Bowerman

04/11/2022

* Added Process View for Jump, Vault, Climb, Sprint.

# Introduction

*This document details the design and purpose of a locomotion system designed for Too Many Eyes. Designed for the Blight Brew Game, the document will cover;*

*⦁* *Usage for the designers.*

*⦁* *UML for the Programmers.*

# Design Goals

The goal of this system is to;

* *Have an immersive and dynamic movement system that all characters adhere to*
* *Be intuitive and approachable for players of all skill levels*
* *Give the player greater exploration abilities*

# Behaviour

*The Locomotion System is a dynamic movement system that all characters in BlightBrew use and adhere to. For the player, locomotion will feel fluid and immersive, and follows the logic patterns of exploration games, where the player is able to traverse the environment effortlessly, and without needing additional abilities/tools to do so.*

* *The* ***Locomotion System*** *handles the* ***Tagged Actors*** *and* ***Character****(s)*
  + *It acts as an interface between all users of the locomotion system*
* ***Tagged Actors*** *are any actor within the map that have been given the gameplay tag(s) of* ***Vaultable*** *or* ***Climbable*** *(The actor cannot have both tags)*
  + *These tags will tell the instigating character which locomotive abilities they are able to use on the chosen Actor*
* ***Character*** *system**is the class which all characters in Blight Brew inherit from.*
  + *Character has implementation for the* ***Stamina Component*** *which manages everything to do with movement*
* ***Player*** *system is the main player system handling the player controls, and player interactions/actions with the* ***Locomotion System***
  + *Player calls its parent functions (****Character****) for interacting with the* ***Stamina Component*** *and* ***Locomotion System***
* *The* ***Stamina Component*** *system handles all logic for movement including:*
  + *Walking*
  + *Sprinting*
  + *Jumping*
  + *Vaulting*
  + *Climbing*
  + *Swimming*

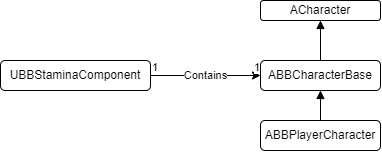
# High Level Design

Graphical user interface, text, application, chat or text message

Description automatically generated

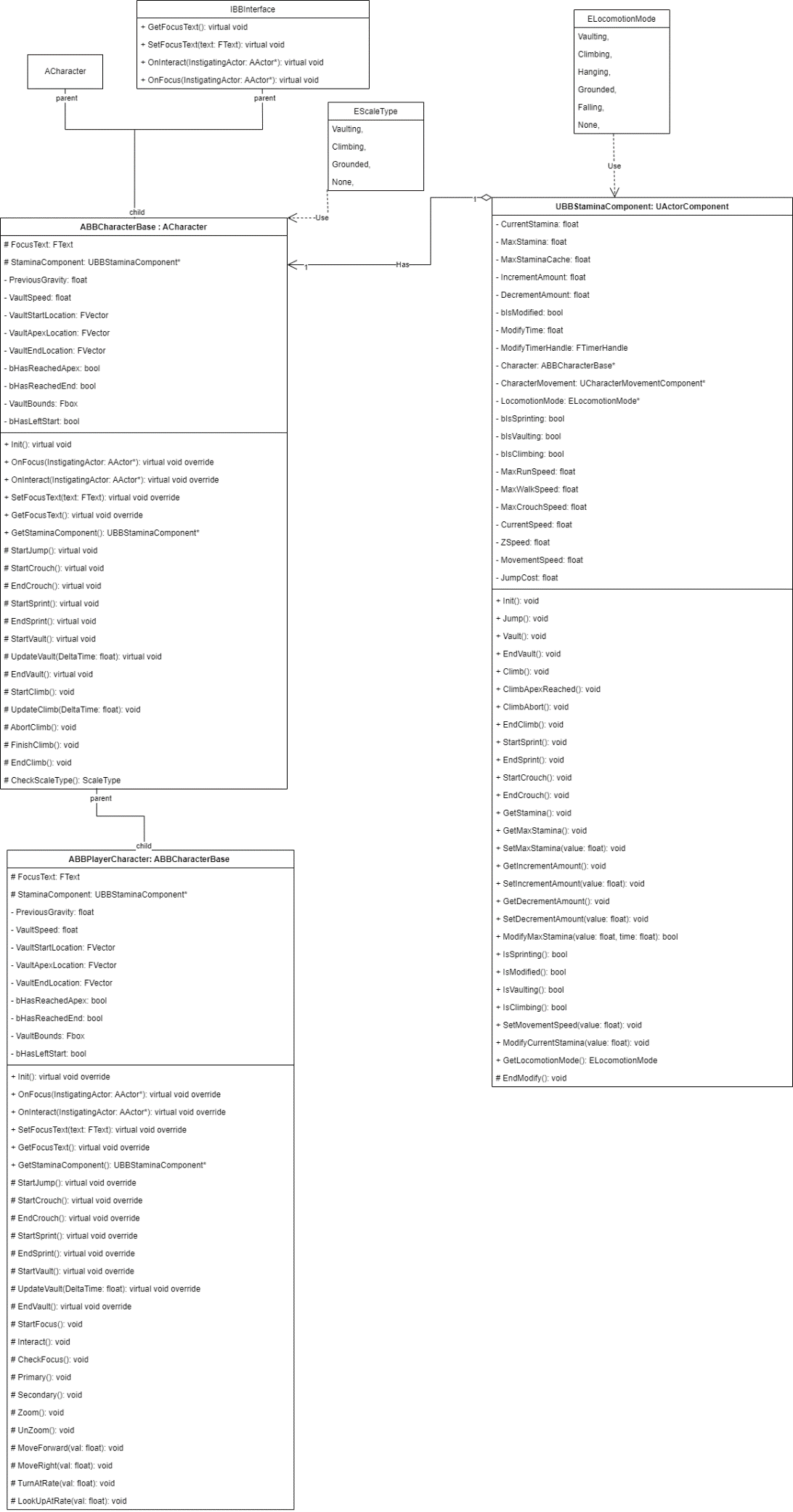
* Player system - the system is our core system and is what the player interacts with to use all our other systems.
* Locomotion system - an advanced movement system we’re using to give us more control over the players movement. This gives us variables we can control, such as stamina, and additional functionality, such as climbing and vaulting.
* Potion system – a system meant to be easy to use for designers. The system allows for the designers as much freedom as possible, while giving the coders a minimal amount of updating to do. This is the main system the player will use to complete puzzles and generate income.
* Inventory system – a system meant to be easy to use for designers. The system allows for the designers as much freedom as possible, while giving the coders a minimal amount of updating to do. The system works as a container to hold items that the player collects.

# Mid Level View



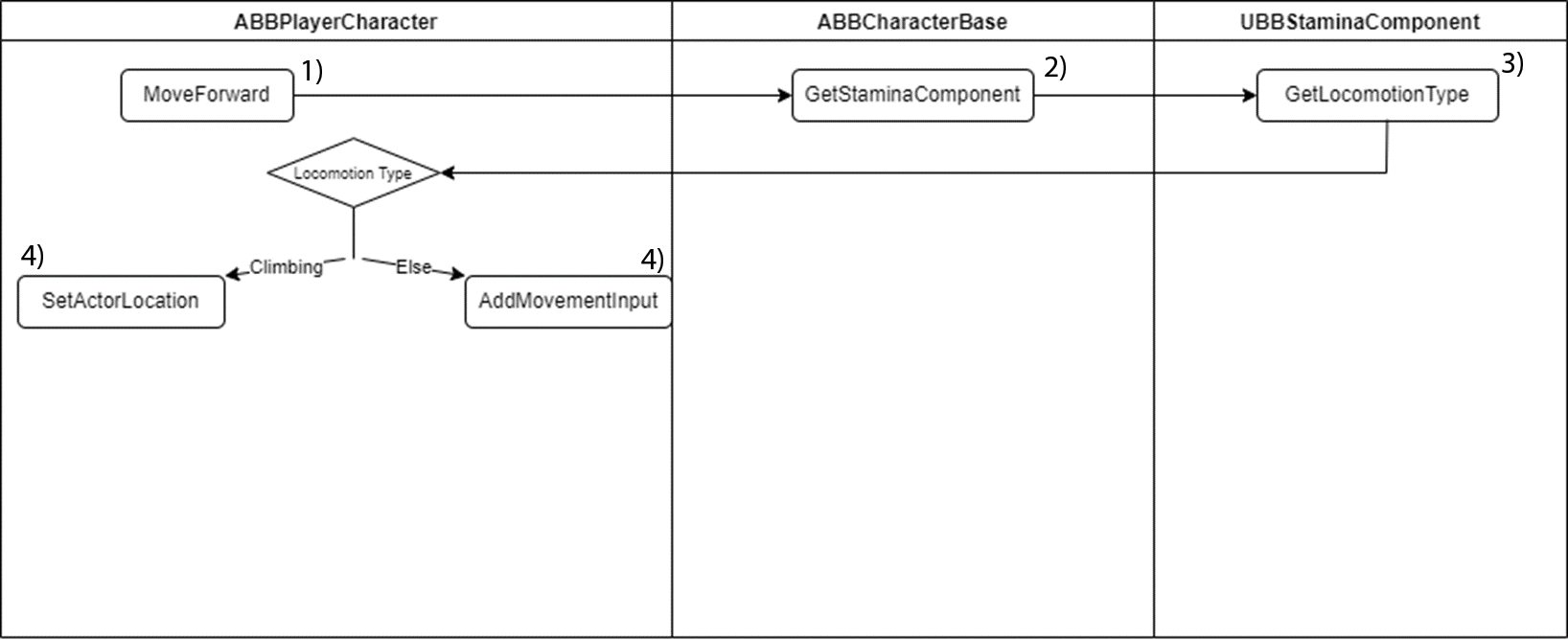
* ***ABBCharacterBase*** *is the class which all characters in BlightBrew inherit from*
  + ***ABBCharacterBase*** *contains 1* ***UBBStaminaComponent****, which any children of the class use to execute Locomotion functions*
* ***UBBStaminaComponent*** *is the centrifuge of the Locomotion System*
  + *The stamina component manages the current locomotive state of the character, the stamina of the character, costs for actions, checks for actions, and the logic for executing locomotive actions*
* ***ABBPlayerCharacter*** *is the class which handles player controls, and player actions/interactions*
  + *For any locomotive actions it uses the* ***UBBStaminaComponent*** *it inherited from* ***ABBCharacterBase***

# Logical View



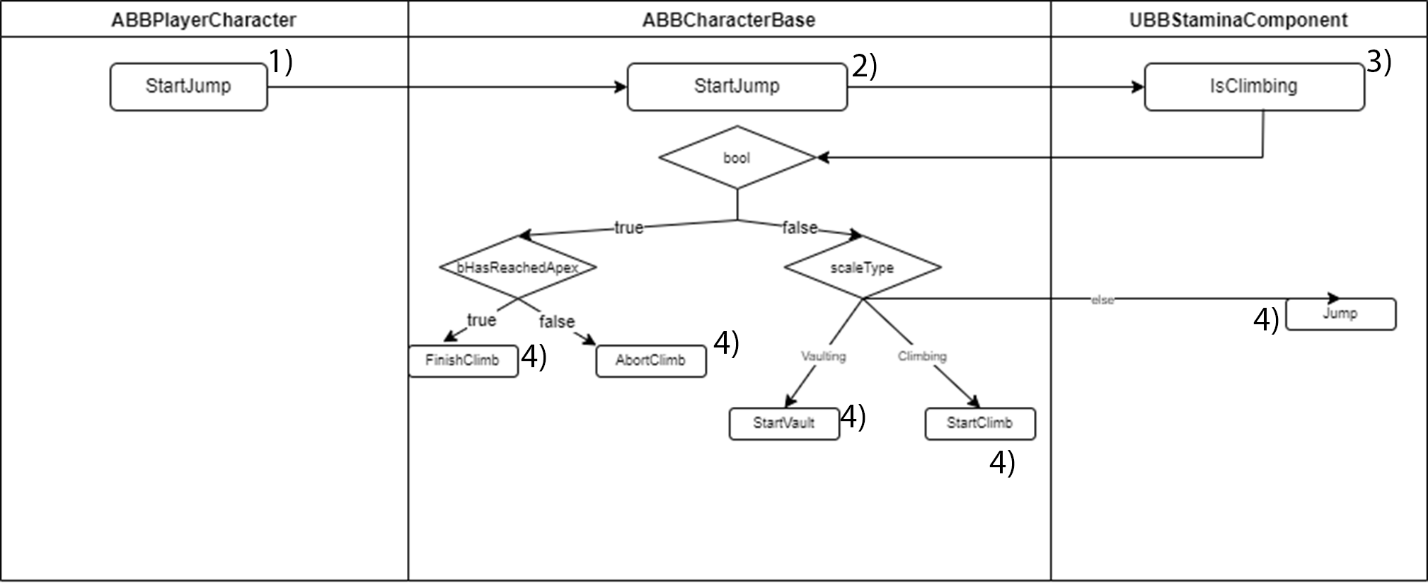
# Process View

## MoveForward



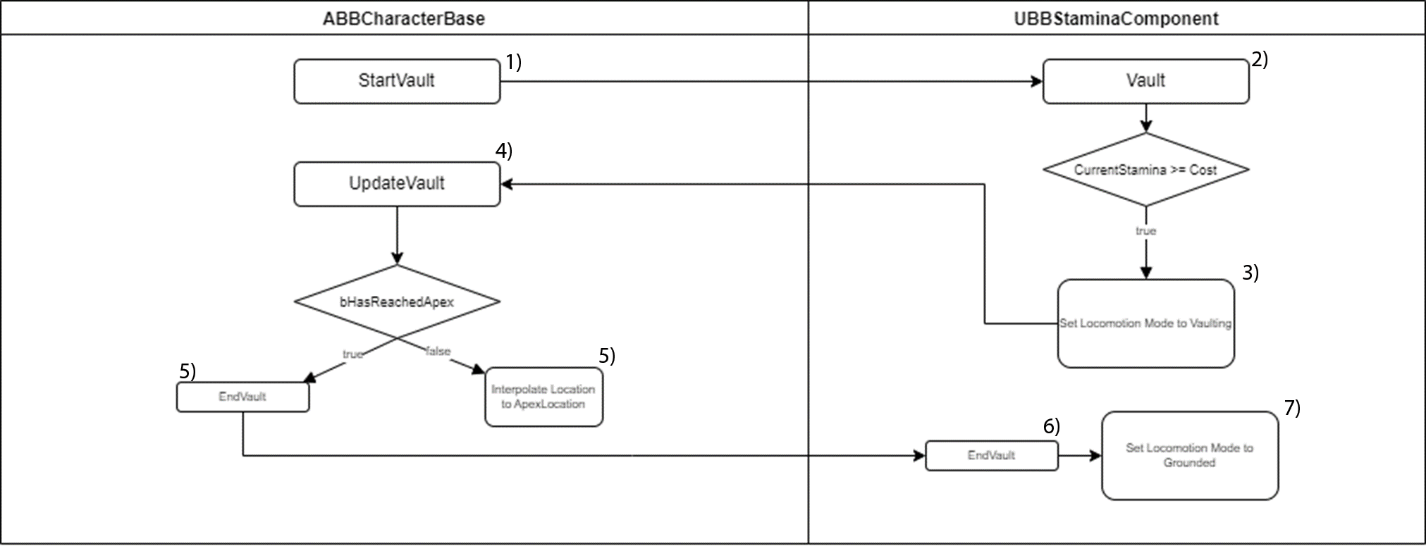
ABBPlayerCharacter listens for the MoveForward input from the Player Controller. It then gets the UBBStaminaComponent from its parent (ABBCharacterBase). From the StaminaComponent, it retrieves the current LocomotionType(ELocomotionType). If the Locomotion Type is Climbing, the PlayerCharacter will apply the MoveForwardValue to its SetActorLocation Z Value. If the Locomotion Type is any other type, (an else statement), then it will apply the movement using AddMovementInput, and passing in the Forward Vector and MoveForwardValue.

## Jump



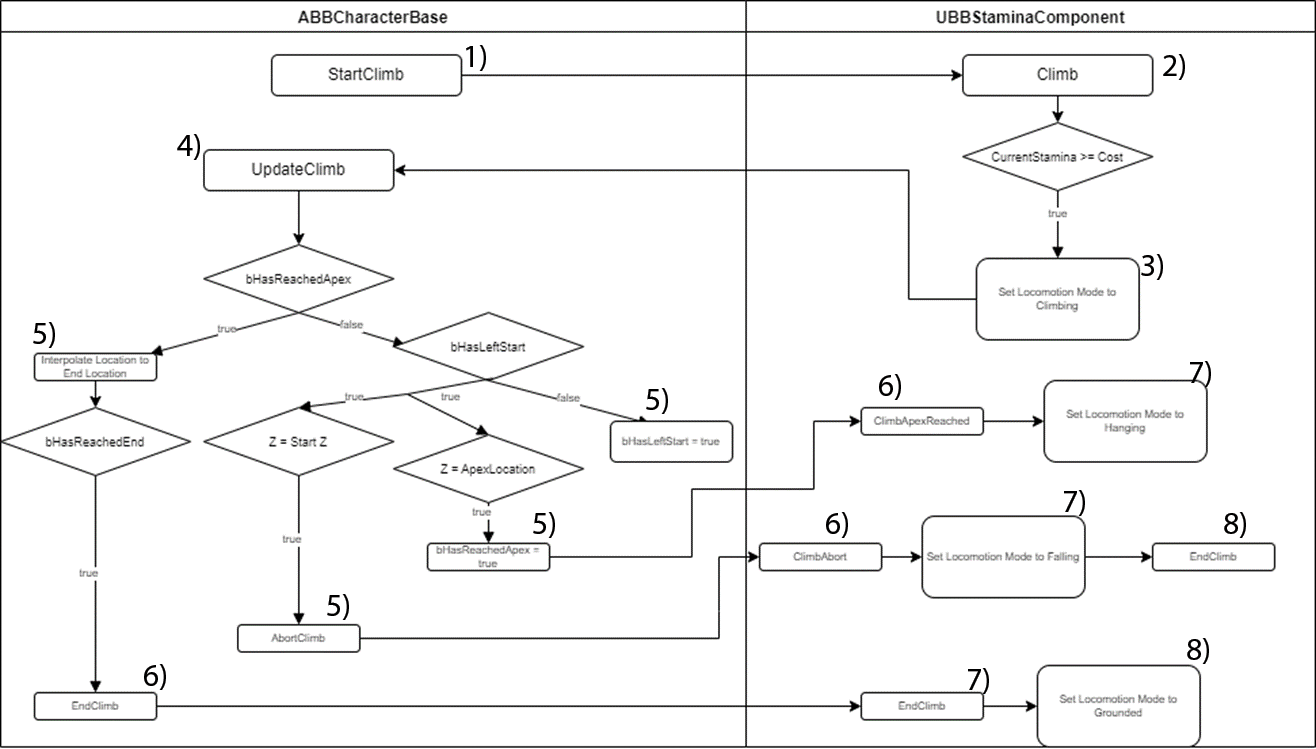
ABBPlayerCharacter listens for the StartJump input from the Player Controller. It then calls the StartJump function from its parent (ABBCharacterBase). From the StaminaComponent, ABBCharacterBase checks if the StaminaComponent is currently Climbing. If the StaminaComponent is climbing the Character will check if they have reached the Apex of their Climb. If they have reached the apex, it will call the FinishClimb function, to finish climbing. Else it will call the AbortClimb function to cancel the climb and jump off the currently climbed object. If the StaminaComponent is not climbing, the Character will check their current Scale Type (EScaleType) which will tell the Character if they should be Vaulting, Climbing, or Jumping. If they should be Vaulting, it will call the StartVault function. IF they should be Climbing, it will call the StartClimb function. Otherwise, it will call the Jump function from ACharacter.

Vault



ABBCharacterBase receives the StartVault from the Jump Process. StartVault calls the UBBStaminaComponent’s Vault function. The Stamina Component checks if it has enough Stamina to afford the action, if true it will set the Locomotion Mode to ELocomotionMode::Vaulting. Back in the Character, the Character’s Tick will call UpdateVault, if the ELocomotionMode is set to Vaulting. UpdateVault will check if the Character has reached the Apex. If it hasn’t, the Character will interpolate their current location to the Vault’s Apex Location, which is set when the Vault is Calculated. If it has, the Character will trigger the EndVault function. EndVault will trigger the Stamina Component’s End Vault function, which will ensure that that the character has properly landed on a solid surface and set the Locomotion Mode to ELocomotionMode::Grounded.

Climb



ABBCharacterBase receives the StartClimb from the Jump Process. StartClimb calls the UBBStaminaComponent’s Climb Function. The Stamina Component checks if it has enough Stamina to afford the action, if true it will set the Locomotion Mode to ELocomotionMode::Climbing. Back in the Character, the Character’s Tick will call UpdateClimb, if the ELocomotionMode is set to Climbing. UpdateClimb will check if the Character has reached the apex.

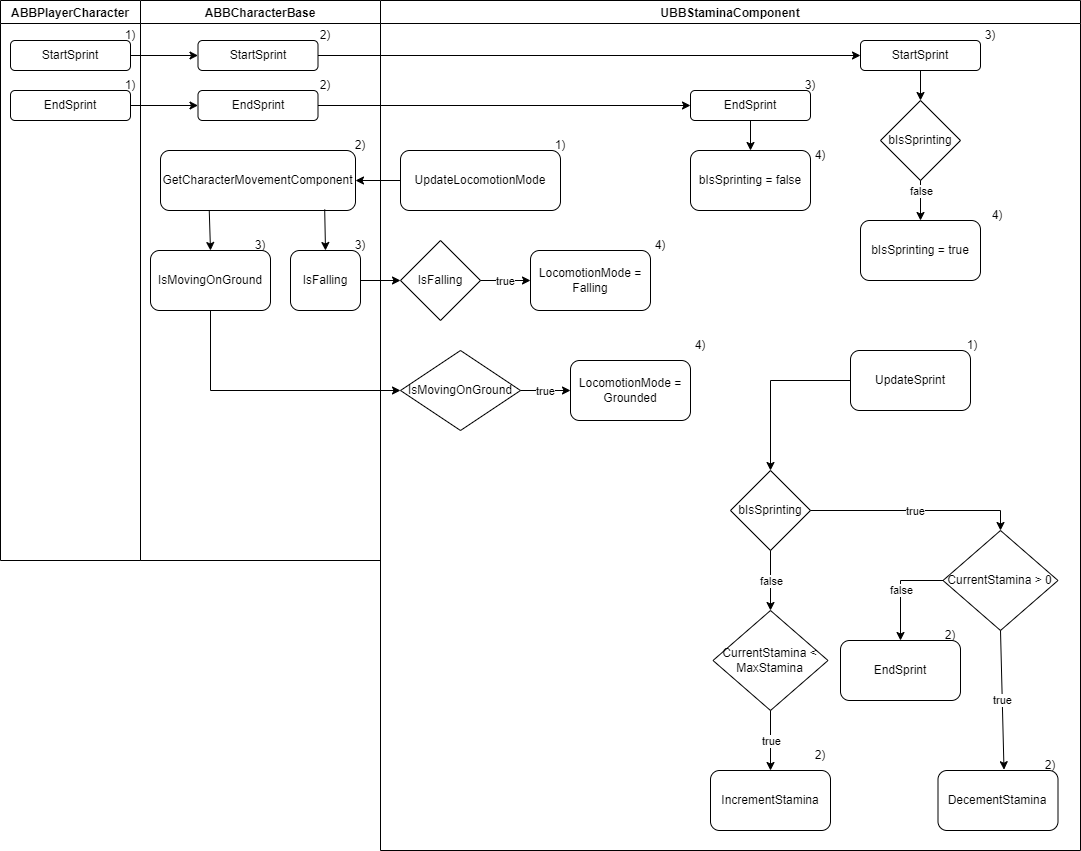
**If it hasn’t reached the apex**, it will then check if it has left it’s start location and set bHasLeftStart accordingly (one time if the character’s location has changed).

**If it has left the start location,** it will check the Characters current location.

**If it has *returned to the start location*,** it will call **AbortClimb** to cancel the climb. **AbortClimb** will call the StaminaComponent’s ClimbAbort Function. StaminaComponent will then set the Locomotion Mode to Falling and call the EndCllimb Function (which resets all the flags for checking a climb/vault).  
  
**If it has reached the Apex Location,** it will set bHasReachedApex to true, and call the StaminaComponent’s ClimbApexReach function. This Function will set the Locomotion Mode to ELocomotionMode::Hanging.

**If it has reached the apex,** the Character will interpolate its location to the ApexEndLocation. If it has reached it, it will call the EndClimb Function. This function will reset local flags & variables, and then call the StaminaComponent’s EndClimb Function. This function will reset local flags & set the Locomotion Mode to ELocomotionMode::Grounded

Sprint



ABBPlayerCharacter listen for the StartSprint input from the Player Controller. It then calls the parent function StartSprint. ABBCharacterBase’s StartSprint calls the UBBStaminaComponent’s StartSprint. This function checks if the component is currently sprinting. If it is not sprinting, it starts sprinting.

ABBPlayerCharacter listen for the EndSprint input from the Player Controller. It then calls the parent function EndSprint. ABBCharacterBase’s EndSprintcalls the UBBStaminaComponent’s EndSprint. This function stops the component from sprinting.

UBBStaminaComponent updates the Locomotion Mode inside of its TickComponent. It retrieves the CharacterMovementComponent from ABBCharacterBase. From the Movement Component it retrieves the IsMovingOnGround state, and if true, sets the Locomotion Mode to ELocomotionMode::Grounded. It also retrieves the IsFalling state, and if true, sets the Locomotion Mode to ELocomotion::Falling.

UBBStaminaComponent updates the UpdateSprint functionality, inside of its TickComponent. It checks if it is currently flagged as sprinting.

**If it is currently sprinting,** it checks if the Stamina has been depleted.

**If it has been depleted, it** flags the controller to EndSprint.

**If it has not been depleted,** it Decrements the current Stamina.

**If it is not currently sprinting,** it checks if the stamina has been recovered fully.

**If it has not been recovered,** it Increments the current Stamina.

# Use Case View

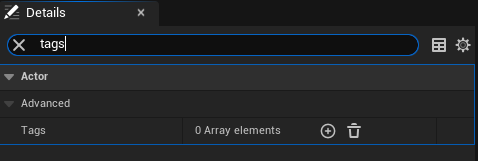
## Using the Locomotion System’s Tags

*The Locomotion System sits in the background of the engine and will recognize changes you make at runtime and apply them to the Locomotion System. One such way you can make changes that are recognized, is by adding a* ***Locomotion Tag*** *to the Actor’s Tags.*

*To access an Actor’s Tags there are two options you can do, depending on the type of actor.*

## If the Actor is a Blueprinted Actor:

* *Open the Blueprint and click on the RootComponent (Highest in the Hierarchy)*
  + *This will usually say (Self) in parentheses.*
* *Go to the* ***Details*** *panel, and search for “Tags”*

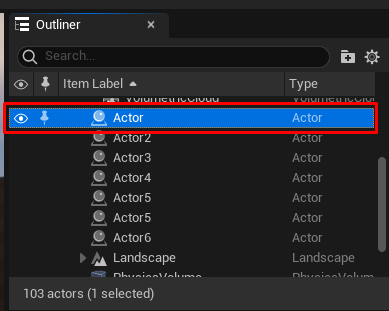


* *In the Tags List, click the plus and add* ***ONE*** *Locomotion Tag.*

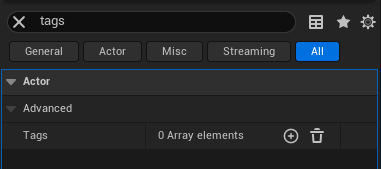
## If the Actor is NOT a Blueprinted Actor:

This will be the most used process, as most Locomotion Tagged Actors will be Environmental.

* *Click on the Actor in the* ***Outliner*** *panel of the Editor.*

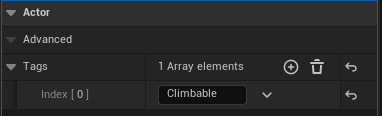


* *Go to the* ***Details*** *panel of the Editor, and search for “Tags”*



* *In the Tags List, click the plus and add* ***ONE*** *Locomotion Tag.*

## Climbable Tag

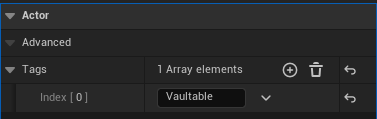


Tags the selected actor to be **CLIMABLE** by the Locomotion System. This will allow Characters to climb the selected actor, so be sure to only put it on an actor that has been designated as a Climbable Object.

**What is a “Climbable Object?”**

A Climbable Object would be one that has a linear vertical surface, in which a character should be able to logically grab onto, and climb vertically, or horizontally.

## Vaultable Tag



Tags the selected actor to be **VAULTABLE** by the Locomotion System. This will allow Characters to vault the selected actor, so be sure to only put it on an actor that has been designated as a Vaultable Object.

**What is a “Vaultable Object?”**

A Vaultable Object would be one that has a short to medium height, in which a character should be able to logically grab onto, and throw themselves over, while moving.