Table 1: `Outlets` for `AffinitiesComponent`

GetUnspentPoints	
▸ Before	`const uint8 OriginalPoints,` `uint8& ReturnedPoints`
▸ After	`const uint8 OriginalPoints,` `const uint8 ReturnedPoints`

SetUnspentPoints	

▸ Before	`const uint8 OriginalPoints,`
	`const uint8 InputPoints,`
	`uint8& SetPoints`

▸ After	`const uint8 OriginalPoints,`
	`const uint8 InputPoints,`
	`const uint8 SetPoints`

• • •

Table 2: `Outlets` for `EffectComponent`

GetStacks	
▸ Before	`const uint16 OGStacks,` `int32& ReturnedStacks`
▸ After	`const uint16 OGStacks,` `const int32 ReturnedStacks`

OnAddEffect	
▸ Before	`const EffectComponent* EffectToAdd`
▸ After	`const EffectComponent* AddedEffect`

OnRemoveEffect	
▸ Before	`const EffectComponent* EffectToRemove`
▸ After	`const EffectComponent* RemovedEffect`

• • •

Table 3: Outlets for LevelComponent

GetBaseExpYield	
▸ Before	`const float OriginalYield`, `float& ReturnedYield`
▸ After	`const float OriginalYield`, `const float ReturnedYield`

GetCXP	
▸ Before	`const uint32 OriginalCXP`, `int32& ReturnedCXP`
Note:	`ReturnedCXP` is `int32&` instead of `uint32&` for Blueprint compatability.
▸ After	`const uint32 OriginalCXP` `const int32 ReturnedCXP`
Note:	`ReturnedCXP` is `const int32` instead of `const uint32` for Blueprint compatability.

GetExpYield	
▸ Before	`const float OriginalYield`, `float& ReturnedYield`, `const uint16 DefeatedLevel`, `const uint16 VictoriousLevel`
Note:	"Defeated" and "Victorious" levels are provided for flexibility (e.g., in case you want to yield exp differently based on level difference, although technically you could always back-calculate the level difference based on the equation and `OriginalYield`).
▸ After	`const float OriginalYied`, `const float ReturnedYield`, `const uint16 DefeatedLevel`, `const uint16 VictoriousLevel`
Note:	"Defeated" and "Victorious" levels are provided for symmetry with respect to the `Before` delegate (since `ReturnedValue` is already calculated, I can't think of why you would need them, but you never know!).

Continued on next page

Table 3: Outlets for LevelComponent (Continued)

GetMaxLevel	
▸ Before	const uint16 DefaultMax, int32& AttemptedMax
Note:	DefaultMax is defined in the code. It should normally be 100, but may change for certain subclasses (e.g., a BossLevelComponent may have a max of 200 instead). AttemptedMax is int32& instead of uint16& for Blueprint compatability.
▸ After	const uint16 DefaultMax const int32 ReturnedMax

GetMinLevel	
▸ Before	const uint16 DefaultMin, int32& AttemptedMin
Note:	DefaultMin is defined in the code. It should normally be 1, but may change for certain subclasses (e.g., a EggLevelComponent may have a min of 0 instead for whatever reason). AttemptedMin is int32& instead of uint16& for Blueprint compatability.
▸ After	const uint16 DefaultMin const int32 ReturnedMin
Note:	ReturnedCXP is const int32 instead of const uint32 for Blueprint compatability.

GetBaseExpYield	
▸ Before	const float OriginalYield, float& ReturnedYield
▸ After	const float OriginalYield, const float ReturnedYield

SetBaseExpYield	
▸ Before	const float OldYield, const float InputYield, float& AttemptedYield
▸ After	const float OldYield const float InputYield, const float NewYield
Note:	▻ OldYield is the yield prior to calling SetBaseExpYield, ▻ InputYield is the original, unmodified input to SetBaseExpYield, ▻ AttemptedYield is the modified value that will be used to set the base exp yield.

Continued on next page

Table 3: Outlets for LevelComponent (Continued)

SetCXP	
▸ Before	const uint32 OldCXP, const int32 InputCXP, int32& AttemptedCXP
Note:	AttemptedCXP is int32& instead of uint32& for Blueprint compatability.
▸ After	const uint32 OldCXP const int32 InputCXP, const uint32 NewCXP
Note:	StatsComponent subscribes to this in order to change stats on level change. ▻ OldCXP is the cumulative experience points prior to calling SetCXP, ▻ InputCXP is the original, unmodified input to SetCXP, ▻ AttemptedCXP is the modified value that will be used to set the cumulative experience points.

• • •

Table 4: Outlets for StatsComponent

ApplyDamage	
▸ Before	float& BasePower, float& CritMultiplier, float& RandFluct, float& Stab, float& TypeAdvantage, UCombatStatsComponent* Attacker, UCombatStatsComponent* OwningStats
Note:	Other quantities, such as the Attacker's attacking Stat, may be calculated from the given quantities.
▸ After	const float BasePower, const float CritMultiplier, const float RandFluct, const float&Stab, const float&TypeAdvantage, UCombatStatsComponent* Attacker, UCombatStatsComponent* OwningStats

Continued on next page

Table 4: `Outlets` for `StatsComponent` (Continued)

ApplyEffects	
▸ Before	`uint16& NumStacks`, `bool& bMutual`, `UMoveData* MoveData`, `UCombatStatsComponent* Attacker`, `UCombatStatsComponent* OwningStats`
Note:	`bMutual` determines whether or not multiple Effects can attach.
▸ After	`const uint16 NumStacks`, `const bool bMutual`, `UMoveData* MoveData`, `UCombatStatsComponent* Attacker`, `UCombatStatsComponent* OwningStats`

CalculateDamage	
▸ Before	`float& BasePower`, `float& CritMultiplier`, `float& RandFluct`, `float& Stab`, `float& TypeAdvantage`, `UCombatStatsComponent* Attacker`, `UCombatStatsComponent* OwningStats`
▸ After	`const float BasePower`, `const float CritMultiplier`, `const float RandFluct`, `const float&Stab`, `const float&TypeAdvantage`, `UCombatStatsComponent* Attacker`, `UCombatStatsComponent* OwningStats`
Note:	The difference between calculating damage and applying damage is theoretical. For example, low-level AI might use `CalculateDamage` to make decisions. On the other hand, applying the damage might invoke some kind of reaction, like raising Physical Attack if hit by a move it's weak to.

Continued on next page

Table 4: Outlets for StatsComponent (Continued)

GetCritMult	
▸ Before	float& BaseMultiplier, float& CritBonus, UCombatStatsComponent* OwningStats
Note:	Total crit bonus is BaseMultiplier + CritBonus; for example, 1.5 + 0.2.
▸ After	const float BaseMultiplier, const float CritBonus, UCombatStatsComponent* OwningStats

ModifyStat	
▸ Before	const EStatEnum TargetStat, const EStatValueType ValueType, const EModificationMode Mode, const float OriginalValue, float& AttemptedValue
▸ After	const EStatEnum TargetStat, const EStatValueType ValueType, const EModificationMode Mode, const float OriginalValue, const float NewValue
Note:	All "ModifyStat" functions from StatsComponent (such as ModifyStatsUniformly or RandomizeStats) go through ModifyStatInternal, which calls this Outlet.

RandomizeStats	
▸ Before	const EStatEnum TargetStat, const FStatRandParams OriginalParams, FStatRandParams&ParamsToBeUsed
▸ After	const EStatEnum TargetStat, const FStatRandParams OriginalParams, const FStatRandParams UsedParams
Note:	The EStatEnum is not the acutal FStat. To get the FStat (such as FHealth), use StatsComponent::GetStat(EStatEnum)

Continued on next page

Table 4: `Outlets` for `StatsComponent` (Continued)

RecalculateStats	

▸ Before	`const EStatEnum TargetStat,`
	`const bool bResetCurrent,`
	`const float OriginalCurrent,`
	`const float OriginalPermanent`

▸ After	`const EStatEnum TargetStat,`
	`const bool bResetCurrent,`
	`const float OriginalCurrent,`
	`const float OriginalPermanent`