3D Combo System

This system centers around three different types of ScriptableObject: MoveSet, Attack, and DamageObject.

To create these objects right-click in the project window and find them under Create/ComboSystem.

To edit the MoveSet double-click on the object.

Edit the Attack in the inspector right clicking on the ColliderTest bar to add or remove slider handles.

DamageObject will likely require you to create a derived class unless you only intend to do a set amount of unmodifiable damage with every hit. This can be done easily using the DamageOverride template found in Create/ComboSystem.

The last thing you’ll need is to add the ComboController component to your fighter, attach the relevant MoveSet and Animator, then assign the controls the fighter can use in the inputs section. You can also adjust how many colliders your attack can hit at any given frame with the hitColliderCount, adjust how long the player has inbetween inputs before the combo resets with ComboClearTimer and finally what your attack can hit with hitmask.

Note that whatever you want to be affected by the attacks need to have a component derived from IDamageTaker.

# MoveSet : ScriptableObject

* Protected basicMoves : [SimpleMove](#_SimpleMove) array
  + Contains all of the basic attacks
* Protected comboMoves : [ComboMove](#_ComboMove) array
  + Contains all of the possible combo attacks
* Public Initialize(ref PlayableGraph) : void
  + Sorts combo moves by complexity
  + Initializes all of the attacks
* Public ProcessCombo(KeyCode[], ref Attack) : int
  + Processes the KeyCode array
  + Sends out an Attack corresponding to the basic attack that matches the last key pressed
  + Returns an int index to the ComboMove that matches that array
* Public FindAttackId(int, KeyCode[]) : int
  + Returns the index of the attack within the found basic attack
* Public GetComboAttack(int) : Attack
  + Returns the combo attack at the relevant index

# SimpleMove

* Private attackKey : KeyCode
  + The key that this move is connected to
* Protected attack : [Attack](#_Attack_:_ScriptableObject)[]
  + the different attacks this move contains in order
* Public AttackCount : int
  + The number of [Attack](#_Attack_:_ScriptableObject)s in the attack array
  + get only
* Public AttackKey : KeyCode
  + Public accessor for attackKey
  + get only
* Public SimpleMove(): constructor
  + Empty as all variables are pre-initialized
* Public SimpleMove(KeyCode, [Attack[]](#_Attack_:_ScriptableObject_1))constructor
  + Initializes the variables to the parameters
* Public GetAttack(int) : [Attack](#_Attack_:_ScriptableObject)
  + Returns the attack from attack at the relevant index
* Public MoveTest(KeyCode) : bool
  + Returns true if the last key pressed meets the requirement of the move

# ComboMove

* Private comboString : KeyCode[]
  + the KeyCode array this move is connected to
* Private attack : [Attack](#_Attack_:_ScriptableObject)
  + The [Attack](#_Attack_:_ScriptableObject) this move contains
* Public AttackObj : [Attack](#_Attack_:_ScriptableObject)
  + The public accessor for attack
  + get only
* Public ComboStringLength : int
  + The length of the comboString
  + get only

* Public ComboMove(): constructor
  + Empty as all variables are pre-initialized
* Public ComboMove([Attack](#_Attack_:_ScriptableObject_1), KeyCode[]): constructor
  + Initializes the variables to the parameters
* Public MoveTest(KeyCode[]) : bool
  + Returns true if the last keys pressed meets the requirement of the move

# Attack : ScriptableObject

* Protected hitBox: Bounds
  + Hitbox the attack uses
* Protected damage: DamageObject
  + Damage of this attack
* Protected hitBoxActiveCheck: [AttackRange](#_AttackRange)
  + When, during the attack, the hitbox is active
  + Normalized from 0-1
* Protected attackLength: float
  + How long the attack lasts
* Protected attackAnimation: AnimationClip
  + Animation that will be played during the attack
* Public Damage: float
  + Public accessor for damage
  + get only
* Public AttackLength: float
  + Public accessor for attackLength
  + get only
* Public GetClip(): AnimationClipPlayable
  + Returns the AnimationClipPlayable for the PlayableGraph system
* Public virtual InitializeAnimation(ref PlayableGraph): void
  + Initializes the AnimationClipPlayable for the PlayableGraph system
* Public virtual PerformAttack(Vector3, ref Colliders[], float, int): int
  + Performs an Overlap box using the hitbox offset by the Vector3 parameter if the hitbox is active during that part of the attack
  + Returns the number of colliders hit
* Public DrawBoundsForDebug(): Bounds
  + Returns the hitbox if the attack is active for drawing

# AttackRange

* Private timeNodes: float[]
  + The points when the attack being active flips
  + Each of these are a percentile between 0 and 1 (mutually inclusive)
  + These are sorted from lowest to highest
* Public AttackRange(): constructor
  + Empty, values are pre-initialized
* Public AttackRange(float[]): constructor
  + Copies the nodes from the float[] parameter clamping values between 0 and 1 and sorting if necessary
* Public AttackColliderActive(float): bool
  + Returns whether the float parameter is larger than an even number (true) or an odd number (false) of time nodes

# ComboController : Monobehavior

* Public moveSet: [MoveSet](#_MoveSet_:_ScriptableObject)
  + The [MoveSet](#_MoveSet_:_ScriptableObject) this controller will use
* Public inputs: KeyCode[]
  + The KeyCodes this ComboController uses
* Protected hitColliderCount: int
  + The number of colliders this controller can hit in any given frame
* Protected comboClearTimer: Timer
  + How long until the timer is reset
* Protected hitableMask: LayerMask
  + The different layers this controller can hit with an attack
* Protected anim: Animator
  + This controllers animator
* Protected playableGraph: PlayableGraph
  + The PlayableGraph that will take over the Animator during an attack
* Protected playableOutput: AnimationPlayableOutput
  + The PlayableGraphOutput for use with the PlayableGraph
* Protected comboAttackActive: bool
  + Whether the current attack is a BasicAttack or a ComboAttack
* Protected currentCombo: List<KeyCode>
  + The current combo the player has hit
* Protected currentAttackTimer: Timer
  + The timer for the current attack
* Protected comboAttack: int
  + An index pointing to the relevant ComboAttack in moveSet
* Protected curAttack: [Attack](#_Attack_:_ScriptableObject_1)
  + The current [Attack](#_Attack_:_ScriptableObject_1) the controller is using
* Protected hitCols: Collider[]
  + The colliders that are returned when processing an attack
* Protected virtual ClearCombo(): void
  + Clears currentCombo and begins the ComboAttack if one has been found
* Protected virtual ManageAttack(): void
  + Ends the [Attack](#_Attack_:_ScriptableObject_1) if currentAttackTimer has finished
  + Processes the attack and if something has been hit, sned the collider and damage data to the [IAttacker](#_IAttacker)
* Protected virtual ManageInput(): void
  + When a key has been pressed determine if that key is in input then, if so, add it to currentCombo, process the new combo, call BeginAttack if an attack has been found
* Protected virtual BeginAttack(): void
  + Initializes the [Attack](#_Attack_:_ScriptableObject_1)
* Protected virtual EndAttack(): void
  + De-initialize the [Attack](#_Attack_:_ScriptableObject_1)

# IAttacker

* Public ProcessHits(int,Collider[], DamageObject): void
  + Shunts dealing damage to another script that will be on the same object as the ComboController