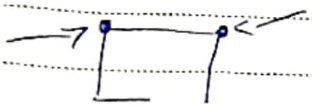
Subject
Date

Root motion warping (?)

motion warping set up → has Notify on itself

Adjust Root motion location



UMotionWarpingComponent::UpDate()

on all relevant MotionWarpingNotify State call OnBecomeRelevant

OnBecomeRelevant:

OnActivateDelegate.IsBound() add (RootMotion Modifier)

what lose bound to Delegates means (?)

motion warping component can iterate over Modifiers and call

ProcessRootmotion

a RootMotionModifier that we are gonna use is SkewWarp

in it's Process Root motion we will adjust the Rootmotion of the montage by target

Lose targets are passed to motionWarping Component

we will specify targets name and data { location, location and Rotation, Transform ...

Trace spect in space infront of us

disable some collision

Subject
Date

① Motion wraping plugin Enable it

⑥ vault over → Create Montage / add motion warping Notify state

during every Motionwarping Notify state we will inteplating to the target point that we will pass the motion warping component

⑤ specify Root motion modifier and add warp target name

JumpToTarget, JumpOverTarget — we access this target point by these nams
↓ ↓
2 target points and 2 Notify states

⑦ Add MotionwarpingComponent C++ class → to overide our component

AG_motionwarping component → in Actor Component

⑧ creat vault Ability from AG_gameplayAbility → GA_Vault
Ablity System → Ablities

⑨ movement Conponant — we want to Press Jump Button and our character perform the best Action for the circustases { Jump, JumpOver, ← Climb (Not in this lecture)

the selection depends on the envisoment

- create list of ablities that will be considerate for this Selection then initialize it from bueprint

١٨٧٠

PAPCO

Subject

Date

① add header to our character class and class def declare the property of motionwarp and a getter function for it

② create components

⑩ chang OnJumpAction Started and call our TryTraversal from movement componant

GA_valt

① constractor ⑨ override from Parent CommitCheck ?

this kind of backup activated in commit function and happen inside of Activate Ablity

we use it when we want to dedact the cost of an ablity like dedacte mana from spell (I think it's a calculation for safty)

we'll check if the environment is sutable or not and if yes we'll cach some target points that we add them during the activation pass to motion warping Component

ActivateAblity and EndAblity are copied from Crouch but they are override

Subject (17)
Date

we can get this location by getting the socket location from our MeshComponent because we just have the base class we can just use it and both staticMesh and SkeletalMesh support

GetSocketLocation

(11) on our character class we just need to handle the Inpute and send correct gameplay Event

OnAttackActionStarted, OnAttackActionEnded

in each of those functions we are going to sending gameplay Events. one for starting the Attack and one for Ending the Attack

for Now we just use 1 event just to Know when to start the ability but in future we might want to know when we ~~sp~~ stop firing for another Ability so we need two Events

(12) ——— Blueprint after creating the Blueprint BP-GA-SingleShot

- we create the specific Tag
- create an AbilityTrigger so this Ability to react to this specific gameplayEvent and be Activated and we are sending this event from our character with inPute
- override the the specifics functions that allows us to handle exactly the case when the Ability was Activated from the Event. because it Provides us the event Payload

Event Activate Ability from Event

on PlayMontageAndWait we are going to wait for will gameplay Event that we're are going to send from our AnimNotify on that event we will fire from our weapon if we had a melee weapon we wouldn't use this we create a custom Event and we connect it to Event Receive

---

for strafe movement we add an array of Effects to our Character. we want to this Effects be applied to our character when we equip that item

by default character movement component handles the movement rotation using a set of boolian variables but we are going to introduce a simple Enum for this option be more clear

the method that we are going to use gameplay Effect for rotating in our movement DirectionType. we would have an specific Tag that we'll apply to our character from that Effect and from our movement component we would subscribe to this tag and for all the cases that the amount of this tag applyed by some Effects is more than zero we would use strafe movement.
when we have zero amount of this tag we would apply the Defult orientation movement again

In our setter we also call ar internal function for Handling Movement Direction and in our Handle function we preform a simple switch
Also for initial settings we should also do it in BeginPlay

To Implement heap five we need a new variable in our weaponStati

we use GameplayEffect and data driven approach to apply damage

we Implement an BaseDamage UProperty for our weapon static because we don't want to have seperate gameplay Effect for every weapon in the game

we will be able to set dynamicaly the magnitod of the Effect to modify the health depending the baseDamage

Subject. (1)

Year. Month. Date. ( )

Inventory component

class created

Actor ItemActor → Actors

ItemActor

1. should be replicated

2. UProperty
   have pointer to itemInstance
   and we're going to use it to correctly create instance of an Item (spawn correct mesh or VFX or...) and it's going to be replicated and use that Instance to access ItemStaticData

3. should handle some basic functions that we previously created in our itemInstance and OnDropped()

4. have logic of ReplicateSubObject from Inventory component
   only bool wrotesomething
   return wrote something

5. it's a Replicated property so call that function on CPP
   #include "Net/UnrealNetwork"

6. create an Init function becuz we need to pass Item Instance to an Item actor that we recently created

```
void Init(UInventoryItemInstance* InInstance);
```

ما با این اون UProperty ای که داریم رو مقدار دهی می‌کنیم

Subject:
Year. Month. Date. ( )

(9) In order to ItemInstance can access the Item actor we go in UItemStaticData and add our class of Item actor we do that In case ItemInstance wants to create one

TSubclassOf<AItemActor> ItemActorClass;

(1) our InventoryList for adding Item is protected. so we also need some external way to add to our Inventory we use InventoryComponent and add the needed functions to it. we need some sort of external Interface

Void AddItem (TSubclassOf<UItemStaticData> InItemStaticData class);
Void RemoveItem( );

in the CPP we just use our InventoryList to add(AddItem) or remove (RemoveItem) from it

we also expose them to blueprint

(1) we also need 22 more functions one is for equiping and unequipping an Item

we can have diffrent signiture applyed here. like Equipper smt with an
Index
specific ItemInstance
specific ItemStaticDataClass
or somthing in future like asomthing from class of ranged weapon

for now We do it just like add and remove functions

(5)

Subject:
Year. Month. Date. ( )

we are going to iterate over InventoryList.GetItemRef() and access ItemInstance and call on Equipped(); or on UnEquipped and then call break;

(9) they are going to be some item that can be considered currently equipped and we want to store some sort of referance to it so that be able fire some events from it like if you equipped an riffle, you want to shoot from it and you want to this riffle to handle shooting Event so for this purposes we want to track currently Equipped Item

In case if it is NullPtr it means we have a zero equipped Item

UProperty(Replicated)
UInventoryItemInstance* currentItem = nullptr

add this to that function for uproperties that are Replicated

this pointer will be replicated and we don't need to do anything for because itemInstances already are being replicated

also
✓ 8. getter function for that pointer GetEquippedItem() const and also expoise it to Blueprint (pure

(10) also in onEquipped and onUnEquipped we set that pointer
item.ItemInstance

(11) add Implimetation in onEquipped and onUnEquipped of InventoryItemInstance
change at 15,10

PAPCO

(11) in OnEquipped we spawn an ItemActor and on OnUnEquipped we destroy an ItemActor and also save the refelance of the ItemActor in ItemInstance

(13) AItemActor* ItemActor is a replicated UProperty of InventoryItemInstance class so that function again

(14) in onRep_Equipped() we can do additional initialization or verifyication

or

for example if our Item is with a static mesh component, that static mesh component is replicated on itself

but some other components my not replicate what we need it to replicate

OnRep_Equipped()

So this is a place that you can verify and spawn the required things that are not handled by Replications of those components out of the box

(12) when we Equipp an Item we need to know that in which socket we should Equip it and so we add this as an additional property to our UItemStaticData

attachment

FName Socket = NAME_None; -> EditdefaultsOnly BlueprintReadonly

(18) we have to find our character and best way to do it is using InventoryComponent because it's owner is going to be our character and using that let our InventoryItemList to know who is the actual owner of the Inventory

we created our ItemInstances in our InventoryList

sometimes ItemInstance dosn't have an owner like when it's dropped

P4PCO

Subject

Date

(۱۷) We can use OnEquipped function of ItemInstance we called this function in Equipping function in Inventory Component

If we change that function to support an Actor later we can know we'll be equipped exactly by ~~the~~ which actor

virtual void onEquipped (AActor* InOwner = nullPtr);

Implimitation on ۲۹:۲۲

for now add these #includs GameFramework/Character.h
Components/SkeletalMeshComponent.h

(۱۷) so far for testing (because we don't have any specific ability for Equipping an Item or Unequipping one) we just go to InventoryComponent and in InitializeComponent() we Equip the first Item

the method is

Compile errors:

ActionGameTypes.h class AItemActor;

ItemActor.h class UInventoryItemInstance;

.cpp #include "Engine/ActorChannel.h"
"Inventory/InventoryItemInstance.h"

Subject

Date finalizing Inventory, Better Inventory tag management

1) InventoryItemInstanc.cpp → so it's our item

on we OnUnequipped add a replicated variable and use it in future wheter the instance actually equipped or not

bEquipped → onDropped ✗
OnUnEquipped ✗
OnEquipped ✓

2) ItemActor.h

we want to place Itemactor in the world and be able to equip it. right now we want to equip ~~this~~ an Item from ItemActor using InventoryItemInstance. in this case the Instance going to be null unless we create an Instance in Itemactor and for this's we need UProperty (EditdefaultsOnly) of TUItem<FStaticData> ItemStaticDataclass

3-1) set ↑ SetReplicateMovement(true) so all Physics or like the Trace we did be replicated and everyone be able to pick an Item from one place

3-2) ~~make~~ set false ~~ge~~ of SetGenerateOverlapEvents(false); of sPher Component in onEqui, OnUnequi, OnDrop ↓ set it to true

3-3 if this Item was placed in the world manoally and its Instance was not Initialized from the Inventory which is could have been dropped from. we are going to initalize it in the BeginPlay

we check if ItemInstance isn't valid and the static dataclass is valid

Subject

Date

~~Inventory Component~~

(3) extend our Inventory component Interface so it could Provide us functinality to equip items with item instances (Add and equip)

٢-٢ a function to equip the next item

Magic Trick ٢-٢

we are going to work with GamplayEvents and GameplayTags In inventory component without any ablities this would make this Inventory component a fully Independal system from any ablities it would Replicate and will handle gamplay Events

(1) virtual void GamplayEventCallback (const FGameplayEventData* Payload)

here we are going to work with gamplaytags because we have to send them inside of the gamplayEventPayload to actually Identify ~~them~~ the Event it's better to store them in a notherway than Initializing them as UPROPERTY from the Editor.

one of such ways is using Static variables

٢-٢ create a unifyed function that will handle any possible Event that would come to this component that we first subscribe and will listen for. and we'll do some logics in this function depending what kind of Gamplay tag coms with that Event

Subject
Date

(7-5) in order to make this Inventory component Independed from ablities, it means it'll have to replicate by itself and because we want to every action to be happening only under Authority we are going to add the ERPCG function in this case the one that'll execute on server. So it means we call it on our client if we need to request something and it'll execute on server

(7-6) add the necessary Includes and add static variables declare them again in .cpp because the declaration alone dosn't allocate memory for the static member in cpp

to Initialize that static variables we are going to have to add a GameplayTagsManager and assine it to our static variables we use GameplayTagManager for this an we'll subscribe to an specific event when we can do this

Subject:
Year Month Date ( )

۲-۸ in EquipNext we get our Items to the Local variables first and then have some conditions checked to if do Nothing we should do something even like if Inventory is empty or if one Item in the Inventory and we equipped it already

Now we add a TargetItem variable which in the end we aim to have the ItemInstance to equip.

and because some Items can't be equipped we'll have to Itelate over them to find the suitable one
Next

Also it has to be equipable and Don't be equil to our currentItem then we set our TargetItem and breaks

}

if currentItem be available and it be equil to targetItem we just return

remember we set our targetItem Initial value to current Item

on other conditions we call UnequipItem and Last of all out of all conditions we equip the TargetItem using EquipItemInstance function

> it means that Item can't be EquiPPed so our current Item should be UnEquiPPed

۲-۹ GamplayEvent Callback

we want to check if we have an Authority or Not and in case If we don't have an Authority it means we are sending this event on the client and we want to something to happen with our Inventory in this case we are going to Root our execution to the Server

to Try to do the action we wanted to do.

In case of Authority we just call our handle function with our Event Payload

(E) on HandleGameplayEventInternal we also do anything in case of an Authority
we are going to get gamplayTag from * GameplayEventCallback Payload and depending on what kinds of tag is that we are going to preform the required action

① EquipItemActorTag

① get InventoryItemInstance from the payload and this come from th ItemActor and if we had it we are going to add that Item
and because how Payload works we also have to call {Const_Cast<UInventoryItemInstance*>(ItemInstance)} to remove the constant from this object because we defenatly know & what it is and we can do it

Next we try to get itemactor itself and try to destroy it (from world)

Also for other tags we just call the respective function

(E.1) on ServerHandleGamplay it's a nice method over all the server is calling its own HandleGameplayEvent and thats works because we only Internal
call on this if ~~we~~ have ~~Authority~~ the right role

Subject (11)

Date

InitializeComponent Now we want to subscribe to this Gamplay Events. this can be done with AbilitySystemComponent that we trying to get from our ~~of~~ Owner

when we send a ~~G~~ameplayEvent to an Actor we actualy send that GameplayEvent to it's Ability Component

Inside of AbilitySystemComponent we have Generic ~~Event~~ GameplayEvent callbacks that we can subscribe to. to handle our GameplayEvents we do it for all 4 tags

(3) Now in ActionGameCharacter we need to add InPute handling functions for Inventory (we send ~~those~~ those Gamplay events on that Inputs

we use EventPayload with the Tag that represent the Event, ~~this tag passed~~ we can access this ~~the~~ Tag from our InventoryComponent static variable

(4) In ItemActor onSpherOverlap we first check for Authority then we send ourself (the ItemActor) as the Event Instigator (later with this we can delet that Item)

in OptinalObjects we are going to send ItemInstance that we had created potantually or recived after being dropped also specify the Event Tag as EquipItemActorTag

bugs and compile

add GameplayTags as an ~~a~~additional dependecies in ActionGameJ.Project

Inside the ItemActor where we're turning on the collision we should do this only in case it was initialy placed in a world other wise we are at risk to imidietly equip it from the overlap Event like when we are dropping it

also need to track ItemState Replication on the client because in the client we also need to handel collisions otherwise they would interfier with some abilities predictions

we can do it on OnRep_ItemState and inside of it we do a switch on the ItemState and handle the collision ourdiently for now we copy the server logic ~~with logic~~ is for collisions later we can what modify this aproach but at least we turn off the collision when we don't need it ~~with anything~~ on the client

Subject single shot Ability

Date

① Add a base class for any Abilities intended to worked with Inventory

from AG_ ~~Ability~~ GameplayAbility ~~Base~~ → GA_InventoryAbility

Path [Ability Systems → Abilities]

Next class is AnimationNotify that ~~she~~'ll able to send Gameplay Events to the owner we going to use this Notify to identify when exactly during the firing montage we actually need to cast the bullete

AnimNotify → AnimNotify_GameplayEvent

[Path AnimNotifies]

② after some declarations we are going to some Initialization and for that we use onGiveAbility

③ create helper functions to assist us to get very important things like weaponStaticData, ItemStaticData, ItemActors...

④ we only need 1 property and that's UInventory Component and we cach it and use it on our helper functions

⑤ on onGiveAbility we are going to get our Inventory component from our owner and save it to our Property

So this Implies At the moment we give this Ability we already have an InventoryComponent on our owner so this ability should be Instanced Per Actor it'll going to Exist on Actor

until it'll be cleared from it or until the actor is going to die

if in our game the Inventory Component is smt that is given at the runtime, you better using the OnActivateAbility() and OnEndAbility() to clean the Property you need

(4) create a getter for ItemActor in InventoryItemInstance

(5) on our type file we are going to add an AnimMontage Property because we need to play the firing animation for the Ability

(6) Ability will Play the montag but we usually send the fire event at a specific fram So now we Impliment our AnimNotify_GameplayEvent this Notify send a GameplayEvent to the owner Actor and while we have this Ability Active we will be able to wait for this gameplay Event and when we'll recive it we will exactly cast the bullete

(7) we can specify only the tag here but for this game we can specify the Payload too
Usually we want to fill some data inside of Payload at runtime. and we can expend this at the future and add the required Logics
we use UAbilitySystemBlueprintLibrary:: SendGameplayEventToActor

(8) we also might need a point where our weapon muzzole is we use that to spawn a Projectile at that location or spawn a vfx. or just take a point to witch to Preform a One Trace