https://www.udemy.com/course/unrealcourse/learn/lecture/31742896

Introduction

- In order to control our `Pawn` we need to use the `User Input`

Interesting links

https://nightails.com/2022/10/16/unreal-engine-enhanced-input-system-in-c/
https://docs.unrealengine.com/5.3/en-US/enhanced-input-in-unreal-engine/

https://www.youtube.com/watch?v=SlkpSJEmrgc **
https://www.youtube.com/watch?v=4wWIjlkSj2w

https://www.youtube.com/watch?v=j53dLKWihE0
https://www.youtube.com/watch?v=SlkpSJEmrgc

Side tutorial

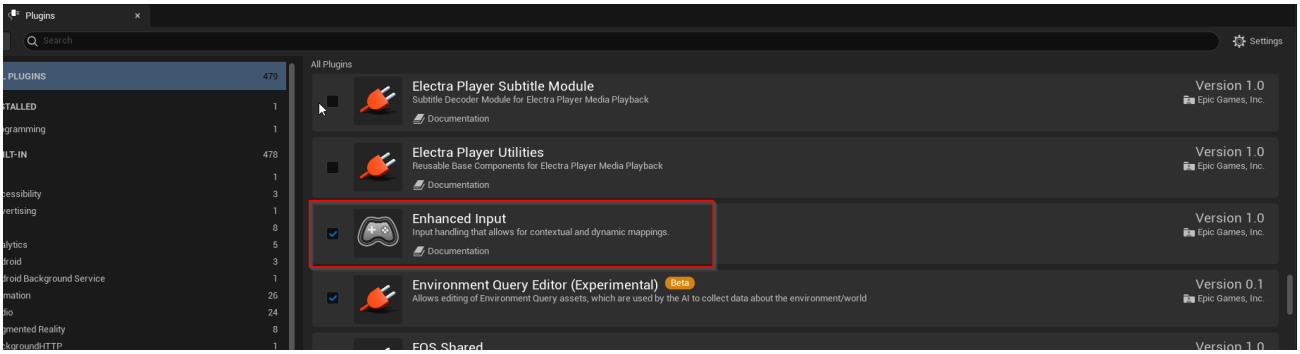
Following this one:
https://www.youtube.com/watch?v=fW1pXOAIviw

- 12:45 to 18:20 makes it in Blueprints, we can skip it if you are not interested in BPs

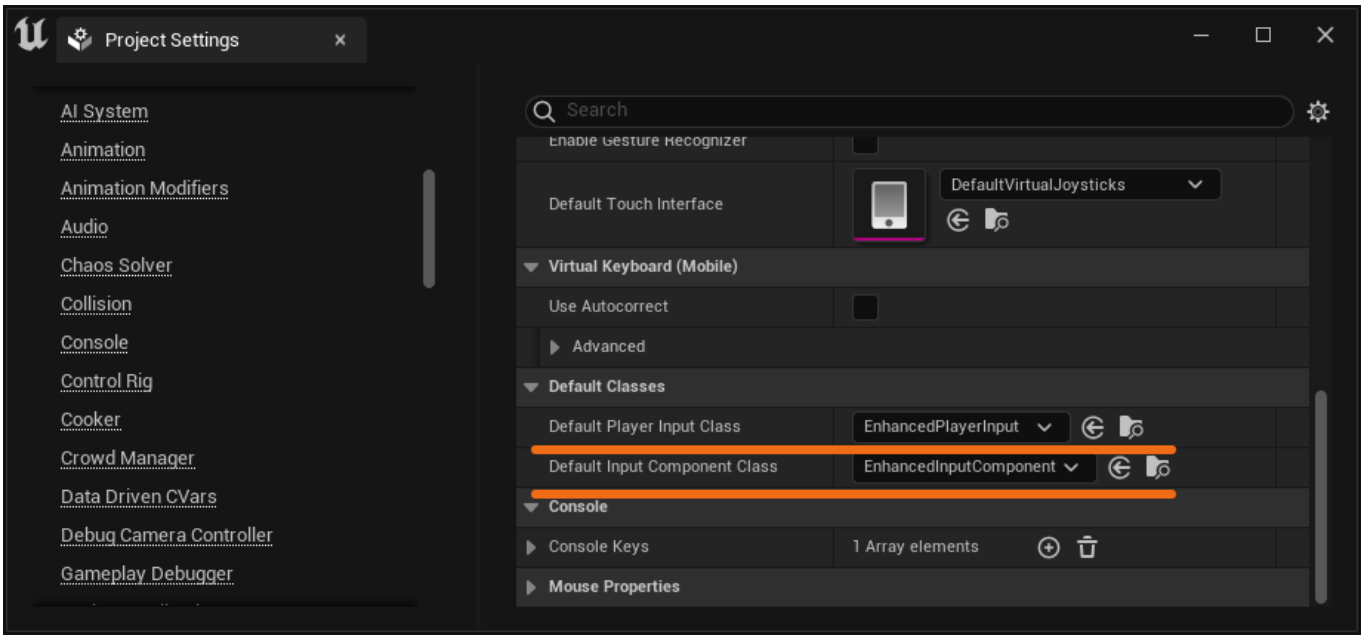
1 - Enabling "Enhanced Input"

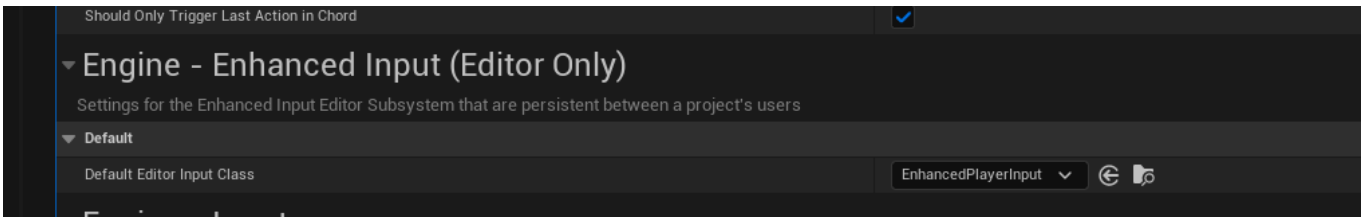
Edit/Plugins/Enhanced Input

- Activate and restart



With the editor open, go to `Edit->Project Settings->Input`. Then scroll down to **Default Classes**, change the **Default Player Input Class** from **PlayerInput** to **EnhancedPlayerInput**, and **Default Input Component Class** from **InputComponent** to **EnhancedInputComponent**.

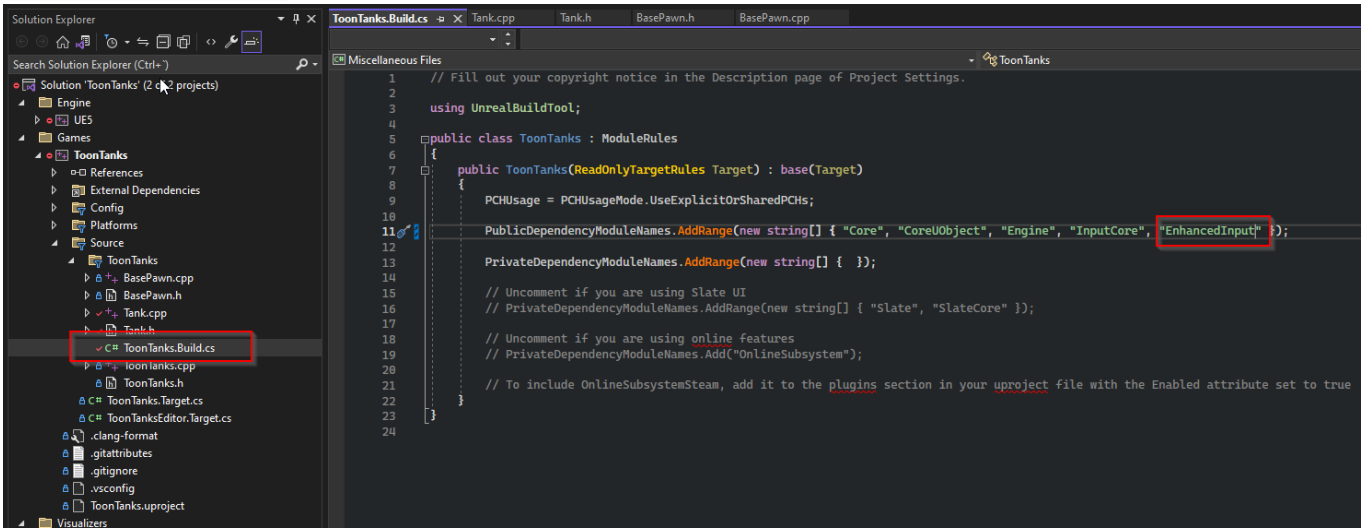




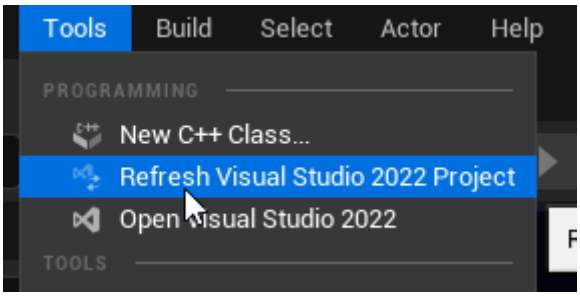

```
# There is no file named "InputActionValue.h"
```

```
did you forgot to include this on the <project>.build.cs file?
>       PublicDependencyModuleNames.AddRange(new string[] { "EnhancedInput" });
```

We need includes from `EnhancedInput`, so we need to add "EnhancedInput" to `ToonTanks.Build.cs`:

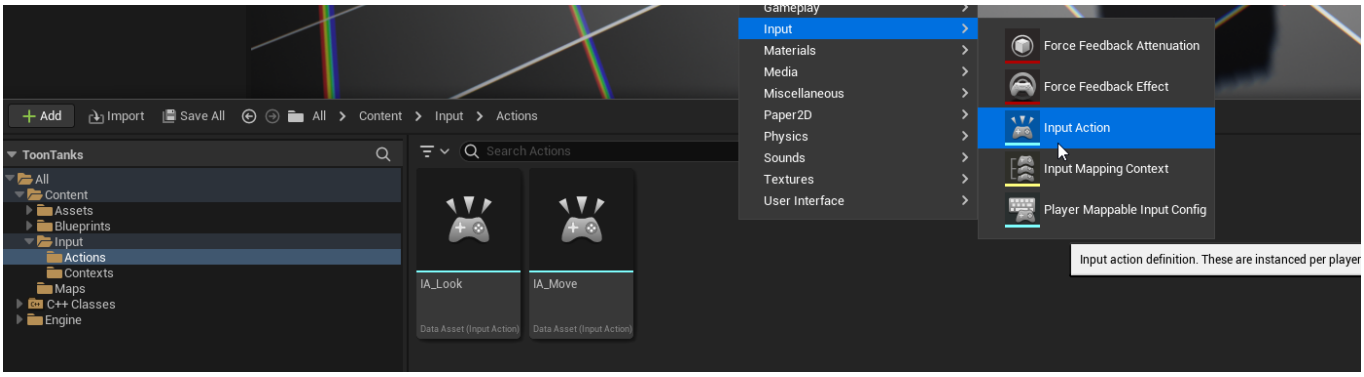


In order to take change we need to regenerate project and restart:
Regenerate project files.

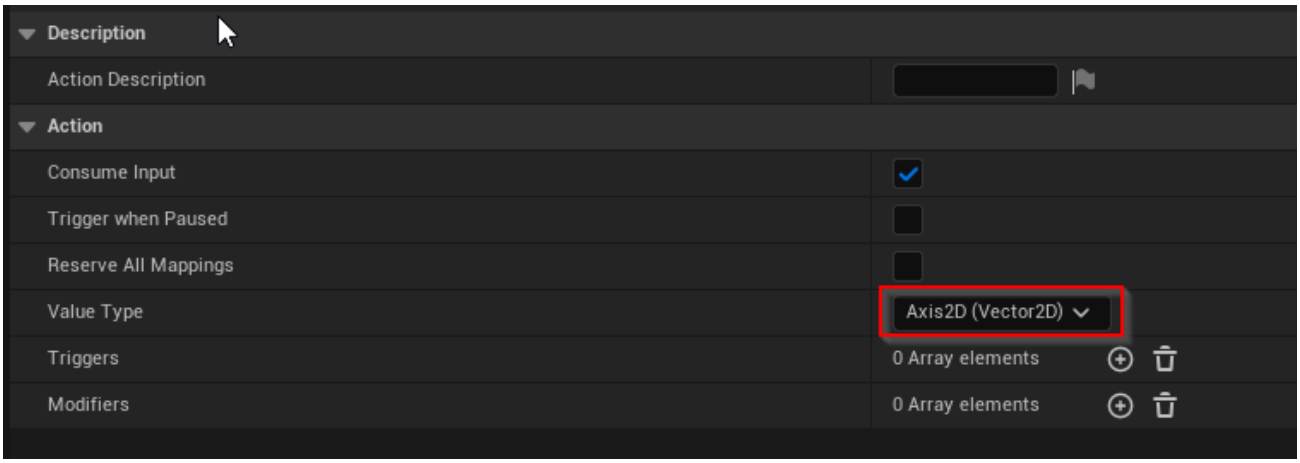


2 - Configuring the Inputs in UE5

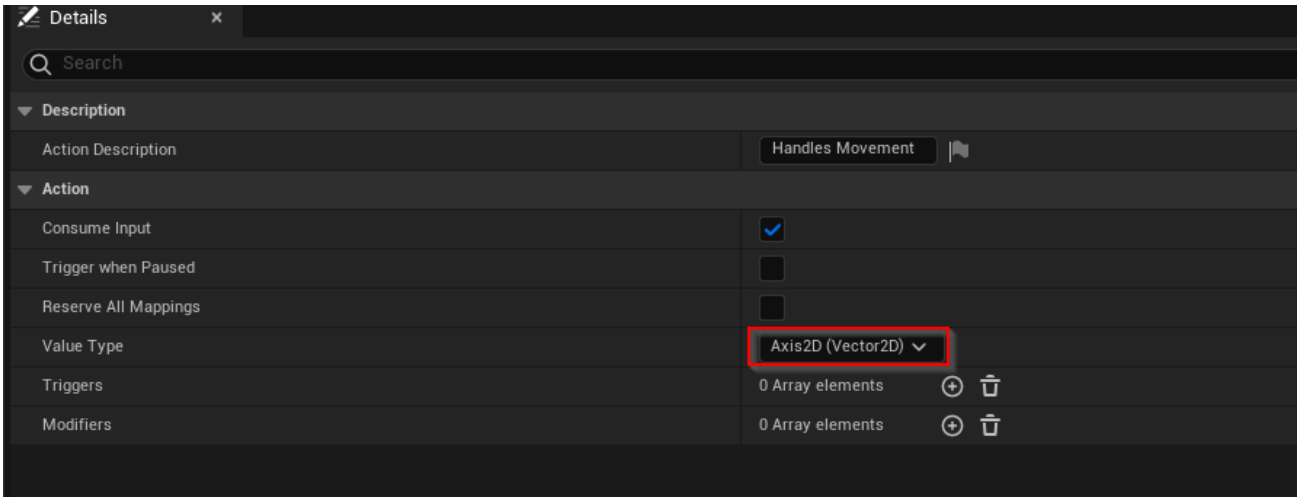
Create the input actions



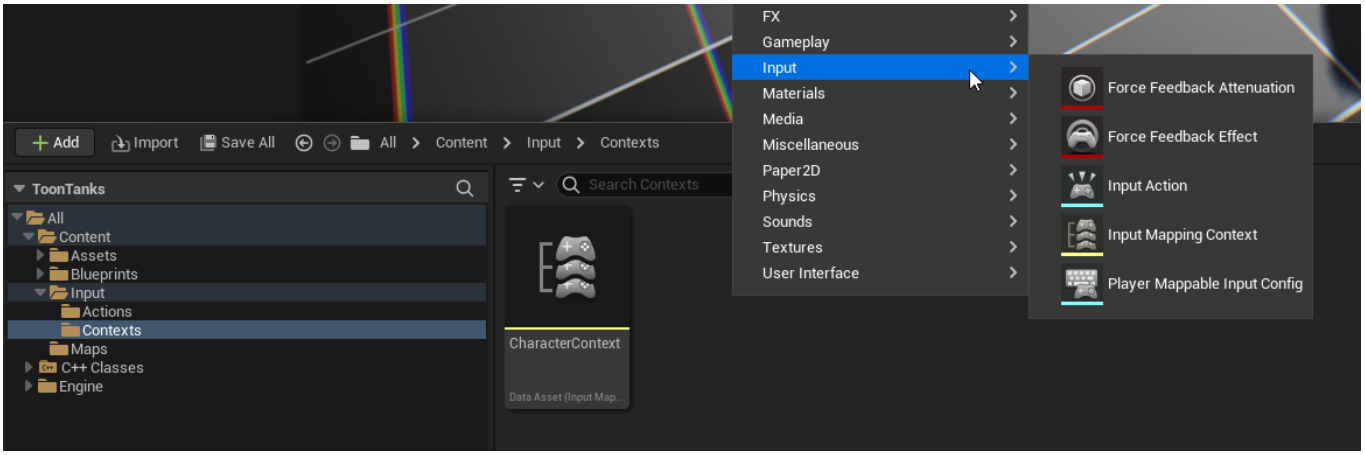
- IA_Look:
  Handle the cursor movement to look, in this case in a 2D movement and we need a 2D look.



- IA_Move:
  Handle the WASD movement to move, in this case in a 2D movement.



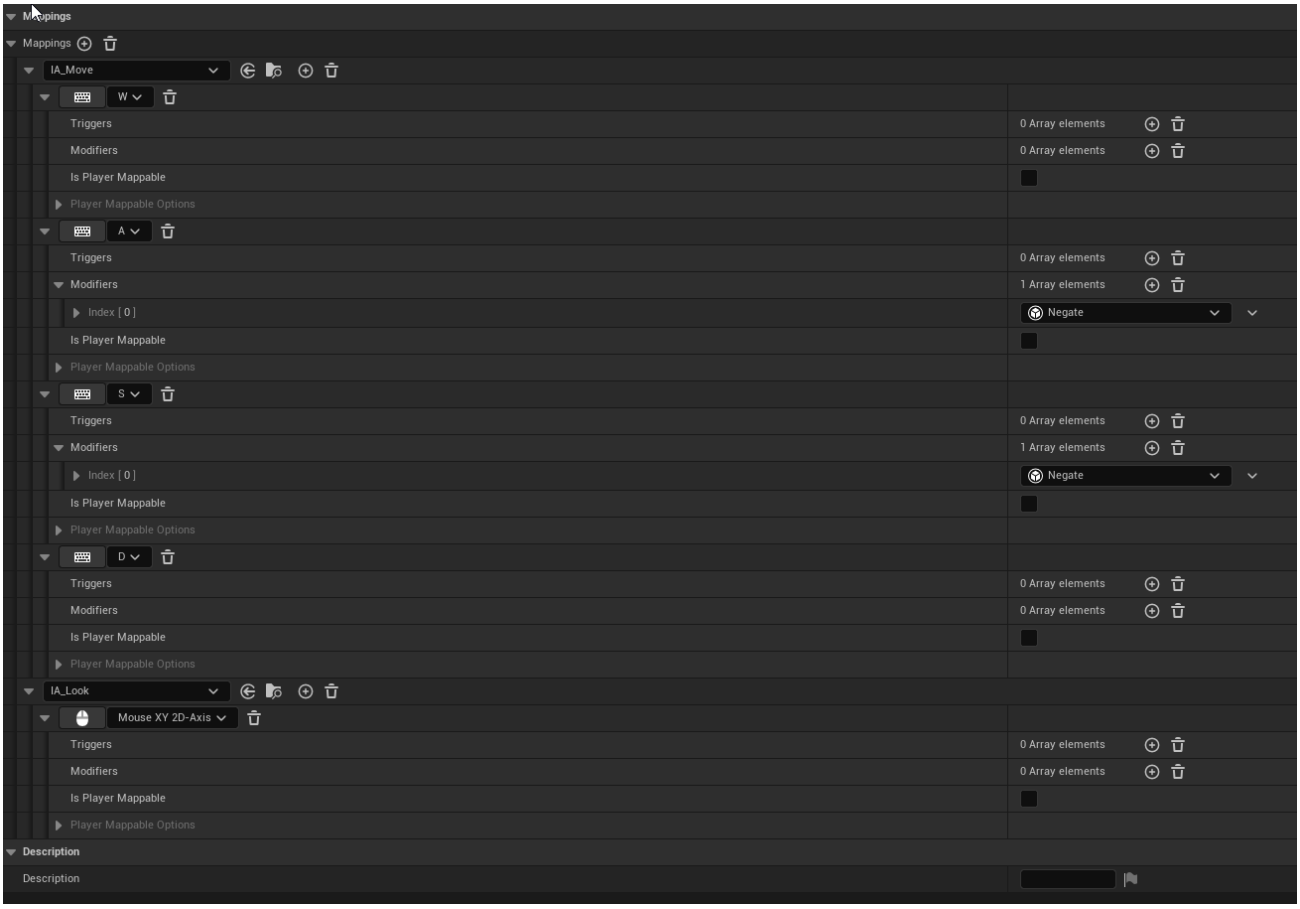
Create the mapping context



Here we are mapping each input to specific input action:

- IA_Look: To mouse XY 2D

- IA_Move: To WASD



NOTE: Modifiers to negate means that we are already receiving the input multiplied by -1

3 - SetupPlayerInputComponent (C++)

Setup UPROPERTY

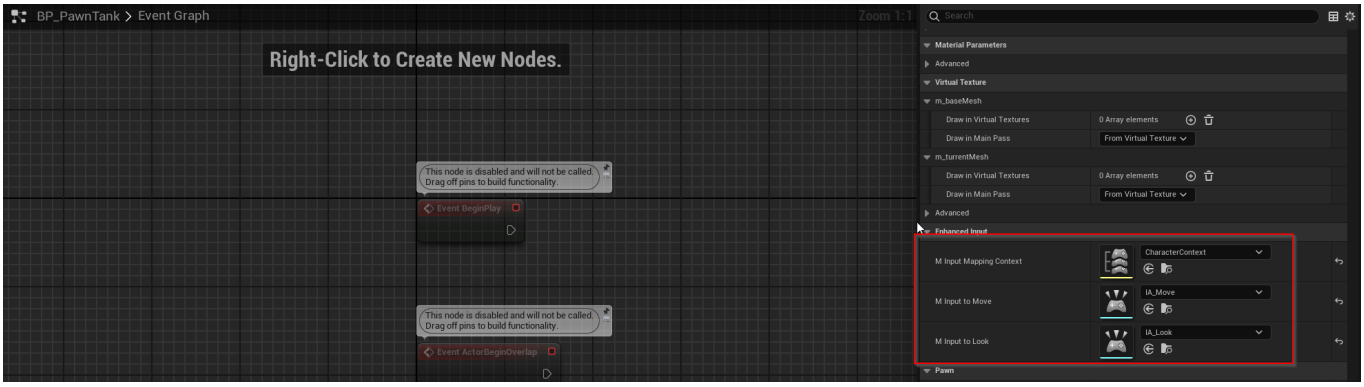
Prior to this we need to setup multiple `UPROPERTY` in order to set the components from the blueprint editor:

```
// Input
UPROPERTY(EditAnywhere, Category = "Enhanced Input")
class UInputMappingContext* m_inputMappingContext;

UPROPERTY(EditAnywhere, Category = "Enhanced Input")
class UInputAction* m_inputToMove;

UPROPERTY(EditAnywhere, Category = "Enhanced Input")
class UInputAction* m_inputToAim;
```

Where we will set the `Mapping Context` and the `Input Actions` (remember to compile):



Bind to code

Following this now we are going to bind it to code
In order to not setup this using blueprints, we need to configure the context on begin play for our Pawn:

```
void ATank::BeginPlay()
{
	Super::BeginPlay();

	// Setup MappingContext
	if (APlayerController* playerController = Cast<APlayerController>
(GetController())) // Get player controller
	{
		if (UEnhancedInputLocalPlayerSubsystem*
enhancedInputLocalPlayerSubsystem =
ULocalPlayer::GetSubsystem<UEnhancedInputLocalPlayerSubsystem>(playerController-
>GetLocalPlayer()))
		{
			enhancedInputLocalPlayerSubsystem-
>AddMappingContext(m_inputMappingContext, 0); // Add context with priority 0
		}
	}
}
```

NOTE: We are adding the mapping context that we are setting to our blueprint.

Following that we are going to bind to each input action a function that will consume it's data:

We can see that `ABasePawn` that inherits from `APawn` already has implemented `void ABasePawn::SetupPlayerInputComponent`

```
// Called to bind functionality to input
virtual void SetupPlayerInputComponent(class UInputComponent* PlayerInputComponent) override;
```

Which is also in `APawn` with no bindings:

```
/** Allows a Pawn to set up custom input bindings. Called upon possession by a PlayerController, using the InputComponent created by CreatePlayerInputComponent(). */
virtual void SetupPlayerInputComponent(UInputComponent* PlayerInputComponent) { /* No bindings by default.*/ }
```

So we need to override it also in our `ATank` class:

```
public:
    void SetupPlayerInputComponent(class UInputComponent* PlayerInputComponent) override;
};
```

```
// Called to bind functionality to input
void ATank::SetupPlayerInputComponent(UInputComponent* PlayerInputComponent)
{

}
```

We need to get the Enhanced Input component, this is possible because we configured the default classes in step 1.

```
// Get the EnhancedInputcomponent
UEnhancedInputComponent* enhancedInputComponent = Cast<UEnhancedInputComponent>
(PlayerInputComponent);
```

```
// Called to bind functionality to input
void ATank::SetupPlayerInputComponent(UInputComponent* PlayerInputComponent)
{
    Super::SetupPlayerInputComponent(PlayerInputComponent);

    // Get the EnhancedInputcomponent
    if (UEnhancedInputComponent* enhancedInputComponent = CastChecked<UEnhancedInputComponent>(PlayerInputComponent))
    {
        // Bind Actions
        enhancedInputComponent->BindAction(m_inputToMove, ETriggerEvent::Triggered, this, &ATank::EnhancedInputMove);
        enhancedInputComponent->BindAction(m_inputToAim, ETriggerEvent::Triggered, this, &ATank::EnhancedInputAim);
    }

}
```

NOTE: We are binding each input action that we are setting to our blueprint.

4 - Handling the FInputActionValue

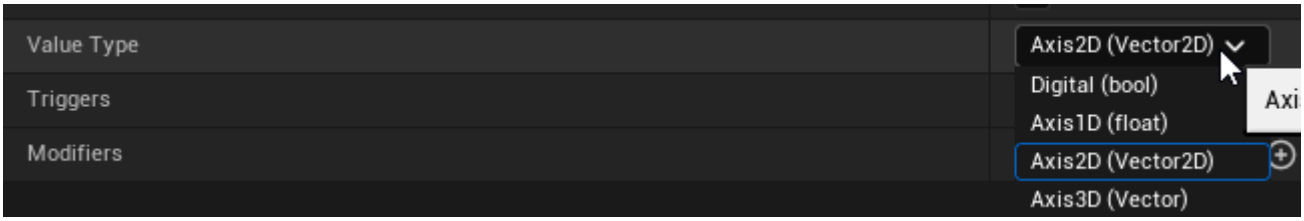
The callback functions shall handle `FInputActionValue`

```
void EnhancedInputMove(const struct FInputActionValue& i_inputActionValue);
void EnhancedInputAim(const struct FInputActionValue& i_inputActionValue);
```

But what is this value exactly?

This will correspond to whatever we are setting in UE InputAction, there are multiple

possibilities:



But we are handling always an `FInputActionValue`, this is because it has specialized constructors.

- So if we set and Axis2D and then we get an Axis1D, it will be all filled with `0.f`

```
// Specialized constructors for supported types
// Converting a value to a different type (e.g. Val = FVector(1, 1, 1); Val = true;) zeroes out any unused components to ensure getters continue to function correctly.
explicit FInputActionValue(bool bInValue) : Value(bInValue ? 1.f : 0.f, 0.f, 0.f),  ValueType(EInputActionValueType::Boolean) {}
FInputActionValue(Axis1D InValue) : Value(InValue, 0.f, 0.f),                       ValueType(EInputActionValueType::Axis1D) {}
FInputActionValue(Axis2D InValue) : Value(InValue, 0.f),                            ValueType(EInputActionValueType::Axis2D) {}
FInputActionValue(Axis3D InValue) : Value(InValue),                                 ValueType(EInputActionValueType::Axis3D) {}

// Build a specific type with an arbitrary Axis3D value
```

5 - Implement Binded functions

For the implementation of the functions we can now follow the udemy tutorial on minute 10 https://www.udemy.com/course/unrealcourse/learn/lecture/31742896 but it does nothing with the input, only prints it.

One more thing is that we need to cast our inputs to the type it should be implemented:

```
void ATank::EnhancedInputMove(const FInputActionValue& i_inputActionValue)
{
    const FVector2D moveVector = i_inputActionValue.Get<FVector2D>();
    const FRotator  moveRotation = FRotator(0.0f, GetController()->GetControlRotation().Yaw, 0.0f);
    //const FRotator    moveRotation = Controller->GetControlRotation();

    //{
    //  const FVector directionVector
    //}

    //AddMovementInput()
    UE_LOG(LogTemp, Display, TEXT("[%s|EnhancedInputMove] MoveRotation: %f,%f,%f deg"), *Super::GetName(), moveRotation.Roll, moveRotation.Pitch, moveRotation.Yaw);
    UE_LOG(LogTemp, Display, TEXT("[%s|EnhancedInputMove] MoveVector: %s"), *Super::GetName(), *moveVector.ToString());
}

void ATank::EnhancedInputAim(const FInputActionValue& i_inputActionValue)
{
    const FVector2D aimVector = i_inputActionValue.Get<FVector2D>();


    //if (GetController())
    //{
    //  AddControllerYawInput(aimVector.X);  // Left & Right
    //  AddControllerYawInput(aimVector.Y); // Up & Down
    //}

    UE_LOG(LogTemp, Display, TEXT("[%s|EnhancedInputAim] aimVector: %s"), *Super::GetName(), *aimVector.ToString());
}
```