Third Person Adventure Template TDD

Section 1 - Project Description

1.1 Project

Third Person Adventure Template

1.2 Description

This project is designed to act as a template and first step to creating Third Person games whether they are shooters, stealth or survival horror.

1.3 Development Plan

The development of this template will be ongoing for a long time, since new features will be added and thus doesn’t have a specific deadline. However, each game that comes out using this template will have its own Game Design Document with a more detailed Development Plan.

Contents

Section 2 - Project Structure

List of all Project Features and Gameplay Mechanics described with a quick summary.

2.1 Content/GameBase/Blueprints

Directory	Summary
Blueprints/HealthSystem	System that manages Health, Damage Response and Death events inside each Character or Actor.
Blueprints/InteractionSystem	System that usually links Players actions such as pressing a button to inspect something.
Blueprints/MainMenu	Takes care of displaying properly the Main Menu when in the Main Menu Level.
Blueprints/PlayerCharacter	Has a Base Character Class with fundamental behavior for a Third Person Game, such as Locomotion, Interaction and Health. This only references assets inside GameBase folder structure.
Blueprints/SaveGame	Manages data from the Main Menu, Pause Menu and important Player Data that needs to be stored for multiple play sessions.
Blueprints/WorldEventManager	System that manages World Events linked to anything that happens in-game. For example, picking up an item or defeating an enemy.
Blueprints/BP_TPA_GameModeBase	Used as a template for child classes.
Blueprints/BP_TPA_PlayerControllerBase	Ideally, takes care of Player Inputs.
Blueprints/BPI_TPA_WinCondtions	Interface implemented inside the GameMode that can be called from any Actor in the World, to initiate a win or lose condition.

Blueprints/ENUM_GameOverTypes	Various options for the cause of GameOver like running out of timer or being defeated by an enemy.

2.2 Content/GameBase/Audio

Here it is included all the important classes for the Audio System

- SC_TPA_Master and SM_TPA_Master
- SC_TPA_Music and SM_TPA_Music
- SC_TPA_SFX and SM_TPA_SFX
- SC_TPA_Voice and SM_TPA_Voice

2.3 Content/GameBase/UI

Contains all the Widget Blueprints that are used for all the Menus in the GameBase folders.

- WBP_TPA_MainMenu
- WBP_TPA_PauseGameMenu
- WBP_TPA_GameSettings
- WBP_TPA_LoadingScreen
- Confirmation screens
- Credit screens
- Menus for each Game Setting (Display, Graphics, Audio and Controls)

This will be converted into its own system with Base Widget Blueprints such as Button Collections and Menu Templates.

2.4 Content/EnhancedInput

New Unreal Engine Input System introduced in the 5.1 version of the engine. Right now it only includes mapping for KeyboardMouse and Gamepad controllers.

No special logic or implementation has been introduced yet.

2.5 Content/LevelPrototyping

Folder initially taken from the Unreal Engine Third Person template that includes assets like Static Meshes that can be used to build a prototype level really fast. Also, includes other general assets like weapon meshes or sounds that can be used for testing purposes only.

2.6 Content/AdvancedLocomotionCT

Special complex locomotion system for Third Person Character that was bought from the Unreal Engine Marketplace. This is the only big asset taken from the Marketplace that is being used for the Template since Animation and Complex Locomotion take a long time to make and have a satisfying system.

This asset has all the right elements for this template. Asset information in the reference section.
Reference 10.1

2.7 Project Settings And Plugins

At this moment, the Project is using the following Plugins:

- Animation Locomotion Library
- Animation Warping

These are new Plugins that are used for the latest locomotion system in the Lyra Game Project. It is used in the AdvancedLocomotionComponent System so they need to be Enabled in order for the system to work properly.

Project Settings to modify:

- Engine → Collision → Add InteractableObject Channel, set to Ignore
- Engine → Collision → Add AttackTrace Channel, set to Ignore
- Engine → Rendering → CustomDepthStencilPass from Enabled to EnabledWithStencil
  This is for the Interaction Material.
  It is also important to add this inside the Post Process Volume (PPV)
  Details → PostProcessMaterials

Section 3 - Health System

3.1 Health System Introduction

ActorComponent that is added to any Actor that we want to add Health properties to, like for example Maximum Health, Current Health, on death events, getting hit behavior, taking and doing damage etc.

In addition a Blueprint Interface was created to allow other classes to trigger damage, heal or check if the Actor is alive.

It's best if both are implemented in the desired Actor. Although, any actor can use the Interface to send damage or heal without the need of the component.

3.2 Health System Setup Guide

Step 1
Add BPC_HealthStats and BPI_HealthInterface to the desired Actor.
Both BPC and BPI should be implemented, for better customization.

Step 2
Implement all the BPI_HealthInterface functions using BPC_HealthStats functions.
This is just base functionality which can be expanded by implementing more logic inside these functions.

Step 4
Create a function SetupHealthComponent and from BPC_HealthStats bind events to OnDeath, OnDamageBlocked and OnDamageResponse.
Create these events inside the Owners EventGraph and bind them properly.
Now you can implement your desired logic inside each event like ending the game OnDeath.
Call this function on BeginPlay.

Implementation Finished

Useful Tip
Use the BPI_HealthInterface Message for TakeDamage inside any Actor you want to damage ThePlayer with and make a ST_DamageInfo.

Useful Tip
You can use the BPI_TPA_WinConditions to call the GameMode function for a GameOver OnDeath.

3.3 Health System Technical Details

Each asset inside the Content/GameBase/Blueprints/HealthSystem will be explained in this section.

3.3.1 BPC_HealthStats

ActorComponent that must be added to any Actor that we want to react to any events regarding Health, like Healing, Death Events, Damage Events etc.

VARIABLES

Name	Summary
Health	**Type** float Current health of the Actor.
MaxHealth	**Type** float Maximum health quantity the Actor can achieve.
IsAlive	**Type** bool Used to check if the Actor is still alive or not.
IsInvincible	**Type** bool Used to check if the Actor can receive any kind of damage.
IsInterruptible	**Type** bool Used to check if the Actor can be interrupted, for example during an attack.
IsBlocking	**Type** bool Used to check if the Actor is blocking to avoid taking damage, reduce damage or check for parry attack behavior.

FUNCTIONS

Name	Summary
EventTick()	Used for Debug purposes only. Tick is usually disabled.
Heal()	**Input** HealthAmount (float) **Output** NewHealth (float) Usually called from any Actor using the HealthInterface. Checks if Actor IsAlive the proceeds to add HealthAmount to the Health variable. Clamps the float value between 0.0 and MaxHealth.
TakeDamage()	**Input** ST_DamageInfo (struct) **Output** DamageSuccesful (bool) Usually called from any Actor using the HealthInterface. Checks if the Actor CanBeDamaged and calls the proper EventDispatcher to decide if the Actor was blocking, is dead or any kind of damage response.

MACROS

Name	Summary
CanBeDamaged	**Input** ShouldDamageInvincible (bool) CanBlock (bool) **Output** BlockDamage (Execution Pin) DoDamage (Execution Pin)

	NoDamage (Execution Pin) Since there are many bool cases to take into account for a damage event, we use this Macro to take them all into account and execute the proper branch.

EVENT DISPATCHERS

Name	Summary
OnDeath	**Input** None Should be implemented inside each Actor Owner. Handles OnDeath event.
OnDamageBlocked	**Input** CanBeParried (bool) Should be implemented inside each Actor Owner. Handles what happens on Block and Parry events.
OnDamageResponse	**Input** DamageResponse (ENUM) ENUM made specifically to describe multiple types of damage where the Actor Owner can react differently to.

3.3.2 BPI_HealthInterface

Interface that allows any Actor to communicate with Owners of the BPC_HealthStats component.

No need to implement the interface on any actor, just send a message.

Actors that use the BPC_HealthStats should implement BPI_HealthInterface for normal behavior and also custom specific behavior when applicable.

Important: All the functions described below need to be implemented inside each owner of the BPC_HealthStat component, using the BPI_HealthInterface.

FUNCTIONS

Name	Summary
GetCurrentHealth()	**Output** Health (float) Returns BPC_HealthStat Health variable.
GetMaxHealth()	**Output** MaxHealth (float) Returns BPC_HealthStat MaxHealth variable.
Heal()	**Input** HealAmount (float) **Output** NewHealth (float) Calls BPC_HealthStats Heal function and returns the new health after healing.
TakeDamage()	**Input** DamageInfo (struct) **Output** DamageSuccesful (bool) Calls BPC_HealthStats TakeDamage function and returns if damage was successful. Note: Should Bind Event Dispatchers from the BPC_HealthStats on the owner as well.
GetIsAlive()	**Output** MaxHealth (float) Returns BPC_HealthStat IsAlive variable.

3.3.3 ENUM_DamageResponse

Specifies how the BPC_HealthStats Owner will react to a type of damage. Can be Stagger, Stun, Knockback etc.

A Switch can be implemented when calling the Event OnDamageResponse.

This is used inside the ST_DamageInfo struct and the BPC_HealthStats.
And also used inside every actor using the BPC_HealthStats or creating a ST_DamageInfo.

3.4.4 ENUM_DamageResponse

Specifies how the BPC_HealthStats Owner will react to a type of damage. Can be Stagger, Stun, Knockback etc.

A Switch can be implemented when calling the Event OnDamageResponse.

This is used inside the ST_DamageInfo struct and the BPC_HealthStats.
And also used inside every actor using the BPC_HealthStats or creating a ST_DamageInfo.

3.5.5 ENUM_DamageType

Specifies if the damage is environmental, melee, projectile etc.

This is used inside the Actor that creates a ST_DamageInfo to create damage.

3.6.6 ST_DamageInfo

When another Actor wants to deal damage to another actor (with BPC_HealthStats) the actor will have to make a ST_DamageInfo and specify the parameters of the Attack.

Like if the attack can be blocked, parried, how much damage it does and what type of damage is it.

Section 4 - Interaction System

4.1 Interaction System Introduction

The purpose of this system is to allow ThePlayer Character to be able to interact with Actors in the World.

This is done by using the BPC_InteractionTraceTP, which is a custom Trace behavior for Third Person games and also the BPI_InteractionInterface which will allow ThePlayer to trigger specific behavior on the BP_InteractableBase.

With the help of a BoxCollision inside ThePlayer Character, ThePlayer will be able to detect close proximity Interactables and be able to Interact with them, choosing the closest one as priority.

Every InteractableObject needs to implement the BPI_InteractionInterface.

At this time it is not tested whether an NPC can use this system properly.

4.2 Interaction System Setup Guide

Step 1
Add a BoxCollision in the PlayerCharacter Base class, scale it and position it properly.

Step 2
In the BoxCollision Details → Collision, make sure to only Overlap with InteractableObject. Ignore everything else.
If you don't have InteractableObject ObjectChannel, add it in the ProjectSettings, set to Ignore.

Step 3
Add the BPC_InteractionTraceTP component.
In the BeginPlay from BPC_InteractionTraceTP call SetPlayerInteractionBox and use the BoxCollision as Input.

Step 4
From BoxCollision add the OnComponentBeginOverlap and use it to enable Tick of BPC_InteractionTraceTP
(It gets disabled when it isn't needed anymore).

Step 5
If you have an EnhancedInput for Interaction, add the Event.
From BPC_InteractionTraceTP → GetFocusedActor
If Valid, call the BPI_InteractionInterface Message OnInteract.
For Caller you can call Self.

Step 6
Create an InteractableBase → BP_InteractableBase
You can either use StaticMesh or any collision for the component.

There needs to be one collision that in the Details → Collision is of Type InteractableObject.

Step 7
Add the BPI_InteractionInterface. Implement all the functions.

Implementation Finished

Useful Tip
Using Event StartFocus and EndFocus we can use SetRenderCustomDepth and SetCustomDepthStencilValue to show a highlight on the object we are focused on.
Also, we can show a Widget to show the correct EnhancedInput to press to Interact.

Useful Tip
Using the Event OnInteract we can do a lot of things for the Interactable.
Adding to Inventory, DestroyActor, trigger WorldEvent, etc.

Useful Tip
Create Child Classes from BP_InteractableBase to create diverse types of interactables

Important: Keep in mind we know an Interactable is an InteractableObject, not because it has the Interface or Component, but because it is of ObjectType InteractableObject.

This might need to change in the future.

4.3 Interaction System Technical Details

Each asset inside the Content/GameBase/Blueprints/InteractionSystem will be explained in this section.

4.3.1 BPC_InteractionTraceTP

ActorComponent that must be added to ThePlayer, alongside with a BoxCollision so we can trace for any Interactable objects using overlaps, and focusing on the closest one.

VARIABLES

Name	Summary
FocusedActor	**Type** Actor Interactable Object that is closest to ThePlayer.
PlayerInteractionBox	**Type** PrimitiveComponent BoxCollision that belongs to ThePlayer. Needs to be set up at BeginPlay on the Player Class.

FUNCTIONS

Name	Summary
EventTick()	EventTick will be Enabled when we overlap with an Interactable Object. It will be disabled when there are no Interactable Objects in the BoxCollision. Used for calling GetClosestInteractable() when PlayerInteractionBox is valid.
SetPlayerInteractionBox()	**Input** PlayerInteractionBox (PrimitiveComp) Used in BeginPlay inside Player Class alongside the BoxCollision to define the PlayerInteractionBox variable.
GetFocusedActor()	**Output** FocusedActor (Actor)

	This is used in ThePlayer Class to validate if there is a FocusedActor so we can Interact with it on Input.
ClearPreviousInteractable()	Calls BPI_InteractionInterface Message EndFocus to make sure to switch properly to another Interactable or disable all interaction when there are no objects inside the collision anymore.
GetClosestInteractable()	This happens on EventTick (every frame) From the PlayerInteractationBox we get all OverlappingActors. Do a ForEachLoop and calculate the GetDistanceTo the Owner. Compare from previous calculation so we can define a new ClosestInteractable. Then, at the end of the loop send a Message to StartFocus.

4.3.2 BPI_InteractionInterface

Interface that allows ThePlayer and other Actors to send Messages to InteractableObjects to trigger them or interact with them.

This only gets implemented on InteractableObjects. Once it's ready, it allows each InteractableObject to have similar or custom behavior on Interact and while Focused.

FUNCTIONS

Name	Summary
OnInteract()	**Input** Caller (Actor) Interact with InteractableObject. The Caller is the Actor interacting with this object, usually ThePlayer on the press of an Input. Can be used as a Message from other Actors as well.
StartFocus()	Usually used inside BPC_InteractionTraceTP to highlight the

	ClosestInteractable. Can be used for other purposes as well.
EndFocus()	Similar to StartFocus() it is used to end the highlight of the ClosestInteractable. Can be used for other purposes as well.

3.3.3 BP_InteractableBase

Parent class for most InteractableObjects.

This has the BPI_InteractionInterface implemented with logic in StartFocus and EndFocus that allows the InteractableObject to be highlighted and make the InteractionWidget visible to ThePlayer.

3.4.4 WBP_InteractionWidget

Simple Widget Blueprint that gets revealed and hidden during Start/End Focus events.

Important: A good idea maybe would be to change this to be not a letter but something more creative.

3.5.5 PP_InteractOutline

Post-process material that needs to be added in the Post Process Volume (PPV) of the Level. PPV → Details → PostProcessMaterials

Section 5 - Main Menu System

5.1 Main Menu System Introduction

Since the Main Menu is a system that will always be present in a videogame in some form, it's very useful to have a template for it.

For this Main Menu System we need:

- Level for Main Menu (LVL_TPA_MainMenu)
- GameMode (BP_TPA_GameMode_MainMenu)
- HUD (BP_TPA_HUD_MainMenu)
- GameInstance (BP_TPA_GameInstance)
- Widget Blueprints

This setup allows the Main Menu System to be independent and easily migratable to other projects.

Important: A development for better UI tools is needed so we can have Widget Blueprint templates like Custom Button Collections and Sub Menu Templates.

5.2 Main Menu System Setup Guide

Step 1
Create Level for main menu. LVL_TPA_MainMenu.

Step 2
Add the GameMode_MainMenu.
Set the DefaultPawn Class to None.

Step 3
Add the HUD Class to the GameMode_MainMenu

Step 4
Inside the HUD Class, BeginPlay call the ShowMainMenu function from the GameInstance.
The GameInstance will take care now of switching between menus.

Implementation Finished

In this case the GameInstance will be shared with the MainMenu and SaveGame System since there is a lot of variables we need to save in the MainMenu and the GameInstance already has a SaveGame System in mind.

5.3 Main Menu System Technical Details

Each asset inside the Content/GameBase/Blueprints/MainMenu and UI will be explained in this section.

5.3.1 BP_TPA_GameMode_MainMenu

Standalone GameMode created just for the MainMenu Level. No logic is implemented here.
DefaultPawnClass should be None
HUD Class should be the BP_TPA_HUD_MainMenu

5.3.2 BP_TPA_HUD_MainMenu

Blueprint that will take care of calling the GameInstace and showing the MainMenu on ThePlayer's HUD.

FUNCTIONS

Name	Summary
BeginPlay	Casts to BP_TPA_GameInstance and calls ShowMainMenu. Displays Main Menu at the start of the Main Menu level.

Also, custom functionality can be added here like adding a new ViewTarget with a camera in the Level to make a more dynamic 3D Main Menu.

5.3.3 BP_TPA_GameInstance

GameInstance used for all the project.
Needs to be setup in the ProjectSettings → Maps&Modes → GameInstanceClass

This Blueprint is used as an Interface for the Main Menu and Pause Menu Systems. So it is important to create Custom Events for every UI event we want, like creating the MainMenu WBP and adding it to the viewport.

5.3.3 WBP_TPA_MainMenu

Main Widget Blueprint that represents the Main Menu of the game and takes care of displaying all the sub menus that come from, Game Settings, Tutorial, Credits etc. As well as loading the proper level when starting to Play.

Section 6 - World Event Manager System

6.1 World Event Manager System Introduction

This World Event Manager System was born because of the complexity that it would take to bring this to the Interaction System and also to put specific logic to just a Button in order to communicate with other Actors in the World.

This way, the World Event Manager component can be added to any Actor that is designed to trigger an event in the World, from Buttons to invisible collisions that spawn sounds.

This simplifies the whole process of adding unnecessary logic to the InteractionSystems and adding similar logic to different classes all the time.

6.2 World Event Manager System Setup Guide

Step 1
Add the BPC_WorldEventManager to the desired class.

Step 2
Inside the Actor BP, from BPC_WorldEventManager call TriggerWorldEvent.

Step 3
Inside your target class or Actor implement the BPI_WorldEventManager Interface function WorldEventTrigger.

Step 4
In the World, select your Actor with the BPC_WorldEventManager.
In the Details, select the component and select your target actors or classes to affect.

6.3 World Event Manager System Technical Details

Each asset inside the Content/GameBase/Blueprints/WorldEventManager

6.3.1 BPC_WorldEventManager

Goes inside any Actor that is going to trigger Events on other actors or spawn new Actors.

Needs to call TriggerWorldEvent at some point inside the Owner.

VARIABLES

Name	Summary
EventType	**Type** ENUM_EventType Describes the Type of Event that will be triggered. For example, SpawnActor, TriggerActor, etc.
ActorsToSapwn	**Type** Actor Class (Array) Classes that will be spawned on EventTrigger.
SoundsToSpawn	**Type** SoundBase (Array) Sounds that will be spawned on EventTrigger.
ActorsToTrigger	**Type** Actor Soft Reference (Array) Actors that should be in the World that will trigger a behavior on EventTrigger
ToogleVisibilityActors	**Type** Actor Soft Reference (Array) Actors that will have their Visibility toggled on EventTrigger.

FUNCTIONS

Name	Summary
TriggerWorldEvent	**Input** EventType (ENUM_EventType) Called from the Owner of BPC_WorldEventManager and specifies the Type of event so we can use a Switch and handle the proper behavior.

6.3.2 BPI_WorldEventManager

Takes care of communicating the BPC_WorldEventManager with any Actor in the World by sending the message function.

FUNCTIONS

Name	Summary
WorldEventTrigger	**Input** Caller (Actor) This gets called inside the BPC_WorldEventManager when the EventType is TriggerActor. Caller will be the Owner of the Component.

6.3.3 ENUM_EventType

Description of multiple types of events that can be handled in the World Event Manager. For example:

- SpawnActor
- SpawnSound
- TriggerActor
- ToggleVisibility

More should be added in the future.

Section 7 - Save Game System

7.1 Save Game System Introduction

Right now the SaveGame is only here to save the Game Settings like Audio Settings and Control Settings.

It will be developed so it can handle Checkpoints and Interactables that have been picked up and more.

6.2 Save Game System Setup Guide

Step 1
Create BP_TPA_SaveGameSettings

Step 2
Check for DoesSaveGameExist
And CreateSaveGameObject or LoadGameFromSlot
This goes in the correct Blueprint, such as WBP_TPA_AudioSettings or the GameMode.

Section 8 – References

8.1 Advanced Locomotion Component v2.4 By CT Game
https://www.unrealengine.com/marketplace/en-US/product/advanced-locomotion-component

8.2 Lyra Starter Game By Epic Games
Lyra animations and mechanics were used for this project
https://www.unrealengine.com/marketplace/en-US/product/lyra

8.3 Lock-On 360 By Zennodez Arts
https://www.unrealengine.com/marketplace/en-US/product/lock-on-360

8.4 Advanced Dialogue System (Replicated) By Altas Studio
https://www.unrealengine.com/marketplace/en-US/product/advanced-dialogue-system-replicated

8.5 Dynamic Combat System - Guns By Grzegorz Szewczyk
https://www.unrealengine.com/marketplace/en-US/product/dynamic-combat-system-guns

8.6 Main Menu System By J.Kaxob
https://www.unrealengine.com/marketplace/en-US/product/main-menu-system-01

8.7 Blockout Starter Pack By Xavier Loux
https://www.unrealengine.com/marketplace/en-US/product/blocking-starter-pack

8.8 Assetsville Town By Assetsville
https://www.unrealengine.com/marketplace/en-US/product/assetsville-town

8.9 Stylized Cyberpunk Girl Pack By VOYAGER 3D
https://www.unrealengine.com/marketplace/en-US/product/stylized-cyberpunk-girl

8.10 Melee Combo Vol 1 By ZzGERTzZ
https://www.unrealengine.com/marketplace/en-US/product/melee-combo-vol-1

8.11 The Immersive Template By DYLO Gaming
https://www.unrealengine.com/marketplace/en-US/product/immersive-template