### Enhancing the Quest System

We'll add the following features to the quest system:

1. Quest Caching: Use TMap to cache quests for quick lookup.
2. Timed Quests: Use timers to handle quests with time constraints.
3. Advanced Data Structures: Use TMap to store and manage quests efficiently.
4. Event Handling: Improve event handling using delegates and broadcasting updates.

### Updated Quest Structure

We'll use the existing FQuest structure but you can extend it with more fields if needed:

cpp

Copy code

#pragma once

#include "CoreMinimal.h"

#include "Engine/DataTable.h"

#include "Quest.generated.h"

USTRUCT(BlueprintType)

struct FQuest : public FTableRowBase

{

GENERATED\_BODY()

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Quest")

FString Name;

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Quest")

FString Description;

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Quest")

bool bIsCompleted;

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Quest")

TArray<FString> Objectives;

// Time in seconds for timed quests, -1 for non-timed quests

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Quest")

float TimeLimit;

// Default constructor

FQuest()

: Name(TEXT("Default Quest")), Description(TEXT("Default Description")), bIsCompleted(false), TimeLimit(-1.0f) {}

};

### Updated Quest Manager Component

We’ll add caching, a timer system, and efficient quest management with TMap.

#### QuestManagerComponent.h

cpp

Copy code

#pragma once

#include "CoreMinimal.h"

#include "Components/ActorComponent.h"

#include "Engine/DataTable.h"

#include "Quest.h"

#include "QuestManagerComponent.generated.h"

DECLARE\_DYNAMIC\_MULTICAST\_DELEGATE\_OneParam(FOnQuestUpdated, const FQuest&, CurrentQuest);

UCLASS(ClassGroup=(Custom), meta=(BlueprintSpawnableComponent))

class YOURGAME\_API UQuestManagerComponent : public UActorComponent

{

GENERATED\_BODY()

public:

UQuestManagerComponent();

protected:

virtual void BeginPlay() override;

public:

UPROPERTY(BlueprintAssignable, Category = "Quest")

FOnQuestUpdated OnQuestUpdated;

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Quest")

TArray<FQuest> ActiveQuests;

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Quest")

UDataTable\* QuestDataTable;

UFUNCTION(BlueprintCallable, Category = "Quest")

void AddQuest(const FQuest& NewQuest);

UFUNCTION(BlueprintCallable, Category = "Quest")

void CompleteQuest(const FString& QuestName);

UFUNCTION(BlueprintCallable, Category = "Quest")

void AddAllQuestsFromDataTable();

UFUNCTION(BlueprintCallable, Category = "Quest")

void StartTimedQuest(const FString& QuestName);

private:

TMap<FString, FQuest> QuestCache;

FQuest CurrentQuest;

FTimerHandle TimerHandle;

void UpdateCurrentQuest();

void CacheQuest(FQuest Quest);

FQuest\* GetCachedQuest(FString QuestName);

void OnTimedQuestCompleted();

};

#### QuestManagerComponent.cpp

cpp

Copy code

#include "QuestSystem/QuestManagerComponent.h"

#include "TimerManager.h"

UQuestManagerComponent::UQuestManagerComponent()

{

PrimaryComponentTick.bCanEverTick = true;

}

void UQuestManagerComponent::BeginPlay()

{

Super::BeginPlay();

AddAllQuestsFromDataTable();

}

void UQuestManagerComponent::AddQuest(const FQuest& NewQuest)

{

ActiveQuests.Add(NewQuest);

CacheQuest(NewQuest);

UE\_LOG(LogTemp, Warning, TEXT("Quest Added: %s"), \*NewQuest.Name);

UpdateCurrentQuest();

}

void UQuestManagerComponent::CompleteQuest(const FString& QuestName)

{

for (FQuest& Quest : ActiveQuests)

{

if (Quest.Name == QuestName)

{

Quest.bIsCompleted = true;

UE\_LOG(LogTemp, Warning, TEXT("Quest Completed: %s"), \*Quest.Name);

break;

}

}

UpdateCurrentQuest();

}

void UQuestManagerComponent::AddAllQuestsFromDataTable()

{

if (QuestDataTable)

{

static const FString ContextString(TEXT("QuestDataContext"));

TArray<FName> RowNames = QuestDataTable->GetRowNames();

for (const FName& RowName : RowNames)

{

FQuest\* Quest = QuestDataTable->FindRow<FQuest>(RowName, ContextString);

if (Quest)

{

AddQuest(\*Quest);

}

}

}

else

{

UE\_LOG(LogTemp, Error, TEXT("QuestDataTable is not assigned."));

}

UpdateCurrentQuest();

}

void UQuestManagerComponent::UpdateCurrentQuest()

{

bool bFoundNextQuest = false;

for (const FQuest& Quest : ActiveQuests)

{

if (!Quest.bIsCompleted)

{

if (!bFoundNextQuest || Quest.Name == CurrentQuest.Name)

{

CurrentQuest = Quest;

OnQuestUpdated.Broadcast(CurrentQuest);

bFoundNextQuest = true;

// Start the timer if the quest has a time limit

if (CurrentQuest.TimeLimit > 0.0f)

{

GetWorld()->GetTimerManager().SetTimer(TimerHandle, this, &UQuestManagerComponent::OnTimedQuestCompleted, CurrentQuest.TimeLimit, false);

}

break;

}

}

}

}

void UQuestManagerComponent::CacheQuest(FQuest Quest)

{

QuestCache.Add(Quest.Name, Quest);

}

FQuest\* UQuestManagerComponent::GetCachedQuest(FString QuestName)

{

return QuestCache.Find(QuestName);

}

void UQuestManagerComponent::OnTimedQuestCompleted()

{

UE\_LOG(LogTemp, Warning, TEXT("Timed Quest Completed: %s"), \*CurrentQuest.Name);

CompleteQuest(CurrentQuest.Name);

}

void UQuestManagerComponent::StartTimedQuest(const FString& QuestName)

{

FQuest\* Quest = GetCachedQuest(QuestName);

if (Quest && Quest->TimeLimit > 0.0f)

{

GetWorld()->GetTimerManager().SetTimer(TimerHandle, this, &UQuestManagerComponent::OnTimedQuestCompleted, Quest->TimeLimit, false);

}

}

### Integrating with the Character Class

#### NinjaCombatCharacter.cpp

cpp

Copy code

#include "NinjaCombatCharacter.h"

#include "QuestSystem/QuestManagerComponent.h"

#include "QuestSystem/QuestWidget.h"

#include "Blueprint/UserWidget.h"

#include "Kismet/GameplayStatics.h"

#include "EnhancedInputSubsystems.h"

void ANinjaCombatCharacter::BeginPlay()

{

Super::BeginPlay();

if (APlayerController\* PlayerController = Cast<APlayerController>(Controller))

{

if (UEnhancedInputLocalPlayerSubsystem\* Subsystem = ULocalPlayer::GetSubsystem<UEnhancedInputLocalPlayerSubsystem>(PlayerController->GetLocalPlayer()))

{

if (DefaultMappingContext)

{

Subsystem->AddMappingContext(DefaultMappingContext, 0);

}

else

{

UE\_LOG(LogTemp, Error, TEXT("DefaultMappingContext is not set"));

}

}

}

// Setup Quest Widget

if (QuestManagerComponent && QuestWidgetClass)

{

QuestWidget = CreateWidget<UQuestWidget>(GetWorld(), QuestWidgetClass);

if (QuestWidget)

{

QuestWidget->AddToViewport();

QuestManagerComponent->OnQuestUpdated.AddDynamic(QuestWidget, &UQuestWidget::UpdateQuestList);

AddQuestsFromDataTable();

}

}

}

void ANinjaCombatCharacter::AddQuestsFromDataTable()

{

if (QuestManagerComponent)

{

QuestManagerComponent->AddAllQuestsFromDataTable();

}

}

void ANinjaCombatCharacter::HandleQuestCompleted()

{

if (QuestManagerComponent && !CurrentQuest.Name.IsEmpty())

{

QuestManagerComponent->CompleteQuest(CurrentQuest.Name);

}

}

### Widget Implementation

Ensure the widget class is set up to update the quest list.

#### QuestWidget.cpp

cpp

Copy code

#include "QuestSystem/QuestWidget.h"

#include "Components/ScrollBox.h"

#include "Components/TextBlock.h"

void UQuestWidget::NativeConstruct()

{

Super::NativeConstruct();

}

void UQuestWidget::UpdateQuestList(const FQuest& Quest)

{

if (QuestScrollBox)

{

QuestScrollBox->ClearChildren();

UTextBlock\* QuestText = NewObject<UTextBlock>(QuestScrollBox);

if (QuestText)

{

QuestText->SetText(FText::FromString(FString::Printf(TEXT("%s: %s"), \*Quest.Name, \*Quest.Description)));

QuestScrollBox->AddChild(QuestText);

}

}

}

### Summary

* Caching: We use TMap to cache quests for quick lookups.
* Timers: We added timed quest functionality using FTimerHandle.
* Efficient Management: TMap helps in efficient quest management.
* Event Handling: Improved event handling with delegates and broadcasting updates.

HOW TO USE THEM quest system:---

To implement different types of quests (e.g., timed quests and non-timed quests) and reflect these changes in the UI, we need to extend the existing quest system. We'll add logic to handle both types of quests and update the UI accordingly.

### Steps to Implement

1. Extend the Quest Structure: Add fields to handle timed quests.
2. Modify the Quest Manager: Handle both timed and non-timed quests.
3. Update the UI Widget: Display quest details, including the timer.
4. Update the Character Class: Implement logic to start and manage quests.

### 1. Extend the Quest Structure

Update FQuest to include a boolean for timed quests and a timer field.

cpp

Copy code

#pragma once

#include "CoreMinimal.h"

#include "Engine/DataTable.h"

#include "Quest.generated.h"

USTRUCT(BlueprintType)

struct FQuest : public FTableRowBase

{

GENERATED\_BODY()

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Quest")

FString Name;

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Quest")

FString Description;

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Quest")

bool bIsCompleted;

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Quest")

TArray<FString> Objectives;

// New properties for timed quests

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Quest")

bool bIsTimed;

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Quest")

float TimeLimit;

// Default constructor

FQuest()

: Name(TEXT("Default Quest")), Description(TEXT("Default Description")), bIsCompleted(false), bIsTimed(false), TimeLimit(0.0f) {}

};

### 2. Modify the Quest Manager

Update QuestManagerComponent to handle different types of quests.

#### QuestManagerComponent.h

cpp

Copy code

#pragma once

#include "CoreMinimal.h"

#include "Components/ActorComponent.h"

#include "Engine/DataTable.h"

#include "Quest.h"

#include "QuestManagerComponent.generated.h"

DECLARE\_DYNAMIC\_MULTICAST\_DELEGATE\_OneParam(FOnQuestUpdated, const FQuest&, CurrentQuest);

UCLASS(ClassGroup=(Custom), meta=(BlueprintSpawnableComponent))

class YOURGAME\_API UQuestManagerComponent : public UActorComponent

{

GENERATED\_BODY()

public:

UQuestManagerComponent();

protected:

virtual void BeginPlay() override;

public:

UPROPERTY(BlueprintAssignable, Category = "Quest")

FOnQuestUpdated OnQuestUpdated;

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Quest")

TArray<FQuest> ActiveQuests;

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Quest")

UDataTable\* QuestDataTable;

UFUNCTION(BlueprintCallable, Category = "Quest")

void AddQuest(const FQuest& NewQuest);

UFUNCTION(BlueprintCallable, Category = "Quest")

void CompleteQuest(const FString& QuestName);

UFUNCTION(BlueprintCallable, Category = "Quest")

void AddAllQuestsFromDataTable();

UFUNCTION(BlueprintCallable, Category = "Quest")

void StartTimedQuest(const FString& QuestName);

private:

TMap<FString, FQuest> QuestCache;

FQuest CurrentQuest;

FTimerHandle TimerHandle;

void UpdateCurrentQuest();

void CacheQuest(FQuest Quest);

FQuest\* GetCachedQuest(FString QuestName);

void OnTimedQuestCompleted();

};

#### QuestManagerComponent.cpp

cpp

Copy code

#include "QuestSystem/QuestManagerComponent.h"

#include "TimerManager.h"

UQuestManagerComponent::UQuestManagerComponent()

{

PrimaryComponentTick.bCanEverTick = true;

}

void UQuestManagerComponent::BeginPlay()

{

Super::BeginPlay();

AddAllQuestsFromDataTable();

}

void UQuestManagerComponent::AddQuest(const FQuest& NewQuest)

{

ActiveQuests.Add(NewQuest);

CacheQuest(NewQuest);

UE\_LOG(LogTemp, Warning, TEXT("Quest Added: %s"), \*NewQuest.Name);

UpdateCurrentQuest();

}

void UQuestManagerComponent::CompleteQuest(const FString& QuestName)

{

for (FQuest& Quest : ActiveQuests)

{

if (Quest.Name == QuestName)

{

Quest.bIsCompleted = true;

UE\_LOG(LogTemp, Warning, TEXT("Quest Completed: %s"), \*Quest.Name);

break;

}

}

UpdateCurrentQuest();

}

void UQuestManagerComponent::AddAllQuestsFromDataTable()

{

if (QuestDataTable)

{

static const FString ContextString(TEXT("QuestDataContext"));

TArray<FName> RowNames = QuestDataTable->GetRowNames();

for (const FName& RowName : RowNames)

{

FQuest\* Quest = QuestDataTable->FindRow<FQuest>(RowName, ContextString);

if (Quest)

{

AddQuest(\*Quest);

}

}

}

else

{

UE\_LOG(LogTemp, Error, TEXT("QuestDataTable is not assigned."));

}

UpdateCurrentQuest();

}

void UQuestManagerComponent::UpdateCurrentQuest()

{

bool bFoundNextQuest = false;

for (const FQuest& Quest : ActiveQuests)

{

if (!Quest.bIsCompleted)

{

if (!bFoundNextQuest || Quest.Name == CurrentQuest.Name)

{

CurrentQuest = Quest;

OnQuestUpdated.Broadcast(CurrentQuest);

bFoundNextQuest = true;

// Start the timer if the quest is a timed quest

if (CurrentQuest.bIsTimed)

{

GetWorld()->GetTimerManager().SetTimer(TimerHandle, this, &UQuestManagerComponent::OnTimedQuestCompleted, CurrentQuest.TimeLimit, false);

}

break;

}

}

}

}

void UQuestManagerComponent::CacheQuest(FQuest Quest)

{

QuestCache.Add(Quest.Name, Quest);

}

FQuest\* UQuestManagerComponent::GetCachedQuest(FString QuestName)

{

return QuestCache.Find(QuestName);

}

void UQuestManagerComponent::OnTimedQuestCompleted()

{

UE\_LOG(LogTemp, Warning, TEXT("Timed Quest Completed: %s"), \*CurrentQuest.Name);

CompleteQuest(CurrentQuest.Name);

}

void UQuestManagerComponent::StartTimedQuest(const FString& QuestName)

{

FQuest\* Quest = GetCachedQuest(QuestName);

if (Quest && Quest->bIsTimed)

{

GetWorld()->GetTimerManager().SetTimer(TimerHandle, this, &UQuestManagerComponent::OnTimedQuestCompleted, Quest->TimeLimit, false);

}

}

### 3. Update the UI Widget

#### QuestWidget.h

cpp

Copy code

#pragma once

#include "CoreMinimal.h"

#include "Blueprint/UserWidget.h"

#include "Quest.h"

#include "QuestWidget.generated.h"

UCLASS()

class YOURGAME\_API UQuestWidget : public UUserWidget

{

GENERATED\_BODY()

public:

UPROPERTY(meta = (BindWidget))

class UScrollBox\* QuestScrollBox;

UFUNCTION(BlueprintCallable, Category = "Quest")

void UpdateQuestList(const FQuest& Quest);

protected:

virtual void NativeConstruct() override;

};

#### QuestWidget.cpp

cpp

Copy code

#include "QuestSystem/QuestWidget.h"

#include "Components/ScrollBox.h"

#include "Components/TextBlock.h"

void UQuestWidget::NativeConstruct()

{

Super::NativeConstruct();

}

void UQuestWidget::UpdateQuestList(const FQuest& Quest)

{

if (QuestScrollBox)

{

QuestScrollBox->ClearChildren();

// Add quest name and description

UTextBlock\* QuestText = NewObject<UTextBlock>(QuestScrollBox);

if (QuestText)

{

QuestText->SetText(FText::FromString(FString::Printf(TEXT("%s: %s"), \*Quest.Name, \*Quest.Description)));

QuestScrollBox->AddChild(QuestText);

}

// If the quest is timed, add a timer display

if (Quest.bIsTimed)

{

UTextBlock\* TimerText = NewObject<UTextBlock>(QuestScrollBox);

if (TimerText)

{

TimerText->SetText(FText::FromString(FString::Printf(TEXT("Time Limit: %f seconds"), Quest.TimeLimit)));

QuestScrollBox->AddChild(TimerText);

}

}

}

}

### 4. Update the Character Class