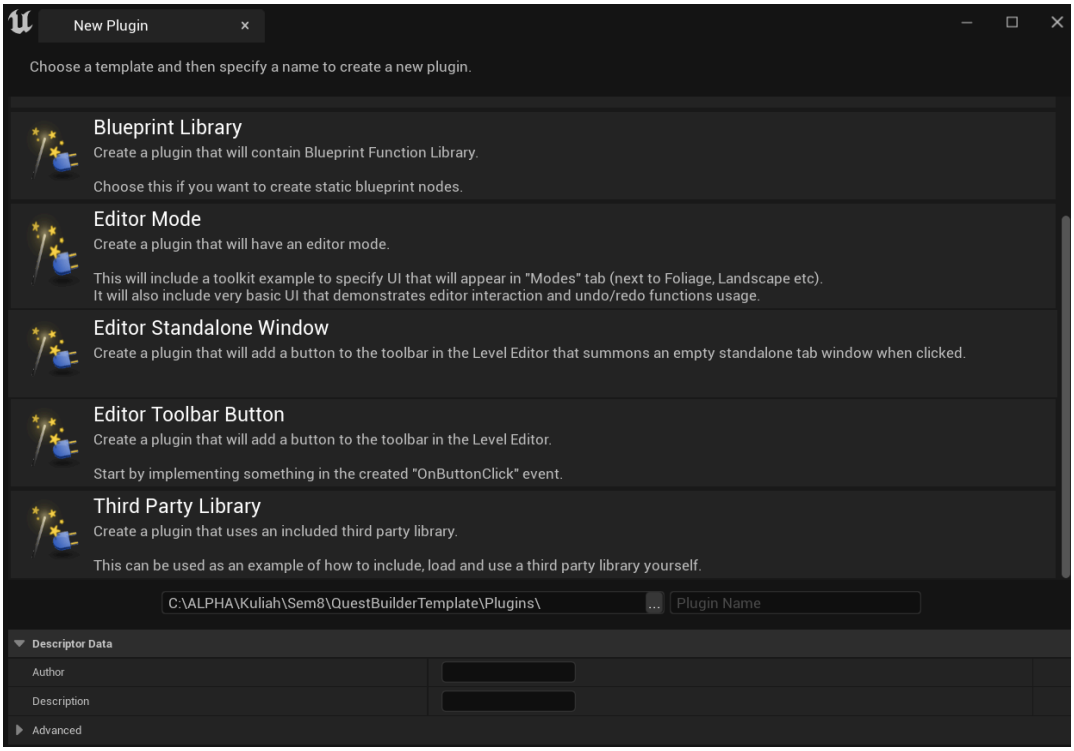
4. IMPLEMENTASI SISTEM

Bab ini akan membahas implementasi sistem sesuai dengan analisa dan desain sistem. Implementasi sistem yang akan dibahas meliputi pengaturan *module plugin*, pembuatan sistem editor, pembuatan sistem runtime, serta pembuatan demo proyek.

4.1 Pengaturan *Module Plugin*

Pada sub-bab ini akan membahas mengenai pengaturan *module* yang akan digunakan pada *plugin*. langkah awal yang diperlukan adalah dengan membuat *plugin* melalui *editor Unreal Engine* seperti pada gambar 4.1. *Module plugin* akan dibagi menjadi 2 bagian meliputi *quest system editor*, dan *quest system runtime.* Kemudian pada gambar 4.2, dilakukan pengaturan detail awal *module* pada file *uplugin* ketika pertama kali dibuat.



Gambar 4.1 Pembuatan plugin

Gambar 4.2 Pengaturan *Module* Pada *File Uplugin*

4.1.1 Pengaturan *Editor Module*

Pada bagian ini akan berisi pengaturan *dependency code* yang akan digunakan pada *module editor.*

Segmen Program 4.1 Pengaturan *Dependency Editor Module*

```
using UnrealBuildTool;

public class Quest_System_Editor : ModuleRules
{
        public Quest_System_Editor(ReadOnlyTargetRules Target) : base(Target)
        {
                PCHUsage = ModuleRules.PCHUsageMode.UseExplicitOrSharedPCHs;

                PublicIncludePaths.AddRange(
                        new string[]
                        {
                        }
                        );
```

```
        PrivateIncludePaths.AddRange(
                new string[]
                {
"Quest_System_Editor/Private",
"Quest_System_Editor/Public",
                }
                );

        PublicDependencyModuleNames.AddRange(
                new string[]
                {
"Quest_System_Runtime",
"Core",
"CoreUObject",
"Engine",
"UnrealEd",
                        "Kismet",
                        "AIGraph",
                        "EditorStyle",
                }
                );

        PrivateDependencyModuleNames.AddRange(
                new string[]
                {
"Quest_System_Runtime",
"AssetTools",
"GraphEditor",
"PropertyEditor",
"EditorStyle",
"Kismet",
"KismetWidgets",
"ApplicationCore",
"ToolMenus",
"Projects",
                        "InputCore",
                        "UnrealEd",
                        "ToolMenus",
                        "CoreUObject",
                        "Engine",
                        "Slate",
                        "SlateCore",
"AIGraph",
                }
                );
```

```
                DynamicallyLoadedModuleNames.AddRange(
                        new string[]
                        {
                        }
                        );
        }
}
```

4.1.2 Pengaturan *Editor Module*

Pada bagian ini akan berisi pengaturan *dependency code* yang akan digunakan pada *module runtime.*

Segmen Program 4.2 Pengaturan *Dependency Runtime Module*

```
using UnrealBuildTool;

public class Quest_System_Runtime : ModuleRules
{
   public Quest_System_Runtime(ReadOnlyTargetRules Target) : base(Target)
   {
      PCHUsage = PCHUsageMode.UseExplicitOrSharedPCHs;
      bLegacyPublicIncludePaths = false;
      ShadowVariableWarningLevel = WarningLevel.Error;

      PublicIncludePaths.AddRange(
         new string[] {
                        }
         );

      PrivateIncludePaths.AddRange(
         new string[] {
            "Quest_System_Runtime/Private",
                        }
         );

      PublicDependencyModuleNames.AddRange(
         new string[]
         {
            "Core",
            "CoreUObject",
            "Engine",
          }
         );
```

```
    PrivateDependencyModuleNames.AddRange(
      new string[]
      {
        "Slate",
        "SlateCore",
        "GameplayTags",
      }
      );

    DynamicallyLoadedModuleNames.AddRange(
      new string[]
      {
                        }
      );
  }
}
```

4.2 Pembuatan Sistem *Editor*

Pada sub-bab ini akan membahas mengenai pembuatan *editor quest* yang meliputi pembuatan *asset editor*, *graph schema, toolbar, editor graph,* serta *editor graph node.*

4.2.1 *Asset Editor*

Berikut merupakan pembuatan *asset editor quest* yang berfungsi untuk mengelola tombol *toolbar*, mengatur dan membuat *layout tab,* serta mengatur perintah agar dapat dijalankan pada *tab* yang telah dibuat. Ketika *Quest Asset* telah selesai dibuat dan dibuka, *Editor* akan langsung mengkonstruksi layout yang sudah terdaftar. Semua data atribut dan parameter akan dimuat dan siap untuk diedit oleh *user.* Kemudian *editor* ini akan menghubungkan semua perintah yang ada pada setiap tombol, dan membuat fungsi sesuai perintah yang ada pada *toolbar* dan *tab* yang lainnya.

Segmen Program 4.3 *Quest Asset Editor*

```
#include "AssetEditor_QuestSystem.h"
#include "QuestSystemGraphNode_Root.h"
#include "Condition/QuestCondition.h"
#include "EngineGlobals.h"
#include "Editor/EditorEngine.h"
#include "Editor.h"
#include "UnrealEdGlobals.h"
```

```
#include "Event/QuestEvent.h"
#include "BlueprintEditor.h"
#include "UObject/ObjectSaveContext.h"
#include "GraphEditorActions.h"
#include "Framework/Commands/GenericCommands.h"
#include "AssetEditorToolbar_QuestSystem.h"
#include "Quest_System_EditorCommands.h"
#include "EdGraph_QuestSystemGraph.h"
#include "EdNode_QuestSystemNode.h"
#include "Kismet2/KismetEditorUtilities.h"
#include "K2Node.h"
#include "EdNode_QuestSystemEdge.h"
#include "AssetRegistry/AssetRegistryModule.h"
#include "HAL/PlatformApplicationMisc.h"
#include "AssetGraphSchema_QuestSystem.h"
#include "QuestSystemEditorUtils.h"
#include "QuestEditorFactory.h"
#include "Kismet2/BlueprintEditorUtils.h"
#include "SMyQuest.h"
#include "Task/QuestTask.h"
#include "Quest_System_Editor.h"
#include <ContentBrowserModule.h>
#include <ContentBrowserFrontEndFilterExtension.h>
#include "Kismet2/KismetDebugUtilities.h"
#include "WorkflowOrientedApp/WorkflowUObjectDocuments.h"
#include "WorkflowOrientedApp/WorkflowCentricApplication.h"
#include "Framework/Notifications/NotificationManager.h"
#include "Widgets/Notifications/SNotificationList.h"
#include <Kismet2/KismetEditorUtilities.h>

void FAssetEditor_QuestSystem::InitQuestSystemGraphAssetEditor(const
EToolkitMode::Type Mode, const TSharedPtr<IToolkitHost>& InitToolkitHost,
UQuestSystemGraph* Graph)
{
	EditingQuestGraph = Graph;
	FGenericCommands::Register();
	FGraphEditorCommands::Register();
	FMyQuestCommands::Register();
	FQuest_System_EditorCommands::Register();

	if (!ToolbarBuilder.IsValid())
	{
		ToolbarBuilder = MakeShareable(new
FAssetEditorToolbar_QuestSystem(SharedThis(this)));
	}

	BindCommands();

	CreateInternalWidgets();

	TSharedPtr<FExtender> ToolbarExtender = MakeShareable(new FExtender);

	ToolbarBuilder->AddQuestSystemToolbar(ToolbarExtender);

	// Layout
	const TSharedRef<FTabManager::FLayout> StandaloneDefaultLayout =
FTabManager::NewLayout("Standalone_GenericGraphEditor_Layout_v1")
		->AddArea
		(
```

```
                        FTabManager::NewPrimaryArea()->SetOrientation(Orient_Vertical)
#if ENGINE_MAJOR_VERSION < 5
                        ->Split
                        (
                                FTabManager::NewStack()
                                ->SetSizeCoefficient(0.1f)
                                ->AddTab(GetToolbarTabId(),
ETabState::OpenedTab)->SetHideTabWell(true)
                        )
#endif // #if ENGINE_MAJOR_VERSION < 5
                        ->Split
                        (

FTabManager::NewSplitter()->SetOrientation(Orient_Horizontal)->SetSizeCoefficient(0.9
f)
                                ->Split
                                (
                                        FTabManager::NewStack()
                                        ->SetSizeCoefficient(0.2f)

->AddTab(FQuestSystemAssetEditorTabs::MyQuestDetailID, ETabState::OpenedTab)
                                )
                                ->Split
                                (
                                        FTabManager::NewStack()
                                        ->SetSizeCoefficient(0.55f)

->AddTab(/*FQuestSystemAssetEditorTabs::ViewportID*/"Document", ETabState::ClosedTab)
                                )
                                ->Split
                                (

FTabManager::NewSplitter()->SetOrientation(Orient_Vertical)->SetSizeCoefficient(0.25f
)
                                        ->Split
                                        (
                                                FTabManager::NewStack()
                                                ->SetSizeCoefficient(0.55f)

->AddTab(FQuestSystemAssetEditorTabs::GenericGraphPropertyID, ETabState::OpenedTab)
                                        )
                                        /*->Split
                                        (
                                                FTabManager::NewStack()
                                                ->SetSizeCoefficient(0.45f)

->AddTab(FQuestSystemAssetEditorTabs::GenericGraphEditorSettingsID,
ETabState::OpenedTab)
                                        )*/
                                )
                        )
                );

        const bool bCreateDefaultStandaloneMenu = true;
        const bool bCreateDefaultToolbar = true;
        FAssetEditorToolkit::InitAssetEditor(Mode, InitToolkitHost,
GenericGraphEditorAppName, StandaloneDefaultLayout, bCreateDefaultStandaloneMenu,
bCreateDefaultToolbar, EditingQuestGraph, false);
        CommonInitialization(Graph, true);
```

```
	RegenerateMenusAndToolbars();
	//InitializeDocumentTab();

	if (Graph->bIsNewlyCreated)
	{
		NewDocument_OnClicked(CGT_NewQuestGraph);
		Graph->bIsNewlyCreated = false;
		Graph->Modify();
	}
	else
	{
		if (GetQuestSystemObj()->QuestGraphPages.Num() > 0)
		{
			OpenDocument(GetQuestSystemObj()->QuestGraphPages[0],
FDocumentTracker::OpenNewDocument);
		}
	}
}

void FAssetEditor_QuestSystem::RegisterTabSpawners(const TSharedRef<FTabManager>&
InTabManager)
{

	DocumentManager->SetTabManager(InTabManager);

	//FWorkflowCentricApplication::RegisterTabSpawners(InTabManager);

	WorkspaceMenuCategory =
InTabManager->AddLocalWorkspaceMenuCategory(LOCTEXT("WorkspaceMenu_GenericGraphEditor
", "Generic Graph Editor"));
	auto WorkspaceMenuCategoryRef = WorkspaceMenuCategory.ToSharedRef();

	FAssetEditorToolkit::RegisterTabSpawners(InTabManager);


InTabManager->RegisterTabSpawner(FQuestSystemAssetEditorTabs::MyQuestDetailID,
FOnSpawnTab::CreateSP(this, &FAssetEditor_QuestSystem::SpawnTab_MyQuest))
		.SetDisplayName(LOCTEXT("Quest Details", "My Quest"))
		.SetGroup(WorkspaceMenuCategoryRef)
		.SetIcon(FSlateIcon(FAppStyle::GetAppStyleSetName(),
"LevelEditor.Tabs.Details"));

	InTabManager->RegisterTabSpawner(FQuestSystemAssetEditorTabs::ViewportID,
FOnSpawnTab::CreateSP(this, &FAssetEditor_QuestSystem::SpawnTab_Viewport))
		.SetDisplayName(LOCTEXT("GraphCanvasTab", "Viewport"))
		.SetGroup(WorkspaceMenuCategoryRef)
		.SetIcon(FSlateIcon(FAppStyle::GetAppStyleSetName(),
"GraphEditor.EventGraph_16x"));


InTabManager->RegisterTabSpawner(FQuestSystemAssetEditorTabs::GenericGraphPropertyID,
FOnSpawnTab::CreateSP(this, &FAssetEditor_QuestSystem::SpawnTab_Details))
		.SetDisplayName(LOCTEXT("DetailsTab", "Property"))
		.SetGroup(WorkspaceMenuCategoryRef)
		.SetIcon(FSlateIcon(FAppStyle::GetAppStyleSetName(),
"LevelEditor.Tabs.Details"));
}

void FAssetEditor_QuestSystem::BindCommands()
```

```
{
	ToolkitCommands->MapAction(FQuest_System_EditorCommands::Get().NewTask,
		FExecuteAction::CreateSP(this,
&FAssetEditor_QuestSystem::CreateNewTask),
		FCanExecuteAction::CreateSP(this,
&FAssetEditor_QuestSystem::CanCreateNewTask),
		FIsActionChecked(),
		FIsActionButtonVisible::CreateSP(this,
&FAssetEditor_QuestSystem::IsNewTaskButtonVisible)
	);


ToolkitCommands->MapAction(FQuest_System_EditorCommands::Get().NewQuestCondition,
		FExecuteAction::CreateSP(this,
&FAssetEditor_QuestSystem::CreateNewQuestCondition),
		FCanExecuteAction::CreateSP(this,
&FAssetEditor_QuestSystem::CanCreateQuestCondition)
	);

	ToolkitCommands->MapAction(FQuest_System_EditorCommands::Get().NewQuestEvent,
		FExecuteAction::CreateSP(this,
&FAssetEditor_QuestSystem::CreateNewQuestEvent),
		FCanExecuteAction::CreateSP(this,
&FAssetEditor_QuestSystem::CanCreateQuestEvent)
	);

	ToolkitCommands->MapAction(FQuest_System_EditorCommands::Get().QuestSetting,
		FExecuteAction::CreateSP(this,
&FAssetEditor_QuestSystem::OpenQuestSetting),
		FCanExecuteAction::CreateSP(this,
&FAssetEditor_QuestSystem::CanOpenQuestSetting)
	);


ToolkitCommands->MapAction(FQuest_System_EditorCommands::Get().AddNewQuestGraph,
		FExecuteAction::CreateSP(this,
&FAssetEditor_QuestSystem::NewDocument_OnClicked, CGT_NewQuestGraph),
		FCanExecuteAction::CreateSP(this,
&FAssetEditor_QuestSystem::InEditingMode),
		FIsActionChecked(),
		FIsActionButtonVisible::CreateSP(this,
&FAssetEditor_QuestSystem::NewDocument_IsVisibleForType, CGT_NewQuestGraph)
	);

}
```

4.2.2 *Toolbar*

Berikut merupakan pembuatan *quest toolbar* yang akan digunakan untuk membuat *task, event,* dan *condition* yang baru. Tombol yang dibuat, khususnya pada pembuatan *blueprint* baru akan memunculkan *class picker*. Dimana *user* dapat memodifikasi data nama *asset* yang akan dibuatnya. Pembuatan *blueprint* baru ini kemudian akan didaftarkan pada *engine* dan akan memunculkan datanya pada parameter yang berada pada *asset editor.*

Segmen Program 4.4 *Quest Toolbar*

```
#include "AssetEditorToolbar_QuestSystem.h"
#include "Quest_System_EditorCommands.h"
#include "Quest_System_EditorStyle.h"
#include "AssetEditor_QuestSystem.h"

#define LOCTEXT_NAMESPACE "AssetEditorToolbar_QuestSystem"

void FAssetEditorToolbar_QuestSystem::AddQuestSystemToolbar(TSharedPtr<FExtender>
Extender)
{
	check(QuestSystemEditor.IsValid());
	TSharedPtr<FAssetEditor_QuestSystem> QuestSystemEditorPtr =
QuestSystemEditor.Pin();

	TSharedPtr<FExtender> ToolbarExtender = MakeShareable(new FExtender);
	ToolbarExtender->AddToolBarExtension("Asset", EExtensionHook::After,
QuestSystemEditorPtr->GetToolkitCommands(), FToolBarExtensionDelegate::CreateSP(this,
&FAssetEditorToolbar_QuestSystem::FillQuestSystemToolbar));
	QuestSystemEditorPtr->AddToolbarExtender(ToolbarExtender);
}

void FAssetEditorToolbar_QuestSystem::FillQuestSystemToolbar(FToolBarBuilder&
ToolbarBuilder)
{
	check(QuestSystemEditor.IsValid());
	TSharedPtr<FAssetEditor_QuestSystem> QuestSystemEditorPtr =
QuestSystemEditor.Pin();

	ToolbarBuilder.BeginSection("Quest System");
	{
		const FText NewTaskLabel = LOCTEXT("NewTask_Label", "New Task");
		const FText NewTaskTooltip = LOCTEXT("NewTask_ToolTip", "Create a new
task node Blueprint from a base class");
		const FSlateIcon NewTaskIcon =
FSlateIcon(FAppStyle::GetAppStyleSetName(), "BTEditor.Graph.NewTask");



ToolbarBuilder.AddToolBarButton(FQuest_System_EditorCommands::Get().NewTask,
			NAME_None,
			NewTaskLabel,
			NewTaskTooltip,
			NewTaskIcon
		);
	}
	ToolbarBuilder.EndSection();

	ToolbarBuilder.BeginSection("Quest System");
	{
		const FText NewConditionLabel = LOCTEXT("NewQuestCondition_Label",
"New Quest Condition");
		const FText NewConditionTooltip = LOCTEXT("NewQuestCondition_ToolTip",
"Create a new Quest Condition Blueprint from a base class");
		const FSlateIcon NewConditionIcon =
FSlateIcon(FAppStyle::GetAppStyleSetName(),
```

```
"BTEditor.Graph.BTNode.Decorator.Conditional.Icon");



ToolbarBuilder.AddToolBarButton(FQuest_System_EditorCommands::Get().NewQuestCondition
,
                        NAME_None,
                        NewConditionLabel,
                        NewConditionTooltip,
                        NewConditionIcon
                );
        }
        ToolbarBuilder.EndSection();

        ToolbarBuilder.BeginSection("Quest System");
        {
                const FText NewEventLabel = LOCTEXT("NewQuestEvent_Label", "New Quest
Event");
                const FText NewEventTooltip = LOCTEXT("NewQuestEvent_ToolTip", "Create
a new Quest Event Blueprint from a base class");
                const FSlateIcon NewEventIcon =
FSlateIcon(FAppStyle::GetAppStyleSetName(), "GraphEditor.CustomEvent_16x");



ToolbarBuilder.AddToolBarButton(FQuest_System_EditorCommands::Get().NewQuestEvent,
                        NAME_None,
                        NewEventLabel,
                        NewEventTooltip,
                        NewEventIcon
                );
        }
        ToolbarBuilder.EndSection();

        ToolbarBuilder.BeginSection("Quest System");
        {
                const FText QuestSettingLabel = LOCTEXT("QuestSetting_Label", "Quest
Builder Setting");
                const FText QuestSettingTooltip = LOCTEXT("QuestSetting_ToolTip",
"Access Quest Builder Setting");
                const FSlateIcon QuestSettingIcon =
FSlateIcon(FAppStyle::GetAppStyleSetName(), "ProjectSettings.TabIcon");



ToolbarBuilder.AddToolBarButton(FQuest_System_EditorCommands::Get().QuestSetting,
                        NAME_None,
                        QuestSettingLabel,
                        QuestSettingTooltip,
                        QuestSettingIcon
                );
        }
        ToolbarBuilder.EndSection();


}

#undef LOCTEXT_NAMESPACE
```

4.2.3 *Editor Graph*

Berikut merupakan editor graph yang akan berperan sebagai interface graph yang dibuat, mengisi dependency tiap node, event, task, dan condition, serta berfungsi dalam membuat dan mengatur letak graph node. *Graph* ini akan melakukan *update* ketika *user* melakukan perintah *save* pada *quest asset.* Dimana pada tiap parameter yang ada akan disesuaikan data-datanya agar saat proses *runtime*, semua *reference* akan sama dengan yang diubah oleh *user.*

Segmen Program 4.5 *Quest Editor Graph*

```
#include "EdGraph_QuestSystemGraph.h"
#include "EdNode_QuestSystemEdge.h"
#include "EdNode_QuestSystemRoot.h"
#include "EdNode_QuestSystemNode.h"
#include "QuestSystemGraphEdge.h"
#include "QuestSystemGraphNode.h"
#include "QuestSystemGraphNode_Objective.h"
#include "Condition/QuestCondition.h"
#include "Event/QuestEvent.h"
#include "Task/QuestTask.h"

void UEdGraph_QuestSystemGraph::RebuildQuestSystemGraph()
{
	//LOG_INFO(TEXT("UQuestSystemGraphEdGraph::RebuildGenericGraph has been
called"));

	UQuestSystemGraph* Graph = GetQuestSystemGraph();

	Clear();

	if (Graph->CategoryDisplayName.IsEmpty())
	{
		Graph->CategoryDisplayName = Graph->GetFName().ToString();
	}

	for (UQuestCondition* QuestCondition : QuestSystem->Conditions)
	{
		if (QuestCondition != nullptr)
		{
			QuestCondition->QuestGraph = Graph;
			QuestCondition->QuestSystem = QuestSystem;
		}
	}

	for (int i = 0; i < Nodes.Num(); ++i)
	{
		if (UEdNode_QuestSystemNode* EdNode =
Cast<UEdNode_QuestSystemNode>(Nodes[i]))
		{
			if (EdNode->QuestSystemGraphNode == nullptr)
```

```
                        continue;

                UQuestSystemGraphNode* QuestSystemGraphNode =
EdNode->QuestSystemGraphNode;

                NodeMap.Add(QuestSystemGraphNode, EdNode);

                QuestSystem->AllNodes.Add(QuestSystemGraphNode);
                QuestSystem->NodeMap.Emplace(QuestSystemGraphNode->ID,
QuestSystemGraphNode);

                for (int PinIdx = 0; PinIdx < EdNode->Pins.Num(); ++PinIdx)
                {
                        UEdGraphPin* Pin = EdNode->Pins[PinIdx];

                        if (Pin->Direction != EEdGraphPinDirection::EGPD_Output)
                                continue;

                        for (int LinkToIdx = 0; LinkToIdx < Pin->LinkedTo.Num();
++LinkToIdx)
                        {
                                UQuestSystemGraphNode* ChildNode = nullptr;
                                if (UEdNode_QuestSystemNode* EdNode_Child =
Cast<UEdNode_QuestSystemNode>(Pin->LinkedTo[LinkToIdx]->GetOwningNode()))
                                {
                                        ChildNode =
EdNode_Child->QuestSystemGraphNode;
                                }
                                else if (UEdNode_QuestSystemEdge* EdNode_Edge =
Cast<UEdNode_QuestSystemEdge>(Pin->LinkedTo[LinkToIdx]->GetOwningNode()))
                                {
                                        UEdNode_QuestSystemNode* Child =
EdNode_Edge->GetEndNode();
                                        if (Child != nullptr)
                                        {
                                                ChildNode =
Child->QuestSystemGraphNode;
                                        }
                                }

                                if (ChildNode != nullptr)
                                {

QuestSystemGraphNode->ChildrenNodes.Add(ChildNode);


ChildNode->ParentNodes.Add(QuestSystemGraphNode);
                                }
                                else
                                {

//LOG_ERROR(TEXT("UEdGraph_GenericGraph::RebuildGenericGraph can't find child
node"));
                                }
                        }
                }

                if (UQuestSystemGraphNode_Objective* ObjectiveNode =
Cast<UQuestSystemGraphNode_Objective>(QuestSystemGraphNode))
```

```
                        {
                                for (UQuestTask* Task : ObjectiveNode->Tasks)
                                {
                                        if (Task != nullptr)
                                        {
                                                Task->ObjectiveNode = ObjectiveNode;
                                                Task->QuestGraph = Graph;
                                                Task->QuestSystem = QuestSystem;
                                        }
                                }
                        }

                        for (UQuestCondition* QuestCondition :
QuestSystemGraphNode->Conditions)
                        {
                                if (QuestCondition != nullptr)
                                {
                                        QuestCondition->QuestGraph = Graph;
                                        QuestCondition->QuestSystem = QuestSystem;
                                }
                        }

                        for (UQuestEvent* QuestEvent : QuestSystemGraphNode->Events)
                        {
                                if (QuestEvent != nullptr)
                                {
                                        QuestEvent->QuestGraph = Graph;
                                        QuestEvent->QuestSystem = QuestSystem;
                                }
                        }

                }
                else if (UEdNode_QuestSystemEdge* EdgeNode =
Cast<UEdNode_QuestSystemEdge>(Nodes[i]))
                {
                        UEdNode_QuestSystemNode* StartNode = EdgeNode->GetStartNode();
                        UEdNode_QuestSystemNode* EndNode = EdgeNode->GetEndNode();
                        UQuestSystemGraphEdge* Edge = EdgeNode->QuestSystemGraphEdge;

                        if (StartNode == nullptr || EndNode == nullptr || Edge ==
nullptr)
                        {

//LOG_ERROR(TEXT("UEdGraph_GenericGraph::RebuildGenericGraph add edge failed."));
                                continue;
                        }

                        EdgeMap.Add(Edge, EdgeNode);

                        Edge->Graph = Graph;
                        Edge->Rename(nullptr, Graph, REN_DontCreateRedirectors |
REN_DoNotDirty);
                        Edge->StartNode = StartNode->QuestSystemGraphNode;
                        Edge->EndNode = EndNode->QuestSystemGraphNode;
                        Edge->StartNode->Edges.Add(Edge->EndNode, Edge);
                }

                if (UEdNode_QuestSystemRoot* RootNode =
Cast<UEdNode_QuestSystemRoot>(Nodes[i]))
```

```
		{
			if (RootNode->QuestSystemGraphNode == nullptr)
				continue;
			QuestSystem->RootNodes.Add(RootNode->QuestSystemGraphNode);
		}

	}

	for (int i = 0; i < QuestSystem->AllNodes.Num(); ++i)
	{
		UQuestSystemGraphNode* Node = QuestSystem->AllNodes[i];
		if (Node->ParentNodes.Num() == 0)
		{
			//use this line below if you want to automatically detect
rootnodes based on their parent
			//Graph->RootNodes.Add(Node);
			SortNodes(Node);
		}

		Node->QuestGraph = Graph;
		Node->QuestSystem = QuestSystem;
		Node->Rename(nullptr, Graph, REN_DontCreateRedirectors |
REN_DoNotDirty);
	}

	QuestSystem->RootNodes.Sort([&](const UQuestSystemGraphNode& L, const
UQuestSystemGraphNode& R)
	{
		UEdNode_QuestSystemNode* EdNode_LNode = NodeMap[&L];
		UEdNode_QuestSystemNode* EdNode_RNode = NodeMap[&R];
		return EdNode_LNode->NodePosX < EdNode_RNode->NodePosX;
	});
}

bool UEdGraph_QuestSystemGraph::Modify(bool bAlwaysMarkDirty)
{
	bool Rtn = Super::Modify(bAlwaysMarkDirty);

	GetQuestSystemGraph()->Modify();

	for (int32 i = 0; i < Nodes.Num(); ++i)
	{
		Nodes[i]->Modify();
	}

	return Rtn;
}

void UEdGraph_QuestSystemGraph::Clear()
{
	QuestSystem->ClearGraph();
	QuestSystem->NodeMap.Reset();
	NodeMap.Reset();
	EdgeMap.Reset();

	for (int i = 0; i < Nodes.Num(); ++i)
	{
		if (UEdNode_QuestSystemNode* EdNode =
Cast<UEdNode_QuestSystemNode>(Nodes[i]))
```

```
		{
			UQuestSystemGraphNode* QuestGraphNode =
EdNode->QuestSystemGraphNode;
			if (QuestGraphNode)
			{
				QuestGraphNode->ParentNodes.Reset();
				QuestGraphNode->ChildrenNodes.Reset();
				QuestGraphNode->Edges.Reset();
			}
		}
	}
}
```

4.2.4 *Graph Schema*

Berikut merupakan pembuatan *graph schema quest* yang berfungsi dalam mengatur *command action* pada *graph* yang telah dibuat untuk membuat *node* baru, mengubah visual *node edge,* serta memberikan aturan pada koneksi antar *node.* Kelas ini akan memberikan perintah untuk setiap aksi yang dilakukan *user* pada area *graph.* Seperti saat melakukan koneksi *node,* dimana *user* akan diberikan *notifikasi* ketika terdapat koneksi yang terbalik.

Segmen Program 4.6 *Quest Graph Schema*

```
#include "AssetGraphSchema_QuestSystem.h"
#include "ToolMenus.h"
#include "Quest_System_Editor.h"
#include "AIGraphTypes.h"
#include "ConnectionDrawPolicy_QuestSystem.h"
#include "Framework/Commands/GenericCommands.h"
#include "Settings/EditorStyleSettings.h"
#include "QuestSystemEditorUtils.h"
#include "EdNode_QuestSystemNode.h"
#include "EdNode_QuestSystemTask.h"
#include "EdNode_QuestSystemState.h"
#include "EdNode_QuestSystemEdge.h"
#include "EdNode_QuestSystemRoot.h"
#include "GraphEditorActions.h"
#include "QuestSystemGraph.h"
#include "QuestSystemGraphEdge.h"
#include "QuestSystemGraphNode_Root.h"
#include "QuestSystemGraphNode_State.h"
#include "QuestSystemGraphNode_Objective.h"
#include "Task/QuestTask.h"

void UAssetGraphSchema_QuestSystem::GetGraphContextActions(FGraphContextMenuBuilder&
ContextMenuBuilder) const
{
	UQuestSystemGraph* Graph =
CastChecked<UQuestSystemGraph>(ContextMenuBuilder.CurrentGraph->GetOuter());

	if (Graph->NodeType == nullptr)
	{
		return;
	}
```

```
	const bool bNoParent = (ContextMenuBuilder.FromPin == NULL);

	const FText AddToolTip = LOCTEXT("NewGenericGraphNodeTooltip", "Add node
here");

	TSet<TSubclassOf<UQuestSystemGraphNode> > Visited;
	FQuest_System_EditorModule& EditorModule =
FModuleManager::GetModuleChecked<FQuest_System_EditorModule>(TEXT("Quest_System_Edito
r"));
	FGraphNodeClassHelper* ClassCache = EditorModule.GetClassCache().Get();

	FCategorizedGraphActionListBuilder TasksBuilder(TEXT("Task"));

	TArray<FGraphNodeClassData> NodeClasses;
	ClassCache->GatherClasses(UQuestTask::StaticClass(), NodeClasses);


	for (auto& NodeClass : NodeClasses)
	{
		if (NodeClass.GetClass(true) == UQuestTask::StaticClass())
			continue;


		UE_LOG(LogTemp, Warning, TEXT("NodeClass: %s"),
*FName::NameToDisplayString(NodeClass.ToString(), false));

		const FText NodeTypeName =
FText::FromString(FName::NameToDisplayString(NodeClass.ToString(), false));

		TSharedPtr<FAssetSchemaAction_QuestSystem_NewNode> AddOpAction =
UAssetGraphSchema_QuestSystem::AddNewNodeAction(TasksBuilder,
NodeClass.GetCategory(), NodeTypeName, FText::GetEmpty());
		UClass* GraphNodeClass = UEdNode_QuestSystemTask::StaticClass();


		UEdNode_QuestSystemNode* OpNode =
NewObject<UEdNode_QuestSystemNode>(ContextMenuBuilder.OwnerOfTemporaries,
GraphNodeClass);
		OpNode->ClassData = NodeClass;
		AddOpAction->NodeTemplate = OpNode;
		AddOpAction->NodeTemplate->QuestSystemGraphNode =
NewObject<UQuestSystemGraphNode>(AddOpAction->NodeTemplate,
UQuestSystemGraphNode_Objective::StaticClass());// NodeClass.GetClass(true));
		if (UQuestSystemGraphNode_Objective* ObjectiveNode = Cast<
UQuestSystemGraphNode_Objective>(AddOpAction->NodeTemplate->QuestSystemGraphNode))
		{
			if
(NodeClass.GetClass(true)->IsChildOf(UQuestTask::StaticClass()))
			{

ObjectiveNode->Tasks.Add(NewObject<UQuestTask>(AddOpAction->NodeTemplate,
NodeClass.GetClass(true)));
			}
			AddOpAction->NodeTemplate->QuestSystemGraphNode->QuestGraph =
Graph;
			AddOpAction->NodeTemplate->QuestSystemGraphNode->ID =
FQuestSystemEditorUtils::FindUniqueNodeName("TaskNode");
```

```
AddOpAction->NodeTemplate->QuestSystemGraphNode->QuestGraph->QuestSystems =
Graph->QuestSystems;
		}
	}
	ContextMenuBuilder.Append(TasksBuilder);


	FCategorizedGraphActionListBuilder StateBuilder(TEXT("Quest State"));
	//State : Complete Quest
	TSharedPtr<FAssetSchemaAction_QuestSystem_NewNode> AddOpAction =
UAssetGraphSchema_QuestSystem::AddNewNodeAction(StateBuilder, FText::GetEmpty(),
LOCTEXT("QuestSystemGraphNodeAction", "Complete Quest"),
LOCTEXT("QuestSystemGraphNodeTooltip", "Complete The Quest"));
	UEdNode_QuestSystemNode* OpNode =
NewObject<UEdNode_QuestSystemNode>(ContextMenuBuilder.OwnerOfTemporaries,
UEdNode_QuestSystemState::StaticClass());
	AddOpAction->NodeTemplate = OpNode;
	AddOpAction->NodeTemplate->QuestSystemGraphNode =
NewObject<UQuestSystemGraphNode>(AddOpAction->NodeTemplate,
UQuestSystemGraphNode_State::StaticClass());
	AddOpAction->NodeTemplate->QuestSystemGraphNode->QuestGraph = Graph;
	AddOpAction->NodeTemplate->QuestSystemGraphNode->ID =
FQuestSystemEditorUtils::FindUniqueNodeName("StateNode");
	AddOpAction->NodeTemplate->QuestSystemGraphNode->QuestGraph->QuestSystems =
Graph->QuestSystems;
	if (UQuestSystemGraphNode_State* NodeState = Cast<
UQuestSystemGraphNode_State>(AddOpAction->NodeTemplate->QuestSystemGraphNode))
	{
		NodeState->QuestState = EQuestState::E_Complete;
	}
	//State : Fail Quest
	AddOpAction = UAssetGraphSchema_QuestSystem::AddNewNodeAction(StateBuilder,
FText::GetEmpty(), LOCTEXT("QuestSystemGraphNodeAction", "Fail Quest"),
LOCTEXT("QuestSystemGraphNodeTooltip", "Fail The Quest"));
	OpNode =
NewObject<UEdNode_QuestSystemNode>(ContextMenuBuilder.OwnerOfTemporaries,
UEdNode_QuestSystemState::StaticClass());
	AddOpAction->NodeTemplate = OpNode;
	AddOpAction->NodeTemplate->QuestSystemGraphNode =
NewObject<UQuestSystemGraphNode>(AddOpAction->NodeTemplate,
UQuestSystemGraphNode_State::StaticClass());
	AddOpAction->NodeTemplate->QuestSystemGraphNode->QuestGraph = Graph;
	AddOpAction->NodeTemplate->QuestSystemGraphNode->ID =
FQuestSystemEditorUtils::FindUniqueNodeName("StateNode");
	AddOpAction->NodeTemplate->QuestSystemGraphNode->QuestGraph->QuestSystems =
Graph->QuestSystems;
	if (UQuestSystemGraphNode_State* NodeState = Cast<
UQuestSystemGraphNode_State>(AddOpAction->NodeTemplate->QuestSystemGraphNode))
	{
		NodeState->QuestState = EQuestState::E_Fail;
	}
	ContextMenuBuilder.Append(StateBuilder);
}

const FPinConnectionResponse UAssetGraphSchema_QuestSystem::CanCreateConnection(const
UEdGraphPin* A, const UEdGraphPin* B) const
{

	// Make sure the pins are not on the same node
```

```
	if (A->GetOwningNode() == B->GetOwningNode())
	{
		return FPinConnectionResponse(CONNECT_RESPONSE_DISALLOW,
LOCTEXT("PinErrorSameNode", "Can't connect node to itself"));
	}

	const UEdGraphPin* Out = A;
	const UEdGraphPin* In = B;

	UEdNode_QuestSystemNode* EdNode_Out =
Cast<UEdNode_QuestSystemNode>(Out->GetOwningNode());
	UEdNode_QuestSystemNode* EdNode_In =
Cast<UEdNode_QuestSystemNode>(In->GetOwningNode());

	if (EdNode_Out == nullptr || EdNode_In == nullptr)
	{
		return FPinConnectionResponse(CONNECT_RESPONSE_DISALLOW,
LOCTEXT("PinError", "Not a valid UQuestGraphEdNode"));
	}

	//Determine if we can have cycles or not
	bool bAllowCycles = false;
	auto EdGraph =
Cast<UEdGraph_QuestSystemGraph>(Out->GetOwningNode()->GetGraph());
	if (EdGraph != nullptr)
	{
		bAllowCycles = EdGraph->GetQuestSystemGraph()->bCanBeCyclical;
	}

	// check for cycles
	FNodeVisitorCycleChecker CycleChecker;
	if (!bAllowCycles && !CycleChecker.CheckForLoop(Out->GetOwningNode(),
In->GetOwningNode()))
	{
		return FPinConnectionResponse(CONNECT_RESPONSE_DISALLOW,
LOCTEXT("PinErrorCycle", "Can't create a graph cycle"));
	}

	FText ErrorMessage;


	if (EdNode_Out->QuestSystemGraphNode->GetOwningQuestGraph()->bEdgeEnabled)
	{
		return
FPinConnectionResponse(CONNECT_RESPONSE_MAKE_WITH_CONVERSION_NODE,
LOCTEXT("PinConnect", "Connect nodes with edge"));
	}
	else
	{
		return FPinConnectionResponse(CONNECT_RESPONSE_MAKE,
LOCTEXT("PinConnect", "Connect nodes"));
	}


	return FPinConnectionResponse(CONNECT_RESPONSE_MAKE, LOCTEXT("PinConnect",
"Connect nodes"));
}

bool UAssetGraphSchema_QuestSystem::TryCreateConnection(UEdGraphPin* A, UEdGraphPin*
```

```
B) const
{
	// We don't actually care about the pin, we want the node that is being
dragged between
	UEdNode_QuestSystemNode* NodeA =
Cast<UEdNode_QuestSystemNode>(A->GetOwningNode());
	UEdNode_QuestSystemNode* NodeB =
Cast<UEdNode_QuestSystemNode>(B->GetOwningNode());

	// Check that this edge doesn't already exist
	if (NodeA->GetOutputPin())
	{
		for (UEdGraphPin* TestPin : NodeA->GetOutputPin()->LinkedTo)
		{
			UEdGraphNode* ChildNode = TestPin->GetOwningNode();
			if (UEdNode_QuestSystemEdge* EdNode_Edge =
Cast<UEdNode_QuestSystemEdge>(ChildNode))
			{
				ChildNode = EdNode_Edge->GetEndNode();
			}

			if (ChildNode == NodeB)
				return false;
		}
	}
	if (UEdNode_QuestSystemState* StateNode =
Cast<UEdNode_QuestSystemState>(NodeA))
	{
		Super::TryCreateConnection(NodeB->GetOutputPin(),
NodeA->GetInputPin());
		return true;
	}
	else if (UEdNode_QuestSystemRoot* RootNode =
Cast<UEdNode_QuestSystemRoot>(NodeB))
	{
		Super::TryCreateConnection(NodeB->GetOutputPin(),
NodeA->GetInputPin());
		return true;
	}
	else if (NodeA && NodeB)
	{
		// Always create connections from node A to B, don't allow adding in
reverse
		Super::TryCreateConnection(NodeA->GetOutputPin(),
NodeB->GetInputPin());
		return true;
	}
	else
	{
		return false;
	}
}
```

4.2.5 *Editor Graph Node*

Berikut merupakan pembuatan editor graph node yang berperan sebagai interface dari node yang dibuat. Kelas ini juga berfungsi dalam mengalokasikan pin input dan output.

Segmen Program 4.7 *Quest Editor Graph Node*

```
include "EdNode_QuestSystemNode.h"
#include "QuestSystemGraphNode.h"

UEdNode_QuestSystemNode::UEdNode_QuestSystemNode()
{
	bIsReadOnly = false;
	bCanRenameNode = true;
}

void UEdNode_QuestSystemNode::AllocateDefaultPins()
{
	CreatePin(EGPD_Input, "MultipleNodes", FName(), TEXT("In"));
	CreatePin(EGPD_Output, "MultipleNodes", FName(), TEXT("Out"));
}

UEdGraphPin* UEdNode_QuestSystemNode::GetInputPin() const
{
	//return Pins[0];
	for (int32 PinIndex = 0; PinIndex < Pins.Num(); PinIndex++)
	{
		if (Pins[PinIndex]->Direction == EGPD_Input)
		{
			return Pins[PinIndex];
		}
	}

	return nullptr;
}

UEdGraphPin* UEdNode_QuestSystemNode::GetOutputPin() const
{
	//return Pins[1];
	for (int32 PinIndex = 0; PinIndex < Pins.Num(); PinIndex++)
	{
		if (Pins[PinIndex]->Direction == EGPD_Output)
		{
			return Pins[PinIndex];
		}
	}

	return nullptr;
}
```

4.2.6 *MyQuest*

Berikut merupakan pembuatan MyQuest yang berperan sebagai tempat membuat dan mengatur quest yang akan ditampilkan pada editor. Dengan memilih quest yang terdapat pada tab ini, akan menampilkan detail atribut pada property panel. Juga terdapat fungsi untuk membuat tombol untuk melakukan penambahan *quest graph.* Ketika *quest* yang baru telah ditambahkan, *MyQuest* akan memberi tahu kepada *Asset Editor* untuk membuka *quest* tersebut

pada *tab* yang baru.

Segmen Program 4.8 *Quest Editor Graph Node*

```
#include "SMyQuest.h"
#include "Framework/Commands/GenericCommands.h"
#include "GraphEditorActions.h"
#include "SGraphActionMenu.h"
#include "SGraphPalette.h"
#include "QuestSystemEditorUtils.h"
#include "Quest_System_EditorCommands.h"
#include "Widgets/Input/SSearchBox.h"
#include "Quest_System_EditorCommands.h"
#include "SlateOptMacros.h"
#include "SQuestPalette.h"

BEGIN_SLATE_FUNCTION_BUILD_OPTIMIZATION
void SMyQuest::Construct(const FArguments& InArgs, TWeakPtr<FAssetEditor_QuestSystem>
InQuestEditor, const UQuestSystemGraph* InQuestSystemGraph)
{
	bNeedsRefresh = false;
	bShowReplicatedVariablesOnly = false;
	QuestEditorPtr = InQuestEditor;
	EdGraph = nullptr;
	TSharedPtr<SWidget> ToolbarBuilderWidget = TSharedPtr<SWidget>();

	SAssignNew(FilterBox, SSearchBox)
		.OnTextChanged(this, &SMyQuest::OnFilterTextChanged);

	// create the main action list piece of this widget
	SAssignNew(GraphActionMenu, SGraphActionMenu, false)
		.OnGetFilterText(this, &SMyQuest::GetFilterText)
		.OnCreateWidgetForAction(this, &SMyQuest::OnCreateWidgetForAction)
		.OnCollectAllActions(this, &SMyQuest::CollectAllActions)
		.OnCollectStaticSections(this, &SMyQuest::CollectStaticSections)
		.OnActionDragged(this, &SMyQuest::OnActionDragged)
		.OnCategoryDragged(this, &SMyQuest::OnCategoryDragged)
		.OnActionSelected(this, &SMyQuest::OnGlobalActionSelected)
		.OnActionDoubleClicked(this, &SMyQuest::OnActionDoubleClicked)
		.OnContextMenuOpening(this, &SMyQuest::OnContextMenuOpening)
		.OnCategoryTextCommitted(this, &SMyQuest::OnCategoryNameCommitted)
		.OnCanRenameSelectedAction(this,
&SMyQuest::CanRequestRenameOnActionNode)
		.OnGetSectionTitle(this, &SMyQuest::OnGetSectionTitle)
		.OnGetSectionWidget(this, &SMyQuest::OnGetSectionWidget)
		.OnActionMatchesName(this, &SMyQuest::HandleActionMatchesName)
		.AlphaSortItems(false)
		.UseSectionStyling(true);

	ChildSlot
	[
		SNew(SVerticalBox)

			+ SVerticalBox::Slot()
			.AutoHeight()
			[
				SNew(SBorder)
```

```
                                        .Padding(4.0f)

.BorderImage(FAppStyle::GetBrush("ToolPanel.GroupBorder"))

.AddMetaData<FTagMetaData>(FTagMetaData(TEXT("MyQuestPanel")))
                                        [
                                                SNew(SVerticalBox)

                                                + SVerticalBox::Slot()
                                                .AutoHeight()
                                                [
                                                        ToolbarBuilderWidget.ToSharedRef()
                                                ]

                                                + SVerticalBox::Slot()
                                                .AutoHeight()
                                                [
                                                        SNew(SHorizontalBox)

                                                        + SHorizontalBox::Slot()
                                                        .FillWidth(1.0f)
                                                        .VAlign(VAlign_Center)
                                                        [
                                                                FilterBox.ToSharedRef()
                                                        ]
                                                ]
                                        ]
                        ]

                        + SVerticalBox::Slot()
                        .FillHeight(1.0f)
                        [
                                GraphActionMenu.ToSharedRef()
                        ]
        ];
        //ResetLastPinType();
        if (!QuestEditorPtr.IsValid())
        {
                Refresh();
        }

        TMap<int32, bool> ExpandedSections;
        ExpandedSections.Add(QuestSectionID::QUESTGRAPH, true);

        GraphActionMenu->SetSectionExpansion(ExpandedSections);

        FCoreUObjectDelegates::OnObjectPropertyChanged.AddRaw(this,
&SMyQuest::OnObjectPropertyChanged);
}
```

4.3 Pembuatan Sistem *Runtime*

Pada sub-bab ini akan membahas mengenai pembuatan *runtime quest* yang meliputi pembuatan *quest component, quest system graph, quest system, quest node, event, condition,* serta *task.*

4.3.1 Pembuatan *Quest Component*

Berikut merupakan pembuatan *quest component.* Kelas ini memiliki peran dalam menjalankan *quest asset* yang telah dibuat oleh *user*. Saat *user* akan menjalankan *quest* yang telah didesain, *component* ini harus di pasang pada *player controller* agar dapat terhubung dengan *quest asset.* Saat *quest* telah dijalankan, semua *reference* akan langsung dimasukkan pada atribut *quest*. Kemudian sistem akan melakukan pengecekan terhadap *condition* yang terdapat pada tiap *quest* terlebih dahulu. Apabila terdapat *quest* dimana *condition*nya telah terpenuhi, maka status dari *quest* tersebut akan menjadi *unlock*ed. Hal ini akan berguna ketika *user* ingin membuat *quest* tersebut dapat diaktifkan melalui *NPC* atau elemen *gameplay* lainnya. Namun apabila *user* ingin mengaktifkan langsung *quest* tersebut ketika *condition* telah memenuhi, *user* dapat mengaktifkan parameter *AutoActivateQuest* pada atribut *quest.* Setelah itu, sistem akan langsung memulai *quest* sesuai progres *node* yang telah dilalui.

Setiap kali *node objective* telah diselesaikan, sebuah *event* akan dijalankan dimana pada *quest component* akan langsung melakukan *evaluasi* pada tiap *quest. Quest yang dievaluasi* akan dilakukan pengecekan *condition* tiap *children node* pada current *node.* Apabila pada terdapat *condition* yang memenuhi, maka sistem akan memilih *node* tersebut sebagai *next node* dan kemudian akan menjalankan perintah yang terdapat pada *node* tersebut. Semua proses tersebut akan dijalankan hingga pada *state node*. Dimana apabila sebuah *quest* mencapai *state node*, *quest* tersebut akan menotifikasi sistem bahwa *quest* yang dijalankan telah sukses atau gagal.

Segmen Program 4.9 *Quest Component*

```
#include "QuestComponent.h"
#include "QuestSystem.h"
#include "Condition/QuestCondition.h"
#include "Event/QuestEvent.h"
#include "Task/QuestTask.h"
#include "UObject/UObjectIterator.h"
#include "QuestSaveObject.h"
#include "QuestSystemGraph.h"
#include "QuestSystemGraphNode_State.h"
#include "QuestSystemGraphNode_Root.h"
#include "QuestSystemGraphNode_Objective.h"
#include "Kismet/GameplayStatics.h"
#include "GameFramework/Controller.h"

bool UQuestComponent::RunQuestGraphs(TArray<UQuestSystemGraph*> QuestAssets)
{
```

```
	//Set Dependency
	OwningController = Cast<AController>(GetOwner());
	QuestComponent = this;

	for (UQuestSystemGraph* QuestGraph : QuestAssets)
	{
		QuestGraph->OwningController = OwningController;
		QuestGraph->QuestComponent = QuestComponent;

		//Add QuestGraph to array
		QuestGraphs.Add(QuestGraph);

		for (UQuestSystem* Quest : QuestGraph->QuestSystems)
		{
			QuestSystemMap.Emplace(Quest->QuestID, Quest);

			if (!bQuestLoaded)
			{
				Quest->QuestState = EQuestState::E_Locked;
				Quest->CurrentNode = nullptr;
				Quest->VisitedNodeIDs.Empty();
			}
		}
	}
	bool bSuccess = true;


	//CheckQuestCondition
	for (UQuestSystemGraph* QuestGraph : QuestGraphs)
	{
		for (UQuestSystem* QuestSystem : QuestGraph->QuestSystems)
		{
			EvaluateQuestCondition(QuestSystem);
			if (QuestSystem->QuestState == EQuestState::E_Active)
			{
				if (QuestSystem->CurrentNode != nullptr)
				{
					BeginNode(QuestSystem);
				}
				else if (QuestSystem->CurrentNode == nullptr)
				{
					RestartQuest(QuestSystem->QuestID);
				}
			}
		}
	}

	OnQuestObjectiveUpdated.Broadcast();
	return bSuccess;
}

void UQuestComponent::UpdateQuest()
{
	for (const auto& It : QuestSystemMap)
	{
		FName Key = It.Key;
		UQuestSystem* QuestSystem = It.Value;
```

```
		EvaluateQuestCondition(QuestSystem);
		if (QuestSystem->CurrentNode != nullptr)
		{
			if (UQuestSystemGraphNode_Objective* ObjectiveNode =
Cast<UQuestSystemGraphNode_Objective>(QuestSystem->CurrentNode))
			{
				if (ObjectiveNode->IsAllTaskCompleted())
				{
					//evaluate next node condition
					EvaluateNextNodeCondition(QuestSystem);
				}
			}
		}
		else if (QuestSystem->QuestState == EQuestState::E_Active &&
QuestSystem->CurrentNode == nullptr)
		{
			RestartQuest(QuestSystem->QuestID);
		}

	}
}

void UQuestComponent::EvaluateNextNodeCondition(UQuestSystem* QuestParam)
{
	UQuestSystemGraphNode* NextNode = nullptr;
	for (UQuestSystemGraphNode* Node : QuestParam->CurrentNode->ChildrenNodes)
	{
		if (NextNode == nullptr && Node->Conditions.IsEmpty())
		{
			NextNode = Node;
		}
		else if (!Node->Conditions.IsEmpty())
		{
			int SumConditionMet = 0;
			for (UQuestCondition* Condition : Node->Conditions)
			{
				if (Condition && Condition->IsConditionMet())
				{
					SumConditionMet++;
				}
			}
			if (SumConditionMet >= Node->Conditions.Num())
			{
				NextNode = Node;
				//break loop to ensure the met condition doesn't get
overrided
				break;
			}
		}
	}

	if (NextNode)
	{
		//End of a Node, Launch Event with the End and Both Type
		for (UQuestEvent* Event : QuestParam->CurrentNode->Events)
		{
			if (Event->EventRunType == EEventRunType::E_Both ||
Event->EventRunType == EEventRunType::E_End)
			{
```

```
                        if (!Event->EndLaunched)
                        {
                                Event->BeginEvent();
                                Event->EndLaunched = true;
                        }
                }
        }
        //Go to next node if exist or condition met
        GoToNextNode(QuestParam, NextNode);

    }

}

void UQuestComponent::EvaluateQuestCondition(UQuestSystem* QuestParam)
{
    int SumConditionMet = 0;
    for (UQuestCondition* Condition : QuestParam->Conditions)
    {
        if (Condition && Condition->IsConditionMet())
        {
            SumConditionMet++;
        }
    }

    switch(QuestParam->QuestState)
    {
        case EQuestState::E_Locked:
            if (SumConditionMet >= QuestParam->Conditions.Num())
            {
                if (QuestParam->bAutoActivateQuest)
                {
                    QuestParam->QuestState = EQuestState::E_Active;
                    OnQuestAdded.Broadcast(QuestParam);
                    return;
                }
                QuestParam->QuestState = EQuestState::E_Unlocked;
            }
        case EQuestState::E_Unlocked:
            if (SumConditionMet >= QuestParam->Conditions.Num())
            {
                if (QuestParam->bAutoActivateQuest)
                {
                    QuestParam->QuestState = EQuestState::E_Active;
                    OnQuestAdded.Broadcast(QuestParam);
                    return;
                }
            }
    }
}

void UQuestComponent::GoToNextNode(UQuestSystem* QuestParam, UQuestSystemGraphNode*
NodeParam)
{
    QuestParam->CurrentNode = NodeParam;
    BeginNode(QuestParam);
}

void UQuestComponent::BeginNode(UQuestSystem* QuestParam)
```

```
{
	//add visited node
	QuestParam->VisitedNodeIDs.AddUnique(QuestParam->CurrentNode->ID);

	if (UQuestSystemGraphNode_Root* RootNode =
Cast<UQuestSystemGraphNode_Root>(QuestParam->CurrentNode))
	{
		EvaluateNextNodeCondition(QuestParam);
		return;
	}

	//Start of the Node, Launch Event with the Start and Both Type
	for (UQuestEvent* Event : QuestParam->CurrentNode->Events)
	{
		if (Event->EventRunType == EEventRunType::E_Both ||
Event->EventRunType == EEventRunType::E_Start)
		{
			if (!Event->StartLaunched)
			{
				Event->BeginEvent();
				Event->StartLaunched = true;
			}
		}
	}

	//Begin Node Functionality
	if (UQuestSystemGraphNode_Objective* ObjectiveNode =
Cast<UQuestSystemGraphNode_Objective>(QuestParam->CurrentNode))
	{
		for (UQuestTask* Task : ObjectiveNode->Tasks)
		{
			if (Task)
			{
				Task->TaskBegin();
			}

		}
	}

	if (UQuestSystemGraphNode_State* StateNode =
Cast<UQuestSystemGraphNode_State>(QuestParam->CurrentNode))
	{
		StateNode->BeginState();
	}
}
```

4.3.2 Pembuatan *Quest System Graph*

Berikut merupakan pembuatan *quest system graph* yang berperan sebagai atribut *quest asset* yang dibuat, serta memberikan aturan pada *graph*.

Segmen Program 4.10 *Quest System Graph*

```
#pragma once
#include "CoreMinimal.h"
```

```
#include "GameplayTagContainer.h"
#include "QuestSystemGraph.generated.h"

UCLASS(Blueprintable)
class QUEST_SYSTEM_RUNTIME_API UQuestSystemGraph : public UObject
{
	GENERATED_BODY()
public:
	UQuestSystemGraph();
	virtual ~UQuestSystemGraph();

	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "QuestSystemGraph")
		FString CategoryDisplayName;

	UPROPERTY(BlueprintReadOnly, Category = "QuestSystemGraph")
		TSubclassOf<UQuestSystemGraphNode> NodeType;

	UPROPERTY(BlueprintReadOnly, Category = "QuestSystemGraph")
		TSubclassOf<UQuestSystemGraphEdge> EdgeType;

	UPROPERTY(BlueprintReadOnly, BlueprintReadOnly, Category =
"QuestSystemGraph")
		FGameplayTagContainer GraphTags;


	/** Set of QuestSystem linked with Quest Graph Pages */
	UPROPERTY(BlueprintReadOnly, Category = "QuestSystemGraph")
		TArray<UQuestSystem*> QuestSystems;

	UPROPERTY(BlueprintReadOnly, BlueprintReadOnly, Category =
"QuestSystemGraph")
		bool bEdgeEnabled;

	UPROPERTY(BlueprintReadWrite, Category = "QuestSystemGraph")
		AController* OwningController;

	UPROPERTY(BlueprintReadWrite, Category = "QuestSystemGraph")
		UQuestComponent* QuestComponent;


#if WITH_EDITORONLY_DATA
	/** Set of quest-graph pages */
	UPROPERTY()
	TArray<TObjectPtr<UEdGraph>> QuestGraphPages;


	UPROPERTY(BlueprintReadOnly, Category = "QuestSystemGraph")
		bool bCanRenameNode;

	UPROPERTY(BlueprintReadOnly, Category = "QuestSystemGraph")
		bool bCanBeCyclical;

	/** Set of documents that were being edited in this blueprint, so we can open
them right away */
	UPROPERTY()
	TArray<struct FEditedDocumentInfo> LastEditedDocuments;

	/** Whether or not this blueprint is newly created, and hasn't been opened in
an editor yet */
```

```
	UPROPERTY(/*transient*/ BlueprintReadOnly, Category = "QuestSystemGraph")
	bool bIsNewlyCreated = true;
#endif

	/** Get all graphs in this QuestEditor */
	void GetAllGraphs(TArray<UEdGraph*>& Graphs) const;
};
```

4.3.3 Pembuatan *Quest System*

Berikut merupakan pembuatan *quest system* yang menampung atribut *pada editor graph.*

Segmen Program 4.11 *Quest System*

```
#pragma once
#include "CoreMinimal.h"
#include "GameplayTagContainer.h"
#include "UObject/NoExportTypes.h"
#include "QuestSystem.generated.h"

UENUM(BlueprintType)
enum class EQuestState : uint8
{
	/**Quest is Active*/
	E_Active		UMETA(DisplayName = "ACTIVE"),
	/**Quest is Completed*/
	E_Complete		UMETA(DisplayName = "COMPLETE"),
	/**Quest is Fail*/
	E_Fail			UMETA(DisplayName = "FAIL"),
	/**Quest is Locked, Waiting to be Unlocked*/
	E_Locked		UMETA(DisplayName = "LOCKED"),
	/**Quest is Unlocked, Waiting to be Activated*/
	E_Unlocked		UMETA(DisplayName = "UNLOCKED"),
};

/** Reward Data for Quest*/
USTRUCT(BlueprintType)
struct FRewardData
{
	GENERATED_BODY()
public:
	UPROPERTY(BlueprintReadWrite, EditAnywhere, meta = (DisplayName = "Reward
Tag"), Category = RewardData)
		FString RewardTag;

	UPROPERTY(BlueprintReadWrite, EditAnywhere, meta = (DisplayName = "Reward
Name"), Category = RewardData)
		FString RewardName;

	UPROPERTY(BlueprintReadWrite, EditAnywhere, meta = (DisplayName = "Reward
Quantity"), Category = RewardData)
		int RewardQuantity = 0;

};
```

```
UCLASS(Blueprintable)
class QUEST_SYSTEM_RUNTIME_API UQuestSystem : public UObject
{
	GENERATED_BODY()
public:

	UPROPERTY(VisibleDefaultsOnly, Category = "QuestAssets")
		UQuestSystemGraph* QuestGraph;

	/** Quest ID */
	UPROPERTY(BlueprintReadOnly, VisibleDefaultsOnly, Category = "QuestSystem")
		FName QuestID;

	UPROPERTY(BlueprintReadOnly, EditAnywhere, Category = "QuestSystem")
		FText Description;

	UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = "QuestSystem")
		TArray<FRewardData> Rewards;

	UPROPERTY(BlueprintReadOnly, EditAnywhere, Category = "Conditions")
		bool bCanQuestBeAborted = false;
	/**
	* TRUE : After the conditions met, activate this quest. if there are no
conditions, activate this quest immediately
	* FALSE : After the conditions met, unlock this quest.
	*/
	UPROPERTY(BlueprintReadOnly, EditAnywhere, Category = "Conditions")
		bool bAutoActivateQuest = false;

	/**Conditions to unlock the Quest*/
	UPROPERTY(Instanced, EditAnywhere, BlueprintReadOnly, Category =
"Conditions")
		TArray<UQuestCondition*> Conditions;


	/** List of All Visited Node IDs in Sequence*/
	UPROPERTY(BlueprintReadWrite, Category = "QuestSystem")
		TArray<FName> VisitedNodeIDs;

	UPROPERTY(BlueprintReadWrite, Category = "QuestSystem")
		EQuestState QuestState = EQuestState::E_Locked;

	UPROPERTY(BlueprintReadWrite, Category = "QuestSystem")
		UQuestSystemGraphNode* CurrentNode;

	UPROPERTY(BlueprintReadOnly, Category = "QuestSystem")
		TArray<UQuestSystemGraphNode*> RootNodes;

	UPROPERTY(BlueprintReadOnly, Category = "QuestSystem")
		TArray<UQuestSystemGraphNode*> AllNodes;

	UPROPERTY(BlueprintReadOnly, Category = "QuestSystem")
		TMap<FName, UQuestSystemGraphNode*> NodeMap;


	UFUNCTION(BlueprintCallable, Category = "QuestSystem")
		int GetLevelNum() const;
```

```
	UFUNCTION(BlueprintCallable, Category = "QuestSystem")
		void GetNodesByLevel(int Level, TArray<UQuestSystemGraphNode*>&
Nodes);

	UFUNCTION(BlueprintCallable, Category = "QuestSystem")
		AController* GetOwningController();

	UFUNCTION(BlueprintCallable, Category = "QuestSystem")
		UQuestComponent* GetQuestComponent();

	UFUNCTION(BlueprintCallable, Category = "QuestSystem")
		UQuestSystemGraph* GetOwningQuestGraph() const;
	void ClearGraph();
};
```

4.3.4 Pembuatan *Quest Node*

Berikut merupakan pembuatan *quest node* yang menampung atribut pada *editor graph node.*

Segmen Program 4.12 *Quest Node*

```
#pragma once

#include "CoreMinimal.h"
#include "UObject/NoExportTypes.h"
#include "QuestSystemGraphNode.generated.h"

UENUM(BlueprintType)
enum class ENodeLimits : uint8
{
	Unlimited,
	Limited
};

UCLASS(Blueprintable, EditInlineNew)
class QUEST_SYSTEM_RUNTIME_API UQuestSystemGraphNode : public UObject
{
	GENERATED_BODY()

public:
	UQuestSystemGraphNode();
	virtual ~UQuestSystemGraphNode();


	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "Detail")
		FText Description;

	UPROPERTY(VisibleDefaultsOnly, BlueprintReadOnly, Category = "Detail")
		FName ID;

	UPROPERTY(Instanced, EditAnywhere, BlueprintReadOnly, Category =
"Conditions")
		TArray<UQuestCondition*> Conditions;
```

```
	UPROPERTY(Instanced, EditAnywhere, BlueprintReadOnly, Category = "Events")
		TArray<UQuestEvent*> Events;

	UPROPERTY(BlueprintReadOnly, Category = "QuestSystemGraphNode")
		UQuestSystemGraph* QuestGraph;

	UPROPERTY(BlueprintReadOnly, Category = "QuestSystemGraphNode")
		UQuestSystem* QuestSystem;

	UPROPERTY(BlueprintReadOnly, Category = "QuestSystemGraphNode")
		TArray<UQuestSystemGraphNode*> ParentNodes;

	UPROPERTY(BlueprintReadOnly, Category = "QuestSystemGraphNode")
		TArray<UQuestSystemGraphNode*> ChildrenNodes;

	UPROPERTY(BlueprintReadOnly, Category = "QuestSystemGraphNode")
		TMap<UQuestSystemGraphNode*, UQuestSystemGraphEdge*> Edges;


	UFUNCTION(BlueprintCallable, Category = "QuestSystemGraphNode")
		virtual UQuestSystemGraphEdge* GetEdge(UQuestSystemGraphNode*
ChildNode);

	UFUNCTION(BlueprintCallable, Category = "QuestSystemGraphNode")
		bool IsLeafNode() const;

	UFUNCTION(BlueprintNativeEvent, BlueprintCallable, Category =
"QuestSystemGraphNode")
		FText GetDescription() const;


	virtual FText GetDescription_Implementation() const;

#if WITH_EDITOR
	virtual bool IsNameEditable() const;

	virtual FLinearColor GetBackgroundColor() const;

	virtual FText GetNodeTitle() const;

	virtual FText GetNodeDescription() const;

	virtual void SetNodeTitle(const FText& NewTitle);

	virtual bool CanCreateConnection(UQuestSystemGraphNode* Other, FText&
ErrorMessage);

	virtual bool CanCreateConnectionTo(UQuestSystemGraphNode* Other, int32
NumberOfChildrenNodes, FText& ErrorMessage);
	virtual bool CanCreateConnectionFrom(UQuestSystemGraphNode* Other, int32
NumberOfParentNodes, FText& ErrorMessage);
#endif
};
```

4.3.5 Pembuatan *Event*

Berikut merupakan pembuatan *event* yang akan digunakan ketika awal atau akhir *node* dijalankan. Pada kelas ini akan terdapat *function BeginEvent* yang berperan dalam menjalankan kode yang akan diimplementasikan oleh *user*

Segmen Program 4.13 *Quest Event*

```
#pragma once
#include "CoreMinimal.h"
#include "UObject/NoExportTypes.h"
#include "QuestEvent.generated.h"

UENUM(BlueprintType)
enum class EEventRunType : uint8
{
	/** will trigger at the beggining of the node */
	E_Start		UMETA(DisplayName = "START"),
	/** will trigger if next node conditions met || at the end of state node */
	E_End		UMETA(DisplayName = "END"),
	/** will trigger twice at start and end of node */
	E_Both		UMETA(DisplayName = "BOTH"),
};

UCLASS(Blueprintable, EditInlineNew, HideDropdown)
class QUEST_SYSTEM_RUNTIME_API UQuestEvent : public UObject
{
	GENERATED_BODY()
public:
	UQuestEvent();

	UPROPERTY(BlueprintReadOnly, Category = "QuestEvent")
		UQuestSystemGraph* QuestGraph;

	UPROPERTY(BlueprintReadWrite, Category = "QuestEvent")
		bool bUseQuestID = false;

	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "QuestEvent", meta =
(GetOptions = "GetAllQuestID", EditCondition = "bUseQuestID == true",
HideEditConditionToggle, EditConditionHides))
		FName QuestID;

	UPROPERTY(BlueprintReadWrite, Category = "QuestEvent")
		bool StartLaunched = false;

	UPROPERTY(BlueprintReadWrite, Category = "QuestEvent")
		bool EndLaunched = false;

	UPROPERTY(BlueprintReadOnly, Category = "QuestEvent")
		UQuestSystem* QuestSystem;

	UPROPERTY(BlueprintReadWrite, EditAnywhere, Category = "QuestEvent")
		EEventRunType EventRunType = EEventRunType::E_Start;

	UFUNCTION(BlueprintNativeEvent, BlueprintCallable, Category = "QuestEvent")
		void BeginEvent();
```

```
	virtual void BeginEvent_Implementation();
};
```

4.3.6 Pembuatan *Task*

Berikut merupakan pembuatan *task* yang akan dijalankan ketika *objective node* sedang berlangsung. Pada kelas ini akan terdapat *function TaskBegin* yang akan dijalankan ketika *task* dimulai. Kemudian *function TaskCompleted* akan dijalankan saat task telah selesai.

Segmen Program 4.14 *Quest Task*

```
#pragma once
#include "CoreMinimal.h"
#include "Tickable.h"
#include "UObject/NoExportTypes.h"
#include "QuestTask.generated.h"

UENUM(BlueprintType)
enum class ETaskState : uint8
{
	E_Active		UMETA(DisplayName = "ACTIVE"),
	E_Complete		UMETA(DisplayName = "COMPLETE"),
};
UCLASS(Blueprintable, EditInlineNew, HideDropdown, HideCategories = ("Hidden"))
class QUEST_SYSTEM_RUNTIME_API UQuestTask : public UObject, public
FTickableGameObject
{
	GENERATED_BODY()

public:

	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "Task")
		FText Description;

	//mark the task as optional and skip this task if the other non-optional task
has been completed
	UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = "Task")
		bool bOptional = false;

	/**This task use amount required to complete task?*/
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Task", meta =
(DisplayName = "bUseAmount?"))
		bool bUseAmount = false;

	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Task", meta =
(EditCondition = "bUseAmount == true", EditConditionHides))
		int RequiredAmount = 1;

	UPROPERTY(BlueprintReadOnly, Category = "QuestTask")
		UQuestSystemGraph* QuestGraph;

	UPROPERTY(BlueprintReadOnly, Category = "QuestTask")
		UQuestSystem* QuestSystem;
```

```
	UPROPERTY(BlueprintReadOnly, Category = "QuestTask")
		UQuestSystemGraphNode_Objective* ObjectiveNode;

	UPROPERTY(BlueprintReadWrite, Category = "QuestTask")
		ETaskState TaskState = ETaskState::E_Active;

	UPROPERTY(BlueprintReadWrite, Category = "QuestTask")
		int CurrentAmount = 0;

	UPROPERTY(BlueprintReadWrite, Category = "QuestTask")
		float DistanceToTarget;

	UFUNCTION(BlueprintImplementableEvent, BlueprintCallable, Category =
"QuestTask")
		void TaskBegin();

	UFUNCTION(BlueprintImplementableEvent, BlueprintCallable, Category =
"QuestTask")
		void TaskTick();

	UFUNCTION(BlueprintImplementableEvent, BlueprintCallable, Category =
"QuestTask")
		void OnTaskCompleted();

	UFUNCTION(BlueprintCallable, Category = "QuestTask")
		bool CompleteTask();

	/**Add amount and check if the required amount has fulfilled, then complete
the task*/
	UFUNCTION(BlueprintCallable, Category = "QuestTask")
		void AddAmount(int AmountToAdd);

	UFUNCTION(BlueprintNativeEvent, BlueprintCallable, Category = "QuestTask")
		FText GetDescription() const;

	virtual FText GetDescription_Implementation() const;

	// FTickableGameObject Begin
	virtual void Tick(float DeltaTime) override;
	virtual ETickableTickType GetTickableTickType() const override
	{
		return ETickableTickType::Always;
	}
	virtual TStatId GetStatId() const override
	{
		RETURN_QUICK_DECLARE_CYCLE_STAT(FMyTickableThing,
STATGROUP_Tickables);
	}

private:
	uint32 LastFrameNumberWeTicked = INDEX_NONE;
};
```

4.3.7 Pembuatan *Condition*

Berikut merupakan pembuatan *condition* yang akan digunakan dalam menentukan *quest state*, serta *branching* pada tiap *node.* Pada kelas ini akan terdapat *function IsConditionMet* yang berperan untuk melakukan evaluasi kondisi terhadap suatu kondisi.

Segmen Program 4.15 *Quest* Condition

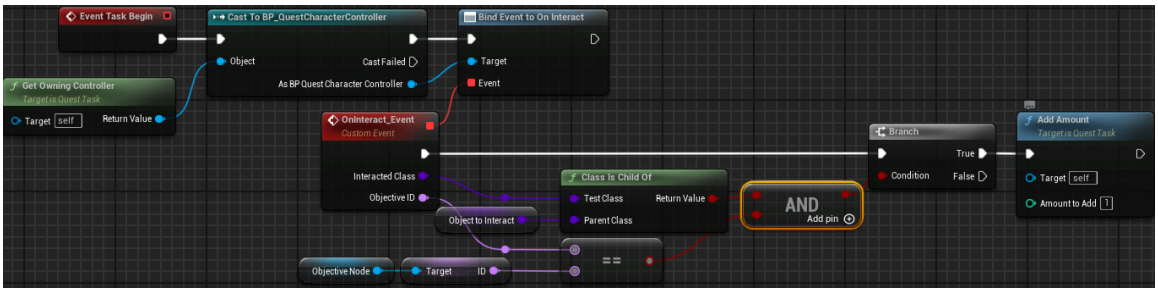
```
#pragma once
#include "CoreMinimal.h"
#include "UObject/NoExportTypes.h"
#include "QuestCondition.generated.h"

class UQuestSystemGraph;

UCLASS(Blueprintable, EditInlineNew, HideDropdown)
class QUEST_SYSTEM_RUNTIME_API UQuestCondition : public UObject
{
	GENERATED_BODY()
public:
	UPROPERTY(BlueprintReadOnly, Category = "QuestCondition")
		UQuestSystemGraph* QuestGraph;

	UPROPERTY(BlueprintReadOnly, Category = "QuestCondition")
		bool bUseQuestID = false;

	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "QuestCondition", meta
= (GetOptions = "GetAllQuestID", EditCondition = "bUseQuestID == true",
HideEditConditionToggle, EditConditionHides))
		FName QuestID;

	UPROPERTY(BlueprintReadOnly, Category = "QuestCondition")
		UQuestSystem* QuestSystem;

	UFUNCTION(BlueprintNativeEvent, BlueprintCallable, Category =
"QuestCondition")
		bool IsConditionMet() const;

		virtual bool IsConditionMet_Implementation() const;
};
```

4.4 Pembuatan Demo Proyek

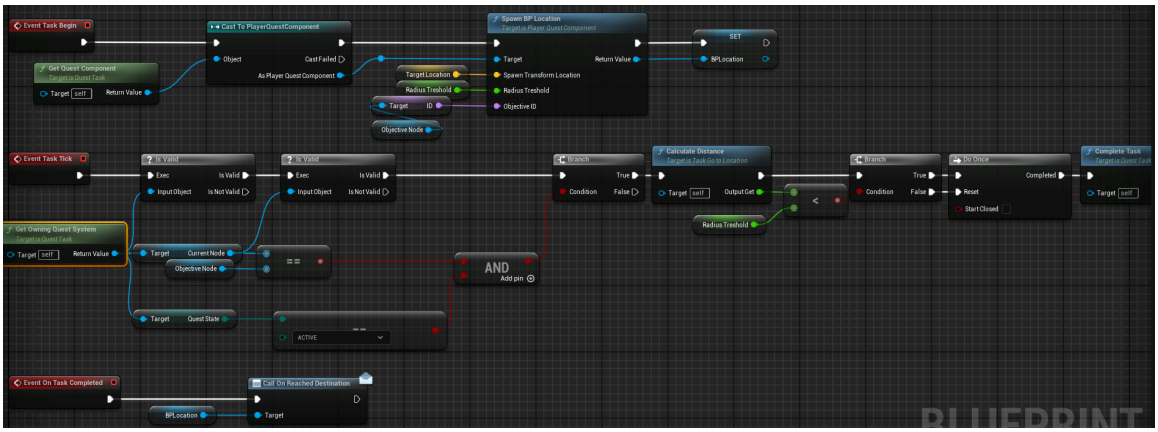
Pada sub-bab ini akan membahas pembuatan demo proyek yang meliputi pembuatan *template task, event*, dan *condition.*

4.4.1 Pembuatan *Template Task*

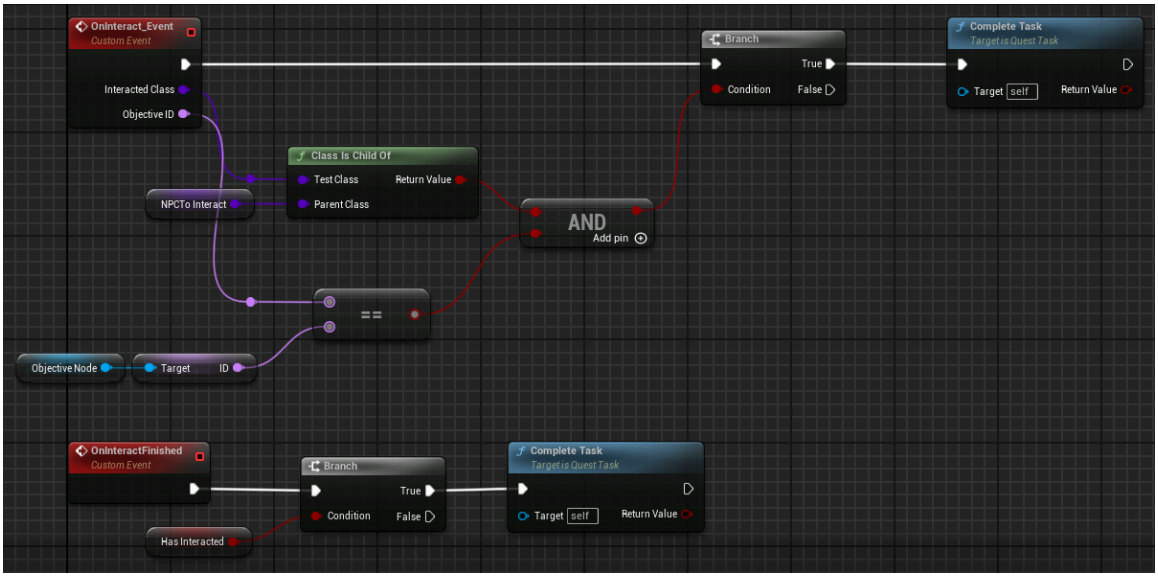
Berikut merupakan pembuatan *template task* yang akan terdiri atas 3 jenis. Gambar 4.3 merupakan pembuatan *Task Interact With Item.* Kemudian gambar 4.4 merupakan pembuatan *Task Go To Location.* Terakhir gambar 4.5 merupakan pembuatan *Task Interact With NPC.*



Gambar 4.3 Pembuatan *Task Interact With Item*



Gambar 4.4 Pembuatan *Task Go To Location*



Gambar 4.5 Pembuatan *Task Interact With NPC*

4.4.2 Pembuatan *Template Event*

Berikut merupakan pembuatan *template event* yang akan terdiri atas 3 jenis. Segmen program 4.15 merupakan pembuatan *Event Start Quest.* Kemudian segmen program 4.16 merupakan pembuatan *Event Restart Quest.* Terakhir segmen program 4.17 merupakan pembuatan *Event Fail Quest.*

Segmen Program 4.16 *Event Start Quest*

```
#include "Event/StartQuest.h"
#include "QuestSystem.h"
#include "QuestComponent.h"
#include "QuestSystemGraph.h"
#include "QuestSystemGraphNode.h"

UStartQuest::UStartQuest()
{
	bUseQuestID = true;
}

void UStartQuest::BeginEvent_Implementation()
{
	GetQuestComponent()->ActivateQuest(QuestID);
}
```

Segmen Program 4.17 *Event Restart Quest*

```
#include "Event/RestartQuest.h"
#include "QuestSystem.h"
#include "QuestComponent.h"
#include "QuestSystemGraph.h"
#include "QuestSystemGraphNode.h"
#include "QuestSystemGraphNode_Objective.h"

URestartQuest::URestartQuest()
{
	bUseQuestID = true;
}

void URestartQuest::BeginEvent_Implementation()
{
	GetQuestComponent()->RestartQuest(QuestID, NodeID, bStartFromNodeID);
}
```

Segmen Program 4.18 *Event Fail Quest*

```
#include "Event/FailQuest.h"
#include "QuestSystem.h"
#include "QuestComponent.h"
#include "QuestSystemGraph.h"
#include "QuestSystemGraphNode.h"

UFailQuest::UFailQuest()
```

```
{
        bUseQuestID = true;
}

void UFailQuest::BeginEvent_Implementation()
{
        GetQuestComponent()->FailQuest(QuestID);

}
```

4.4.3 Pembuatan *Template Condition*

Berikut merupakan pembuatan *template event* yang akan terdiri atas 3 jenis. Segmen program 4.18 merupakan pembuatan *Condition Is Quest Active.* Kemudian segmen program 4.19 merupakan pembuatan *Condition Is Quest Completed.* Terakhir segmen program 4.20 merupakan pembuatan *Condition Is Quest Failed.*

Segmen Program 4.19 *Condition Is Quest Active*

```
#include "Condition/IsQuestActive.h"
#include "QuestComponent.h"
#include "QuestSystem.h"

UIsQuestActive::UIsQuestActive()
{
    bUseQuestID = true;
}
bool UIsQuestActive::IsConditionMet_Implementation() const
{
    if (UQuestSystem* FoundedQuest = GetQuestComponent()->QuestSystemMap.FindRef(QuestID))
    {
        switch (FoundedQuest->QuestState)
        {
            case EQuestState::E_Active :
                return true;
            default:
                return false;
        }
    }
    return false;
}
```

Segmen Program 4.20 *Condition Is Quest Completed*

```
#include "Condition/IsQuestCompleted.h"
#include "QuestComponent.h"
#include "QuestSystem.h"

UIsQuestCompleted::UIsQuestCompleted()
{
    bUseQuestID = true;
}

bool UIsQuestCompleted::IsConditionMet_Implementation() const
```

```
{
    if (UQuestSystem* FoundedQuest = GetQuestComponent()->QuestSystemMap.FindRef(QuestID))
    {
        switch (FoundedQuest->QuestState)
        {
        case EQuestState::E_Complete:
            return true;
        default:
            return false;
        }
    }
    return false;
}
```

Segmen Program 4.21 *Condition Is Quest Failed*

```
#include "Condition/IsQuestFailed.h"
#include "QuestComponent.h"
#include "QuestSystem.h"

UIsQuestFailed::UIsQuestFailed()
{
    bUseQuestID = true;
}

bool UIsQuestFailed::IsConditionMet_Implementation() const
{
    if (UQuestSystem* FoundedQuest = GetQuestComponent()->QuestSystemMap.FindRef(QuestID))
    {
        switch (FoundedQuest->QuestState)
        {
        case EQuestState::E_Fail:
            return true;
        default:
            return false;
        }
    }
    return false;
}
```