**Technical Document**

**Lord of Kings**



**Assassins of Luna**

| Simon Alzate | simonalzate992@gmail.com |
| --- | --- |
| Sean Sheheen | PrinceofDemons7@gmail.com |
| Juan Diego Lugo | LateIsRight@juan.com |
| Melvin Santiago | wunderwaffe4life44@gmail.com |
|  |  |

Contents

[**Technical Design Overview** 4](#_heading=h.gjdgxs)

[**DEVELOPMENT ENVIRONMENT** 5](#_heading=h.30j0zll)

[Developers: 5](#_heading=h.1fob9te)

[Content: 5](#_heading=h.3znysh7)

[**System Architecture** 6](#_heading=h.2et92p0)

[System Architecture Flow Chart 6](#_heading=h.tyjcwt)

Gameplay Framework Classes 8

[System Architecture Context Model Description](#_heading=h.3dy6vkm) 9

[**Module Breakdowns** 9](#_heading=h.1t3h5sf)

[MODULE NAME: 9](#_heading=h.4d34og8)

[**PRODUCTION PLANS** 49](#_heading=h.2s8eyo1)

[Code Review Plan (Optional) 49](#_heading=h.17dp8vu)

[Integration Plan (Required) 50](#_heading=h.3rdcrjn)

[Testing Plans (Required) 52](#_heading=h.26in1rg)

[**Game Folder Hierarchy** 53](#_heading=h.lnxbz9)

# **Technical Design Overview**

We are using an Unreal Engine. Unreal Engine uses OOP as the primary way of drawing objects and ECP as the standard for using the classes. We are using C++ for almost the entire project. Blueprint will be used but later converted as c++. We have 3d parties Assets that will be used in our project (some assets may contain functionalities already bare minimum relative to the assets only not the implementation of the project).

**DEVELOPMENT ENVIRONMENT**

## Developers:

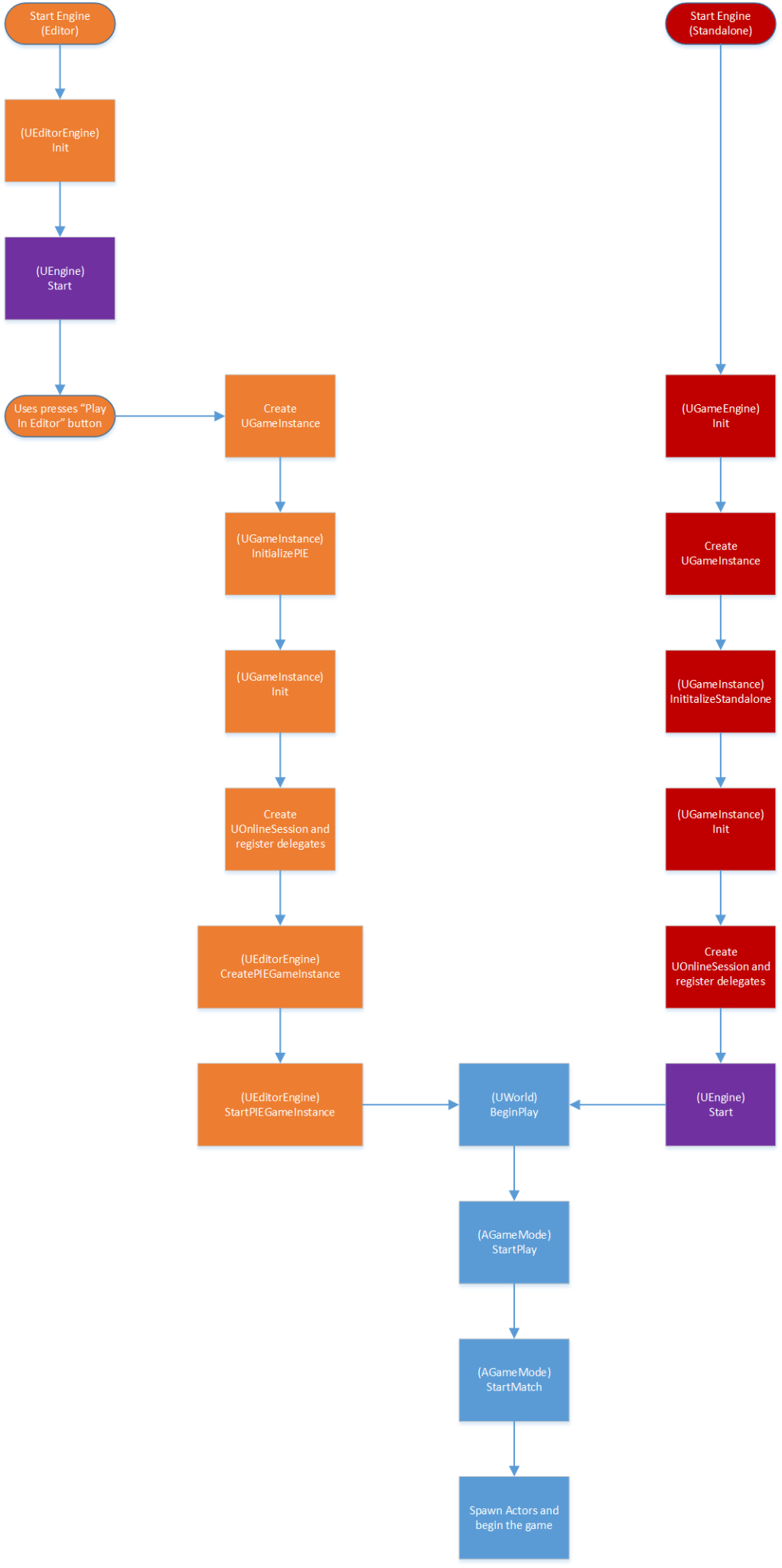
* Visual Studio 2019 (C++)
* Unreal Engine 4.27/5.
* Blueprints.

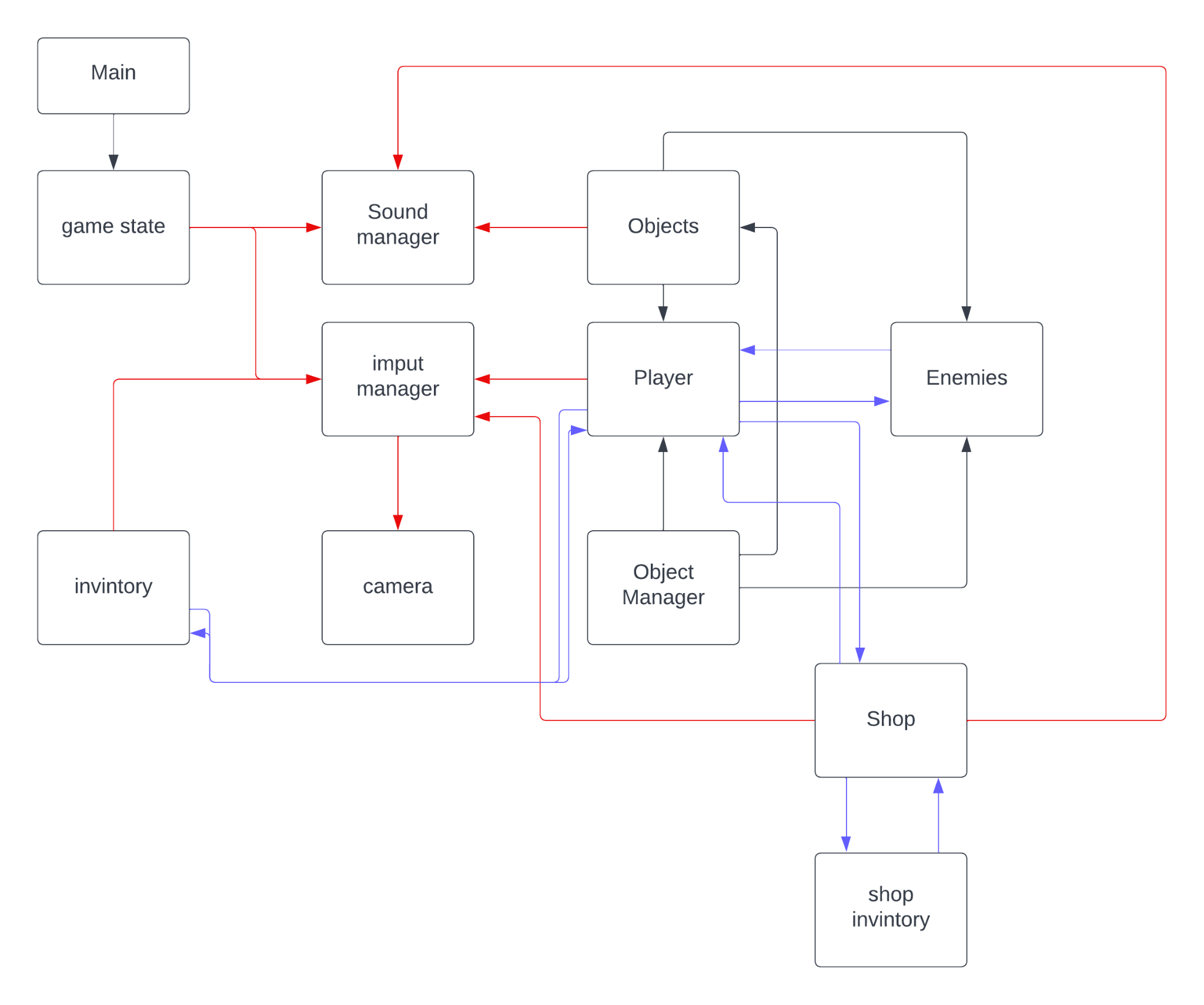
## Content:

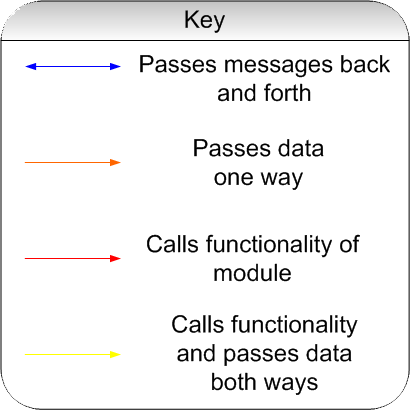
* Maya 2023.
* Blender 3.1.2.
* Photoshop 2022.
* Canva.
* OBS.
* Unreal Marketplace.

# **System Architecture**

## System Architecture Flow Chart





****

**Gameplay Framework Classes**

#### Representing Players, Friends, and Foes in the World

**Pawn** - A [Pawn](https://docs.unrealengine.com/4.26/en-US/InteractiveExperiences/Framework/Pawn) is an Actor that can be an "agent" within the world.

**Character** - A [Character](https://docs.unrealengine.com/4.26/en-US/InteractiveExperiences/Framework/Pawn/Character) is a humanoid-style Pawn. It comes with a CapsuleComponent for collision and a CharacterMovementComponent by default.

#### Controlling Pawns With Player Input or AI Logic

**Controller** - [Controller](https://docs.unrealengine.com/4.26/en-US/InteractiveExperiences/Framework/Controller) is an Actor that is responsible for directing a Pawn.

**PlayerController** - A [PlayerController](https://docs.unrealengine.com/4.26/en-US/InteractiveExperiences/Framework/Controller/PlayerController) is the interface between the Pawn and the human player controlling it.

AIController - An AIController is what it sounds like; a simulated "will" that can control a Pawn.

#### Displaying Information to Players

**HUD** - A [HUD](https://docs.unrealengine.com/4.26/en-US/InteractiveExperiences/Framework/UIAndHUD) is a "heads-up display", or the 2D on-screen display that is common in many games.

**Camera** - The PlayerCameraManager is the "eyeball" for a player and manages how it behaves.

#### Setting and Tracking the Rules of the Game

**GameMode** - The concept of a "game" is split into 2 classes. The [Game Mode and Game State](https://docs.unrealengine.com/4.26/en-US/InteractiveExperiences/Framework/GameMode) is the definition of the game, including things like the game rules and win conditions. It only exists on the server.

**GameState** - The [GameState](https://docs.unrealengine.com/4.26/en-US/InteractiveExperiences/Framework/GameMode#gamestate) contains the state of the game, which could include things like the list of connected players, the score, where the pieces are in a chess game, or the list of what missions you have completed in an open-world game.

**PlayerState** - A PlayerState is the state of a participant in the game, such as a human player or a bot that is simulating a player. Non-player AI that exists as part of the game would not have a PlayerState.

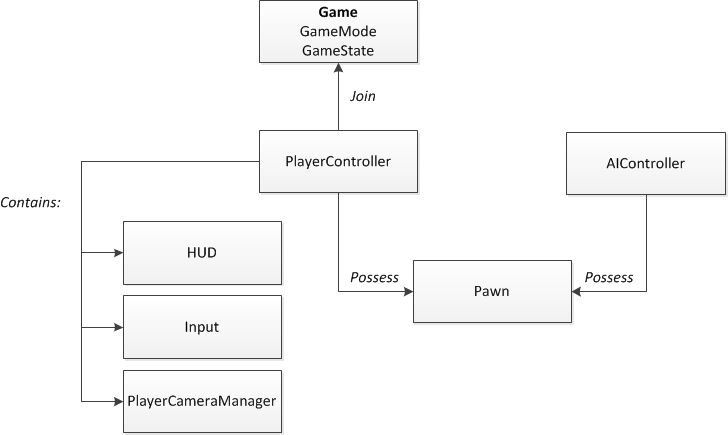
#### 

#### 

#### 

#### 

#### Framework Class Relationships

****

## System Architecture Context Model Description

This is more about C++ examples usages and best Practices. the Code in this are the one we are using for this project

**C++ Classes**

**Object -** Usage Network replication

**Example:**

void USAction::StartAction\_Implementation(AActor\* Instigator){}

GetOwningComponent()->OnActionStarted.Broadcast(GetOwningComponent(), this);

**Actor Component -** ActorComponent is the base class for components that define reusable behavior that can be added to different types of Actors. ActorComponents that have a transform are known as SceneComponents and those that can be rendered are PrimitiveComponents.

**Example:**

void USActionComponent::AddAction(AActor\* Instigator, TSubclassOf<USAction> ActionClass){}

USAction\* NewAction = NewObject<USAction>(GetOwner(), ActionClass);

NewAction->StartAction(Instigator);

**AnimInstance -** Anim instance you are referring to is a C++ object of class UAnimInstance. Animation Blueprint (AnimBP) is in fact a class derived from UAnimInstance (or from any class which inherits from it)

**Example:**

void USAnimInstance::NativeInitializeAnimation(){}

ActionComp = Cast<USActionComponent>(OwningActor->GetComponentByClass(USActionComponent::StaticClass()));

**Character -** A type of pawn that includes the ability to walk around.

**Example:**

SetupPlayerInputComponent(UInputComponent\* PlayerInputComponent){}

PlayerInputComponent->BindAxis("MoveForward", this, &ASCharacter::MoveForward);

PlayerInputComponent->BindAction("PrimaryAttack", IE\_Pressed, this, &ASCharacter::PrimaryAttack);

**Actor -** An Actor is any object that can be placed into a level, such as a Camera, static mesh, or player start location.

**Example:**

ASItemChest::ASItemChest(){}

BaseMesh = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("BaseMesh"));

RootComponent = BaseMesh;

**Game Mode Base -** While certain fundamentals, like the number of players required to play, or the method by which those players join the game, are common to many types of games, limitless rule variations are possible depending on the specific game you are developing.

**Example:**

void ASGameModeBase::InitGame(const FString& MapName, const FString& ErrorMessage){}

// (Save/Load logic moved into new SaveGameSubsystem)

USSaveGameSubsystem\* SG = GetGameInstance()->GetSubsystem<USSaveGameSubsystem>();

**Blueprint Function Library -** Blueprint Function Libraries are a collection of static functions that provide utility functionality not tied to a particular gameplay object. These libraries can be grouped into logical function sets, e.g. AI Blueprint Library, or contain utility functions that provide access to many different functional areas, e.g. System Blueprint Library.

**Example:**

bool USGameplayFunctionLibrary::ApplyDamage(AActor\* DamageCauser, AActor\* TargetActor, float DamageAmount){}

USAttributeComponent\* AttributeComp = USAttributeComponent::GetAttributes(TargetActor);

**Interface -** Interface classes are useful for ensuring that a set of (potentially) unrelated classes implement a common set of functions. This is very useful in cases where some game functionality may be shared by large, complex classes that are otherwise dissimilar. For example, a game might have a system whereby entering a trigger volume can activate traps, alert enemies, or award points to the player. This might be implemented by a "ReactToTrigger" function on traps, enemies, and point-awarders.

**Example:**

UFUNCTION(BlueprintCallable, BlueprintNativeEvent)

void Interact(APawn\* InstigatorPawn);

**Primary Data Asset -** Primary Assets can be manipulated directly by the Asset Manager via their [Primary Asset ID](https://docs.unrealengine.com/en-US/API/Runtime/CoreUObject/UObject/FPrimaryAssetId), which is obtained by calling GetPrimaryAssetId. In order to designate Assets made from a specific UObject class as Primary Assets, override GetPrimaryAssetId to return a valid FPrimaryAssetId structure. Secondary Assets are not handled directly by the Asset Manager, but instead are loaded automatically by the Engine in response to being referenced or used by Primary Assets. By default, only UWorld Assets (levels) are Primary; all other Assets are Secondary.

**Example:**

FPrimaryAssetId GetPrimaryAssetId() const override

return FPrimaryAssetId("Monsters", GetFName());

**Player Controller -** The PlayerController implements functionality for taking the input data from the player and translating that into actions, such as movement, using items, firing weapons, etc.

**Example:**

void ASPlayerController::TogglePauseMenu(){}

UGameplayStatics::SetGamePaused(this, true);

**Player State -** A PlayerState is created for every player on a server (or in a standalone game). PlayerStates are replicated to all clients, and contain network game relevant information about the player, such as playername, score, etc.

**Example:**

void ASPlayerState::SavePlayerState\_Implementation(USSaveGame\* SaveObject){}

SaveObject->SavedPlayers.Add(SaveData);

**Save Game -** Save the contents of the SaveGameObject to a platform-specific save slot/file. Note: This will write out all non-transient properties, the SaveGame property flag is not checked Target is Gameplay Statics

**Example:**

FPlayerSaveData\* USSaveGame::GetPlayerData(APlayerState\* PlayerState){}

return SavedPlayers.FindByPredicate([&](const FPlayerSaveData& Data) { return Data.PlayerID == PlayerID; });

**Developer Settings -**

**Example:**

/\* Default slot name if UI doesn't specify any \*/

UPROPERTY(Config, EditAnywhere, BlueprintReadOnly, Category = "General") FString SaveSlotName;

// Default value while nothing is specified in the DefaultGame.ini

SaveSlotName = "SaveGame02";

**Game Instance Subsystem -** [UGameInstanceSubsystem](https://docs.unrealengine.com/5.0/en-US/API/Runtime/Engine/Subsystems/UGameInstanceSubsystem/) Base class for auto instanced and initialized systems that share the lifetime of the game instance

**Example:**

void USSaveGameSubsystem::HandleStartingNewPlayer(AController\* NewPlayer){}

PS->LoadPlayerState(CurrentSaveGame);

**User Widget -** The user widget is extensible by users through the WidgetBlueprint.

**Example:**

void USWorldUserWidget::NativeTick(const FGeometry& MyGeometry, float InDeltaTime){}

ParentSizeBox->SetVisibility(bIsOnScreen ? ESlateVisibility::HitTestInvisible : ESlateVisibility::Collapsed);

**AI controller -**While the PlayerController relies on the human player to make decisions on what to do, the AIController is more focused on responding to input from the environment and game world. The job of the AIController is to observe the world around it and make decisions and react accordingly without explicit input from a human player.

**Example:**

void ASAIController::BeginPlay(){}

RunBehaviorTree(BehaviorTree);

**BT Server -**

**Example:**

void USBTService\_CheckAttackRange::TickNode(UBehaviorTreeComponent& OwnerComp, uint8\* NodeMemory, float DeltaSeconds){}

BlackBoardComp->SetValueAsBool(AttackRangeKey.SelectedKeyName, (bWithinRange && bHasLOS));

**BT TaskNode -** Task are leaf nodes of behavior tree, which perform actual actions

**Example:**

EBTNodeResult::Type USBTTask\_RangedAttack::ExecuteTask(UBehaviorTreeComponent& OwnerComp, uint8\* NodeMemory){}

return NewProj ? EBTNodeResult::Succeeded : EBTNodeResult::Failed;

**Architecture Diagram**

**Main**

Start -> AGameMode -> Spawn Actors -> Begin Game

**CGame**

UGameEngine -> UGameInstance -> UEngine -> BeginPlay

Spawn actors -> UGameInstance -> Shutdown()

UWorld -> AGameMode -> Spawn Actors and begin the game

**StateMachine**

StartMatch -> Spawn Actors -> RemoveActor() -> Shutdown()

Object Manager -> Object -> Update() -> Render()

**ObjectFactory**

ObjectManager ->AddObject()

**ObjectManager**

Object -> Update() -> Render()

Spawn Actors -> RemoveActor()

**Game System Interactions**

**Components Defines:**

**Character Movement** - The character movement component is an Actor Component that provides an encapsulated movement system with common modes of movement for humanoid characters, including walking, falling, swimming, and flying. The character movement component also features robust network gameplay integration.

**Interaction Comp** - The Interaction component is an Actor Component that provides all gameplay Interaction systems with common modes of general interaction for humanoid characters to the game's actors, including interaction for chests, shrines, doors, and items. The Interaction component also features robust network gameplay integration.

**Attribute Comp** - The Attribute component is an Actor Component that provides all gameplay Attribute/stats systems with common modes of general info for humanoid characters, including Health, mana, stamina, and Mons(money). The Attribute component also features robust network gameplay integration.

**Action Comp** - The Action movement component is an Actor Component that provides all gameplay action combat systems with common modes of general gameplay for humanoid characters, including attack, block, reaction, and doge. The Action component also features robust network gameplay integration.

**Interface -** Gameplay Interface - The Gameplay Interface is an Unreal Interface that communicate all gameplay interactions system functions with common modes of general gameplay for humanoid characters, including interact, block, reaction, and doge. The Gameplay Interface also features robust network gameplay integration.

***PLAYER***

* TO (Player Interaction).
  + AI.
  + Level.
  + Shrines.
  + Items.
  + Water (maybe).
* BY (Who interacts with the player).
  + AI.
  + Shrines.
  + Items.
  + Deathboxes (Outside of bounds/Underwater).
  + Water (maybe).
* COMP (Components of the player).
  + Character Movement
  + Interaction Comp
  + Attribute Comp
  + Action Comp
* Interface.
  + Gameplay Interface

***Zombie AI***

* TO ( Zombie Interaction).
  + AI
  + Level.
  + Player
* BY (Who interacts with the Zombie).
  + Player
  + AI
  + Items
* COMP (Components of the Zombie).
  + Character Movement.
  + Pawn Sensing Comp.
  + Attribute Comp.
  + Action Comp.
* Interface.
  + Gameplay Interface.

***Human Melee AI***

* TO ( Human Melee Interaction).
  + Player.
  + AI.
  + Level.
* BY (Who interacts with the Human Melee).
  + Player.
  + AI.
  + Items.
* COMP (Components of the Human Melee).
  + Character Movement.
  + Pawn Sensing Comp.
  + Attribute Comp.
  + Action Comp.
* Interface.
  + Gameplay Interface.

***Human Ranged AI***

* TO ( Human Range Interaction).
  + Player.
  + AI.
  + Level.
* BY (Who interacts with the Human Range).
  + Player.
  + AI.
  + Items.
* COMP (Components of the Human Range).
  + Character Movement.
  + Pawn Sensing Comp.
  + Attribute Comp.
  + Action Comp.
* Interface.
  + Gameplay Interface.

***Shopkeeper AI***

* TO ( Shopkeeper Interaction).
  + Player.
* BY (Who interacts with the Shopkeeper).
  + Player.
* COMP (Components of the Shopkeeper).
  + N/A.
* Interface.
  + Gameplay Interface.

***Boss AI***

* TO (Boss Interaction).
  + Player.
  + AI.
  + Level.
* BY (Who interacts with the Boss).
  + Player
  + Items.
  + AI.
* COMP (Components of the Boss).
  + Character Movement.
  + Pawn Sensing Comp.
  + Attribute Comp.
  + Action Comp.
* Interface.
  + Gameplay Interface.

***Wildlife AI***

* TO (Wildlife Interaction).
  + Player.
  + AI.
  + Level.
* BY (Who interacts with the Wildlife).
  + Player.
  + AI.
  + Items.
* COMP (Components of the Wildlife).
  + Character Movement.
  + Pawn Sensing Comp.
  + Attribute Comp.
* Interface.
  + Gameplay Interface.

***Weather System***

* TO (Weather Interaction).
  + Level.
* BY (Who interacts with the Weather).
  + Level.
* COMP (Components of the Weather).
  + N/A.
* Interface.
  + N/A.

***Currency System***

* TO (Currency Interaction).
  + Player.
  + AI.
* BY (Who interacts with the Currency).
  + Player.
  + Shopkeeper.
* COMP (Components of the Currency)
  + Attribute Comp.
* Interface.
  + N/A.

***Item Drops***

* TO (Item Interaction).
  + Player.
  + AI.
  + Level.
* BY (Who interacts with the Item).
  + AI.
  + Items.
* COMP (Components of the Item)
  + Attribute Comp.
* Interface.
  + Interaction.

***Skill Tree***

* TO (Skill Interaction).
  + Player
* BY (Who interacts with the Skill).
  + Player.
* COMP (Components of the Skill)
  + Attribute Comp.
* Interface.
  + N/A.

***Audio System***

* TO (Audio Interaction).
  + Player.
  + AI.
  + Level.
* BY (Who interacts with the Audio).
  + Player.
  + AI.
  + Level
* COMP (Components of the Audio)
  + N/A.
* Interface.
  + N/A.

***Level***

* TO (Level Interaction).
  + Player.
  + AI.
* BY (Who interacts with the Level).
  + Player
  + AI.
* COMP (Components of the Level)
  + N/A.
* Interface.
  + Interaction.

***Network System***

* TO (Network Interaction).
  + Player.
  + AI.
  + Level.
  + Items.
  + Objects
* BY (Who interacts with the Network).
  + Player.
  + AI.
  + Level.
  + Items
* COMP (Components of the Network)
  + N/A.
* Interface.
  + Interaction.

***AI Companion System***

* TO (AI Companion Interaction).
  + Player.
  + AI.
  + Level.
* BY (Who interacts with the AI Companion).
  + Player.
  + AI.
* COMP (Components of the AI Companion)
  + Character Movement.
  + Pawn Sensing Comp.
  + Attribute Comp.
  + Action Comp.
* Interface.
  + Interaction.

***Shrine / Save System System***

* TO (Shrine/ Save System Interaction).
  + Player.
  + AI.
* BY (Who interacts with the Shrine/ Save System).
  + Player.
* COMP (Components of the Shrine/ Save System)
  + Attribute Comp.
* Interface.
  + Interaction.

***Dismemberment System***

* TO (Dismemberment Interaction).
  + AI.
* BY (Who interacts with the Dismemberment).
  + Player.
  + AI.
* COMP (Components of the Dismemberment)
  + N/A.
* Interface.
  + N/A.

***Map System***

* TO (Map Interaction).
  + Player.
* BY (Who interacts with the Map).
  + Player.
* COMP (Components of the Map)
  + N/A.
* Interface.
  + N/A.