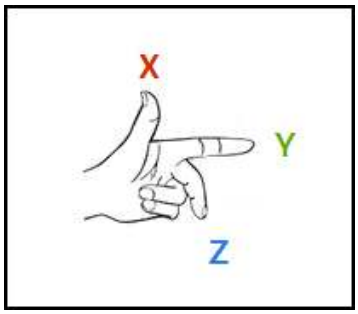
06_Actor Transform

1. Actor Transform

1) 언리얼 좌표계

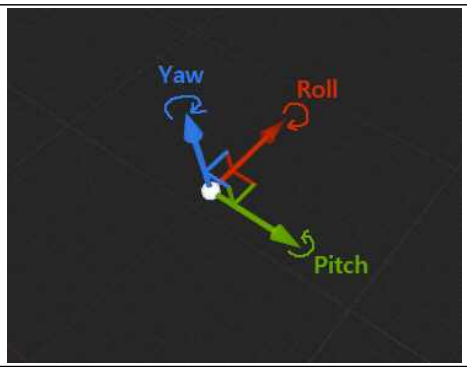
- 언리얼 엔진은 왼손 좌표계를 사용한다.



- 좌표계 : 왼손 좌표계(Z up)

	Forward(1, 0, 0) Right(0, 1, 0) Up(0, 0, 1)

- 회전 (Pitch, Yaw, Roll)

	좌/우 축은 Pitch 상/하 축은 Yaw 앞/뒤 축은 Roll

- Backface Culling : counter - clockwise

  : 반시계 방향의 삼각형이 전면(시계방향의 삼각형들을 그리지 않는다)

2) 액터의 이동

- 먼저, 소환된 액터가 소멸되는 코드를 주석 처리하자.

A1ActorSpawner.cpp

```
#include "A1ActorSpawner.h"
#include "A1Actor.h"

AA1ActorSpawner::AA1ActorSpawner()
{
	// Set this actor to call Tick() every frame.  You can turn this off to improve performance if you don't need it.
	PrimaryActorTick.bCanEverTick = true;

	static ConstructorHelpers::FClassFinder<AActor> SpawnA1TestActorRef(TEXT("/Script/Engine.Blueprint'/Game/Blueprints/BP_A1TestActor.BP_A1TestActor_C'"));
	if (SpawnA1TestActorRef.Succeeded())
	{
		SpawnA1TestActorClass = SpawnA1TestActorRef.Class;
	}
}

void AA1ActorSpawner::BeginPlay()
{
	Super::BeginPlay();

	FVector SpawnLocation(0.0f, 200.0f, 0.0f);
	FRotator SpawnRotation(0.0f, 0.0f, 0.0f);

	SpawnedA1Actor = GetWorld()->SpawnActor<AA1Actor>(SpawnLocation, SpawnRotation);

	// 5초후에 삭제
	//SpawnedA1Actor->SetLifeSpan(5.0f);

	SpawnLocation = FVector(0.0f, -200.0f, 0.0f);
	SpawnedA1TestActor = GetWorld()->SpawnActor<AActor>(SpawnA1TestActorClass, SpawnLocation, SpawnRotation);

	// 월드에서 바로 삭제
	//GetWorld()->DestroyActor(SpawnedA1TestActor);
}

void AA1ActorSpawner::Tick(float DeltaTime)
{
	Super::Tick(DeltaTime);

}
```

① SetActorLocation

- 소환된 A1Actor를 매Tick마다 이동시키는 코드를 추가해보자.
- SetActorLocation함수를 이용하여 액터의 현재 위치를 갱신해 보자.

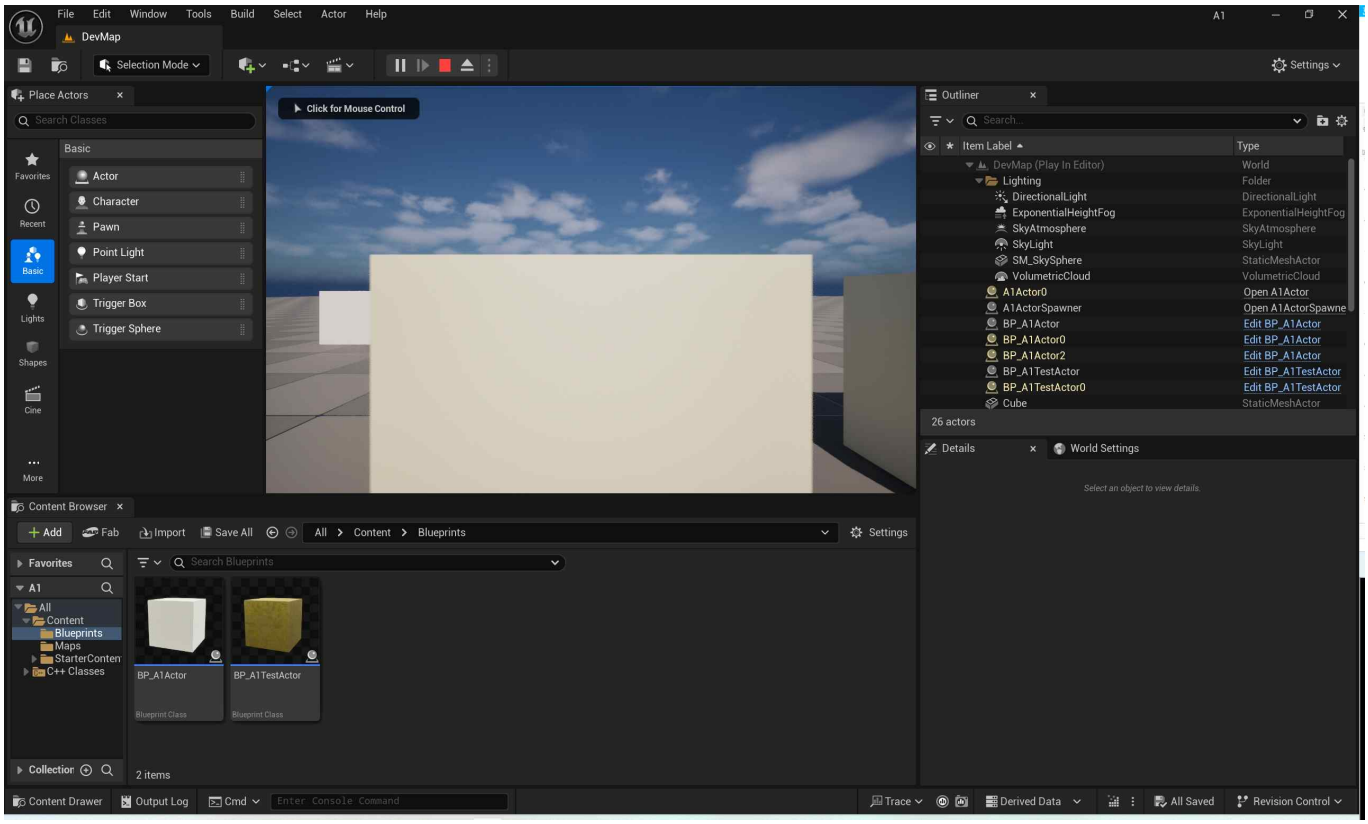
A1Actor.h

```
// Fill out your copyright notice in the Description page of Project Settings.

#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "A1Actor.generated.h"

class UA1Object;

UCLASS()
class A1_API AA1Actor : public AActor
{
	GENERATED_BODY()

public:
	// Sets default values for this actor's properties
	AA1Actor();

protected:
	// Called when the game starts or when spawned
	virtual void BeginPlay() override;

public:
	// Called every frame
	virtual void Tick(float DeltaTime) override;

protected:
	UPROPERTY(VisibleAnywhere, BlueprintReadWrite)
	TObjectPtr<class UStaticMeshComponent> Body;

	UPROPERTY(EditAnywhere, BlueprintReadWrite)
	float MovementSpeed = 50.0f;
};
```

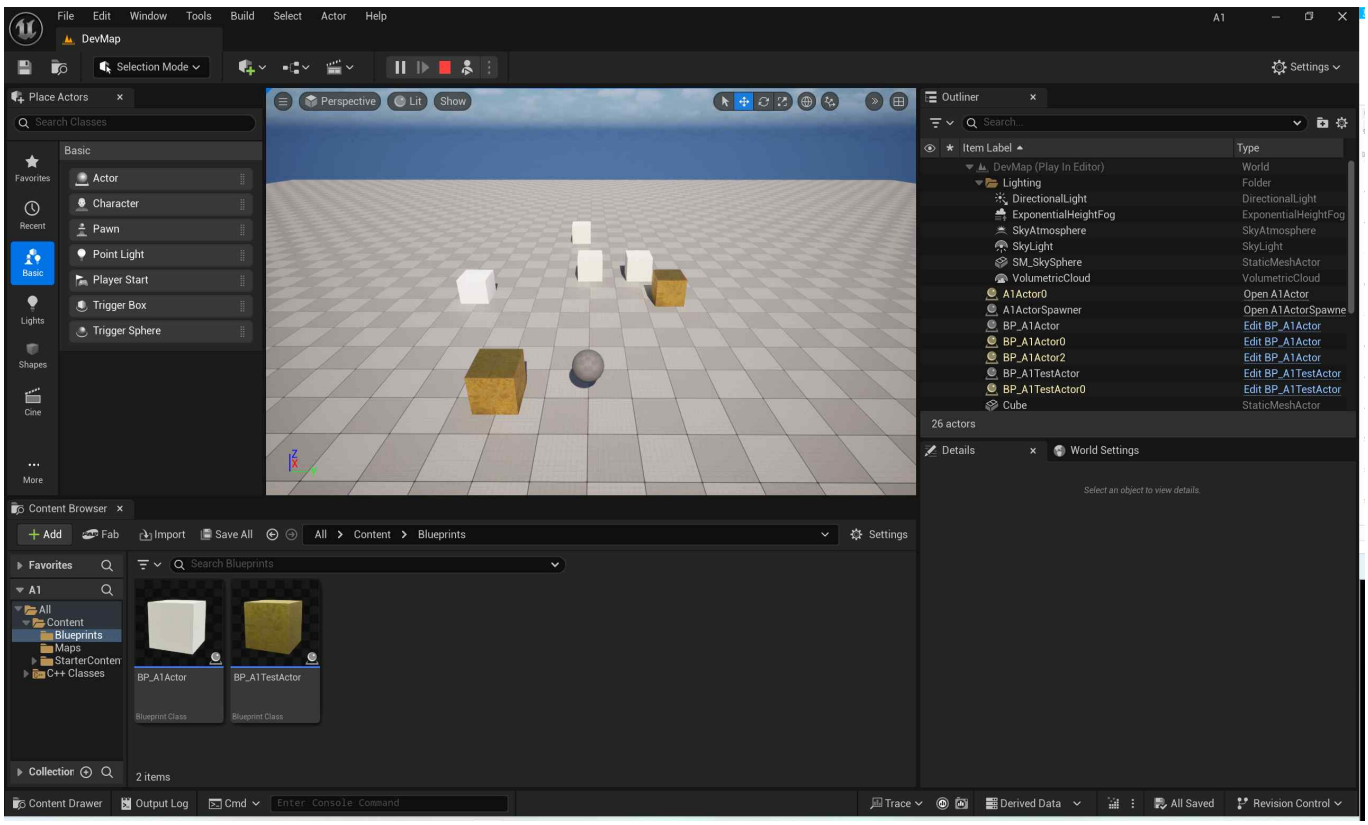
A1Actor.cpp

```
// Fill out your copyright notice in the Description page of Project Settings.



#include "A1Actor.h"
#include "A1Object.h"
#include "Components/StaticMeshComponent.h"

// Sets default values
AA1Actor::AA1Actor()
{
	// Set this actor to call Tick() every frame.  You can turn this off to improve performance if you don't need it.
	PrimaryActorTick.bCanEverTick = true;

	Body = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Body"));
	SetRootComponent(Body);

	static ConstructorHelpers::FObjectFinder<UStaticMesh> BodyMeshRef(TEXT("/Script/Engine.StaticMesh'/Game/StarterContent/Shapes/Shape_Cube.Shape_Cube'"));
	if (BodyMeshRef.Succeeded())
	{
		Body->SetStaticMesh(BodyMeshRef.Object);
	}
}

// Called when the game starts or when spawned
void AA1Actor::BeginPlay()
{
	Super::BeginPlay();
}

// Called every frame
void AA1Actor::Tick(float DeltaTime)
{
	Super::Tick(DeltaTime);

	FVector Location = GetActorLocation();
	FVector NewLocation = Location + FVector::ForwardVector * MovementSpeed * DeltaTime;
	SetActorLocation(NewLocation);
}
```

- 빌드 후 언리얼 에디터를 실행해 보자.



- 소환된 BP_A1Actor가 앞으로 이동하는 것을 확인할 수 있다.
- F8을 눌러 디버그 카메라로 살펴보면 더 잘 보인다.



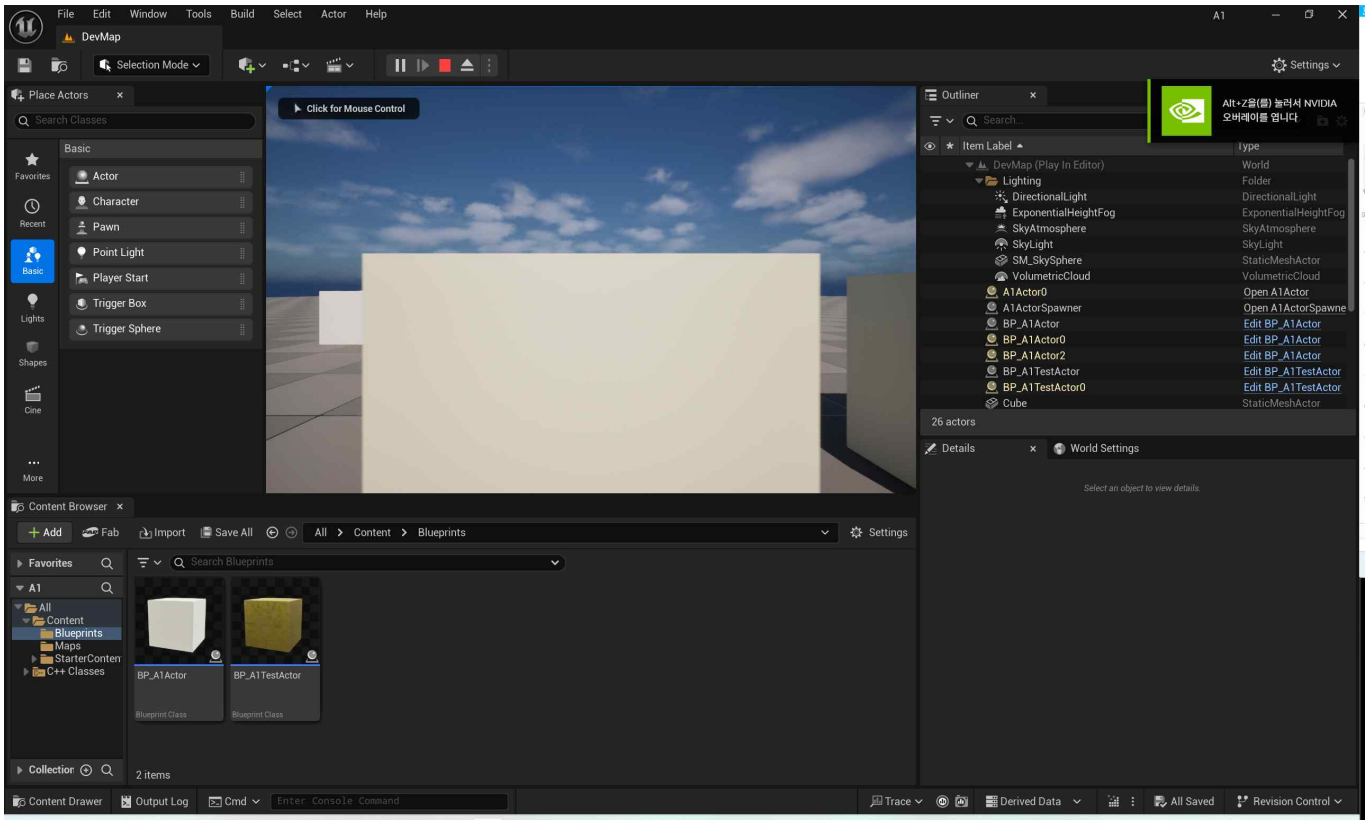
② AddActorWorldOffset

- 다음으로 AddActorWorldOffset 함수도 살펴보자.
- 현재 액터의 월드 좌표에 벡터를 더해주는 연산을 한다.

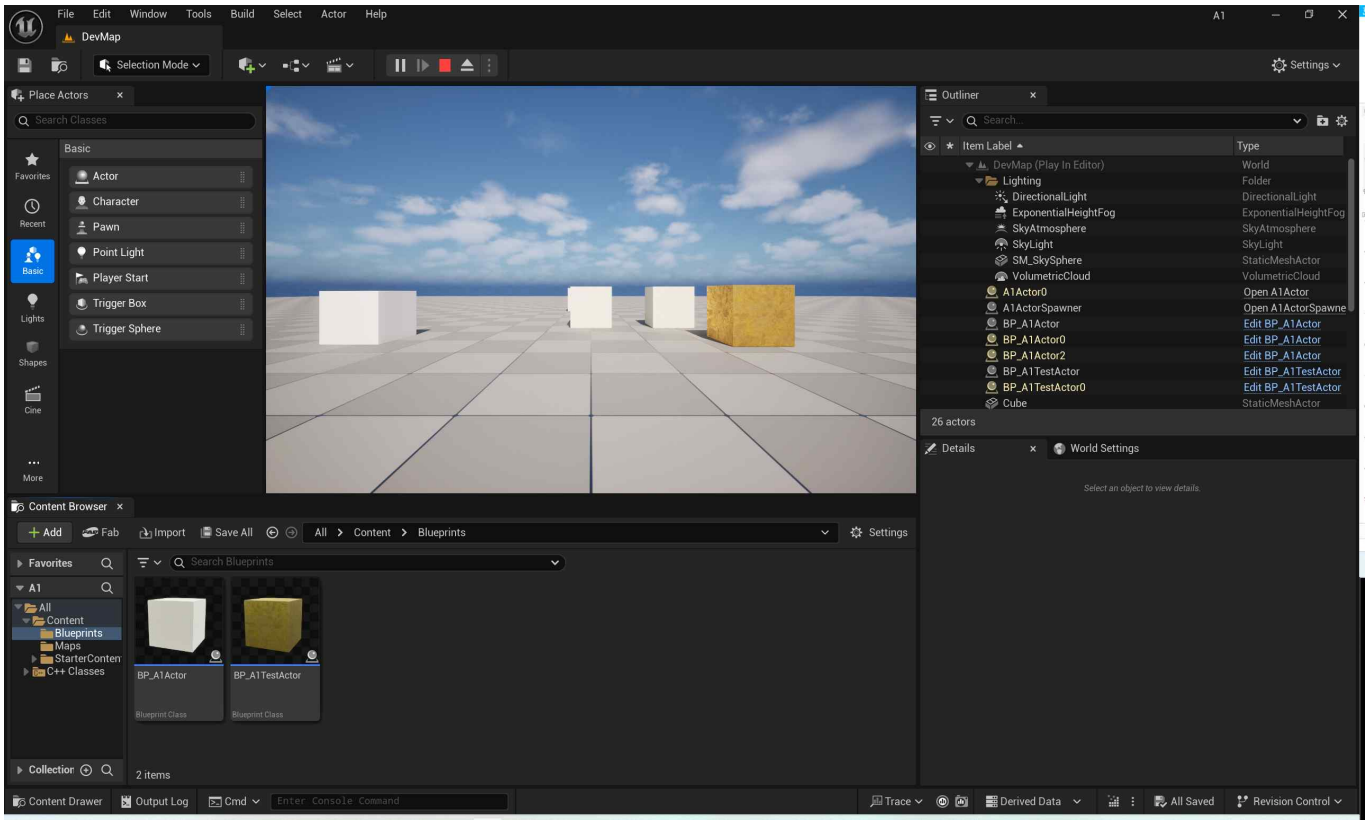
A1Actor.cpp

```
// Fill out your copyright notice in the Description page of Project Settings.



#include "A1Actor.h"
#include "A1Object.h"
#include "Components/StaticMeshComponent.h"

// Sets default values
AA1Actor::AA1Actor()
{
	// Set this actor to call Tick() every frame.  You can turn this off to improve performance if you don't need it.
	PrimaryActorTick.bCanEverTick = true;

	Body = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Body"));
	SetRootComponent(Body);

	static ConstructorHelpers::FObjectFinder<UStaticMesh> BodyMeshRef(TEXT("/Script/Engine.StaticMesh'/Game/StarterContent/Shapes/Shape_Cube.Shape_Cube'"));
	if (BodyMeshRef.Succeeded())
	{
		Body->SetStaticMesh(BodyMeshRef.Object);
	}
}

// Called when the game starts or when spawned
void AA1Actor::BeginPlay()
{
	Super::BeginPlay();
}

// Called every frame
void AA1Actor::Tick(float DeltaTime)
{
	Super::Tick(DeltaTime);

	//FVector Location = GetActorLocation();
	//FVector NewLocation = Location + FVector::ForwardVector * MovementSpeed * DeltaTime;
	//SetActorLocation(NewLocation);

	AddActorWorldOffset(FVector::ForwardVector * MovementSpeed * DeltaTime);
}
```

- 빌드 후 언리얼 에디터를 실행해 보자.



- 게임을 실행해 보자.
- 소환된 BP_A1Actor가 앞으로 이동하는 것을 확인할 수 있다.



③ Target Actor

- Target Actor로 이동하는 기능을 구현해보자.

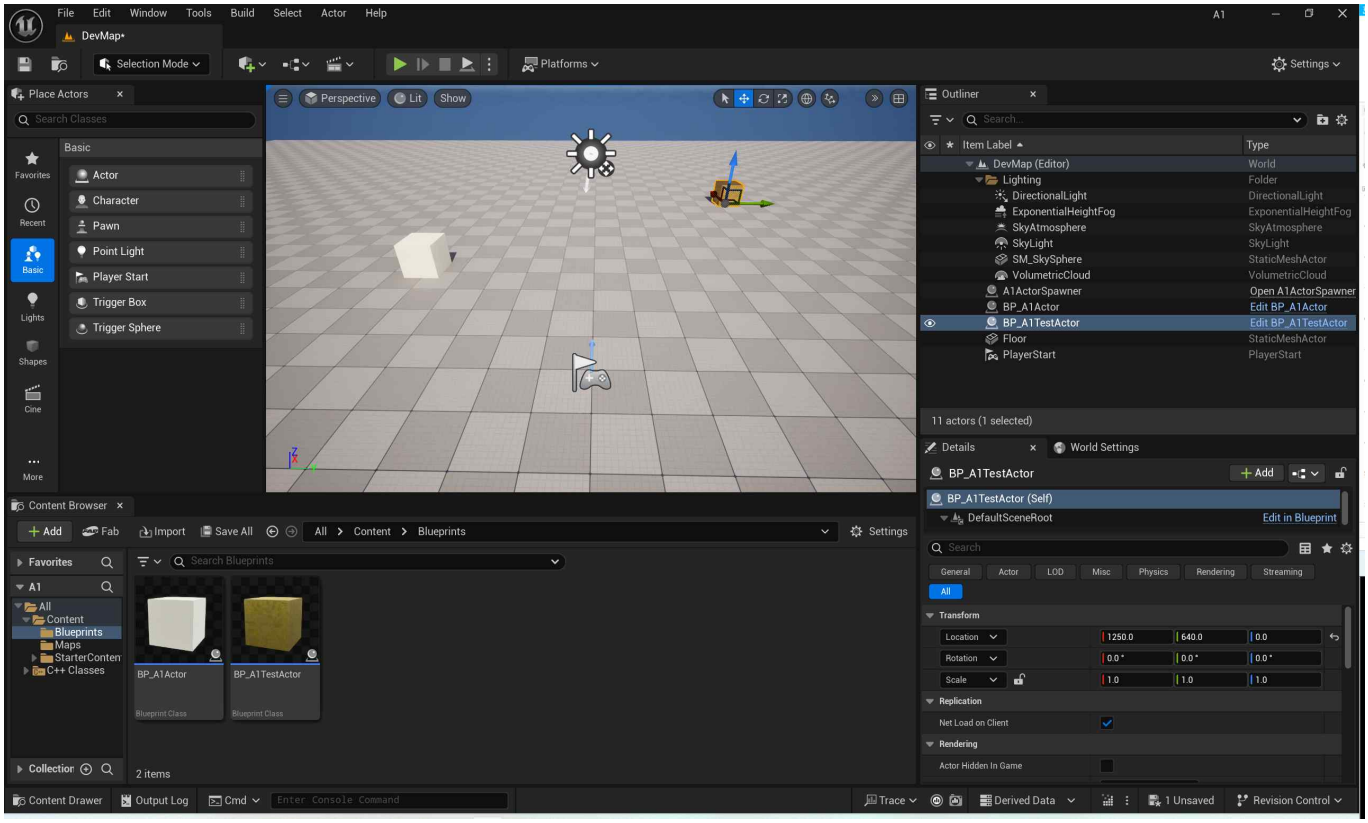
A1Actor.h

```
// Fill out your copyright notice in the Description page of Project Settings.

#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "A1Actor.generated.h"

class UA1Object;

UCLASS()
class A1_API AA1Actor : public AActor
{
	GENERATED_BODY()

public:
	// Sets default values for this actor's properties
	AA1Actor();

protected:
	// Called when the game starts or when spawned
	virtual void BeginPlay() override;

public:
	// Called every frame
	virtual void Tick(float DeltaTime) override;

protected:
	UPROPERTY(VisibleAnywhere, BlueprintReadWrite)
	TObjectPtr<class UStaticMeshComponent> Body;

	UPROPERTY(EditAnywhere, BlueprintReadWrite)
	float MovementSpeed = 50.0f;

	UPROPERTY(EditAnywhere, Category = Target)
	TObjectPtr<AActor> Target;
};
```

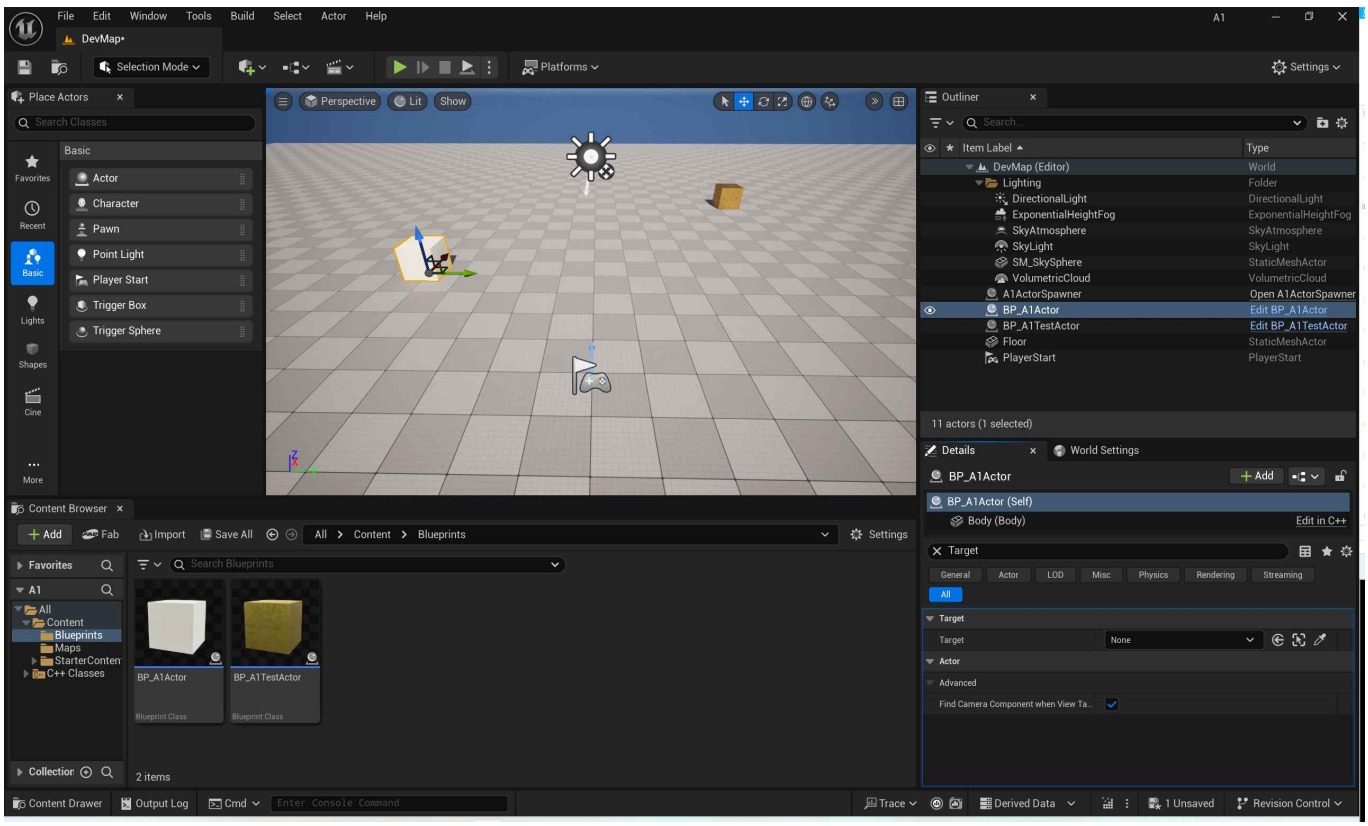
A1Actor.cpp

```
#include "A1Actor.h"
#include "A1Object.h"
#include "Components/StaticMeshComponent.h"

AA1Actor::AA1Actor()
{
	PrimaryActorTick.bCanEverTick = true;

	Body = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Body"));
	SetRootComponent(Body);

	static ConstructorHelpers::FObjectFinder<UStaticMesh>
BodyMeshRef(TEXT("/Script/Engine.StaticMesh'/Game/StarterContent/Shapes/Shape_Cube.Shape_Cube'"));
	if (BodyMeshRef.Succeeded())
	{
		Body->SetStaticMesh(BodyMeshRef.Object);
	}
}

void AA1Actor::BeginPlay()
{
	Super::BeginPlay();
}

void AA1Actor::Tick(float DeltaTime)
{
	Super::Tick(DeltaTime);

	//FVector Location = GetActorLocation();
	//FVector NewLocation = Location + FVector::ForwardVector * MovementSpeed * DeltaTime;
	//SetActorLocation(NewLocation);

	//AddActorWorldOffset(FVector::ForwardVector * MovementSpeed * DeltaTime);

	if (Target != nullptr)
	{
		FVector Location = GetActorLocation();
		FVector Direction = Target->GetActorLocation() - GetActorLocation();

		AddActorWorldOffset(Direction.GetSafeNormal() * DeltaTime * MovementSpeed);
	}
}
```

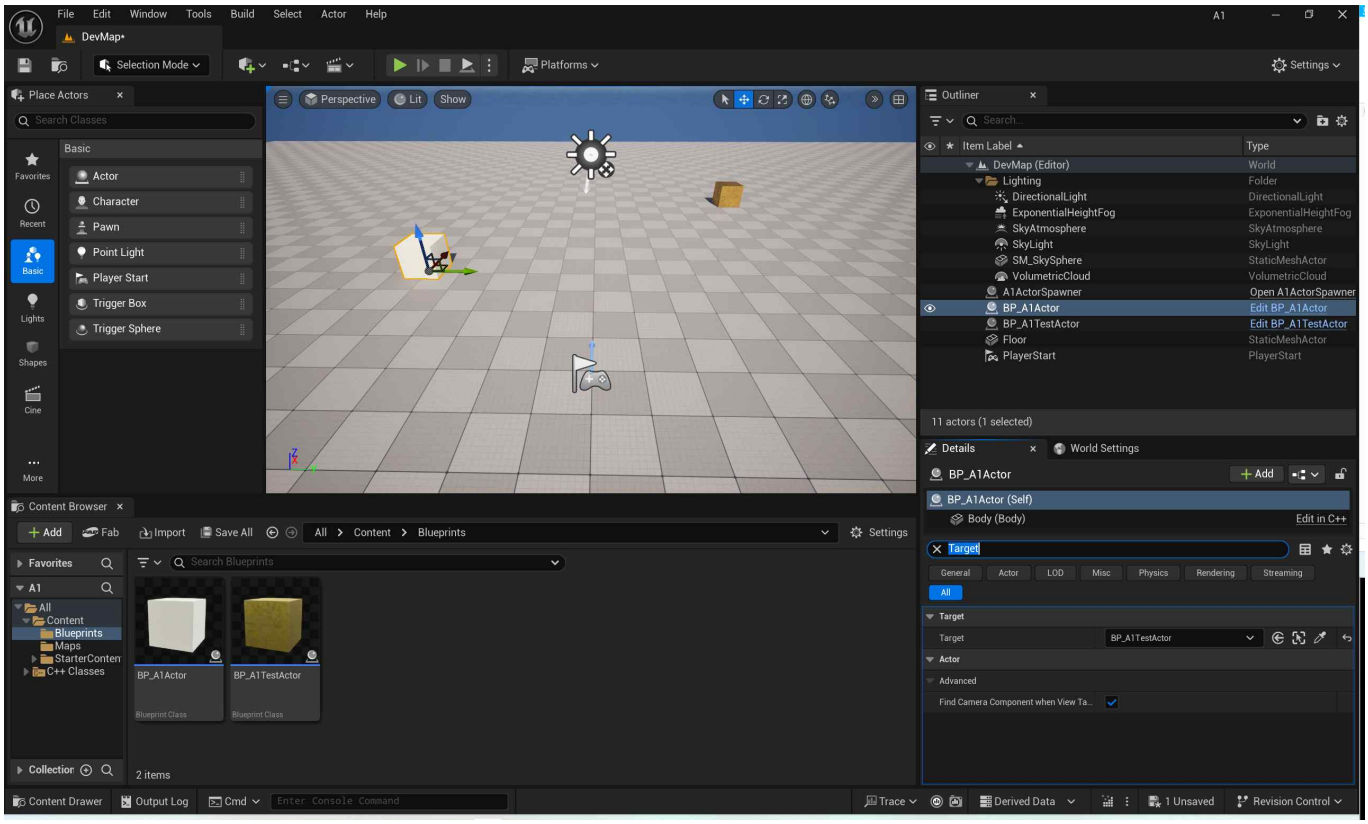
- 빌드 후 언리얼 엔진을 켜자.
- 레벨에 배치된 BP_A1Actor과 BP_A1TestActor를 적절하게 배치해보자.



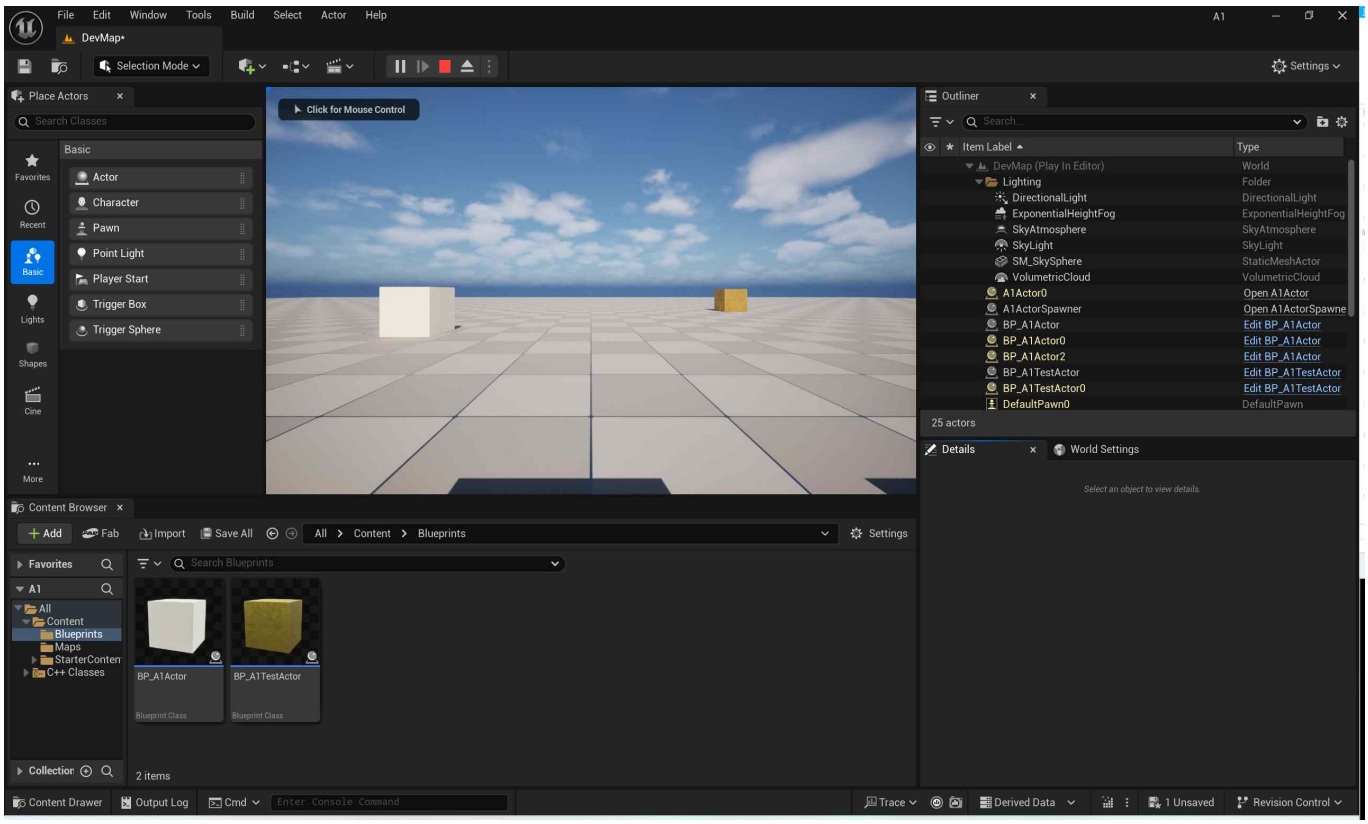
- 레벨에 배치된 BP_A1Actor를 선택하자.
- 디테일 창에서 Target검색해 보자.



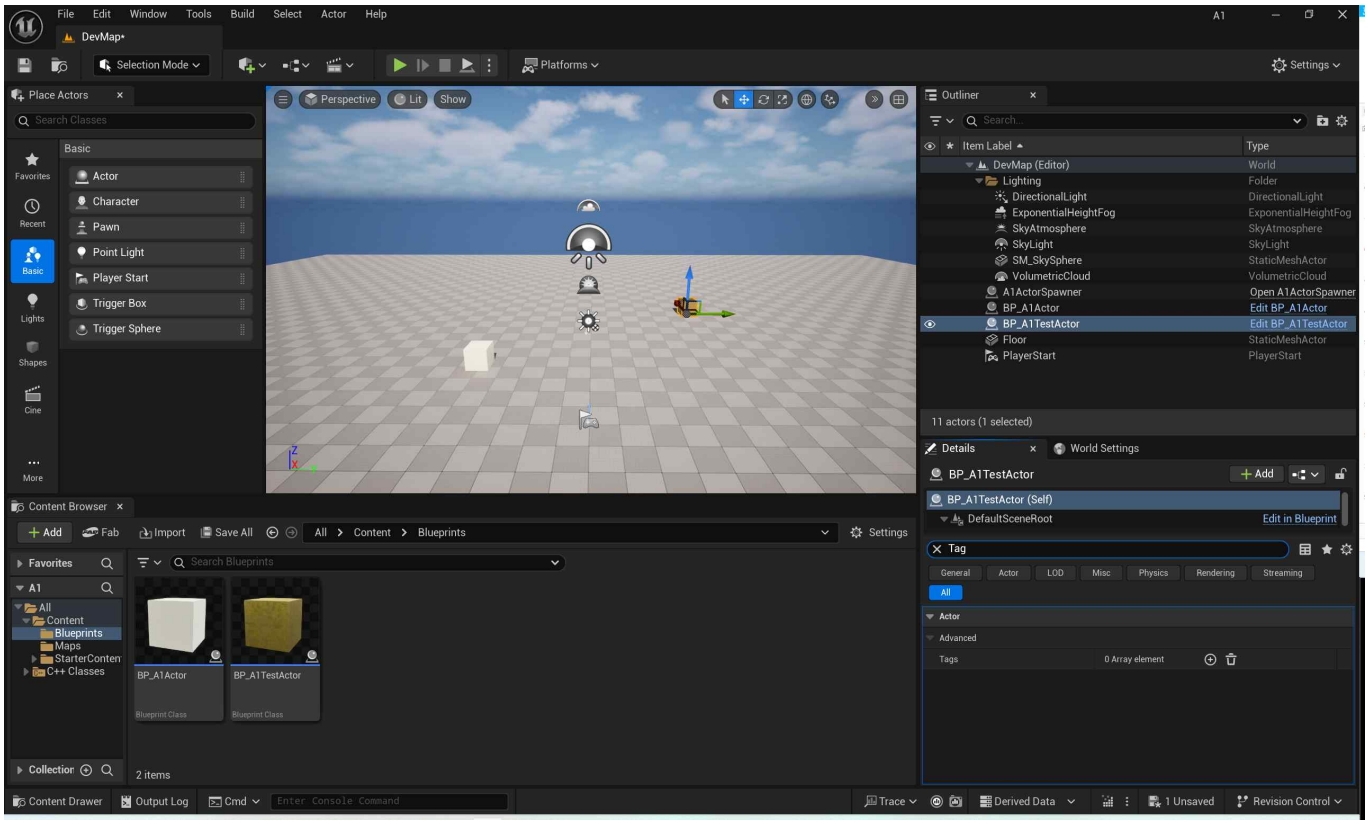
- BP_A1Actor에 Target을 BP_TestActor로 설정하자.



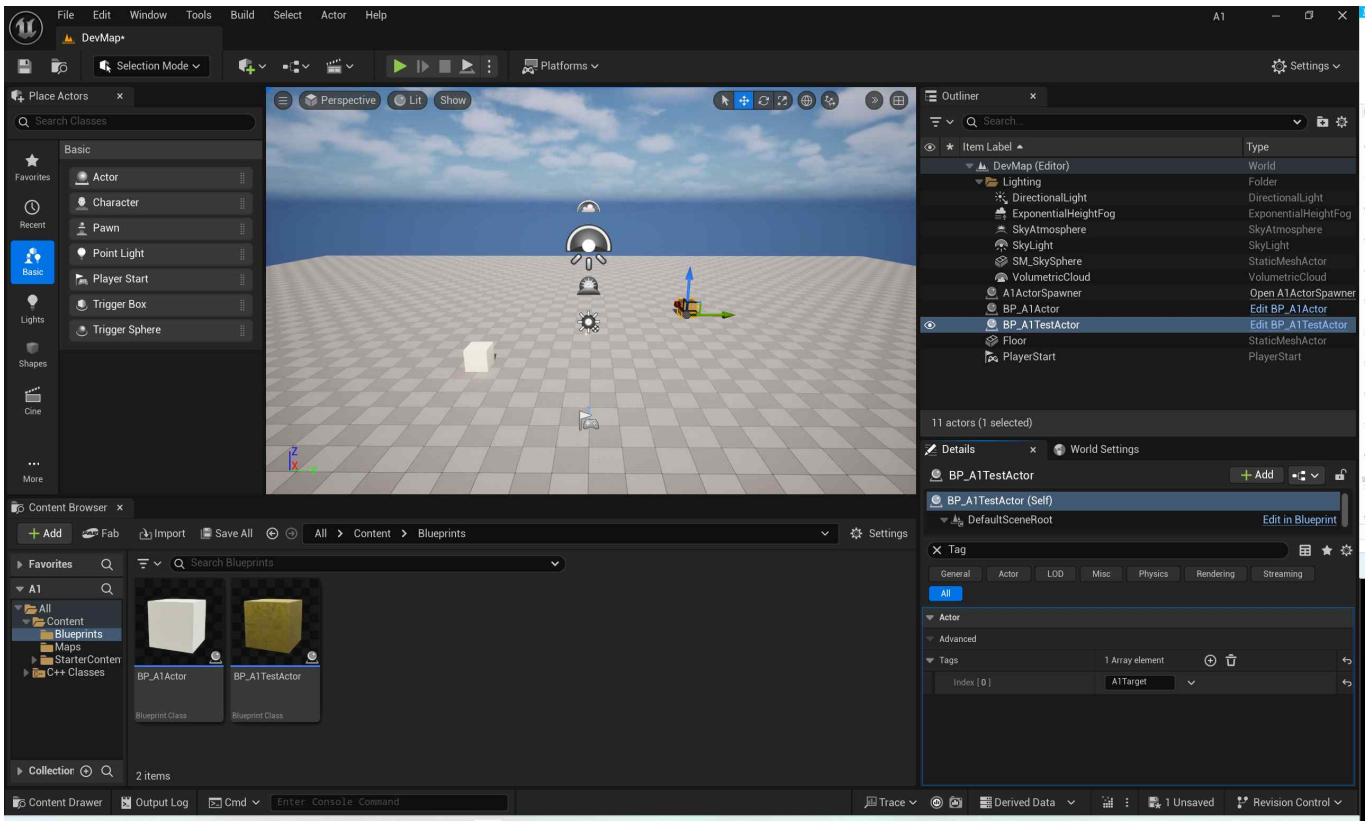
- 게임을 실행하여 BP_A1Actor가 BP_TestActor를 향해 이동하는지 확인하자.



- 현재는 배치된 액터에 직접적으로 타겟을 설정하여 이동시켰다.
- 이번에는 런타임 시간에 타겟을 찾아서 이동하는 코드를 구현해보자.
- 이를 위해서 레벨에 배치된 Actor를 찾는 함수들을 활용하면 된다.
- 이번에는 태그를 활용해서 찾아보자. BP_TestActor를 선택하고 Tag를 검색하자.



- A1Target 이라고 태그를 추가하자.

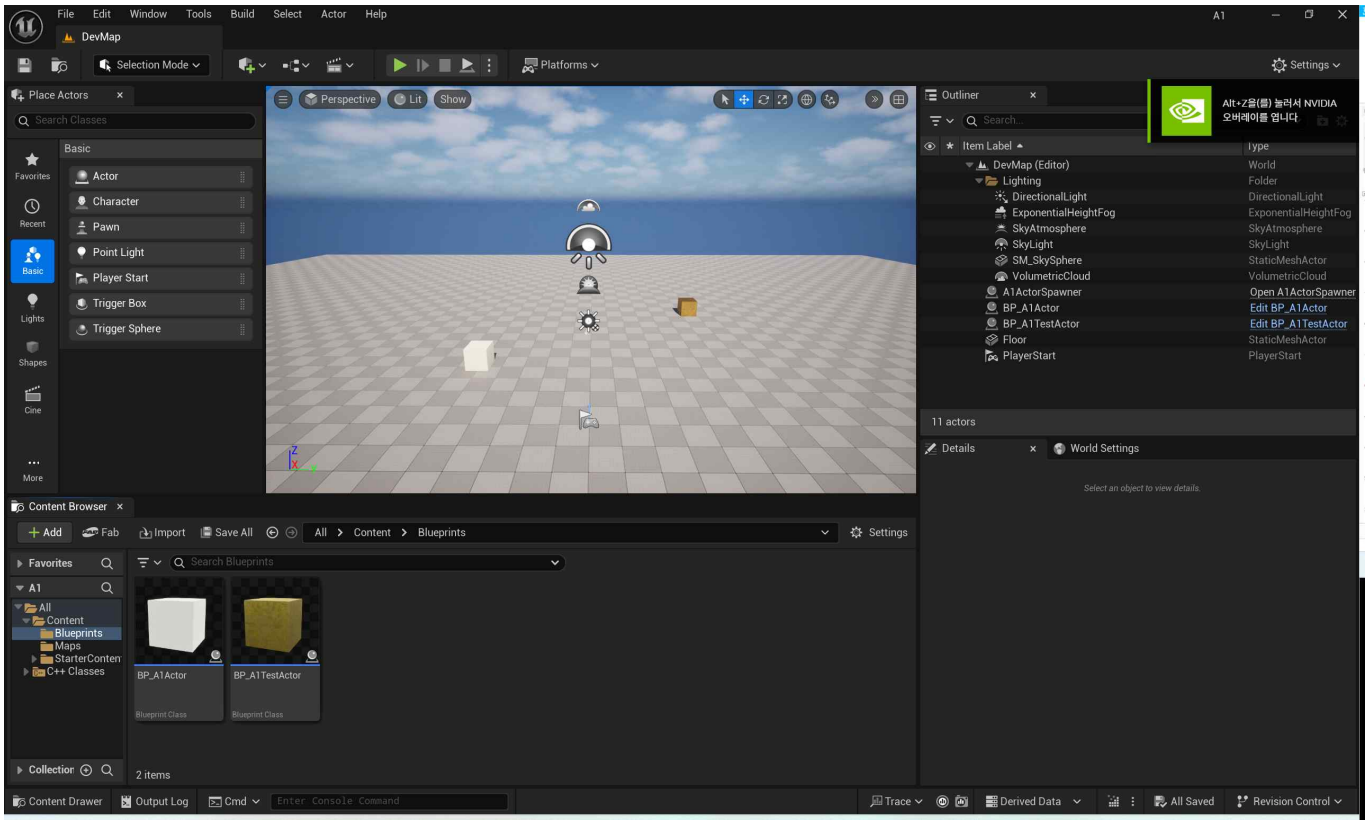


- 이제 게임이 시작될 때, BeginPlay가 될 때, Target을 설정해보자.

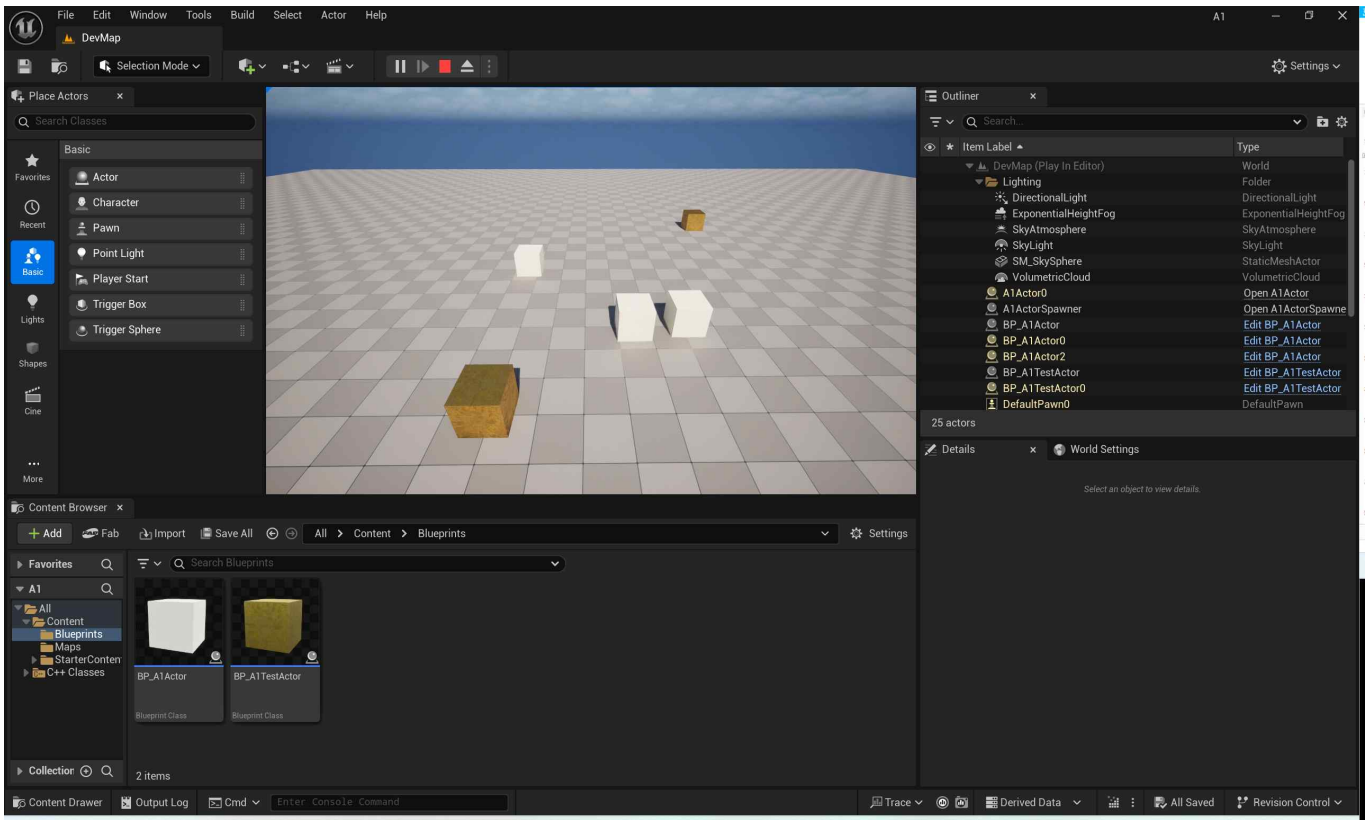
A1Actor.cpp

```
#include "A1Actor.h"
#include "A1Object.h"
#include "Components/StaticMeshComponent.h"
#include "Kismet/GameplayStatics.h"

AA1Actor::AA1Actor()
{
	PrimaryActorTick.bCanEverTick = true;

	Body = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Body"));
	SetRootComponent(Body);

	static ConstructorHelpers::FObjectFinder<UStaticMesh>
BodyMeshRef(TEXT("/Script/Engine.StaticMesh'/Game/StarterContent/Shapes/Shape_Cube.Shape_Cube'"));
	if (BodyMeshRef.Succeeded())
	{
		Body->SetStaticMesh(BodyMeshRef.Object);
	}
}

void AA1Actor::BeginPlay()
{
	Super::BeginPlay();

	TArray<AActor*> Actors;
	UGameplayStatics::GetAllActorsWithTag(GetWorld(), TEXT("A1Target"), OUT Actors);

	if (Actors.Num() > 0)
	{
		Target = Actors[0];
	}
}

void AA1Actor::Tick(float DeltaTime)
{
	Super::Tick(DeltaTime);

	if (Target != nullptr)
	{
		FVector Location = GetActorLocation();
		FVector Direction = Target->GetActorLocation() - GetActorLocation();

		AddActorWorldOffset(Direction.GetSafeNormal() * DeltaTime * MovementSpeed);
	}
}
```

- 빌드 후 언리얼 에디터를 실행하자.



- 게임을 실행해서 A1ActorSpawner가 소환한 BP_A1Actor가 Target을 향해 이동하는지 확인하자.



3) 액터의 회전

- SetActorRoation 함수를 적용해보자.
- 회전의 FRotator(Pitch, Yaw, Roll) 순서인 것을 기억하자.

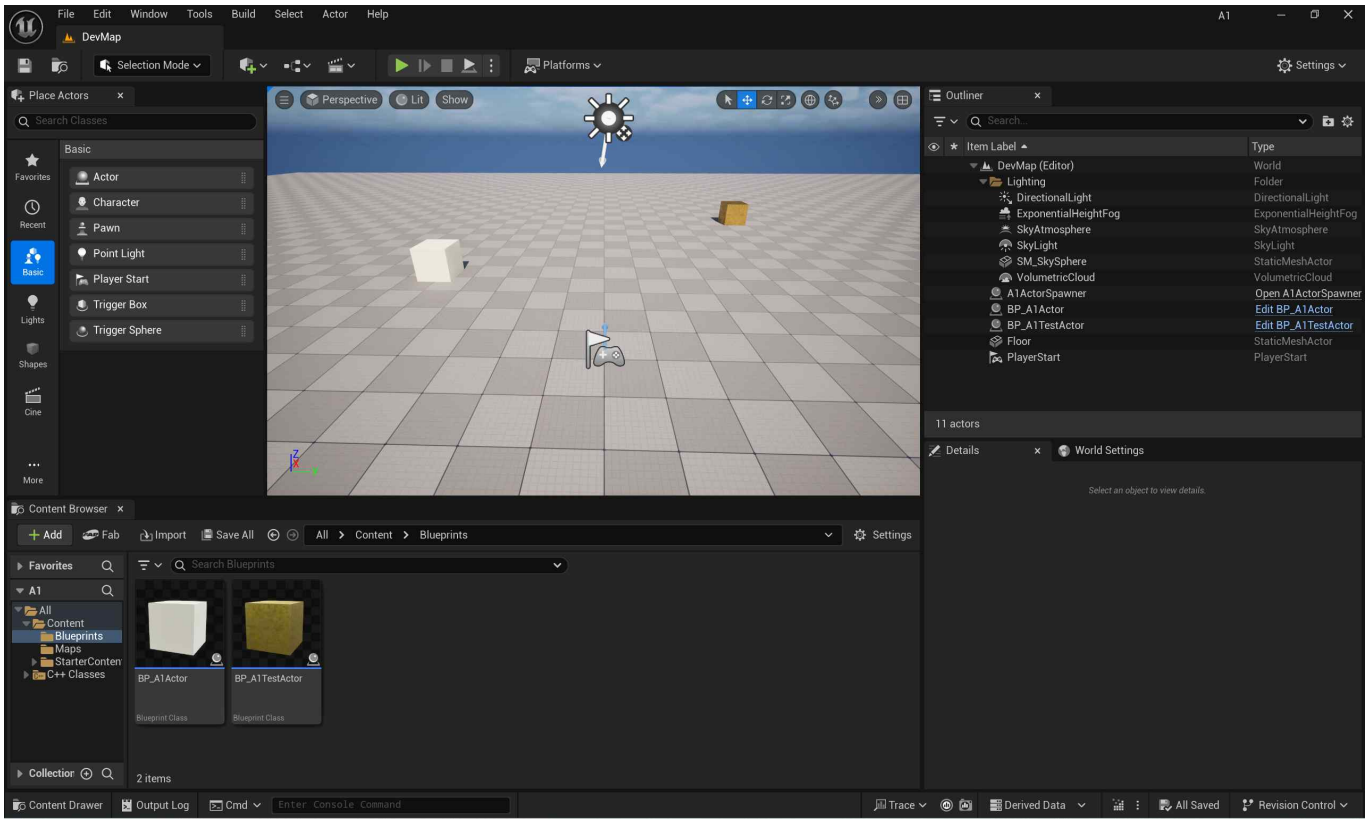
A1Actor.h

```
// Fill out your copyright notice in the Description page of Project Settings.

#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "A1Actor.generated.h"

class UA1Object;

UCLASS()
class A1_API AA1Actor : public AActor
{
	GENERATED_BODY()

public:
	// Sets default values for this actor's properties
	AA1Actor();

protected:
	// Called when the game starts or when spawned
	virtual void BeginPlay() override;

public:
	// Called every frame
	virtual void Tick(float DeltaTime) override;

protected:
	UPROPERTY(VisibleAnywhere, BlueprintReadWrite)
	TObjectPtr<class UStaticMeshComponent> Body;

	UPROPERTY(EditAnywhere, BlueprintReadWrite)
	float MovementSpeed = 50.0f;

	UPROPERTY(EditAnywhere, Category = Target)
	TObjectPtr<AActor> Target;

	UPROPERTY(EditAnywhere, BlueprintReadWrite)
	float RotationRate = 45.f;
};
```

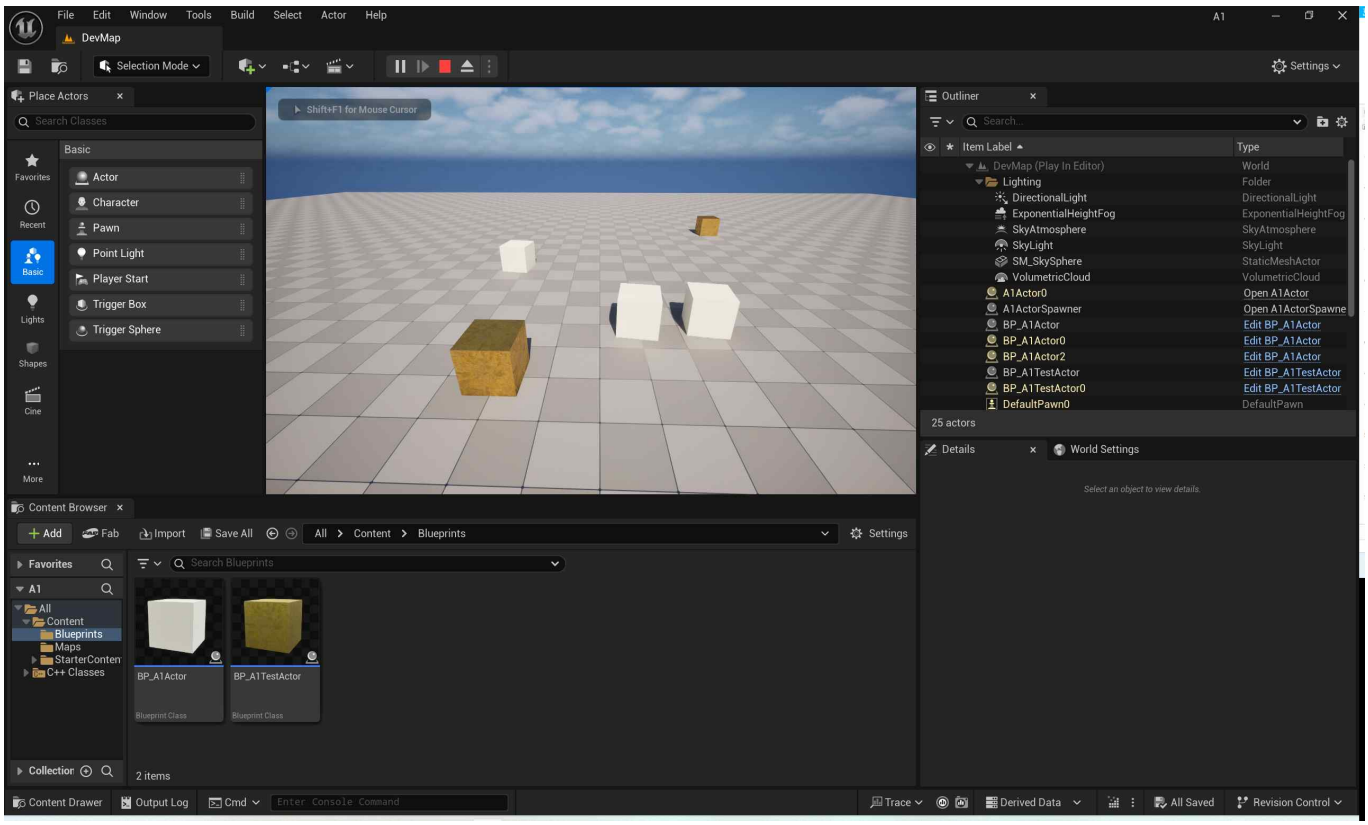
A1Actor.cpp

```
#include "A1Actor.h"
#include "A1Object.h"
#include "Components/StaticMeshComponent.h"
#include "Kismet/GameplayStatics.h"

AA1Actor::AA1Actor()
{
	PrimaryActorTick.bCanEverTick = true;

	...
}

void AA1Actor::BeginPlay()
{
	Super::BeginPlay();

	...
}

void AA1Actor::Tick(float DeltaTime)
{
	Super::Tick(DeltaTime);

	//FVector Location = GetActorLocation();
	//FVector NewLocation = Location + FVector::ForwardVector * MovementSpeed * DeltaTime;
	//SetActorLocation(NewLocation);

	//AddActorWorldOffset(FVector::ForwardVector * MovementSpeed * DeltaTime);

	if (Target != nullptr)
	{
		FVector Location = GetActorLocation();
		FVector Direction = Target->GetActorLocation() - GetActorLocation();

		AddActorWorldOffset(Direction.GetSafeNormal() * DeltaTime * MovementSpeed);
	}

	FRotator Rotation = GetActorRotation();
	FRotator NewRotation = FRotator(Rotation.Pitch, Rotation.Yaw + RotationRate * DeltaTime, Rotation.Roll);
	SetActorRotation(NewRotation);
}
```

- 빌드 후, 언리얼 에디터를 실행하자.



- 게임을 실행해서 회전되는지 확인하자.

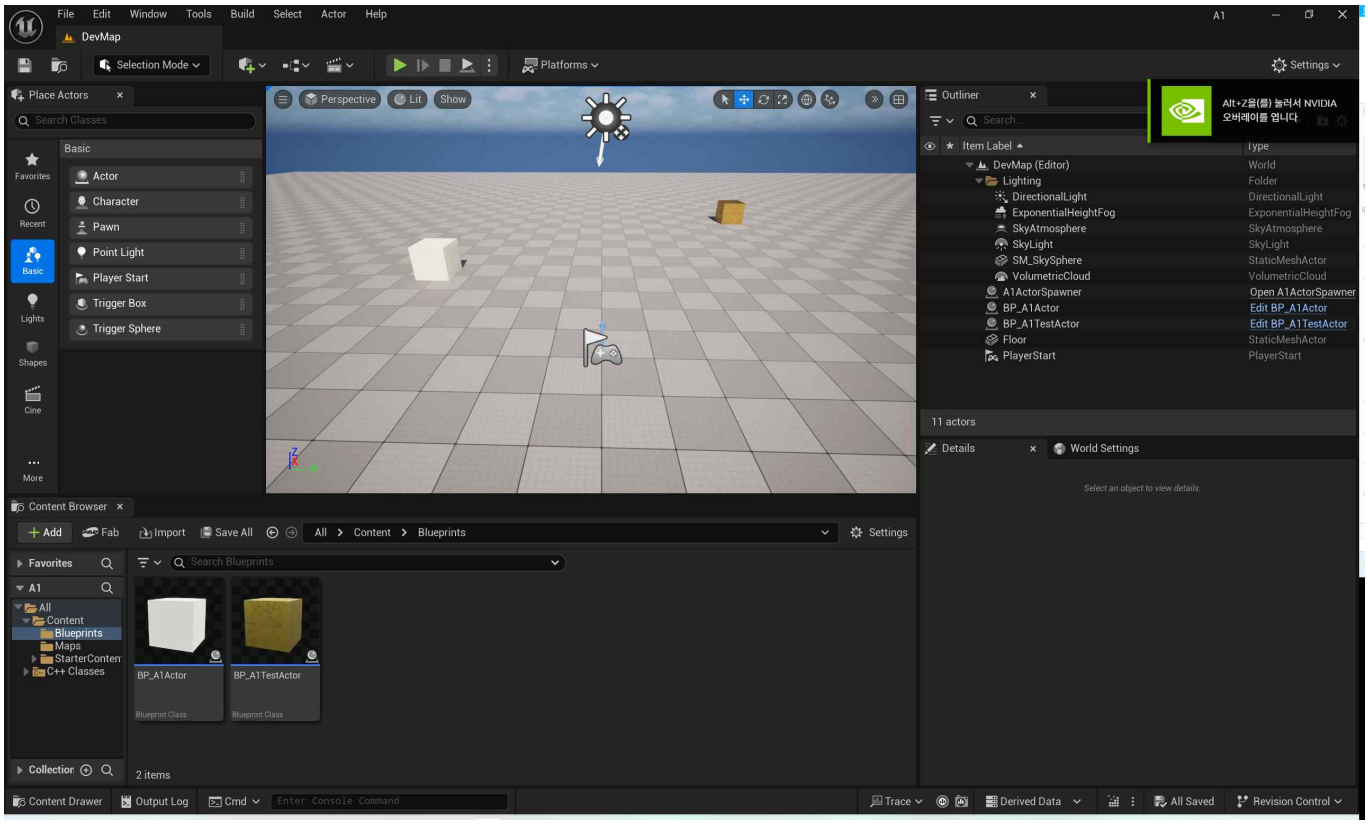


- AddActorWorldRotation 함수를 적용해보자.

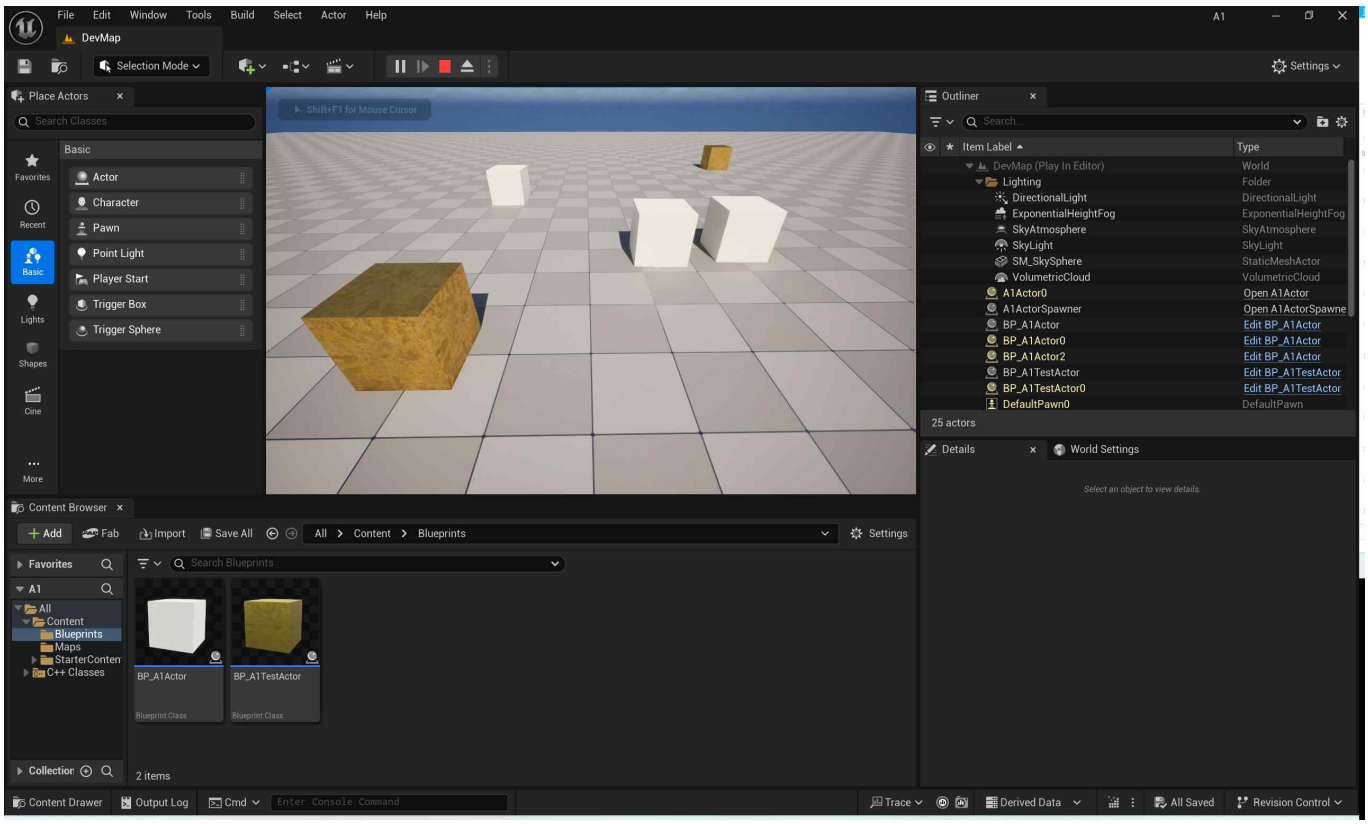
A1Actor.cpp

```
#include "A1Actor.h"
#include "A1Object.h"
#include "Components/StaticMeshComponent.h"
#include "Kismet/GameplayStatics.h"

AA1Actor::AA1Actor()
{
	PrimaryActorTick.bCanEverTick = true;

	...
}

void AA1Actor::BeginPlay()
{
	Super::BeginPlay();

	...
}

void AA1Actor::Tick(float DeltaTime)
{
	Super::Tick(DeltaTime);

	//FVector Location = GetActorLocation();
	//FVector NewLocation = Location + FVector::ForwardVector * MovementSpeed * DeltaTime;
	//SetActorLocation(NewLocation);

	//AddActorWorldOffset(FVector::ForwardVector * MovementSpeed * DeltaTime);

	if (Target != nullptr)
	{
		FVector Location = GetActorLocation();
		FVector Direction = Target->GetActorLocation() - GetActorLocation();

		AddActorWorldOffset(Direction.GetSafeNormal() * DeltaTime * MovementSpeed);
	}

	//FRotator Rotation = GetActorRotation();
	//FRotator NewRotation = FRotator(Rotation.Pitch, Rotation.Yaw + RotationRate * DeltaTime,
Rotation.Roll);
	//SetActorRotation(NewRotation);

	AddActorWorldRotation(FRotator(0.f, RotationRate * DeltaTime, 0.f));
}
```

- 빌드 후, 언리얼 에디터를 실행하자.



- 게임을 실행해서 회전하는지 확인하자.



4) Component

- 액터에 붙일 수 있는 오브젝트 타입인 컴포넌트를 살펴보자.

① Actor Components

```
// Engine\Source\Runtime\Engine\Classes\Components\ActorComponent.h
class ENGINE_API UActorComponent : public UObject, public IInterface_AssetUserData
```

- 모든 컴포넌트의 베이스 컴포넌트이다. Transform 이 없기 때문에 위치, 회전값이 없으며 렌더링 되지 않는다.
- 액터의 움직임, 인벤토리, 속성 관리 등등의 추상적인 동작 구현에 적절하다.

② Scene Components

```
// Engine\Source\Runtime\Engine\Classes\Components\SceneComponent.h
class ENGINE_API USceneComponent : public UActorComponent
```

- Actor Component 를 상속받은 컴포넌트. Transform 을 가지나 렌더링되지 않는다.
- 카메라, 스프링 팔, 물리적인 힘과 제약 조건, 오디오 등 렌더링이 필요없는 위치 기반 동작 구현에 적절하다.

③ Primitive Components

```
// Engine\Source\Runtime\Engine\Classes\Components\PrimitiveComponent.h
class ENGINE_API UPrimitiveComponent : public USceneComponent, public INavRelevantInterface
```

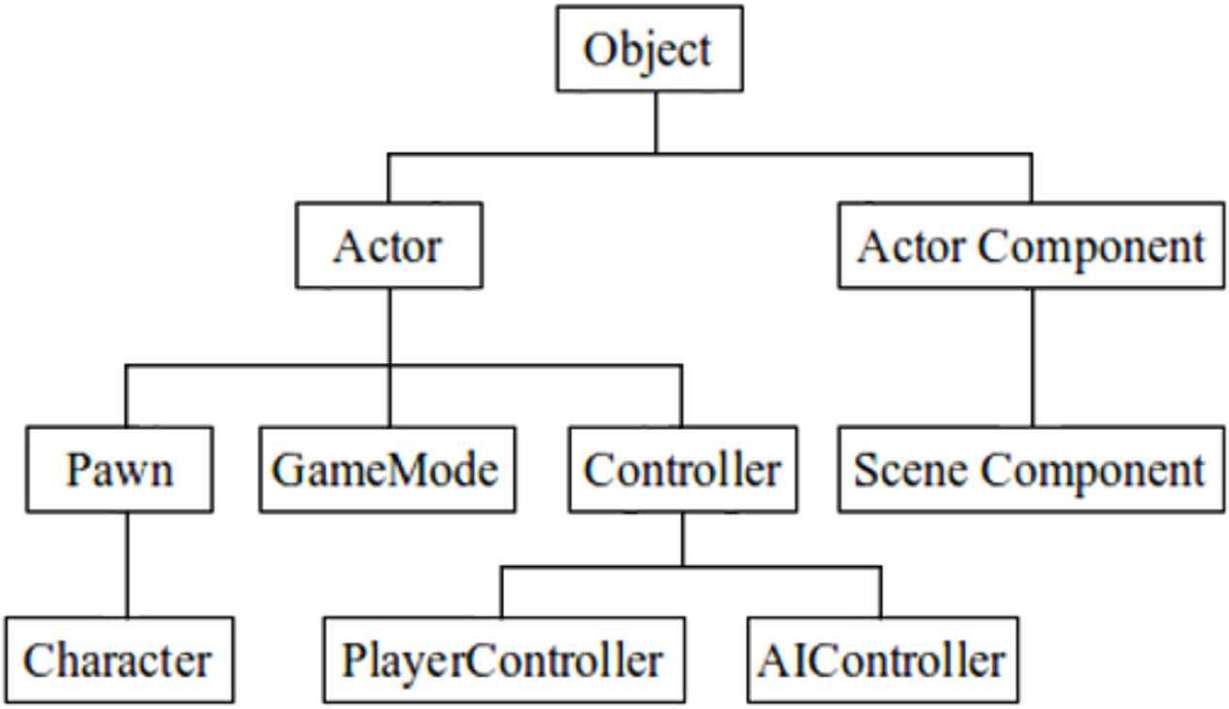
- 카메라, 스프링 팔, 물리적인 힘과 제약 조건, 오디오 등 렌더링이 필요없는 위치 기반 동작 구현에 적절하다.
- Scene Component 를 상속받은 컴포넌트. Transform 을 가지며 렌더링 된다.
- 시각적 요소의 렌더링, 충돌 영역 등의 기하학적 표현에 사용되는 베이스 클래스.
- 스태틱 메시, 스켈레탈 메시, 스프라이트나 빌보드, 파티클 시스템, 박스, 캡슐 등의 충돌 볼륨 등이 포함된다.
- material 은 멤버 변수로 갖지 않지만, 대신 비어있는 virtual get/set 함수가 존재한다. (GetMaterial / SetMaterial)

④ Mesh Components

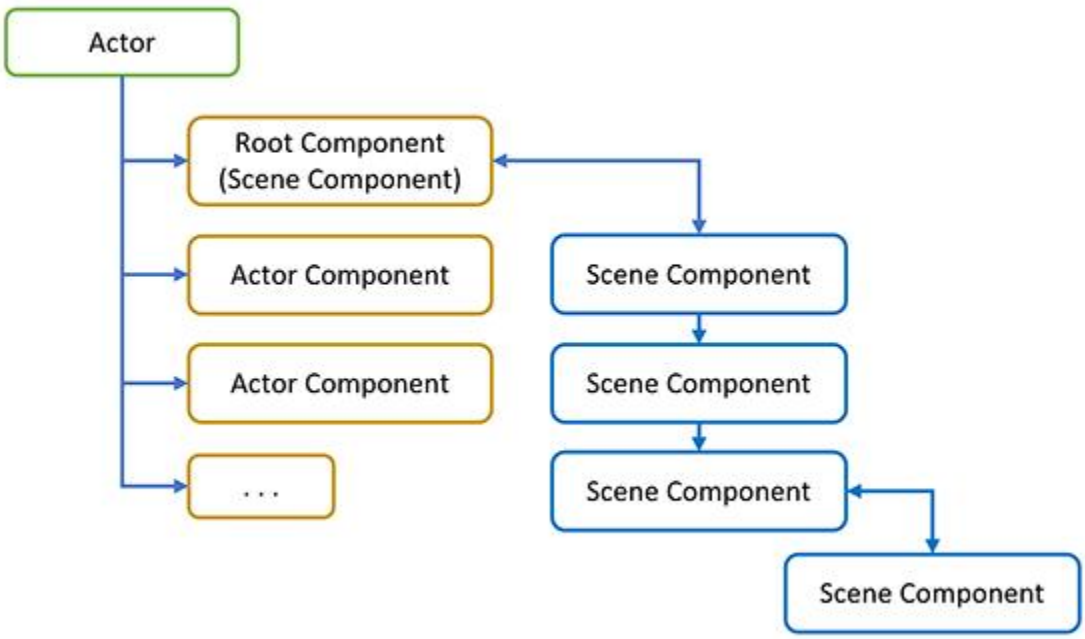
```
// Engine\Source\Runtime\Engine\Classes\Components\MeshComponent.h
class ENGINE_API UMeshComponent : public UPrimitiveComponent
```

- Primitive Component 를 상속받은 컴포넌트. Mesh 와 material 배열을 가진다.
- Mesh 를 렌더링하는 컴포넌트들의 베이스 클래스

※ Object Hierarchy

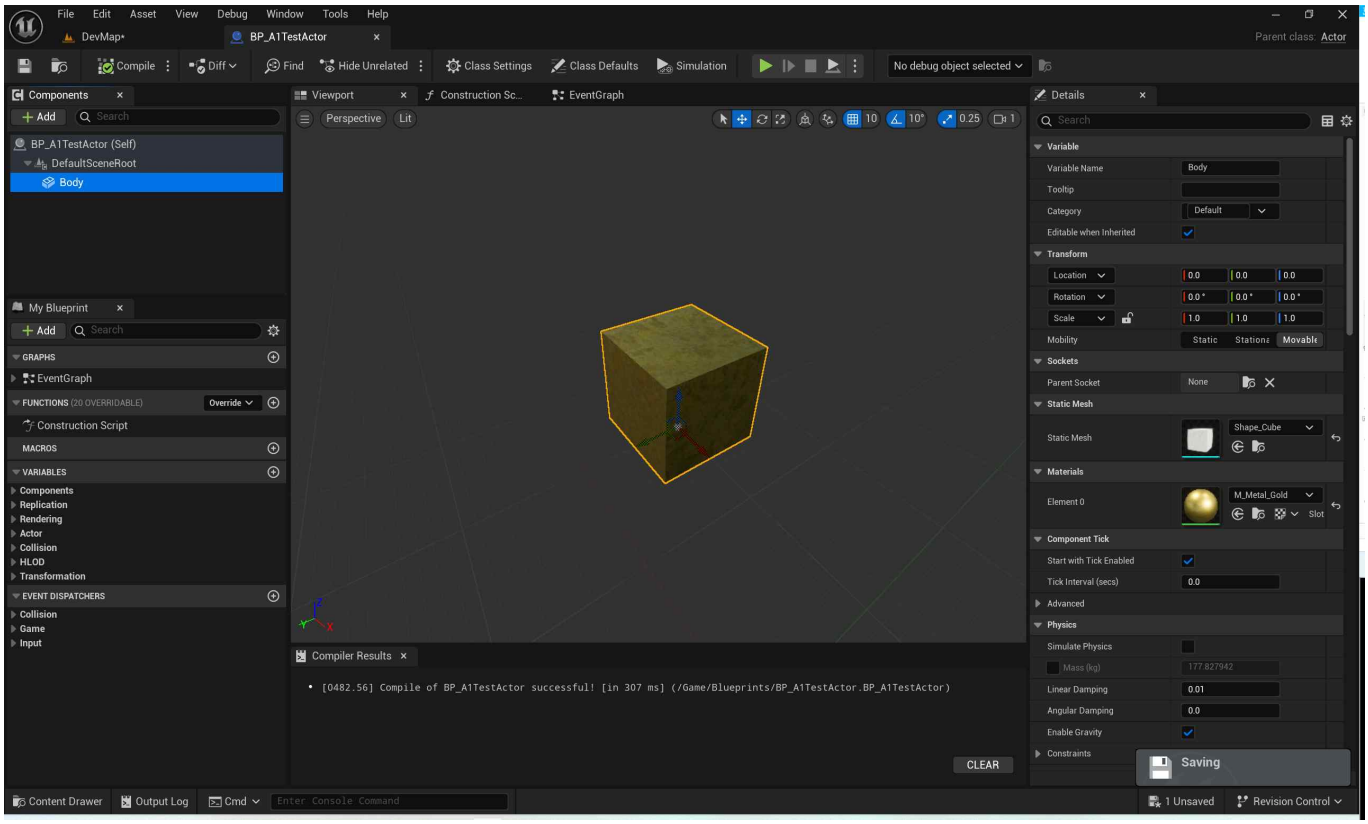


※ 런타임에 구성되는 클래스 구성 요소의 계층 구조

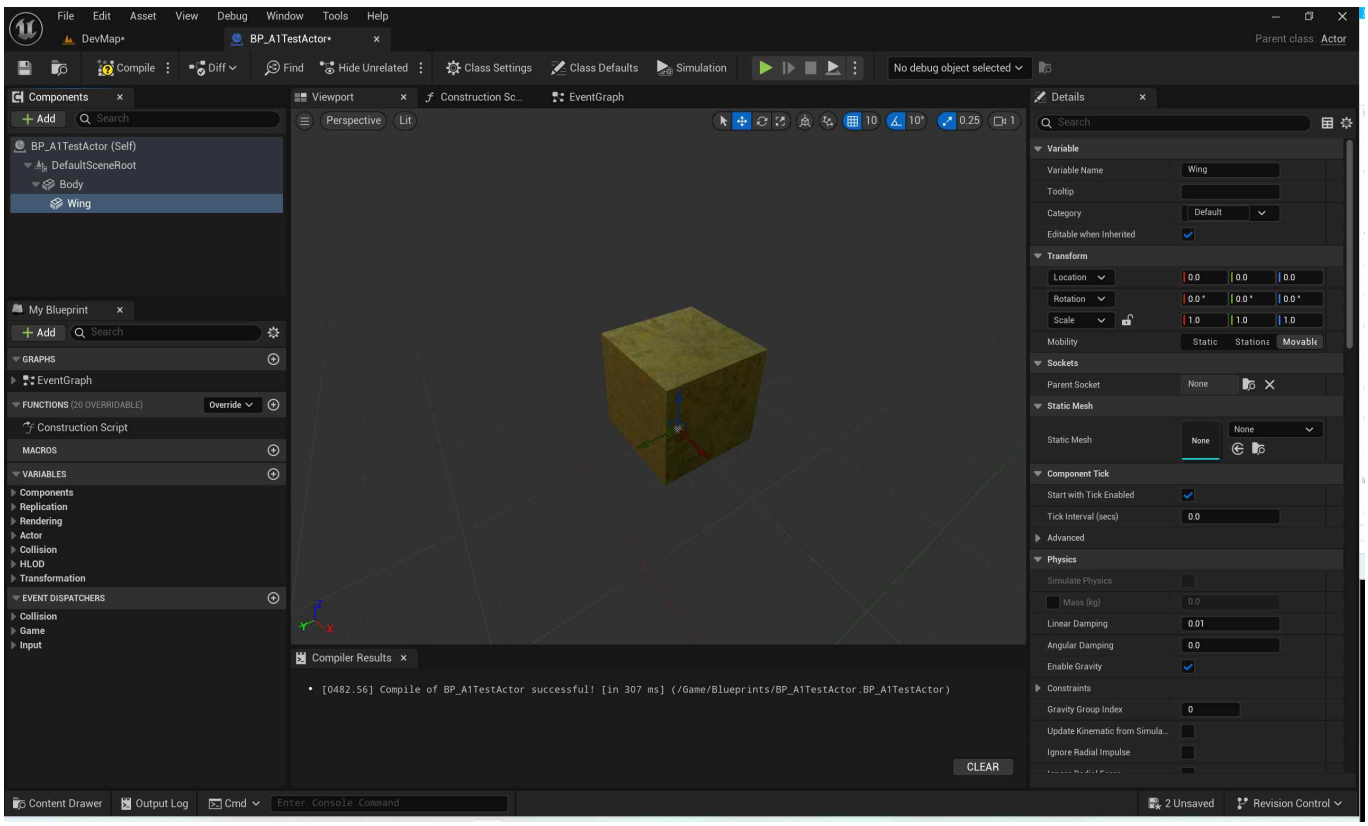


5) 계층구조

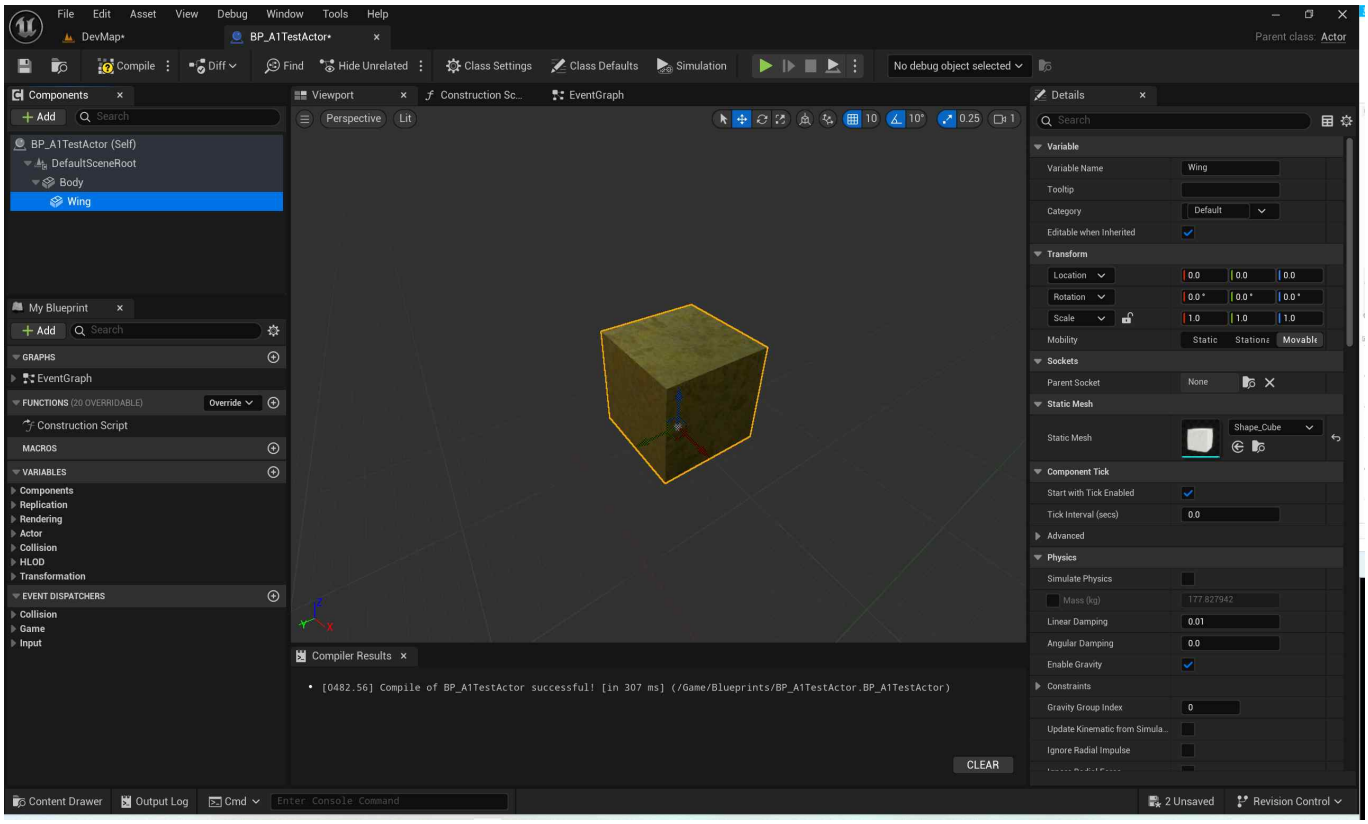
- 계층구조라는 것은 Transform을 가진 Components에서 성립된다.
- BP_A1TestActor를 열어서 Body를 선택하고 Static Mesh Component를 하나더 추가해보자.



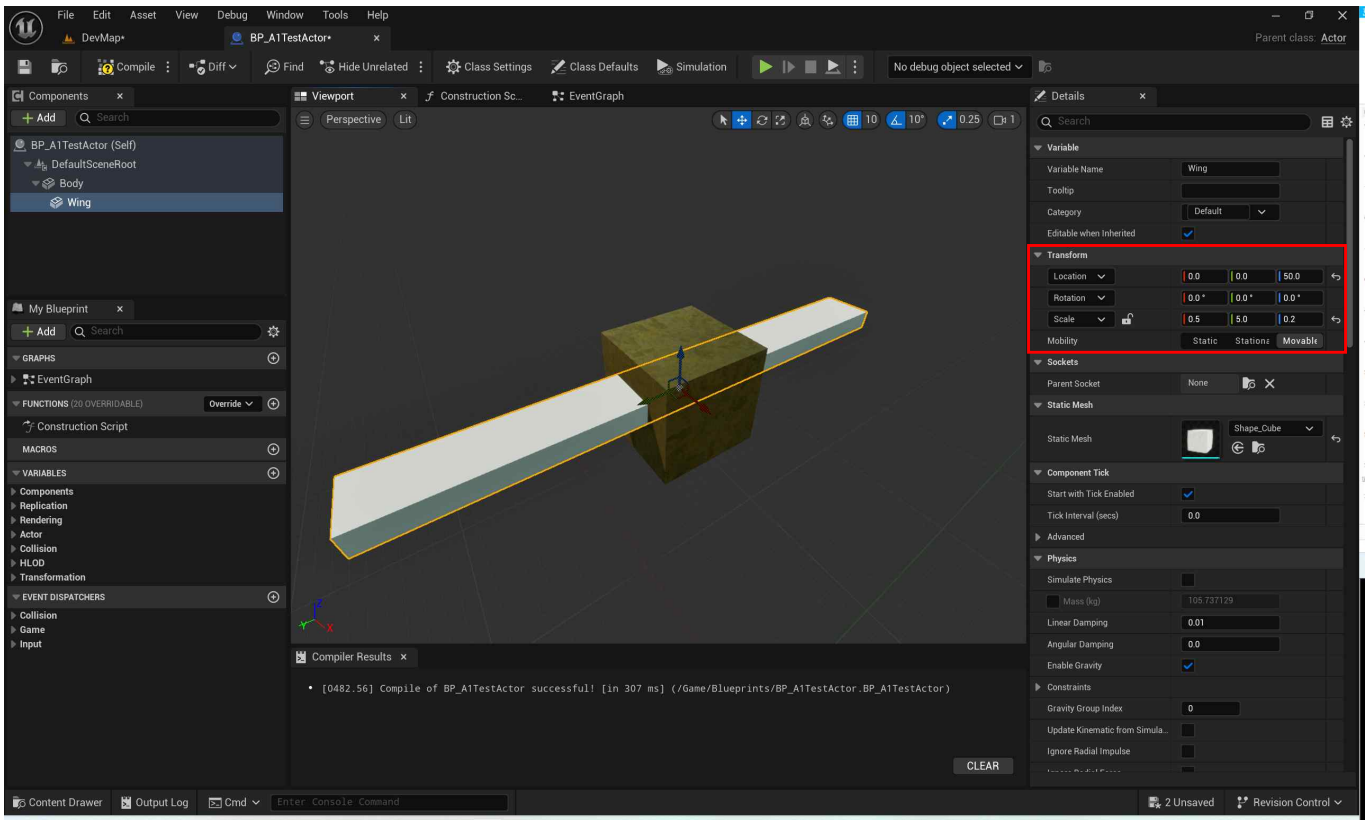
- 추가한 **Static Mesh Component**의 이름은 **Wing**으로 설정하자.



- 추가한 Wing의 Static Mesh를 Shape_Cube로 설정하자.



- Shape_Cube의 Location과 Scale을 아래와 같이 설정하자.



- 해당 컴포넌트 기능을 A1Actor에서 다시 구현해 보자.

A1Actor.h

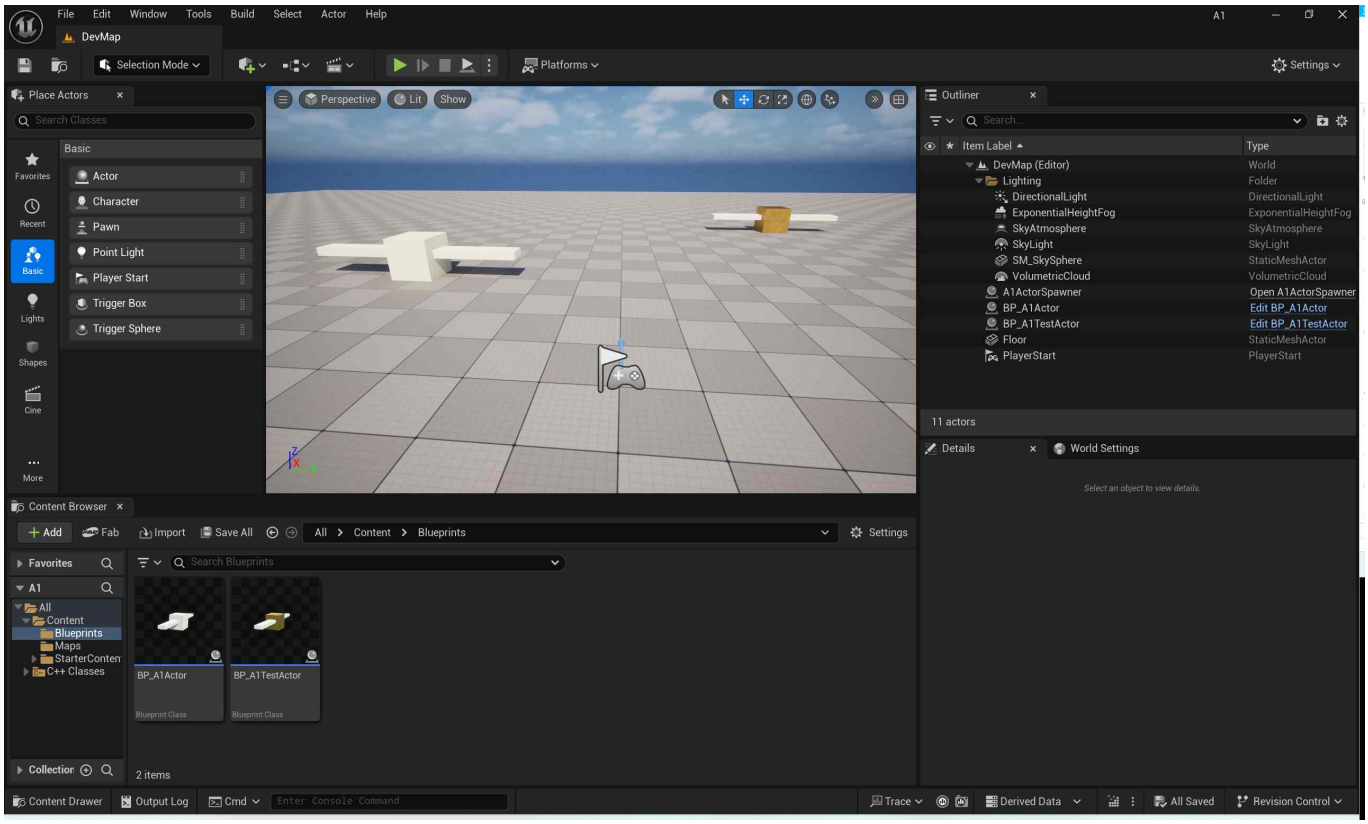
```
// Fill out your copyright notice in the Description page of Project Settings.

#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "A1Actor.generated.h"

class UA1Object;

UCLASS()
class A1_API AA1Actor : public AActor
{
	GENERATED_BODY()

public:
	// Sets default values for this actor's properties
	AA1Actor();

protected:
	// Called when the game starts or when spawned
	virtual void BeginPlay() override;

public:
	// Called every frame
	virtual void Tick(float DeltaTime) override;

protected:
	UPROPERTY(VisibleAnywhere, BlueprintReadWrite)
	TObjectPtr<class UStaticMeshComponent> Body;

	UPROPERTY(VisibleAnywhere, BlueprintReadWrite)
	TObjectPtr<class UStaticMeshComponent> Wing;

	UPROPERTY(EditAnywhere, BlueprintReadWrite)
	float MovementSpeed = 50.0f;

	UPROPERTY(EditAnywhere, Category = Target)
	TObjectPtr<AActor> Target;

	UPROPERTY(EditAnywhere, BlueprintReadWrite)
	float RotationRate = 45.f;
};
```

A1Actor.cpp

```
#include "A1Actor.h"
#include "A1Object.h"
#include "Components/StaticMeshComponent.h"
#include "Kismet/GameplayStatics.h"

// Sets default values
AA1Actor::AA1Actor()
{
	PrimaryActorTick.bCanEverTick = true;

	Body = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Body"));
	SetRootComponent(Body);

	static ConstructorHelpers::FObjectFinder<UStaticMesh>
BodyMeshRef(TEXT("/Script/Engine.StaticMesh'/Game/StarterContent/Shapes/Shape_Cube.Shape_Cube'"));
	if (BodyMeshRef.Succeeded())
	{
		Body->SetStaticMesh(BodyMeshRef.Object);
	}

	Wing = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Wing"));
	Wing->SetupAttachment(Body);
	Wing->SetRelativeLocation(FVector(0.0f, 0.0f, 50.0f));
	Wing->SetRelativeScale3D(FVector(0.5f, 5.0f, 0.2f));

	if (BodyMeshRef.Succeeded())
	{
		Wing->SetStaticMesh(BodyMeshRef.Object);
	}
}

// Called when the game starts or when spawned
void AA1Actor::BeginPlay()
{
	Super::BeginPlay();

	TArray<AActor*> Actors;
	UGameplayStatics::GetAllActorsWithTag(GetWorld(), TEXT("A1Target"), OUT Actors);

	if (Actors.Num() > 0)
	{
		Target = Actors[0];
	}
}
```

```
// Called every frame
void AA1Actor::Tick(float DeltaTime)
{
	Super::Tick(DeltaTime);

	//FVector Location = GetActorLocation();
	//FVector NewLocation = Location + FVector::ForwardVector * MovementSpeed * DeltaTime;
	//SetActorLocation(NewLocation);

	//AddActorWorldOffset(FVector::ForwardVector * MovementSpeed * DeltaTime);

	if (Target != nullptr)
	{
		FVector Location = GetActorLocation();
		FVector Direction = Target->GetActorLocation() - GetActorLocation();

		AddActorWorldOffset(Direction.GetSafeNormal() * DeltaTime * MovementSpeed);
	}

	//FRotator Rotation = GetActorRotation();
	//FRotator NewRotation = FRotator(Rotation.Pitch, Rotation.Yaw + RotationRate * DeltaTime,
Rotation.Roll);
	//SetActorRotation(NewRotation);

	AddActorWorldRotation(FRotator(0.f, RotationRate * DeltaTime, 0.f));
}
```

- 빌드 후 언리얼 에디터를 실행해 보자.



- 게임을 실행해 보면 A1Actor의 컴포넌트가 Root를 기준으로 상대 좌표로 붙어서 함께 이동하는 것을 확인할 수 있다.