LuaSampleGameMode.h

```h
// Copyright Epic Games, Inc. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"
#include "GameFramework/GameMode.h"
#include "Components/TextRenderComponent.h"
#include "MyLuaState.h"
#include "LuaSampleGameMode.generated.h"

/** GameMode class to specify pawn and playercontroller */
UCLASS(minimalapi)
class ALuaSampleGameMode : public AGameMode
{
    GENERATED_BODY()
public:
    ALuaSampleGameMode();

    UPROPERTY()
    UTextRenderComponent*       TextRender = nullptr;       // テキストレンダ

    UPROPERTY()
    UMyLuaState*    MyLuaState = nullptr;       // Luaステート

    UPROPERTY()
    FLuaValue       ScriptTable;        // スクリプトテーブル

    void        Tick(float delta_seconds) override;     // ティック

    UFUNCTION(BlueprintCallable)
    void        Initialize(UTextRenderComponent* text_render);      // 初期化
};
```

LuaSampleGameMode.cpp

```cpp
// Copyright Epic Games, Inc. All Rights Reserved.

#include "LuaSampleGameMode.h"
#include "LuaSamplePlayerController.h"
#include "LuaSamplePawn.h"
#include "LuaBlueprintFunctionLibrary.h"
#include "LuaSampleBlockGrid.h"

ALuaSampleGameMode::ALuaSampleGameMode()
{
    // no pawn by default
    DefaultPawnClass = ALuaSamplePawn::StaticClass();
    // use our own player controller class
    PlayerControllerClass = ALuaSamplePlayerController::StaticClass();

    PrimaryActorTick.bCanEverTick = true;
}

// ティック
void ALuaSampleGameMode::Tick(float delta_seconds)
{
    Super::Tick(delta_seconds);
    if(MyLuaState){
        MyLuaState->UpdateElapsedTime(delta_seconds);
    }
}

// 初期化
void ALuaSampleGameMode::Initialize(UTextRenderComponent* text_render)
{
    TextRender = text_render;
    TextRender->SetRelativeLocation(FVector(420.0f, 520.0f, 0.0f));
    TextRender->SetRelativeRotation(FRotator(90.0f, 0.0f, -180.0f));
    TextRender->SetRelativeScale3D(FVector::OneVector);
    TextRender->SetWorldSize(100.0f);
    TextRender->SetHorizontalAlignment(EHorizTextAligment::EHTA_Left);
    TextRender->SetVerticalAlignment(EVerticalTextAligment::EVRTA_TextTop);
    TextRender->SetTextRenderColor(FColor::Yellow);

    MyLuaState = NewObject<UMyLuaState>(this, TEXT("MyLuaState"));      // Luaステートを生成
    FString     script_path = FPaths::Combine(MyLuaState->LuaPath, TEXT("Start.lua"));      // スクリプトのパスを生成
    ScriptTable = ULuaBlueprintFunctionLibrary::LuaRunNonContentFile(this, MyLuaState->StaticClass(), script_path, false);        // スクリプトをロード&実行
    if(ScriptTable.IsNil() == false){
        GetWorld()->SpawnActor<ALuaSampleBlockGrid>();      // グリッドを生成
    }
}
```

LuaSampleBlockGrid.h

```
// Copyright Epic Games, Inc. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "MyLuaComponent.h"
#include "LuaSampleBlockGrid.generated.h"

/** Class used to spawn blocks and manage score */
UCLASS(minimalapi)
class ALuaSampleBlockGrid : public AActor
{
    GENERATED_BODY()

    /** Dummy root component */
    UPROPERTY(Category = Grid, VisibleDefaultsOnly, BlueprintReadOnly, meta = (AllowPrivateAccess = "true"))
    class USceneComponent* DummyRoot;

    /** Text component for the score */
    UPROPERTY(Category = Grid, VisibleDefaultsOnly, BlueprintReadOnly, meta = (AllowPrivateAccess = "true"))
    class UTextRenderComponent* ScoreText;

public:
    ALuaSampleBlockGrid();

    /** How many blocks have been clicked */
    int32 Score;

    /** Number of blocks along each side of grid */
    UPROPERTY(Category=Grid, EditAnywhere, BlueprintReadOnly)
    int32 Size;

    /** Spacing of blocks */
    UPROPERTY(Category=Grid, EditAnywhere, BlueprintReadOnly)
    float BlockSpacing;

protected:
    // Begin AActor interface
    virtual void BeginPlay() override;
    // End AActor interface

public:
    /** Handle the block being clicked */
    void AddScore();

    /** Returns DummyRoot subobject **/
    FORCEINLINE class USceneComponent* GetDummyRoot() const { return DummyRoot; }
    /** Returns ScoreText subobject **/
    FORCEINLINE class UTextRenderComponent* GetScoreText() const { return ScoreText; }

    UPROPERTY()
    ALuaSampleGameMode*     GameMode = nullptr;         // ゲームモード

    UPROPERTY()
    UMyLuaComponent*        MyLuaComponent = nullptr;   // Luaコンポーネント

    UPROPERTY()
    TArray<UBlockInfo*>     ArrayBlockInfo;             // ブロック情報配列
    float       BlockScale = 1.0f;      // ブロックスケール
    float       GridMargin = 40.0f;     // グリッド間隔

    void        Tick(float delta_seconds) override;     // ティック
    void        OnClick(int32 index);       // クリック時
    void        OnOverlap(int32 index, bool is_on);     // オーバーラップ時
    void        AddBlockInfo(int32 index, ALuaSampleBlock* block);      // ブロック情報を追加
    void        UpdateBlock(int32 index);       // ブロックを更新

    UFUNCTION()
    void        UpdateGrid(FLuaValue arg_table);        // グリッドを更新

    UFUNCTION()
    void        SetTitleText(FLuaValue arg_text);       // タイトルテキストを設定

    UFUNCTION()
    void        SetScoreText(FLuaValue arg_text);       // スコアテキストを設定
};
```

LuaSampleBlockGrid.cpp

```
// Copyright Epic Games, Inc. All Rights Reserved.

#include "LuaSampleBlockGrid.h"
#include "LuaSampleBlock.h"
#include "Components/TextRenderComponent.h"
#include "Engine/World.h"
#include "LuaSampleGameMode.h"

#define LOCTEXT_NAMESPACE "PuzzleBlockGrid"

ALuaSampleBlockGrid::ALuaSampleBlockGrid()
{
    // Create dummy root scene component
    DummyRoot = CreateDefaultSubobject<USceneComponent>(TEXT("Dummy0"));
    RootComponent = DummyRoot;

    // Create static mesh component
    ScoreText = CreateDefaultSubobject<UTextRenderComponent>(TEXT("ScoreText0"));
    ScoreText->SetRelativeLocation(FVector(200.f,0.f,0.f));
    ScoreText->SetRelativeRotation(FRotator(90.f,0.f,0.f));
    ScoreText->SetText(FText::Format(LOCTEXT("ScoreFmt", "Score: {0}"), FText::AsNumber(0)));
    ScoreText->SetupAttachment(DummyRoot);

    // Set defaults
    Size = 3;
    BlockSpacing = 300.f;

    PrimaryActorTick.bCanEverTick = true;
}

void ALuaSampleBlockGrid::BeginPlay()
```

```
{
    Super::BeginPlay();

    static const float      BLOCK_SIZE_BASIS = 260.0f;
    static const int32      COUNT_BLOCK_DEFAULT = 3;
    float       block_size_total, block_size;

    GameMode = Cast<ALuaSampleGameMode>(GetWorld()->GetAuthGameMode());
    MyLuaComponent = NewObject<UMyLuaComponent>(this, TEXT("MyLuaComponent"));
    MyLuaComponent->Initialize(this);

    block_size_total = ((BLOCK_SIZE_BASIS * COUNT_BLOCK_DEFAULT) + (GridMargin * 2.0f));
    block_size = ((block_size_total - (GridMargin * (Size - 1))) / Size);
    BlockScale = (block_size / BLOCK_SIZE_BASIS);
    BlockSpacing = (block_size + GridMargin);

    ScoreText->SetRelativeLocation(FVector(40.0f, 520.0f, 0.0f));
    ScoreText->SetRelativeRotation(FRotator(90.0f, -180.0f, 0.0f));
    ScoreText->SetRelativeScale3D(FVector::OneVector);
    ScoreText->SetWorldSize(80.0f);
    ScoreText->SetHorizontalAlignment(EHorizTextAligment::EHTA_Left);
    ScoreText->SetVerticalAlignment(EVerticalTextAligment::EVRTA_TextTop);
    ScoreText->SetTextRenderColor(FColor::White);

    // Number of blocks
    const int32 NumBlocks = Size * Size;
    // Loop to spawn each block
    for(int32 BlockIndex=0; BlockIndex<NumBlocks; BlockIndex++)
    {
        const float XOffset = (BlockIndex/Size) * BlockSpacing; // Divide by dimension
        const float YOffset = (BlockIndex%Size) * BlockSpacing; // Modulo gives remainder

        // Make position vector, offset from Grid location
        const FVector BlockLocation = FVector(XOffset, YOffset, 0.f) + GetActorLocation();

        // Spawn a block
        ALuaSampleBlock* NewBlock = GetWorld()->SpawnActor<ALuaSampleBlock>(BlockLocation, FRotator(0,0,0));

        // Tell the block about its owner
        if (NewBlock != nullptr)
        {
            NewBlock->OwningGrid = this;
            AddBlockInfo(BlockIndex, NewBlock);
        }
    }
}

void ALuaSampleBlockGrid::AddScore()
{
    // Increment score
    Score++;

    // Update text
    ScoreText->SetText(FText::Format(LOCTEXT("ScoreFmt", "Score: {0}"), FText::AsNumber(Score)));
}

// ティック
void ALuaSampleBlockGrid::Tick(float delta_seconds)
{
    Super::Tick(delta_seconds);
    if(MyLuaComponent){
        MyLuaComponent->Execute(delta_seconds);
    }
}

// クリック時
void ALuaSampleBlockGrid::OnClick(int32 index)
{
    TArray< FLuaValue >     args;
    args.Add(FLuaValue(index));
    MyLuaComponent->LuaCallTableKey(GameMode->ScriptTable, KEY_SCRIPT_FUNC_ON_CLICK, args);
}

// オーバーラップ時
void ALuaSampleBlockGrid::OnOverlap(int32 index, bool is_on)
{
    TArray< FLuaValue >     args;
    args.Add(FLuaValue(index));
    args.Add(FLuaValue(is_on));
    MyLuaComponent->LuaCallTableKey(GameMode->ScriptTable, KEY_SCRIPT_FUNC_ON_OVERLAP, args);
}
// ブロック情報を追加
void ALuaSampleBlockGrid::AddBlockInfo(int32 index, ALuaSampleBlock* target)
{
    static const float      BLOCK_SCALE_Z = 0.25f;
    float       loc_x = ((float)(index / Size) - ((Size - 1) * 0.5f)) * BlockSpacing;
    float       loc_y = ((float)(index % Size) - ((Size - 1) * 0.5f)) * BlockSpacing;

    target->BlockIndex = index;
    target->SetActorLocation(FVector(loc_x, loc_y, 0.0f));
    target->BlockMesh->SetRelativeScale3D(FVector(BlockScale, BlockScale, BLOCK_SCALE_Z));
    ArrayBlockInfo[index]->Target = target;
    UpdateBlock(index);
}

// ブロックを更新
void ALuaSampleBlockGrid::UpdateBlock(int32 index)
{
    UBlockInfo*     info = ArrayBlockInfo[index];
    ALuaSampleBlock*    target = info->Target;

    target->bIsActive = info->bIsActive;
    target->SetColor(info->Color);
    target->TextRender->SetWorldSize(info->FontSize * BlockScale);
    target->TextRender->SetText(FText::FromString(info->Text));
}

// グリッドを更新
void ALuaSampleBlockGrid::UpdateGrid(FLuaValue arg_table)
{
    TArray<int32>       array_index = MyLuaComponent->UpdateBlockInfo(arg_table);
    for(int32 i=0; i<array_index.Num(); i++){
        UpdateBlock(array_index[i]);
    }
```

```
}

// タイトルテキストを設定
void ALuaSampleBlockGrid::SetTitleText(FLuaValue arg_text)
{
    GameMode->TextRender->SetText(FText::FromString(arg_text.ToString()));
}

// スコアテキストを設定
void ALuaSampleBlockGrid::SetScoreText(FLuaValue arg_text)
{
    ScoreText->SetText(FText::FromString(arg_text.ToString()));
}

#undef LOCTEXT_NAMESPACE
```

LuaSampleBlock.h

```
// Copyright Epic Games, Inc. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "Components/TextRenderComponent.h"
#include "LuaSampleBlock.generated.h"

/** A block that can be clicked */
UCLASS(minimalapi)
class ALuaSampleBlock : public AActor
{
    GENERATED_BODY()

    /** Dummy root component */
    UPROPERTY(Category = Block, VisibleDefaultsOnly, BlueprintReadOnly, meta = (AllowPrivateAccess = "true"))
    class USceneComponent* DummyRoot;

public:
    UPROPERTY()
    UTextRenderComponent*       TextRender = nullptr;       // テキストレンダ

    UPROPERTY()
    UMaterialInstance*      GrayMaterial = nullptr;         // マテリアル（灰色）

    UPROPERTY()
    TArray<UMaterialInstance*>      ArrayMaterial;          // マテリアル配列

    int32       BlockIndex = 0;     // ブロックインデクス

    void        SetColor(int32 color_no);       // 色を設定

    /** StaticMesh component for the clickable block */
    UPROPERTY(Category = Block, VisibleDefaultsOnly, BlueprintReadOnly, meta = (AllowPrivateAccess = "true"))
    class UStaticMeshComponent* BlockMesh;

public:
    ALuaSampleBlock();

    /** Are we currently active? */
    bool bIsActive;

    /** Pointer to white material used on the focused block */
    UPROPERTY()
    class UMaterial* BaseMaterial;

    /** Pointer to blue material used on inactive blocks */
    UPROPERTY()
    class UMaterialInstance* BlueMaterial;
    /** Pointer to orange material used on active blocks */
    UPROPERTY()
    class UMaterialInstance* OrangeMaterial;

    /** Grid that owns us */
    UPROPERTY()
    class ALuaSampleBlockGrid* OwningGrid;

    /** Handle the block being clicked */
    UFUNCTION()
    void BlockClicked(UPrimitiveComponent* ClickedComp, FKey ButtonClicked);

    /** Handle the block being touched  */
    UFUNCTION()
    void OnFingerPressedBlock(ETouchIndex::Type FingerIndex, UPrimitiveComponent* TouchedComponent);

    void HandleClicked();

    void Highlight(bool bOn);

public:
    /** Returns DummyRoot subobject **/
    FORCEINLINE class USceneComponent* GetDummyRoot() const { return DummyRoot; }
    /** Returns BlockMesh subobject **/
    FORCEINLINE class UStaticMeshComponent* GetBlockMesh() const { return BlockMesh; }
};
```

LuaSampleBlock.cpp

```cpp
// Copyright Epic Games, Inc. All Rights Reserved.

#include "LuaSampleBlock.h"
#include "LuaSampleBlockGrid.h"
#include "UObject/ConstructorHelpers.h"
#include "Components/StaticMeshComponent.h"
#include "Engine/StaticMesh.h"
#include "Materials/MaterialInstance.h"

ALuaSampleBlock::ALuaSampleBlock()
{
    // Structure to hold one-time initialization
    struct FConstructorStatics
    {
        ConstructorHelpers::FObjectFinderOptional<UStaticMesh> PlaneMesh;
        ConstructorHelpers::FObjectFinderOptional<UMaterial> BaseMaterial;
        ConstructorHelpers::FObjectFinderOptional<UMaterialInstance> BlueMaterial;
        ConstructorHelpers::FObjectFinderOptional<UMaterialInstance> OrangeMaterial;
        ConstructorHelpers::FObjectFinderOptional<UMaterialInstance> GrayMaterial;
        FConstructorStatics()
            : PlaneMesh(TEXT("/Game/Puzzle/Meshes/PuzzleCube.PuzzleCube"))
            , BaseMaterial(TEXT("/Game/Puzzle/Meshes/BaseMaterial.BaseMaterial"))
            , BlueMaterial(TEXT("/Game/Puzzle/Meshes/BlueMaterial.BlueMaterial"))
            , OrangeMaterial(TEXT("/Game/Puzzle/Meshes/OrangeMaterial.OrangeMaterial"))
            , GrayMaterial(TEXT("/Game/Puzzle/Meshes/GrayMaterial.GrayMaterial"))
        {
        }
    };
    static FConstructorStatics ConstructorStatics;

    // Create dummy root scene component
    DummyRoot = CreateDefaultSubobject<USceneComponent>(TEXT("Dummy0"));
    RootComponent = DummyRoot;

    // Create static mesh component
    BlockMesh = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("BlockMesh0"));
    BlockMesh->SetStaticMesh(ConstructorStatics.PlaneMesh.Get());
    BlockMesh->SetRelativeScale3D(FVector(1.f,1.f,0.25f));
    BlockMesh->SetRelativeLocation(FVector(0.f,0.f,25.f));
    BlockMesh->SetMaterial(0, ConstructorStatics.BlueMaterial.Get());
    BlockMesh->SetupAttachment(DummyRoot);
    BlockMesh->OnClicked.AddDynamic(this, &ALuaSampleBlock::BlockClicked);
    BlockMesh->OnInputTouchBegin.AddDynamic(this, &ALuaSampleBlock::OnFingerPressedBlock);

    // Save a pointer to the orange material
    BaseMaterial = ConstructorStatics.BaseMaterial.Get();
    BlueMaterial = ConstructorStatics.BlueMaterial.Get();
    OrangeMaterial = ConstructorStatics.OrangeMaterial.Get();
    GrayMaterial = ConstructorStatics.GrayMaterial.Get();
    ArrayMaterial.Add((UMaterialInstance*)BaseMaterial);
    ArrayMaterial.Add(GrayMaterial);
    ArrayMaterial.Add(BlueMaterial);
    ArrayMaterial.Add(OrangeMaterial);

    TextRender = CreateDefaultSubobject<UTextRenderComponent>(TEXT("BlockText"));
    TextRender->SetRelativeLocation(FVector(0.0f, 0.0f, 60.0f));
    TextRender->SetRelativeRotation(FRotator(90.0f, 180.0f, 0.0f));
    TextRender->SetHorizontalAlignment(EHorizTextAligment::EHTA_Center);
    TextRender->SetVerticalAlignment(EVerticalTextAligment::EVRTA_TextCenter);
    TextRender->SetTextRenderColor(FColor::Black);
    TextRender->AttachToComponent(RootComponent, FAttachmentTransformRules::KeepRelativeTransform);
}

void ALuaSampleBlock::BlockClicked(UPrimitiveComponent* ClickedComp, FKey ButtonClicked)
{
    HandleClicked();
}

void ALuaSampleBlock::OnFingerPressedBlock(ETouchIndex::Type FingerIndex, UPrimitiveComponent* TouchedComponent)
{
    HandleClicked();
}

void ALuaSampleBlock::HandleClicked()
{
    OwningGrid->OnClick(BlockIndex + 1);
}

void ALuaSampleBlock::Highlight(bool bOn)
{
    OwningGrid->OnOverlap((BlockIndex + 1), bOn);
}

void ALuaSampleBlock::SetColor(int32 color_no)
{
    int32   index = FMath::Clamp<int32>(color_no, 0, (ArrayMaterial.Num() - 1));
    BlockMesh->SetMaterial(0, ArrayMaterial[index]);
}
```