MyLuaState.h

```
#pragma once

#include "CoreMinimal.h"
#include "LuaState.h"
#include "MyLuaState.generated.h"

// Lua ステートクラス
UCLASS()
class UMyLuaState : public ULuaState
{
    GENERATED_BODY()
public:
    FString     LuaPath;    // Lua パス

    UMyLuaState();          // コンストラクタ
    void        ReceiveLuaError_Implementation(const FString& error_text);      // Lua エラー受信（実装）
    void        UpdateElapsedTime(float delta_seconds);     // 経過時間を更新

    UFUNCTION()
    void        Print(FLuaValue arg_text);      // 文字列を出力

    UFUNCTION()
    void        Quit();     // 終了
};
```

MyLuaState.cpp

```cpp
#include "MyLuaState.h"
#include "LuaBlueprintFunctionLibrary.h"
#include "Kismet/GameplayStatics.h"
#include "Kismet/KismetSystemLibrary.h"
#include "LuaSampleGameMode.h"

// Lua キー（グローバル）
#define KEY_GLOBAL_STRING_LUA_PATH          "lua_path"          // Lua パス
#define KEY_GLOBAL_FLOAT_ELAPSED_TIME       "elapsed_time"      // 経過時間
#define KEY_GLOBAL_FUNC_PRINT               "print"             // 文字列を出力
#define KEY_GLOBAL_FUNC_QUIT                "quit"              // 終了

// コンストラクタ
UMyLuaState::UMyLuaState(void)
{
    LuaPath = FPaths::Combine(FPaths::ProjectContentDir(), TEXT("NonAssets/Lua/"));
    Table.Add(KEY_GLOBAL_STRING_LUA_PATH, ULuaBlueprintFunctionLibrary::LuaCreateString(LuaPath));
    Table.Add(KEY_GLOBAL_FLOAT_ELAPSED_TIME, ULuaBlueprintFunctionLibrary::LuaCreateNumber(0.0f));
    Table.Add(KEY_GLOBAL_FUNC_PRINT, ULuaBlueprintFunctionLibrary::LuaCreateUFunction(UMyLuaState::StaticClass(),
(GET_FUNCTION_NAME_CHECKED(UMyLuaState, Print)).ToString()));
    Table.Add(KEY_GLOBAL_FUNC_QUIT, ULuaBlueprintFunctionLibrary::LuaCreateUFunction(UMyLuaState::StaticClass(),
(GET_FUNCTION_NAME_CHECKED(UMyLuaState, Quit)).ToString()));
}

// Lua エラー受信（実装）
void UMyLuaState::ReceiveLuaError_Implementation(const FString& error_text)
{
    FPlatformMisc::MessageBoxExt(EAppMsgType::Ok, *error_text, TEXT("Error"));
    Quit();
}

// 経過時間を更新
void UMyLuaState::UpdateElapsedTime(float delta_seconds)
{
    FLuaValue       lua_value = ULuaBlueprintFunctionLibrary::LuaGetGlobal(this, UMyLuaState::StaticClass(), KEY_G
LOBAL_FLOAT_ELAPSED_TIME);
    ULuaBlueprintFunctionLibrary::LuaSetGlobal(this, UMyLuaState::StaticClass(), KEY_GLOBAL_FLOAT_ELAPSED_TIME, (l
ua_value.ToFloat() + delta_seconds));
}

// 文字列を出力
void UMyLuaState::Print(FLuaValue arg_text)
{
    GEngine->AddOnScreenDebugMessage(INDEX_NONE, FLT_MAX, FColor::Cyan, *(arg_text.ToString()));
}

// 終了
void UMyLuaState::Quit(void)
{
    UKismetSystemLibrary::QuitGame(this, nullptr, EQuitPreference::Quit, false);
}
```

MyLuaComponent.h

```h
#pragma once

#include "CoreMinimal.h"
#include "LuaComponent.h"
#include "MyLuaState.h"
#include "MyLuaComponent.generated.h"

class ALuaSampleBlock;
class ALuaSampleBlockGrid;
class ALuaSampleGameMode;

// Lua キー（スクリプト）
#define KEY_SCRIPT_FUNC_INITIALIZE      "initialize"        // 初期化
#define KEY_SCRIPT_FUNC_EXECUTE         "execute"           // 実行
#define KEY_SCRIPT_FUNC_ON_CLICK        "on_click"          // クリック時
#define KEY_SCRIPT_FUNC_ON_OVERLAP      "on_overlap"        // オーバーラップ時

// ブロック情報
UCLASS()
class UBlockInfo : public UObject
{
    GENERATED_BODY()
public:
    UPROPERTY()
    ALuaSampleBlock*    Target = nullptr;

    bool        bIsActive = false;
    int32       Color = 0;
    float       Scale = 1.0f;
    float       FontSize = 180.0f;
    FString     Text;
};

// Lua コンポーネントクラス
UCLASS()
class UMyLuaComponent : public ULuaComponent
{
    GENERATED_BODY()
public:
    UPROPERTY()
    ALuaSampleBlockGrid*    Owner = nullptr;    // 所有者

    UMyLuaComponent(){}
    void        Initialize(ALuaSampleBlockGrid* owner);     // 初期化
    void        Execute(float delta_seconds);               // 実行
    TArray<int32>       UpdateBlockInfo(FLuaValue arg_table = FLuaValue());     // ブロック情報を更新
};
```

MyLuaComponent.cpp

```cpp
#include "MyLuaComponent.h"
#include "LuaSampleBlockGrid.h"
#include "LuaBlueprintFunctionLibrary.h"
#include "Kismet/GameplayStatics.h"
#include "LuaSampleGameMode.h"

// Lua キー（ローカル）
#define KEY_LOCAL_INT_GRID_SIZE             "grid_size"         // グリッドサイズ
#define KEY_LOCAL_FLOAT_GRID_MARGIN         "grid_margin"       // グリッド間隔
#define KEY_LOCAL_TABLE_BLOCK_INFO          "block_info"        // ブロック情報
#define KEY_LOCAL_FUNC_SET_TITLE_TEXT       "set_title_text"    // タイトルテキストを設定
#define KEY_LOCAL_FUNC_SET_SCORE_TEXT       "set_score_text"    // スコアテキストを設定
#define KEY_LOCAL_FUNC_UPDATE_GRID          "update_grid"       // グリッドを更新

// Lua キー（ブロック情報）
#define KEY_BLOCK_INFO_BOOL_IS_ACTIVE       "is_active"         // アクティブか
#define KEY_BLOCK_INFO_INT_COLOR            "color"             // 色
#define KEY_BLOCK_INFO_FLOAT_FONT_SIZE      "font_size"         // フォントサイズ
#define KEY_BLOCK_INFO_STRING_TEXT          "text"              // テキスト

// 初期化
void UMyLuaComponent::Initialize(ALuaSampleBlockGrid* owner)
{
    Owner = owner;
    LuaState = UMyLuaState::StaticClass();

    // Lua テーブルを作成
    Table.Add(KEY_LOCAL_INT_GRID_SIZE, ULuaBlueprintFunctionLibrary::LuaCreateInteger(Owner->Size));
    Table.Add(KEY_LOCAL_FLOAT_GRID_MARGIN, ULuaBlueprintFunctionLibrary::LuaCreateNumber(Owner->GridMargin));
    Table.Add(KEY_LOCAL_TABLE_BLOCK_INFO, ULuaBlueprintFunctionLibrary::LuaCreateTable(Owner, LuaState));
    Table.Add(KEY_LOCAL_FUNC_SET_TITLE_TEXT, ULuaBlueprintFunctionLibrary::LuaCreateUFunction(Owner, (GET_FUNCTION
_NAME_CHECKED(ALuaSampleBlockGrid, SetTitleText)).ToString()));
    Table.Add(KEY_LOCAL_FUNC_SET_SCORE_TEXT, ULuaBlueprintFunctionLibrary::LuaCreateUFunction(Owner, (GET_FUNCTION
_NAME_CHECKED(ALuaSampleBlockGrid, SetScoreText)).ToString()));
    Table.Add(KEY_LOCAL_FUNC_UPDATE_GRID, ULuaBlueprintFunctionLibrary::LuaCreateUFunction(Owner, (GET_FUNCTION_NA
ME_CHECKED(ALuaSampleBlockGrid, UpdateGrid)).ToString()));

    // Lua 関数を実行
    LuaCallTableKey(Owner->GameMode->ScriptTable, KEY_SCRIPT_FUNC_INITIALIZE, TArray<FLuaValue>());

    // パラメータを更新
    Owner->Size = FMath::Clamp<int32>(LuaGetField(KEY_LOCAL_INT_GRID_SIZE).ToInteger(), 1, 8);
    Owner->GridMargin = FMath::Clamp<float>(LuaGetField(KEY_LOCAL_FLOAT_GRID_MARGIN).ToFloat(), 0.0f, 80.0f);
    UpdateBlockInfo();
}

// 実行
void UMyLuaComponent::Execute(float delta_seconds)
{
    TArray<FLuaValue>       args;

    // Lua 関数を実行
    args.Add(FLuaValue(delta_seconds));
    LuaCallTableKey(Owner->GameMode->ScriptTable, KEY_SCRIPT_FUNC_EXECUTE, args);
}
```

```
// ブロック情報を更新
TArray<int32> UMyLuaComponent::UpdateBlockInfo(FLuaValue arg_table)
{
    int32       block_count_max = (Owner->Size * Owner->Size);

    // 初回は配列を作成
    if(Owner->ArrayBlockInfo.Num() < 1){
        for(int32 i=0; i<block_count_max; i++){
            Owner->ArrayBlockInfo.Add(NewObject<UBlockInfo>());
        }
    }

    // インデクス配列を作成
    TArray<int32>       array_index;

    for(int32 i=0; i<block_count_max; i++){
        if(arg_table.IsNil()){
            array_index.Add(i);     // 引数が無い場合は全て登録
        }else{
            FLuaValue       lua_value = arg_table.GetFieldByIndex(i + 1);
            if(lua_value.IsNil()){
                break;
            }
            int32       index = (lua_value.ToInteger() - 1);
            array_index.Add(FMath::Clamp<int32>(index, 0, (block_count_max - 1)));
        }
    }

    // ブロック情報を更新
    FLuaValue       table_block_info = LuaGetField(KEY_LOCAL_TABLE_BLOCK_INFO);

    for(int32 i=0; i<array_index.Num(); i++){
        FLuaValue       lua_value = table_block_info.GetFieldByIndex(array_index[i] + 1);
        UBlockInfo*     block_info = Owner->ArrayBlockInfo[array_index[i]];

        block_info->bIsActive = lua_value.GetField(KEY_BLOCK_INFO_BOOL_IS_ACTIVE).ToBool();
        block_info->Color = lua_value.GetField(KEY_BLOCK_INFO_INT_COLOR).ToInteger();
        block_info->FontSize = lua_value.GetField(KEY_BLOCK_INFO_FLOAT_FONT_SIZE).ToFloat();
        block_info->Text = lua_value.GetField(KEY_BLOCK_INFO_STRING_TEXT).ToString();
    }

    return array_index;
}
```