```
#include "Vector2.h"
#include "GameScene.h"
#include "DxLib.h"
#include "GameCommon.h"
#include "SceneManager.h"
#include "GameScene.h"
#include "Stage.h"
#include "Box.h"

Box::Box(GameScene* scene)
{
    mGameScene = scene;
    mSceneManager = scene->GetSceneManager();
}

void Box::Init(Vector2 mapPos)
{

    mImage = LoadGraph("Image/Box.png");

    // マップ座標をスクリーン座標へ変換
    mPos = {
        mapPos.x * BLOCK_SIZE,
        mapPos.y * BLOCK_SIZE
    };

    mMvSPos = { 0, 0 };
    mMvEPos = { 0, 0 };

    mStepMove = 0.0f;

    mDir = DIR::DOWN;

    mIsStayStorage = false;

    // ×
    //mState = STATE::IDLE;
    ChangeState(STATE::IDLE);

}

void Box::Update(void)
{

    switch (mState)
    {
    case Box::STATE::IDLE:
        break;
```

```
    case Box::STATE::MOVE:
    case Box::STATE::BACK_MOVE:
    {

        // 経過時間を取得して、処理時間を計測
        mStepMove += mSceneManager->GetDeltaTime();

        // 経過率を算出
        float t = mStepMove / TIME_MOVE;

        // 率に応じた座標(開始座標～終了座標)
        mPos = Vector2::Lerp(mMvSPos, mMvEPos, t);
        if (t >= 1.0f)
        {
            mPos = mMvEPos;
            ChangeState(STATE::IDLE);
            return;
        }
        break;

    }
    }

}

void Box::Draw(void)
{

    int x = GAME_AREA_X + mPos.x;
    int y = GAME_AREA_Y + mPos.y;

    if (mIsStayStorage == true)
    {
        // 設置されている
        DrawGraph(x, y, mImage, true);
        // 色の乗算
        SetDrawBlendMode(DX_BLENDMODE_MULA, 255);
        DrawGraph(x, y, mImage, true);
        // ブレンドモードオフ
        SetDrawBlendMode(DX_BLENDMODE_NOBLEND, 0);
    }
    else
    {
        // 設置されていない
        DrawGraph(x, y, mImage, true);
    }
```

```
    // 色の合成お試し
    //DrawTest();

}

void Box::DrawTest(void)
{

    int miniSize = 4;
    int size = BLOCK_SIZE * miniSize;

    // 描画空間を作成
    int img01 = MakeScreen(size, size, false);

    // 作成した描画空間を描画対象とする
    SetDrawScreen(img01);

    // 黒 GetColor(0, 0, 0)    白 GetColor(255, 255, 255)
    int c01 = 255 / (BLOCK_SIZE * 2);
    for(int y = 0;y < BLOCK_SIZE;y++)
    {
        for (int x = 0; x < BLOCK_SIZE; x++)
        {

            int tmp = c01 * (x + y);
            DrawBox(
                (x * miniSize),
                (y * miniSize),
                (x * miniSize) + miniSize,
                (y * miniSize) + miniSize,
                GetColor(tmp, tmp, tmp), true
            );

        }
    }

    // 描画対象を裏画面にする
    SetDrawScreen(DX_SCREEN_BACK);
    // 裏画面にオリジナルの描画空間を描画
    DrawGraph(20, 20, img01, true);



    // 描画空間を作成
    int img02 = MakeScreen(size, size, false);

    // 作成した描画空間を描画対象とする
    SetDrawScreen(img02);
```

```
// 黒 GetColor(0, 0, 0)     白 GetColor(255, 255, 255)
int c02 = 255 / BLOCK_SIZE;
for (int y = 0; y < BLOCK_SIZE; y++)
{
    for (int x = 0; x < BLOCK_SIZE; x++)
    {

        int tmp = c02 * x;
        DrawBox(
            (x * miniSize),
            (y * miniSize),
            (x * miniSize) + miniSize,
            (y * miniSize) + miniSize,
            GetColor(tmp, tmp, tmp), true
        );

    }
}


// 描画対象を裏画面にする
SetDrawScreen(DX_SCREEN_BACK);
// 裏画面にオリジナルの描画空間を描画
DrawGraph(200, 20, img02, true);

// 合成その①　色の乗算
//-----------------------------------------
int imgMix01 = MakeScreen(size, size, false);
SetDrawScreen(imgMix01);
DrawGraph(0, 0, img01, true);

// DxLibの色の乗算
SetDrawBlendMode(DX_BLENDMODE_MULA, 255);
DrawGraph(0, 0, img02, true);
// ブレンドモードをオフ
SetDrawBlendMode(DX_BLENDMODE_NOBLEND, 0);

// 描画対象を裏画面にする
SetDrawScreen(DX_SCREEN_BACK);
DrawGraph(400, 20, imgMix01, true);
//-----------------------------------------

// 合成その②　色の乗算(手作りVer)
//-----------------------------------------
int imgMix02 = MakeScreen(size, size, false);
SetDrawScreen(imgMix02);
```

```
for (int y = 0; y < BLOCK_SIZE; y++)
{
    for (int x = 0; x < BLOCK_SIZE; x++)
    {

        float tmp1 = c01 * (x + y);
        float tmp2 = c02 * x;

        // 色の乗算
        float tmp =
            (tmp1 / 255.0f) * (tmp2 / 255.0f) * 255.0f;

        DrawBox(
            (x * miniSize),
            (y * miniSize),
            (x * miniSize) + miniSize,
            (y * miniSize) + miniSize,
            GetColor(tmp, tmp, tmp), true
        );

    }
}

// 描画対象を裏画面にする
SetDrawScreen(DX_SCREEN_BACK);
DrawGraph(600, 20, imgMix02, true);
//-----------------------------------------


// 合成その③　色の加算
//-----------------------------------------
int imgMix03 = MakeScreen(size, size, false);
SetDrawScreen(imgMix03);
DrawGraph(0, 0, img01, true);

// DxLibの色の加算
SetDrawBlendMode(DX_BLENDMODE_ADD, 255);
DrawGraph(0, 0, img02, true);
// ブレンドモードをオフ
SetDrawBlendMode(DX_BLENDMODE_NOBLEND, 0);

// 描画対象を裏画面にする
SetDrawScreen(DX_SCREEN_BACK);
DrawGraph(400, 200, imgMix03, true);
//-----------------------------------------
```

```
    // 合成その④　色の加算(手作りVer)
    //-------------------------------------------
    int imgMix04 = MakeScreen(size, size, false);
    SetDrawScreen(imgMix04);

    for (int y = 0; y < BLOCK_SIZE; y++)
    {
        for (int x = 0; x < BLOCK_SIZE; x++)
        {

            float tmp1 = c01 * (x + y);
            float tmp2 = c02 * x;

            // 色の加算(足し算、255を超える場合は255にする)
            //-------------------------------------------
            float tmp =
                ((tmp1 / 255.0f) + (tmp2 / 255.0f)) * 255.0f;
            if (tmp > 255.0f)
            {
                tmp = 255.0f;
            }
            //-------------------------------------------

            DrawBox(
                (x * miniSize),
                (y * miniSize),
                (x * miniSize) + miniSize,
                (y * miniSize) + miniSize,
                GetColor(tmp, tmp, tmp), true
            );

        }
    }

    // 描画対象を裏画面にする
    SetDrawScreen(DX_SCREEN_BACK);
    DrawGraph(600, 200, imgMix04, true);
    //-------------------------------------------

}

void Box::Release(void)
{
    DeleteGraph(mImage);
}

Vector2 Box::GetPos(void)
{
```

```
    return mPos;
}

void Box::Push(DIR dir)
{

    // ①方向の保持
    mDir = dir;

    // ②移動状態に遷移させる
    ChangeState(STATE::MOVE);

}

bool Box::IsPossiblePush(DIR dir)
{

    // ①-①移動先のマップ座標
    Vector2 mapPos = mPos;
    mapPos.x /= BLOCK_SIZE;
    mapPos.y /= BLOCK_SIZE;

    // 移動先座標
    switch (dir)
    {
    case DIR::DOWN:
        mapPos.y += 1;
        break;
    case DIR::LEFT:
        mapPos.x -= 1;
        break;
    case DIR::RIGHT:
        mapPos.x += 1;
        break;
    case DIR::UP:
        mapPos.y -= 1;
        break;
    }

    // 移動先の壁との衝突チェック
    if (mGameScene->GetStage()->IsCollosion(mapPos) == true)
    {
        return false;
    }

    // 移動先の荷物との衝突チェック
    Vector2 mvEPos = mapPos;
    mvEPos.x *= BLOCK_SIZE;
```

```
    mvEPos.y *= BLOCK_SIZE;
    Box* box = mGameScene->GetCollsionBox(mvEPos);
    if (box != nullptr)
    {
        return false;
    }

    return true;

}

bool Box::IsStayStorage(void)
{
    return mIsStayStorage;
}

void Box::BackMove(GameScene::History his)
{
    mHistroy = his;
    ChangeState(STATE::BACK_MOVE);
}

void Box::ChangeState(STATE state)
{

    // 状態を変更する
    mState = state;

    // 状態ごとの初期処理を行う
    switch (mState)
    {
    case Box::STATE::IDLE:
    {

        // 荷物置き場判定
        Storage* storage =
            mGameScene->GetCollsionStorage(mPos);

        if (storage != nullptr)
        {
            mIsStayStorage = true;
        }
        else
        {
            mIsStayStorage = false;
        }

        break;
```

```
    }
    case Box::STATE::MOVE:
    {
        // ③線形補間による移動処理を作成

        // 経過時間を初期化
        mStepMove = 0.0f;

        // 移動元座標を現在座標に
        mMvSPos = mMvEPos = mPos;

        // 移動先座標
        switch (mDir)
        {
        case DIR::DOWN:
            mMvEPos.y += BLOCK_SIZE;
            break;
        case DIR::LEFT:
            mMvEPos.x -= BLOCK_SIZE;
            break;
        case DIR::RIGHT:
            mMvEPos.x += BLOCK_SIZE;
            break;
        case DIR::UP:
            mMvEPos.y -= BLOCK_SIZE;
            break;
        }
        break;

    }

    case Box::STATE::BACK_MOVE:
    {
        // 経過時間を初期化
        mStepMove = 0.0f;

        // 方向を設定
        mDir = mHistroy.dir;

        // 移動元座標
        mMvSPos = mPos;

        // 移動先座標
        mMvEPos = mHistroy.boxPos;

        break;
    }
    }
```

}