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

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

void Unit::Init(Vector2 mapPos)
{

    LoadDivGraph(
        "Image/Unit.png",
        NUM_ANIM_X * NUM_ANIM_Y,
        NUM_ANIM_X, NUM_ANIM_Y,
        IMG_SIZE, IMG_SIZE,
        &mImages[0][0], true);

    LoadDivGraph(
        "Image/UnitPush.png",
        NUM_ANIM_X * NUM_ANIM_Y,
        NUM_ANIM_X, NUM_ANIM_Y,
        IMG_SIZE, IMG_SIZE,
        &mImagesPush[0][0], true);

    // 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;

    mCntAnim = 0;

    mIsPushing = false;
```

```
    // ×基本的には、直接変更はダメです
    //mState = STATE::IDLE;
    ChangeState(STATE::IDLE);

}

void Unit::Update(void)
{

    switch (mState)
    {
    case Unit::STATE::IDLE:

        if (keyNew[KEY_P1_RIGHT])
        {
            mDir = DIR::RIGHT;
            ChangeState(STATE::MOVE);
            return;
        }

        if (keyNew[KEY_P1_LEFT])
        {
            mDir = DIR::LEFT;
            ChangeState(STATE::MOVE);
            return;
        }

        if (keyNew[KEY_P1_UP])
        {
            mDir = DIR::UP;
            ChangeState(STATE::MOVE);
            return;
        }

        if (keyNew[KEY_P1_DOWN])
        {
            mDir = DIR::DOWN;
            ChangeState(STATE::MOVE);
            return;
        }

        mIsPushing = false;

        break;

    case Unit::STATE::MOVE:
    case Unit::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;

    }
    default:
        break;
    }

}

void Unit::Draw(void)
{

    int animId = 1;
    switch (mState)
    {
    case Unit::STATE::IDLE:

        // アニメーションしない
        animId = 1;
        break;

    case Unit::STATE::MOVE:
    case Unit::STATE::BACK_MOVE:

        mCntAnim += 1;
        animId = (mCntAnim / SPEED_SLOW_ANIM) % CNT_ANIM;
        if (animId == 3)
        {
            animId = 1;
        }
        break;
    }
```

```
    int image = -1;
    if (mIsPushing == true)
    {
        // 押し出し画像
        image = mImagesPush[(int)mDir][animId];
    }
    else
    {
        // 通常画像
        image = mImages[(int)mDir][animId];
    }

    DrawGraph(
        GAME_AREA_X + mPos.x, GAME_AREA_Y + mPos.y,
        image, true);

}

void Unit::Release(void)
{

    for (int y = 0; y < NUM_ANIM_Y; y++)
    {
        for (int x = 0; x < NUM_ANIM_X; x++)
        {
            DeleteGraph(mImages[y][x]);
            DeleteGraph(mImagesPush[y][x]);
        }
    }

}

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

bool Unit::IsEnableBack(void)
{
    return mState == STATE::IDLE;
}

// 状態遷移
void Unit::ChangeState(STATE state)
{
```

```
    // 状態を変更
    mState = state;

    // 状態ごとの初期処理を行う
    switch (mState)
    {
    case Unit::STATE::IDLE:
        break;
    case Unit::STATE::MOVE:
    {
        // 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;
        }

        // 移動先座標(mMvEPos)を、マップ座標に変換
        Vector2 mapPos = { 0 ,0 };
        mapPos.x = mMvEPos.x / BLOCK_SIZE;
        mapPos.y = mMvEPos.y / BLOCK_SIZE;

        // 移動先の衝突チェック
        if (mGameScene->GetStage()->IsCollosion(mapPos) == true)
        {
            ChangeState(STATE::IDLE);
            return;
        }

        // 移動先に荷物があるかチェック
        Box* box = mGameScene->GetCollsionBox(mMvEPos);
        if (box != nullptr)
        {
```

```
            mIsPushing = true;

            // 荷物が進行方向に移動できるかどうか
            if (box->IsPossiblePush(mDir) == true)
            {
                // 動いて！(Boxに命令する)
                box->Push(mDir);
            }
            else
            {

                // Boxに命令はしない

                // ユニットも移動しないようにする
                ChangeState(STATE::IDLE);
                return;

            }

        }
        else
        {
            mIsPushing = false;
        }

        // 移動情報が確定
        //-----------------------------------------------
        if (mIsPushing == true)
        {
            // 荷物を押し出している場合
            mGameScene->RegistHistory(mDir, mMvSPos, box);
        }
        else
        {
            // 荷物を押し出していない場合
            mGameScene->RegistHistory(mDir, mMvSPos, nullptr);
        }
        //-----------------------------------------------

        // スコア記録
        mGameScene->PlusCntMove();

        break;
    }
    case Unit::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.unitPos;

        break;
    }
    default:
        break;
    }

}
```