```
#include <vector>
#include <string>
#include <fstream>
#include <stack>
#include "DxLib.h"
#include "Utility.h"
#include "KeyCheck.h"
#include "GameCommon.h"
#include "SceneManager.h"
#include "Stage.h"
#include "Unit.h"
#include "Box.h"
#include "Storage.h"
#include "Fader.h"
#include "TimeLimit.h"
#include "GameScene.h"

GameScene::GameScene(SceneManager* manager) : SceneBase(manager)
{
    mStageNo = 1;

    ChangeState(STATE::GAME);

    mFader = new Fader();
    mFader->Init();

    mTimeLimit = new TimeLimit(manager);

}

/// <summary>
/// 初期化
/// </summary>
/// <param name=""></param>
/// <returns></returns>
void GameScene::Init(void)
{

    // ステージクリア画像の読み込み
    mImageClear = LoadGraph("Image/Congratulations.png");

    // ステージ
    mStage = new Stage(this);
    mStage->Init(mStageNo);

    // コードでステージ設定
    //SetStage();
```

```
    // 外部ファイルを使用してステージ設定
    LoadGimmickData();

    // 外部ファイルを使用してBestスコアを読み取る
    LoadScore();

    // 現在のステージのBestスコアを取得する
    mBestScore = GetBestScore();

    mStepClear = 0.0f;

    // 時間制限
    mTimeLimit->Start(mStageNo * 60);

    // 移動歩数の初期化
    mCntMove = 0;

}

/// <summary>
/// 更新ステップ
/// </summary>
/// <param name=""></param>
/// <returns></returns>
void GameScene::Update(void)
{

    if (keyTrgDown[KEY_SYS_START])
    {
        mSceneManager->ChangeScene(SCENE_ID::GAMEOVER, true);
    }

    switch (mState)
    {
    case GameScene::STATE::GAME:
        UpdateGame();
        break;
    case GameScene::STATE::CLEAR:
        UpdateClear();
        break;
    case GameScene::STATE::CHANGE_STAGE:
        UpdateChangeStage();
        break;
    }

}

void GameScene::UpdateGame(void)
```

```
{

    mStage->Update();
    mUnit->Update();

    int size;

    // 荷物
    size = mBoxes.size();
    for (int i = 0; i < size; i++)
    {
        mBoxes[i]->Update();
    }

    // 荷物置き場
    size = mStorages.size();
    for (int i = 0; i < size; i++)
    {
        mStorages[i]->Update();
    }

    // 操作を戻す(Nキー)
    if (keyTrgDown[KEY_P1_A]
        && mHistoryBack.size() > 0
        && mUnit->IsEnableBack())
    {

        // 最後の操作情報を取得
        History his = mHistoryBack.top();

        // 荷物を移動させているか判断
        if (his.box != nullptr)
        {
            // Boxに巻き戻し処理をさせる
            his.box->BackMove(his);
        }

        // 最後の操作情報をユニットに渡して巻き戻し処理をさせる
        mUnit->BackMove(his);

        // 最後の操作情報を削除
        mHistoryBack.pop();

        // スコア
        MinusCntMove();

    }
```

```
    // 時間制限
    mTimeLimit->Update();
    if (mTimeLimit->IsTimeOver() == true)
    {
        mSceneManager->ChangeScene(SCENE_ID::GAMEOVER, true);
        return;
    }

    // クリア判定
    bool isClear = true;
    size = mBoxes.size();
    for (int i = 0; i < size; i++)
    {
        // 判定処理
        if (mBoxes[i]->IsStayStorage() == false)
        {
            isClear = false;
            break;
        }
    }

    if (isClear == true)
    {

        if (mStageNo >= MAX_STAGE_NO)
        {
            // ゲームオーバー画面へ遷移
            mSceneManager->ChangeScene(SCENE_ID::GAMEOVER, true);
        }
        else
        {

            // 次のステージへ
            //ChangeStage();

            // クリアー表示
            ChangeState(STATE::CLEAR);
            return;

        }
    }

}

void GameScene::UpdateClear(void)
{

    // 経過時間
```

```
    mStepClear += mSceneManager->GetDeltaTime();

    // ３秒経過したら次のStateへ移行
    if (mStepClear > TIME_CLEAR_MESSAGE)
    {
        ChangeState(STATE::CHANGE_STAGE);
        return;
    }

}

void GameScene::UpdateChangeStage(void)
{

    mFader->Update();

    Fader::FADE_STATE state = mFader->GetState();

    switch (state)
    {
    case Fader::FADE_STATE::FADE_OUT:
        // 段々暗くする
        if (mFader->IsEnd() == true)
        {
            // ステージ切り替え
            ChangeStage();
            // 徐々に明るくする
            mFader->SetFade(Fader::FADE_STATE::FADE_IN);
        }
        break;
    case Fader::FADE_STATE::FADE_IN:
        // 段々明るくする
        if (mFader->IsEnd() == true)
        {
            mFader->SetFade(Fader::FADE_STATE::NONE);
            ChangeState(STATE::GAME);
        }
        break;
    }



}

/// <summary>
/// 描画処理
/// </summary>
/// <param name=""></param>
```

```
void GameScene::Draw(void)
{

    SetDrawScreen(DX_SCREEN_BACK);

    // 画面のクリア
    ClearDrawScreen();

    switch (mState)
    {
    case GameScene::STATE::GAME:
        DrawGame();
        break;
    case GameScene::STATE::CLEAR:
        DrawClear();
        break;
    case GameScene::STATE::CHANGE_STAGE:
        DrawChangeStage();
        break;
    }

}

void GameScene::DrawGame(void)
{

    int size;

    mStage->Draw();

    // 荷物置き場
    size = mStorages.size();
    for (int i = 0; i < size; i++)
    {
        mStorages[i]->Draw();
    }

    mUnit->Draw();

    // 動的配列
    size = mBoxes.size();
    for (int i = 0; i < size; i++)
    {
        mBoxes[i]->Draw();
    }

    // 時間制限
    mTimeLimit->Draw();
```

```
    // スコア表示
    DrawScore();

}

void GameScene::DrawClear(void)
{

    DrawGame();

    // ステージクリア画像の表示
    DrawGraph(
        SCREEN_SIZE_X / 2 - (500 / 2),
        SCREEN_SIZE_Y / 2,
        mImageClear, true
    );

}

void GameScene::DrawChangeStage(void)
{
    DrawGame();
}

void GameScene::DrawScore(void)
{

    int x1;
    int x2;
    int y1 = 10;
    int y2 = 60;
    int width = 100;
    int charX;
    int charY = 20;

    // ベストスコア
    x1 = 300;
    x2 = x1 + width;
    DrawBox(x1, y1, x1 + width, y2, 0x000000, true);

    charX = ((x1 + x2) / 2) - 10;
    SetFontSize(32);
    DrawFormatString(
        charX, charY, 0xffffff, "%d", mBestScore);
    SetFontSize(20);
    DrawString(
        x1, y1, "Best", 0x00bfff);
```

```
    // 現在のスコア
    x1 += width + 20;
    x2 = x1 + width;
    DrawBox(x1, y1, x1 + width, y2, 0x000000, true);

    charX = ((x1 + x2) / 2) - 10;
    SetFontSize(32);
    DrawFormatString(
        charX, charY, 0xffffff, "%d", mCntMove);

}

/// <summary>
/// リソース解放
/// </summary>
/// <param name=""></param>
/// <returns></returns>
void GameScene::Release(void)
{

    mStage->Release();
    delete mStage;

    mUnit->Release();
    delete mUnit;

    // 動的配列
    int size = mBoxes.size();
    for (int i = 0; i < size; i++)
    {
        mBoxes[i]->Release();
        delete mBoxes[i];
    }
    // 動的配列のサイズをゼロにしますよ
    mBoxes.clear();


    // 荷物置き場
    size = mStorages.size();
    for (int i = 0; i < size; i++)
    {
        mStorages[i]->Release();
        delete mStorages[i];
    }
    // 動的配列のサイズをゼロにしますよ
    mStorages.clear();
```

```
    DeleteGraph(mImageClear);

    // 操作履歴
    while (mHistoryBack.size() > 0)
    {
        //auto tmp = mHistoryBack.top;
        //tmp.Release();
        //delete tmp;
        mHistoryBack.pop();
    }

}

Stage* GameScene::GetStage(void)
{
    return mStage;
}

Box* GameScene::GetCollsionBox(Vector2 pos)
{

    Box* ret = nullptr;

    Vector2 boxPos;
    int size = mBoxes.size();
    for (int i = 0; i < size; i++)
    {

        boxPos = mBoxes[i]->GetPos();

        // 座標の比較
        if (pos.x == boxPos.x && pos.y == boxPos.y)
        {

            // 座標が一致したらBoxを返り値にする
            ret = mBoxes[i];
            break;

        }

    }

    return ret;

}

Storage* GameScene::GetCollsionStorage(Vector2 pos)
{
```

```
    Storage* ret = nullptr;

    Vector2 storagePos;
    int size = mStorages.size();
    for (int i = 0; i < size; i++)
    {

        storagePos = mStorages[i]->GetPos();

        // 座標の比較
        if (pos.x == storagePos.x && pos.y == storagePos.y)
        {

            // 座標が一致したらStorageを返り値にする
            ret = mStorages[i];
            break;

        }

    }

    return ret;

}

std::string GameScene::GetCsvPathGround(int stageNo)
{

    std::string ret = "";

    ret += FILE_PATH_CSV;
    ret += std::to_string(stageNo);
    ret += "/";
    ret += FILE_NAME_GROUND;

    return ret;

}

std::string GameScene::GetCsvPathGimmick(int stageNo)
{

    std::string ret = "";

    ret += FILE_PATH_CSV;
    ret += std::to_string(stageNo);
    ret += "/";
```

```
    ret += FILE_NAME_GIMMICK;

    return ret;

}

std::string GameScene::GetCsvPathScore(void)
{

    // Bestスコアファイルパスを取得する関数を完成させましょう
    std::string ret = "";

    //ret += "StageData/Score.csv";

    ret += "StageData/";
    ret += FILE_NAME_SCORE;
    return ret;

}

void GameScene::RegistHistory(DIR dir, Vector2 pos, Box* box)
{

    Vector2 boxPos = { 0, 0 };
    if (box != nullptr)
    {
        boxPos = box->GetPos();
    }

    History his = {dir, pos, box, boxPos};
    mHistoryBack.push(his);

}

void GameScene::PlusCntMove(void)
{
    mCntMove += 1;
}

void GameScene::MinusCntMove(void)
{
    mCntMove -= 1;
}

void GameScene::ChangeStage(void)
{
    Release();
    mStageNo += 1;
```

```
    Init();
}

void GameScene::SetStage(void)
{


    // 荷物
    Box* tmpBox;

    // 荷物置き場
    Storage* tmpStorage;

    switch (mStageNo)
    {
    case 1:

        // ユニット
        mUnit = new Unit(this);
        mUnit->Init({ 10, 10 });

        // 荷物
        tmpBox = new Box(this);
        tmpBox->Init({ 12, 10 });
        mBoxes.push_back(tmpBox);

        tmpBox = new Box(this);
        tmpBox->Init({ 17, 10 });
        mBoxes.push_back(tmpBox);

        // 荷物置き場
        tmpStorage = new Storage(this);
        tmpStorage->Init({ 16, 10 });
        mStorages.push_back(tmpStorage);

        tmpStorage = new Storage(this);
        tmpStorage->Init({ 11, 11 });
        mStorages.push_back(tmpStorage);
        break;

    case 2:

        // ユニット
        mUnit = new Unit(this);
        mUnit->Init({ 15, 8 });

        // 荷物
        tmpBox = new Box(this);
```

```
        tmpBox->Init({ 15, 10 });
        mBoxes.push_back(tmpBox);

        tmpBox = new Box(this);
        tmpBox->Init({ 16, 11 });
        mBoxes.push_back(tmpBox);

        tmpBox = new Box(this);
        tmpBox->Init({ 17, 12 });
        mBoxes.push_back(tmpBox);

        // 荷物置き場
        tmpStorage = new Storage(this);
        tmpStorage->Init({ 16, 10 });
        mStorages.push_back(tmpStorage);

        tmpStorage = new Storage(this);
        tmpStorage->Init({ 16, 11 });
        mStorages.push_back(tmpStorage);

        tmpStorage = new Storage(this);
        tmpStorage->Init({ 16, 12 });
        mStorages.push_back(tmpStorage);

        break;
    }

}

void GameScene::LoadGimmickData(void)
{

    // ファイルパスを取得する
    std::string filePath =
        GetCsvPathGimmick(mStageNo);

    // ファイルを読み込む
    std::ifstream ifs(filePath);

    Box* tmpBox;
    Storage* tmpStorage;

    // １行ずつ読み込む
    int y = 0;
    std::string line;
    while (getline(ifs, line))
    {
```

```
        // 分割
        std::vector<std::string> strvec =
            Utility::Split(line, ',');

        // Xで分割されたstrvec
        int size = strvec.size();
        for (int x = 0; x < size; x++)
        {

            int chipNo = stoi(strvec[x]);
            switch (chipNo)
            {
            case 4:
                // 荷物
                tmpBox = new Box(this);
                tmpBox->Init({ x, y });
                mBoxes.push_back(tmpBox);
                break;
            case 5:
                // 荷物置き場
                tmpStorage = new Storage(this);
                tmpStorage->Init({ x, y });
                mStorages.push_back(tmpStorage);
                break;
            case 6:
                // ユニット
                mUnit = new Unit(this);
                mUnit->Init({ x, y });
                break;
            }

        }

        y++;

    }

}

void GameScene::LoadScore(void)
{

    // ファイルパスを取得する
    std::string filePath = GetCsvPathScore();

    // ファイル【全体】を読み込む
    std::ifstream ifs(filePath);
```

```
    // 【全体】を【行】に分ける
    std::string line;
    while (getline(ifs, line))
    {
        // 【行】を【値】に分ける
        std::vector<std::string> strvec =
            Utility::Split(line, ',');

        int stageNo = stoi(strvec[0]);
        int bestScore = stoi(strvec[1]);

        // 【値】を【動的配列】に格納する stoi
        mBestScores.emplace(stageNo, bestScore);

    }

}

void GameScene::SaveScore(void)
{

    // ①動的配列内のBestスコア更新
    //---------------------------------------
    // mapの中のkeyチェック
    if (0 < mBestScores.count(mStageNo))
    {
        // mapの中にkeyがある場合
        mBestScores[mStageNo] = mCntMove;
    }
    else
    {
        // mapの中にkeyがない場合
        mBestScores.emplace(mStageNo, mCntMove);
    }
    //---------------------------------------

    // ②Score.csvへの書き込み
    //   （ファイル全体を書き換える）
    //---------------------------------------
    // ファイルパスを取得する
    std::string filePath = GetCsvPathScore();

    // ファイル【全体】を読み込む
    std::ofstream ofs = std::ofstream(filePath);

    // 【行】ごとにファイルに書き込む
    std::string line;
    for (std::pair<int, int> p : mBestScores)
```

```
    {
        line = "";
        line += std::to_string(p.first);
        line += ",";
        line += std::to_string(p.second);
        ofs << line << std::endl;
    }
    ofs.close();
    //----------------------------------------

}

int GameScene::GetBestScore(void)
{

    int ret = 999;

    // mapの中のkeyチェック
    if (0 < mBestScores.count(mStageNo))
    {
        // mapに対象のキーがある
        std::map<int, int>::iterator it = mBestScores.find(mStageNo);
        ret = it->second;
    }

    // 現在のステージのBestスコア
    return ret;

}

void GameScene::ChangeState(STATE state)
{

    mState = state;
    switch (mState)
    {
    case GameScene::STATE::GAME:
        break;
    case GameScene::STATE::CLEAR:
        mStepClear = 0.0f;
        // Bestスコア更新処理
        if (mCntMove < mBestScore)
        {
            SaveScore();
        }
        break;
    case GameScene::STATE::CHANGE_STAGE:
        // 段々暗くする
```

```
        mFader->SetFade(Fader::FADE_STATE::FADE_OUT);
        break;
    }

}
```