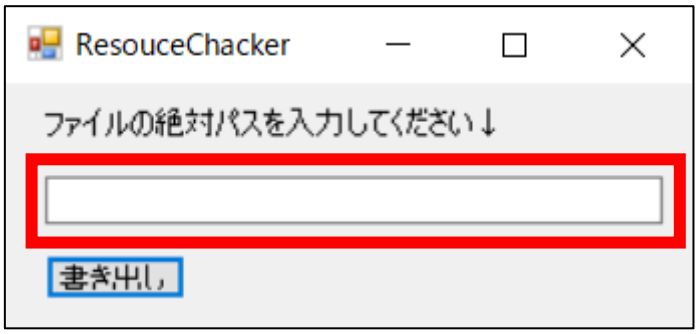
ResouceChacker

リソースマネージャを利用しやすくするツールです。

この資料の一番最後に私のリソースマネージャの使用方法を記しておきます。



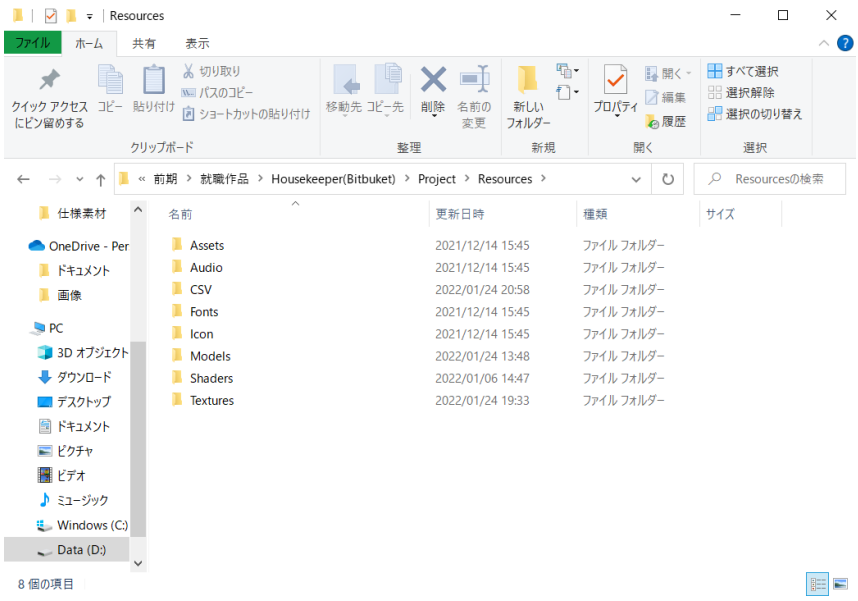
<使い方>

1）赤四角(□)の中にゲームのプロジェクトのリソースまでの絶対パスを入力します。(図１参照)

2）[書き出し]をクリックし、ファイルの保存場所及び名前を指定してファイルを保存します。

3）プロジェクトの中に保存したヘッダーファイルと実行ファイルを追加します。

<図１>

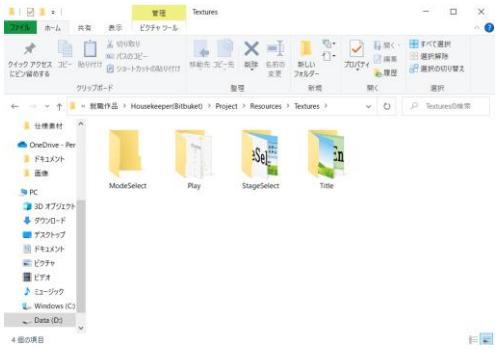


入力例)

D:¥授業¥2 年¥前期¥就職作品¥Housekeeper(Bitbuket)¥Project¥Resources

※Models ディレクトリ、Textures ディレクトリ内に各シーンごとにリソースを分けておくこと！

例)



<保存されるファイルの各種項目>

{ヘッダーファイル}

□・・・リソースを管理する列挙型の名前を管理する列挙型

□…リソースを用いるために必要な情報をもつ構造体

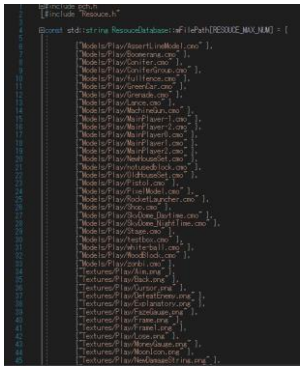
□…各リソースの番号を管理する列挙型

(□の数だけ列挙型があります。)

```
struct ResouceRange
{
    //最小
    int mMin;
    //最大
    int mMax;

};

enum class ReadRange
{
    PLAY_MODELS_RESOUCE,
    PLAY_TEXTURES_RESOUCE,
    STAGESELECT_TEXTURES_RESOUCE,
    TITLE_TEXTURES_RESOUCE,
    NUM,

};

struct ResouceDatabase
{
    //リソース最大数
    static const int RESOUCE_MAX_NUM = 73;
    //ファイルのパスを管理するデータベース変数
    static const std::string mFilePath[RESOUCE_MAX_NUM];
    //列挙型の幅を管理する変数
    static const ResouceRange mEnumClassRange[static_cast<int>(ReadRange::NUM)];
};

enum class PlayModelsResouce
{
    MODEL__ASSERTLINEMODEL = 0,
    MODEL__BOOMERANG,
    MODEL__CONIFER,
    MODEL__CONIFERGROUP,
    MODEL__FULLFENCE,
    MODEL__GREENCAR,
    MODEL__GRENADE,
    MODEL__LANCE,
    MODEL__MACHINEGUN,
    MODEL__MAINPLAYER_1,
    MODEL__MAINPLAYER_2,
    MODEL__MAINPLAYER0,
    MODEL__MAINPLAYER1,
```

{実行ファイル}

主にファイルのパスを管理しています。
触る必要はありません。



この部分は触らないでください→

<私のリソースマネージャ>

・テクスチャとモデルをそれぞれ連想配列で管理しています。
・キーは int 型となっています。ここには追々説明するキーとなる列挙型
(以下、{列挙型 A}とします。)を入れることになります。

```
class ResourceManager
{
public:
    //インスタンスの取得
    static ResourceManager* GetInstance()
    {
        static ResourceManager instance;
        return &instance;
    }
private:

    //変数
    //テクスチャ保持用マップ
    std::unordered_map<int, Microsoft::WRL::ComPtr
                    <ID3D11ShaderResourceView>> mpTextureDB;

    //CMOモデル保持用マップ
    std::unordered_map<int, std::unique_ptr<DirectX::Model>>
        mpCmoModelDB;
```

・Load 関数を呼ぶことによって特定のリソースのみを読み込むことができます。
・引数には追々説明する構造体を管理する列挙型(以下、{列挙型 B}とします。)を入れます。

```cpp
/*=======================================
リソース読み込み
引数：読み込むリソース群の幅
=======================================*/
void ResourceManager::Load(ReadRange range)
{
    int min[ 0 ];
    int max[ 0 ];
    //範囲を指定する
    min = ResouceDatabase::mEnumClassRange[static_cast<int>(range)].mMin;
    max = ResouceDatabase::mEnumClassRange[static_cast<int>(range)].mMax;

    std::string filePath = "Resources/";

    //マルチバイト文字をワイド文字に変換
    wchar_t* wcs = L"none";//変換用変数
    const wchar_t* wideFilePath = L"none";  //テクスチャの相対パス

    //範囲内のみ読み込む
    for (int i = min; i < max+1; i++)
    {
        //ファイル名を設定する
        filePath += ResouceDatabase::mFilePath[i];

        wcs = new wchar_t[filePath.length() + 1];
        mbstowcs(wcs, filePath.c_str(), filePath.length() + 1);
        wideFilePath = wcs;

        //もしCMOの文字がなかったら(テクスチャならば)
        if (filePath.rfind(".cmo") == Utility::EXCEPTION_NUM)
        {
            this->RegisterTexture(wideFilePath,i);
        }
        else
        {
            this->GetCmoModel(wideFilePath, i);
        }
        filePath = "Resources/";
        delete[] wcs;
    }
}
```

<使用例>

{リソースの読み込み}

・Load 関数の引数に■の列挙型内の読み込みたい列挙型名を入れます。

```cpp
    //リソース関係の初期設定
    //リソースマネージャに画像を保存する
    auto pRM = ResourceManager::GetInstance();
    //データ初期化
    pRM->Finalize();
    //リソースの読み込み
    pRM->Load(ReadRange::PLAY_MODELS_RESOUCE);
    pRM->Load(ReadRange::PLAY_TEXTURES_RESOUCE);
```

・こうすることで各種列挙型のリソースを読み込むことができます。

{使用}

Int 型に変換した■の列挙型の要素を入れることで、リソースを取得することができます。

```
108
109     //リソースマネージャの準備
110     auto pRM = ResourceManager::GetInstance();
111
112     //プレイヤー
113     auto palyermodel = pRM->GetCmoModel(static_cast<int>(PlayModelsResouce::MODEL__MAINPLAYER0));
114     mpPlayerModel->SetModel(palyermodel);
115
```

※[参考ファイル]内の[Test]ディレクトリ内の Resouces ディレクトリを用いて試すことができます。