Takasaki taiga

PORTFOLIO

髙﨑　白琥

自己紹介！

ABOUT

名前：髙﨑　白琥(タカサキ　タイガ)
出身：福岡情報ITクリエイター専門学校

メール：sindo12345taiga@gmail.com
Git：https://github.com/Ajakong

ゲームプログラマー志望
趣味　ゲーム/演技

C：2年以上、C++：2年以上、C#/Unity：1年半以上、DX
ライブラリ：2年以上、 GitHub：2年以上、
Photoshop：2年以上、Illustrator：3ヶ月、
Word/Excel/PowerPoint：2年以上、
Premiere Pro：3ヶ月、Maya/Blender：半年

制作作品！

作品名

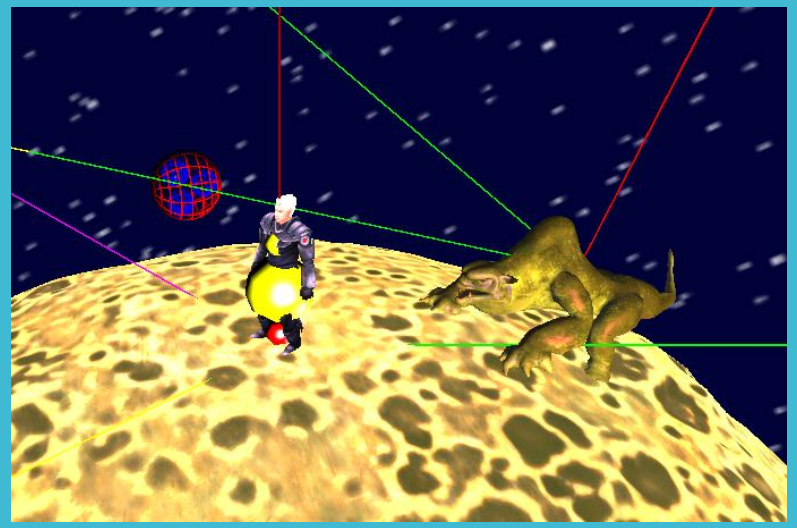
Astro Seeker

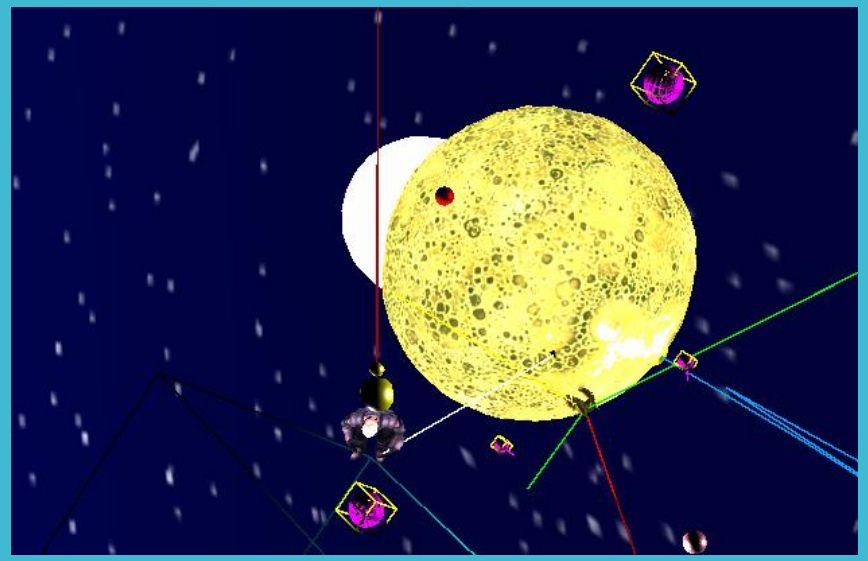
制作環境:C++/DxLib
/GitHub Desktop

ゲームについて

Astro Seekerは宇宙を舞台にした、惑星がフィールドという独特なゲームです。

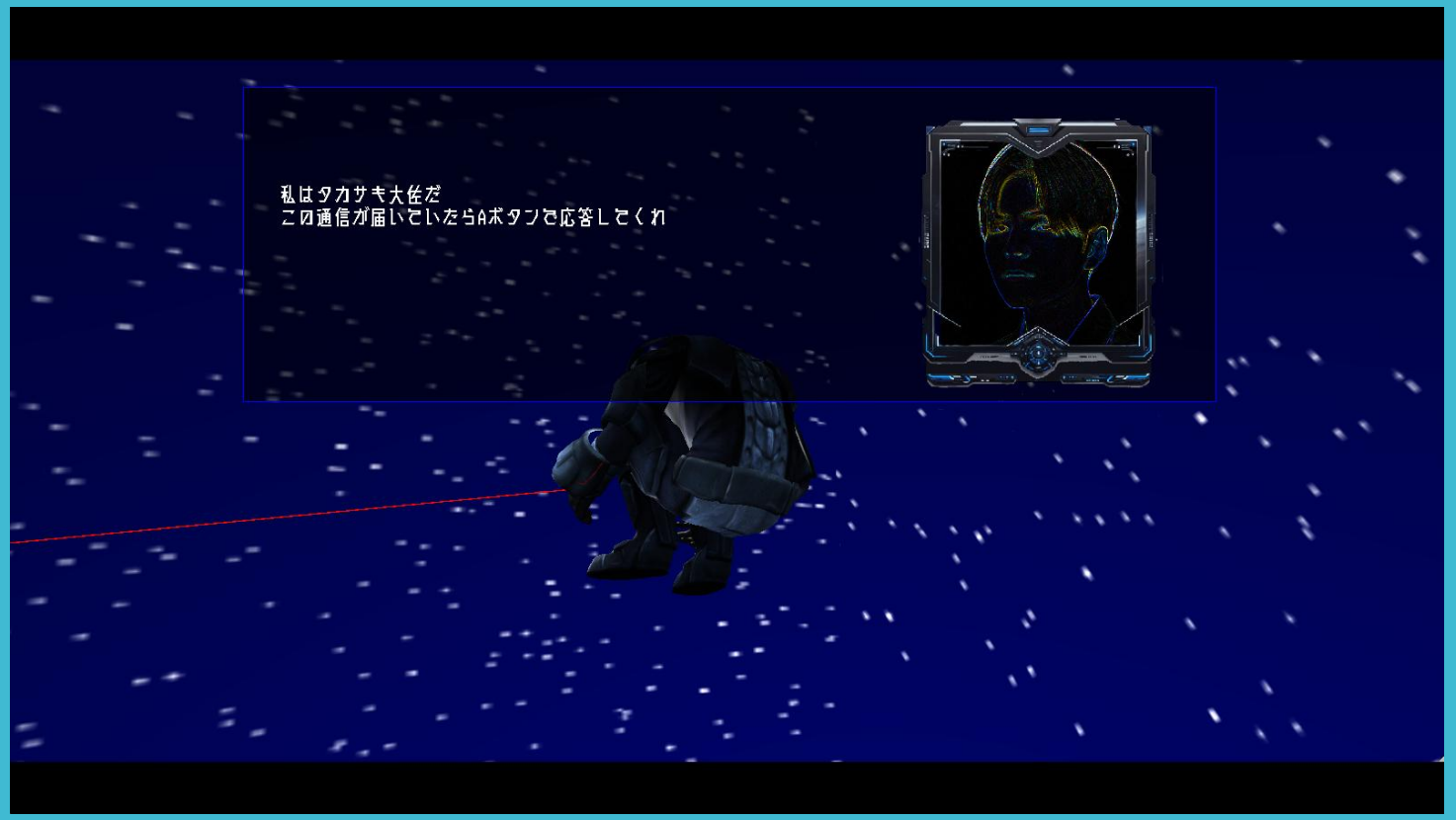
最終目標:強力なアイテムをもって強くなったボスを倒す。





惑星ごとに用意されているギミックを攻略することで、
次の惑星に進めるようになります。

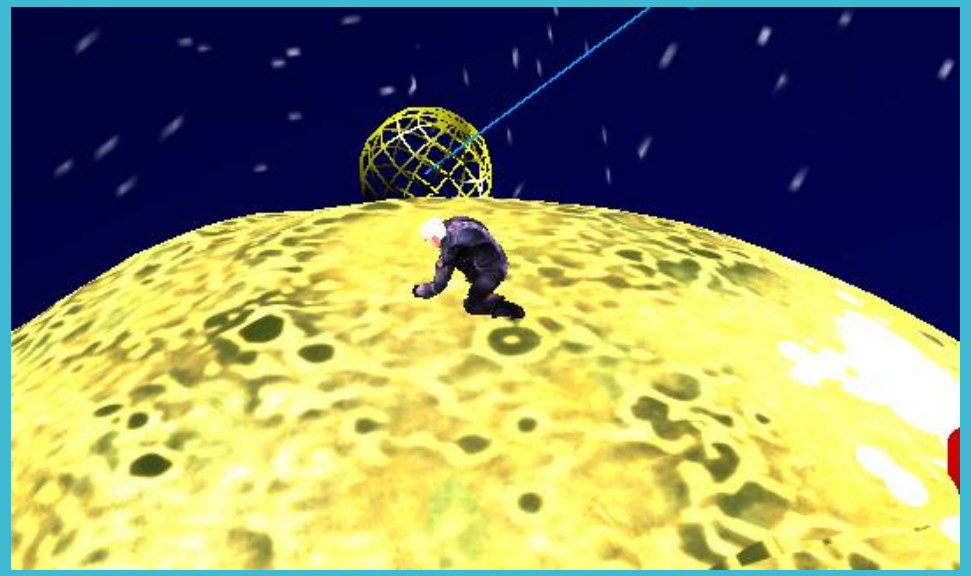
ゲームについて

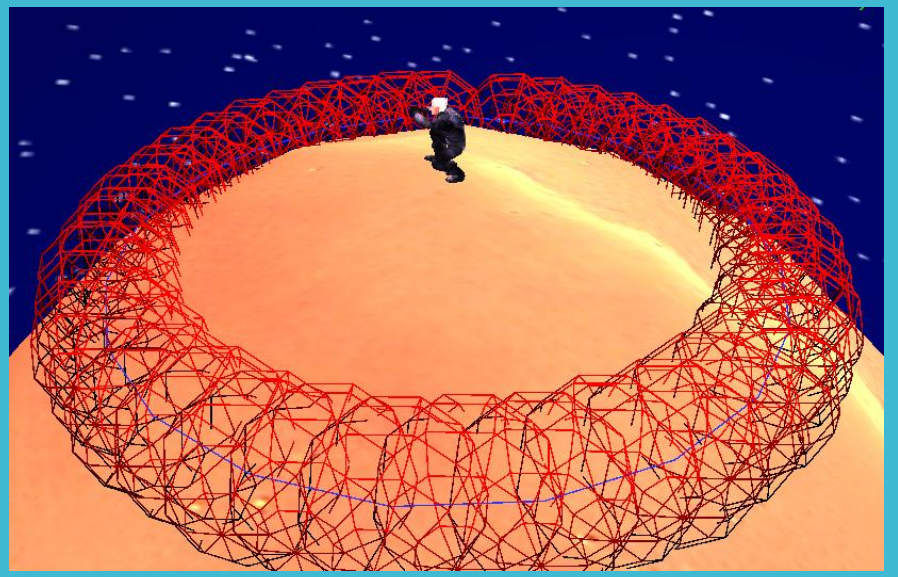


キャラクターと会話をしながら進めていくストーリー

ゲームについて

アイテムを取得することでできることが増える！





コードについて

ステートパターンの実装

```
/// <summary>
/// 弾の種類によって中身を入れ替える
/// </summary>
playerState_t m_shotUpdate;
/// <summary>
/// 現在のジャンプアクションによって中身を入れ替える
/// </summary>
playerState_t m_jumpActionUpdate;
/// <summary>
/// ジャンプ中の落下攻撃
/// </summary>
playerState_t m_dropAttackUpdate;
```

メンバ関数ポインタを利用して、
オブジェクトの状態遷移をしています。

呼び出し

```
void Player::DropAttackUpdate()
{
    //落下攻撃の更新
    (this->*m_dropAttackUpdate)();
}
```

メンバ関数ポインタの中身なる
入れ替える関数たち

```
//ジャンプ中の特殊アクション
/*m_jumpActionUpdateで使う*/
/// <summary>
/// ジャンプ中のアクション統括
/// </summary>
void JumpActionUpdate();
/// <summary>
/// ジャンプ中にスピン
/// </summary>
void JumpingSpinUpdate();
/// <summary>
/// ジャンプ中に加速
/// </summary>
void JumpBoostUpdate();

//ジャンプ中の特殊攻撃
/*m_dropAttackUpdateで使う*/
/// <summary>
/// 落下攻撃統括
/// </summary>
void DropAttackUpdate();
/// <summary>
/// ヒップドロップ攻撃
/// </summary>
void NormalDropAttackUpdate();
/// <summary>
/// 最大火力落下攻撃
/// </summary>
void FullPowerDropAttackUpdate();
```

感じた利点は、ステートごとに関数を分けるので処理の混雑を防げる

コードについて

Unityをツールとして活用したオブジェクト配置

Unity側で出力

```
1 個の参照
private static bool WriteTransformData(BinaryWriter bw)
{
    //現在選択中のオブジェクトを取ってくる
    GameObject topObject = Selection.activeGameObject;
    if (topObject == null)
    {
        EditorUtility.DisplayDialog("", "トップレベルオブジェクトを選択してください", "閉じる");
        return false;
    }
    bw.Write(CalculateLeafCount(topObject));
    WriteRecursiveData(bw, topObject, new Vector3(0, 0, 0), new Vector3(0, 0, 0), new Vector3(1.0f, 1.0f, 1.0f));

    return true;
}

1 個の参照
private static bool OutputData(string fileName)
{
    FileStream fs = File.Create(fileName);
    BinaryWriter binaryWriter = new BinaryWriter(fs);
    if (!WriteTransformData(binaryWriter))
    {
        EditorUtility.DisplayDialog("失敗しました", fileName + "出力に失敗しました", "閉じる");
        return false;
    }
    fs.Close();
    return true;
}
```

C++で読み込み

```
std::string fileName = "Data/Info/data.loc";
//開くファイルのハンドルを取得
int handle = FileRead_open(fileName.c_str());

//読み込むオブジェクト数が何個あるか取得
int dataCnt = 0;
FileRead_read(&dataCnt, sizeof(dataCnt), handle);
//読み込むオブジェクト数分の配列に変更する
m_objectData.resize(dataCnt);

//配列の数分回す
for (auto& loc : m_objectData)
{
    //名前のバイト数を取得する
    byte nameCnt = 0;
    FileRead_read(&nameCnt, sizeof(nameCnt), handle);
    //名前のサイズを変更する
    loc.name.resize(nameCnt);
    //名前を取得する
    FileRead_read(loc.name.data(), sizeof(char) * static_cast<int>(loc.name.size()), handle);

    //タグのバイト数を取得する
    byte tagCnt = 0;
    FileRead_read(&tagCnt, sizeof(tagCnt), handle);
    //タグのサイズを変更する
    loc.tag.resize(tagCnt);
    //タグを取得する
    FileRead_read(loc.tag.data(), sizeof(char) * static_cast<int>(loc.tag.size()), handle);

    //座標を取得する
    FileRead_read(&loc.pos, sizeof(loc.pos), handle);
    //回転を取得する
    FileRead_read(&loc.rot, sizeof(loc.rot), handle);
    //大きさを取得する
    FileRead_read(&loc.scale, sizeof(loc.scale), handle);
```

デバッグをするためにわざわざ配置をする機能を作らなくていい！！