

プロフィール

名前 : 田代 昂汰

出身 : 福岡情報ITクリエイター専門学校

GitHub : <https://github.com/itasi29>

趣味 : ゲーム・ルービックキューブ・釣り

スキル

- C++ : 2年半
- C# : 2年
- HLSL : 1年
- Unity : 2年
- Git Hub : 2年半

顔写真

目次

- 01. プロフィール
- 02. 最終制作作品
- 03. 制作実績
- 04. 今後の目標

最終制作作品

作品名 : THE GATE

ジャンル : 謎解き3Dアクションゲーム

開発環境 : DxLib, C++, HLSL

対応機種 : PC

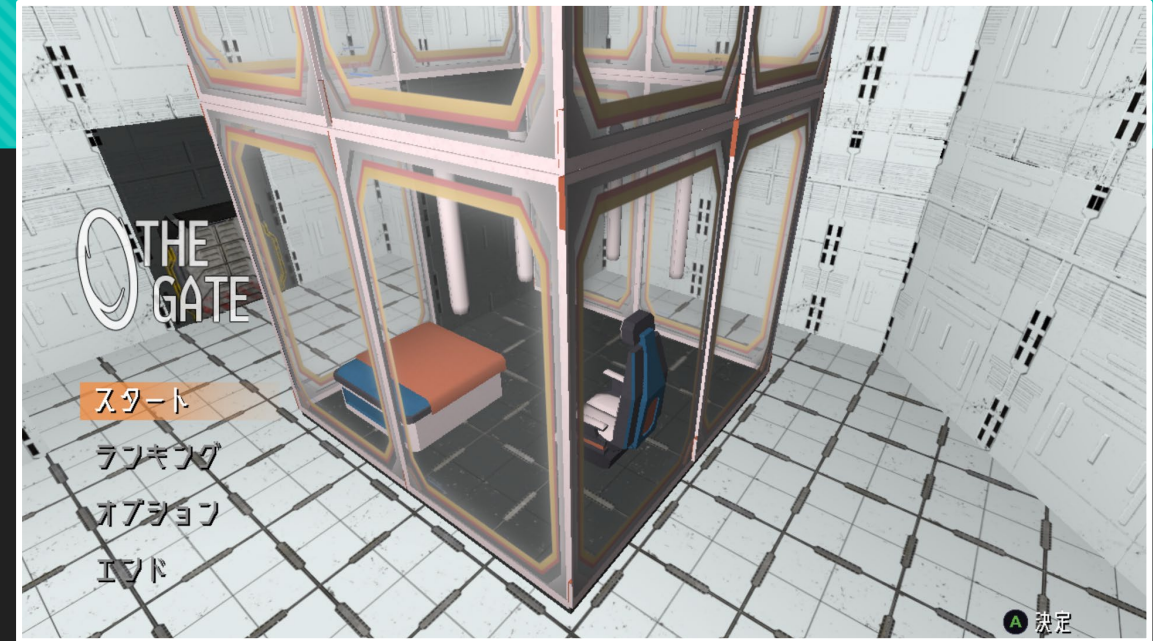
制作期間 : 約4カ月

制作時期 : 2年次10月～2年次2月頭

担当 : モデル・UI・一部ステージ制作以外全て

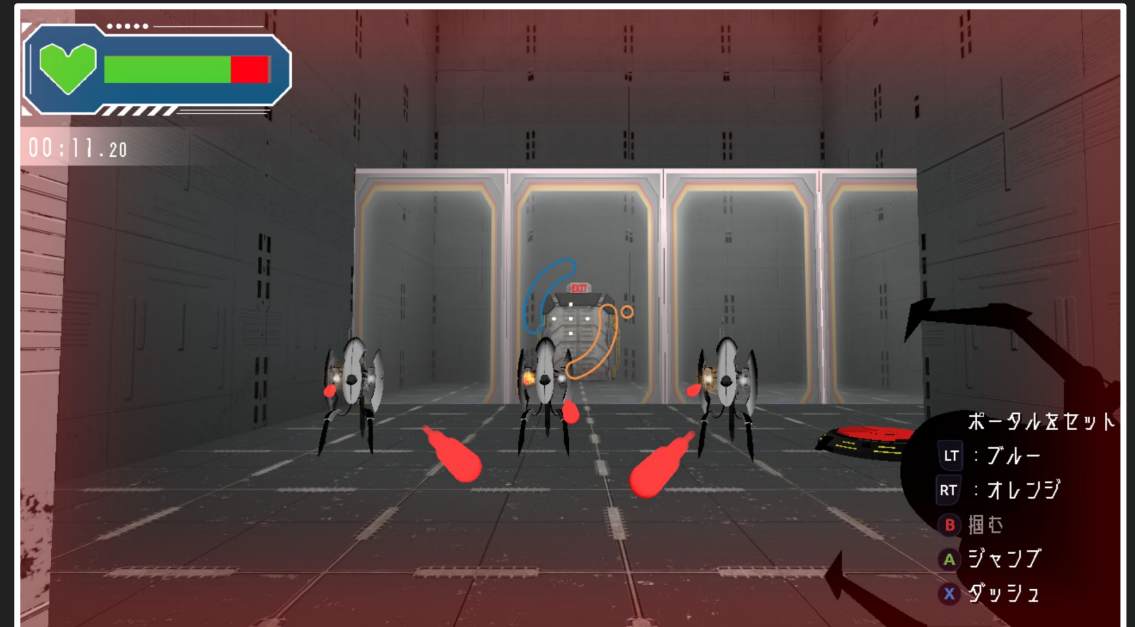
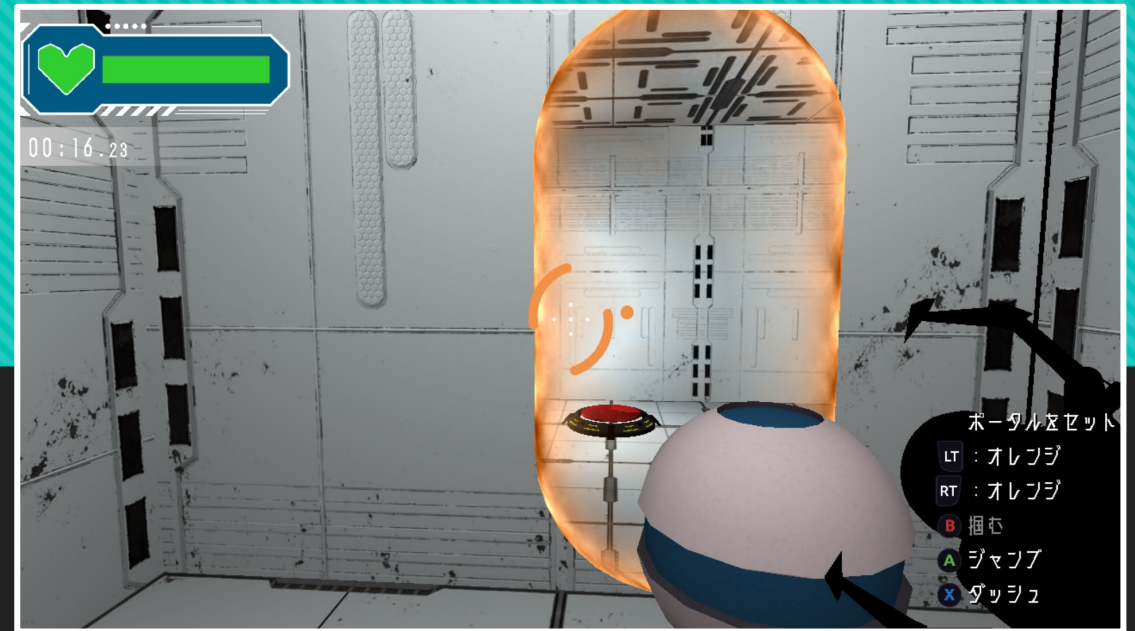
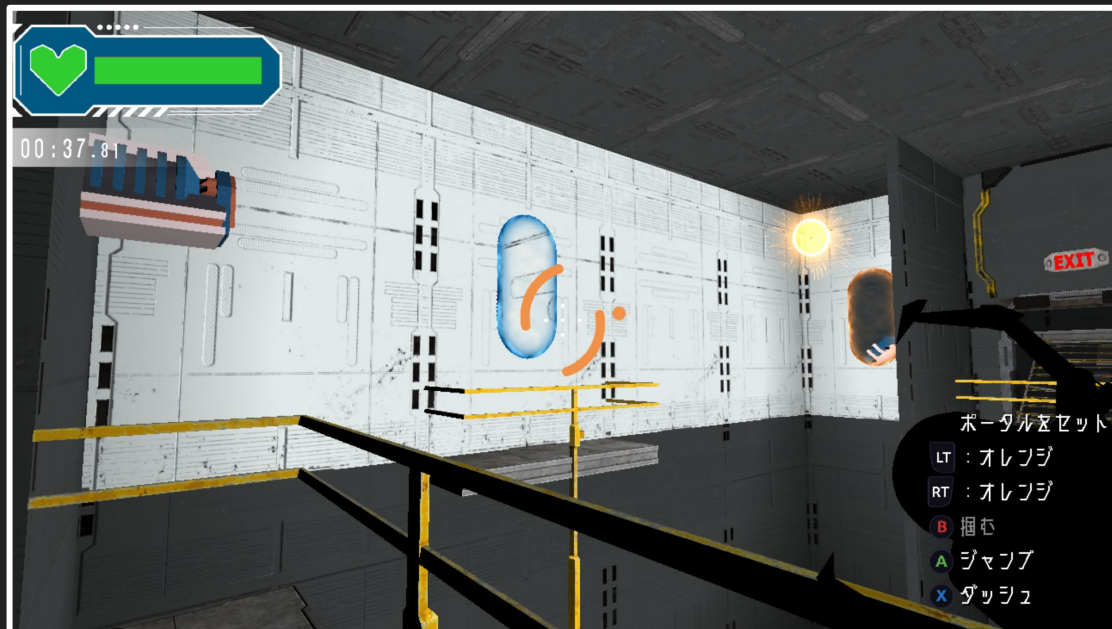
Git : https://github.com/itasi29/THE_GATE.git

動画URL : <https://youtu.be/fZ-xfAzHb9Y?si=qkCO6bSA03Rvn8dl>



内容

ゲートを駆使して
ギミックを攻略せよ！



※実質ポー...t...

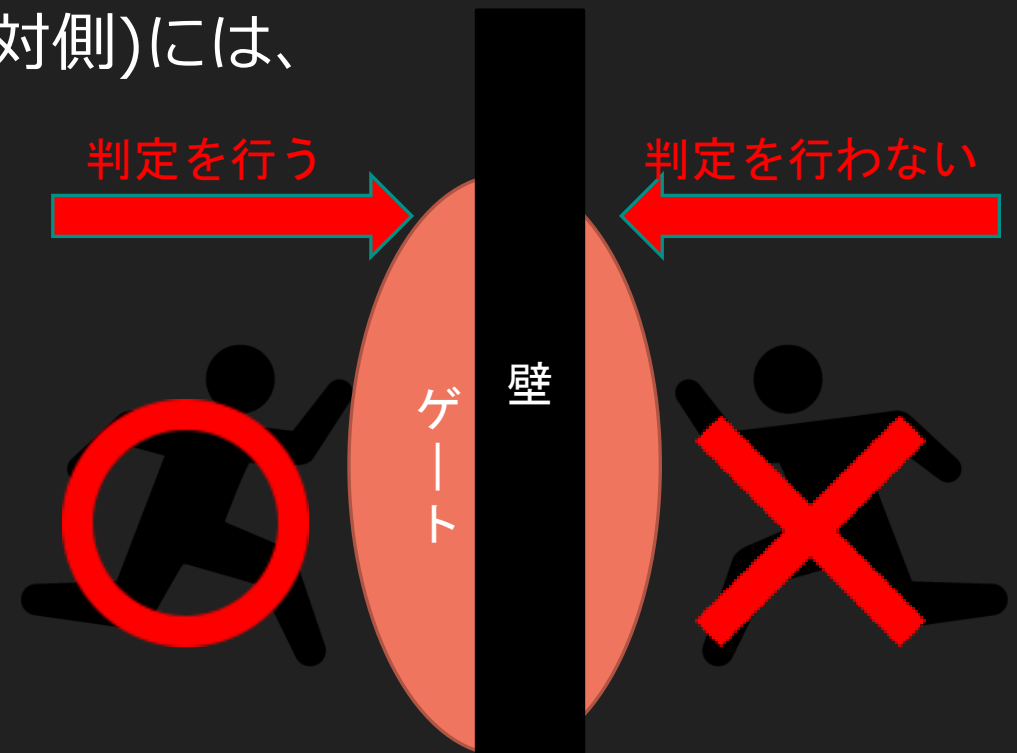
実装：ゲート

Code : Object/Gate/Gate.cpp

ゲートのワープ判定にはプレイヤーまでのベクトルとゲートの法線の内積を用いて判定を実装
また、ゲート判定侵入時から負の場合(反対側)には、判定を行わないようにし処理不可の軽減

```
bool Gate::CheckWarp(const Vec3& targetPos)
{
    // ゲートから対象に向くベクトルとゲートの法線の内積が-になったらワープ可能
    const auto& gateTotarget = targetPos - (m_rigid.GetPos() + m_collider->center);
    auto dot = Vec3::Dot(m_norm, gateTotarget);
    return dot < 0.0f;
}
```

▲判定用コード

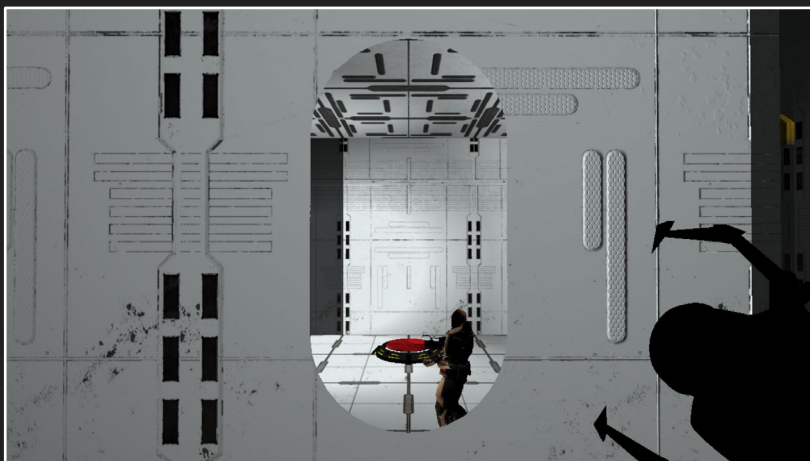


実装：ゲート

Code : Shader/GatePS.hlsl

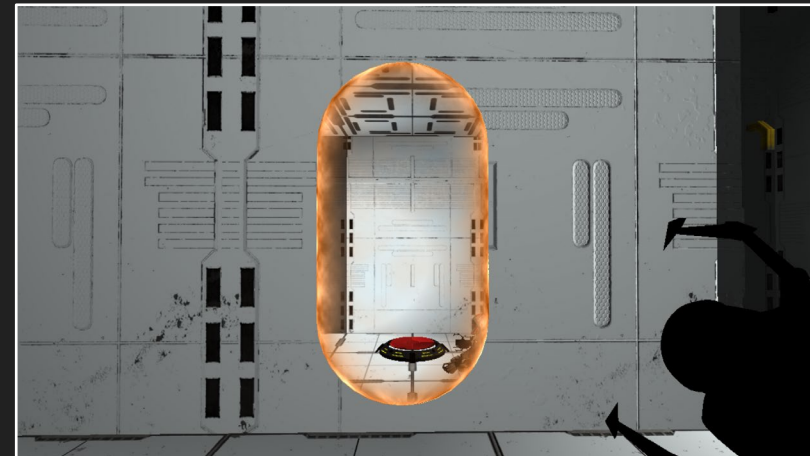
HLSLを用いてゲートの周りの演出を強化

▼何もせずに描画



▲縁を追加

▼Dissolveによる縁のもやもや追加



実装：プレイヤー

Code : Object/Player.cpp, h

プレイヤーはステートパターンを用いて
管理することにより状態の管理、追加を
しやすい構造に

```
// プレイヤーの状態
enum class PlayerState
{
    IDLE,
    WALK,
    RUN,
    JUMP,
    AERIAL,
    LANDING,
    HIT,
    DEATH,
};
```

▲ステート

```
/// <summary>
/// 待機状態に遷移
/// </summary>
void OnIdle();
/// <summary>
/// 歩き状態に遷移
/// </summary>
void OnWalk();
/// <summary>
/// 走り状態に遷移
/// </summary>
void OnRun();
```

▲状態遷移関数

```
void Player::IdleUpdate()
{
    // 1ループしたら
    if (m_anim->IsLoop())
    {
        // アイドルアニメーションをランダムで変更
        auto rand = Random::GetInstance().GetRand(0, ANIM_IDLE_RAND_RATE);
        if (rand < ANIM_IDLE_NORMAL_RATE)
        {
            m_anim->Change(ANIM_IDLE);
        }
        else if (rand < ANIM_IDLE_RELAX_1_RATE + ANIM_IDLE_NORMAL_RATE)
        {
            m_anim->Change(ANIM_RELAX_1);
        }
        else
        {
            m_anim->Change(ANIM_RELAX_2);
        }
    }

    if (!m_camera) return;

    auto& input = Input::GetInstance();
    const auto& trigger = input.GetTriggerData();
    // 左スティックが入力されたら移動状態に遷移
    if (trigger.LStick.SqLength() > 0.0f)
    {
        OnWalk();
        return;
    }

    // ジャンプに遷移
    if (input.IsTriggerd(Command::BTN_JUMP))
    {
        OnJump();
        return;
    }
}
```

▲待機状態での更新関数

実装：自作ライブラリ

Code : Geomtryファイル内

ゲーム内で計算を楽に行えるよう
ベクトル・行列・クォータニオンの
クラスを作成

また、イージングや最近接点を
計算できる関数も作成

```
/// <summary>  
/// 軸を基準に回転させるクォータニオンを作成  
/// </summary>  
/// <param name="angle">回転度合い(度数法)</param>  
/// <param name="axis">軸</param>  
/// <returns>クォータニオン</returns>  
static Quaternion AngleAxis(float angle, const Vec3& axis)  
{  
    Quaternion result;  
  
    float halfRad = angle * Game::DEG_2_RAD * 0.5f;  
    float sin = std::sin(halfRad);  
    float cos = std::cos(halfRad);  
    auto normAxis = axis.GetNormalized();  
    assert(normAxis.Sqlength() > 0.0f && "軸がありません");  
  
    result = Quaternion(normAxis.x * sin, normAxis.y * sin, normAxis.z * sin, cos);  
  
    return result;  
}
```

▲クォータニオンクラス

```
/// <summary>  
/// 射影  
/// </summary>  
/// <param name="projection">射影するベクトル</param>  
/// <param name="dir">射影後の方向となるベクトル</param>  
/// <returns>射影ベクトル</returns>  
static Vec3 Projection(const Vec3& projection, const Vec3& dir)  
{  
    auto dirN = dir.GetNormalized();  
    auto d = Dot(dirN, projection);  
    return dirN * d;  
}
```

▲ベクトルクラス

```
/// <summary>  
/// ベクトルをスケールにした行列を返す  
/// </summary>  
/// <param name="size">スケール</param>  
/// <returns>スケールに値が入った行列</returns>  
static Matrix4x4 Scale(const Vec3& size)  
{  
    Matrix4x4 result;  
    result.Identity();  
  
    result.m[0][0] = size.x;  
    result.m[1][1] = size.y;  
    result.m[2][2] = size.z;  
  
    return result;  
}
```

▲4x4行列クラス

実装：外部ファイル化

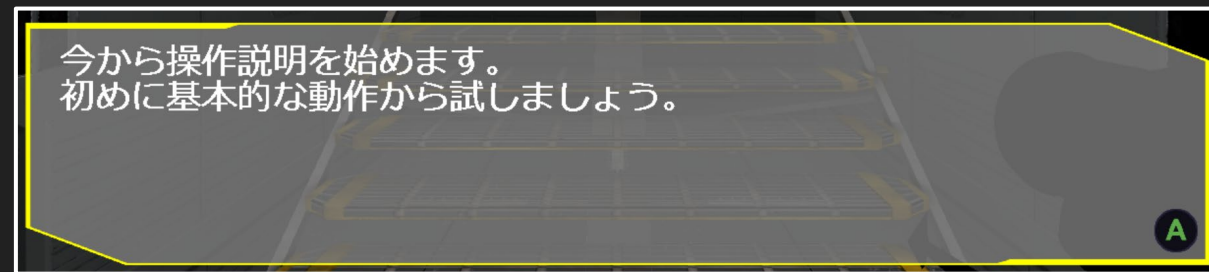
Code：一部マネージャークラス

Data：ExcelMasterファイル内

テキストデータなど変更が多いものをExcelファイルで記入し、
CSVファイル化したものをゲーム内で読み込むことで変更に強く

ID	連続フラグ	連続ID	画像描画フラグ	使用画像ID	文字描画間隔	文字列
N_1_1	1	N_1_2	0		5	今から操作説明を始めます。
N_1_2	0		1	I_EXPO_CAMERA_0	5	初めに視点を動かしてみます。
N_2_1	1	N_2_2	0		5	OK！視点操作は完璧です。
N_2_2	1	N_2_3	1	I_EXPO_CAMERA_1	5	もし、カメラの動きが早い場合は、調整してください。
N_2_3	0		1	I_EXPO_MOVE	5	それでは、次に移動をしてみましょう。
N_3_1	1	N_3_2	0		5	OK！次はダッシュをしてみましょう。
N_3_2	0		1	I_EXPO_DASH	5	移動をしながらXボタンを押すとダッシュができます。
N_4_1	1	N_4_2	0		5	OK！次はジャンプをしてみましょう。
N_4_2	0		1	I_EXPO_JUMP	5	Aボタンを押すことでジャンプができます。

▲Excelファイルにてデータの書き込み

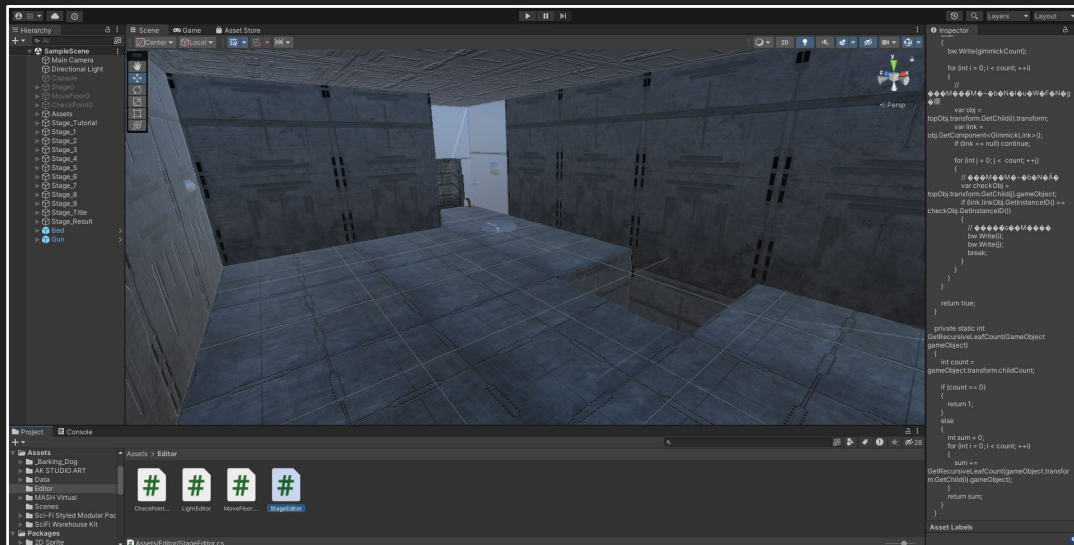


▲実際にゲーム内で描画

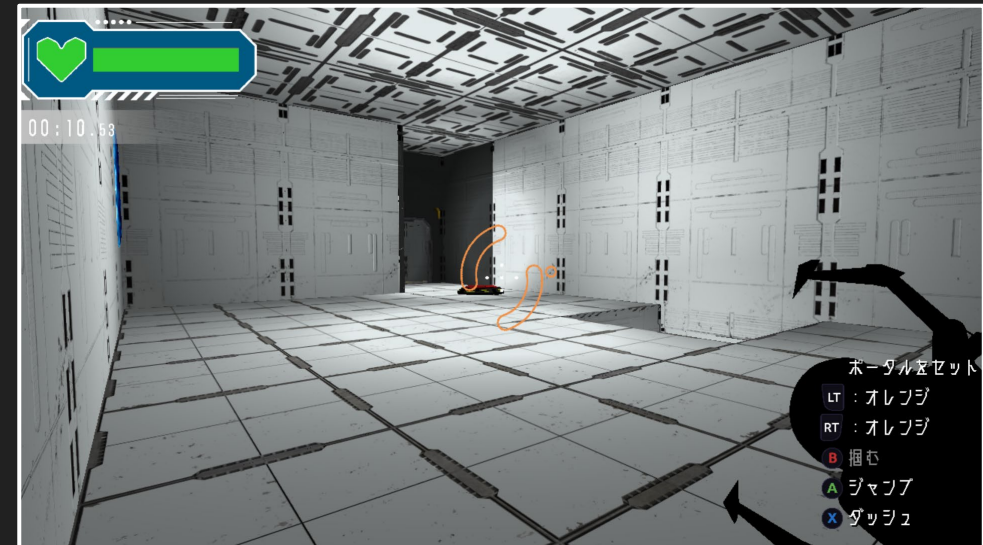
実装：ステージ制作

Code : StageDataManager.cpp

UnityをEditorとして使用
ステージに必要な情報を出力、ゲーム内で読み込み



▲Unityにてステージを作成



▲ゲーム内で読み込む

制作実績① 個人製作

作品名 : サージック!

ジャンル : レースゲーム

制作環境 : DxLib, C++

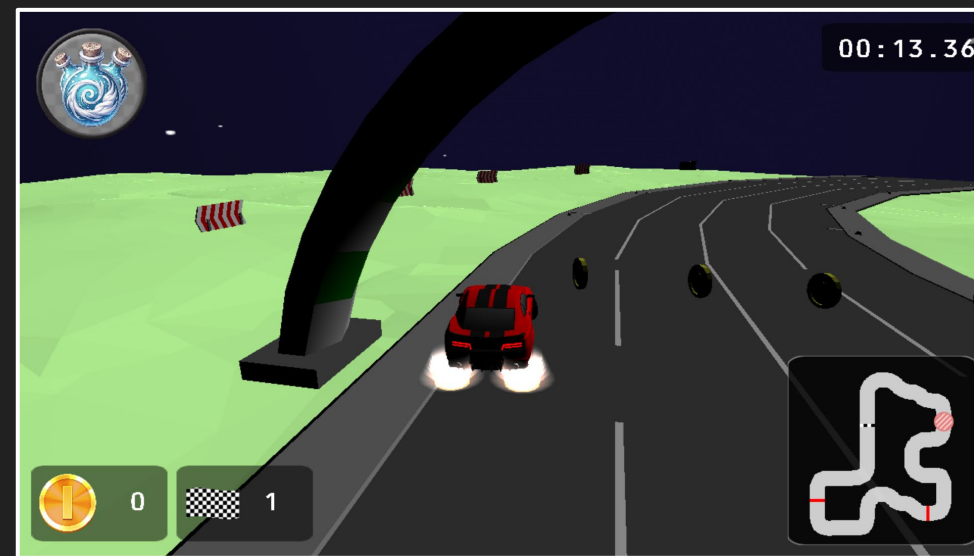
制作期間 : 3カ月

制作時期 : 2024年6月～2024年9月

目的、学んだ事 :

- ・3D空間での当たり判定
- ・簡易的な物理挙動
- ・オブジェクト指向

Git : <https://github.com/itasi29/Cirgic.git>



制作実績② 個人製作

作品名 : CircleRoom

ジャンル : 2Dアクション

制作環境 : DxLib, C++

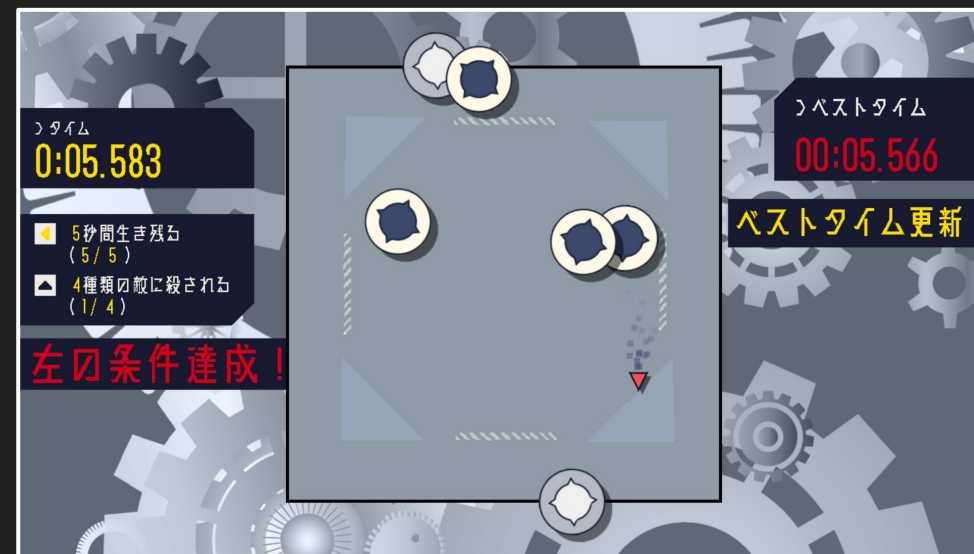
制作期間 : 4カ月

制作時期 : 2023年11月～2024年2月

目的、学んだ事 :

- ・ゲーム制作の流れ
- ・csvファイル使った外部データ化
- ・HLSLを使ったシェーダ

Git : <https://github.com/itasi29/CircleRoom.git>



制作実績③ チーム製作

作品名 : キツネたろう二世の大冒険

ジャンル : 3Dアクション

制作環境 : Unity, C#

制作期間 : 4カ月

制作時期 : 2024年9月～2024年12月

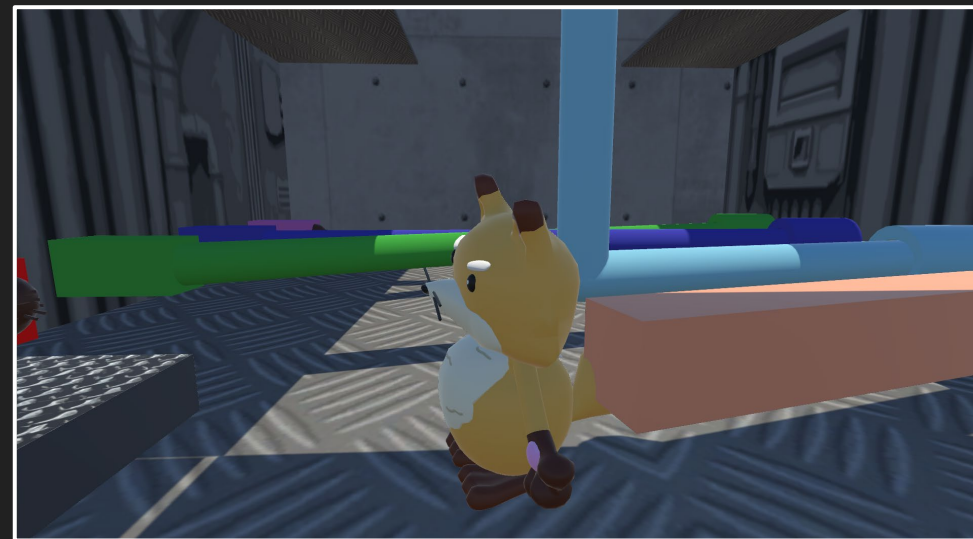
担当 :

- ・2D, 3Dパズルギミック
- ・タイトル、クレジット

目的、学んだ事 :

- ・多職種とのチーム制作
- ・同年代以外とのチーム制作

Git : <https://github.com/itasi29/FoxProject.git>



制作実績④ チーム製作

作品名 : FallingCoin

ジャンル : 2Dアクション

制作環境 : Unity, C#

制作期間 : 3カ月

制作時期 : 2023年6月～2023年8月

担当 :

- ・プレイヤー、カメラ
- ・アイテム

目的、学んだ事 :

- ・ゲーム制作の大変さ
- ・ゲーム制作の基礎

Git : <https://github.com/itasi29/FallingCoin.git>



今後の目標

キャラクター制御に強いプログラマーになりたいと考えています。
そのため、よりゲームらしい挙動をできるよう
数学や物理の知識を学び続けていきます。