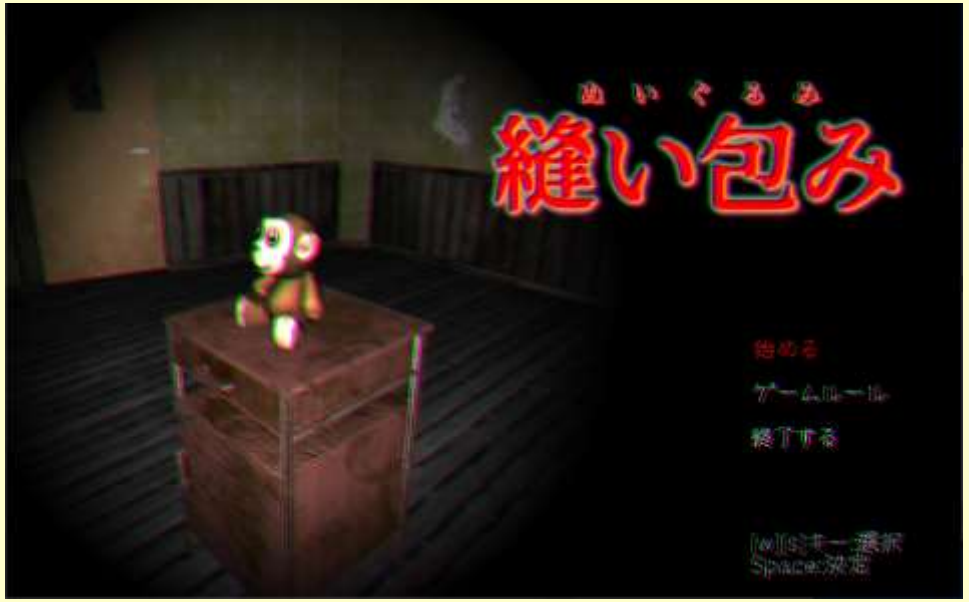
ゲーム概要



タイトル：縫い包み
ジャンル：3D脱出ホラーゲーム

プラットフォーム：PC
制作人数：1人
制作期間：4か月 （2023年10月～2024年1月）
開発環境：DXライブラリ
使用言語：C++、HLSL

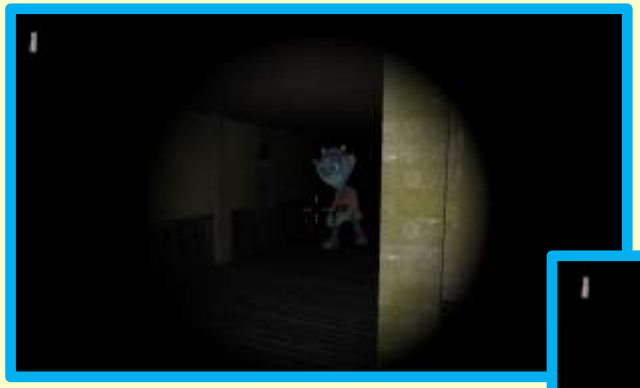
ゲームルール

館を徘徊する化け物から逃げながら、ゴールを開け、脱出を目指す3D脱出ゲームです。

ステージ上にはアイテムとして、護符とぬいぐるみがそれぞれ配置されています。

アイテムの護符を使うと、化け物を怯ませ時間を稼ぐことが出来ます。

ぬいぐるみを全部、焼却炉で燃やすことで、スタート地点のゴールを開くことが出来ます。



◀徘徊する化け物

ステージ上のアイテム▶

◀閉じられているゴール

アピールポイント①　『関数ポインタ』

◇関数ポインタ

```
//Updata関数で実行される関数を指し示すポインタ
std::function<void(void)> updateFunc_;
```

◇更新処理関数

```
//更新処理
void UpdatePatrol(void);                    //巡回状態の更新
void UpdateTracking(void);                  //追跡状態の更新
void UpdateFainting(void);                  //気絶状態の更新
void UpdateLostPlayer(void);                //ロスト状態の更新
void UpdateGameOver(void)override;          //ゲームオーバー状態の更新
```

◇Update関数の中身

```
void Tracker::Update(void)
{
    //現在の更新関数の呼び出し
    updateFunc_();

    //音源位置の設定
    footstepsSound_->SetSoundPos(transform_.pos_);

    //3D情報の更新
    transform_.Update();
    // アニメーション再生
    animController_->Update();
}
```

各Update、Draw関数の処理をstd::function機能を使うことで、管理しやすくしました。

以前まではキャラなどをSTATEパターンで、Switch文を使い、Update関数などを動作していました。

ただあまりにも見づらく冗長的であったため、更新関数を関数ポインタとして管理するようにしました。これによりキャラの状態を操作しやすくなりました。

アピールポイント①　『関数ポインタ』

◇以前までのやり方

```
void Player::Update(void)
{
    //各状態に応じたUpdate関数の呼び出し
    switch (mState)
    {
    case ACTOR_STATE::IDLE:
    {
        IdleUpdate();
        break;
    }
    case ACTOR_STATE::RUN:
    {
        RunUpdate();
        break;
    }
    case ACTOR_STATE::DEAD:
    {
        DeadUpdate();
        break;
    }
    case ACTOR_STATE::MAGIC:
    {
        MagicUpdate();
        break;
```

◇ std::functionを使用したやり方

```
void Tracker::Update(void)
{
    //現在の更新関数の呼び出し
    updateFunc_();

    //音源位置の設定
    footstepsSound_->SetSoundPos(transform_.pos_);

    //3D情報の更新
    transform_.Update();
    // アニメーション再生
    animController_->Update();
}
```

冗長なコードでしたが、 std::function機能を使用することで、各々の更新処理と共通処理のみとなり、コードの読みやすさが向上しました。

アピールポイント②　『HLSLシェーダー』

◇3Dモデル描画用のピクセルシェーダー一部

```
    //ライトごとにライティング処理
    for (int i = 0; i < DX_D3D11_COMMON_CONST_LIGHT_NUM; i++)
    {
        ProcessLight(input.viewPos, ray, normal, common.light[i],
        common.material.power, totalDiffuse, totalSpecular);
    }

    //環境光を適用
    totalDiffuse += common.material.ambientEmissive.rgb;

    //明度にマテリアルの色を適用
    totalDiffuse *= common.material.diffuse.rgb;
    totalSpecular *= common.material.specular.rgb;

    //元の色にマテリアルを適用
    result.rgb *= totalDiffuse;
    result.rgb += totalSpecular;
    result.a *= common.material.diffuse.a * base.factorColor.a;

    //加算色を適用
    result.rgb += base.drawAddColor.rgb;

    //フォグカラー
    float fog = common.fog.color;
    //フォグを適用
    result.rgb *= input.fogFactor;

    //0~1に揃えて返す
    return saturate(result);
}
```

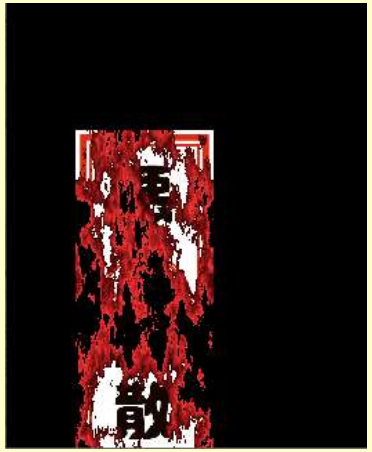
3Dでのホラー感を演出するためにも様々な演出をHLSLシェーダーを使用して作成しました。

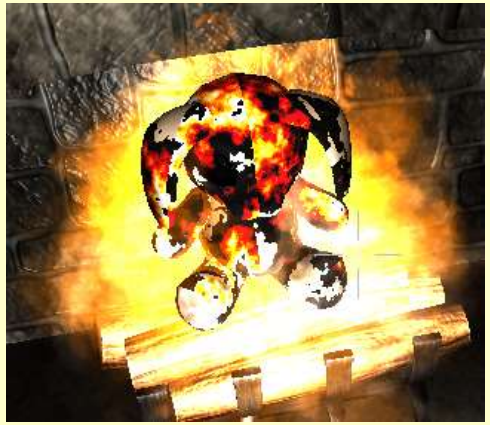
- 頂点シェーダーでの3Dモデル描画
- フォグ
- スフィアマップ
- ディゾルブ
- ポストエフェクト　　etc…

その中でも、リアリティあるグラフィックを表現するためのシェーダー作りに尽力しました。

アピールポイント②　『HLSLシェーダー』

◇シェーダー：ディゾルブ





作成したシェーダー集動画:

URL：https://youtu.be/FE67rZRyq7c

◇リアリティを追求したシェーダー

◀法線マップ

スフィアマップ▶

シェーダー適応前と適応後の差分動画

URL：https://youtu.be/fZ4wzwYG5bw