松田 永久

2004 06/08

所有スキル

C 1年

C++ 1年

Unity 半年

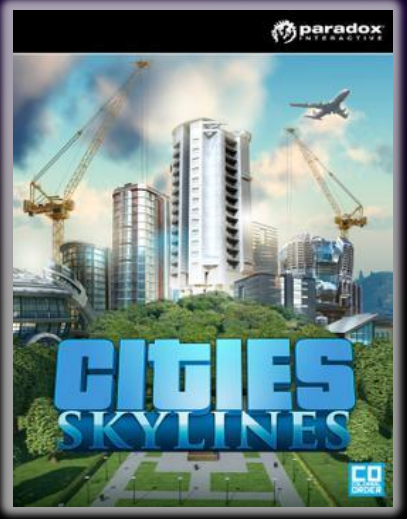
UNREAL ENGINE 半年

Microsoft DirectX 2年

好きなゲーム

FPS

サッカー



シミュレーション

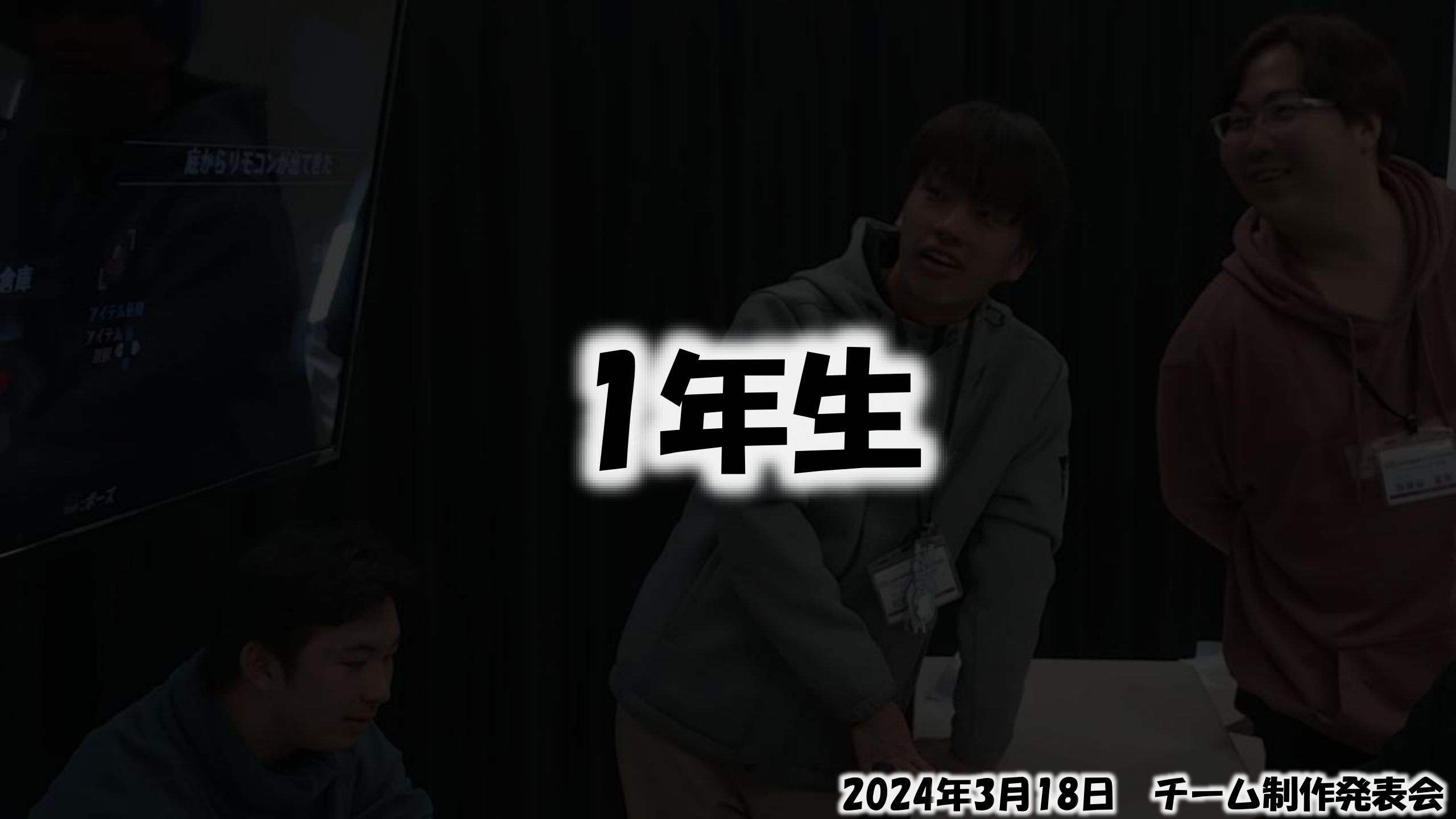
趣味

サッカー観戦

Vtuber視聴

にじさんじ

Vぴいすぽっ!

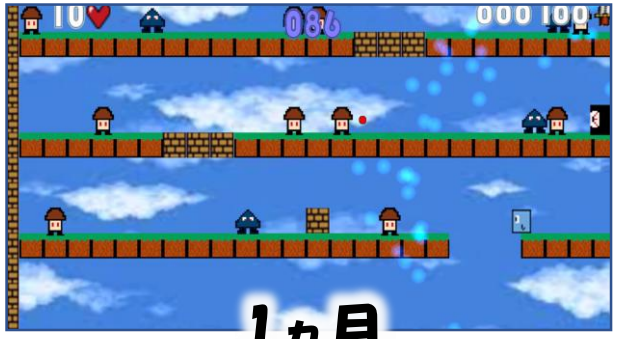


2024年3月18日　チーム制作発表会

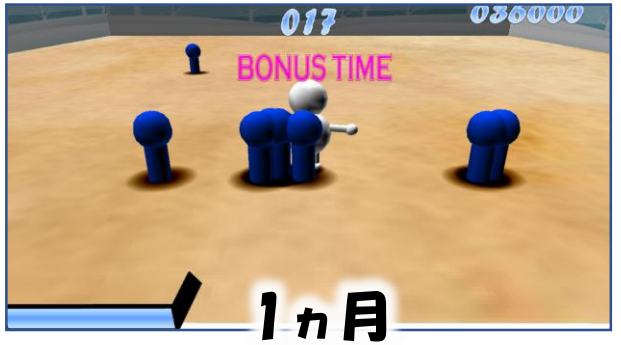
1年次に作成したゲーム

個人

3ヵ月



1ヵ月

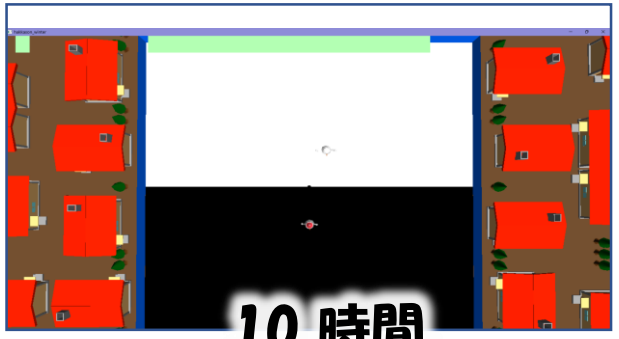


1ヵ月

チーム

3ヵ月

10 時間

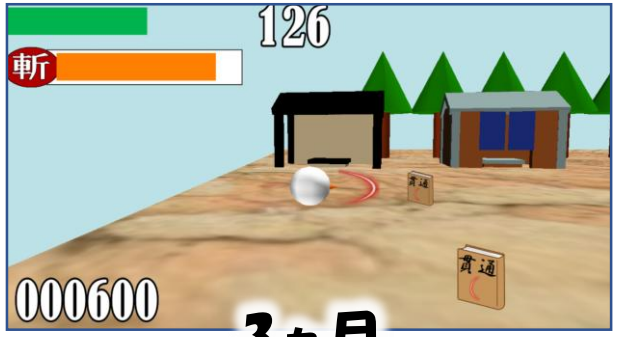


10 時間


2年生
2024年10月13日　SAPPRO GAME CAMP

2年次に作成したゲーム

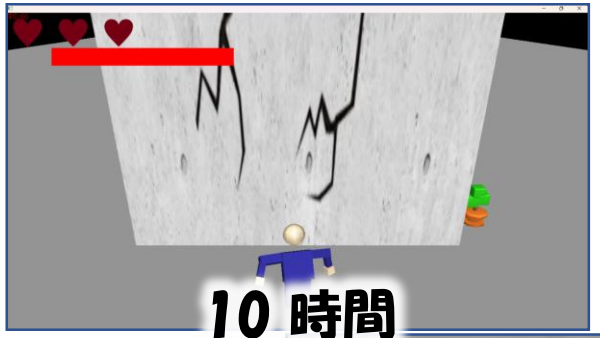
個人

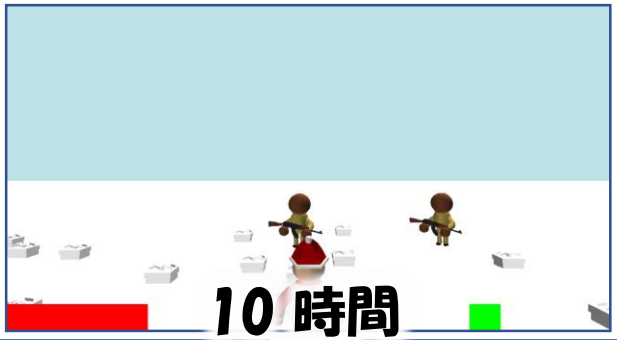


3ヵ月

4ヵ月

チーム


10 時間


10 時間

9 時間

10 時間

2 日間 ゲーム内の画像が見つかりませんでした

SHUTTER FLUX

制作期間:2024年10月2日～2025年2月7日
開発環境:VisualStudio2019
使用言語:C++14　　担当箇所:企画から制作まで
2年次個人2作品目

狙い撃ち！敵を欺け！

スモークや壁を活用して敵の裏をつけ！

混乱している間に、敵を一方的に**攻撃**だ！
敵にダメージを当てると一定時間、**壁越し**に位置を確認できるぞ！

ウルトやスキルで敵を翻弄しよう！

スキルの**ブリンク**を利用して敵の攻撃を回避！
ウルトを敵に当ててて**スタン**させよう！

技術アピール

ステートパターンによる行動の切り替え

状態によってクラスを分け、それぞれの行動を
切り替えやすくしました。キャラクターやカメラで使用しています。

```cpp
//==========================================
//カメラステートクラス
//==========================================
class CCameraState
{
public:
    virtual void FreeView(CCamera* camera);
    virtual void ThirdView(CCamera* camera);
    virtual void Ult(CCamera* camera);
};
```

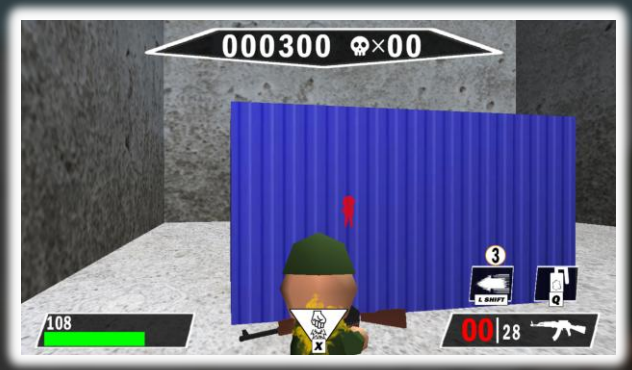
```cpp
//==========================================
//自由なカメラ
//==========================================
class CFreeView : public CCameraState
{
public:
    virtual void FreeView(CCamera* camera)override;
private:
    static constexpr float FREEVIEW_LENGTH = 200.0f; //自由視点時の距離
};

//==========================================
//三人称状態
//==========================================
class CThirdView : public CCameraState
{
public:
    virtual void ThirdView(CCamera* camera)override;
private:
    static constexpr float THIRDVIEW_LENGTH = 130.0f; //サードパーソンビュー時の距離
};

//==========================================
//ウルト状態
//==========================================
class CUltCameraState : public CCameraState
{
public:
    virtual void Ult(CCamera* camera) override;
private:
    static constexpr float ULT_LENGTH = 200.0f; //ウルト時の距離
};
```

```cpp
//==========================================
//三人称の状態
//==========================================
void CThirdView::ThirdView(CCamera* camera)
{
    camera->SetLength(THIRDVIEW_LENGTH);

    camera->ThirdViewCamera();

    //キーボード情報取得
    CInputKeyboard* pKeyboard = CManager::GetInstance()->GetKeyboard();

#ifdef _DEBUG
    if (pKeyboard->GetTrigger(DIK_B))
    {
        camera->ChangeCameraState(new CFreeView);
    }
#endif // _DEBUG

}

//==========================================
//ウルトの状態
//==========================================
void CUltCameraState::Ult(CCamera* camera)
{
    camera->SetLength(ULT_LENGTH);
    camera->ThirdViewCamera();
}
```

技術アピール

ステンシルバッファによる戦術

弾を当ててから一定時間、
壁越しに赤くくりぬき描画をするようにしています。




```
//=============================================
// 描画
//=============================================
void CMask::Draw()
{
    LPDIRECT3DDEVICE9 pDevice = CManager::GetInstance()->GetRenderer()->GetDevice();    // デバイスのポインタ
    // ステンシルテストを有効にする
    pDevice->SetRenderState(D3DRS_STENCILENABLE, TRUE);
    // 比較参照値を設定する
    pDevice->SetRenderState(D3DRS_STENCILREF, REFARENCE_VALUE);
    // ステンシルマスクを指定する
    pDevice->SetRenderState(D3DRS_STENCILMASK, 255);
    // ステンシル比較関数を指定する
    pDevice->SetRenderState(D3DRS_STENCILFUNC, D3DCMP_EQUAL);
    // ステンシル結果に対しての反映設定
    pDevice->SetRenderState(D3DRS_STENCILPASS, D3DSTENCILOP_KEEP);  // Zテスト・ステンシルテスト成功
    pDevice->SetRenderState(D3DRS_STENCILFAIL, D3DSTENCILOP_KEEP);      // Zテスト・ステンシルテスト失敗
    pDevice->SetRenderState(D3DRS_STENCILZFAIL, D3DSTENCILOP_KEEP);     // Zテスト失敗・ステンシルテスト成功

    CObject2D::Draw();

    // ステンシルテストを無効にする
    pDevice->SetRenderState(D3DRS_STENCILENABLE, FALSE);
}
```

技術アピール

Binファイルでの入出力

ブロックは外部エディタで出力、ゲーム内で読み込み
スコアはウェーブごとにスコアを出力しリザルトで読み込んで
最終スコアを計算しています。

```
//=============================================
//ブロックをロード
//=============================================
void CWave::LoadBlock(const std::string& pFileName)
{
    //ファイルの読み込み
    std::ifstream File(pFileName, std::ios::binary);

    //ファイルが開かなかったら関数を抜ける
    if (!File.is_open())
    {
        return;
    }

    //生成するブロック数読み込み用
    int NumBlock = 0;

    File.read(reinterpret_cast<char*>(&NumBlock), sizeof(int));

    if (NumBlock > 0)
    {
        //生成するブロックの情報を持つ変数
        std::vector<LOAD_BLOCK> info(NumBlock);

        //ファイルから読み込んだデータを格納
        File.read(reinterpret_cast<char*>(info.data()), sizeof(LOAD_BLOCK) * NumBlock);

        //イテレータで回して生成
        for (auto& itr : info)
        {
            CBlock::Create(itr.type, itr.pos, itr.rot);
        }
    }

    File.close();
}
```

```
//=============================================
//スコア書き出し
//=============================================
void CScore::ExportScore(const std::string& FileName)
{

    std::ofstream File(FileName,std::ios::binary);

    if (!File.is_open())
    {
        return;
    }

    File.write(reinterpret_cast<const char*>(&m_nScore), sizeof(int));
    File.close();

}
```