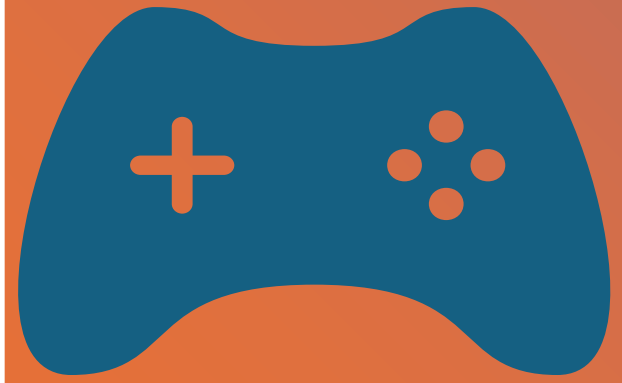
作品詳細



HAL名古屋
ゲーム4年制学科
ゲーム制作専攻
大﨑遥喜

オンライン対戦ゲーム

＜作品概要＞

Winsockを使い同じLAN内での
1対1で剣で斬りあって戦う
通信対戦ゲームを作成しました。

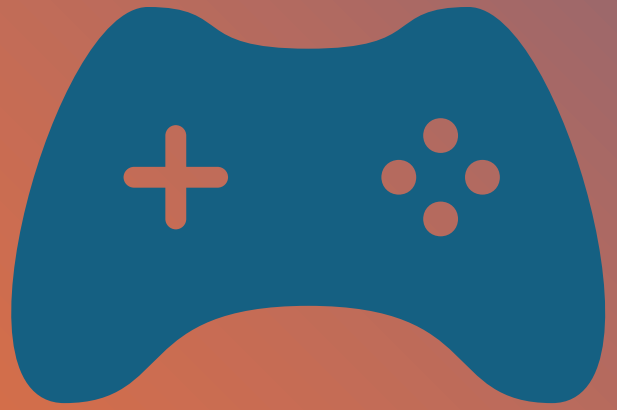
＜開発情報＞

開発環境 : DirectX11 / Winsock

使用言語 : C++ / HLSL

開発形態 : 個人製作

開発期間 : 2023年10月～
2024年1月



同じプログラムで二つ起動するとつながるようになっております。
GUIDを固定値で設定し、ブロードキャストをして
同じプログラムが動いているかを探します。

```
    MSGDATA     SendMessage;        // ユーザ列挙要求送信用
    int         nStatus;            // socket関数戻値格納用
    sockaddr_in toAddr;             // 送信先アドレス
    SendMessage.Msg.Header.nType = MSGTYPE_MACH_REQUEST;
    SendMessage.Msg.Header.nSeqNo = g_nNextRequestNo;
    memcpy(&SendMessage.Msg.Body0.guid, &g_MyGuid, sizeof(g_MyGuid));

    toAddr.sin_addr.S_un.S_addr = inet_addr("255.255.255.255");

    toAddr.sin_family = AF_INET;
    toAddr.sin_port = htons(PORT_NO);
    // 送信
    sendto(g_sockNo, SendMessage.szData, sizeof(SendMessage.szData), 0, (sockaddr *)&toAddr, sizeof(toAddr));
```

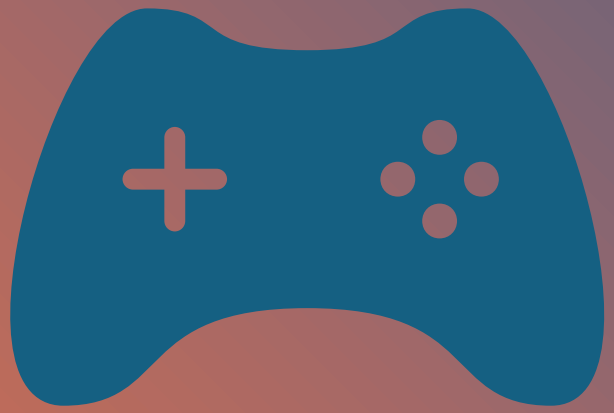
```
// データ受信
nStatus = recvfrom(g_sockNo,        // ソケット番号
    (char *)g_recvBuffer.szData,    // 受信バッファ
    sizeof(g_recvBuffer),           // 受信バッファバイト長
    0,
    (sockaddr*)&fromAddr,           // 送信元アドレス
    &nLen);                         // 第5引数のバイト長をセット

g_IpAdoless = fromAddr.sin_addr.S_un.S_addr;

// ソケットエラーでなければ処理
if (nStatus != SOCKET_ERROR)
{
    // 列挙要求の受信時の処理をプレイヤー情報に変更
    switch (g_recvBuffer.Msg.Header.nType) {
    case MSGTYPE_MACH_REQUEST:// マッチング要求を受け取った時（マッチング前）
        if (IsEqualGUID(g_recvBuffer.Msg.Body0.guid, g_MyGuid)) {
            SendMessage.Msg.Header.nType = MSGTYPE_MACH_RESPONSE;
            //strcpy(SendMessage.szData, "test");

            g_enemyAddr = fromAddr; // 相手のを入れる

            //  相手に送り返す
            sendto(g_sockNo, SendMessage.szData, sizeof(SendMessage.szData), 0, (sockaddr *)&fromAddr, sizeof(fromAddr));
        }
        break;
    case MSGTYPE_MACH_RESPONSE://マッチング要求への応答を受け取った時（マッチング後）
        g_enemyAddr = fromAddr;
        g_SetMach = true;
        break;
    case MSGTYPE_PLAYER_STATE:  // 相手のプレイヤー情報
        // 列挙応答を受信した際の処理をプレイヤー情報に変更する
        // g_ClientList にプレイヤー情報を追加または更新する処理を記述する
        g_ClientList[g_nListCnt].fromAddr = fromAddr;
        // ポジション
        DirectX::XMFLOAT3 pos2;
        pos2.x = g_recvBuffer.Msg.Body1.Position.x;
        pos2.y = g_recvBuffer.Msg.Body1.Position.y;
        pos2.z = g_recvBuffer.Msg.Body1.Position.z;
        // 向き
        DirectX::XMFLOAT3 rote2;
        rote2.x = g_recvBuffer.Msg.Body1.Rotation.x;
        rote2.y = g_recvBuffer.Msg.Body1.Rotation.y;
        rote2.z = g_recvBuffer.Msg.Body1.Rotation.z;
```

受信側ではブロードキャストされた
同じGUIDで同じプログラムだった場合
相手のIPアドレスを格納し
そこにデータを送り返します。

うまく送り返せた場合現在の処理を抜けます。

処理を抜けれた場合マッチング成功のため
マッチ後の処理を行います。

受信側は送られてきたデータを格納し
必要なところで呼び出せるように処理をしています。



```
MSGDATA     SendMessage;        // ユーザ列挙要求送信用
int         nStatus;            // socket関数戻値格納用
sockaddr_in toAddr;             // 送信先アドレス
SendMessage.Msg.Header.nType = MSGTYPE_PLAYER_STATE;
SendMessage.Msg.Header.nSeqNo = g_nNextRequestNo;
//memcpy(&SendMessage.Msg.Body1, &g_MyGuid, sizeof(g_MyGuid));
// ポジション
auto pos = UDP::get_instance().GetPlayerPos();
SendMessage.Msg.Body1.Position.x = pos.x;
SendMessage.Msg.Body1.Position.y = pos.y;
SendMessage.Msg.Body1.Position.z = pos.z;
// 向き
auto rote = UDP::get_instance().GetPlayerRate();
SendMessage.Msg.Body1.Rotation.x = rote.x;
SendMessage.Msg.Body1.Rotation.y = rote.y;
SendMessage.Msg.Body1.Rotation.z = rote.z;
// アタック
auto Attack = UDP::get_instance().GetAttack();
SendMessage.Msg.Body1.Attack = Attack;
// 当たったかどうか
auto Hit = UDP::get_instance().GetHit();
SendMessage.Msg.Body1.Hit = Hit;
if (Attack == true) {
    Hit = false;
}
// HP
auto HP = UDP::get_instance().GetHP();
SendMessage.Msg.Body1.HP = HP;

// 送信先の設定
toAddr.sin_addr.S_un.S_addr = g_enemyAddr.sin_addr.S_un.S_addr;

toAddr.sin_family = AF_INET;
toAddr.sin_port = htons(PORT_NO);


// シーケンス番号の更新
// 列挙数のクリア
g_nListCnt = 0;
// 現在の番号を更新
g_nRequestNo = g_nNextRequestNo;
// 次使う番号を更新
g_nNextRequestNo++;

// 送信処理
sendto(g_sockNo, SendMessage.szData, sizeof(SendMessage.szData), 0, (sockaddr *)&toAddr, sizeof(toAddr));
```

送信側では、相手がデータを受け取り
IPアドレスを格納します。
そのIPアドレスに向かって送り返します。

これができれば通信ができているため
次の処理を行います。

相手のIPアドレスに向かってデータを送ります。
この際、プレイヤーのモデルや、
エフェクトなどの情報をを送ってしまうと
ネットワークが重くなってしまうため、
必要最小限のポジションや向きなどのデータを
送ることでできるだけゲームが
軽く動くように工夫しました。