같은그림찾기 게임 및 채팅 프로그램

학 과	전자공학부 전자공학전공
학 번	2014140043 2017142026
이 름	서동준, 이예찬

목 차

6.

1. 개요

1.1 개발 배경 및 목적

게임을 하면서 단순 1:1 플레이가 아닌 다중 사용자 접속으로 게임을 플레이한다.

다중 사용자가 게임을 진행하다 보니 의사소통의 부재가 생기므로 게임을 하면서 동시에 채팅까지 할 수 있는 시스템을 만든다.

1.2 범위 및 목표

1:N 통신을 이용하면서 하나의 서버의 다중 클라이언트가 접속하는 프로그램이다. 목표 범위는 2~4명 정도의 사용자 간 통신이다.

1.3 정의

본 프로그램은 방에 입장해서 사용자끼리 게임을 진행하면서 채팅을 가능하게 한 1:N TCP/IP 소켓 통신 기반 같은그림찾기 게임이다.

2. 시스템 구성

2.1 시스템 기능

1) 서버오픈

통신의 전반적인 진행을 담당하는 서버를 오픈한다.

2) IP 연결

서버에 해당하는 IP주소를 입력하여 연결시키면 자신의 대화명을 입력할 수 있는 상황으로 넘어가게 된다.

3) 방 생성

게임을 진행하고 상대방과 통신하기 위해 만드는 방으로 방을 만들게 되면 방의 정보가 서버의 Log에 기록된다. 서버는 클라이언트에게 개설되어진 방의 정보를 주고, 클라이언트의 개설방 메뉴에서 만들어진 방에 입장할 수 있다.

방에 입장하면 채팅을 입력하면 서버의 Log에 기록하고, 서버에서 또 다른 클라이언트에게 정보를 넘겨준다.

4) 대화명 변경

자신 혹은 상대방이 만든 닉네임을 바꾸게 하는 역할

5) 게임 준비

게임을 시작하기에 앞서 기다리는 신호로 서로 게임 준비를 누를 경우 3초 후에 게임이 시작된다.

6) 카드판 배열

랜덤으로 카드판이 배열되고, 시작하기 전에 전체 카드의 앞면을 보여준다.

7) 게임 진행

카드 두 개를 선택해서 같은 카드이면 이미지를 숨기고 카운트 수를 늘린다. 모든 카드를 다 뒤집었다고 카운트되면 게임을 종료한다.

8) 승률

게임 진행 도중에 클라이언트가 지금까지의 전적을 요청한다. 전적을 클라이언트의 대화창에 뿌려준다.

9) 나가기

게임이 끝나고 상대방과의 연결을 끊고 싶을 경우 나가기를 눌러서 빠져나간다

2.2 사용자 특성 및 성능

본 시스템은 서버 1개, 클라이언트 N개를 사용하는 소켓 통신이고, IP주소가 최소 2개는 필요하다. 연결 시 서버, 클라이언트 간 동일한 포트 번호와 같은 네트워크 환경에서 정상적으로 작동한다.

2.3 인터페이스

- ✓ 서버와 클라이언트로 구성
- ✓ 서버의 리스트박스에 생성된 방, Log 표시
- ✓ 클라이언트는 서버에 접속한 이후 대화명 변경 버튼을 통해 변경 가능.
- ✓ 연결 이후 게임 시작 전에 3초간 모든 카드의 앞면을 보여줌
- ✓ 한쪽 클라이언트의 게임이 종료되면 반대편 클라이언트에 메시지를 보냄
- ✓ 게임 진행 중 서버의 접속이 끊기면 클라이언트에게 메시지 보냄

2.4 패킷 구조

Client : 클라이언트 정보
Message : 메시지 종류
Data Size : 데이터 사이즈(오류 검출)
Payload : 데이터

```
#define SER_MSG_ROOMINFO     100
#define SER_MSG_CHATSTR      101
#define SER_MSG_JOINABLE     102
#define SER_MSG_CREATEABLE   103
#define SER_MSG_NAMECNGABLE  104
#define SER_MSG_USERINFO     105
```

서버에서 사용되는 메시지

```
#define CLN_MSG_ROOMINFO     202
#define CLN_MSG_ROOMSEL      203
#define CLN_MSG_CHATSTR      204
#define CLN_MSG_CREATEROOM   205
#define CLN_MSG_NAMECHANGE   206
#define CLN_MSG_ROOMOUT      207

#define CLN_FILE_READY       301
#define CLN_FILE_GO          302
#define CLN_FILE_TRANS       303
```

클라이언트에서 사용되는 메시지

```
// 게임 동작 관련 상수
#define GAME_MSG_READY 401
#define GAME_MSG_START 402
#define GAME_MSG_PLACE 403
#define GAME_MSG_FINISH 404
#define GAME_MSG_WIN 405
#define GAME_MSG_LOSE 406
```

게임 내에서 사용되는 메시지

2.5 메시지 정보

서버	메시지
SER_MSG_ROOMINFO	방 정보 메시지
SER_MSG_CHATSTR	채팅 문자열 메시지
SER_MSG_JOINABLE	입장 가능 여부 메시지
SER_MSG_NAMECNGABLE	대화명 변경 가능 여부 메시지
SER_MSG_USERINFO	사용자 정보 메시지

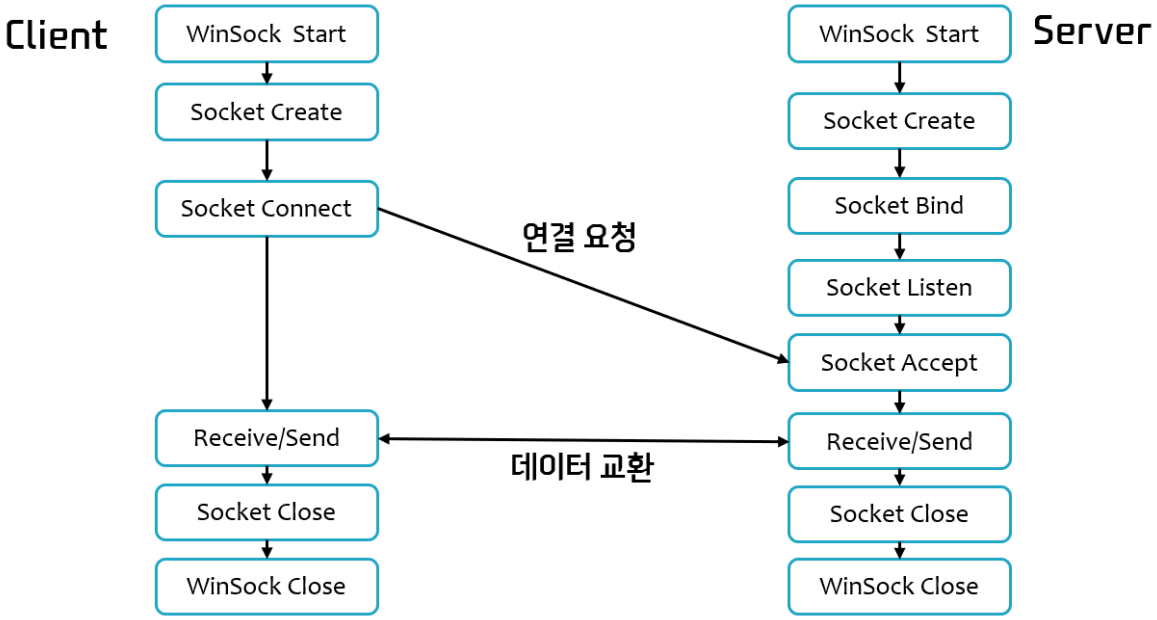
클라이언트	메시지
CLN_MSG_ROOMINFO	방 정보 메시지
CLN_MSG_ROOMSEL	방 선택 메시지
CLN_MSG_CHATSTR	채팅 문자열 메시지
CLN_MSG_CREATEROOM	대화명 변경 가능 여부 메시지
CLN_MSG_NAMECHANGE	사용자 정보 메시지
CLN_MSG_ROOMOUT	방 탈출 정보 메시지

클라이언트	메시지
GAME_MSG_READY	게임 준비 메시지
GAME_MSG_START	게임 시작 메시지
GAME_MSG_PLACE	카드 배치 메시지
GAME_MSG_FINISH	게임 종료 메시지

3. 세부 시스템

3.1 시스템 구조도

3.2 시스템 흐름도



서버에서 소켓을 뚫어 데이터를 주고받을 수 있는 통로를 만든 이후, Accept와 Receive는 다중 쓰레드 형태로 구성해서 다중 클라이언트 통신을 할 수 있게 한다. 모든 동작을 끝마치면 소켓을 close한다.

3.3 시스템 구현

3.3.1 서버

1) 서버오픈

통신의 전반적인 진행을 담당하는 서버를 오픈한다.

```
void CChatRoomServerDlg::OnServeropen()
{
    if(!m_bOpen)
    {
        if(WSAStartup(MAKEWORD(2, 2), &m_wsaData)!= 0)
        {
            AfxMessageBox("WSAStartup() Error!");
            return;
        }

        m_hSock = socket(PF_INET, SOCK_STREAM, 0);
        if(m_hSock == INVALID_SOCKET)
        {
            AfxMessageBox("socket() Error!");
            WSACleanup();
            return;
        }

        memset(&m_sAddr, 0, sizeof(m_sAddr));
        m_sAddr.sin_family = AF_INET;
        m_sAddr.sin_addr.s_addr = htonl(INADDR_ANY);
        m_sAddr.sin_port = htons(9190);

        if(bind(m_hSock, (SOCKADDR*)&m_sAddr, sizeof(m_sAddr))
            == SOCKET_ERROR)
        {
            AfxMessageBox("bind() Error!");
            closesocket(m_hSock);
            WSACleanup();
            return;
        }

        if(listen(m_hSock, 5) == SOCKET_ERROR)
        {
            AfxMessageBox("listen() Error!");
            closesocket(m_hSock);
            WSACleanup();
            return;
        }

        AfxMessageBox("서버 오픈!");
        m_bOpen = TRUE; //서버 오픈
        ROOMINFO waitingRoom;
        waitingRoom.roomName = "Wating Room";
        m_roomListVec.push_back(waitingRoom);

        UpdateRoomListBox();

        AfxBeginThread(AcceptThread, this);
    }
    else
        AfxMessageBox("Now Open");
}
```

Winsock() -> socket() -> bind() -> Listen() 소켓을 뚫고 Accept 쓰레드를 실행한다.

2) 방 설정

클라이언트가 접속할 방을 정의하고 Receive Thread를 실행한다.

```
UINT AcceptThread(LPVOID arg)
{
    CChatRoomServerDlg* pArg = (CChatRoomServerDlg*)arg;

    REVTHREADARG revArg;

    while(1)
    {
        SOCKET hCSock;
        SOCKADDR_IN cAddr;
        int cAddrSize = sizeof(cAddr);

        hCSock = accept(pArg->m_hSock, (SOCKADDR*)&cAddr, &cAddrSize);
        if(hCSock == INVALID_SOCKET)
        {
            AfxMessageBox("accept() Error!");
            break;
        }

        CString startMsg;
        startMsg.Format("Clnt Connection : %s", inet_ntoa(cAddr.sin_addr));
        pArg->WriteLog(startMsg);

        CLNTINFO clntInfo;
        clntInfo.sock = hCSock;
        clntInfo.name = "익명";
        clntInfo.roomName = "Wating Room";
        clntInfo.ip = inet_ntoa(cAddr.sin_addr);
        clntInfo.ready = false;

        g_cs.Lock();
        pArg->m_roomListVec[0].member.push_back(clntInfo);
        g_cs.Unlock();

        pArg->m_connectList.AddString(clntInfo.name);////

        revArg.pMainDlg = pArg;
        revArg.clntInfo = clntInfo;

        AfxBeginThread(RecvThread, &revArg);
    }
    return 0;
}
```

3) 데이터 수신

무한 루프로 클라이언트로부터 데이터 수신여부를 검사한다.

```
UINT RecvThread(LPVOID arg)
{
    REVTHREADARG* pArg = (REVTHREADARG*)arg;
    CChatRoomServerDlg* pMainDlg = pArg->pMainDlg;
    CLNTINFO clntInfo = pArg->clntInfo;

    while(1)
    {
        int msg = 0;
        int dataSize = 0;
        void* pData = NULL;

        if(!RecvPacket(clntInfo.sock, &msg, &dataSize, &pData))
            break;
        else
        {
            pMainDlg->PacketProcessing(clntInfo, msg, dataSize, pData);
        }

        if(pData)
            delete[] pData;
    }
```

4) 패킷 처리

메시지의 종류에 따라 패킷을 처리한다.

```
void CChatRoomServerDlg::PacketProcessing(CLNTINFO& clnt, int msg, int dataSize, void* pData)
{
    switch(msg)
    {
    case CLN_MSG_ROOMINFO:
        { ... }
        break;

    case CLN_MSG_CHATSTR:
        { ... }
        break;

    case CLN_MSG_ROOMSEL:
        { ... }
        break;

    case CLN_MSG_ROOMOUT:
        { ... }
        break;

    case CLN_MSG_CREATEROOM:
        { ... }
        break;

    case CLN_MSG_NAMECHANGE:
        { ... }
        break;

    case GAME_MSG_READY:
        { ... }
        break;

    case GAME_MSG_START:
        break;

    case GAME_MSG_PLACE:
        break;
    case GAME_MSG_FINISH:
        { ... }
        break;

    default:
        AfxMessageBox("PacketProcessing() Error!");
        break;
    }
}
```

5) 서버 종료

동작이 끝났으면 소켓을 닫는다.

```
void CChatRoomServerDlg::OnDestroy()
{
    CDialog::OnDestroy();

    closesocket(m_hSock);
    WSACleanup();
}
```

3.3.2 클라이언트

1) 서버에 연결

통신을 위해 서버에 연결한다.

```
void CMfcChatRoomClntDlg::OnConnection()
{
    if(!m_bConn)
    {
        if(WSAStartup(MAKEWORD(2, 2), &m_wsaData) != 0)
        {
            AfxMessageBox("WSAStartup() Error!");
            return;
        }

        m_hCSock = socket(PF_INET, SOCK_STREAM, 0);

        if(m_hCSock == INVALID_SOCKET)
        {
            AfxMessageBox("socket() Error!");
            WSACleanup();
            return;
        }

        CString strIp;
        GetDlgItemText(IDC_IPADDRESS, strIp);

        memset(&m_sAddr, 0, sizeof(m_sAddr));
        m_sAddr.sin_family = AF_INET;
        m_sAddr.sin_addr.s_addr = inet_addr(strIp);
        m_sAddr.sin_port = htons(9190);

        if(connect(m_hCSock, (SOCKADDR*)&m_sAddr, sizeof(m_sAddr))
            == SOCKET_ERROR)
        {
            AfxMessageBox("connect() Error!");
            closesocket(m_hCSock);
            WSACleanup();
            return;
        }

        m_bConn = TRUE;
        ButtonStateChage();

        OnNamechange();

        SendPacket(m_hCSock, CLN_MSG_ROOMINFO, 0, 0);

        AfxBeginThread(RecvThread, this);
    }
    else
        AfxMessageBox("이미 접속중!");
}
```

2) 데이터 수신

무한 루프로 클라이언트로부터 데이터 수신여부를 검사한다.

```
UINT RecvThread(LPVOID arg)
{
    CMfcChatRoomClntDlg* pDlg = (CMfcChatRoomClntDlg*)arg;
    SOCKET hCSock = pDlg->m_hCSock;
    CLNTINFO dump;
    while(1)
    {
        int msg = 0;
        int dataSize = 0;
        void* pData = NULL;

        if(!RecvPacket(hCSock, &msg, &dataSize, &pData))
            break;
        else
            pDlg->PacketProcessing(dump, msg, dataSize, pData);

        if(pData)
            delete[] pData;
    }
    ::SendMessage(pDlg->GetSafeHwnd(), WM_COMMAND, (WPARAM)IDC_DISCONNECTION, 0);

    AfxMessageBox("RecvThread 접속종료");
    pDlg->gamestate = STOP;
    pDlg->m_hint_flag = 0;
    pDlg->m_mouse_diable = 0;
    pDlg->Invalidate();

    return 0;
}
```

3) 패킷 처리

메시지의 종류에 따라 패킷을 처리한다.

```
void CMfcChatRoomClntDlg::PacketProcessing(CLNTINFO& clnt, int msg, int dataSize, void* pData)
{
    switch(msg)
    {
    case SER_MSG_ROOMINFO:
        UpdateRoomList((char*)pData);
        break;

    case SER_MSG_CHATSTR:
        { ... }
        break;

    case SER_MSG_JOINABLE:
        if(*((int*)pData)){ ... }
        else{ ... }
        break;

    case SER_MSG_CREATEABLE:
        if(*((int*)pData)){ ... }
        else{ ... }
        break;

    case SER_MSG_NAMECNGABLE:
        if(*((int*)pData))
            AfxMessageBox("닉네임이 생성 되었습니다");
        else{ ... }

        break;
    case GAME_MSG_READY:
        break;

    case GAME_MSG_START:
        { ... }
        break;

    case GAME_MSG_PLACE:
        { ... }
        break;

    case GAME_MSG_FINISH:
        // ...
        { ... }
        break;

    case SER_MSG_USERINFO:
        if(*((int*)pData)){ ... }
        else
        {
            m_userListBox.ResetContent();
            m_usrlst.RemoveAll();
        }
        break;

    }
}
```

4) 게임 진행 및 종료

게임을 진행하고 모든 카드의 짝을 다 맞추었으면 서버로 게임 종료 패킷 보냄

```
if(correctCount == 18){
    char finish[7] = {'F', 'I', 'N', 'I', 'S', 'H'};
    SendPacket(m_hCSock, GAME_MSG_FINISH, strlen(finish), finish);
    correctCount = 0;

    CString str;
    int nLen = m_chatingEdit.GetWindowTextLength();
    m_chatingEdit.SetSel(nLen, nLen);
    str = "****** 게임을 승리하셨습니다. ******₩r₩n";
    m_chatingEdit.ReplaceSel(str);

    memset(&m_card_table, 0, sizeof(m_card_table));
    gamestate = STOP;
    m_hint_flag = 0;
    m_mouse_diable = 0;
    Invalidate();
}
```

5) 연결 종료

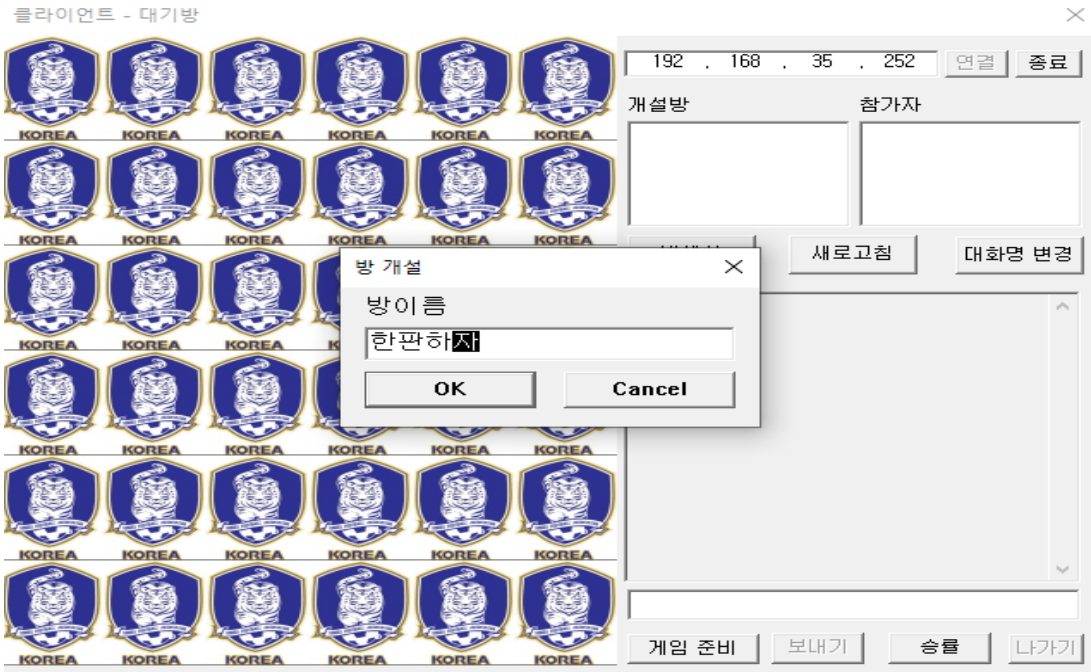
접속 종료

```
void CMfcChatRoomClntDlg::OnDisconnection()
{
    if(m_blnRoom)
        SendPacket(m_hCSock, CLN_MSG_ROOMOUT, 0, 0);

    closesocket(m_hCSock);
    WSACleanup();
    m_bConn = m_blnRoom = FALSE;
    ButtonStateChage();

    m_userListBox.ResetContent();
    m_roomListBox.ResetContent();
}
```

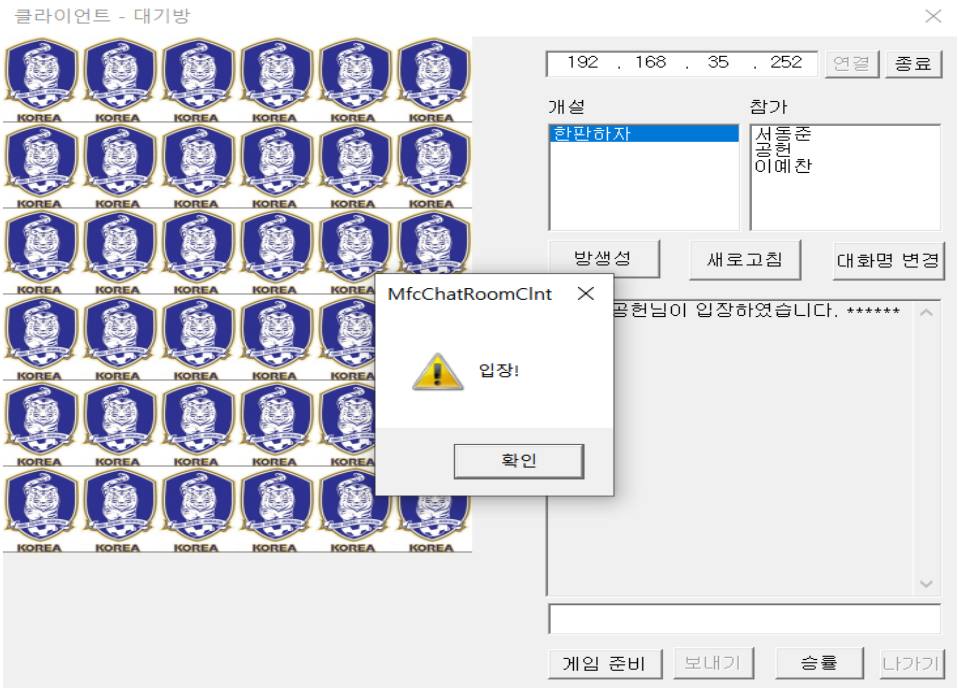
4. 시스템 테스트 결과

4.1 서버 오픈

4.2 방생성



4.3 방 입장



4.4 대화 및 준비 완료

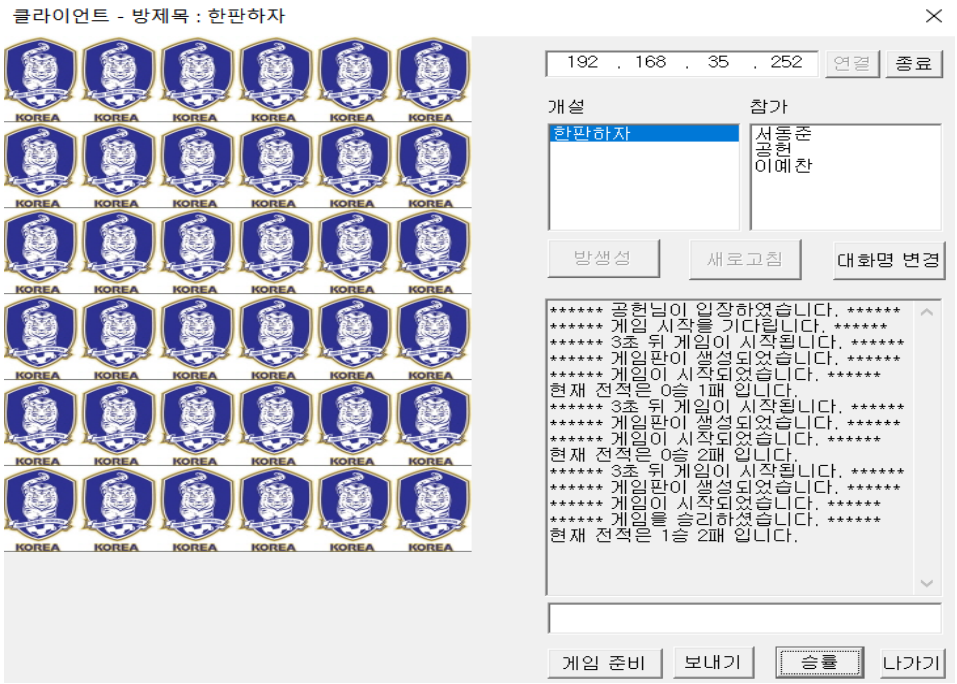
4.5 게임 시작 전 3초간 앞면 보여줌

4.6 게임 진행 - 같은 그림 짝 맞추기

4.7 게임 승리 & 패배

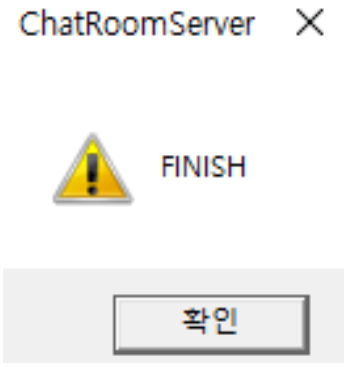
(총3판게임 -> client 1 : 2승1패 client 2 : 1승2패 client 3 : 0승3패)

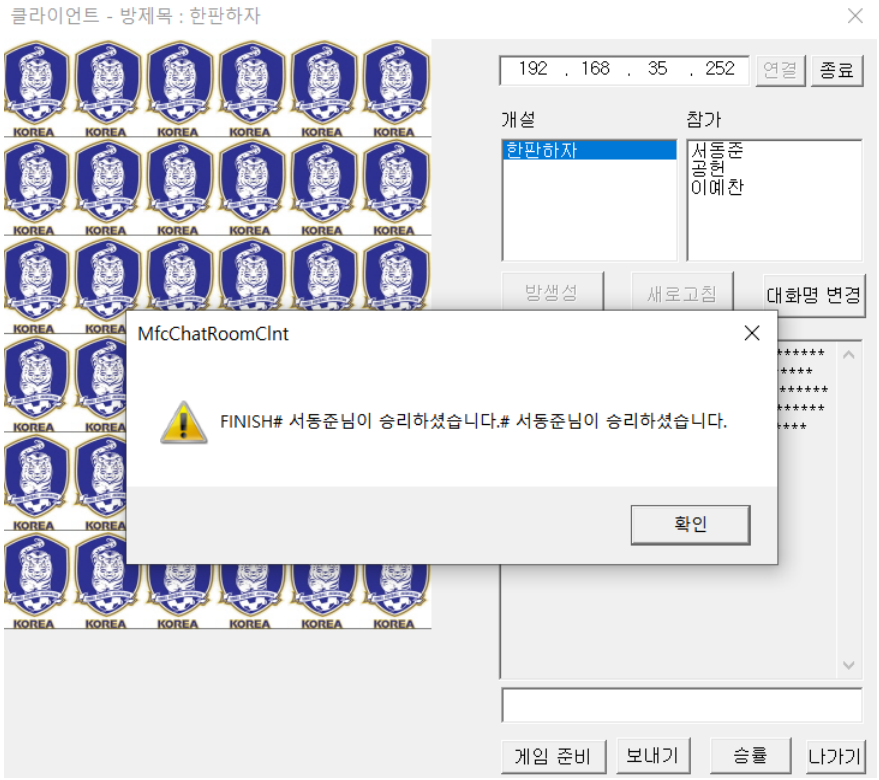
(client 1)



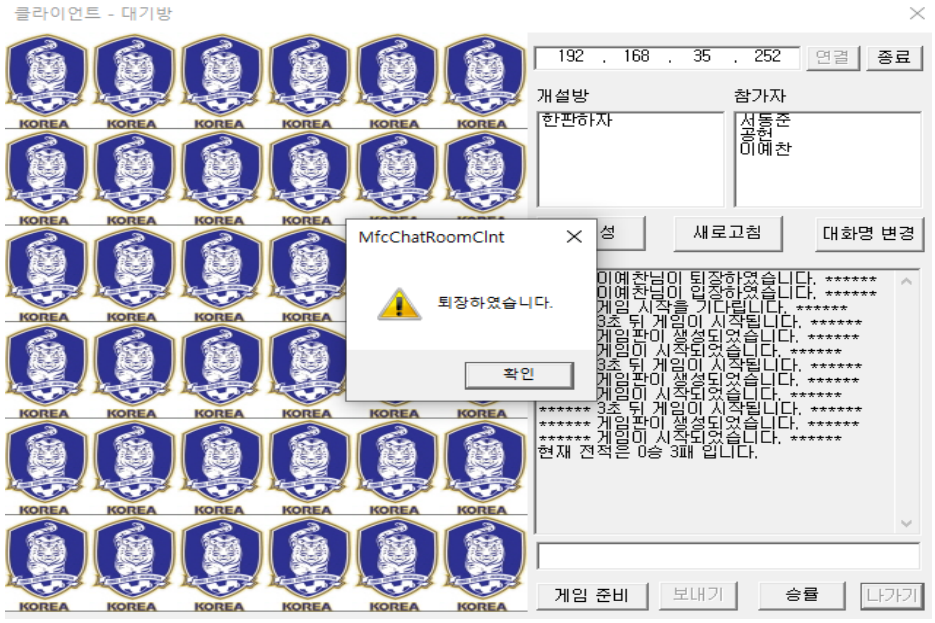
(client 2)

4.8 상대편 클라이언트에 승리 메시지

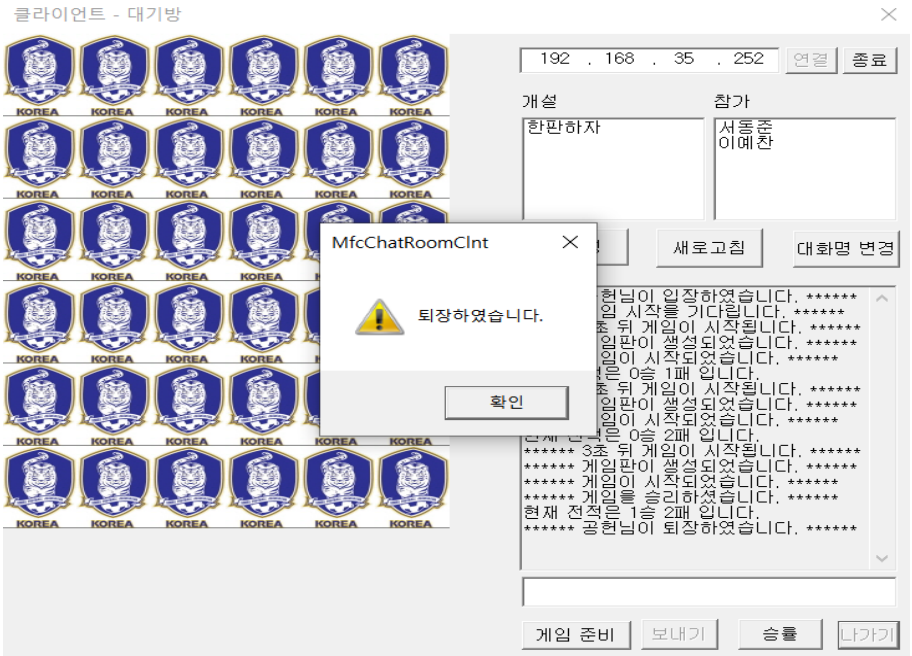




4.9 퇴장 메시지

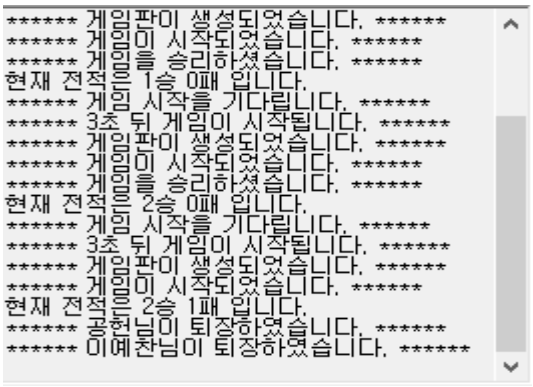


(client 1)



(client 2)

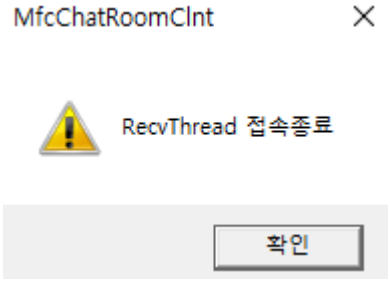
4.10 클라이언트 대화창 및 서버 Log



(client 1)

(Server Log)

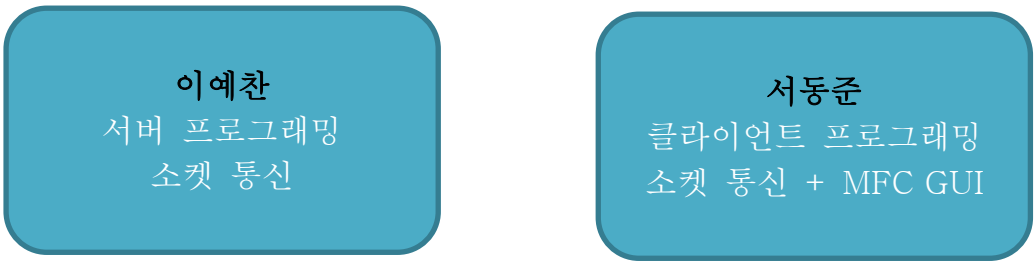
4.11 서버 닫을 경우 클라이언트에 메시지



5. 역할 분담 및 실습 소감

5.1 역할 분담

서버를 먼저 구성하여 소켓 통신의 전반적인 진행을 시켰고 클라이언트가 접속하기 위한 세팅을 했다. 뚫어 놓은 소켓에 클라이언트가 접속하여 데이터를 주고받기 전에 클라이언트 측 GUI를 MFC 이용하여 설정한 뒤, 송수신 패킷을 메시지를 이용하여 구별하였다.



5.2 실습 소감

기존에 있던 MFC 게임에 소켓통신을 추가하기 위해 게임의 진행방식도 많이 바꾸어야 한다는 걸 깨달았다.

통신을 위해 소켓을 뚫을 때 미리 정의된 Winsock을 이용하여 통신을 진행하기 때문에 갖다 쓰기만 하면 되서 편했다.

1:N 통신을 위해서 고려해야할 사항이 많다는 걸 알게 되었고, 처음에는 채팅과 게임을 따로 생각해서 갈피를 잡지 못했었는데, 채팅이나 게임이나 패킷을 보내고 메시지에 따라 처리하면 되기 때문에 후반에 와서는 쉽게 느껴졌다.