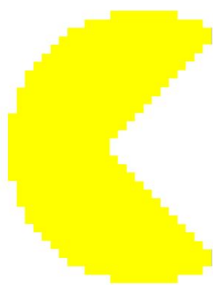


[별첨1]
[표지]

작품 설명서 멀티플레이어 팩맨



PACMAN

방 만들기 접속하기

안할래요

반	5
팀명	팩맨

팀 구성	팀 대표 20510 손정훈
	팀원1 20509 박민준
	팀원2 20520 이찬희
	팀원3 20525 최우혁
개발 언어 및 환경	C++ / Windows 10

[본문]

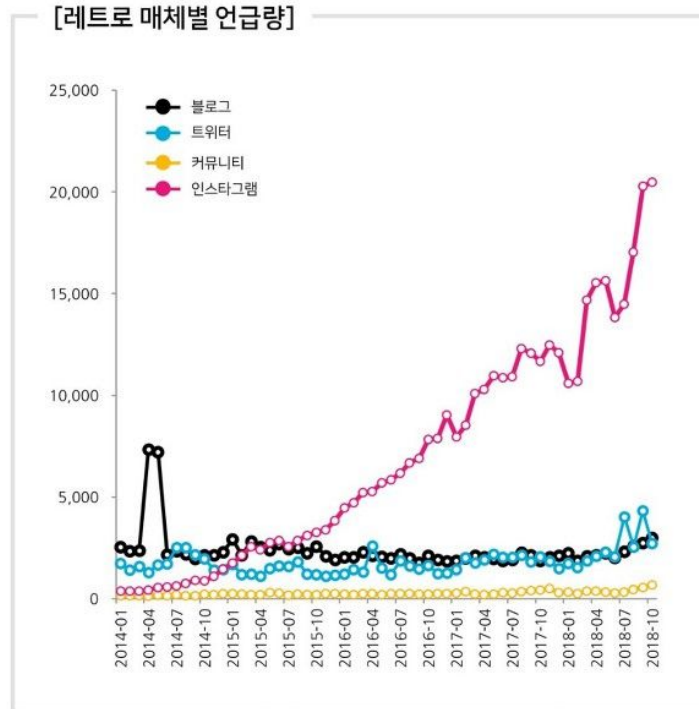
1. 작품 개요

가. 개발 동기 및 기대효과

- 콘텐츠 선정 사유, 벤치마킹 사례 및 분석 결과, 작품의 유용성 등

저희는 게임을 만들기 위해 시장을 조사하던 중 요즘 유행하는 트렌드를 한번 생각해봤습니다.

그러던 중 **레트로** 라는 키워드가 요즘 유행 한다는 것을 확인하였고 이에 따라서 과거의 게임은 뭐가 있을지 고민을 하였습니다.



* 탐색어: (주어)레트로||retro+(제외어:이벤트(관련 RT포함)||나눔||이벤||추천||입금||계좌||금액||증정)/
(원문)합하다||인스타그램||인스타
* 기간:2017.01.01.~2018.10.31.

(레트로의 언급량 증가를 보여줌)

과거 유행 하던 게임을 찾아봤는데 팩맨, 덤, 스트리트 파이터, 슈퍼 마리오, 갤러그 등 많은 게임을 발견하였습니다. 그 중 저희의 기술적으로 가능할 만한 게임을 고민을 하던 중 팩맨은 만들 수 있다고 생각 하였습니다. 하지만 저희는 그냥 팩맨을 만드는것은 큰 차별성이 없다고 생각을 하였고 ‘어떻게 하면 차별성을 넣을 수는 없을까?’라고 고민을 하였습니다. 처음에는 3D 로 게임을 만들자, 멀티플레이를 만들자, 맵을 유저가 만들자, 등등 의견이 나왔고 저희는 이 중 winsock을 한 번 사용해 보고 싶었고 그래서 멀티 플레이를 만들었습니다. 이 작품을 만들어서 저희는 40,50 시대들이 과거 게임의 향수를 느끼게 만들고 싶습니다.

나. 게임 개발 환경 및 사용 환경

- 개발 환경

VS 2019, DirectX SDK June 2010

ASUSTeK COMPUTER INC.

AMD Ryzen 7 4800HS with Radeon Graphics 2.90 GHz

16.0GB(15.4GB 사용 가능)

64비트 운영 체제, x64 기반 프로세서

펜 지원

Windows 정품 인증을 받았습니다.

튼튼한 정품윈도우

- 사용 환경

(os, 시연을 위해 사용자 환경에서 미리 설치되어야 하는 sw 명시)

DirectX 최종 사용자 런타임.

[Download DirectX 최종 사용자 런타임 웹 설치 관리자 from Official Microsoft Download Center](#)

다. 제작 과정


제작기간, 제작에 사용한 프로그램, 팀 프로젝트 진행방법 및 팀원 협업 과정을 자세히 제시

개발 일정	날짜	개발 계획
	11월 17일	개발 환경 세팅, 타이틀 씬 제작
	11월 18일~ 23일	클라이언트 개발, 충돌 처리.
	11월 23일~ 30일	서버 개발, 이동 코드 제작
	12월 01일	클라이언트와 서버의 통신을 위한 코드 작성
	12월 03일	게임 실제 테스트, 버그 픽스 , 빌드, 제출을 위한 exe 제작

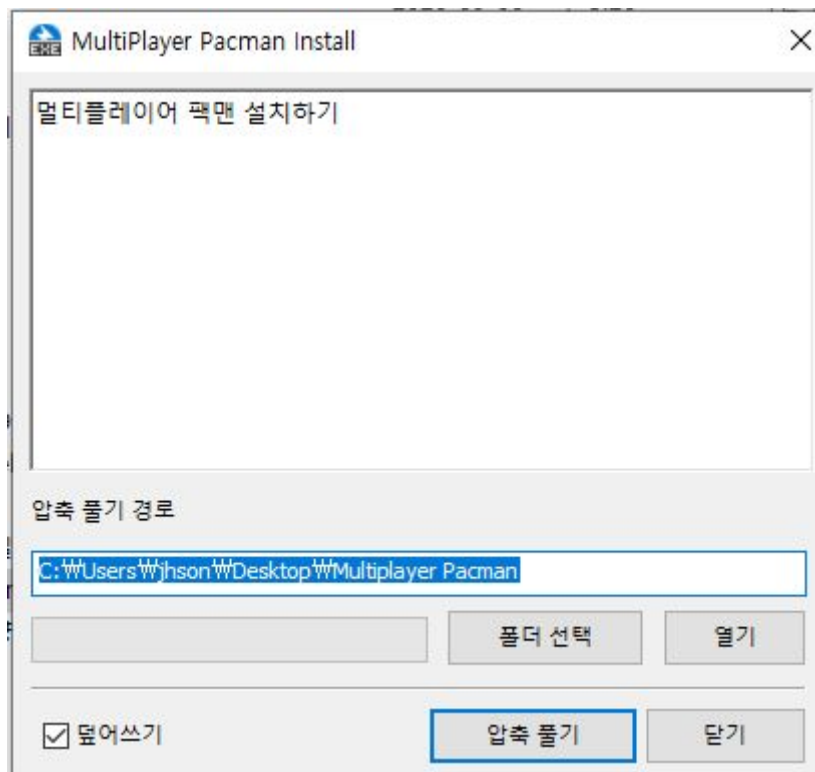
2. 게임 설치 방법

가. 게임 설치 방법

같이 보내진 압축 파일을 풀고 그 안에 있는

☒  Multiplayer Pacman.exe

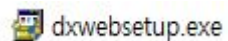
이 파일을 실행합니다



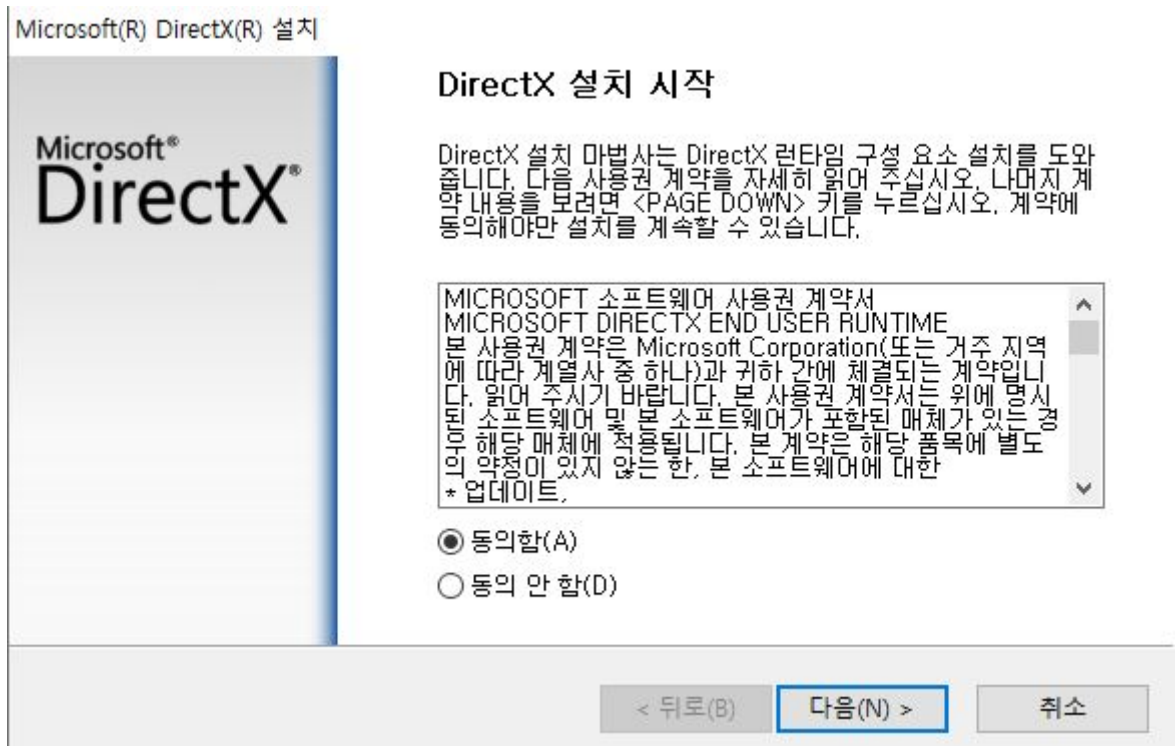
압축 풀기를 선택합니다



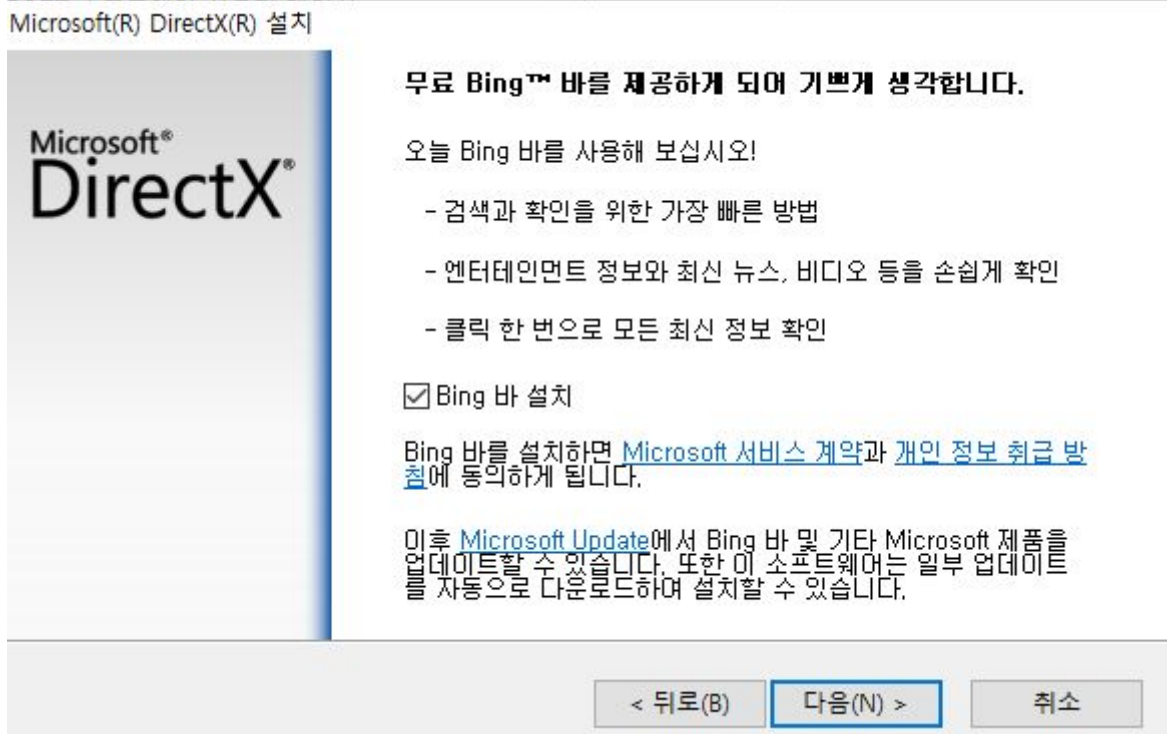
바탕 화면에 폴더가 생깁니다



폴더 안에 있는 dxwebsetup.exe 를 실행합니다



동의하고 다음





설치하면 기쁘지 않습니다 설치를 체크 해제하고 다음을 선택합니다

나. 게임 실행 방법

실행 파일 위치 및 파일명 기재

압축을 푼 폴더(경로를 변경하지 않았다면 데스크탑에 있습니다) 안에 Multiplayer Pacman(FullScreen 혹은 Windowed).exe 를 실행합니다.

 ElementDungeon(FullScreen).exe

 ElementDungeon(Windowed).exe


FullScreen은 전체화면으로 Windowed는 창모드로 실행됩니다

3. 게임 소개

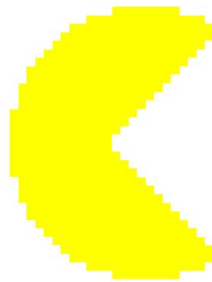
가. 사용 설명서

** 실행하시기 전 게임을 실행하시려면 두명의 사용자가 필요합니다. 로컬에서 두개를 실행시키는 방법은 하단에 나와있습니다.

 ElementDungeon(FullScreen).exe

 ElementDungeon(Windowed).exe

둘중하나를 클릭해 실행하면



PACMAN

방 만들기 접속하기

안할래요


전체화면으로 이 창이 뜹니다.

혹시 잘못 눌러서 켜졌다면 안할래요를 클릭하고 Y를 눌러 끌 수 있습니다.

방 만들기를 클릭하면



이런 창이 뜹니다. 액세스 허용을 눌러주세요.

최소화된  아이콘을 눌러 다시 게임을 켜 주세요
현재 인원 : 1

대기중입니다

시작하기

현재 방에서 대기중인 인원이 뜹니다.


멀티플레이어 게임이기 때문에 공유기를 사용중이시라면 외부에서 접속하려면 포트포워딩을 진행해야합니다. 공유기별 포트포워딩 방법은 아래 링크에 나와있습니다.


1234 포트를 열어주면 됩니다.

[IPTIME 공유기 포트 포워딩 설정 가이드 - 코디엔에스 매뉴얼 / 가이드 \(codns.com\)](#)

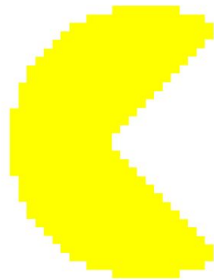
포트가 열려있다면 접속하려는 사람에게 본인의 ip 주소 (ex) 233.64.234.345
를 알려주면 됩니다.

접속하려는 사람은

 ElementDungeon(FullScreen).exe

 ElementDungeon(Windowed).exe

둘중 하나를 골라 게임을 켜고



PACMAN

방 만들기

접속하기

안할래요

에서 접속하기를 눌러

233.44.345.234

ip를 숫자로 입력하고 엔터를 누르면 연결됩니다.
만약 서버가 응답하지 않는 경우(포트포워딩을 잘못했거나 서버가 열려있지 않을 때)

접속 오류

233.44.345.234

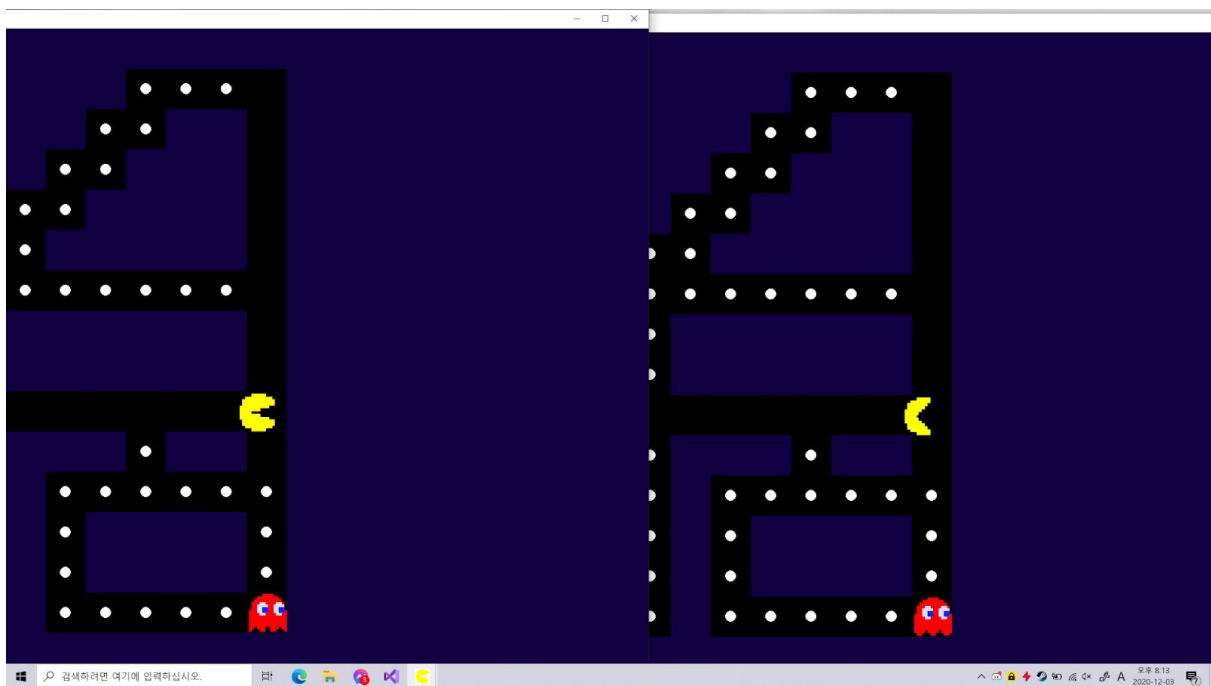
접속 오류가 뜹니다.

현재 인원 : 2

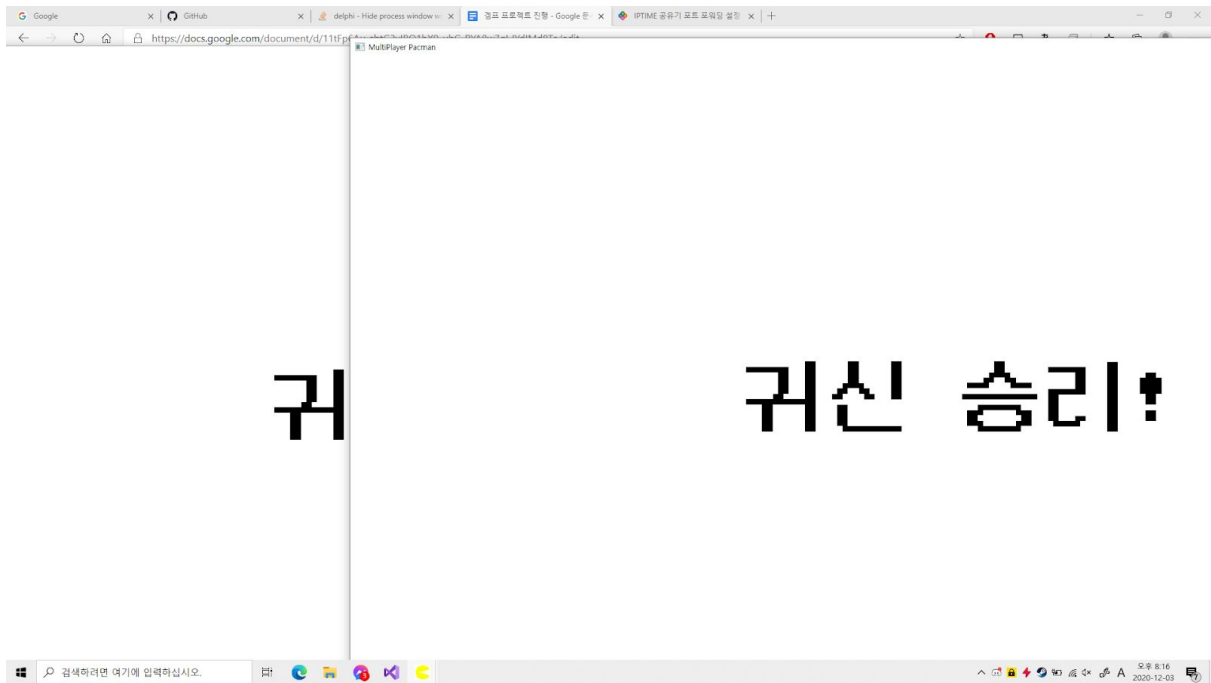
대기중입니다

시작하기

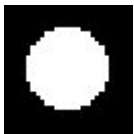
접속이 성공적으로 되면 호스트의 화면에 인원이 늘어납니다. 여기서 시작하기를 누르면 게임이 시작됩니다.



하얀걸 전부 먹으면 팩맨의 승리
귀신이 잡으면 귀신 승리



양쪽에 이렇게 뜨고 게임이 종료됩니다.



<- 이렇게 생긴 큰 하얀걸 먹으면 3초동안 팩맨이 귀신과 닿았을 때 귀신을 스텐시킵니다.

두대의 컴퓨터가 없어 어쩔 수 없이 로컬에서 테스트 해보려면

 ElementDungeon(Windowed).exe 이걸 두번 실행합니다.

하나는 방 만들기, 하나는 접속하기를 선택합니다.

접속하려고 하는 클라이언트에서



127.0.0.1

127.0.0.1을 치고 엔터를 클릭 후 호스트에서 게임을 시작합니다.

단* 엔진 특성상 렌더되지 않는 화면이 있는 프로그램 혹은 사용자가 화면의 위치를 바꾸는 등의 행동을 하는 동안 연산과 출력을 멈추기 때문에 서버와의 통신 오류 혹은 이동 관련 오류 등이 생길 수 있기 때문에 권장하지 않습니다.

나. 기술적 난이도

```
si.wShowWindow = SW_HIDE;
si.dwFlags = STARTF_USESHOWWINDOW;
CreateProcess(_T("SocketServer.exe"), NULL, NULL, NULL, false, 0, NULL, NULL, &si, &pi);
```

```
1 SocketClient::CloseServer() {
    TerminateProcess(pi.hProcess, 0);
    CloseHandle(pi.hProcess);
    CloseHandle(pi.hThread);
}
```

서버를 실행하기 위해 사용한 코드입니다. 서버는 SocketServer.exe 로 실행하지만 호스트가 서버를 실행하기 위해 따로 SocketServer.exe 를 눌러 켜지 않아도 되도록 CreateProcess를 이용해 서버를 켜줍니다. 서버를 킬 때 서버 콘솔창을 띄우지 않게 하기 위해서 STARTUPINFO 구조체에 있는 wShowWindow를 SW_HIDE(창을 가림) 으로 설정하고 dwFlags를 STARTF_USESHOWWINDOW(wShowWindow 설정을 사용함)으로 설정해 창이 뜨지 않게 했습니다. 게임을 종료했을 때 SocketServer.exe 프로세스는 자동으로 종료되지 않기 때문에 TerminateProcess를 통해 미리 PROCESS_INFORMATION 구조체에 담아준 hProcess HANDLE을 넣어줘 종료시키고 핸들을 닫습니다. 이 CloseServer 함수는 WinMain 함수가 종료될 때 실행되도록 해서 게임이 꺼지면 자동으로 서버도 꺼지도록 구현했습니다.

소켓 클라이언트가 매 씬마다 새로 생성된다면 (일반적인 생성법인 SocketClient* sock = new SocketClient(); 등으로 생성할 경우) 서버와의 통신이 중간에 끊어지기 때문에 SocketClient를 정적으로 만들어 씬이 변경되더라도 서버와의 통신을 유지할 수 있도록 하였습니다.

```
#define Socket SocketClient::instance()

static SocketClient* instance() {
    static SocketClient* instance = new SocketClient();
    return instance;
}
```

#define을 통해 Socket->함수명(); 으로 서버와의 통신을 할 수 있도록 만든 코드.

서버 코드

```
WSADATA wsa;

if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0)
    return -1;
```

WinSock2 초기화를 위해 사용한 코드.

```
SOCKADDR_IN serveraddr;
ZeroMemory(&serveraddr, sizeof(serveraddr));
serveraddr.sin_family = AF_INET;
serveraddr.sin_port = htons(1234);
```

1234 포트로 들어오는 IPV4 요청을 받도록 함.

```
return_val = bind(listen_sock, (SOCKADDR*)&serveraddr, sizeof(serveraddr));
```

ip주소와 포트번호에 바인딩합니다. 이때 이미 이 포트 번호를 사용중인 프로그램이 있다면 오류를 냅니다.

```

return_val = listen(listen_sock, SOMAXCONN);

HANDLE hThread;
DWORD ThreadID;

while (1)
{
    addrlen = sizeof(clientaddr);

    client_sock = accept(listen_sock, (SOCKADDR*)&clientaddr, &addrlen);
    if (client_sock == INVALID_SOCKET) {
        printf("accept 에러\n");
        continue;
    }

    printf("ip 주소 = %s, internalPort = %d\n", inet_ntoa(clientaddr.sin_addr), ntohs(clientaddr.sin_port));

    hThread = CreateThread(NULL, 0, ProcessClient, (LPVOID)client_sock, 0, &ThreadID);
    if (hThread == NULL)
    {
        printf("쓰레드생성실패\n");
    }
    else
    {
        CloseHandle(hThread);
    }
}

```

이때 ProcessClient 쓰레드를 만듭니다. 서버로 들어온 요청을 쓰레드로 만들어 여러 요청을 한번에 처리할 수 있도록 합니다. 멀티쓰레딩 방식을 사용하지 않으면 한번에 하나의 요청만 처리할 수 있기 때문에 두개 이상의 클라이언트의 접속이 불가능합니다.

```

DWORD WINAPI ProcessClient(LPVOID arg)
{
    SOCKET client_sock = (SOCKET)arg;
    char buf[BUFSIZE + 1];
    SOCKADDR_IN clientaddr;
    int addrlen;
    int retval;

    addrlen = sizeof(clientaddr);
    getpeername(client_sock, (SOCKADDR*)&clientaddr, &addrlen);

```

ProcessClient에서는

클라이언트에서 보낸 정보를 받기 위해 소켓을 하나 만들고, 정보를 저장할 char 배열을 만듭니다.

```

while (1)
{
    retval = recv(client_sock, buf, BUFSIZE, 0);
    if (retval == SOCKET_ERROR)
    {
        cout << "recv 에러\n";
        break;
    }
    else if (retval == 0)
        break;
}

```

반복적으로 수신하기 위해 while 문 안에 넣어서 계속 수신합니다.

수신한 정보에 맞게 서버에서 처리합니다

```
if ((strcmp(buf, "join")) == 0) {
    ++playerNumber;
    playerPos.push_back(new Vector2());
    retval = send(client_sock, to_string(playerNumber).c_str(), to_string(playerNumber).size(), 0);
}
if ((strcmp(buf, "start")) == 0) {
    gamestate = "start";
}
if ((strcmp(buf, "gameclear")) == 0) {
    gamestate = "gameclear";
}
if ((strcmp(buf, "gameover")) == 0) {
    gamestate = "gameover";
}
if ((strcmp(buf, "gamestate")) == 0) {
    retval = send(client_sock, gamestate.c_str(), gamestate.size(), 0);
}
if ((strcmp(buf, "currentplayernum")) == 0) {
    retval = send(client_sock, to_string(playerNumber).c_str(), 1024, 0);
}
```

플레이어의 위치처리를 위해 vector를 이용해서 플레이어의 위치를 넣었습니다.

`vector<Vector2*> playerPos;` (Vector2는 단순하게 float x; float y; 로 선언된 클래스입니다)

```

if (s.substr(0, 3) == "pos") {
    stringstream ss;
    ss.str(s);
    vector<string> tempStringVector;
    string tempString;
    while (getline(ss, tempString, ',')) {
        tempStringVector.push_back(tempString);
    }
    switch (s.at(3))
    {
        case '1':
            playerPos[0]->x = stof(tempStringVector[1]);
            playerPos[0]->y = stof(tempStringVector[2]);
            break;
        case '2':
            playerPos[1]->x = stof(tempStringVector[1]);
            playerPos[1]->y = stof(tempStringVector[2]);
            break;
        case '3':
            playerPos[2]->x = stof(tempStringVector[1]);
            playerPos[2]->y = stof(tempStringVector[2]);
            break;
        case '4':
            playerPos[3]->x = stof(tempStringVector[1]);
            playerPos[3]->y = stof(tempStringVector[2]);
            break;
        default:
            break;
    }
    string posSend = "pos";
    for (int i = 0; i < playerPos.size(); i++) {
        posSend.append(",");
        posSend.append(to_string(playerPos[i]->x));
        posSend.append(",");
        posSend.append(to_string(playerPos[i]->y));
    }
    posSend.append("\0");
    retval = send(client_sock, posSend.c_str(), 1024, 0);
}

```

플레이어의 포지션이 수신되면 그 포지션 값을 string에서 float으로 stof(stringtofloat)을 이용해 바꿔준 뒤 playerPos vector 안에 넣어줍니다. 그리고 playerPos vector 전체를 다시 보내 다른 플레이어의 위치를 반영하게끔 합니다.

클라이언트에서 서버에서 온 정보 처리


```

int SocketClient::Init(string ip)
{
    cout << "SocketClient Init" << endl;
    WSStartup(MAKEWORD(2, 2), &wsaData);
    hSocket = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);

    tAddr.sin_family = AF_INET;
    tAddr.sin_port = htons(1234);
    tAddr.sin_addr.s_addr = inet_addr(ip.c_str());
    int connectResult = connect(hSocket, (SOCKADDR*)&tAddr, sizeof(tAddr));
    return connectResult;
    //send(hSocket, "test", 1024, 0);
}

```

일단 Init을 실행해서 서버와 똑같이 WinSock을 초기화 해줍니다. 그리고 connect를 통해 인자로 넘어온 ip 주소로 접속합니다.

```

void SocketClient::GetPlayerPos()
{
    ZeroMemory(cBuffer, 1024);
    recv(hSocket, cBuffer, 1024, 0);
    vector<string> vector_string;
    string temp_string;
    string s = cBuffer;
    stringstream ss;
    if (s.substr(0,3) == "pos") {
        ss.str(cBuffer);
        while (getline(ss, temp_string, ',')) {
            vector_string.push_back(temp_string);
        }
    }
    int j = 0;
    for (int i = 1; i < vector_string.size(); i+=2) {
        playerPos[j]->x = stof(vector_string[i]);
        playerPos[j]->y = stof(vector_string[i + 1]);
        j++;
    }
    //cout << playerPos[1]->x << endl;
}

```

서버에서 보내준 플레이어의 위치 정보를 엔진에서 사용할 수 있도록 ZeroVec 으로 변경해줍니다. 서버와 마찬가지로 ZeroVec vector 안에 플레이어 위치를 넣습니다.

이동 관련 코드
회전부터

```

void TestScene::TurnAnimation(float x, float y, float vx, float vy)
{
    float virtualPlayerPos[2] = { virtualPlayer->Pos().x, virtualPlayer->Pos().y };
    float playerPos[2] = { player->Pos().x, player->Pos().y };
    if (playerPos[0] != x) {
        if (x < playerPos[0]) {
            player->SetRot(0);
        }
        else {
            player->SetRot(180);
        }
    }
    else if (playerPos[1] != y) {
        if (y < playerPos[1]) {
            player->SetRot(90);
        }
        else {
            player->SetRot(-90);
        }
    }
    if (virtualPlayerPos[0] != vx) {
        if (vx < virtualPlayerPos[0]) {
            virtualPlayer->SetRot(0);
        }
        else {
            virtualPlayer->SetRot(180);
        }
    }
    if (virtualPlayerPos[1] != vy) {
        if (vy < virtualPlayerPos[1]) {
            virtualPlayer->SetRot(90);
        }
        else {
            virtualPlayer->SetRot(-90);
        }
    }
}

```

플레이어가 어느 방향을 향하고 있는지 서버로 보내주게되면 서버에서 처리해야할 정보가 늘어나기 때문에 클라이언트에서 이전 프레임과의 위치 차이를 검사해 스프라이트를 회전시킵니다.

이동 코드

```
//키보드 인풋 받아서 버퍼에 넣음
if (ZeroInputMgr->GetKey(VK_UP) == INPUTMGR_KEYDOWN) {
    directionBuffer = Direction::UP;
}
else if (ZeroInputMgr->GetKey(VK_DOWN) == INPUTMGR_KEYDOWN) {
    directionBuffer = Direction::DOWN;
}
else if (ZeroInputMgr->GetKey(VK_LEFT) == INPUTMGR_KEYDOWN) {
    directionBuffer = Direction::LEFT;
}
else if (ZeroInputMgr->GetKey(VK_RIGHT) == INPUTMGR_KEYDOWN) {
    directionBuffer = Direction::RIGHT;
}
```

일단 키 입력을 받으면 버퍼에 그에 맞는 Direction 열거형을 넣습니다.

```
if (directionBuffer != direction) {
    switch (directionBuffer)
    {
        case TestScene::Direction::RIGHT:
            if (direction == Direction::LEFT) {
                direction = directionBuffer;
            }
            break;
        case TestScene::Direction::LEFT:
            if (direction == Direction::RIGHT) {
                direction = directionBuffer;
            }
            break;
        case TestScene::Direction::UP:
            if (direction == Direction::DOWN) {
                direction = directionBuffer;
            }
            break;
        case TestScene::Direction::DOWN:
            if (direction == Direction::UP) {
                direction = directionBuffer;
            }
            break;
        default:
            break;
    }
}
```

현재 진행 방향이 버퍼와 다를 경우 버퍼와 진행 방향이 서로 180도면 바로 회전시킵니다.

```

//타일 가운데에 있는지 검사
if ((abs(fmod(playerX, 64)) < 5) && (abs(fmod(playerY, 64)) < 5)) {
    //충돌 처리
    int relativeX = round(playerX) / 64;
    int relativeY = round(playerY) / 64;
    direction = directionBuffer;

    switch (direction)
    {
    case TestScene::Direction::RIGHT:
        if (map[relativeY][relativeX + 1] == 1) {
            player->SetPos(round(playerX), round(playerY));
            isMoveable = false;
        }
        break;
    case TestScene::Direction::LEFT:
        if (map[relativeY][relativeX - 1] == 1) {
            player->SetPos(round(playerX), round(playerY));
            isMoveable = false;
        }
        break;
    case TestScene::Direction::UP:
        if (map[relativeY-1][relativeX] == 1) {
            player->SetPos(round(playerX), round(playerY));
            isMoveable = false;
        }
        break;
    case TestScene::Direction::DOWN:
        if (map[relativeY + 1][relativeX] == 1) {
            player->SetPos(round(playerX), round(playerY));
            isMoveable = false;
        }
        break;
    default:
        break;
    }
}

```

그렇지 않으면 타일 가운데에 플레이어가 있는지 확인하고 가운데에 있는 경우라면 회전시킵니다.

회전한 후 진행 방향에 있는 타일이 벽인지 아닌지 검사합니다. 벽이면 정지하고 벽이 아니면 그대로 이동합니다.

```
byte map[16][22] = {
    {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
    {1,0,0,0,0,0,2,1,0,0,0,0,1,1,1,1,0,0,0,2,1},
    {1,0,1,1,1,1,0,1,0,1,0,1,1,0,1,1,1,0,0,1,1,0,1},
    {1,0,1,1,1,1,0,1,0,0,1,0,1,1,1,0,0,1,1,1,0,1},
    {1,0,0,0,0,0,0,1,1,0,1,0,1,1,0,0,1,1,1,1,0,1},
    {1,0,1,1,1,1,0,1,1,0,1,0,1,0,0,1,1,1,1,1,0,1},
    {1,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,1,1,0,1,1,1,1,0,1,0,1,0,1,1,1,1,1,1,0,1},
    {1,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,1,1,0,1,1,1,1,0,1,0,1,0,1,1,1,1,1,1,0,1},
    {1,0,1,1,0,1,1,1,1,0,1,0,1,0,1,1,1,1,1,1,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,1},
    {1,1,0,1,0,1,0,1,0,1,1,1,1,0,1,0,1,1,1,0,1,1,0,1},
    {1,1,0,1,0,1,0,0,0,0,0,1,0,1,0,1,0,0,0,0,0,0,1},
    {1,0,0,1,0,1,1,1,1,0,1,0,1,0,1,0,1,1,1,1,0,1},
    {1,0,1,1,0,1,0,0,0,0,0,1,0,1,0,1,0,1,1,1,0,1},
    {1,2,0,0,0,0,0,0,1,1,1,1,0,0,0,1,0,0,0,0,0,0,1},
    {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}
};
```

타일의 위치는 byte형 2차원 배열을 통해 표현하고 플레이어의 위치를 그에 맞게 계산합니다.

게임 시작 코드

```
if (ZeroInputMgr->GetKey(VK_LBUTTON) == INPUTMGR_KEYDOWN && startSprite->IsOverlapped(ZeroInputMgr->GetClientPoint())) {

    switch (Socket->GetCurrentPlayerNumber()) {
    case 2:
        Socket->SendStringToServer("start\0");
        cout << "sending" << endl;
        break;
    case 3:
        break;
    case 4:
        break;
    default:
        errorFont->SetString("인원이 충분하지 않습니다");
        break;
    }
}
```

현재 서버에 접속해있는 인원을 토대로 2인용 맵, 3인용 맵, 4인용 맵을 따로 실행할 수 있습니다. 다만 맵을 여러개 만들지는 못했습니다.

다. 향후 개선/발전 계획

맵을 만들 시간이 없어 2인용까지만에 지원을 못했습니다. 서버와 클라이언트가 최대 4인까지 만들어져있어 맵을 마저 만들어 목표했던 기능을 만들고 싶습니다. 또한 밸런스 조절을 더 해볼 생각입니다. 팩맨이 한번 귀신과 닿으면 패배하는데 플레이하면서 너무 익숙해져버렸는지 거의 팩맨이 이기는 상황입니다. 고스트에게 스킬을 주어주는 등 조금 더 팩맨에게 어려운 요소를 추가하고 싶습니다.

귀신 애니메이션이 팩맨과 같이 회전으로 처리되는데 귀신은 눈만 움직여야 하기 때문에 이부분은 추가적인 코드로 보완해야할점입니다.