네트워크 게임 프로그래밍 개발 기획서

2012182045 최재용, 2014182021 성소윤

**게임 장르**

2D 탑뷰 슈팅게임

**설명**

우주 배경의 3:3 멀티플레이 슈팅게임으로 상대팀의 모선를 파괴하는 것이 목표이다.

맵의 양 끝에 각 팀의 모선이 있고 모선에서 조금 떨어진 위치에서 함선들이 소환된다. 모선의 체력은 일정 시간동안 공격받지 않으면 느리게 회복된다.

플레이어는 W,A,S,D 키를 이용해서 자신의 함선을 월드 좌표계 기준으로 북, 서, 남, 동 방향으로 이동시킬 수 있다. 함선은 플레이어의 마우스를 항상 바라보며, 마우스 좌클릭을 하면 함선의 기본공격, 우클릭을 하면 각 함선의 특수기술을 사용한다.

함선은 3종류가 있다.

1. 공격형

상대방의 함선과 모선을 공격하는 것에 특화된 함선. 좌클릭을 하면 Hit-Scan 방식의 레이저 공격을, 우클릭을 하면 점점 가속되어 날아가며 원하는 시점에서 터트릴 수 있는 투사체 미사일을 발사한다.

1. 방어형

상대의 공격을 방어하는 것에 특화된 함선. 좌클릭을 누르면 평범한 투사체를 발사하는 공격을 하고, 우클릭을 하면 정면에서 날아오는 모든 공격을 막는 방어막을 펼친다. 방어막을 펼치는 동안에는 방어막 게이지가 빠르게 감소하고, 방어막을 사용하지 않는 동안 게이지가 서서히 증가한다.

1. 지원형

아군의 생존을 담당하는 함선. 좌클릭은 조그마한 투사체를 발사하는데 적을 공격할 수 없고 아군에게 닿을 경우 아군의 체력을 회복시킨다. 우클릭은 순식간에 짧은 거리를 이동하는 이동기로, 사용 횟수가 제한되어 있으며 일정시간마다 남은 사용 횟수가 증가한다.

파괴된 함선은 일정 시간이 지난 뒤에 각자의 소환장소에서 다시 나타난다.

맵에는 고정된 행성들이 있어서 파괴되지 않고 부딪히는 모는 총알을 흡수한다.

각 함선의 크기를 1이라고 했을 경우, 맵의 크기는 250 X 250 이다.

**개발 환경**

Windows 64Bit, Microsoft Visual Studio 2015(VC++), DirectX 11, Box2D

**High Level Design**

Server와 Client의 framework는 상태기계이다.

Server의 상태

1. Ready : 플레이어들이 캐릭터와 팀을 고르고 채팅을 할 수 있는 상태. 2:2(각 팀 반드시 공격수 포함하도록) 또는 3:3이 되었을 때 방장(가장 먼저 들어온 플레이어. 나가면 다음 순으로)에게 start 버튼이 활성화 된다. 방장이 이 버튼을 누르면 게임 시작!
2. Play : 게임이 시작되고 종료될 때 까지의 상태. 실질적인 게임 로직이 이루어지는 부분.
3. End : 승패가 결정 난 상태. 승패를 보여주고 현재 접속한 Player들을 데리고 다시 Ready 상태로 돌아간다.

Client의 상태

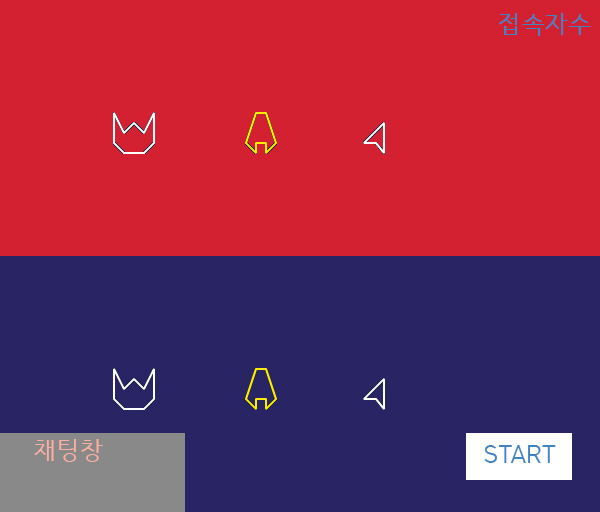
1. Login : IP(우리는 서버가 유동 IP를 사용하기 때문에), 아이디(8자이내)를 입력하고 로그인 버튼을 누르면, 서버의 상황에 따라 로그인 허가 또는 거절을 받는다.
2. Ready : 남아있는 캐릭터 중 선택하고 다른 플레이어와 채팅할 수 있다.
3. Play : wasd 키, 마우스 좌우버튼을 이용하여 캐릭터를 조종하여 게임을 진행할 수 있다.
4. End : 승패의 결과를 보고 나갈지 결정.

**SERVER**

Server와 Client는 TCP프로토콜을 이용해 통신한다. 이때 서버의 포트번호는 1130으로 정한다..

Listen과 물리 처리를 담당하는 main thread와 모든 Client에 대한 전용 thread를 갖는다.

1. Ready State



Server의 main thread에선 NWframework.Update()를 호출하여 listen socket이 Client의 접속을 받고 accept() 까지 호출 후 해당 클라이언트에게 접속이 허가 또는 거절 되었는지 send()로 알려준다. 거절한 경우 해당 Client 전용 socket을 닫는다. 허가 한 경우엔 NWframework.RegisterClient()에 Client 전용 socket을 넘기며 호출하여 접속한 Client의 전용 thread를 생성한다.

Client들의 정보는 Client 구조체 배열(길이 최대 6)로 관리한다. 이때 배열을 들어온 순서대로 유지한다. 따라서 항상 배열의 맨 앞의 Client가 방장이다. 배열의 중간에 위치하는 Player가 나오면 뒤의 데이터들을 당겨온다. 이 배열의 Index들은 Player Number로 Player를 구분하는 데 사용한다.

이제부턴 매프레임 Client 전용 thread 함수가 send(), recv()를 처리한다.

Ready 상태 Client 전용 thread 함수

{

클라이언트 ID 받는다.

해당 클라이언트 ID 정보 입력

클라이언트에게 다른 클라이언트들의 정보를 넘겨준다.

While(true){

고정 + 가변길이로 데이터를 받는다.

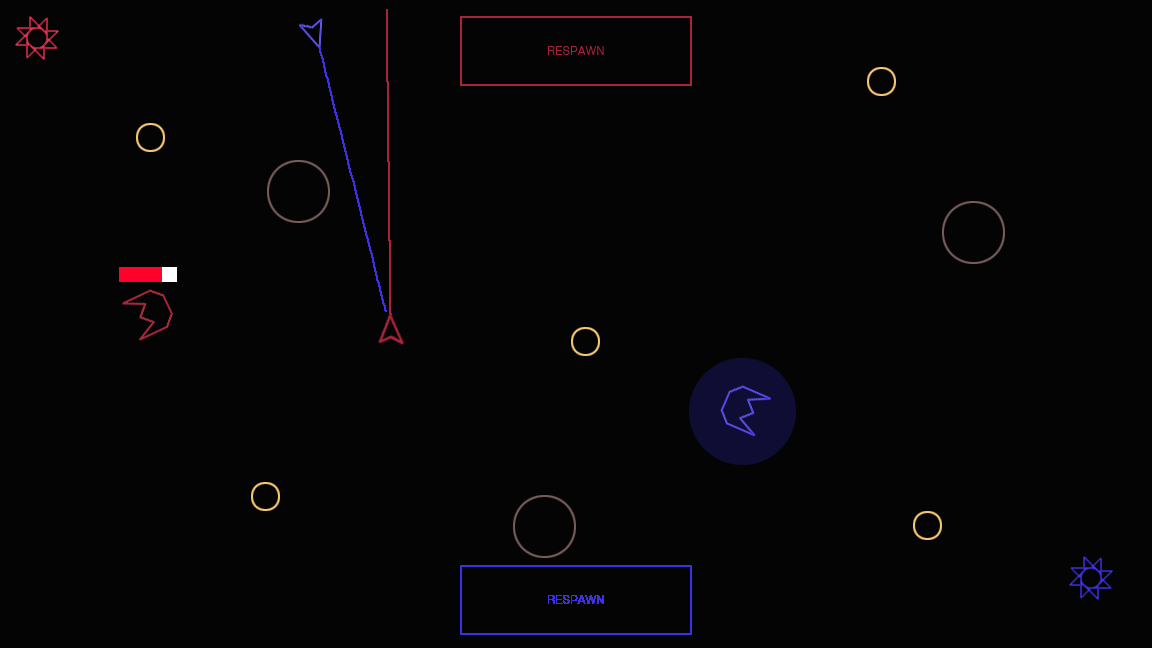
고정 + 가변길이로 데이터를 넘긴다.

}

}

Client 6명이 접속해 있거나 Play 상태가 되면 listen socket을 닫는다.

2. Play State



이제부턴 frame rate를 신경 쓰면서 로직을 만들어야 한다.

앞서 NWframework가 상태기계라 하였는데, 따라서 NWframework.Update() 호출 시 Ready상태와 다른 동작을 한다. Main thread에선 NWframework.Update()를 호출하여 서버에선 클라이언트들로부터 받은 사용자 입력 정보를 이용해 물리처리를 한다. 클라이언트 전용 스레드에선 매프레임 클라이언트와 데이터를 주고 받는다. 이때 클라이언트 전용 스레드 함수는 NWframework의 멤버함수로 상태에 따라 내부에서 다른 함수를 호출한다. 따라서 클라이언트와 데이터를 주고 받는 방식이 앞과 달라진다.

Play 상태 Client 전용 thread 함수

{

While(true){

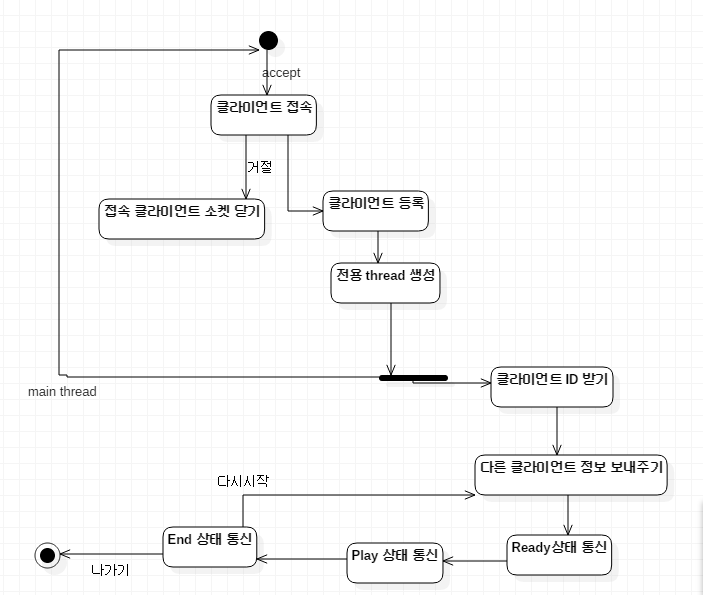
고정 데이터를 받는다.

고정 + 가변길이로 데이터를 넘긴다.

}

}

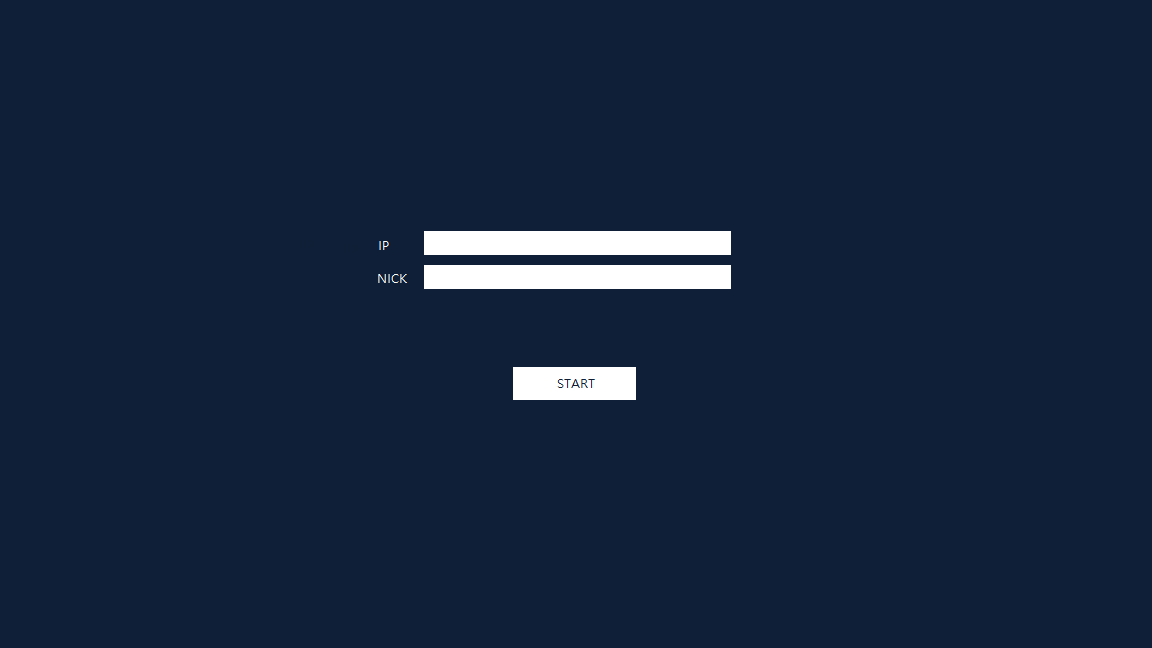
Activity Diagram



**CLIENT**

Main thread에서 렌더링을 담당하고, 네트워크와 통신 전용 thread를 따로 둔다.

1. Login State



8자 이내로 ID를 입력 받는다. 이때 ID 중복 허용한다. 따라서 서버에서는 클라이언트를 번호를 부여해서 구분한다.

사정상 서버 컴퓨터가 매번 바뀔 것 이기 때문에 IP주소 또한 따로 입력한다.

2. Ready State

서버에서 넘겨 받은 정보로 선택할 수 있는 캐릭터는 컬러로, 선택된 캐릭터는 회색으로 그린다. 다른 클라이언트들의 정보를 서버와 마찬가지로 배열에 유지한다. 또한 서버가 준 채팅데이터를 채팅창에 표시한다. 방장일 경우 클라이언트 쪽에서 조건에 따라 START 버튼 활성화 한다.

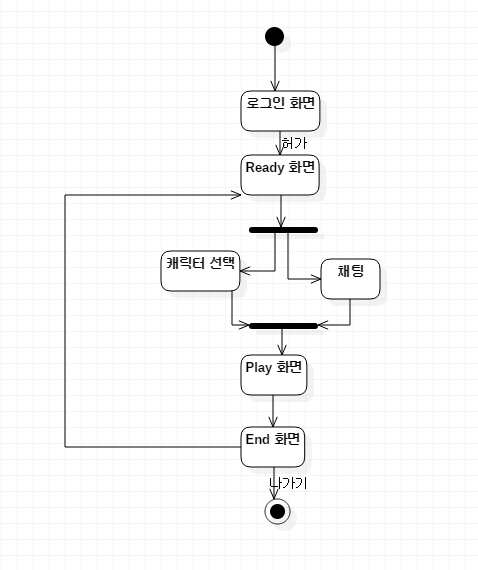
3. Play State

서버에 사용자 입력 정보를 넘긴다. 서버로부터 모든 플레이어의 정보, 총알 정보, 이펙트 정보 등을 받아서 렌더링하는 데 이용한다. 사실상 클라이언트는 정보 해석, 렌더링에 집중한다.

4. End State

서버로부터 승패 정보를 받아 화면에 표시한다. 이때 나가기를 선택할 수 있으며, 나가지 않으면 자동으로 Ready 상태로 돌아간다.

Activity Diagram



**APPLICATION PROTOCOL**

1. Ready State

명령어 + 데이터의 조합으로 보낸다. 명령어(크기 고정)에 따라 데이터 읽는 방식을 약속해 놓는다.

1.1 Client -> Server

명령어 Set

|  |  |  |
| --- | --- | --- |
| 명령어 | 데이터 | 비고 |
| CLIENT\_PICK | Pick character | enum으로 미리 약속 |
| CLIENT\_CHAT | Len, String |  |
| CLINET\_START |  | 방장만 보낼 수 있다 |

Client에선 한 프레임 안에서 발생한 명령어 집합의 길이를 고정길이로 먼저 보내고 가변길이로(명령어1 + 데이터1) + (명령어2 + 데이터2) + … 을 묶어서 보낸다. 그러면 Server에선 명령어(크기 고정)를 먼저 읽고 다음 데이터를 명령어에 따라서 읽는다.

1.2 Server -> Client

명령어 Set

|  |  |  |
| --- | --- | --- |
| 명령어 | 데이터 | 비고 |
| IN | ID | 들어온 순서대로 배열로 관리 |
| OUT | Player Number | 나가면 배열 재정렬 |
| SELECT | Player Number + character | 동시성 제어 필요 |
| CHAT | Player Number + len + string |  |
| START |  | 모든 플레이어에게 |

주고 받는 방법은 Client->Server 방법과 같다.

2. Play State

2.1 Client -> Server

사용자 입력에 대한 정보를 보낸다. 이때 데이터 길이는 고정되어 있다.

데이터는 회전각 + wasd Key Up/Down + 마우스 좌우 Button Up/Down으로 구성된다.

|  |  |
| --- | --- |
| 회전각 | 1 0 1 1 1 0 0 0 (‘1’은 Down ‘0’은 Up을 의미) |

w a s d L R

이때 회전각

theta를 라디안으로 넘긴다.

theta

2.2 Server -> Client

모든 Player 데이터 순서대로(위치, 회전각, 체력) + 모선 체력 + 총알 개수 + -상황에 따라 총알 데이터(Player Number–쏜 사람, 위치) + 레이저 데이터(Player Number + End Point) + Effect Data + Event Data- 를 보낸다.

보내는 데이터 블록

가변길이

고정길이

|  |  |
| --- | --- |
| 모든 Player 데이터 | |
| 모선 체력 | |
| 총알 개수 | |
| 총알 데이터 | |
| 레이저 명령어 | 레이저 데이터 |
| 이펙트 명령어 | 이펙트 데이터 |
| 이벤트 명령어 | 이벤트 데이터 |

이 규칙에 맞게 데이터를 보내고 클라이언트는 이 규칙에 맞게 데이터를 해석한다.

Play State 명령어 Set

|  |  |  |
| --- | --- | --- |
| 명령어 | 데이터 | 비고 |
| LASER | Player Number, End Point |  |
| EFFECT | Player Number,EFFECT TYPE |  |
| OUT | Player Number | 중간에 다른 플레이어가 나갔을 경우 |

**LOW LEVEL**

**서버 주요 모듈 디자인**

- ServerFramework

서버의 핵심 클래스로 클라이언트 접속 처리, 클라이언트 요구 처리, 물리처리 등을 담당한다.

■ Operation

void Initiate()

윈속 초기화 -> 대기 소켓 생성, 바인드 및 대기 상태로 설정

void Update()

내부에서 상태에 따라 다른 함수 호출한다.

- Ready State

클라이언트 Accept -> 현재 접속자 수에 따라 허가, 거절 메시지 보내기

->(if 거절일 경우 해당 클라이언트 전용 소켓 닫기)

-> (if 방장이 Start 버튼을 누르면) 대기 소켓 닫기

- Play State

물리 처리 함수 호출

- End State

나가는 클라이언트 소켓 닫기

void RegisterClient(socket client\_sock)

Client배열에 새로운 Client추가, 해당 Client와 데이터를 주고받을 전용 thread 생성

void DeleteClient(unsigned int PlayerNum)

현제 접속자수 수정 및 배열 재정렬, 해당 클라이언트 소켓 닫기

DWORD WINAPI ProcessClient(unsigned int PlayerNum)

클라이언트와 데이터를 주고받는 전용 thread 함수.

이 안에서 상태에 따라 호출하는 함수 다르게 동작.

Ready State일 때 처음 이 함수가 호출 되면 클라이언트 ID를 받고 저장하고 클라이언트한테 다른 클라이언트 들의 정보를 넘겨준다. 그 다음 while문 안에서 다음의 함수를 호출한다.

DWORD ReadyStateProcessClient(unsigned int PlayerNum)

{

고정길이로 클라이언트가 보낼 데이터 크기를 받는다.

크기 > 0 이면 데이터 받는다.

받아서 Ready State 명령어 Set에 따라 데이터를 읽고 처리한다.

|  |  |
| --- | --- |
| 명령어 | 처리 방법 |
| CLIENT\_PICK | ServerFramework내에 각 ClientThread 전용 명령어 Queue를 가지고 있는데, 이 Queue는 Critical Section으로 동기화되는 공유메모리이다. 이 명령어 Queue는 한 프레임 동안에 발생한 일들 중 해당 클라이언트가 알아야 하는 명령어와 데이터들을 모아 놓는다.  자신을 제외한 다른 클라이언트들의 명령어 Queue에 SELECT 정보를 넣는다.  또한 ServerFramework내의 Client 배열에서 Pick 정보를 입력한다. |
| CLIENT\_CHAT | 위와 비슷하다. |
| CLINET\_START | 위와 비슷하며, ServerFramework의 상태를 바꾼다. |

해당 클라이언트의 명령어 Queue의 길이를 보내고, 명령어 Queue의 데이터를 보낸다.

}

DWORD PlayStateProcessClient(unsigned int PlayerNum)

{

사용자 입력 정보를 고정길이로 받는다.

물리 처리 thread가 물리처리 공유메모리를 사용하는지 확인한다. (Event)

물리 처리 정보를 공유메모리의 해당 클라이언트의 영역에 입력한다.

물리처리 thread가 물리처리를 완료할 때까지 기다린다. (Event)

이때 물리 처리는 최대 45tick rate를 유지하도록 타이머를 설정한다.

물리처리 결과 및 다른 플레이어 정보 등을 보낸다.

}

DWORD EndStateProcessClient(unsigned int PlayerNum)

해당 클라이언트가 나가면 소켓 닫고 배열 재정렬.

■ Data

배열 데이터 구조가 많은데 Player Number가 인덱스 역할을 하며 공유메모리에 대한 동시성 제어를 최소화 하기 위한 방법이다.

ClientData[6]

ClientThreadCmdQueue\*[6]

PhysicalInput[6]

Timer

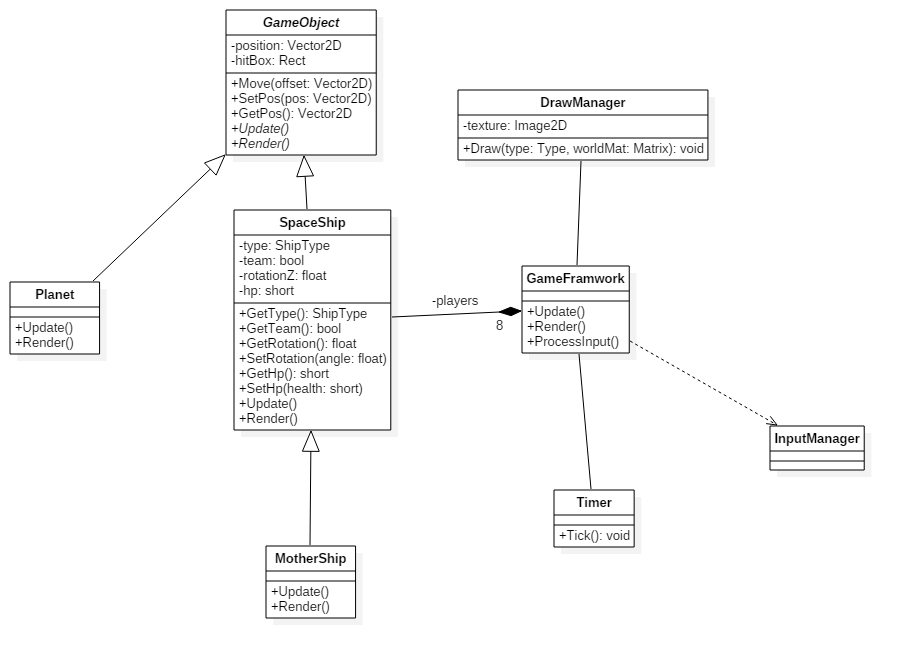
Client Number

BulletData Array

MotherShipData[2]

**클라이언트 주요 모듈 디자인**

Class Diagram



공유 메모리 : ClientData[6], BulletData Array, LaserData[2], MotherShipData[2]

- ClientFramework

프레임워크에서는 렌더링에 집중한다. 데이터를 주고 받는 thread가 따로 존재하는 데 여기서 공유 메모리가 존재하게 되므로 Critical Section을 이용한다.

■ Operation

void ProcessInput()

서버한테 보낼 사용자 입력 정보를 명령어 + 데이터 구조의 집합으로 묶는다.

void Update()

ProcessInput()을 호출하며 클라이언트 단 게임 로직-방장이면 Start 조건확인-을 수행한다. 게임이 최대 45fps가 되도록 타이머 설정.

void Render()

서버로부터 받은 클라이언트 데이터 등을 Critical Section Enter를 이용 지금 통신에 사용하고 있지 않으면 그리는데 사용한다.

- DWORD WINAPI ProcessNW(socket)

상태에 따라 호출하는 함수 달라지도록

서버한테 상태에 맞는 데이터를 보내고 받은 데이터를 해석해서 공유메모리에 Critical Section Enter를 이용하여 동시성 제어를 걸고 갱신한다.

**팀원 별 역할분담**

|  |  |
| --- | --- |
| 재용 | 소윤 |
| Login 화면  Ready State Client  Play State Server  물리처리 | Ready State Server  Play State Client  End State Server/Client  렌더링 |

**개발 일정**

