네트워크 게임 프로그래밍 추진계획서

# 1. 개요

• **기존 게임: *방과 후 축구한판* (컴퓨터 그래픽스 / 임성훈)  
• 서버 환경: Windows Server  
• 통신 방식: TCP/IP  
• 개발 환경: Visual Studio, C++, OpenGL**

# 2. 애플리케이션 기획

**방과 후 축구 한 판은 3인이 플레이할 수 있는 축구게임**으로서 각 플레이어는 물리 법칙이 존재하는 축구장에서 축구공을 찰 수 있으며, 드리블, 슈팅이 가능하다. (골키퍼는 AI)

게임 시작 전 **회원가입 그리고 로그인**을 해야 한다. **(ID/PASSWORD)**잔디, 야외, 들판, 식물이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

**• 맵(World) 개요:  
상호작용하는 오브젝트는 총 5가지** 정도로  
**플레이어, 축구공, 축구장(큰 배경공간), 골대, 골키퍼**가 있다.

골을 넣은 플레이어에게 점수가 1점이 기록된다. 게임을 시작하고 5분이 지나면 게임은 종료되고, **승패판정 후 경기결과**가 나온다. 이후 로그인 정보를 통해 경기 전적이 기록되어, 게임에서 **경기 전적**이 보이게끔 설정한다.

게임 인원은 최대 3명으로 방을 하나 만들고3명이 들어오면 게임을 시작한다.

게임 도중 **채팅 기능**을 사용할 수 있다.

• 플레이어(Player) 개요:조작키는 다음과 같다.

방향키: 이동  
E: 달리기  
D: 슛  
좌우 방향키와 슈팅을 조합하면 골대의 왼쪽 부분, 오른쪽 부분으로 찬다.  
Z와 조합하면 기본 슛이 아닌 감아차기가 나간다. (z, y 뿐 아니라 x도 바뀐다.)  
C와 조합하면 조금 더 낮고 강력한 슈팅을 발사한다.

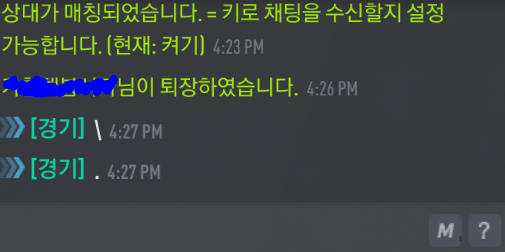
**SPACE: 태클  
플레이어와 다른 플레이어의 위치, 공 소유 등을 계산 후 조건에 따라 공을 뺏을 수 있도록 한다.**



**Enter: 채팅 기능 사용  
채팅 UI를 제작하여 서로 채팅을 쓰고 볼 수 있도록 한다.**

|  |
| --- |
| **Object** |
| **glm::vec3 position{};**  **glm::vec3 rotation{};** |
| **Object();**  **~Object();** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Player** | **Ball** | **Keeper** | **Goalpost** | **Camera** | **Light** |
| **vec3 velocity, direction,**  **float acceleration**  **bool hasball,[[1]](#footnote-1) keystates[][[2]](#footnote-2)** | **vec3 velocity, direction,**  **float acceleration**  **float rotationspeed** | **vec3 velocity**  **vec3 movement\_range[[3]](#footnote-3)**  **bool hasball** |  | **vec3 direction**  **vec3 up** | **vec3 color**  **vec3 viewpos**  **float ambientlight** |
| **Player(x, y, z)**  **~Player()**  **Move(), Draw()**  **Sprint(), Shoot() , Tackle()** | **Ball(x, y, z)**  **~Ball()**  **Move(), Draw()** | **Keeper(x, y, z)**  **~Keeper()**  **Move(), Draw()**  **Movement()** | **GoalPost(),**  **~GoalPost()**  **Draw()** | **Camera()**  **~Camera()** | **Light()**  **~Light()** |

****

|  |
| --- |
| **BackGround** |
|  |
| **BackGround();**  **~BackGround()**  **Draw()** |

# 3. 클래스 개요

|  |  |
| --- | --- |
| **Grond** | **Wall** |
|  |  |
| **Ground()**  **~Ground()**  **Draw()** | **Wall()**  **~Wall()**  **Draw()** |

# 4. High Level Design

•**프로그램 흐름도  
게임 시작 전 로그인 흐름**

FileSystem

F

2 ready?

connect

send(ID/PW)

recv

accept

Send(T/F, P1/2)

Listening

T

send(ready)

recv

Login?

F

T

MakeSession

recv

**게임 시작 후 흐름**

AI Goal keeper

recv\_Input

key apply

Ball Calculate

Collision System

Input

Update Score  
Position Reset

GameSession  
Destroy

Goal?

F

T

GameOver  
Win/Lose

Rendering

End?

F

T

Data send

UI Update  
Close

**멀티 쓰레드 흐름**

New data

Rendering(new data or lastest data)

re

Packet  
Type?

**ProcessClient(1)  
ProcessClient(2)  
ProcessClient(3)**

**rocessClient(1)**

If tick

GameSessionLoop

Calculation

Send?

Return and send data

Save

ServerReceiverThread

Game Loop

Event(Input)

F

F

F

Gameover?

End

End

Gameover?

ClientReceiverThread()  
GameSessionThread()

Server

Client

• 클라이언트 ↔ 서버 전송 과정:

**1: 접속 및 인증 (Login)**

1. **[Client]** **Connect** **[Server]**
   * 클라이언트는 서버의 고정 IP와 포트(예: 127.0.0.1:7777)로 TCP연결요청
2. **[Server]** **Accept** **[Client]**
   * 서버(메인 스레드)가 연결을 수락하고, 이 클라이언트를 담당할 스레드(Client Handler)를 생성
3. **[Client]** **Req\_Login** ({ID, PW}) **[Server]**
   * 클라이언트가 로그인 UI에서 ID/PW를 입력받아 서버로 전송합니다.
4. **[Server: Process]**
   * Req\_Login 패킷을 수신
   * 파일 시스템을 조회하여 ID/PW 일치 여부를 확인하고 전적 데이터(W/L)를 로드합니다.
5. **[Server]** **Res\_Login** ({Success: True, Wins: 10, Losses: 5}) **[Client]**
   * 로그인 결과를 전적 데이터와 함께 클라이언트에 전송
   * 서버는 이 클라이언트의 상태를 **[InLobby]** (로비 진입)로 변경합니다.
6. **[Client: Process]**
   * Res\_Login 패킷을 수신합니다.
   * (Success: True) 로비 씬(Scene)으로 전환하고, 전적을 UI에 표시
   * (Success: False"로그인 실패" 메시지를 표시

**2: 매치메이킹 (Matchmaking)**

1. **[Client]** **Req\_Ready** **[Server]**
   * 플레이어가 로그인 한 순간 Req\_Ready 상태로 전환
2. **[Server: Process]**
   * Req\_Ready 패킷을 수신합니다.
   * 이 클라이언트(P1)의 상태를 **[Waiting]** (매칭 대기)으로 변경하고 \*\*매칭 대기열(Queue)\*\*에 추가합니다.
   * *(잠시 후, 다른 클라이언트(P2)도 1~2 과정을 거쳐 Waiting 상태로 대기열에 추가됩니다.)*
3. **[Server: Match Logic]**
   * 서버의 메인 로직이 대기열에 2명(P1, P2)이 있는 것을 확인합니다.
   * 대기열에서 P1, P2를 꺼냅니다.
   * GameSession 객체를 생성하고, 이 세션을 담당할 \*\*새 스레드(Game Loop Thread)\*\*를 시작시킵니다.
   * P1, P2의 상태를 \*\*[InGame]\*\*으로 변경하고, 이들이 속한 GameSession을 지정해줍니다.
4. **[Server]** **Notify\_MatchStart** **[Client P1]**
5. **[Server]** **Notify\_MatchStart [Client P2]**
   * 매칭된 두 클라이언트에게 "게임 시작!" 신호를 동시에 전송
6. **[Client P1, P2: Process]**
   * Notify\_MatchStart 패킷을 수신합니다.
   * 즉시 로비 씬을 닫고, 인게임(축구장) 씬을 로드합니다.
   * \*\*[In-Game Loop]\*\*를 시작합니다.

**3: 인게임 루프 (In-Game Loop)**

**클라이언트 측 루프 (Client Loop)**

(매 프레임 실행)

1. **[Process]** Input: 플레이어의 현재 키보드 상태(눌린 키, 떼어진 키)를 감지
2. **[Client]** **Client\_Input** ({W:on, D:off, Z:on...}) **[Server]**
   * 감지한 키 상태를 서버에 전송
3. **[Process]** Recv: 서버로부터 Server\_GameState 패킷이 올 때까지 대기(수신)
4. **[Process]** Rendering: **수신한 Server\_GameState 데이터**를 바탕으로 P1, P2, 공, 골키퍼의 위치와 회전값을 화면에 **강제로 덮어씌웁니다.**
5. *(1번으로 돌아가 반복)*

**서버 측 루프 (GameSession Loop)**

(서버의 고정 Tick Rate, 예: 1초에 60회 실행)

1. **[Process]** Recv Inputs: P1, P2가 보낸 Client\_Input 패킷 중 가장 최신 것을 수집
2. **[Process]** Logic: 수집된 입력을 서버 내의 P1, P2 캐릭터에 적용
3. **[Process]** AI: 골키퍼 AI 로직을 실행
4. **[Process]** Physics: **물리 엔진을 1 스텝 실행**하여 모든 충돌과 움직임을 계산
5. **[Process]** Rules:
   * (Goal?) 점수 갱신, 위치 리셋 P1, P2에게 **Notify\_Goal** 패킷 즉시 전송.
   * (End == 0?) **[루프 종료]** (4로 이동)
6. **[Server]** **Server\_GameState** ({All Positions, Score, Time...}) **[Client P1]**
7. **[Server]** **Server\_GameState** ({All Positions, Score, Time...}) **[Client P2]**
   * 물리 연산이 끝난 '최종 결과'를 모든 클라이언트에게 전송합니다.
8. *(1번으로 돌아가 반복)*

**4: 게임 종료 (Game Over)**

1. **[Server: Process]**
   * (3의 루프가 종료됩니다.)
   * 최종 스코어 기준으로 승패를 판정
   * 파일 시스템을 열어 P1, P2의 전적(W/L)을 업데이트합니다.
2. **[Server]** **Notify\_GameOver** ({Result: Win, Score: 3:1}) **[Client P1]**
3. **[Server]** **Notify\_GameOver** ({Result: Lose, Score: 1:3}) **[Client P2]**
   * 두 클라이언트에게 최종 결과를 전송
4. **[Client: Process]**
   * Notify\_GameOver 패킷을 수신
   * 인게임 루프를 중지하고 "승리/패배" 결과 창을 표시
5. **[Server]** **Close Connection**
   * 서버가 P1, P2와의 TCP 연결을 종료
   * 서버에서 GameSession 객체와 스레드를 파괴(Destroy)
6. **[Client: Process]**
   * 서버와의 연결이 끊어진 것을 감지
   * 자동으로 최초의 '로그인 씬'으로 화면 전환

# 5. Low Level Design

<프로토콜 설계>  
• Server :

|  |  |  | **이름** | **데이터 (Payload)** | **목적 (Flowchart 매핑)** | |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  | recv\_IDPW | struct PacketLogin | 클라이언트로부터 ID와 PASSWORD값을 받음. | |
|  |  |  | ClientReceiverThread() |  | 클라이언트 접속을 각각의 스레드로 만듦 | |
|  |  |  | LoginResponse | bool bSuccess (1:성공, 0:실패)  int PlayerID, int Win, int Loss (성공 시)  string Message (실패 사유) | 로그인 요청에 대한 결과 응답. 5 | |
|  |  |  | send\_login | struct PacketLoginResult | Login Response에서 반환된 bool 값을 클라이언트에 전송 | |
|  |  |  | recv\_ready | struct PacketGameReady | 로그인에 성공한 클라이언트에서 받은 ready값을 반환 | |
|  |  |  | 3ready | bool P1\_ready, P2\_ready, P3\_ready | 세 플레이어에게 true값을 동시에 받을 때까지 ready값을 받는 것을 반복 | |
|  |  |  | MatchFound | string OpponentID, int OpponentWins, int OpponentLosses | | 매치가 성사되었음을 알리고, 접속한 상대 정보를 전달. |
|  |  |  | MainThread |  | | 3플레이어가 접속했다면 플레이어가 게임 루프를 실행단계로 진입 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  | LoadGame | Vector P1\_StartPos, Vector P2\_StartPos, Vector P3 StartPos | | 클라이언트에게 게임 맵 로드 및 오브젝트 초기 배치를 명령. |
|  |  |  | BroadcastChat | string SenderID, string Message | | 채팅 메시지를 상대방에게 중계. 6 |
|  |  |  | recv\_input | int P1/P2\_keystates | | 클라이언트에서 받은 input 값을 반환 |
|  |  |  | GameStateUpdate | uint SequenceNum Vector P1\_Pos, Vector P2\_Pos, Vector Ball\_Pos, Quaternion Ball\_Rot Vector GK1\_Pos, int ScoreP1, int ScoreP2 int SecondsRemaining | | 서버가 물리/AI 연산을 마친 모든 동적 오브젝트의 최종 상태를 전송. Goal? 분기 로직을 반영하여, 점수와 시간을 이 패킷에 포함합니다. 클라이언트는 이 패킷을 받아 점수판(UI)을 갱신하고, 점수 변경을 감지하여 "골" 사운드를 재생합니다. (Update Score 로직 포함) 공과 플레이어의 위치도 골 직후 리셋된 위치가 포함되어 전송됩니다. |
|  |  |  | send\_renderdata | struct PacketPlayerData | | 모든 연산 후 클라이언트에서 렌더링을 위한 데이터를 보내줌 |
|  |  |  | Player::Tackle() | uint SequenceNum Vector P1\_Pos, Vector P2\_Pos, Vector Ball\_Pos | | 플레이어가 다른 플레이어의 공을 뺏을 수 있도록 기능 추가. |
|  |  |  | GameOver | time\_t start, end | | 게임 종료 조건(time)이 만족하면 게임을 종료 |
|  |  |  | send\_gameover | | struct PacketGameover | 게임이 끝나면 클라이언트에게 알림. |
|  |  |  | Update Score | | string WinnerID  int FinalScoreP1, FinalScoreP2 | 게임 종료 후 경기 스코어에 따라 전적을 수정 |
|  |  |  | send\_score | | struct PacketUserData | 수정한 전적을 파일 시스템에게 보냄. |
|  |  |  | ReturnToLogin | | int NewWins, int NewLosses | 서버가 전적 파일 저장을 완료했음을 알리고, 로그인 씬으로 돌아가도록 명령. |
|  |  |  |

• Client :

| **이름** | **데이터 (Payload)** | **목적 (Flowchart 매핑)** |
| --- | --- | --- |
| LoginRequest | string ID, string Password | 회원가입 또는 로그인을 요청. |
| send\_IDPW | struct PacketLogin | 받은 ID와 PW값을 서버에 전송 |
| recv\_login | struct PacketLoginResult | 서버에서 받은 bool 값을 반환 |
| login | string ID, string Password | recv\_login에서 true를 받을 때까지 로그인 과정 반복 |
| send\_ready | struct PacketGameReady | bool 변수 ready값이 변화할 때마다 서버에 전송 |
| PlayerChat | string Message (채팅 내용) | ENTER키 입력 시 채팅을 입력시킬 수 있는 UI가 나오고 채팅 메시지를 입력할 수 있음. |
| send\_chat | string Message (채팅 내용) | 입력된 채팅 메시지를 server에 보냄 |
| PlayerInput | struct PacketInputkey | 플레이어의 현재 키보드 입력 상태를 압축하여 전송. |
| recv\_gameover | struct PacketGameover | 서버로부터 Game이 끝났다는 bool값을 받음 |
| UI Update Close() |  | GameoverUI를 보여주며 로그인 화면으로 돌아갈 수 있도록 함 |
| ServerReciverThread() |  | 서버에서 오는 데이터를 받기 위한 스레드 생성 함수 |
| DWORD WINAPI ProcessClient(LPVOID arg) {   1. 필요 변수 선언(예: int retval, SOCKET client\_sock, PacketHeader header 등등), 2. 클라이언트 정보 얻기 3. 데이터 받기,   예시)while(1)  retval = recv(client\_sock, (char\*)&header, sizeof(PacketHeader), MSG\_WAITALL);  switch (header.type )  case PKT\_INPUT:  PacketInputkey key; key 1 = recv\_Input(key); if …   1. 최신 데이터 저장 데이터를 담을 수 있는 전역변수(예: PacketInputKey key[])에 저장. 저장하기 전 lock을 걸어야함.   } | | |
| void GameSessionLoop() {  while(1)   1. Sleep until next tick 2. Lock을 걸고 연산 시작 3. 먼저 저장된 최신 데이터를 가져옴. (ProcessClient에서 받은 데이터를 저장한 전역변수. 만약 데이터가 없다면 예외 처리) 4. 저장된 최신 데이터로 연산   GameStateUpdate(key1, key2, key3)   1. 다음 틱을 위해 저장된 최신 데이터를 초기화.(초기화 한 후 lock 해제) 2. 데이터 보내기 send\_rederdata(client\_sock1) send\_rederdata(client\_sock2) send\_rederdata(client\_sock3)   } | | |

## <패킷 설계 예시>

**서버 -> 클라이언트**

struct PacketHeader {  
uint16\_t type; 패킷 종류 (ex. 로그인, 유저데이터, 키입력 등)  
uint16\_t size; 전체 패킷 크기  
};

struct PacketLoginResult {  
PacketHeader header;  
uint8\_t success;  
char message[64]; // 실패 시 이유 or 성공 메시지  
};

struct PacketUserData {  
PacketHeader header;  
UserData data;  
};

struct UserData {  
int totalMatch;  
int win;  
int lose;  
float winRate;  
};

struct PacketRenderData {  
PacketHeader header;  
PlayerData p\_data;  
BallData b\_data;  
KeeperData k\_data;  
GoalPostData g\_data;  
CameraData c\_data;  
LightData l\_data;  
BackGroundData bg\_data;  
};

struct LightData {  
vec3 position;  
vec3 viewpos;  
vec3 color;  
float ambient;  
클래스 개요 참고  
};

struct KeeperData {  
vec3 position;  
vec3 rotation;  
vec3 velocity;  
클래스 개요 참고  
};

struct GoalPostData {  
vec3 position;  
vec3 rotation;  
};

struct BallData {  
vec3 position;  
vec3 rotation;  
vec3 velocity;  
클래스 개요 참고  
};

struct PlayerData {  
int id;  
vec3 position;  
vec3 rotation;  
vec3 velocity;  
클래스 개요 참고  
};

struct PacketGameover {  
PacketHeader header;  
uint8\_t gameover;  
};

struct CameraData {  
vec3 position;  
vec3 direction;  
vec3 up;  
클래스 개요 참고  
};

**클라이언트 -> 서버**

struct PacketInputspeicialkey {  
PacketHeader header;  
uint32\_t specialkey;  
};

struct PacketInputkey {  
PacketHeader header;  
uint32\_t key;  
};

struct PacketLogin {  
PacketHeader header;  
char userID[32];  
char userPW[32];  
};

struct PacketGameReady {  
PacketHeader header;  
uint8\_t ready  
};

struct PacketChatMessage {  
PacketHeader header;  
string Message;  
};

# 6. 역할분담

**- A: 김용채**  
서버 통신, 네트워크 동기화

**- B: 박지성**  
클라이언트 UI, 멀티 스레드, 데이터 관리(filesystem)

**- C: 임성훈**  
채팅 UI, 게임 로직 구현, 패킷 구조 설계

# 7. 개발 일정

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | SUN | MON | TUE | WED | THU | FRI | SAT |
| **10** | 26 | 27 | 28 | 29 | 30 | 31 | 1 |
| **11** | 2  connect(), listening(), accept() | 3 | 4 | 5 | 6  ProcessClient()  ProcessServer() | 7 | 8 |
| 9  recv\_input() send\_renderdata() | 10 | 11 | 12 | 13 | 14 | 15  1차토의 피드백 후Project Progress Report 작성 |
| 16 | 17 | 18 | 19 | 20 | 21 | 22 |
| 23  send\_IDPW(), recv\_IDPW()  send\_login()  recv\_login() | 24 | 25 | 26 | 27 | 28 | 29 2차토의 |
| **12** | 30  send\_ready(), recv\_ready(), 2ready? | 1 | 2 | 3 | 4 | 5 | 6 3차토의 |
| 7 | 8 | 9 | 10  최종 테스트 및 결과물 제출 | 11 | 12 | 13 |

**A: 김용채**

**B: 박지성**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | SUN | MON | TUE | WED | THU | FRI | SAT |
| **10** | 26 |  |  |  |  |  | 1 |
| **11** | 2 |  | 4 | 5  ClientReveiveThread() ServerReceiveThread() | 6 | 7 | 8 |
| 9 | 10 | 11 MainThread() | 12 | 13 | 14 GameSessionLoop() | 15 1차토의 피드백 후Project Progress Report 작성 |
| 16 | 17 ProcessGameStateUpdate() | 18 | 19 LoadGameSceneUI() | 20 | 21  UpdateInGameUI() | 22 |
| 23 | 3 ValidateLogin() | 28 | 29  LoadLoginScene(); | 30 | 31  RegisterUser() | 29 2차토의 |
| **12** | 30 | 1 OnClick\_LoginScene () | 2 | 3 | 4 | 5 | 6 3차토의 |
| 7 | 8 | 9 | 10  최종 테스트 및 결과물 제출 | 11 | 12 | 13 |

**C: 임성훈**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | SUN | MON | TUE | WED | THU | FRI | SAT |
| **10** | 26 | 27 | 28 | 29 | 30 기존Code  Client/Server 나누기 | 31 packet 선언 | 1  PlayerInput() |
| **11** | 2 | 3 | 4 Gameover() | 5 | 6 | 7 | 8  send\_gameover()  recv\_gameover() |
| 9 | 10 | 11 | 12  Tackle() | 13 | 14  UI Update Close() | 15 1차토의 피드백 후Project Progress Report 작성 |
| 16 send\_score() | 17 | 18 | 19  PlayerChat() | 20 | 21 | 22  Update Score() |
| 23 | 24 | 25 | 26 send\_chat() | 27 | 28 BroadcastChat() | 29  2차토의 |
| **12** | 30 ChatUI() | 1 | 2 | 3 | 4 | 5 | 6 3차토의 |
| 7 | 8 | 9 | 10  최종 테스트 및 결과물 제출 | 11 | 12 | 13 |

1. hasball: **볼 소유권** [↑](#footnote-ref-1)
2. keystates[]: **조작키 입력 값** [↑](#footnote-ref-2)
3. movement\_range: **골키퍼 움직임 반경** [↑](#footnote-ref-3)