

네트워크 게임 프로그래밍(01)

Team 프로젝트

# 추진 계획서

ver. 3



---

팀원

---

이태현

---

윤혜린

---

지민우

---

# 목차

<b>1. 개요</b>	4
<b>1.1 게임 제목</b>	4
<b>1.2 게임 설명</b>	4
<b>1.3 개발자</b>	4
<b>1.4 사용 버전 관리 프로그램</b>	4
<b>1.5 작업 IDE</b>	4
<b>1.6 사용 프로토콜</b>	4
<b>2. 애플리케이션 기획</b>	5
<b>2.1 게임 요소</b>	5
<b>2.1.1 조작키</b>	5
<b>2.1.2 게임 플레이 방식</b>	5
<b>2.1.3 라운드 별 맵 사진 첨부</b>	6
<b>2.1.4 기존 게임에서의 수정 사항</b>	7
<b>2.1.5 시간 여유 시 추가할 게임 요소</b>	7
<b>3. High-Level Design</b>	8
<b>3.1 Main Game Flow</b>	8
<b>3.2 Start Flow</b>	9
<b>3.3 Collision Flow</b>	10
<b>3.4 Each Round Score Flow</b>	10
<b>3.5 Total Score Sum Flow</b>	11
<b>4. Low-Level Design</b>	12
<b>4.1 데이터 흐름 개요</b>	12
<b>4.2 공유 변수</b>	12
<b>4.3 구조체 선언</b>	12

4.3 네트워크 함수 API.....	15
4.3.1 클라이언트API .....	15
4.3.2 서버 API.....	17
5. 팀원별 역할 분담.....	21
6. 일정표.....	23

# 1. 개요

## 1.1 게임 제목

주차의 달인

## 1.2 게임 설명

4명의 플레이어들이 3라운드에 걸쳐서 누가 먼저 빠르게 주차를 하는지 경쟁을 하는 4인 멀티플레이 게임

플레이어가 직접 기어, 브레이크, 가속 페달, 핸들을 조작하여 차를 움직여 장애물에 최대한 부딪히지 않고, 스테이지마다 정해진 주차 공간에 주차를 성공하면 스테이지를 클리어하는 게임

3인칭 뷰로 차를 내려다보는 시점을 제공(차량의 위치, 주차 구역을 명확하게 볼 수 있다.)

3인칭 뷰에서 마우스 좌클릭 드래그를 통해 시점을 돌릴 수 있다.

## 1.3 개발자

2024-2 컴퓨터 그래픽스 최종 프로젝트 개발 (지민우, 조성욱)

## 1.4 사용 버전 관리 프로그램

GitHub

## 1.5 작업 IDE

Visual Studio

## 1.6 사용 프로토콜

1. 차 위치/이동 데이터/충돌 판정/게임 진행 명령 UDP사용

이유: 플레이어가 플레이할 때 차량 반응이 즉각적이어야 함, 빠른 응답이 중요

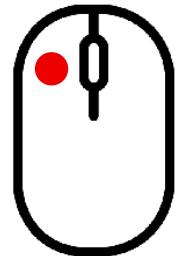
2. 로비, 매치, 결과 저장은 TCP 사용

이유: 반드시 정확히 도착해야 하는 정보들은 신뢰성을 보장하는 TCP를 사용한다.

## 2. 애플리케이션 기획

### 2.1 게임 요소

#### 2.1.1 조작키



**W:** 가속    **SPACE:** 브레이크    **Q/E:** 기어 조작 (위/아래)    **ESC:** 일시정지

(핸들 내부) **MLB 드래그:** 핸들 조작 (바퀴 회전)

(핸들 외부) **MLB 드래그:** 카메라 시점 회전

#### 2.1.2 게임 플레이 방식

총 3라운드로 이루어진 갈수록 주차 난이도가 상승하는 구조

시간: 주차하는데 걸린 시간에 따라 점수 부여

충돌: 주차하면서 충돌(벽, 장애물, 다른 플레이어)한 횟수에 따라 점수 감소

다른 플레이어와 충돌할 시, 충돌이 일어난 차량들은 점수 감점

ex) 3명의 차량이 서로 충돌 시, 3명 다 점수 감점

충돌 여부 판단: 충돌을 유발한 가해자 플레이어를 감점

라운드 클리어: 정해진 주차 구역으로 차를 이동하여 기어를 Parking으로 변

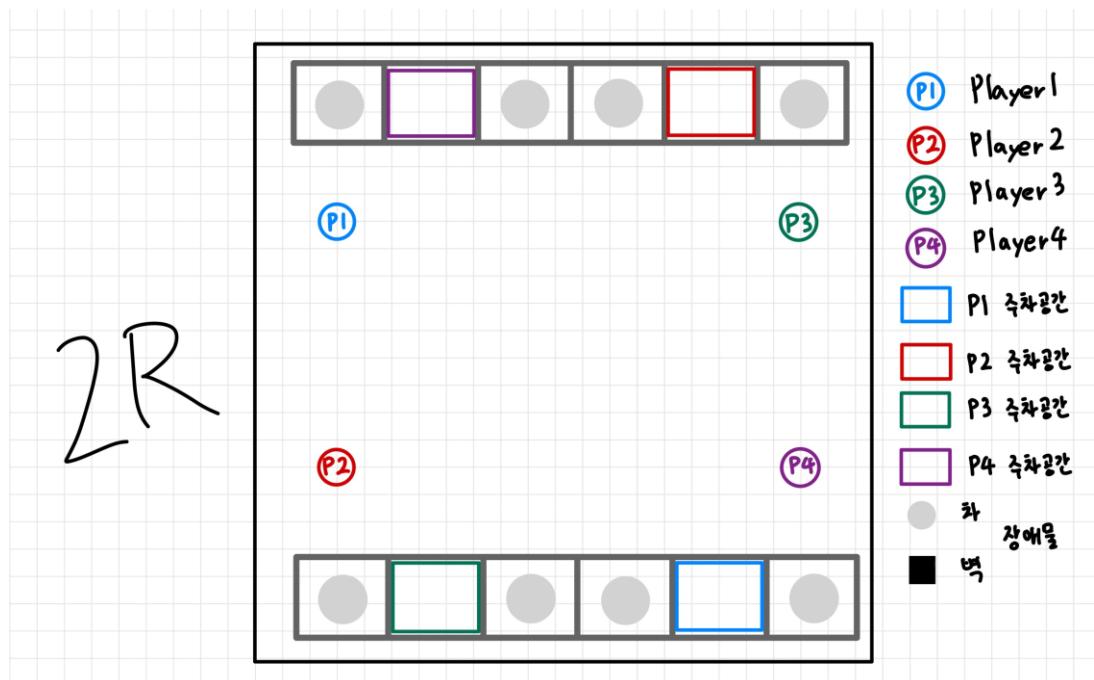
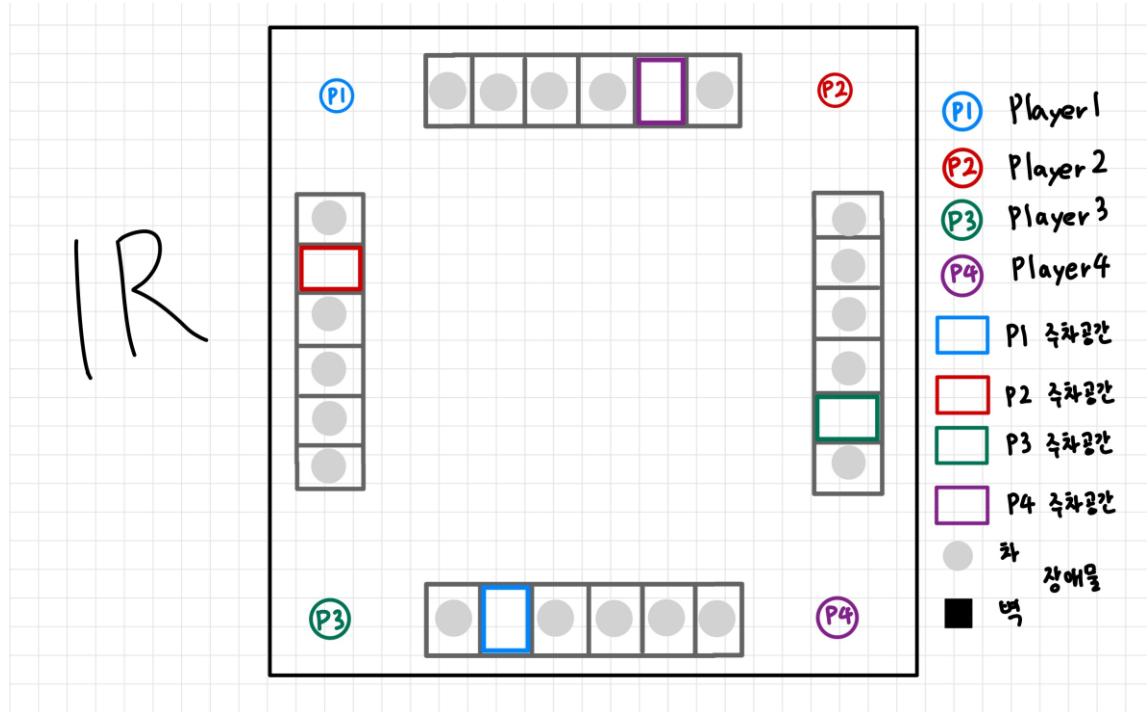
경 시 주차 성공, 모든 플레이어의 주차가 끝난 후 점수를 계산  
하여 더 높은 점수를 가진 플레이어가 승리

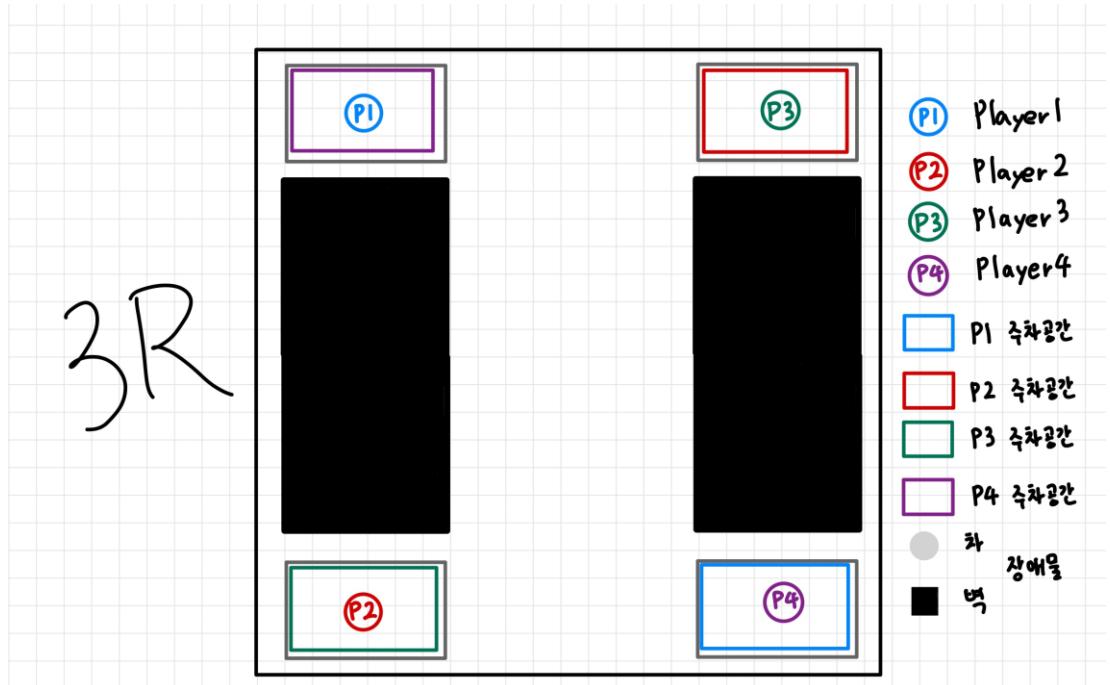
라운드 별로 정해진 시간 내에 주차 완료해야 한다.

최종 승리 판정: 라운드 난이도 별로 점수 가중치를 다르게 주어 점수 계산

후 3라운드 합산하여 최종 점수가 가장 높은 사람이 최종 승  
리자

### 2.1.3 라운드 별 맵 사진 첨부





#### 2.1.4 기존 게임에서의 수정 사항

[수정 전] 후진 시 후면 카메라 전체 화면

[수정 후] 후진 시 미니 후면 카메라 화면 별도 분리 + 기존 화면 유지

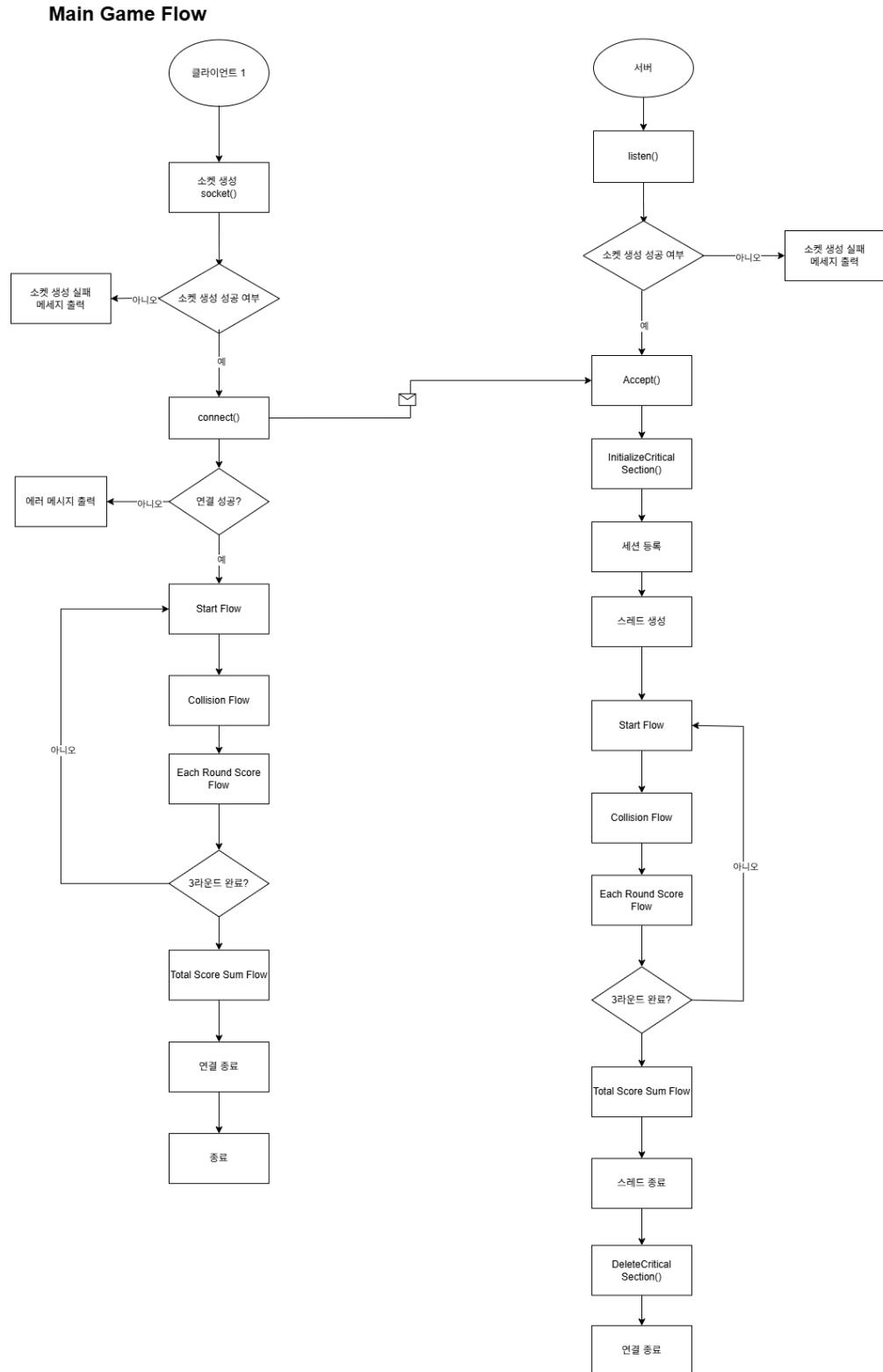
#### 2.1.5 시간 여유 시 추가할 게임 요소

- 1) 바닥 특정 영역 – 느린 속도의 영역(초록색 색칠된 영역)  
빠른 속도의 영역(빨간색 색칠된 영역)
- 2) 경적 울리기 – 상대방 주차 방해 요소  
정해진 시간 초마다 1번씩 사용 가능  
일시적으로 상대방이 몇 초간 이동하지 못하도록 방해하는 스킬
- 3) 상대방 차량 충돌 시 가해자 차량, 피해자 차량 구분하여 가해자 차량에게만 점수 감점하기
- 4) 미니맵 구현

### 3. High-Level Design

#### 3.1 Main Game Flow

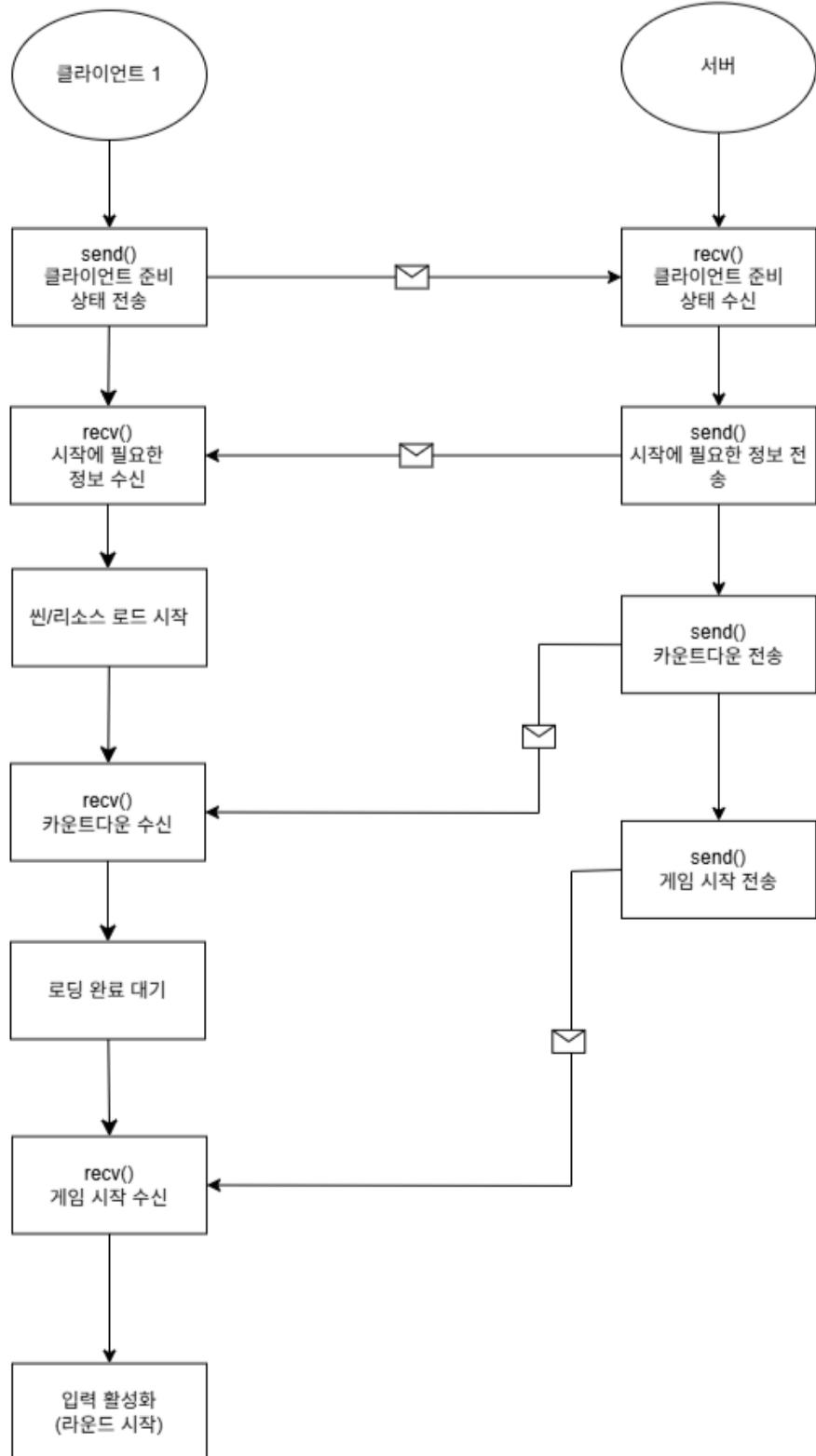
전반적인 게임 기능의 흐름을 담았다.



### 3.2 Start Flow

게임 시작될 때의 기능 구현에 대한 흐름을 담았다. (매칭, 게임 시작 문구)

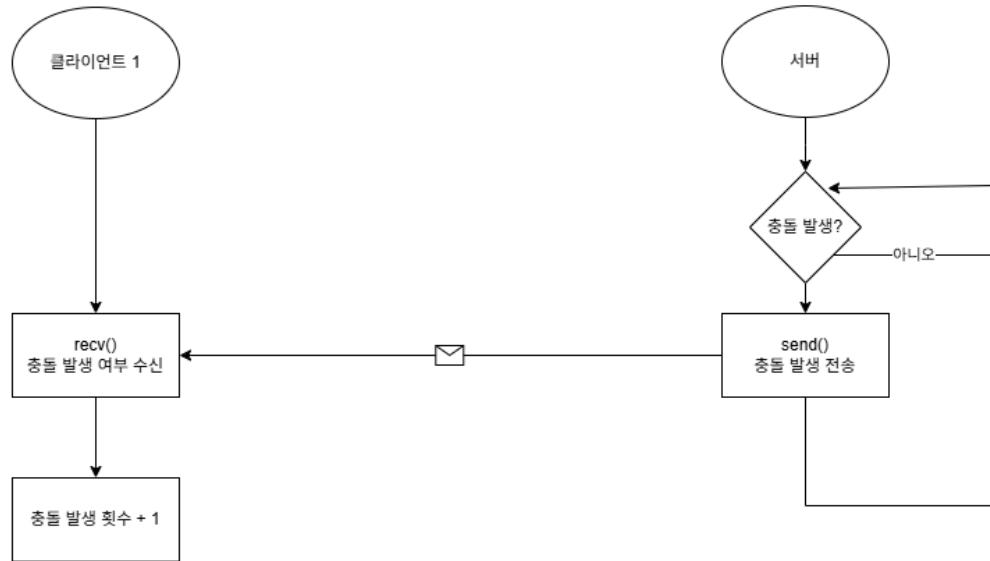
#### Start Flow



### 3.3 Collision Flow

게임 진행 중 충돌 상황을 담았다.

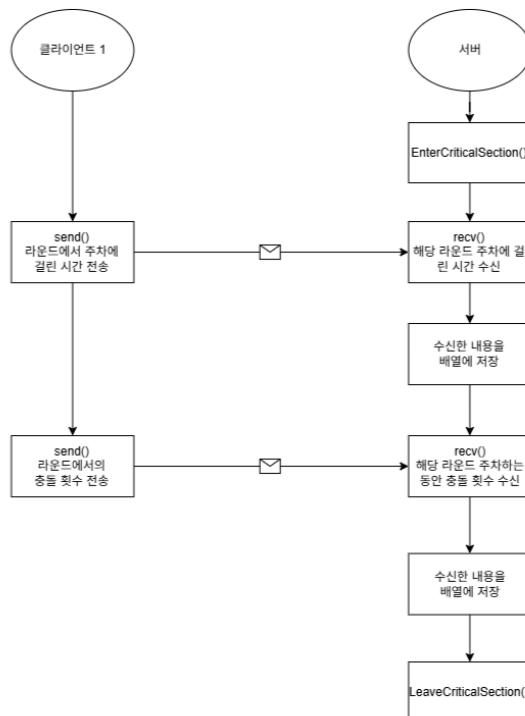
**Collision Flow**



### 3.4 Each Round Score Flow

각 단계마다 점수를 서버에게 보내는 상황을 담았다.

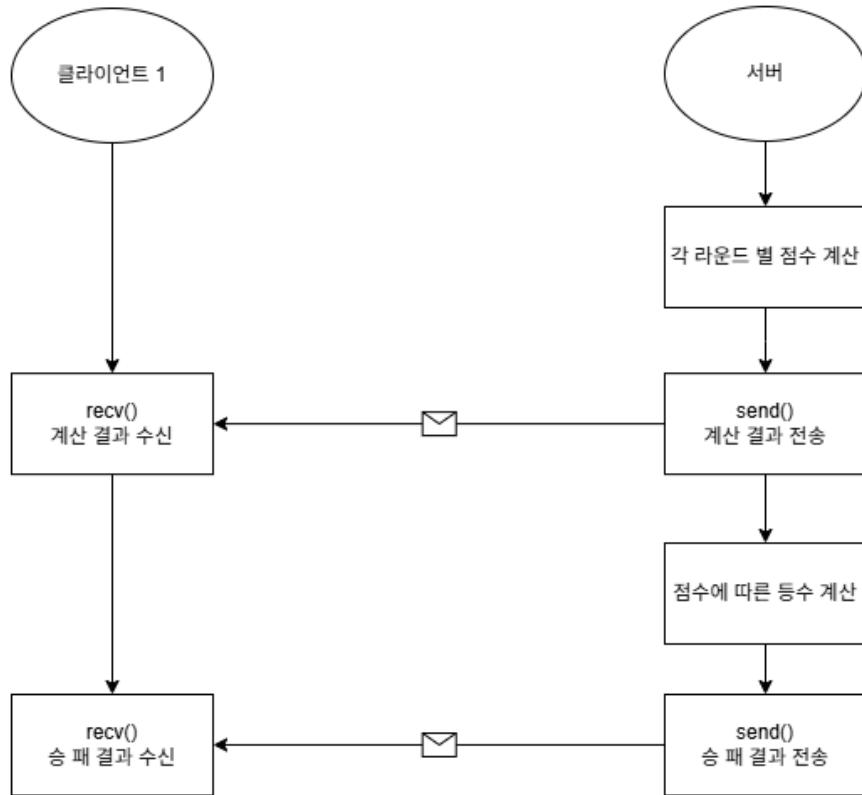
**Each Round Score Flow**



### 3.5 Total Score Sum Flow

3단계의 게임 진행이 모두 끝나고 점수 환산 및 등수에 대해 담았다.

#### Total Score Sum Flow



## 4. Low-Level Design

### 4.1 데이터 흐름 개요

- 1) 클라이언트 → 서버: 입력키만 보냄
- 2) 서버: 그 입력으로 물리/충돌을 시뮬레이션해서 권위 좌표를 결정
- 3) 서버 → 클라이언트: 권위 좌표(및 속도/상태)를 브로드캐스트로 각 클라이언트에게 전달
- 4) 클라이언트: 수신한 권위 좌표로 자기 상태를 동기화(보정)

### 4.2 공유 변수

S2C\_GameStateUpdatePacket의 구성요소를 간단히 서버측의 모든 코드 블록에서 임계영역을 지정하여 하나의 쓰레드만 접근 가능하도록 한다.

### 4.3 구조체 선언

#### [PlayerData]

플레이어의 시각적 상태를 정의

```
struct PlayerData {  
    // --- 식별자 ---  
    int     playerID;  
    // --- 위치 및 회전 (실시간 동기화) ---  
    float   car_dx;  
    float   car_dy;  
    float   car_dz;  
    float   car_rotateY;  
  
    // --- 시각적 상세 상태 (애니메이션용) ---  
    float   front_wheels_rotateY;  
    float   wheel_rect_rotateX;  
    GearState currentGear;  
};
```

## [PlayerGameStats]

서버가 관리하는 플레이어의 경쟁 상태

```
struct PlayerGameStats {  
    int playerID;  
    int collisionCount;      // 충돌 횟수 (서버가 카운트)  
    float parkingSec;        // 주차 완료까지 걸린 시간 (서버가 기록)  
    bool isParked;           // 주차 완료 여부  
};
```

## [PlayerFinalScore]

게임 종료 시 서버가 계산하여 클라이언트에 전송할 최종 점수 내역

```
// 게임 종료 시 점수 내역을 담을 구조체  
struct PlayerFinalScore {  
    int playerID;  
    // 1. 원시 데이터  
    float parkingTimeSeconds; // 총 주차 시간  
    int collisionCount;      // 총 충돌 횟수  
  
    // 2. 변환된 점수  
    int timeScore;            // 주차 시간으로 획득한 점수  
    int collisionPenalty;     // 충돌로 인해 감점된 점수  
    int finalScore;           // 최종 점수 (timeScore - collisionPenalty)  
};
```

## [PacketType]

```
enum PacketType : uint8_t {  
    C2S_PlayerUpdate,      // (클->서) 내 위치/상태 주기적 업데이트  
    S2C_GameStart,         // (서->클) [유니캐스트] 게임 시작 (스테이지 정보 포함)  
    S2C_GameStateUpdate,   // (서->클) [유니캐스트] 모든 플레이어 최신 상태  
    S2C_GameOver          // (서->클) [유니캐스트] 게임 종료 및 결과  
};
```

## [C2S(Client To Server)패킷]

```
// C2S_PlayerUpdate: 내 상태의 주기적 업데이트
struct C2S_PlayerUpdatePacket {
    PacketType type = C2S_PlayerUpdate;
    int     playerID;
    Playerkey myData; // Playerkey는 그래픽스에서 선언된 방향키들의
                      // 입력을 담은 변수들의 구조체를 뜻함
                      // (그래픽스 수정하면서 이름 변경될 수도 있음)
};
```

## [S2C(Server To Client)패킷]

```
// S2C_GameStart: 매칭 완료, 게임 시작
struct S2C_GameStartPacket {
    PacketType type = S2C_GameStart;
    int     stageID;           // 1, 2, 3 (기존 current_stage)
};

// S2C_GameStateUpdate: 주기적인 게임 상황 유니캐스트
struct S2C_GameStateUpdatePacket {
    PacketType type = S2C_GameStateUpdate;
    int     srvElapsedSec;    // 서버 기준 경과 시간
    PlayerData   playerStates[4]; // 플레이어의 실시간 위치 (상대방 렌더링용)
    PlayerGameStats playerStats[4]; // 플레이어의 총돌 횟수, 주차 여부 (UI 표시
                                  // 용), 클라의 차량이 주차 구역에 다 들어갔을 시 주차 선의 색깔 변화(흰색 -> 초록색)
};

// S2C_GameOver: 모든 플레이어가 주차 완료 시 결과 전송
enum GameWinner : uint8_t {
    Player_0,
    Player_1,
    Player_2,
    Player_3,
    Draw
};

struct S2C_GameOverPacket {
    PacketType      type = S2C_GameOver;
    GameWinner      winner[4];        // 점수 높은 순으로 배열된 등수, 매 단계마다
                                      // 점수              에 따른 등수가 갱신되어 플레이어에게 점수와 함
                                      // 께 보여진다. (1~4등)
    PlayerFinalScore finalScores[4][3]; // 각 플레이어(4명)의 라운드별(3라운드)
                                      // 점수 내역
};
```

## 4.3 네트워크 함수 API

### 4.3.1 클라이언트 API

#### DWORD WINAPI Client\_UDP()

- 설명: UDP 전용 네트워크 송수신 담당 스레드 함수, 실시간으로 빨리 처리해야 하는 이동, 충돌과 관련되어 패킷이 송수신된다.
- 동작
  - 송신: 클라이언트에서 입력된 키 값을 서버에게 전송(C2S\_PlayerUpdatePacket 을 전송)
  - 수신:
    1. ProcessData() 호출하여 해당 클라이언트의 데이터만 추출
    2. S2C\_GameStateUpdate 수신 시: OtherPlayerStates()를 호출하여 상대방 차량의 상태를 업데이트, bool isEnterParking = TRUE 인 차량이 있을 시 EnterParkingArea()를 호출
    3. 충돌 발생 시 데이터에 넣어줌

#### DWORD WINAPI Client\_TCP()

- 설명: TCP 전용 네트워크 송수신 담당 스레드 함수, 정확성이 필요한 게임 시작 메시지, 게임 종료 시 점수 및 등수 결과에 대한 처리
- 동작:
  - 송신:
    1. 클라이언트가 주차 구역에 차량을 완전히 넣고 기어를 Parking 으로 변경했을 때 C2S\_ReportParked 를 송신
    2. 클라이언트의 게임 시작 준비 상태를 보고
  - 수신:
    1. ProcessData() 호출하여 해당 클라이언트의 데이터만 추출
    2. S2C\_GameStart 수신 시: S2C\_GameStartPacket 에서 게임 메시지를 전달받습니다.
    3. S2C\_GameStateUpdate 수신 시: displayUI()를 호출하여 게임 진행 상황을 클라이언트 화면에 업데이트합니다.

4. S2C\_GameOver 수신 시: pause\_mode = true 로 설정하고, 서버에서 전달받은 게임 결과를 기반으로 해당 단계에서의 등수 및 점수 결과 창을 띄웁니다.

#### **void OtherPlayerStates()**

- 설명: playerStates[opponentID]를 이용해 상대방 차량을 렌더링합니다.

#### **void displayUI()**

- 설명: 서버가 계산해준 playerStats로 클라이언트의 UI(전체 차들 중 주차 여부, 충돌 횟수, 경과 시간)를 갱신합니다.

#### **Void displayMessage()**

- 설명 : S2C\_GameStart 수신 시 게임 메시지 출력

#### **void Network\_RecvCollisions()**

1. - 설명: 메인 함수 while 문 안에서 호출되는 함수, 서버에서 충돌 발생시 보낸 보정 좌표를 통해 클라이언트에서 충돌 차량들에 대한 렌더링을 합니다.
2. 동작
  - 1) 충돌 확인(패킷 수신 확인)
  - 2) 보정 좌표를 통해 현재 이동거리와 보정좌표의 이동 거리의 10%씩 서서히 이동시키기
  - 3) 렌더링

#### **void ProcessData()**

- 설명: 서버로부터 받은 여러 클라이언트들의 구조체 배열 중 해당 클라이언트의 데이터만 추출하는 함수, 추출한 데이터를 다른 함수로 넘긴다.

#### **void EnterParkingArea()**

- 설명: 서버로부터 주차공간에 클라이언트의 차량이 들어왔음을 전달받고 주차공간의 색깔이 초록색으로 변하도록 한다.

### 4.3.2 서버 API

#### DWORD WINAPI Server\_TCP()

- 설명: TCP 전용 네트워크 송수신 담당 스레드 함수, 클라이언트당 1 개씩 스레드 함수가 생성된다. 게임 시작 메시지, 게임 종료 시 점수 및 등수 결과에 대한 처리를 한다.

- 동작:

수신

1. 플레이어의 주차 상태(C2S\_ReportParked)
2. 클라이언트의 게임 시작 준비 상태

송신

1. 만약 4 명(0, 1, 2, 3)이 모두 접속했다면 Server\_StartMatch() 호출하여 초기화 및 게임 시작 메시지 전송
2. 충돌 발생 시: PlayerGameStats 타입의 충돌 횟수를 1 증가시키고 S2C\_GameStateUpdatePacket 을 해당 클라이언트에게 전송
3. 게임 시간 초과되었을 시 Server\_EndMatch()를 호출한 뒤, 반환받은 패킷을 통해 4 명의 플레이어에게 유니캐스트한다.

#### DWORD WINAPI Server\_UDP()

- 설명: UDP 전용 네트워크 송수신 담당 스레드 함수, 이동, 충돌에 대한 처리를 합니다.

- 동작:

송신:

1. 충돌 발생 시: Server\_CheckAllCollisions() 호출하여 보정 좌표를 계산하고 클라에게 보정좌표를 보내준다.
2. 주기적 상태 업데이트 (유니캐스트):  
GameLoop() 등의 함수를 통해 게임의 경과 시간을 확인하고 주기적으로 다음 로직을 실행한다.  
반복문을 사용하여 현재 접속 중인 \*\*모든 클라이언트(4 명)\*\*를 순회한다.

반복문 내에서 S2C\_GameStateUpdatePacket 을 생성하고, 이를 해당 클라이언트의 UDP 주소로 유니캐스트한다.

수신:

1. C2S\_PlayerUpdate 수신 시: Server\_movement()를 호출하여 해당 클라이언트의 이동 및 상태를 계산합니다.

### **void Server\_movement()**

- 설명: 클라이언트로부터 들어온 방향키를 통해 이동 및 이동과 관련된 주차선 색깔 변경, 주차 완료 이벤트 처리/게임 완료되었는지 확인하는 함수(Server\_CheckGameOver) 호출

- 동작:

1. C2S\_PlayerUpdate 수신 시: 클라이언트로부터 전달받은 Playerkey(플레이어의 키 정보)를 토대로 이동을 계산(방향+속도)하여 PlayerData playerStates 에 갱신  
여기서 클라이언트가 주차구역 내부로 다 들어갔을 시 EnterParkingArea()를 호출하여 주차선이 흰색->초록색으로 변동하도록 한다.
2. bool isEnterParking = TRUE이고 클라이언트로부터 주차를 의미하는 P 키가 입력됐을 시, parkingSec 기록 후 클라이언트의 주차 완료 여부를 나타내는 bool isParked = TRUE;로 바꾸고 currentGear == PARK 로 설정 이후 Server\_CheckGameOver() 호출.

### **void Server\_HandleNC(Address clientAddress)**

- 설명: 새 유저 접속 처리, main 함수 accept 호출 이후에 실행된다.

- 동작:

1. playerID 할당 (0~3 번).

### **void Server\_StartMatch()**

- 설명: 4 명의 플레이어가 모이면 게임 시작 전 플레이어의 상태를 초기화하고 게임 시작 카운트 다운 이후 게임시작 메시지를 전송합니다.

- 동작:

1. 스테이지(stageID) 결정, 매치 시작 시간 기록.
2. 모든 플레이어의 PlayerGameStats 초기화 (충돌 0, 시간 0, isParked=false).
3. S2C\_GameStartPacket 을 4 명의 플레이어에게 유니캐스트.
4. 클라이언트에게 카운트다운 3,2,1 메시지와 게임 시작 메시지 전송(유니캐스트)

#### **void Server\_CheckAllCollisions()**

- 설명: 모든 충돌을 서버가 권위적으로 판정합니다.
- 동작:
  1. 환경 충돌: 각 플레이어의 위치와 서버에 로드된 맵 데이터(벽, 장애물)를 비교하여 충돌했는지 판정합니다.
  2. 플레이어 충돌: 4 명의 각 플레이어는 바운딩 박스(Bounding Box)를 가진다. 서로의 바운딩 박스를 비교하여 두 차량이 겹쳤는지(충돌했는지) 판정합니다.
  3. 패널티 적용: 충돌이 감지되면(환경 또는 플레이어), 해당 플레이어(들)의 playerStats[N].collisionCount++를 실행합니다. (참고: 연속적인 충돌로 점수가 무한정 오르지 않도록, "충돌 상태가 아니었다가 충돌 상태로 진입하는" 순간에만 1 회 카운트하는 로직(cooldown 또는 state-check)이 필요)
  4. 클라이언트들이 서로의 차량이 충돌했을 시 각 차량의 최종 위치, 속도를 계산 후 겹침없이 밀어낸 보정 좌표 값을 계산합니다.

#### **void Server\_CheckGameOver()**

- 설명: 4 명의 플레이어가 모두 주차했는지 확인하는 함수
- 동작: 만약 4 명이 다 주차 완료했을 시 Server\_EndMatch() 호출.

#### **S2C\_GameOverPacket Server\_EndMatch()**

- 설명: 각 단계마다 4 명의 플레이어의 플레이가 종료되었을 때 호출됩니다. 각 플레이어의 해당 단계에서의 최종 점수를 계산하고, 등수를 판별하여 S2C\_GameOverPacket 를 반환합니다.

- 동작:

1. 4 명의 플레이어의 PlayerGameStats 를 가져옵니다.
2. PlayerFinalScore scoreN = SrvCalcScore(N, playerStats[N]);를 호출합니다.
3. finalScores 배열 (score0, score1, score2)을 채웁니다.
4. S2C\_GameOverPacket 을 생성하고, Rank()호출하여 winner 전역 배열 변수에 누적된 단계에서의 점수를 바탕으로 가장 높은 순서의 등수별로 플레이어를 정렬하여 넣어줍니다.
5. 이 패킷을 반환합니다.

### **PlayerFinalScore SrvCalcScore(int playerId, const PlayerGameStats& stats)**

- 설명: 한 플레이어의 PlayerGameStats 를 기반으로 점수 공식에 따라 PlayerFinalScore 구조체를 계산하여 반환합니다.
- 매개변수: playerId: 점수를 계산할 플레이어 ID. stats: 해당 플레이어의 (서버가 판정한)collisionCount 와 parkingSec 가 포함된 통계.
- 반환값: PlayerFinalScore (점수 내역이 모두 채워진 구조체).

- 동작 (예시 공식):

```
PlayerFinalScore result;
result.playerID = playerId;
result.parkingTimeSeconds = stats.parkingTimeSeconds;
result.collisionCount = stats.collisionCount;

// --- (1) 주차 시간 점수 계산 ---
// 예: 기본 10,000점, 1초당 100점 감점. 최소 0점.
int timeScore = 10000 - (int)(stats.parkingTimeSeconds * 100);
result.timeScore = (timeScore > 0) ? timeScore : 0; // 최소 0점 보장

// --- (2) 충돌 감점 계산 ---
// 예: 충돌 1회당 500점 감점 (PvE, PvP 모두 동일하게 적용)
result.collisionPenalty = stats.collisionCount * 500;

// --- (3) 최종 점수 계산 ---
result.finalScore = result.timeScore - result.collisionPenalty;

return result
```

### **void Server\_GameStateSend()**

- 설명: 서버에서의 게임의 경과 시간을 업데이트

### **GameWinner Rank()**

- 설명: 최종 점수를 바탕으로 winner 전역 배열 변수에 등수에 맞춰서 제일 점수가 높은 순서로 넣어준다.

### **void ServerParkingArea()**

- 설명: 클라이언트의 차량이 주차구역에 들어간다면 각 클라 화면에서 주차선 색깔의 변화를 주도록 주차 구역을 다 들어갔다는 정보를 준다.  
S2C\_GameStateUpdatePacket에서 PlayerGameStats 타입의 playerStats 변수에서 bool isEnterParking = TRUE로 바꾼다.

## **5. 팀원별 역할 분담**

**이태현** – Server\_HandleNC, Server\_CheckAllCollisions(), Server\_movement(), Server\_CheckGameOver(), Server\_UDP() 구현

**윤혜린** – displayUI(), SrvCalcScore, Server\_EndMatch(), Rank(), Server\_StartMatch(), Network\_RecvCollisions(), ProcessData(), Client\_TCP() 구현

**지민우** – Client\_UDP(), OtherPlayerStates(), EnterParkingArea(), ServerParkingArea() 구현, 그래픽스 수정



## 6. 일정표

### <이태현>

일	월	화	수	목	금	토
						11/1 서버 기초 골자 구현 - main()
11/2	11/3 main() - TCP 접속	11/4 main() - UDP 접속	11/5	11/6	11/7 Server_Handl eNC playerID 할 당	11/8
11/9 Server_movement PlayerUpdate 수신 시의 동작	11/10	11/11	11/12 Server_move ment 주차 완료 판정	11/13	11/14	11/15 Server_Check GameOver 4명 주차 완 료 여부 체 크 후 EndMatch()
11/16	11/17 Server_Check Collision() 장애물과의 충돌 판정 체크	11/18	11/19	11/20 Server_Check Collision() 장애물과 충 돌 시 보정 움직임	11/21	11/22

11/23	11/24 Server_Check Collision() 상대방과의 충돌 판정 체크	11/25	11/26	11/27	11/28 Server_Check Collision() 상대방과 충 돌 시 보정 위치 움직임	11/29
11/30	12/1	12/2 Server_Check Collision 점검 및 충 돌 시 가해 자 피해자 구분 구현 시도	12/3	12/4	12/5	12/6
12/7	12/8	12/9	12/10	12/11		

## <윤혜린>

일	월	화	수	목	금	토
						11/1
11/2	11/3	11/4	11/5	11/6	11/7 클라이언트 네트워크 main함수에 네트워크 함 수 구현(충돌 제외)	11/8 ProcessData() )
11/9 Server_Start Match()	11/10	11/11	11/12 Server_EndM atch()	11/13	11/14	11/15 Network_Re cvCollisions() 충돌 확인 및 좌표 이 동
11/16 Network_Re cvCollisions() 차량 렌더링	11/17	11/18	11/19 Rank()	11/20	11/21	11/22 displayUI()
11/23 Client_TCP() (S2C_GameO ver 수신 시 제외)	11/24	11/25	11/26 Client_TCP() (S2C_GameO ver 수신 시)	11/27	11/28	11/29 SrvCalcScore (
11/30	12/1 최종 점검	12/2	12/3	12/4	12/5	12/6
12/7	12/8	12/9	12/10	12/11		

## <지민우>

일	월	화	수	목	금	토
						11/1
11/2	11/3 (그래픽스 수정) game_state.c / game_state.h	11/4	11/5 (그래픽스 수정) mesh.c / mesh.h	11/6	11/7 (그래픽스 수정) shader.c / shader.h	11/8 (그래픽스 수정) car.c / car.h
11/9	11/10 (그래픽스 수정) input_handle r.c / input_handle r.h	11/11	11/12 (그래픽스 수정) collision.c / collision.h / environment.c / environment.h	11/13	11/14 (그래픽스 수정) renderer.c / renderer.h	11/15 (그래픽스 수정) main.c / main.h
11/16	11/17 (그래픽스 수정) 모든 게임 기능이 분리 후에도 정상 동작하는지 확인 후 디버깅	11/18	11/19 (그래픽스 수정) 최종 정리 / 주석 작성	11/20	11/21 Client_UDP() C2S_PlayerUpdatePacket 송신 로직	11/22 OtherPlayerStates() S2C_GameStateUpdate 패킷 수신 시, 상대방 차량 렌더링
11/23	11/24 OtherPlayerStates() 구현 완료 및 디버깅	11/25	11/26 ServerParkingArea() 서버에서 주차 구역 진입 판단 및 isEnterParking 상태 갱신 로직	11/27	11/28 EnterParkingArea() isEnterParking 기반으로 주차 공간 색 변경 로직	11/29 Client_UDP() Client_UDP() 내 모든 수신 로직 통합 및 테스트

11/30	12/1 최종 점검 및 통합 테스트	12/2	12/3	12/4	12/5	12/6
12/7	12/8	12/9	12/10	12/11		