네트워크 게임 프로그래밍(01)

Team 프로젝트

추진 계획서

팀원
이태현
윤혜린
지민우

목차

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, 스테이지 마다 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

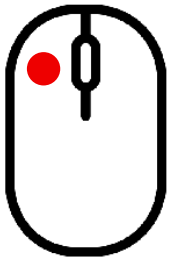
2. 로비, 매치, 결과 저장은 TCP 사용

   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

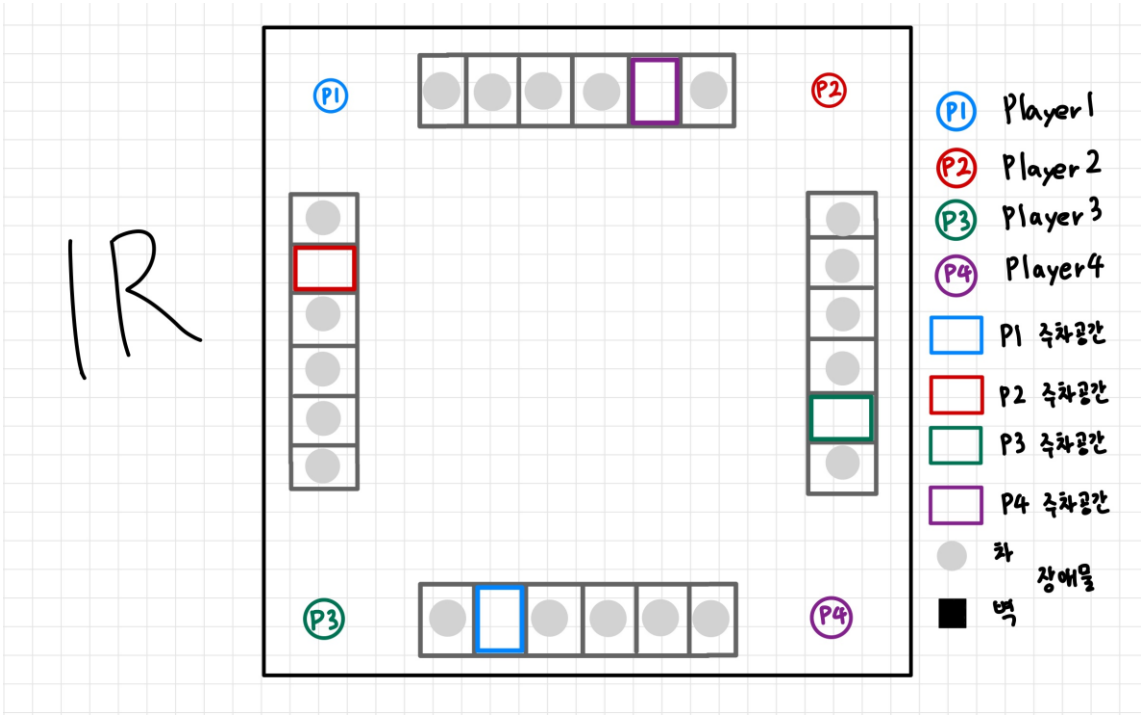
충돌 여부 판단: 충돌을 유발한 가해자 플레이어를 감점

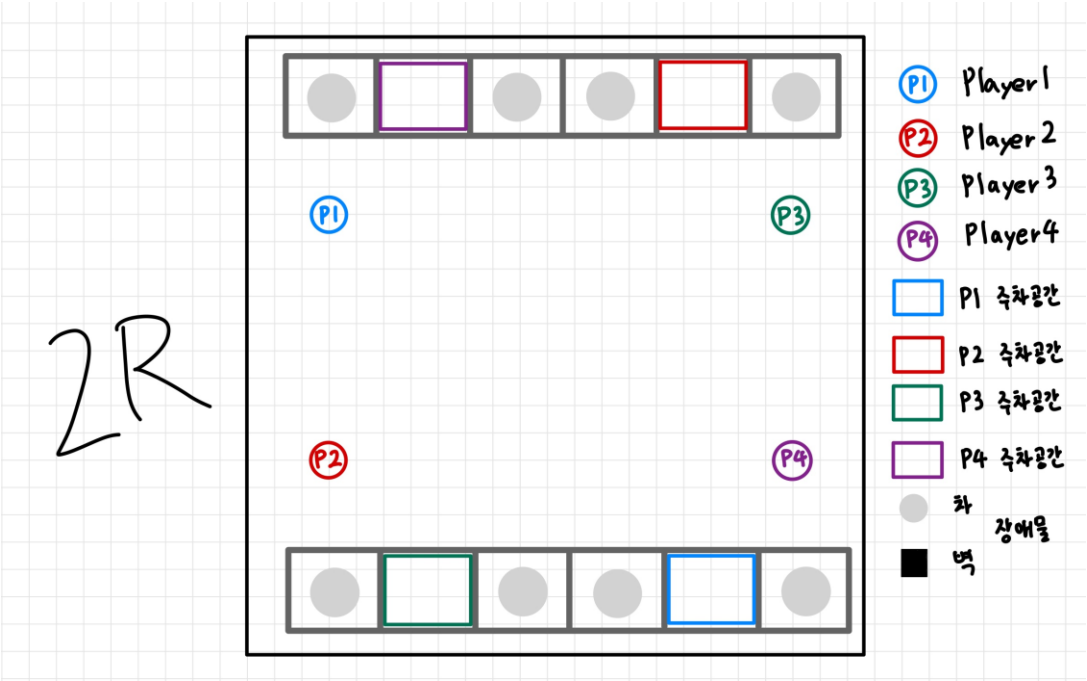
라운드 클리어: 정해진 주차 구역으로 차를 이동하여 기어를 Parking으로 변경 시 주차 성공, 모든 플레이어의 주차가 끝난 후 점수를 계산하여 더 높은 점수를 가진 플레이어가 승리

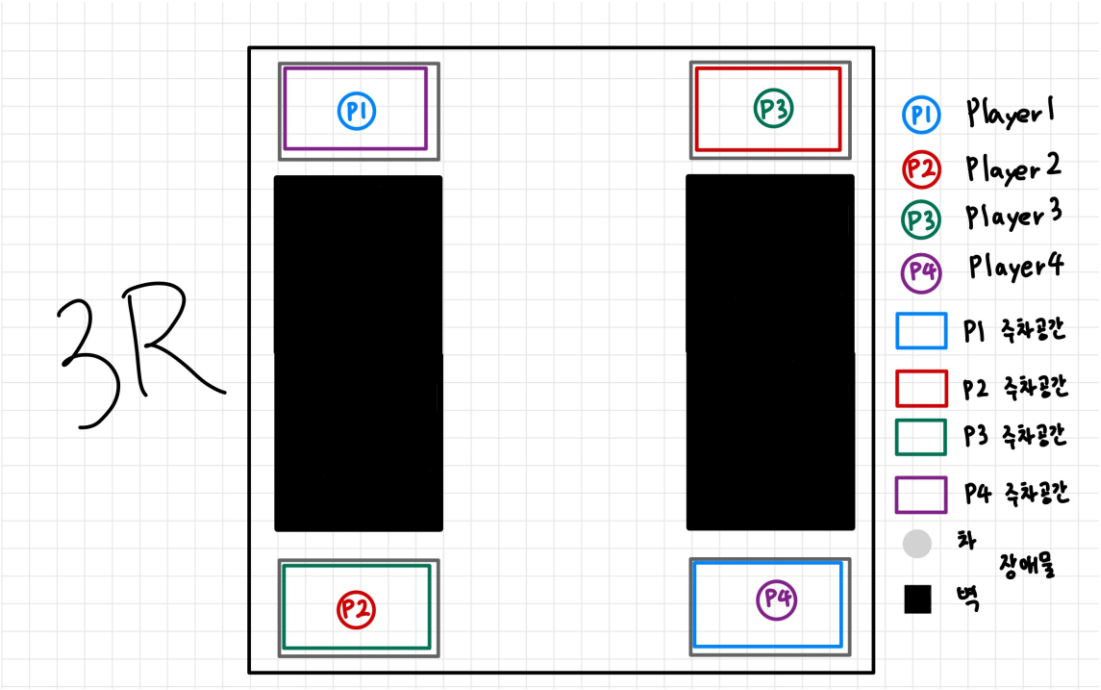
라운드 별로 정해진 시간 내에 주차 완료해야 한다.

최종 승리 판정: 라운드 난이도 별로 점수 가중치를 다르게 주어 점수 계산 후 3라운드 합산하여 최종 점수가 가장 높은 사람이 최종 승리자

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

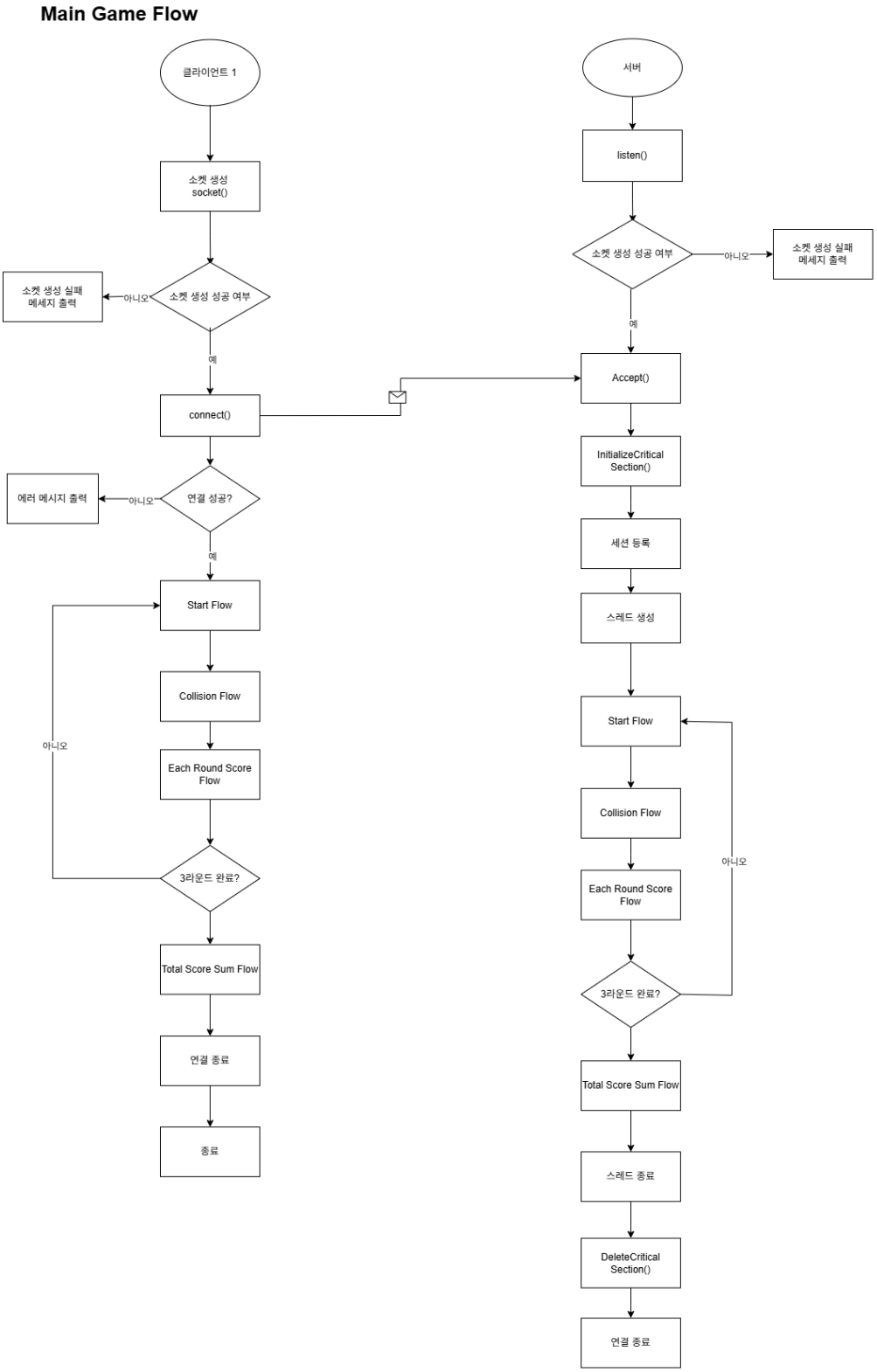
2.1.5 시간 여유 시 추가할 게임 요소

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

3. High-Level Design

3.1 Main Game Flow

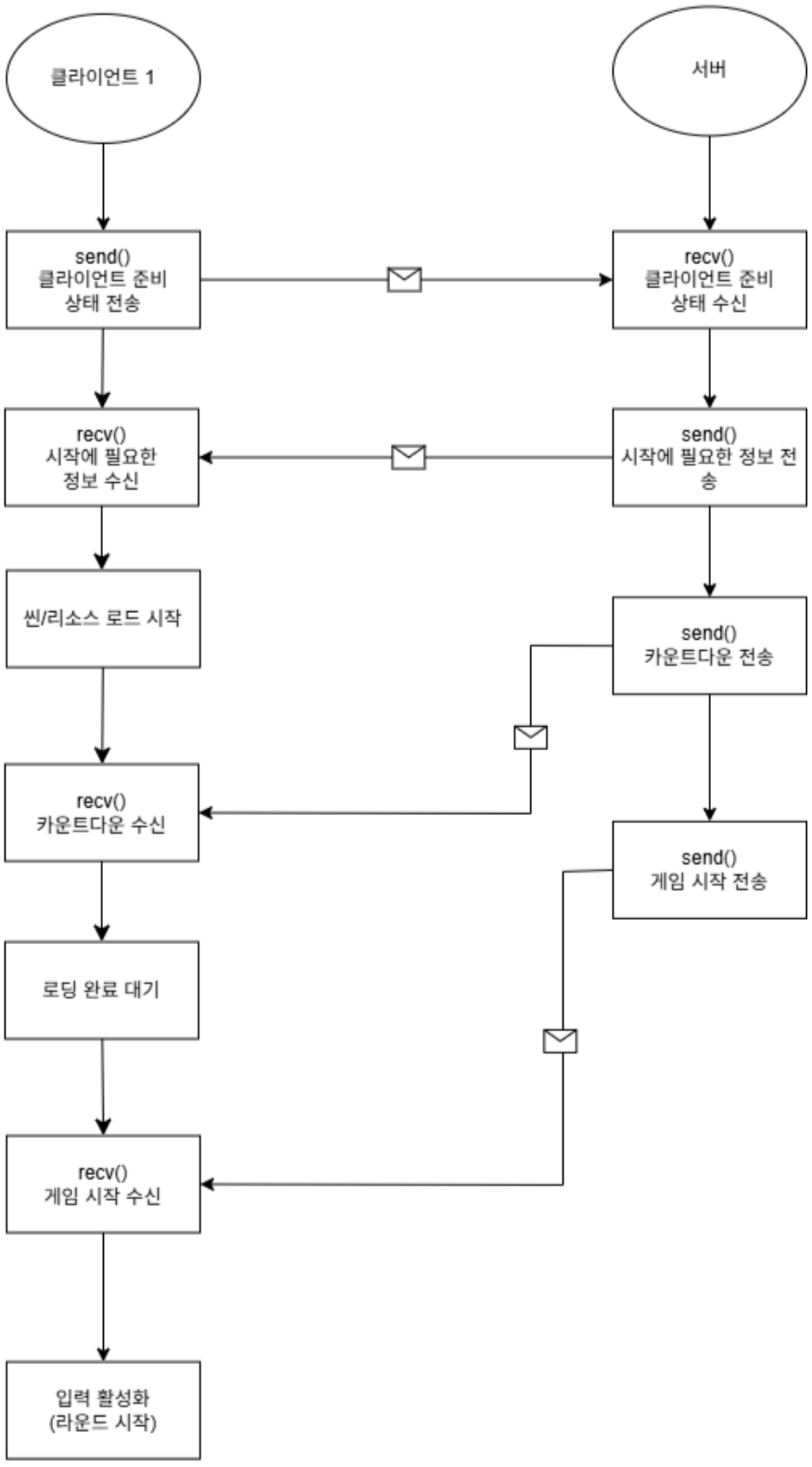
전반적인 게임 기능의 흐름을 담았다.



3.2 Start Flow

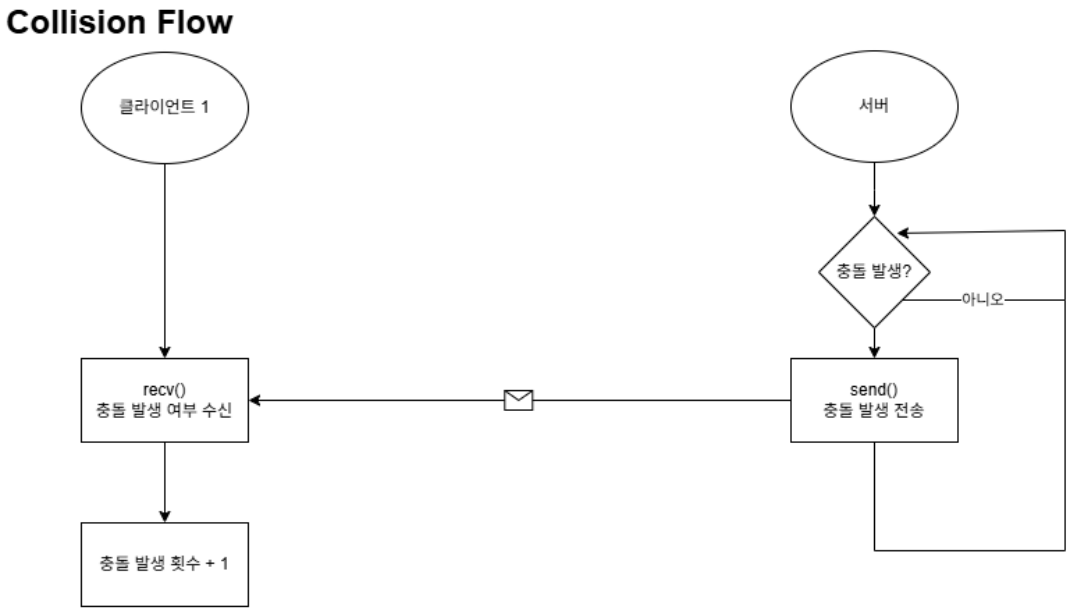
게임 시작될 때의 기능 구현에 대한 흐름을 담았다. (매칭, 게임 시작 문구)

Start Flow



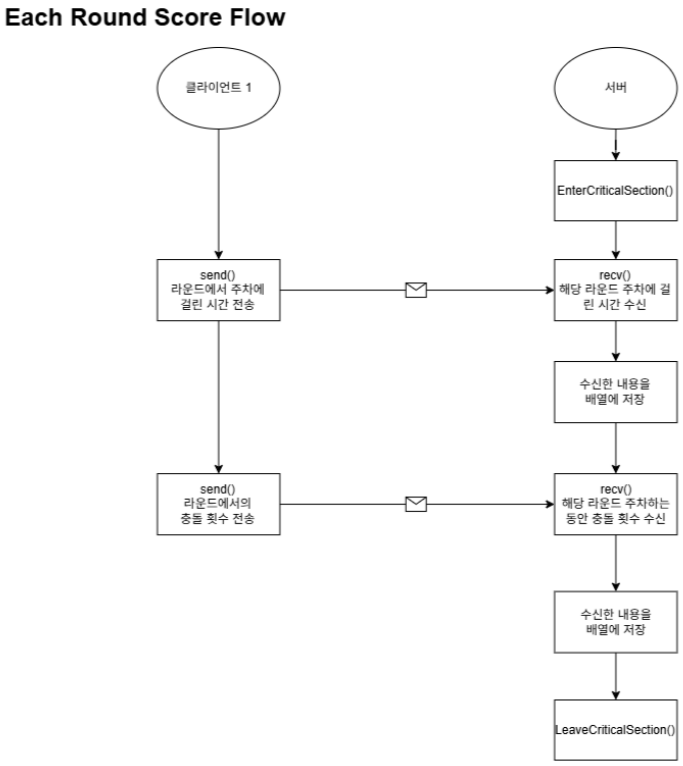
3.3 Collision Flow

게임 진행 중 충돌 상황을 담았다.



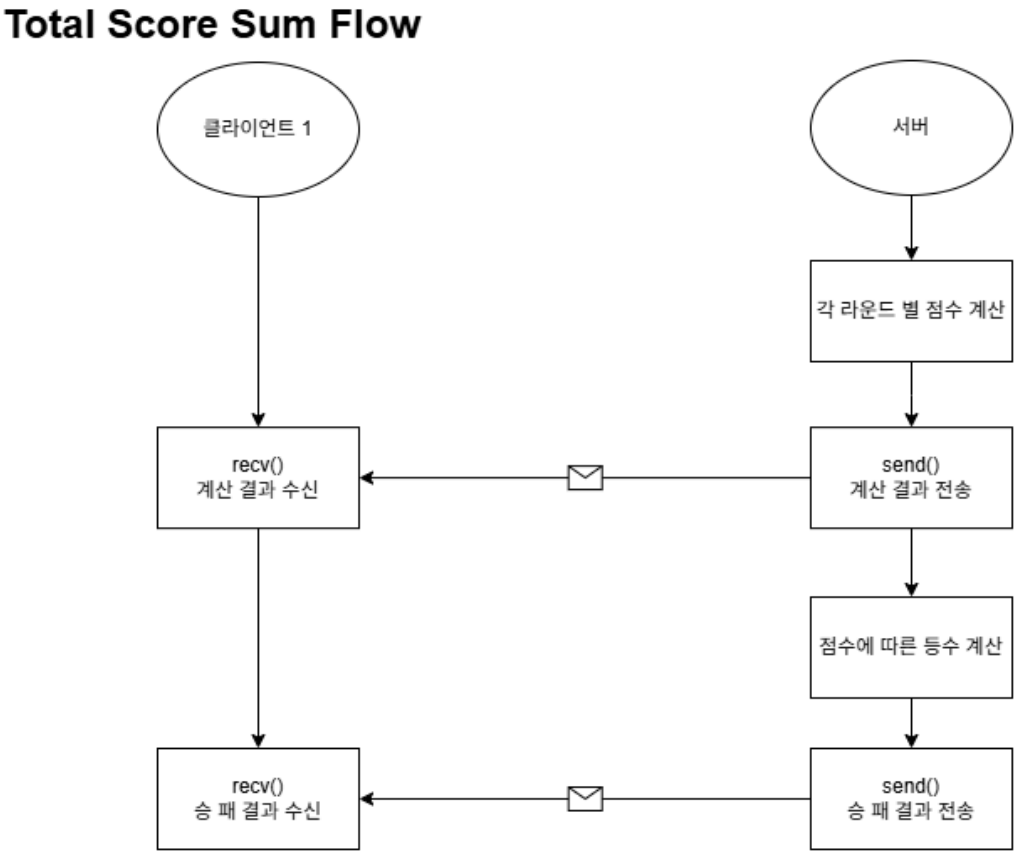
3.4 Each Round Score Flow

각 단계마다 점수를 서버에게 보내는 상황을 담았다.



3.5 Total Score Sum Flow

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



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_GameStateUpdatePacket형 변수 -> 임계영역으로 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
C2S_ReportParked,      // (클->서) "방금 주차 완료" 이벤트 전송
S2C_GameStart,         // (서->클) [유니캐스트] 게임 시작 (스테이지 정보 포함)
S2C_GameStateUpdate,   // (서->클) [브로드캐스트] 모든 플레이어 최신 상태
S2C_GameOver           // (서->클) [브로드캐스트] 게임 종료 및 결과
};
```

[C2S(Clinet 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)
};

// C2S_ReportParked: 주차 완료 시 1회 전송
struct C2S_ReportParkedPacket {
PacketType  type = C2S_ReportParked;
int        playerID;
// (클라이언트는 isParked == true && currentGear == PARK 일 때 전송)
};
```

[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 브로드캐스트
struct S2C_GameStateUpdatePacket {
PacketType  type = S2C_GameStateUpdate;
int           srvElapsedSec;     // 서버 기준 경과 시간
PlayerData     playerStates[2];   // 플레이어의 실시간 위치 (상대방 렌더링용)
PlayerGameStats playerStats[2];    // 플레이어의 충돌 횟수, 주차 여부 (UI 표시
용)
};

// 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 클라이언트API

void Network_SendUpdate()

- 설명: 클라이언트에서 입력된 키 값을 서버에게 전송합니다. C2S_PlayerUpdatePacket 을 전송합니다.

void Network_ReportParked()

- 설명: 클라이언트는 isParked 가 true 가 되고 currentGear == PARK 가 되는 순간 1 회만 호출됩니다. C2S_ReportParkedPacket 을 전송합니다.

DWORD WINAPI Network_ReceivePackets()

- 설명: 별도 스레드에서 계속 호출되어 서버 패킷을 수신하고 그에 따라 로컬 게임 상태를 변경합니다.

- 주요 동작:

1. S2C_GameStart 수신 시: 초기화된 S2C_GameStartPacket 에서 ProcessBroadcast()를 통해 해당되는 클라의 정보만 추출하고 게임 메시지를 전달받아 클라이언트 화면 출력
2. S2C_GameStateUpdate 수신 시: ProcessBroadcast()를 통해 해당되는 클라의 정보만 추출하고 OtherPlayerStates()를 호출하여 상대방 차량의 상태를 업데이트 하고 displayUI()를 호출하여 게임 진행 상황을 업데이트합니다.

3. S2C_GameOver 수신 시: ProcessBroadcast()를 통해 해당되는 클라의 정보만 추출하고 pause_mode = true 로 설정하고, 서버에서 전달 받은 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 playerStats 로 클라이언트의 UI(전체 차들 중 주차 여부, 충돌 횟수, 경과 시간)를 갱신합니다.

void Network_RecvCollisions()

- 설명: 서버에서 충돌 발생시 보낸 보정 좌표를 통해 클라이언트에서 충돌 차량들에 대한 렌더링을 합니다.

void ProcessBroadcast()

- 설명: 서버로부터 받은 브로드 캐스트를 해당 클라이언트의 데이터만 추출하는 함수, 추출한 데이터를 다른 함수로 넘긴다.

4.3.2 서버 API

DWORD WINAPI Server_GameLoop()

- 설명: 서버의 메인 루프. 패킷 수신, 충돌 판정, 게임 상태 브로드캐스트를 반복합니다. 접속하는 클라이언트들에게 멀티 쓰레드를 통해 처음 시작되는 함수이다.

- 동작:

1. Server_ReceivePackets() 호출: 클라이언트로부터 패킷을 수신하여 playerStates 갱신 및 C2S_ReportParked 처리.
2. Server_CheckAllCollisions() 호출: [신규 함수 호출] 갱신된 playerStates 를 기반으로 모든 충돌(환경, 플레이어)을 검사하고 playerStats[N].collisionCount 를 갱신.

3. Server_BroadcastGameState() 호출: 갱신된 S2C_GameStateUpdatePacket 을 주기적으로 브로드캐스트(경과 시간, 실시간 위치만 주는 함수)

void Server_ReceivePackets()

- 설명: 클라이언트로부터 모든 패킷을 수신하고 처리합니다.

- 동작:

1. C2S_PlayerUpdate 수신 시: 해당 플레이어의 PlayerData 를 서버 마스터 상태(playerStates)에 갱신
2. C2S_ReportParked 수신 시: 해당 플레이어의 isParked = true 로 설정, parkingSec 기록. 이후 Server_CheckGameOver() 호출.

void Server_HandleNC(Address clientAddress)

- 설명: 새 유저 접속 처리.

- 동작:

1. playerID 할당 (0~3 번).
2. 만약 4 명(0, 1, 2, 3)이 모두 접속했다면 Server_StartMatch() 호출.

void Server_StartMatch()

- 설명: 4 명의 플레이어가 준비되면 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 브로드캐스트.
4. 클라이언트에게 카운트다운 3,2,1 메시지와 게임 시작 메시지 전송(유니캐스트)

void Server_CheckAllCollisions()

- 설명: Server_GameLoop 에서 주기적으로 호출되어 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 해당 클라이언트들에게 한번에 브로드캐스트로 보내줍니다.

void Server_CheckGameOver()

- 설명: C2S_ReportParked 수신 시 호출되어, 4 명의 플레이어가 모두 주차했는지 확인하고 각 단계별 주차 시간 초과 시 호출됩니다.
- 동작: 만약 4 명이 다 주차 완료 혹은 주어진 주차 시간 초과 시 Server_EndMatch() 호출.

void Server_EndMatch()

- 설명: 각 단계마다 4 명의 플레이어의 플레이가 종료되었을 때 호출됩니다. 각 플레이어의 해당 단계에서의 최종 점수를 계산하고, 등수를 판별하여 S2C_GameOverPacket 을 브로드캐스트합니다.
- 동작:

1. 4 명의 플레이어의 PlayerGameStats 를 가져옵니다.
2. PlayerFinalScore scoreN = SrvCalcScore(N, playerStats[N]);를 호출합니다. finalScores 배열 (score0, score1, score2)을 채웁니다.

3. S2C_GameOverPacket 을 생성하고, Rank()호출하여 winner 전역 배열 변수에 누적된 단계에서의 점수를 바탕으로 가장 높은 순서의 등수별로 플레이어를 정렬하여 넣어줍니다.
4. 이 패킷을 4 명의 플레이어에게 브로드캐스트합니다.

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_BroadcastGameState()

- 설명: 클라이언트에게 실시간 위치, 서버에서의 게임의 경과 시간을 브로드캐스트를 통해 전달

GameWinner Rank()

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

5. 팀원별 역할 분담

이태현 – 서버 충돌 체크, 전반적인 서버의 구조

윤혜린 – 클라이언트 충돌 처리, 기타 서버, 클라 네트워크 기능

지민우 – 게임 그래픽스 수정, 그래픽스와 관련된 네트워크 기능

6. 일정표

<이태현>

일	월	화	수	목	금	토
						11/1 서버 Main함수 기초 골자 구현1
11/2 서버 Main함수 기초 골자 구현2	11/3 Server_GameLoop()1 대략적인 틀만 잡기	11/4	11/5 Server_ReceivePackets()1	11/6	11/7	11/8 Server_HandleNC
11/9 Server_ReceivePackets()2	11/10	11/11 Server_CheckGameOver()	11/12	11/13	11/14 Server_BroadcastGameState()1	11/15 오류점검
11/16 Server_CheckGameOver() – 장애물 충돌1	11/17	11/18 Server_CheckGameOver() – 장애물 충돌2	11/19	11/20	11/21 Server_CheckGameOver() – 상대방 충돌1	11/22 Server_CheckGameOver() – 상대방 충돌2
11/23 Server_CheckGameOver() – 상대방 충돌3	11/24	11/25 Server_CheckGameOver() – 상대방 충돌4	11/26	11/27	11/28	11/29
11/30 Server_CheckGameOver() – 상대방 충돌5	12/1	12/2	12/3 Server_GameLoop()2 전체적으로 종합해서 추가 수정	12/4	12/5	12/6
12/7	12/8 최종 점검	12/9	12/10	12/11		

<윤혜린>

일	월	화	수	목	금	토
						11/1
11/2	11/3	11/4 클라이언트 네트워크 기초 골자 구현1	11/5 클라이언트 네트워크 기초 골자 구현2	11/6 displayUI()	11/7	11/8 SrvCalcScore
11/9 Network_ ReceivePackets()1 대략적인 틀만 잡기	11/10	11/11	11/12 Server_EndMatch()	11/13	11/14	11/15 오류점검
11/16 GameWinnerRank()	11/17	11/18	11/19 Server_StartMatch()	11/20	11/21	11/22 Network_ RecvCollisions()1-장애물 충돌
11/23 Network_ RecvCollisions()2-장애물 충돌	11/24	11/25 Network_ RecvCollisions()3-상대방 충돌	11/26	11/27	11/28	11/29 Network_ RecvCollisions()3-상대방 충돌
11/30 ProcessBroadcast()	12/1	12/2	12/3 Network_ ReceivePackets()2 전체적으로 종합해서 추가 수정	12/4	12/5	12/6
12/7	12/8 최종점검	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 (그래픽스 수정) mesh.c / mesh.h	11/5 (그래픽스 수정) shader.c / shader.h	11/6	11/7 (그래픽스 수정) car.c / car.h	11/8 (그래픽스 수정) input_handler.c / input_handler.h
11/9	11/10 (그래픽스 수정) collision.c / collision.h / environment.c / environment.h	11/11 (그래픽스 수정) renderer.c / renderer.h	11/12 (그래픽스 수정) main.c / main.h	11/13	11/14 (그래픽스 수정) 모든 게임 기능이 분리 후에도 정상 동작하는지 확인 후 디버깅	11/15 (그래픽스 수정) 최종 정리 / 주석 작성
11/16	11/17 Network_SendUpdate()1	11/18	11/19 Network_SendUpdate()2	11/20	11/21	11/22
11/23	11/24 OtherPlayerStates()1	11/25	11/26 OtherPlayerStates()2	11/27	11/28	11/29
11/30	12/1 Network_ReportParked()	12/2	12/3	12/4	12/5	12/6
12/7	12/8 최종 점검	12/9	12/10	12/11		