네트워크 게임 프로그래밍(01)

Team 프로젝트

추진 계획서

팀원
이태현
윤혜린
지민우

<개요>

[게임 제목]

주차의 달인

[게임 설명]

1대1로 3라운드에 걸쳐서 누가 먼저 빠르게 주차를 하는지 경쟁을 하는 2인 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 내려다 보는 시점을 제공(차량의 위치, 주차 구역을 명확하게 볼 수 있다.)

3인칭 뷰에서 마우스 좌클릭 드래그를 통해 시점을 돌릴 수 있다.

[개발한 사람]

2024-2 컴퓨터 그래픽스 최종 프로젝트 개발 (**지민우**, 조성욱)

[버전 관리 프로그램]

GitHub

[작업 IDE]

Visual Studio

[사용할 프로토콜]

1. 차 위치/이동 데이터/충돌 판정/게임 진행 명령 UDP사용

이유 : 플레이어가 플레이할 때 차량 반응이 즉각적이어야함, 빠른 응답이 중요

2. 로비, 매치, 결과 저장은 TCP 사용

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

<애플리케이션 기획>

[게임의 요소]

1. 조작키

W: 가속

SPACE: 브레이크

Q/E: 기어 조작 (위/아래)

ESC: 일시정지

(핸들 내부)MLB 드래그: 핸들 조작 (회전)

(핸들 외부)MLB 드래그: 카메라 시점 회전 (3인칭 뷰에서)

2. 게임 플레이 방식

총 3라운드로 이루어진 갈수록 주차 난이도가 상승하는 구조

시간: 주차하는데 걸린 시간에 따라 점수 부여

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

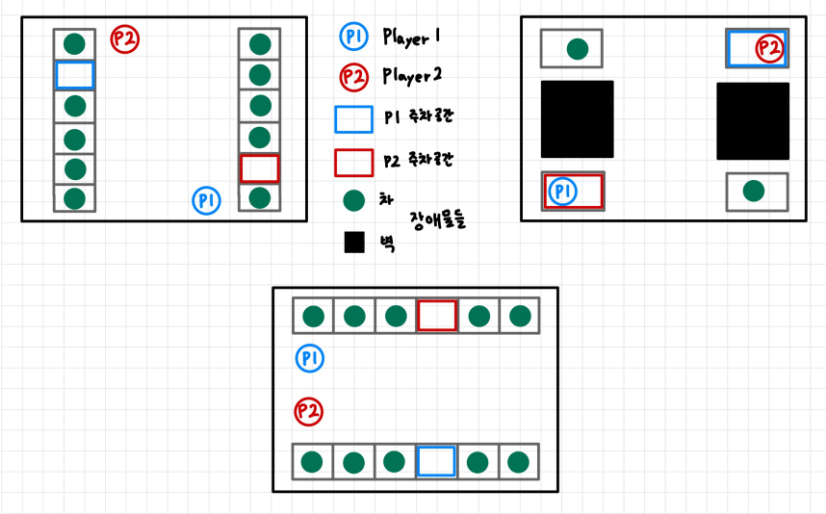
충돌 여부 판단: 충돌을 유발한 가해자 플레이어를 감점

단계 별 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

리타이어 시스템: 먼저 주차에 성공한 플레이어 이후 일정 시간 안에 상대 플레이어가 주차에 성공하지 못하면 주차 실패로 간주(주차 점수 0점)

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

3. 라운드 별 맵 사진 첨부



4. 기존 게임에서의 수정 사항

[수정 전] 후진 시 후면 카메라 전체 화면

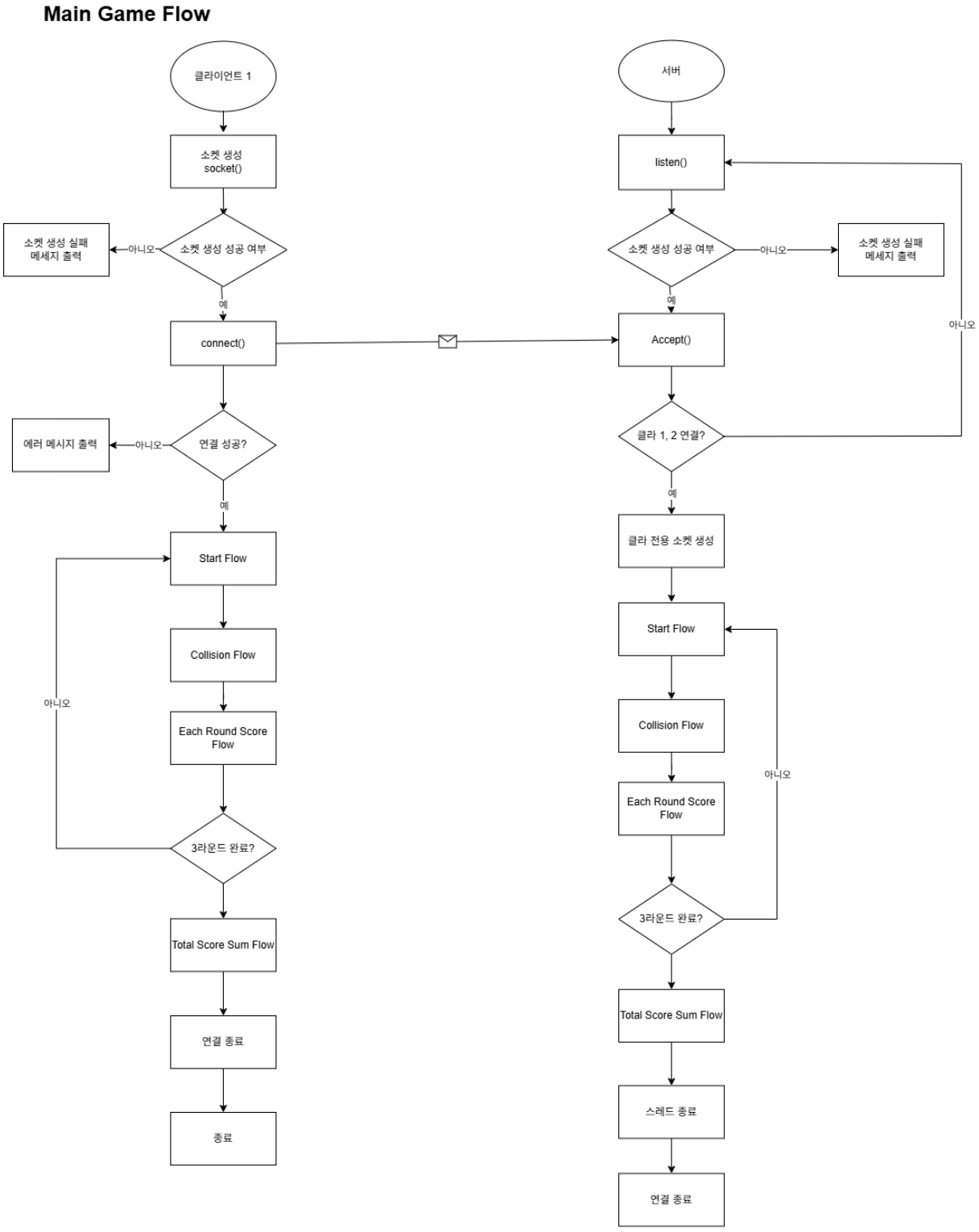
[수정 후] 후진 시 미니 후면 카메라 화면 별도 분리 + 기존 화면 유지

5. 시간 남을 시 추가 게임 요소 넣기

1) 바닥 특정 영역 – 느린 속도의 영역(초록색 색칠된 영역)
빠른 속도의 영역(빨간색 색칠된 영역)

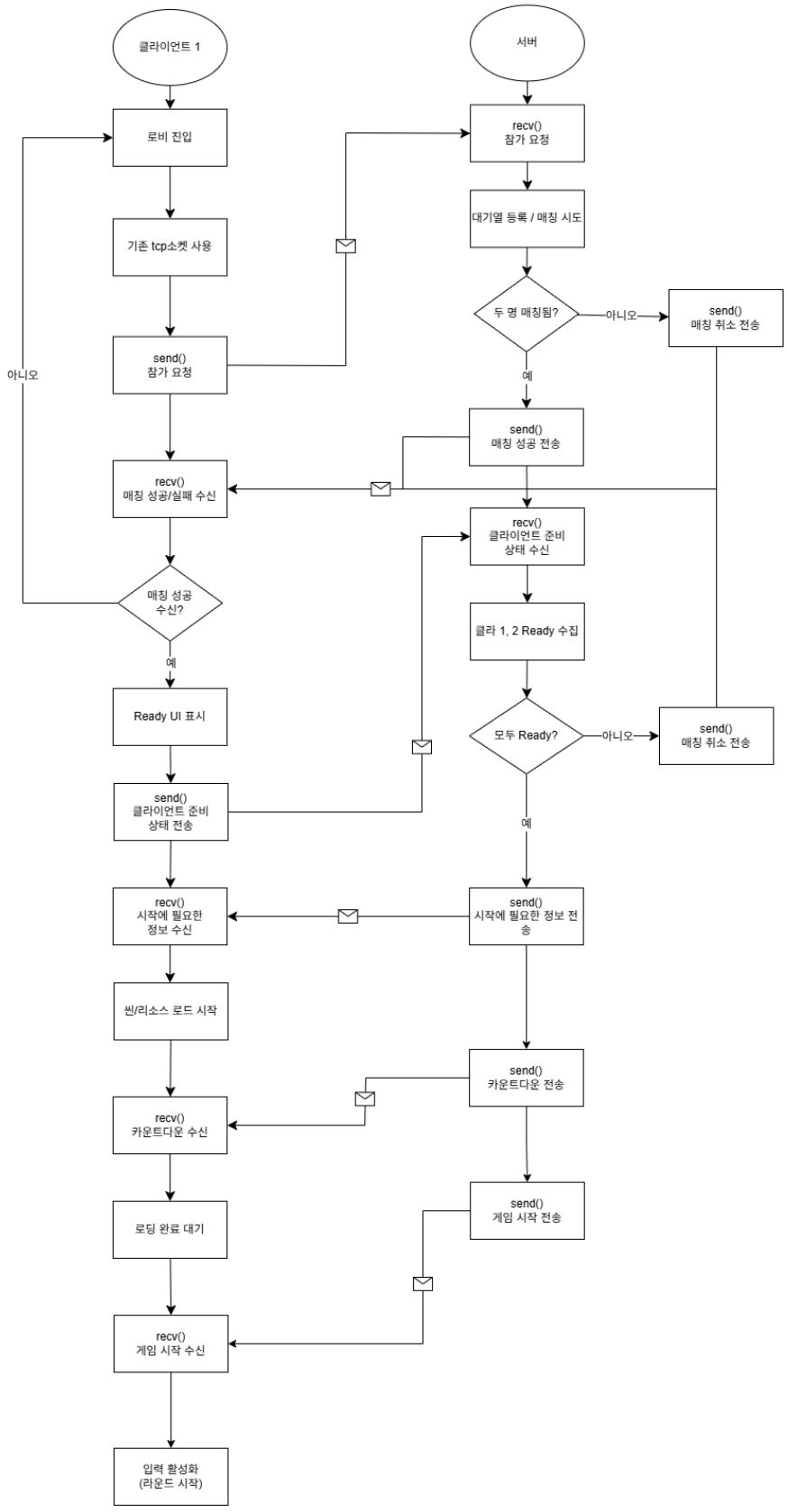
2) 경적 울리기 – 상대방 주차 방해 요소
정해진 시간 초마다 1번씩 사용 가능
일시적으로 상대방이 몇초간 이동하지 못하도록 방해하는 스킬

<High-Level Design>



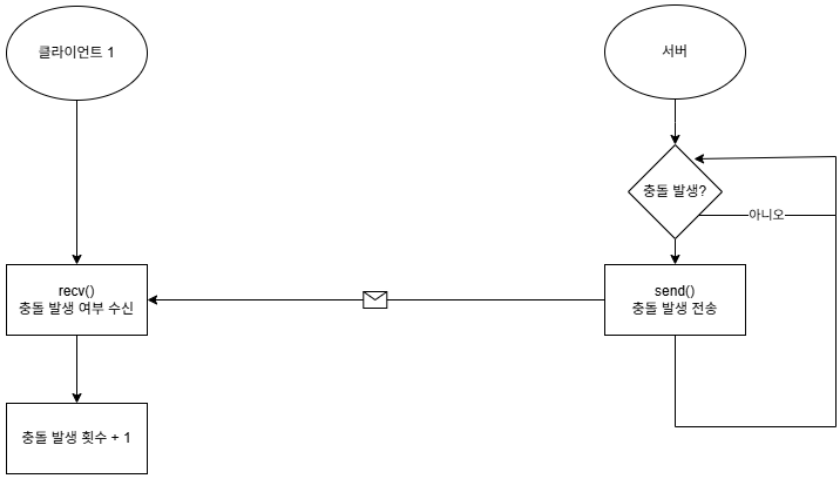
Main Game Flow는 전반적인 게임 기능의 흐름을 담았다.

Start Flow



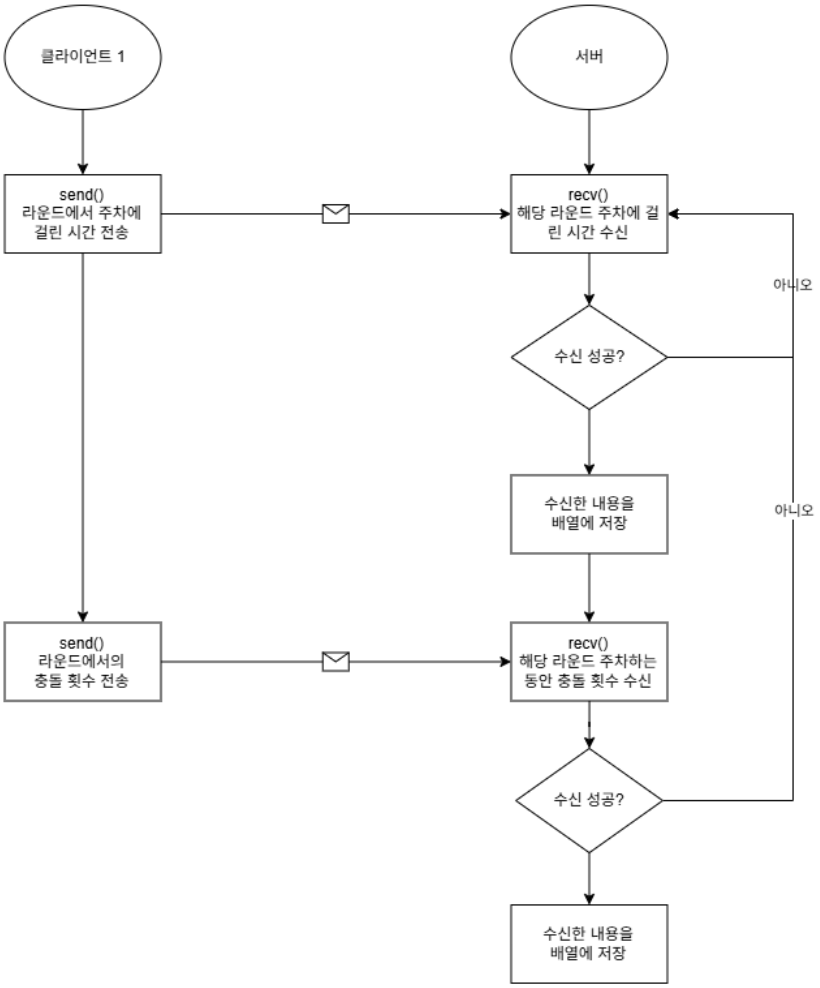
Start Flow는 게임 시작될 때의 기능 구현에 대한 흐름을 담았다. (매칭, 게임 시작 문구)

Collision Flow



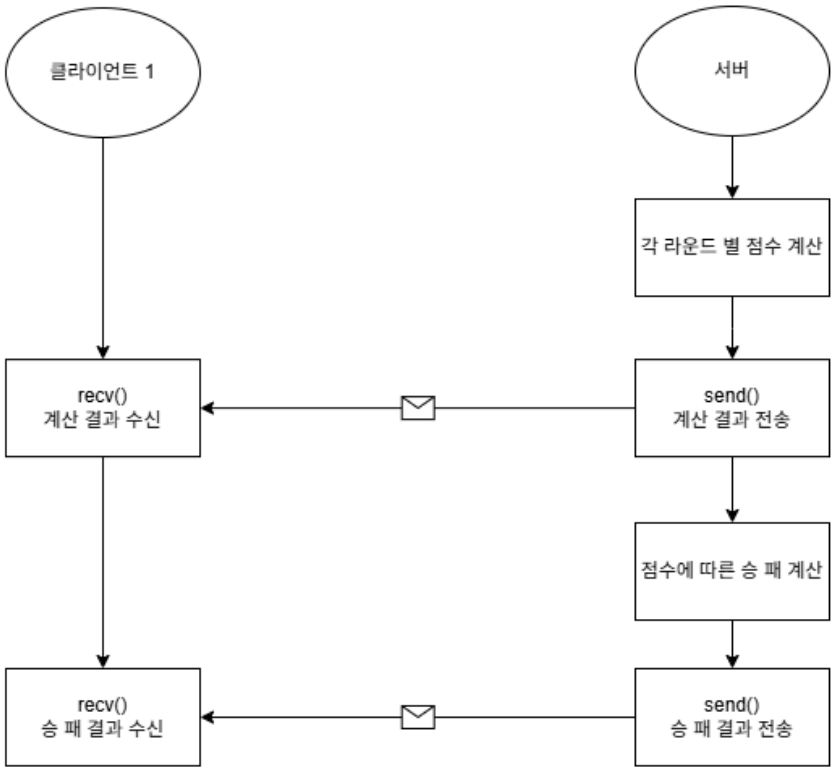
Collision Flow 는 게임 진행 중 충돌 상황을 담았다.

Each Round Score Flow



Each Round Score Flow 는 각 단계마다 점수를 서버에게 보내는 상황을 담았다.

Total Score Sum Flow



Total Score Sum Flow는 3단계의 게임 진행이 모두 끝나고 점수 환산 및 승패에 대해 담았다.

<Low-Level Design>

-구조체 선언-

[PlayerData]

플레이어의 시각적 상태를 정의

```
struct PlayerData {
// --- 식별자 ---
int             playerID;                       // 서버가 할당 (0 또는 1)
// --- 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       parkingTimeSeconds;         // 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_JoinRequest,             // (클->서) 게임 참가 요청
C2S_PlayerUpdate,          // (클->서) 내 위치/상태 주기적 업데이트
C2S_ReportParked,          // (클->서) "방금 주차 완료" 이벤트 전송
S2C_JoinAcknowledge,       // (서->클) 참가 수락 (내 ID 할당)
S2C_GameStart,               // (서->클) [브로드캐스트] 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_JoinRequest 는 내용 없이 전송)

// C2S_PlayerUpdate: 내 상태의 주기적 업데이트
struct C2S_PlayerUpdatePacket {
PacketType   type = C2S_PlayerUpdate;
int          playerID;
PlayerData   myData; // 위치, 회전, 기어, 바퀴 등
};

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

[S2C(Server To Client)패킷]

```
// S2C_JoinAcknowledge: 참가 요청에 대한 응답
struct S2C_JoinAcknowledgePacket {
PacketType   type = S2C_JoinAcknowledge;
int          yourPlayerID;          // 서버가 할당한 "나"의 ID (0 또는
1)
};

// S2C_GameStart: 2 명 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           serverElapsedSeconds;    // 서버 기준 경과 시간
PlayerData    playerStates[2];     // [0], [1] 플레이어의 실시간 위치 (상대방 렌더링용)
PlayerGameStats playerStats[2];     // [0], [1] 플레이어의 충돌 횟수, 주차 여부 (UI 표시용)
};

// S2C_GameOver: 양쪽 다 주차 완료 시 결과 전송
enum GameWinner : uint8_t {
Player_0,
Player_1,
Draw
};

struct S2C_GameOverPacket {
PacketType          type = S2C_GameOver;
GameWinner          winner;                 // 최종 승자
PlayerFinalScore    finalScores[2];         // [0], [1]번 플레이어의 상세 점수 내역
};
```

-네트워크 함수 API-

[클라이언트 API]

bool Network_InitAndJoin(const char* serverIP, int port, int& outPlayerID)

- 설명: main()에서 호출. 서버에 C2S_JoinRequest 를 보내고, S2C_JoinAcknowledgePacket 을 받아 outPlayerID 를 설정합니다. 매치메이킹 대기 상태로 들어갑니다.
- 리턴: true (참가 수락), false (서버 연결 실패)

void Network_SendUpdate(const PlayerData& myData)

- 설명: TimerFunction_UpdateMove 에서 주기적으로 호출되어 C2S_PlayerUpdatePacket 을 전송합니다.
- 매개변수: myData (현재 나의 PlayerData)

void Network_ReportParked()

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

void Network_ReceivePackets()

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

S2C_GameStart 수신 시: nextStage()를 호출하고 타이머를 시작합니다. S2C_GameStateUpdate 수신 시: playerStates[opponentID]를 이용해 상대방 차량을 렌더링하고, 서버가 계산해준 playerStats 로 UI(충돌 횟수, 경과 시간)를 갱신합니다. S2C_GameOver 수신 시: pause_mode = true 로 설정하고, finalScores 를 기반으로 승/패 결과 창을 띄웁니다.

[서버 API]

void Server_GameLoop()

- 설명: 서버의 메인 루프. 패킷 수신, 충돌 판정, 게임 상태 브로드캐스트를 반복합니다.
- 동작:

1. Server_ReceivePackets(): 클라이언트로부터 패킷을 수신하여 playerStates 갱신 및 C2S_ReportParked 처리.

2. Server_CheckAllCollisions(): [신규 함수 호출] 갱신된 playerStates 를 기반으로 모든 충돌(PvE, PvP)을 검사하고 playerStats[N].collisionCount 를 갱신.
3. Server_BroadcastGameState(): 갱신된 S2C_GameStateUpdatePacket 을 주기적으로 브로드캐스트.

void Server_ReceivePackets()

- 설명: 클라이언트로부터 모든 패킷을 수신하고 처리합니다.
- 주요 동작: C2S_JoinRequest 수신 시: Server_HandleNewConnection() 호출. C2S_PlayerUpdate 수신 시: 해당 플레이어의 PlayerData 를 서버 마스터 상태(playerStates)에 갱신. C2S_ReportParked 수신 시: 해당 플레이어의 isParked = true 로 설정, parkingTimeSeconds 기록. 이후 Server_CheckGameOver() 호출.

void Server_HandleNewConnection(Address clientAddress)

- 설명: 새 유저 접속 처리.
- 동작:

1. playerID 할당 (0 또는 1).
2. S2C_JoinAcknowledgePacket 전송.
3. 만약 2 명(0, 1)이 모두 접속했다면 Server_StartMatch() 호출.

void Server_StartMatch()

- 설명: 두 플레이어가 준비되면 매치를 시작합니다.

동작:

1. 스테이지(stageID) 결정, 매치 시작 시간 기록.
2. 모든 플레이어의 PlayerGameStats 초기화 (충돌 0, 시간 0, isParked=false).
3. S2C_GameStartPacket 을 두 플레이어에게 브로드캐스트.

void Server_CheckAllCollisions()

- 설명: Server_GameLoop 에서 주기적으로 호출되어 모든 충돌을 서버가 권위적으로 판정합니다.
- 동작:

1. 환경 충돌 (PvE): playerStates[0]의 위치와 **서버에 로드된 맵 데이터(벽, 장애물)**를 비교하여 충돌했는지 판정합니다. playerStates[1]의 위치와 맵 데이터를 비교하여 충돌했는지 판정합니다.
2. 플레이어 충돌 (PvP): playerStates[0]의 바운딩 박스(Bounding Box)와 playerStates[1]의 바운딩 박스를 비교하여 두 차량이 겹쳤는지(충돌했는지) 판정합니다.
3. 패널티 적용: 충돌이 감지되면(PvE 든 PvP 든), 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)

void Server_CheckGameOver()

- 설명: C2S_ReportParked 수신 시 호출되어, 두 플레이어가 모두 주차했는지 확인합니다.
- 동작: 만약 playerStats[0].isParked && playerStats[1].isParked 라면 Server_EndMatch() 호출.

void Server_EndMatch()

- 설명: 두 플레이어가 모두 주차를 완료했을 때 호출됩니다. 각 플레이어의 최종 점수를 계산하고, 승자를 판별하여 S2C_GameOverPacket 을 브로드캐스트합니다.
- 동작:

1. 두 플레이어(0, 1)의 PlayerGameStats 를 가져옵니다.
2. PlayerFinalScore score0 = Server_CalculateScore(0, playerStats[0]);를 호출합니다.

3. PlayerFinalScore score1 = Server_CalculateScore(1, playerStats[1]);를 호출합니다.
4. score0.finalScore 와 score1.finalScore 를 비교하여 GameWinner (Player_0, Player_1, or Draw)를 결정합니다.
5. S2C_GameOverPacket 을 생성하고, winner 변수와 finalScores 배열 (score0, score1)을 채웁니다.
6. 이 패킷을 두 플레이어에게 브로드캐스트합니다.

PlayerFinalScore Server_CalculateScore(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 parkingTimeSeconds 가 포함된 통계.
- 리턴: 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 10,000 점, 1 초당 100 점 감점. 최소 0 점.

int timeScore = 10000 - (int)(stats.parkingTimeSeconds * 100);

result.timeScore = (timeScore > 0) ? timeScore : 0; // 최소 0 점 보장


// --- (2) 충돌 감점 계산 ---
```

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

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

return result;
```

<팀원별 역할 분담>

이태현 – 서버

윤혜린 – 서버/클라

지민우 – 클라이언트

<일정표>

이태현

일	월	화	수	목	금	토
						11/1 S2C_ GameStateUpdat
11/2	11/3	11/4 S2C_ GameStartPacket	11/5	11/6	11/7 S2C_ JoinAcknowledgePacket	11/8 Player GameStats
11/9	11/10 Server_ GameLoop()	11/11	11/12 Server_ ReceivePackets()	11/13	11/14 Server_ HandleNewConnection (Address clientAddress)	11/15 Server_CheckAllC
11/16	11/17 Server_CheckGameOver()	11/18	11/19	11/20	11/21	11/22
11/23	11/24	11/25	11/26	11/27	11/28	11/29
11/30	12/1	12/2	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 C2S_ReportParkedPacket
11/2 GameWinne	11/3	11/4	11/5	11/6	11/7	11/8 S2C_GameOverPacket
11/9 PlayerFinalScore	11/10	11/11	11/12 Network_ReceivePackets()	11/13	11/14	11/15 Server_StartMatch()
11/16 Server_EndMatch()	11/17	11/18 PlayerFinalScore Server_ CalculateScore(int playerID, const PlayerGameStats& stats)	11/19	11/20	11/21	11/22
11/23	11/24	11/25	11/26	11/27	11/28	11/29
11/30	12/1	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 게임 그래픽스 수정1
11/2	11/3 게임 그래픽스 수정2	11/4 PlayerData	11/5 C2S_PlayerUpdatePacket	11/6	11/7	11/8 Network_InitAndJoin(const char* serverIP, int port, int& outPlayerID)
11/9	11/10 Network_SendUpdate(const PlayerData& myData)	11/11	11/12 Network_ReportParked()	11/13	11/14	11/15 PacketType
11/16	11/17	11/18	11/19	11/20	11/21	11/22
11/23	11/24	11/25	11/26	11/27	11/28	11/29
11/30	12/1	12/2	12/3	12/4	12/5	12/6
12/7	12/8	12/9	12/10	12/11		