네트워크 게임 프로그래밍

추진 계획서

학번	이름
2015182016	손 채영
2015182014	성 예솔
2015182045	홍 혜령

목차

1. 애플리케이션 기획

a. 게임 소개

ㄱ. 게임 이름: 메이플스토리

ㄴ. 게임 장르: Massively Multiplayer Online 롤 플레잉 게임

ㄷ. 게임 컨셉: 2인의 플레이어가 대규모 사냥 컨텐츠 '파티 퀘스트'에 참여하여 몬스터를 사냥하고 게임 머니 '메소' 를 얻는다.

ㄹ. 2D 횡 스크롤 방식

b. 기획 목표

ㄱ. 기존에 개발하였던 싱글 플레이 게임 '메이플스토리'에 TCP 서버-클라이언트의 개념을 적용하여 프로그램을 재 작성함으로써 소켓 프로그래밍의 동작 원리와 소켓 함수에 대한 깊은 이해도를 갖는다.

ㄴ. 서버와 클라이언트 간의 데이터 송수신 시 고려해야 할 항목들에 대한 고찰을 통하여 데이터 전송 방식을 이해하고 활용한다.

ㄷ. 싱글 플레이만이 가능하였던 기존 게임에 '파티 퀘스트'라는 컨텐츠를 추가함으로써, 멀티 스레드의 개념을 적용하여 다중 클라이언트를 동시에 처리할 수 있는 멀티 스레드 TCP 서버를 작성하고 멀티 스레드 프로그래밍의 기본 원리를 이해하고 활용한다.

c. 게임 플레이

ㄱ. 타이틀 화면에서 접속할 서버의 IP 주소를 입력한다.

ㄴ. 캐릭터 생성 화면에서 플레이 할 캐릭터의 ID와 직업을 정한 뒤 캐릭터를 생성한다.

ㄷ. 캐릭터 생성 후 '필드' (로비 기능)에서 다른 플레이어의 접속을 기다린다.

ㄹ. 2인의 플레이어가 모여 '파티' 를 구성한 후 NPC를 통해 사냥터에 입장한다.

ㅁ. 파티 단위로 몬스터를 사냥하여 NPC가 요구한 퀘스트 완료 조건을 충족하면 파티 퀘스트가 끝난다.

ㅂ. 파티 퀘스트 완료 후, 파티 내 플레이어를 대상으로 킬 관여율을 비교하고 순위를 매기는 랭킹 시스템을 통하여 게임 결과를 알 수 있는 창이 출력된다.

ㅅ. 게임 결과 창에서 '확인' 버튼을 누르면 다시 필드로 돌아간다.

d. 조작 방법

ㄱ. 이동: 방향키 ←, →

ㄴ. 엎드리기: 방향키 ↓

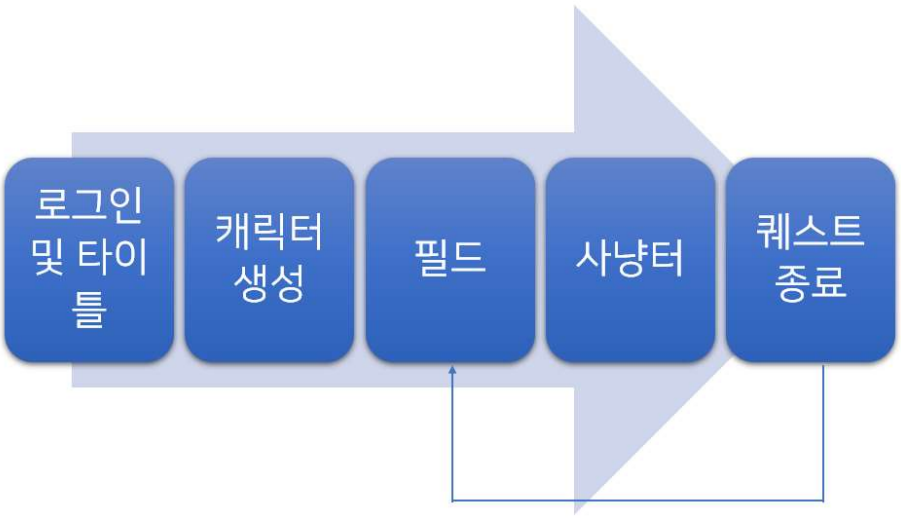
ㄷ. 밧줄 타기: 방향키 ↑

ㄹ. 점프: SPACE 키

ㅁ. 기본 공격: Q

ㅂ. 스킬 1, 2, 3: W, E, R

e. 게임 흐름



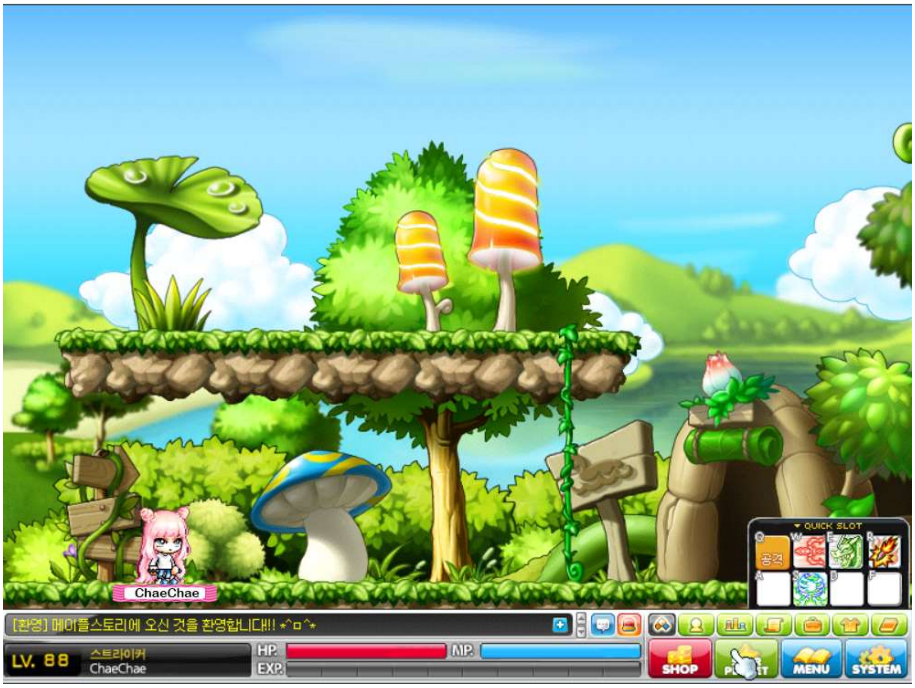
ㄱ. 로그인 및 타이틀 창

- 게임 타이틀 화면 및 IP 접속 창이 출력되고 접속하고자 하는 서버 IP 주소를 입력할 수 있다.

ㄴ. 캐릭터 생성 씬

- 캐릭터 생성 시 직업을 선택할 수 있고, ID 를 키보드로 입력하여 정할 수 있다.

ㄷ. 필드 (로비) 씬



- 서버에 접속해 있는 플레이어의 ID 와 직업에 해당하는 캐릭터 그래픽이 모든 클라이언트의 화면에 출력된다.
- 한 명의 클라이언트가 맨 우측에 위치한 NPC 를 클릭하면 파티 퀘스트가 수락되고, 접속 중인 모든 클라이언트가 사냥터로 들어간다. 2 인의 클라이언트가 접속 중이어야 한다.

ㄹ. 사냥터 씬 (게임 플레이)

- NPC 로부터 초록 버섯과 커플 버섯을 n 마리 잡아 줄 것을 부탁 받는다.
- 몬스터의 생성 및 소멸 상태에 맞게 그래픽이 출력된다.
- 모든 클라이언트는 모든 몬스터의 상태 정보를 갖는다.
- 퀘스트 조건에 충족될 시 결과 창이 뜨고 파티 퀘스트가 종료된다.
- 결과 창에서 '확인' 버튼을 누르면 다시 로비 씬으로 돌아간다.

2. High-Level 디자인

a. 서버 설계 내용

ㄱ. 서버 실행

- 서버 프로그램을 실행하면 Winsock 초기화를 한다. 대기 전용 소켓을 생성하고 bind() 함수를 통해 서버의 지역 IP 주소와 지역 포트 번호를 설정하고, listen() 함수를 통해 대기 소켓을 클라이언트의 접속을 기다리는 LISTENING 상태로 만든다.
- 클라이언트는 캐릭터 닉네임(ID)과 직업을 정한 뒤 connect() 함수를 통하여 서버와 연결해 온다. 그리고 나서 서버는 해당 클라이언트와 통신하는 통신 전용 소켓을 생성한다. 이 전용 소켓을 이용하여 클라이언트의 캐릭터 생성 완료 및 필드로 접속하라는 메시지를 보낸다. 또한 서버는 캐릭터의 닉네임, 직업 정보 값을 받아 Object Vector에 저장한다. Object의 순차적 접근에 용이하기 때문에 Vector을 이용해 관리한다.

ㄴ. 멀티 스레드를 이용한 데이터 통신

- 접속한 클라이언트에게 전용 스레드가 할당되어 통신 소켓을 생성하고, 스레드 내부에서 클라이언트의 'PlayerInfo' 구조체를 받아온다. PlayerInfo 구조체는 사용자 정의 구조체로서, 플레이어의 Ready 여부와 능력치 정보, 클라이언트 인덱스 값, 위치 등의 정보를 포함한다. (Low-Level 디자인 항목에서 상세히 기술)
- 클라이언트가 보낸 키 Input 값과 PlayerInfo 구조체를 받아서 Input 값에 따라 이동된 좌표 값 또는 스킬 이펙트 정보 등의 작용을 다시 클라이언트에 보낸다.
- 서버의 스레드 내부에서 플레이어-몬스터, 플레이어의 이펙트-몬스터, 플레이어-메소의 충돌 등 모든 충돌을 체크하는 함수를 호출하여 오브젝트 간의 상호작용에 따라 변경된 정보 값을 클라이언트에 보낸다. 이 정보 값은 주로 오브젝트의 생성과 소멸을 담고 있다.
- 퀘스트를 완료하게 되면 게임 결과 창을 띄우고 파티 퀘스트가 종료된다. 서버는 게임 종료 조건에 충족되었는지 지속적으로 확인하고, 각각 클라이언트에게 모든 플레이어들의 킬 관여 정보와 이에 따른 퀘스트 결과 화면을 띄우라는 메시지를 보낸다.
- 클라이언트의 종료 (정상, 강제) 메시지를 받게 될 경우, 해당 클라이언트의 스레드의 소켓을 종료하고 스레드를 제거한다.

b. 클라이언트 설계 내용

ㄱ. 로그인 및 타이틀

- 클라이언트의 Winsock을 초기화 한다. 사용자의 키보드 입력을 통해 클라이언트가 접속할 서버의 IP 주소를 입력 받고 저장한다.

ㄴ. 캐릭터 생성

- 사용자가 플레이 할 캐릭터의 닉네임(ID)과 직업을 입력 및 선택하여 만들어진 캐릭터 정보를 서버로 보낸다. 서버는 해당 정보 값을 받아 Object Vector에 저장한다. Object의 순차적 접근에 용이하기 때문에 Vector을 이용해 관리한다.
- 캐릭터 생성이 완료되어 connect() 함수를 통해 서버에 연결을 요청한다. 접속이 허용되면 서버로부터 캐릭터 생성 완료 및 필드로 접속하라는 메시지를 받게 되고 필드 씬으로 넘어간다.

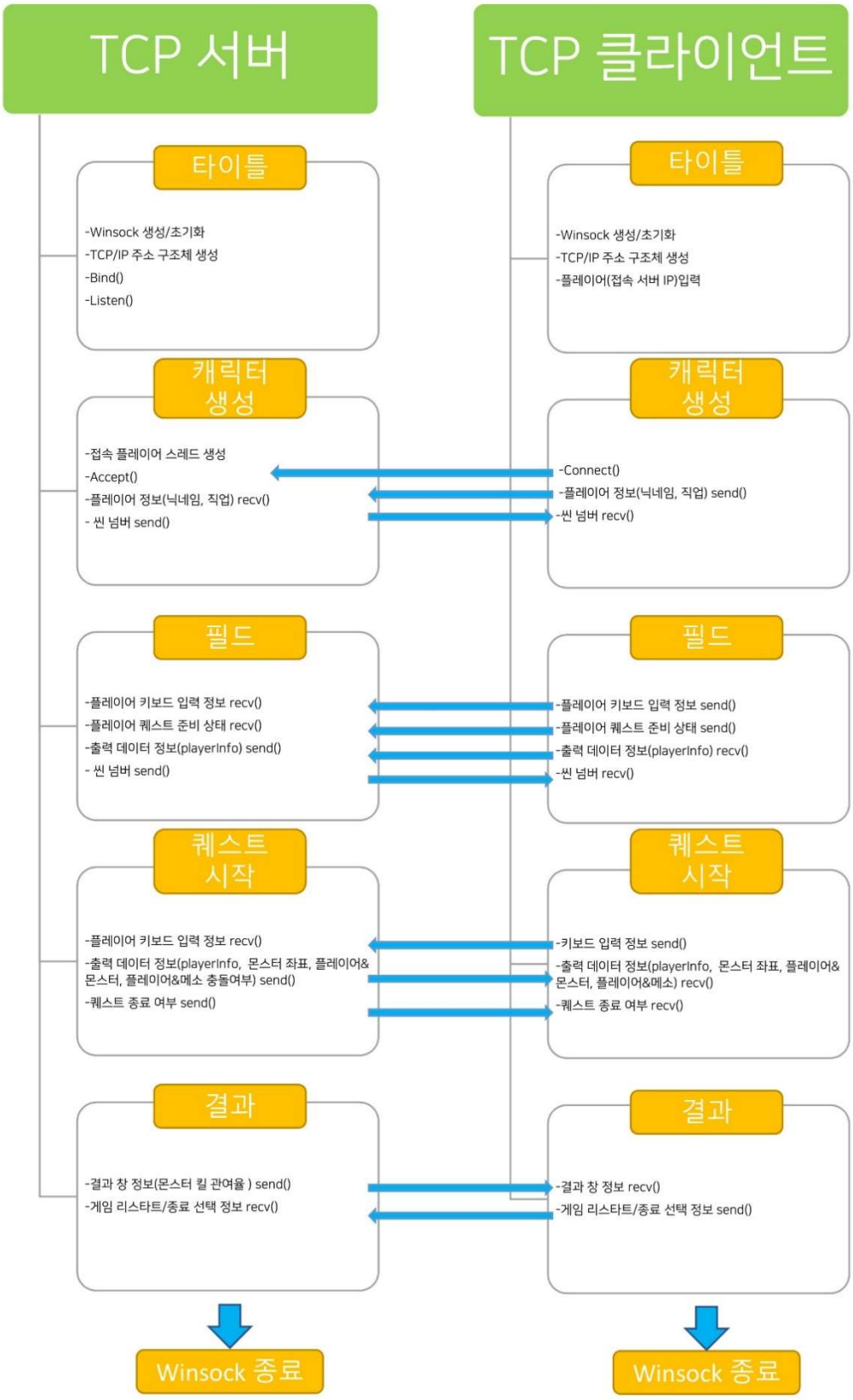
ㄷ. 게임 플레이 - 필드

- 클라이언트는 서버의 Send() 함수를 통해 다른 클라이언트의 PlayerInfo 구조체를 실시간으로 받고 이 정보를 통해 다른 클라이언트의 캐릭터 그래픽 및 모션을 출력한다.
- 접속한 플레이어는 필드 내 NPC를 클릭하여 자신의 파티 대기 상태를 true 값으로 바꿀 수 있다. 설정 상 사냥터는 2명의 플레이어가 파티를 이루어야 참가가 가능한데, 파티 대기 상태가 true 인 플레이어가 2명이 되면 해당하는 클라이언트는 모두 자동으로 사냥터 씬으로 넘어간다.

ㄹ. 게임 플레이 - 사냥터

- 사냥터에 입장하면 각 플레이어는 서버로부터 몬스터의 'MonsterInfo' 구조체와 다른 클라이언트의 PlayerInfo 구조체를 실시간으로 수신하고 정보에 맞게 그래픽을 출력한다.
- 플레이어의 이동 및 스킬 사용을 위한 사용자의 키 입력 이벤트가 생기면, 클라이언트의 키 Input 값과 PlayerInfo 구조체를 서버로 전송한다. 서버는 그 정보 값을 받아 이동된 좌표 값 또는 스킬 이펙트 정보를 다시 모든 클라이언트로 보낸다.
- 파티가 퀘스트를 완료하면 서버에게 퀘스트 완료 메시지와 킬 관여율 정보를 수신하고 결과 화면을 출력한다. 이때 퀘스트 결과 화면에는 파티원의 정보와 킬 관여율에 따른 랭킹이 출력된다.
- 각 플레이어는 퀘스트 결과 창에서 '확인' 버튼을 누르면 다시 필드로 돌아간다. 이때 만들어진 파티원의 대기 상태는 모두 false 값으로 초기화된다.

3. Low-Level 디자인



a. 데이터 송수신 패킷 정의

ㄱ. 데이터를 송수신할 때, 다음과 같은 구조체를 정의한 뒤 패킷에 담아 보낸다. 서버는 클라이언트의 구분을 위해 클래스 패킷의 길이 (고정 길이)에 데이터 (가변 길이)를 붙여 보낸다.

이름	설명
typedef struct MySize { float cx; float cy; } MYSIZE;	오브젝트의 사이즈 정보를 나타낸다. 총 8바이트 크기를 전송한다. 화면에서 그래픽을 출력할 때 주로 쓰인다.
typedef struct MyPoint { float x; float y; } MYPOINT;	오브젝트의 위치 좌표 정보를 나타낸다. 총 8바이트 크기를 전송한다. 화면에서 그래픽을 출력할 때와 오브젝트 간의 충돌을 체크할 때 주로 쓰인다.
~~typedef struct Info {~~ ~~OBJECT_TYPE type;~~ ~~MYPOINT pt;~~ ~~MYSIZE size;~~ ~~FRAME frame;~~ ~~int hp;~~ ~~}INFO;~~	~~오브젝트가 공통으로 갖고 있어야 하는 정보 (오브젝트 타입, 좌표, 사이즈, 애니메이션 프레임 정보, HP) 를 나타낸다.~~ ~~총 24바이트 크기를 전송한다.~~
typedef struct PlayerInfo { char id; int key; bool ready; MYPOINT pt; MYSIZE size; PLAYER_JOB job; int hp; int money; FRAME frame; float attackAccValue; ~~INFO info;~~ ~~int mp;~~ } PLAYERINFO;	플레이어 오브젝트만이 갖고 있어야 하는 정보 (아이디, 클라이언트 키 값, 파티 퀘스트 대기 상태, 위치 좌표, 플레이어의 사이즈, 플레이어의 직업, hp, 소지금, 프레임 정보, 누적 데미지 양, ~~INFO 구조체, mp~~) 를 나타낸다. 총 45바이트 크기를 전송한다. 상태 변화를 다른 클라이언트에 알릴 때 정보를 전송하는 용도로 주로 쓰인다.
typedef struct MonsterInfo { int key; MYPOINT pt;	몬스터 오브젝트만이 갖고 있어야 하는 정보 (몬스터 인덱스 key 값, 좌표, 사이즈, hp, 소지금, ~~Info 구조체, 인덱스 num~~) 를 나타낸다. 총 28바이트 크기를 전송한다.

MYSIZE size; int hp; int money; ~~INFO info;~~ ~~Int num;~~ } MONSTERINFO;	접속 중인 클라이언트들에게 몬스터의 생성 및 소멸, 상태 변화 등의 정보를 전송하는 용도로 주로 쓰인다.

b. 서버 관련 함수

	이름	설명
송신	void SendNewPlayerInLobby(PLAYERINFO info)	- 새로 접속한 클라이언트에 인덱스 값(key)을 매긴 후, 해당 클라이언트의 PlayerInfo를 모든 클라이언트에게 송신한다.
	void SendPlayerInfo(PLAYERINFO info)	- 서버에 접속 중인 모든 클라이언트의 PlayerInfo를 다른 클라이언트에게 송신한다.
	void SendQuestCompleted(bool completed, vector<PLAYERINFO> *p)	- 퀘스트 완료 조건 충족이 되었을 때 호출된다. - 퀘스트가 완료되었음을 플레이어들에게 알리고, 킬 관여율 정보를 보내기 위해 서버에 저장된 vector<PlayerInfo>의 주소 값을 모든 클라이언트에게 전송한다.
	void SendSceneChanged(SCENE_TYPE e)	- 클라이언트에게 어떤 씬으로 화면 전환을 할지 알린다.
	void SendCurrentMonsterInfo(MONSTERINFO info)	- 이동 또는 특정 동작을 하는 몬스터의 정보를 모든 클라이언트에 송신한다.
	void SendNewMonsterInfo(MONSTERINFO info)	- 새로 생성된 몬스터의 정보를 모든 클라이언트에 송신한다.
	Void SendTimerInfo(int time)	- 게임 서버의 시간을 클라이언트에게 송신한다. (몬스터 이동패턴 구현에 이용)
수신	void RecvNewPlayerInfoAccesed(int id, JOB_TYPE e)	- 새로 접속한 클라이언트의 ID와 job을 받는다.

	void RecvKeyInput(int key, int input)	-클라이언트로부터 키 입력 정보를 받고, key 값을 통하여 저장해 둔 vector<PLAYERINFO> 에 접근하여 해당하는 클라이언트의 상태를 바꿔 준다.

c. **클라이언트 관련 함수**

	이름	설명
송신	void SendNewPlayerInfoAccessed (int ID, enum job)	- 서버에 접속한 후, 서버에게 새 접속을 알리며 캐릭터 생성 창에서 정한 ID와 job 정보를 보낸다.
	void SendKeyInput (int key, int input)	- 플레이어가 키보드 입력을 할 때 호출된다. - 클라이언트의 키보드 입력 정보와 클라이언트의 key(인덱스) 값을 서버로 전송한다. - 플레이어의 좌우 이동, 동작, 스킬 공격, 기본 공격이 해당한다.
수신	void RecvPlayersInfoForRender (PLAYERINFO info)	- 변경되는 클라이언트들의 정보에 따라 화면을 갱신한다. - 서버에서 받은 다른 플레이어들의 갱신된 정보들을 key 값을 통해 몇 번 클라이언트인지 찾고, 갱신하여 render 한다.
	void RecvSceneChanged(enum SCENE)	- 서버로부터 어떤 씬으로 화면을 전환할지 알림 받는다.
	void RecvQuestCompleted(bool completed, vector<float> vec)	- 퀘스트 완료 조건 충족이 되었을 때 호출한다. - 퀘스트가 완료되었음을 서버로부터 알림 받고, 킬 관여율을 저장해 둔 vector를 넘겨 저장하여 모든 클라이언트의 킬 관여율과 랭크를 저장하여 출력한다.
	Void RecvTimerInfo(int time)	- 게임 서버 시간을 받는다.

d. 기타 함수

	이름	설명
충돌 체크	void RectCollision(INFO dst, INFO src)	- 오브젝트 간의 충돌을 검사한다.
데이터 동기화 스레드	DWORD WINAPI ClientThread(SOCKET clientsock)	- 각 클라이언트와의 통신을 담당할 스레드 함수이다. - RecvKeyInput() 함수에서 키 값을 수신한다. - CalculateThread() 함수에서 연산된 정보들을 클라이언트로 송신한다. - 초기화된 클라이언트의 정보도 보내준다.
	DWORD WINAPI CalculateThread()	- 각종 연산을 담당하기 위한 스레드 함수이다. - 충돌 체크 함수들을 관리한다.

e. 스레드 동기화 관리

ㄱ. 공유 자원으로는 vector<PLAYERINFO>, vector<MONSTERINFO>가 있다.

ㄴ. 흐름

- 서버는 클라이언트로부터 Recv 해서 공유 자원에 저장한다.
- 갱신한 공유 자원을 다른 클라이언트에 Send (동기화) 한다.

ㄷ. Recv 후 Send를 하는 순서를 지켜야 하기 때문에 **이벤트 방식**을 사용한다.

ㄹ. Send Event, Recv Event 두 이벤트로 설계한다.

4. 팀원 별 역할 분담

a. 서버

구현 내용	성예솔	손채영	홍혜령
기본적인 네트워크 환경 구축	✔	✔	✔
타이머 관리하는 스레드 작업 및 관련 함수 구현	✔		
클라이언트 간 접속 동기화		✔	
클라이언트 키 입력 recv		✔	
클라이언트 간 캐릭터 위치, 스킬, 이펙트 동기화		✔	
클라이언트 간 파티 접속 동기화			✔
몬스터 생성 및 소멸 동기화	✔		
몬스터 위치 동기화	✔		
충돌체크 처리 및 모션 동기화			✔
메소 생성 및 소멸 동기화			✔
게임 정상 종료 및 강제 종료 처리			✔

b. 클라이언트

구현 내용	성예솔	손채영	홍혜령
기존 게임 유지 보수	✔	✔	✔
게임 시스템 설계 및 구현		✔	
게임 UI 설계 및 디자인			✔
씬 구조 관리 및 구성		✔	
몬스터 종류 추가	✔		
플레이어 아이디 및 캐릭터 생성	✔		
기본적인 네트워크 환경 구축		✔	
서버로부터 받은 타이머 함수 적용	✔		
클라이언트 간 접속 동기화		✔	
플레이어의 위치, 스킬, 이펙트 동기화		✔	
파티창 구현(아이디, 대기 상태 등)			✔
몬스터 생성 및 소멸 동기화	✔		
몬스터 위치 및 모션 동기화	✔		
메소의 생성 및 소멸 동기화			✔
충돌체크 후 모션, 이펙트 동기화			✔
게임 결과 창			✔
게임 정상 종료 및 강제 종료 처리			✔

5. 개발 환경

- 운영체제: Windows 10
- 통신 프로토콜: TCP/IP
- 사용 언어: C, C++
- 컴파일러: Visual Studio 2017
- 라이브러리: Win API, fmod
- 관리 툴: GitHub

6. 개발 일정

	성 예솔	손 채영	홍 혜령
10/31	기존 게임 유지 보수	서버 프로젝트 기반 구축	기존 게임 유지 보수
11/1	기존 게임 유지 보수	클라이언트, 서버 - 전반적 게임 시스템 설계 (기존에 클라이언트에 몰려 있던 클래스들을 적절한 위치에 재분배)	기존 게임 유지 보수
11/2	기존 게임 유지 보수	게임 시스템 구현 작업 (설계 내용을 토대로)	기존 게임 유지 보수
11/3	기본적인 네트워크 환경구축	기존 게임 유지 보수 (디버깅, 코드 최적화)	필드 내 UI 설계 및 디자인
11/4	기본적인 네트워크 환경구축	기존 게임 유지 보수 (플레이어 직업 및 이펙트 추가)	필드 내 UI 설계 및 디자인

11/5	타이머 관리하는 스레드 생성	씬 구조 관리 및 구성	NPC 구현
11/6	서버 타이머 송신 구현	클라이언트 내 기본 네트워크 환경 구축	NPC 구현
11/7	클라이언트 타이머 수신 구현	클라이언트 내 기본 네트워크 환경 구축	파티창 UI 설계 및 디자인
11/8	몬스터 패킷 데이터 정의	서버의 플레이어 데이터 통신 스레드 구현	접속한 플레이어 정보 관리
11/9	서버의 몬스터 생성, 소멸 송신설계 및 구현	서버의 플레이어 데이터 통신 스레드 구현	파티창 내 플레이어 아이디, 상태 등 구현

11/10	서버의 몬스터 생성, 소멸 송신설계 및 구현	플레이어 간 접속 동기화(클라이언트 내 작업)	파티창 내 플레이어 아이디, 상태 등 구현
11/11	서버의 몬스터 생성, 소멸 송신설계 및 구현	플레이어 간 접속 동기화(클라이언트 내 작업)	플레이어 간 파티창 동기화
11/12	클라이언트의 몬스터 생성, 소멸 수신설계 및 구현	플레이어 간 접속 동기화(서버 내 작업)	플레이어 간 파티창 동기화
11/13	클라이언트의 몬스터 생성, 소멸 수신설계 및 구현	플레이어 간 접속 동기화(서버 내 작업)	사냥터 내 UI 설계 및 디자인
11/14	클라이언트의 몬스터 생성, 소멸 수신설계 및 구현	플레이어 간 접속 동기화(서버 내 작업)	사냥터 내 UI 설계 및 디자인

11/15	**중간 점검 및 코드 일체화**		
11/16	스레드 동기화 구현	플레이어 간 접속 동기화 테스팅, 마무리 작업	캐릭터-몬스터 충돌 체크
11/17	스레드 동기화 구현	키 입력 정보 동기화를 위한 프로토콜 정의 및 설계	캐릭터-몬스터 충돌 체크
11/18	스레드 동기화 구현	키 입력 정보 동기화 (클라이언트 키 입력 send)	충돌 후 모션
11/19	휴식	키 입력 정보 동기화 (클라이언트 키 입력 send)	캐릭터 이펙트-몬스터 충돌 체크

11/20	몬스터 이동패턴 및 위치 정보 패킷 데이터에 추가	키 입력 정보 동기화 (서버 키 입력 recv)	캐릭터 이펙트-몬스터 충돌 체크
11/21	서버 몬스터 위치 송신설계 및 구현	키 입력 정보 동기화 (서버 키 입력 recv)	충돌 후 모션
11/22	서버 몬스터 위치 송신설계 및 구현	키 입력 정보 동기화 테스팅, 마무리 작업	메소 생성 및 위치 동기화
11/23	서버 몬스터 위치 송신설계 및 구현	휴식	메소 생성 및 위치 동기화
11/24	클라이언트 몬스터 위치 수신설계 및 구현	클라이언트 간의 동기화를 위한 프로토콜 정의 및 설계	캐릭터-메소 충돌 체크

11/25	클라이언트 몬스터 위치 수신설계 및 구현	플레이어 간 위치 동기화 (클라이언트 send)	캐릭터-메소 충돌 체크
11/26	클라이언트 몬스터 위치 수신설계 및 구현	플레이어 간 위치 동기화 (서버 recv)	메소 소멸 동기화
11/27	클라이언트 몬스터 애니메이션 수신설계 및 구현	플레이어 간 위치 동기화 (서버가 다른 클라이언트에 send)	결과창 UI 설계 및 디자인
11/28	클라이언트 몬스터 애니메이션 수신설계 및 구현	플레이어 간 스킬 동기화 (클라이언트 send)	파티원 정보 관리
11/29	몬스터 hp<0 소멸 처리	플레이어 간 스킬 동기화 (서버 recv)	결과창 내 파티원 정보, 킬 관여율 등 구현

11/30	몬스터 hp<0 소멸 처리	플레이어 간 스킬 동기화 (서버가 다른 클라이언트에 send)	결과창 내 파티원 정보, 킬 관여율 등 구현
12/1	**중간 점검 및 코드 일체화**		
12/2	스레드 동기화 추가 작업 및 개선	플레이어 간 위치, 스킬 동기화 테스팅 및 추가 작업	결과창 내 킬 관여율에 따른 랭킹
12/3	스레드 동기화 추가 작업 및 개선	플레이어 간 이펙트 동기화 (클라이언트 send)	결과창 내 킬 관여율에 따른 랭킹
12/4	스레드 동기화 추가 작업 및 개선	플레이어 간 이펙트 동기화 (서버 recv, 다른 클라이언트에 send)	게임 정상 종료 처리

12/5	스레드 동기화 추가 작업 및 개선	클라이언트 간의 이펙트 동기화 테스팅 및 마무리 작업	게임 정상 종료 처리
12/6	미완성 내용 추가 작업	미완성 내용 추가 작업	게임 강제 종료 처리
12/7	미완성 내용 추가 작업	미완성 내용 추가 작업	게임 강제 종료 처리
12/8	테스트 및 버그 수정		
12/9			

12/10	
12/11	최종 발표