P o r t f o l i o

김기윤

서 버 프 로 그 래 머

Phone. 010 - 5100 - 2974

E-mail. kimkiyoun33@gmail.com

Education. 한국공학대학교(구 한국산업기술대학교) 게임공학과 졸업예정(2023.02)

Birth. 1997. 09.10

Address. 경기도 의정부시

Skills

언어

- C
- C++11
- python
- Lua Script
- java script

클라이언트

- DirectX 12
- OpenGL
- Unity
- Unreal4

서버

- IOCP
- 멀티쓰레딩
- Ms SQL

Etc.

- github
- Google cloud platform

CONTENTS

01 SSU(졸업작품) - 팀 프로젝트

개발 환경
- 사용 언어 : C, C++ 11, Lua
- 클라이언트 : Direct X 12
- 서버 : IOCP
- MS SQL
- Unity(Terrain & Object 배치)

02 게임 서버 프로그래밍 - 개인 프로젝트

개발 환경
- 사용 언어 : C, C++ 11, Lua
- 클라이언트 : SFML
- 서버 : IOCP
- MS SQL

03 네트워크 게임 프로그래밍 - 팀 프로젝트

개발 환경
- 사용 언어 : C, C++ 11
- 클라이언트 : 윈도우 프로그래밍
- 서버 : Socket Thread

04 Client Projects

클라이언트 프로젝트
- Python : 팀 프로젝트
- OpenGL : 팀프로젝트
- Unreal Engine4 : 개인 프로젝트
- Unity(UI 개선 프로젝트) : 팀 프로젝트

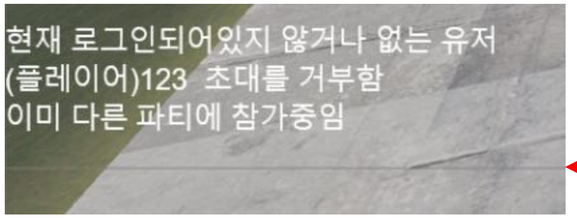
SSU(졸업작품)

게임 장르	MMORPG
작업 기간	2022.01~2022.07
작업 인원	3명 - 클라이언트 1명 - 서버 2명
구현내용 (팀 내 역할)	서버 프로그래머 - 파티 시스템 - 몬스터 : 길찾기(A* 알고리즘), 로밍 - Raid 보스 - 섹터 분할(최적화 및 동접 향상)
깃허브	https://github.com/hjs0913/SSU
비디오 링크	https://www.youtube.com/watch?v=QtUVavobUL8

파티시스템 / 레이드 시작

01

● 파티시스템



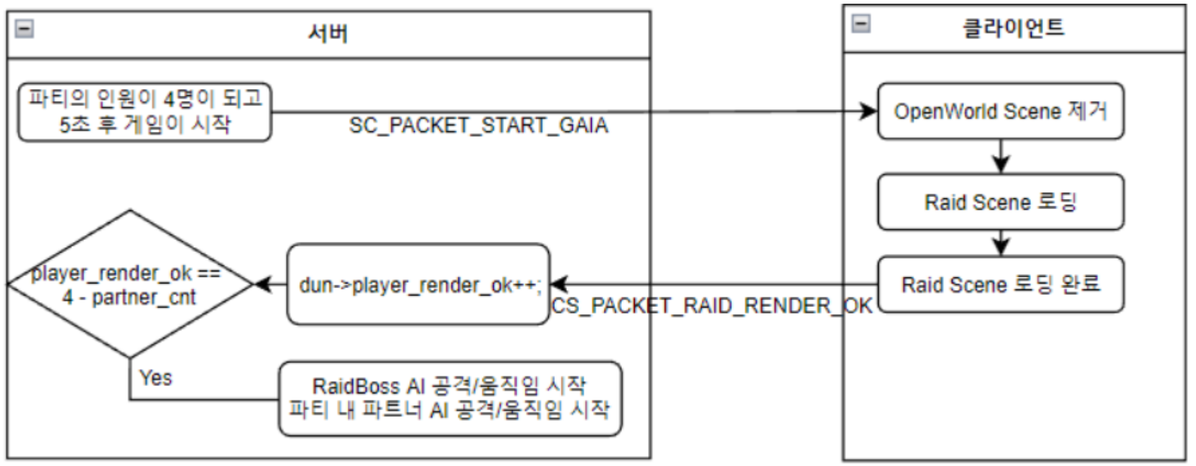
기획의도에 충실하게 맞추어 구현

- 파티 시스템 기능 : 방만들기, 방들어가기/방나가기, AI추가
- 방만들기, 방들어가기를 하고자하는 플레이어는 어떠한 방(파티)에도 참가하고 있지 않아야 한다.
- 방은 동료 AI를 제외하고 플레이어가 한명도 없을 때 파괴된다.
- 파티원이 4명이 되면 5초 후 게임시작, 5초내에 파티원 중 누군가가 파티를 나가면 게임시작이 안된다.

구현하면서 발생한 문제점을 기획자(팀원)와 협의하면서 수정

- 방(파티) 입장 후 채팅을 하면 전체 채팅이 아닌 파티 채팅으로 변경
- 초대하기 기능을 넣으면 보다 좋은 게임이 될 것이라 생각하여 회의 후 추가
- 초대하기 버튼을 누른 후 초대하고자하는 플레이어의 닉네임을 입력하면 초대장이 발송되며 초대받은 플레이어는 수락/거부가 가능하며 10초가 지나면 자동을 거부된다.
- 초대에 실패할 시 **채팅창에 초대실패 이유 제공(초대하는 사람이 초대가 제대로 됐는지 확인할 수 있도록)**

● 레이드 시작



- 문제점 : 클라이언트에서 레이드 인던이 로딩되기 전에 서버에서 게임이 먼저 진행되는 문제점을 발견

- 해결방안 : 파티원 클라이언트가 렌더링을 완료하면
CS_PACKET_RAID_RENDER_OK패킷을 보내주고
서버에서는 player_render_ok++를 해준다
'player_render_ok == 4 - 파트너의 수'이면 AI를 움직이기 시작한다

Raid 보스 AI, 몬스터 AI

● Raid 보스 AI

- Lua스크립트를 통해 보스의 스탯, 정보 쉽게 조정
- 3개의 패턴 존재(랜덤으로 3개중 1개가 실행)
- 일부 패턴 Lua스크립트로 작성

```
my_id = 99999; my_element = 0; my_lv = 100; my_name = "가이아"; my_hp = 1000000; my_physical_attack = 350;
my_magical_attck = 400; my_physical_defence = 500; my_magical_defence = 350; my_basic_attack_factor = 50;
my_defence_factor = 0.0018; my_x = 300; my_y = 0; my_z = 300;
p1_first_position_x = 0; p1_first_position_z = 0; p1_second_position_x = 0; p1_second_position_z = 0;
p1_third_position_x = 0; p1_third_position_z = 0; p1_fourth_position_x = 0; p1_fourth_position_z = 0;

function set_uid(id)
   my_id = id;
   return my_element, my_lv, my_name, my_hp, my_physical_attack, my_magical_attck,
        my_physical_defence, my_magical_defence, my_basic_attack_factor, my_defence_factor;
End

function pattern_one_set_position(x, z, look_x, look_z, right_x, right_z)
    p1_first_position_x = x + look_x*50; p1_first_position_z = z + look_z*50;
    p1_second_position_x = x - look_x*50; p1_second_position_z = z - look_z*50;
    p1_third_position_x = x + right_x*50; p1_third_position_z = z + right_z*50;
    p1_fourth_position_x = x - right_x*50; p1_fourth_position_z = z - right_z*50;
    return p1_first_position_x, p1_first_position_z, p1_second_position_x, p1_second_position_z,
       p1_third_position_x, p1_third_position_z, p1_fourth_position_x, p1_fourth_position_z;
end

function pattern_one_set_damage(player)
    player_x = API_get_x(player); player_z = API_get_z(player);
    dis1=math.sqrt(math.pow((player_x - p1_first_position_x),2) + math.pow((player_z - p1_first_position_z),2));
    dis2=math.sqrt(math.pow((player_x - p1_second_position_x),2)+math.pow((player_z - p1_second_position_z),2));
    dis3=math.sqrt(math.pow((player_x - p1_third_position_x),2)+math.pow((player_z - p1_third_position_z), 2));
    dis4=math.sqrt(math.pow((player_x - p1_fourth_position_x),2)+math.pow((player_z - p1_fourth_position_z), 2));
    p_hp = API_get_hp(player);
    if((dis1 < 20) or (dis2 < 20) or (dis3 < 20) or (dis4 < 20)) then
        return p_hp - 2000;
    else
        return p_hp;
    end
end
```

첫번째 패턴의 위치를 잡는다

첫번째 패턴의 위치와 플레이어 위치를 계산해 데미지를 준다

●몬스터 AI

- Lua스크립트를 통해 스탯, 정보 조정

  (보스와는 다르게 오픈월드에서 스폰될 위치를 설정)

```
my_id = 99999; my_element = 6; my_lv = 19; my_name = "타락한 개구리"; my_hp = 120000;
my_physical_attack = 180; my_magical_attck = 0; my_physical_defence = 300;
my_magical_defence = 130; my_basic_attack_factor = 10; my_defence_factor = 0.0002;
my_x = 0; my_y = 0; my_z = 0;

min_x = 1900; max_x = 2200; min_z = 3080; max_z = 3380;

function set_uid(id, x, y, z)
   my_id = id; my_x = x; my_y = y; my_z = z;
   return my_element,my_lv, my_name, my_hp, my_physical_attack, my_magical_attck,
        my_physical_defence, my_magical_defence, my_basic_attack_factor,
            my_defence_factor;
End

function event_npc_move(player)
   player_x = API_get_x(player); player_z = API_get_z(player);
   x = API_get_x(my_id); z = API_get_z(my_id);
   if (math.abs(my_x - x) > 120) or (math.abs(my_z - z) > 120) then
      return false;
   else
      -- 쫓아가는 것은 C로 짠다
      return true;
   end
end
```

스폰될 위치 x,z의 최소값과 최댓값

최대 이동거리 검사

- my_x, my_y, my_z에 처음 스폰위치를 저장시켜서 이후 최대 이동 거리 판단에 이용

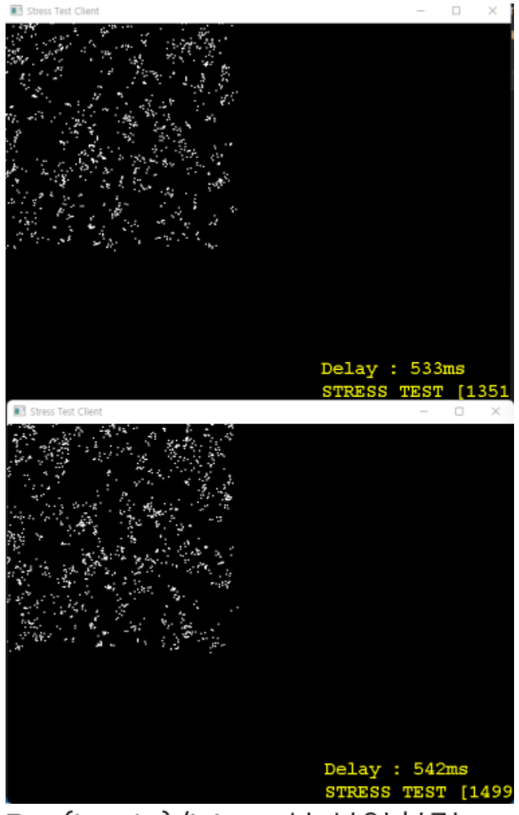
섹터분할(최적화 및 동접향상)

01

- 동접을 늘리기 위해 도입
- 객체 이동시 시야판정을 위해 모든 객체를 검색해야하는 오버헤드를 줄이는 역할을 함
- 전체 월드를 섹터로 분할해서 이동하는 객체와 인접한 섹터에 있는 객체만 검색하도록 한다

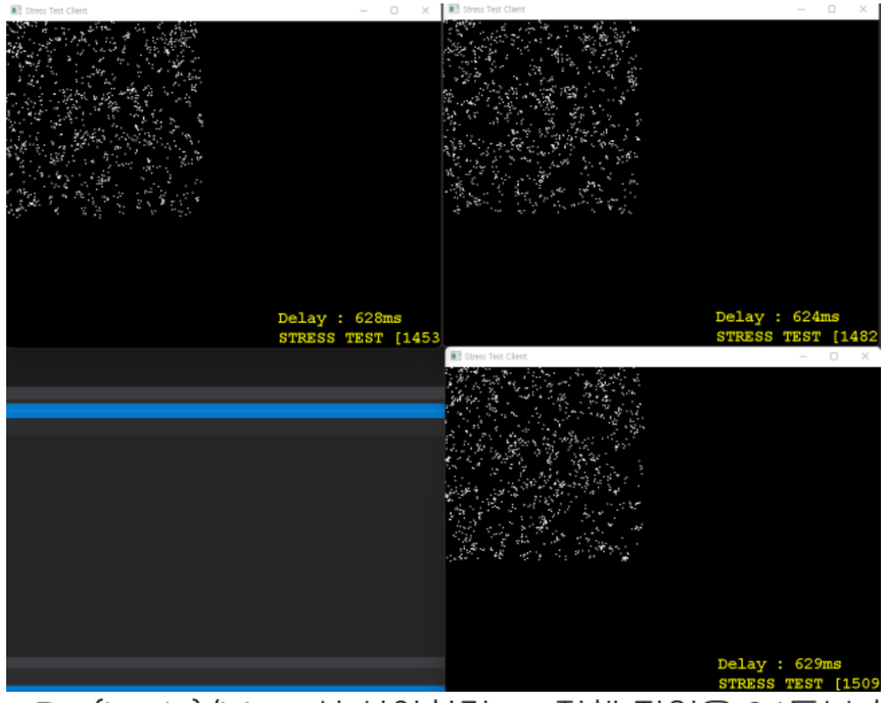
하드웨어 성능
- CPU : AMD RYZEN 7 5800H(8core)
- RAM : 16GB

●섹터 분할 전(동접 2500)



Put(Login)/Move시 시야처리
-> 모든 플레이어를 상대로 시야처리
((접속해 있는 플레이어 + Npc의 갯수) 번 탐색)

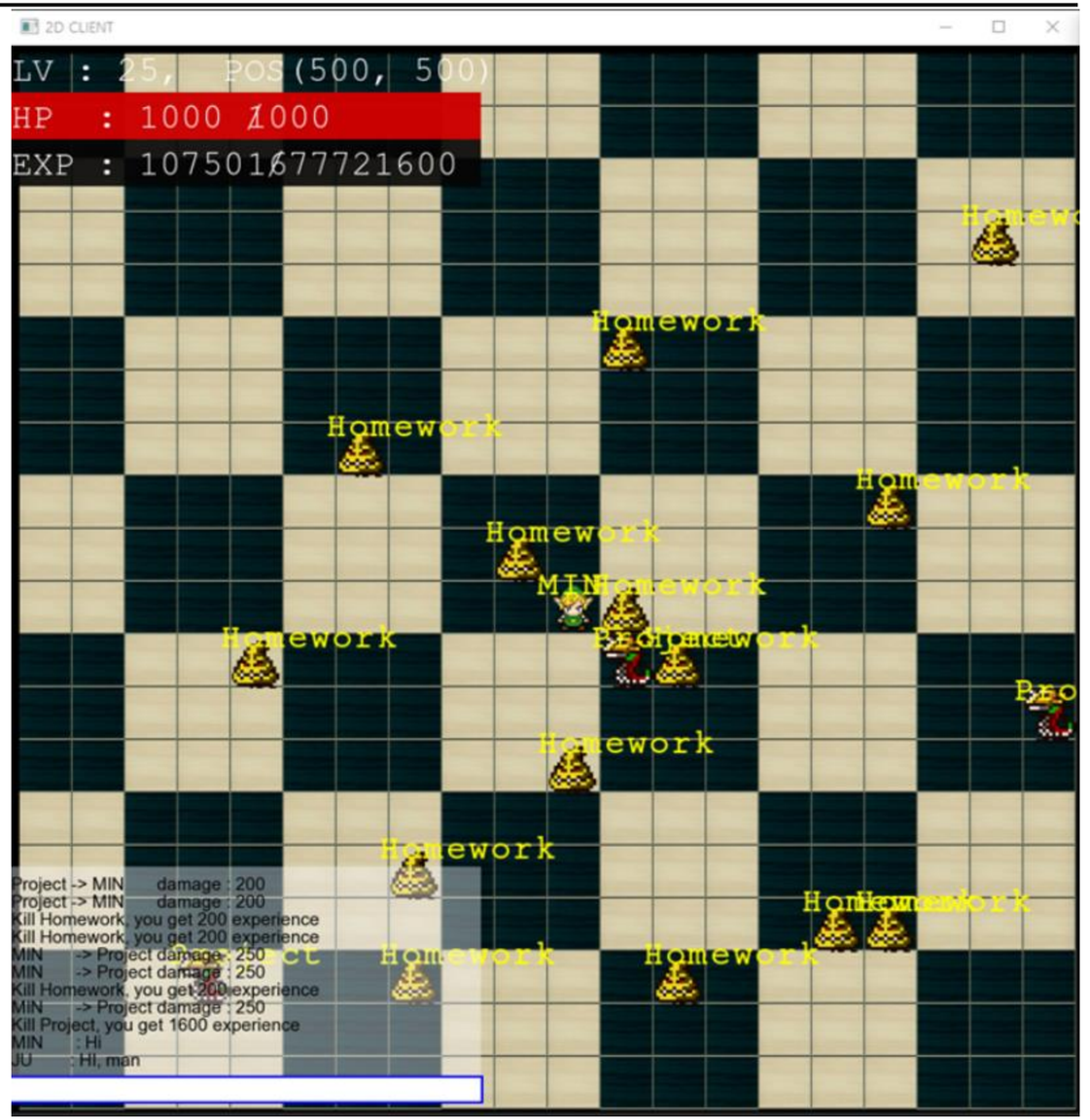
●섹터 분할 후(동접 4200)



- Put(Login)/Move시 시야처리 => 전체 지역을 64등분 하여 자신이 해당하는 섹터와 그 주위의 8개의 섹터에 있는 플레이어에 한해 시야처리
(플레이어가 골고루 퍼져 있다는 가정하에 ((9/64)*섹터분할 전 탐색 수) 만큼 탐색)
- 추가적으로 Move와 Look의 send를 합쳐보내줌으로써 Network I/O를 줄여 SystemCall를 덜 호출하도록 변경

게임 서버 프로그래밍

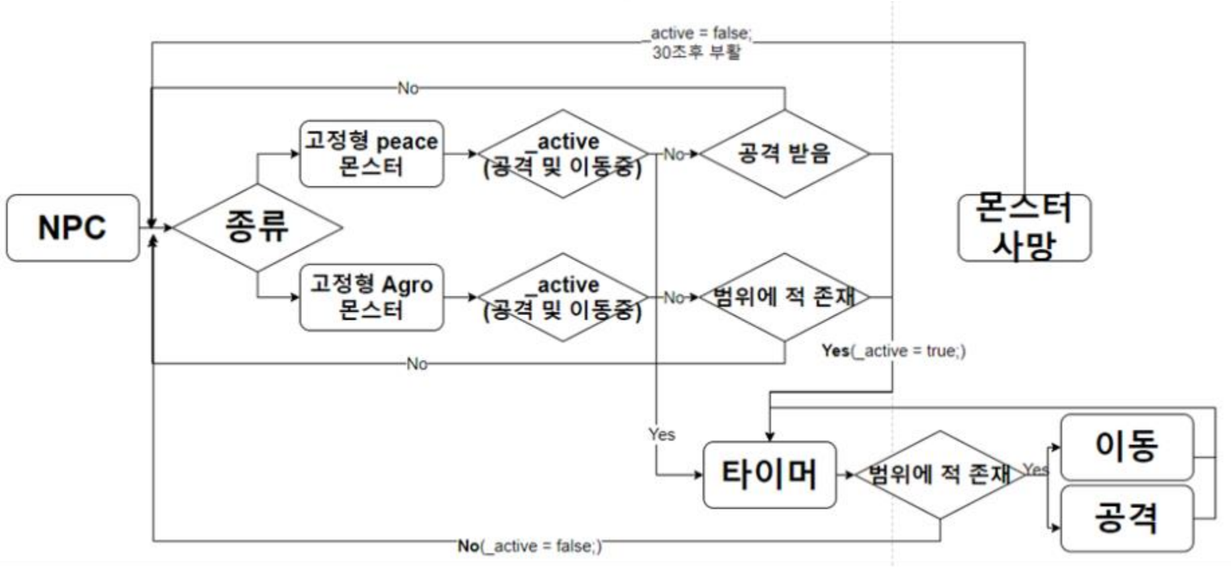
게임 장르	2D MMORPG
작업 기간	2021.11~2021.12
작업 인원	1명(개인 프로젝트)
구현내용	기술 - 멀티쓰레드 IOCP이용 - DB(My SQL, ODBC 사용) - Lua Script 이용 컨텐츠 - 고정형 몬스터(때려야지 따라오는 몬스터) - Agro 몬스터(범위에 들어오면 따라오는 몬스터) - 기본 공격과 스킬 2개 - 채팅 기능
깃허브	https://github.com/Asias-ara/2021-2GameServer-TermProject



NPC

02

Game Flow Chart (Npc)



Npc 자료구조

```
class Npc
{
protected:
	char_name[MAX_NAME_SIZE];
	int_id;
	int_hp;
	int_maxhp;
	short_lv;
	int_exp;
	short_x, _y;
	TRIBE_tribe;
	STATE_state;
	atomic_bool_active;// NPC가 가만히 안있고 움직일때

public:
	mutexstate_lock;
	mutexlua_lock;
	lua_State* L;
```

- Name, id, hp, maxhp, level, exp(사용하지 않음), x좌표, y좌표, tribe(객체의 종류), state(살아있음, 죽음을 나타냄) 기본적으로 게임을 하기 위한 변수들
- _active의 변수의 경우 타겟이 있을 경우 true, 타겟이 없거나 사라진 경우(시야에 벗어남, 타겟이 죽음, 타겟이 로그아웃하는 경우) false
- _active가 false가 되면 자기자리가 아닌 경우 자기자리로 되돌아가고 공격을 멈춤

Npc 설계시 유의했던 점

- 루아 스크립트를 사용하기 위한 가상머신 변수가 존재
- 공유자원의 데이터 레이스를 줄이기 위해 **state_lock과 lua_lock이 존재, 나머지 변수에 대해서는 오류보다 성능이 중요하기 때문에 lock을 사용 안함**
- 한 객체의 LUA 스크립트가 **멀티쓰레드에 의해 동시에 호출되는 경우 에러가(스택이 뒤섞임) 발생**하므로 lua_lock를 사용해서 겹침을 방지

DataBase

02

- 플레이어의 정보를 DataBase에 저장하였습니다.
- 로그인을 할 때, 새로운 사용자를 등록할 때, 사용자의 캐릭터 정보를 저장해야 할 때 DB를 엑세스 한다

● Stored Procedure 사용

- DB 성능 향상을 위해 Stored Procedure를 사용했습니다.
- 플레이어 검색

```
ALTER PROCEDURE [dbo].[search_player]
        @param int
AS
BEGIN
        SET NOCOUNT ON;
        SELECT p_id, p_name, p_x, p_y, p_hp, p_lv, p_exp, p_maxhp
        FROM player_data WHERE p_id = @param
END
```

- 플레이어 위치 저장

```
ALTER PROCEDURE [dbo].[save_player_info]
            @id_param int,
            @x_param int,
            @y_param int,
            @hp_param int,
            @lv_param int,
            @exp_param int,
            @maxhp_param int
AS
BEGIN
            SET NOCOUNT ON;
            UPDATE player_data SET p_x = @x_param, p_y = @y_param, p_hp = @hp_param, p_lv = @lv_param,
p_exp = @exp_param, p_maxhp = @maxhp_param WHERE p_id = @id_param
END
```

(database.cpp과 DataBase 저장 프로시져 내 SQL)

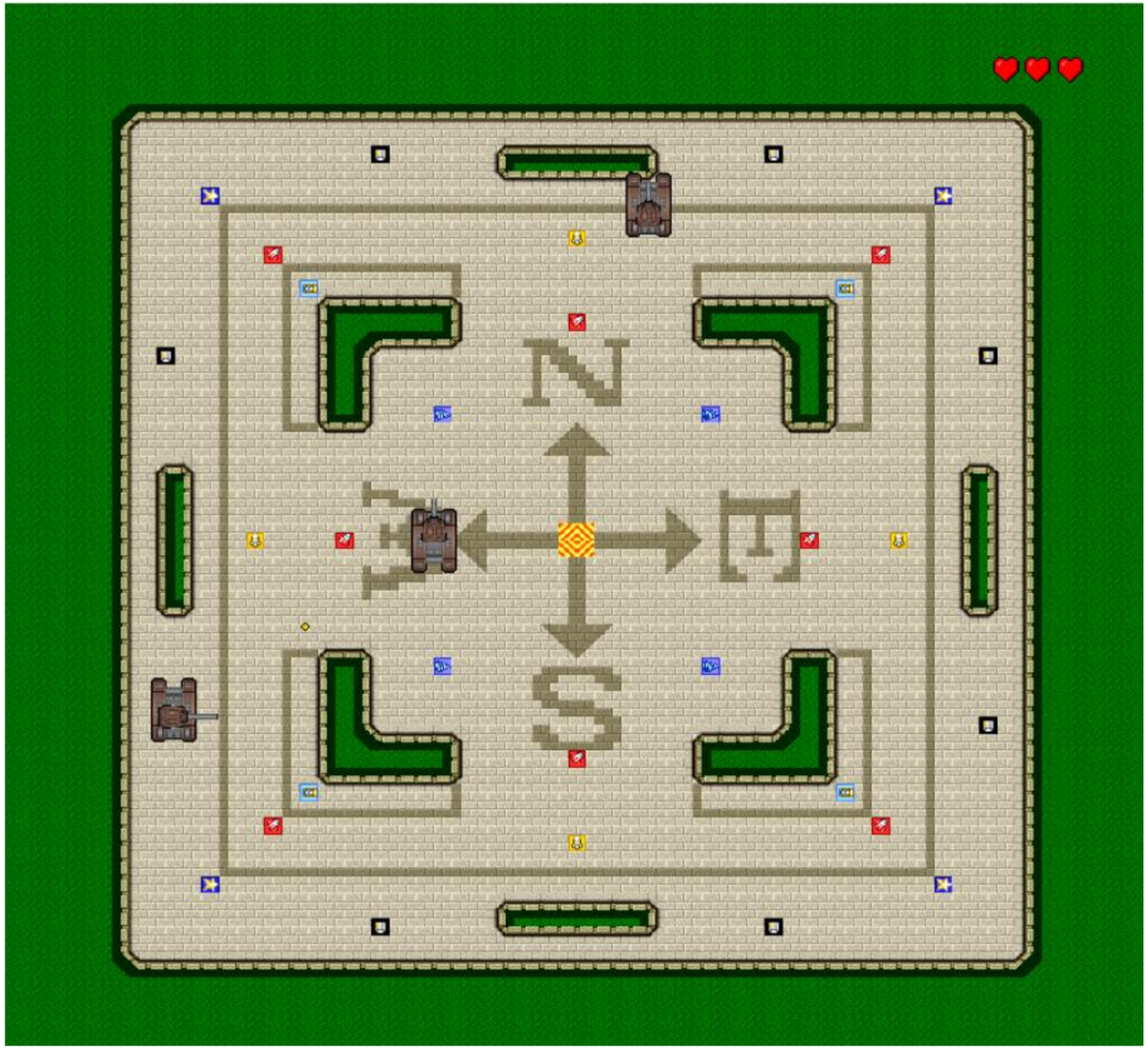
DB 접근하는 경우

- 로그인 : 플레이어 검색
- 로그아웃 : 플레이어 정보 저장

저장 프로시저(Stored Procedure)를 사용하여 SQL문에 접근하고 DB를 관리합니다.

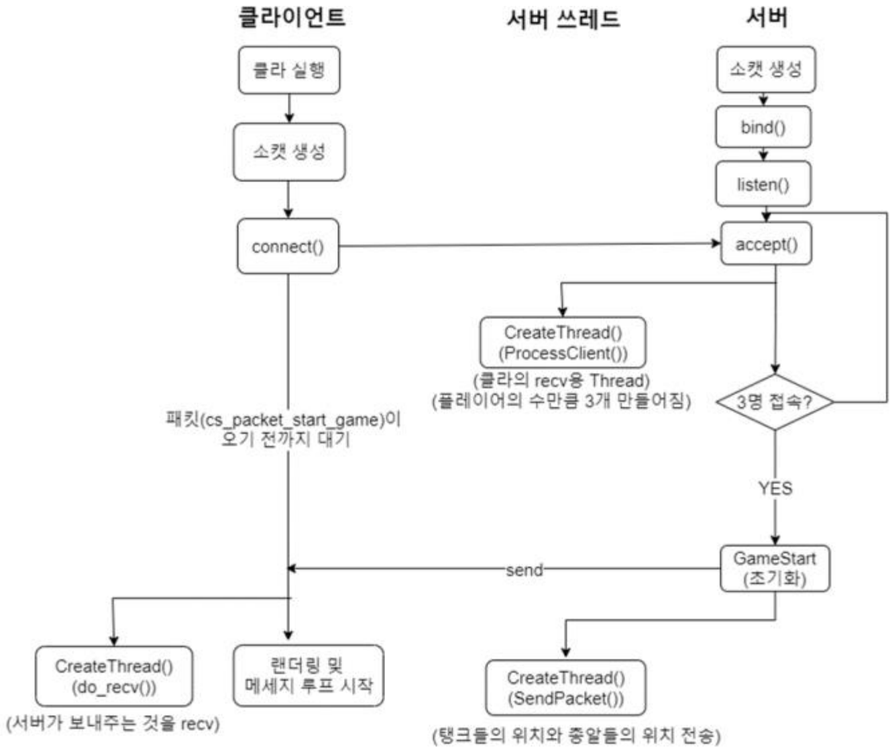
네트워크 게임 프로그래밍

게임 장르	3인 2D 탱크 슈팅 게임
작업 기간	2021.11~2021.12
작업 인원	3명 - 클라이언트 1명 - 서버 2명
구현내용 (역할)	서버 프로그래머 - 게임 로그인 처리 - 서버 스레드간 동기화 - 총알 충돌처리
깃허브	https://github.com/Asias-ara/NGP_TermProject



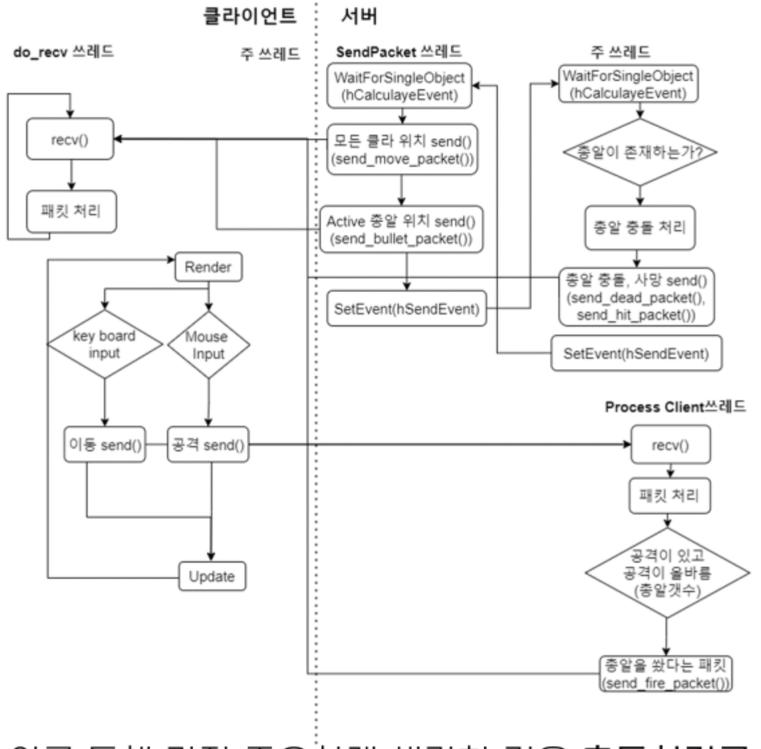
Game Flow Chart

Login Flow Chart



- 클라이언트가 접속시 각각에 대응하는 Thread(ProcessClient())를 만들며 이 쓰레드는 클라이언트가 보내는 패킷을 받는 역할을 한다
- 세명이 접속시 패킷을 보내는 Thread(SendPacket())를 하나 더 만들어 준다
- 마지막에는 총 5개의 쓰레드(주 쓰레드 1개, ProcessClient 쓰레드 3개, SendPacket 쓰레드 1개)가 생긴다

Game Flow Chart



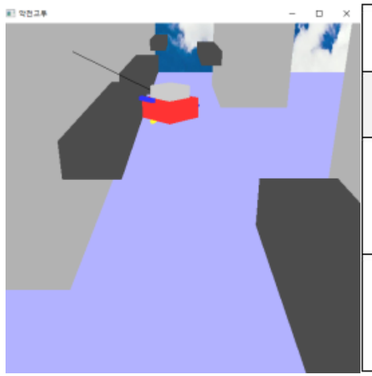
- 팀원과 논의를 통해 가장 중요하게 생각한 것은 **충돌처리를 하는 주 쓰레드**이며 send는 주 쓰레드 한 후 처리하는게 옳다고 생각하여 SendPacket쓰레드와 주쓰레드를 **Event방식으로 공유자원에 대한 동기화**를 하였다.

- 지금 보니 굳이 WaitForSingleObject를 사용할 필요는 없어 보이고, 멀티쓰레드 대신 Overlapped IO로 처리하는 것이 적절해 보인다.

Client Projects

04

게임 장르	공중 화장실 찾는 프로그램 (프로그램명 : 똥터)
작업 기간	2021.04~2021.06
작업 인원	클라이언트 2명
개발 환경	Python
구현 내용 (개인 역할)	UI, 텔레그램 연동, Gmail 연동, Read XML
깃허브	https://github.com/boy0501/ScriptLanguage
비디오 링크	https://www.youtube.com/watch?v=9xJmV3VbE9A



게임 장르	레이싱 게임(게임명 : 악전고투)
작업 기간	2020.11~2020.12
작업 인원	2명 (클라이언트 2명)
개발 환경	- OpenGL - C++ 11

게임 장르	핵 앤 슬래시
작업 기간	2021.05~2021.06
작업 인원	1명
개발 환경	Unreal Engine 4 (BluePrint)

Client Projects

전체적으로 보기가 어지러움

프로젝트	UI 개선 프로젝트
작업 기간	2021.04~2021.05
작업 인원	3명(기획자 1명, 그래픽 1명, 클라이언트 1명) (본인 역할 : 클라이언트)
개발 환경	2D Unity

프로젝트 목표 및 진행과정

- 모바일 게임의 작은 화면으로 인한 피로감을 UI를 최대한 간단하게 만들어 피로감을 덜어내도록 한다.
- 카테고리를 만든 후 의미가 비슷한 것은 한 곳으로 묶고 기능이 중복되는 것은 합친다.(Ex. 스킬 + 하이퍼 + V매트릭스 => 스킬)
- 버튼을 보기 좋게 카테고리별 한 줄 씩 나타낸다

이외의 클라이언트 프로젝트 영상

유튜브 링크 (1분 40초)

https://www.youtube.com/watch?v=fxH1n6ZDOX8

영상 내 프로젝트 설명(기간, 인원, 개발 환경)

- 오목 AI : 2020.04 ~2020.05, 개인 프로젝트, C언어
- OpenGL : 2020.03 ~ 202.05, 개인 프로젝트, OpenGL/C++11
- Unreal4 : 2020.04, 개인프로젝트, Unreal4(Blueprint)