

한국산업기술대학교 게임공학과

포트폴리오

서버 부분

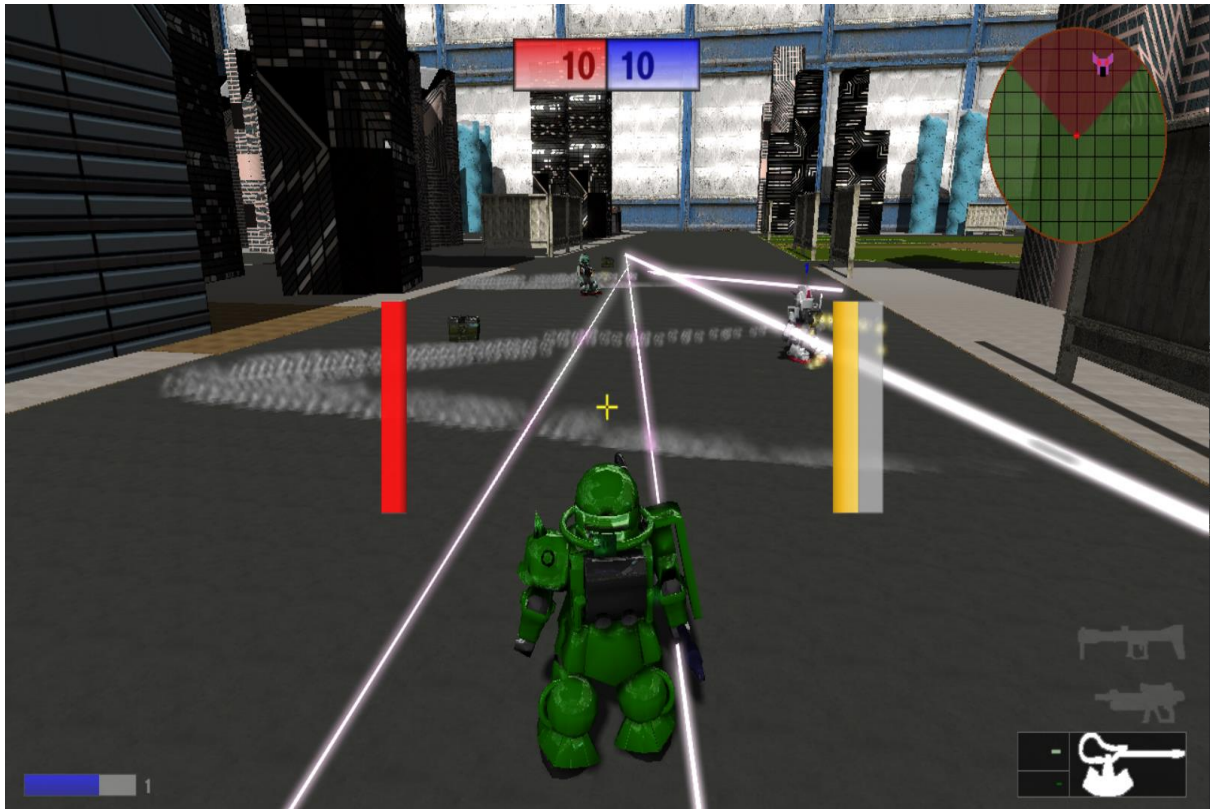
김대훈

2019-9-22

목차

1. 알약 전사.....	2
기본 설명	2
세부 설명	2
제작하면서 어려웠던 점.....	6
2. Get Eat Fish	7
기본 설명	7

1. 알약 전사



기본 설명

프로젝트 소스: <https://github.com/GogleEarth/Portfolio/tree/master/Pills-Fighter>

사용 언어: C++11

개발 환경: Visual Studio 2017 64bit

장르: 3 인칭 로봇 대전 액션

제작 인원: 3 인 (서버&아트 [본인], 클라이언트, 기획 & 클라이언트)

제작 기간: 약 13 개월

목표: 윈도우 소켓, IOCP 를 활용하여 3 인칭 대전 액션 게임서버 제작

세부 설명

1. 게임 구조:

알약전사는 SF 배경의 3 인칭 로봇 전투 액션 게임으로 최대 4:4 로 진행할 수

있습니다.

구조적으로 최대 1000 명의 유저를 받을 수 있으며, 최대 10 개의 방을 만들 수 있습니다.

유저는 로비에서 닉네임을 바꾸거나 방을 만들고 참여할 수 있으며, 방을 만든 플레이어가 게임시작 버튼을 누르면 모든 유저의 로딩이 끝나는 것을 기다린 뒤 게임이 시작됩니다.

플레이 가능한 캐릭터는 3 종류가 있으며 모든 캐릭터는 근접 무기 1 종, 원거리 무기 2 종을 가지고 있습니다.

상대편 캐릭터를 처치하고 상대편 스코어를 0 으로 만드는 것이 승리 목적이며, 게임 중에는 선택된 맵에 따라서 시간의 경과에 따라 이벤트가 발생합니다.

2. 서버 상세설명:

IOCP 사용 이유:

이전 서버는 방의 update 를 하나의 스레드가 담당해서 처리했기 때문에 오브젝트의 수가 많아질수록 해당 스레드가 병목이여서 제대로 된 성능을 낼 수가 없었습니다. 이를 해소하고 성능 향상을 위해 IOCP 를 사용하였습니다.

A. 스레드 종류와 수

가) **Accept_thread(1 개):** 유저의 접속을 받아들이는 스레드로 유저가 접속을 하면 현재 존재하는 방의 정보를 유저에게 알려준 후 소켓을 IOCP 에 연결해주고, 수신대기상태로 만듭니다.

나) **Worker_threads(4 개(cpu 의 코어 수)):** 서버가 해야할 대부분의 작업을 하는 스레드로 유저와 통신을 주고받고 패킷의 해석 및 처리, Timer_thread 로부터 받은 각종 이벤트를 처리합니다.

다) **Timer_thread(1 개):** 시간에 맞춰 발생할 이벤트를

PostQueuedCompletionStatus() 함수를 통해 Worker_thread 에 알려주는 스레드로 오브젝트의 update, 방의 update, 플레이어 리스폰 등의 이벤트를 알려줍니다. 프레임 워크가 가지고 있는 우선순위 큐에 인자 값(오브젝트 index, 방 번호, 이벤트 시간, 발생할 이벤트)을 가지는 Evnet 구조체를 넣어 발생 시간이 되면, IOCP 에 이벤트를 추가하고, Worker_thread 에서 해당 이벤트를 처리합니다.

- B. **패킷:** 패킷 구조체의 처음 2BYTE 는 1BYTE 는 패킷의 사이즈, 다음 1BYTE 는 패킷의 타입이 들어가며 뒤는 패킷에 따라 필요한 정보가 추가로 붙는 형태입니다.

```
typedef struct PKT_PLAYER_INFO
{
    BYTE      PktSize;
    BYTE      PktId;
    BYTE      ID;
    XMFLOAT4X4 WorldMatrix;
    WEAPON_TYPE Player_Weapon;
    BOOL      isChangeWeapon;
    ANIMATION_TYPE Player_Up_Animation;
    BOOL      isUpChangeAnimation;
    float      UpAnimationPosition;
    ANIMATION_TYPE Player_Down_Animation;
    BOOL      isDownChangeAnimation;
    float      DownAnimationPosition;
    int        State;
}PKT_PLAYER_INFO;

typedef struct PKT_PLAYER_LIFE
{
    BYTE      PktSize;
    BYTE      PktId;
    BYTE      ID;
    DWORD     HP;
    DWORD     AMMO;
}PKT_PLAYER_LIFE;

typedef struct PKT_PICK_ITEM
{
    BYTE      PktSize;
    BYTE      PktId;
    BYTE      ID;
    BYTE      Item_type;
    DWORD     HP;
    DWORD     AMMO;
}PKT_PLAYER_PICK_AMMO;

typedef struct PKT_CREATE_OBJECT
{
    BYTE      PktSize;
    BYTE      PktId;
    OBJECT_TYPE Object_Type;
    XMFLOAT4X4 WorldMatrix;
    int        Object_Index;
    ROBOT_TYPE Robot_Type;
}PKT_CREATE_OBJECT;

typedef struct PKT_DELETE_OBJECT
{
    BYTE      PktSize;
    BYTE      PktId;
    int        Object_Index;
}PKT_DELETE_OBJECT;
```

- C. **Overlapped 구조체:** WSAOVERLAPPED 구조체를 확장하여 사용한 구조체로 Worker_thread 에서 GetQueuedCompletionStatus()함수를 이용해 send 와 recv 만이 아닌 다양한 이벤트 처리를 위해서는 확장된 WSAOVERLAPPED 구조체가 필요하였습니다.

```
struct Overlapped
{
    WSAOVERLAPPED    overlapped_;
    WSABUF            wsa_buffer_;
    char              packet_buffer_[MAX_BUFFER];
    EVENT_TYPE        event_type_;
    float             elapsed_time_;
    int               room_num_;
};
```

- 가) event_type_ – 이벤트의 타입
- 나) elapsed_time_ – 이벤트는 대부분 1 프레임(16ms)단위로 실행되는데 그보다 빨리 실행되었을 경우 16ms 로 보정해 주기위한 변수
- 다) room_num_ – 이벤트가 실행될 방 번호
- 라) wsa_buffer_ – WSARecv(), WSASend()에 인자로 들어갈 WSABUF 구조체
- 마) packet_buffer_ – wsa_buffer 에 buffer 가 들어갈 메모리 공간

제작하면서 어려웠던 점.

1. 충돌체크의 어려움

- 오브젝트의 속도가 너무 빠르면 플레이어의 캐릭터를 통과하여 충돌검사가 안되는 터널현상이 있었습니다. 이를 해결하기 위해 DirectX의 BoundingBox의 Intersects 함수를 활용하여 BoundingBox와 벡터간의 충돌검사를 통해 오브젝트의 이전위치에서 플레이어의 BoundingBox까지의 거리와 오브젝트의 이전위치에서 현재위치까지의 거리를 비교하여 터널현상을 해결할 수 있었습니다.

```
bool Scene::check_collision_player(int object)
{
    // 오브젝트의 이전위치
    FXMVECTOR origin = XMVector3(&Objects_[object].get_prev_position());
    // 오브젝트의 진행방향
    FXMVECTOR direction = XMVector3(&Objects_[object].get_look());
    float distance;

    for (int i = 0; i < MAX_CLIENT; ++i)
    {
        if (Objects_[i].get_play())
        {
            if (!Objects_[i].get_is_die())
            {
                for (auto playeraabb : Objects_[i].get_aabbs())
                {
                    // 오브젝트의 이전위치에서 진행방향으로 광선을 쏘아 플레이어의 AABB와 충돌 검사를 한다.
                    // 충돌검사 결과가 true이면 distance에 광선이 충돌된 위치까지의 거리가 반환된다.
                    if (playeraabb.Intersects(origin, direction, distance))
                    {
                        // 오브젝트의 이전위치와 현재위치사이의 거리(len)를 구한다.
                        XMVECTOR length = Vector3::Subtract(Objects_[object].get_position(), Objects_[object].get_prev_position());
                        float len = Vector3::Length(length);

                        // len이 distance보다 크다면 오브젝트의 이전위치와 현재위치 사이에 플레이어가 있었던것이므로 충돌처리를 한다.
                        // distance가 len보다 크다면 오브젝트는 아직 플레이어와 충돌하지 않은 상태이다.
                        if (distance <= len)
                        {
                            if (i != Objects_[object].get_owner_id())
```

2. 디버그의 어려움

- 클라이언트와 패킷을 주고받으면서 버그가 발생했을 때 클라이언트에서 패킷을 잘못 읽어서 발생하는 문제인지 서버가 잘못 보내서 발생한 문제인지 찾는 것이 힘들었습니다. 대부분 서버나 클라이언트에서 패킷을 보낼 때 다른 값을 넣어 보내는 경우가 많았고, 이후에는 패킷을 주고받을 때 먼저 제대로 된 값이 오는지 도스창에 출력을 해보고 문제가 없으면 기능을 구현하였습니다.

2. Get Eat Fish



기본 설명

프로젝트 소스: https://github.com/GoogleEarth/Portfolio/tree/master/Get_Eat_Fish

사용 언어: Python3

개발 환경: PyCharm community 2017, Pico2d

장르: 아케이드

제작 인원: 1 인

제작 기간: 약 3 개월

게임내용: 최대한 오래 살아남는 것이 목적인 게임으로 시작 시 임의로 힘과 행운 스탯이 정해집니다. 플레이어는 상단의 주황색 게이지가 없어지기 전에 낚시를 하여 물고기를 먹어야 하고, 상단의 주황색 게이지는 시간이 지날수록 더욱 빠르게 감소합니다. 낚시를 할 때에는 하단의 제한시간 안에 빨간색 선이 주황색 칸 안에 위치하게 하면 낚시에 성공합니다. 물고기를 낚을 때 마다 임의로 정해진 물고기의

레벨만큼 플레이어의 힘이 오르고 플레이어의 힘은 낚시를 하는데 도움을 주는 역할을 합니다.