**Graduate Project**

**For Game Software Major**

**Report**

**2020.5.20**

**JunYoung Lee**

목차

**1. 서론1**

1) 장르2

2) 플랫폼2

3) 구현 언어2

4) 상용 엔진2

5) 라이브러리2

**2. 구성도4**

**3. 구현 내용4**

1) IOCP 서버4

2) 패킷**4**

(1) 마샬링**4**

(2) 암·복호화**4**

3) 분산서버**4**

4) 홈페이지**4**

(1) 프론트 엔드**4**

(2) 백 엔드**4**

5) JSON**4**

(1) 서버 정보**4**

(2) 월드 맵**4**

6) 공간 분할 기법**4**

**4. 결과4**

**1. 서론**

**1) 장르**

오픈월드 MMORPG

**2) 플랫폼**

Windows PC

**3) 구현 언어**

**Client:** C#

**Server:** C++

**Authentication:** JS

**4) 상용 엔진**

Unity3D Engine

**5) 라이브러리**

**Client:** NewtonSoft.Json

**Server:** rapidjson, libcurl

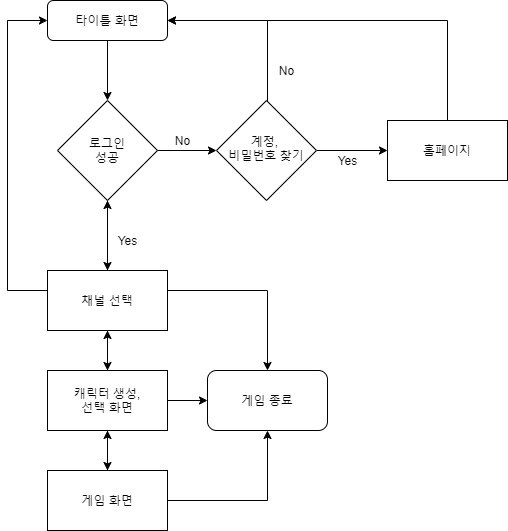
**Authentication Back-end:** express, mongoose, nodemailer, dotenv, bcryptjs

**Authentication Front-end:** react, material-ui, axios, immutable, react-router-dom, redux, styled-components

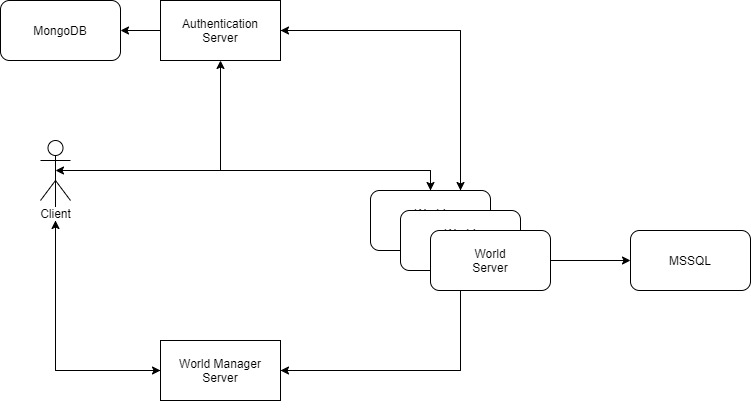
**2. 구성도**

**(1) Flow chart**

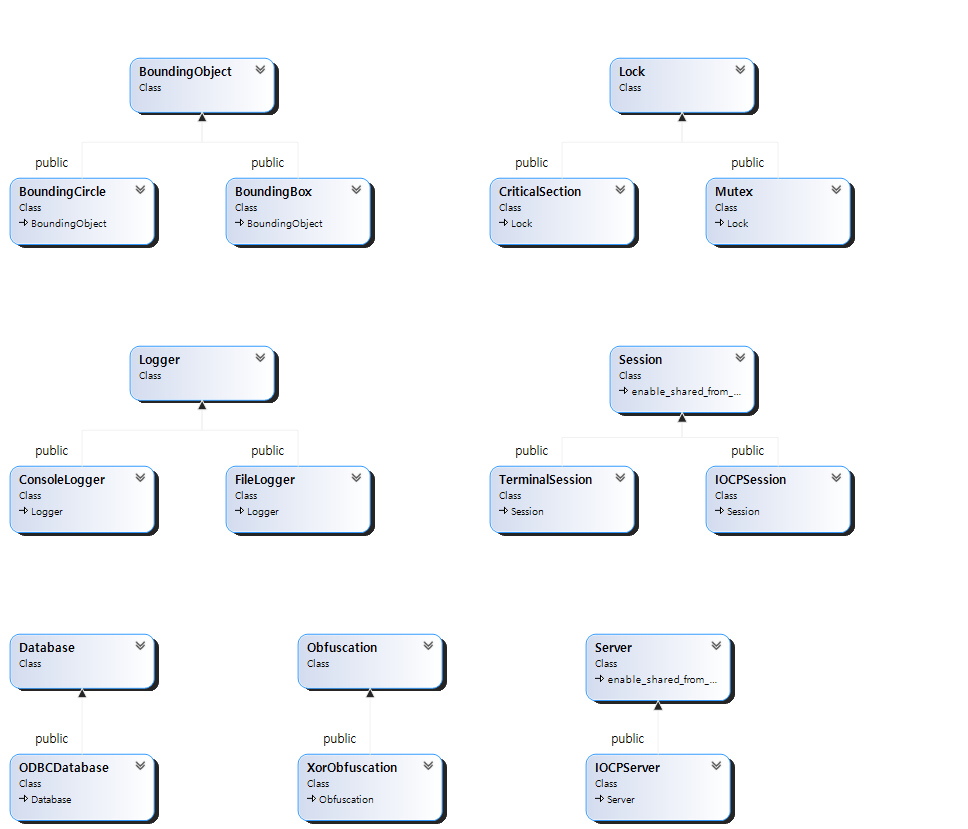
**# Client**

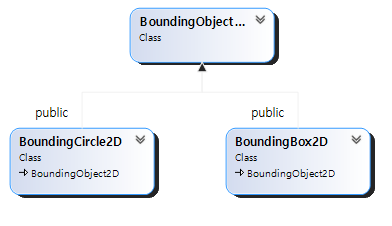
****

**# Server**

****

**(2) 클래스 다이어그램**

****

****

****

**3. 구현 내용**

**1) IOCP Server**

게임 서버는 Windows 환경에서 개발했다. 과거에 비해 최근 업계에서도 linux 보다는 Windows 서버를 선호한다고 들었다. 그 이유는 linux보다 Windows의 경우 지원하는 기능, 개발의 편의성 때문이었다. 이러한 이유도 있었고, 클라이언트와 병행하여 작업해야 했기에 많은 부분을 신경 쓸 수 없어서 기술적인 부분을 좀 더 집중할 수 있도록 클라이언트와 서버 모두 동일한 플랫폼 환경에서 제작하게 됐다. 단순히 Overlapped IO, 혹은 select 등을 사용한 다양한 서버 모델이 있지만 제작하고자 하는 장르가 MMORPG이므로 많은 커넥션을 받을 수 있어야 하고, 보다 효율적으로 쓰레드를 관리할 수 있기 때문에 IOCP 모델을 사용하여 구현했다.

**2) 패킷**

**(1) 마샬링**

클라이언트는 C#이고 서버는 C++로 구현했기 때문에 다른 언어 간의 통신을 위해 마샬링이 필요했다. 클라이언트를 기준으로 서버에서 보낼 패킷을 바이트로 변환하여 스트림 형태로 보낸다.

**(2) 암·복호화**

패킷은 스누핑의 가능성이 있기 때문에 클라이언트에게 보내기 전에 암호화하는 작업을 거쳐야 한다. AES, DES, RSA 등 다양한 암호화 알고리즘이 있지만 패킷을 암호화와 복호화 하는 것에 적지않은 비용이 들며 이는 서버 부하에도 직접적인 영향을 미칠 수 있다. 또한, 서버 측에서 암호화 한다고 해도 클라이언트가 패킷을 받았을 때 복호화 과정을 거쳐야 하기에 전문적인 해커라면 클라이언트를 확인하여 어떤 알고리즘을 썼는지 알 수 있기에 패킷의 암호화와 복호화에 그리 큰 비용을 쓸 필요는 없다고 판단된다. 따라서, 단순히 XOR을 사용한 암호화 기법을 채택했다. XOR 연산만으로도 충분히 스누핑을 방지 하고 데이터의 유효성은 서버에서 체크하는 것이 옳다고 생각된다.

**3) 분산서버**

분산서버는 말 그대로 서버의 기능을 분리하여 별도의 서버로 두어 병목 현상을 줄이고자 함에 있다. 다만, 서버와 서버 간의 통신이 필요하기 때문에 실제 연산 시간에 비해 통신 시간이 더 오래 걸린다면 불필요한 분산이 되기 때문에 이를 잘 고려해서 구현해야 한다. 처음 서버 구조를 설계했을 때는 게임 서버 단일 구조로 구성을 했었다. 그러다 보니 모든 기능에 대한 처리를 한 곳에서 구현해야 했기에 코드가 읽기 어려워지는 것은 물론 하나의 서버에서 많은 패킷을 처리해야 했기에 비효율적이었다. 그래서 분산서버를 고민했고 처음에는 통신 시간을 고려하지 않고 기능적으로 모두 분리하여 설계를 했었다. 예를 들어, 유저들의 채팅을 담당하는 채팅서버, 로그인과 회원가입 등 유저의 인증을 관리하는 인증 서버, 게임 로직을 관리하는 월드 서버, 아이템의 거래를 위한 경매장 서버, 그리고 각 서버들의 데이터를 저장할 DB 서버들로 구성했었다. 물론 테스트 환경은 로컬이었기에 속도는 이상없이 빨랐고 제대로 된 성능 테스트를 해볼 수 없었다. 적절한 서버의 구조를 설계하고자 했으나 MMORPG 서버의 설계 구조에 대해 완벽하게 설명되어 있는 자료는 적어 참고하기에 어려움이 있었다. 그러다가 서버 구조에 대한 컨퍼런스 자료를 보게 됐다. 테라의 경우에는 중계 서버가 있고 다른 월드 서버에게 적절히 패킷을 다시 보내는 형식으로 되어 있었고, NC소프트에서는 클라이언트가 게임서버에 직접 접속하는 방식으로 구성되어 있었다. 중계 서버가 있으면 다른 월드 서버를 관리하기에 용이하며 데이터의 수집 등 다양한 이점이 있다. 다만, 결국 클라이언트가 게임 서버에 도달하기 위해서는 중계 서버를 거쳐서 가야하기에 중계서버에 병목현상이 발생할 수도 있을 것 같다. 반면, 직접적으로 클라이언트가 게임 서버에 접속하게 되면 데이터 수집과 월드 서버를 관리하기에 어려움이 있을 듯하다. 그래서 이러한 특징들을 적절히 섞어서 설계했다.

**4) 홈페이지**

계정 인증 서버 역시, IOCP 기반의 서버로 구현하고자 했으나 회원가입, 로그인 계정 찾기, 비밀번호 찾기 정도의 간단한 처리만 필요하고 게임 내에서 직접 회원가입과 같은 일련의 절차를 밟는 것보단 브라우저 상에서 진행하는 것이 유저에게는 익숙할 것으로 판단되어 자바스크립트로 구현했다.

**(1) 프론트 엔드**

홈페이지에서는 이메일로 인증하여 해당 이메일로 인증 코드를 보내 인증을 받아야만 회원가입을 할 수 있고, 회원가입 후 만약 잊어버렸다면 가입 시 사용한 이메일로 아이디와 비밀번호를 찾을 수 있다. 바닐라 자바스크립트만으로도 충분히 만들 수 있지만 React.js를 활용하여 좀 더 View에 집중하여 개발을 하고자 했다.

**(2) 백 엔드**

백 엔드로는 Node.js와 Express.js를 활용하여 구현했다. 기본적으로 IOCP 서버와 다르게 싱글 스레드를 기반으로 동작하는 비동기 서버이기 때문에 구현이 그리 어렵지 않고 빠르게 구현할 수 있었다. 다만, 자바스크립트 개발자가 직접 말했던 콜백지옥과 같은 코드가 만들어지는 상황이 있어 가독성이 떨어지는 코드가 만들어졌었다. 이러한 문제점은 promise 패턴이나 async await으로 해결할 수 있었는데 React.js의 경우에는 자바스크립트 최신 문법인 es6를 지원하여 문제 없었으나 기본적으로 Node.js와 Express.js 에서는 es6 문법을 지원하지 않아 이를 해결하는데 상당히 많은 시간을 투자했었다. 그래서 결국 찾은 해결방안은 babel이라는 transfiler로 es6 코드에서 es5코드로 변환하는 과정을 거쳐 es6의 문법을 사용할 수 있었다.

데이터베이스는 NoSQL인 mongoDB를 사용했다. mongoDB에서는 스키마가 별도로 없고 컬렉션이라는 것으로 데이터를 관리한다. 자바스크립트에서 사용하는 객체와 동일하게 사용할 수 있어서 구현하기에 용이하여 mongo DB를 선택했다.

**5) JSON**

**과거에는 XML이라는 것을 이용하여 데이터를 관리 했으나, 최근에는 JSON을 활용해 키+값으로 이루어진 테이블 형식의 객체로 데이터를 관리하는 추세이다. 그래서, 데이터를 관리하기위해 JSON을 선택했다.**

**(1) 서버 정보**

**서버에 필요한 정보들을 미리 저장하여 코드의 가독성을 높이고 상황에 맞게 서버의 설정을 변경하고자 했다. 서버의 이름, 서버에서 사용할 쓰레드의 개수, 그리고 분산 서버를 위한 다른 서버의 IP, Port 등의 정보를 넣었다.**

**(2) 월드맵 빌드**

**광활한 월드 맵 상에서 길 찾기 기능이 없다면, 유저는 상당한 불편함을 해소할 것이다. 따라서, 길 찾기 기능을 제공하고자 했고 이를 위해서는 월드 맵을 갈 수 있는 지역과 갈 수 없는 지역으로 구분하여 길 찾기 알고리즘을 적용해야 했다. 그러기 위해서는 월드 맵 정보를 계산하여 가지고 있을 필요가 있었다. 현재는 300\*300 정도의 작은 맵이지만 좀 더 규모가 확장된다면 월드맵의 정보를 계산하기에는 너무 많은 시간이 걸릴 것이다. 그래서 미리 그 정보들을 계산하여 저장하고 로딩 화면에서 JSON으로 데이터를 불러와 빌드 시간을 단축하고자 했다.**

**6) 공간 분할 기법**

오픈월드 MMORPG 특성상 넓은 지역에서 많은 유저가 활동하기 때문에 이를 구분없이 한 유저의 행동을 다른 모든 유저에게 알리고자 한다면 서버는 이를 견디지 못하고 다운될 것이다. 따라서, 공간을 나눠 유저에게 필요한 정보들만 받도록 해야 한다. 처음 고려했던 방법은 쿼드 트리를 활용하여 일정 크기의 블록 단위로 나눠 해당 블록에 있는 유저들만 서로 상호작용이 가능하도록 만들고자 했었다. 하지만, 쿼드 트리로 하게 되면 이진 탐색을 해야 하며 재귀 호출의 구조를 가지기 때문에 과연 성능상 이점이 있을지 의문이 생겼다. 물론 비트연산을 통한 직접 접근 쿼드트리 방식도 있으나, 경계선이 모호하며 사각형의 형태로 경계를 구분하기에 어색함이 발생할 수도 있을 것 같았다. 그래서, 플레이어의 시야를 기준으로 다른 플레이어들이 자신의 범위 안에 존재하는지를 판단하여 정보를 받을지 결정하고자 했다. 다만, 아직 해결하지 못한 문제점은 정확히 경계선상에서 빈번하게 움직일 때가 문제가 된다. 이는 범위 안에 들어온 시간을 체크하여 유저가 일정 시간이 지났을 때 삭제하는 방식으로 구현하면 될 것 같다.

몬스터 역시 시야를 기준으로 관리한다. 다만, 몬스터의 상위 계층에는 몬스터 생성 구역이라는 오브젝트가 있어 이를 기준으로 설정했다. 몬스터가 생성되는 구역에는 유저를 인식하는 최대 범위가 있고, 이보다 작은 범위로 몬스터의 활동 반경을 제한하여 몬스터가 유저 인식 범위를 벗어났을 때를 방지하고자 했다.

유저가 접속했을 때, 이동할 때 등 특정한 이벤트가 발생했을 때만 유저의 시야를 업데이트 하여 서버의 부하를 줄이고자 했고, 몬스터 생성 구역을 기준으로 플레이어가 이동하여 시야에 들어오면 해당 플레이어를 인식 목록에 추가하고 몬스터에게 이를 알린다. 몬스터가 플레이어를 쫓아가는 방식은 게임들마다 모두 다르다. 일반적으로는 길 찾기를 하여 직접 플레이어를 쫓아가는 방식을 채택하거나 직선거리로 쫓아가고 가는 길에 장애물이 있으면 뚫고 지나가는 식으로 하여 단순하게 구현하는 방식도 있다. 전자의 경우 길 찾기를 해야 하기 때문에 길 찾기를 위한 별도의 서버를 분리하는 것이 하나의 방법일 것으로 보이며 후자의 경우에는 자칫하면 게임성을 떨어뜨릴 수 있을 것 같다. 내가 채택한 방법은 플레이어가 이동한 경로를 따라 가는 것이다. 플레이어가 이동한 경로라면 그 지역은 분명 갈수 있는 지역임이 자명하기에 신뢰하고 따라간다는 것이다. 다만, 플레이어가 비정상적인 동작으로 이동을 하였다면 문제가 심각해질 수도 있을 것도 같다.

**4. 결과**

평소 MMORPG 장르를 많이 해봤기에 세세한 부분도 신경 써 구현하고자 했다. 클라이언트와 서버 그리고 로그인 부분 역시 처음부터 구현해야 했기에 어려움이 많았다. 특히, 서버는 이론은 알고 있었지만 구현을 밑바닥부터 시작해야 했고, 클라이언트와 병행해야 했고 알면 알수록 더 많은 기능을 구현해보고자 하나씩 늘리다 보니 규모가 커진 것은 물론 관리하기에 만만치 않았다. 결국, 처음 기획했던 내용을 모두 구현하지 못한 것에 아쉬움이 남는다. 온라인 게임을 할 때 왜 이 부분은 이렇게 구현했을까 싶었던 의문들이 직접 구현해보면서 그럴 수밖에 없었다는 것을 깨닫는 순간도 있었다. UX적으로 불편함이 분명 존재하지만 어느정도 성능과 타협 해야만 하는 상황도 다소 있었다. 웹 프로그래밍에서는 프론트 엔드와 백 엔드를 같이 개발하는 사람을 보고 풀 스택 개발자라고 하는 얘기들이 있다. 사실 웹 프로그래밍 특성상 프론트 엔드와 백 엔드는 물론 DB까지 알고 있어야 개발을 할 수 있기에 원래의 의미가 살짝 변질된 느낌이 있지만 풀 스택 개발자라는 것은 지향점이지 모든 것을 다 잘하기에는 어려움이 있는 것 같다. 이번 프로젝트를 진행하면서 클라이언트, 서버, 웹, DB 모두 개발하면서 비록 어려움이 있었지만, 전반적인 온라인 게임이라는 것의 흐름을 이해하고 스스로 부족한 부분이 무엇인지 알 수 있었다. 조금 아쉬웠던 것은 월드맵은 정적이므로 A\* 알고리즘을 이용해서 미리 최적의 경로를 찾고 이를 데이터로 저장해서 길 찾기 서버로 만들고자 했으나, 최적화하지 않은 A\* 알고리즘으로는 N\*N 행렬일 경우 N\*N \* (N\*N-1) = O(N^4) 정도의 연산이 필요했다. 구현한 게임 경우에는 월드맵을 블록들로 나누었을 때 가로 300개, 세로 300개로 나뉜다. 따라서, 총 8,099,910,000번의 연산이 필요하다. 빌드 툴로 구현해봤으나, 너무 많은 시간이 걸려서 포기했다. 다만, 좀 더 최적화를 해서 구현한다면 O(1)로 길 찾기가 가능해지기 때문에 충분히 다시 도전해볼만 하다고 생각한다.