3계층 구조 Multi-player Online 웹 게임 시스템 구축

팀원 : 201412824 이지훈
201412779 강형석
201412790 김진규

I. 주제

게임은 단순한 유흥 거리를 넘어 하나의 문화로 자리 잡았다. 그러나 일반적으로 어떤 게임을 다수가 함께 즐기기에는 PC와 모바일 간의 플랫폼 호환성, 또한 추가적인 프로그램을 설치해야 하는 까다로움이 따른다. 그에 대한 해결책으로 chrome, Fire Fox와 같은 웹 브라우저상에서 실행되는 HTML5 기반의 웹게임을 구현하며 그 게임을 실시간으로 다수가 함께 즐길 수 있도록 개발한다.

또한 3계층(3-Tire) 구조의 아키텍처를 채용하여 자칫 복잡해질 수 있는 게임의 구조를 보다 간편하게 구성하며 각각의 계층을 팀원들이 적절하게 역할 분담을 하여 효율적인 프로젝트 진행과 협업을 배우는 것을 목표로 하며 Docker Image를 통한 프로그램의 배포를 통해 Docker의 사용법을 배운다.

II. 과제의 범위

본 과제의 목표는 3계층 구조의 실시간 웹 게임을 구축하며 총 3개의 계층을 팀원들이 각자 개발 후 Docker Image로 배포해 통합하는 과정을 가지는 것이다. 3개의 계층은 다음과 같다.

- HTML5 게임을 제공하는 **클라이언트**
- 게임 내부적으로 실시간 통신을 수행할 **게임 서버**
- 게임의 정적 정보들(사용자, 채널 정보 등)을 처리하고 제공하는 **API 서버**

가. 클라이언트

본 과제에서의 클라이언트 역할은 HTML5로 개발된 게임이다. 또한 단판 위주에 다른 플레이어와 경쟁하는 방식의 웹게임들을 io게임 이라고 하는데 이러한 io 게임을 이번 과제의 클라이언트로 채택하였다. 또한 게임의 기술적인 부분은 완벽하게 구현하나 외적인 그래픽적 요소들은 한계가 있기 때문에 어느 정도 구매하여 사용하기로 하였다.

■ 게임의 규칙

게임의 목표는 다수의 다른 유저들의 공격을 피해 최대한 높은 생존 시간을 가지는 것을 목표로 하며 여기서 생존 시간은 게임의 점수로 작용한다. 게임이 지루해지는 것을 막기 위해 다양한 특수 아이템들을 제공한다.

■ 게임의 종료

다른 플레이어에게 공격을 당해 자신의 캐릭터의 생명력이 0이 될 경우 그 플레이어의 게

임은 즉시 종료되며 다른 플레이어들의 게임은 계속해서 진행한다.

■ 게임의 시점

게임의 시점으로는 위에서 본 시점을 뜻하는 탑뷰(Top view) 시점을 채택하였다. 그 이유는 이번 과제에서 그래픽적인 요소를 구하는데 한계가 있는데 탑뷰 시점의 게임들은 다른 시점의 게임보다 그러한 요소들이 적게 사용되기 때문이다.

나. 게임 서버

1대의 기기에서만 즐기는 로컬 게임이 아닌 다수의 기기에서 다수의 플레이어가 상호작용하여 즐기는 멀티 게임에서 사용자의 상호작용을 담당하는 서버로, Node.js와 웹 소켓 프로토콜을 기반으로 하여 작동한다. 이번 과제에서는 게임의 전반적인 플레이어들의 상호작용 로직 처리와 클라이언트와 서버 간의 상호작용을 구현한다.

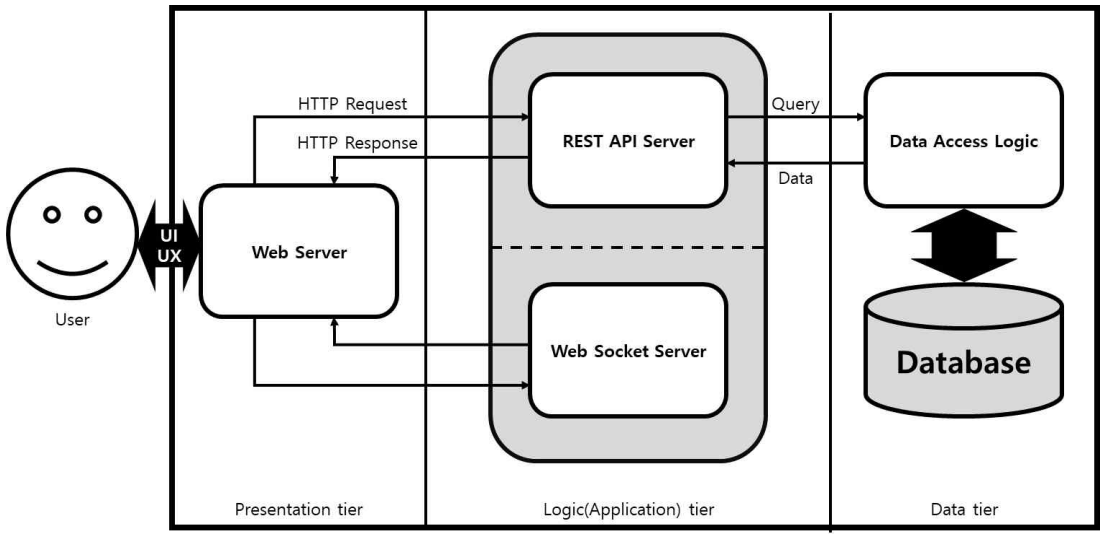
다. API 서버

API 서버는 게임 로직과 직접적인 연관이 있지는 않지만, 그에 필요한 정보들을 제공하는 역할을 맡는다. 예를 들어, 게임 이용자가 게임을 제공하는 웹 서버에 접속해 자신의 계정을 만들고 로그인하거나, 서버 전체 사용자의 순위 등을 확인할 때 API 서버가 관련 정보들을 제공하게 된다. API 서버는 HTTP를 사용한 REST API 서버로 구현한다. API 서버 개발은 데이터베이스 구축과 함께 진행되어야 하며 데이터베이스로는 NoSQL을 사용한다. API 서버 개발은 HTTP 메소드에 대한 이해, RESTful한 서버 설계, 데이터베이스 구축에 관한 전반적인 지식 등이 필요하다.

III. 과제 수행에 필요한 기반 지식, 기술

가. 관련 지식

■ 3계층 구조 (3-tier Architecture)



3계층 구조란 클라이언트-서버 시스템의 일종으로, 시스템을 표현 계층, 논리 계층, 데이터 계층 셋으로 나눠 각각 다른 플랫폼상에서 구현한 것이다. 포괄적인 개념으로 다층 구조(Multi-tier Architecture)가 있으며 일반적으로 3계층 구조가 가장 많이 사용된다. 3계층 구조는 사용자 인터페이스와 데이터베이스 서버로 이루어진 2계층의 단점을 보완하는 구조이다. 기존 두 계층 사이에 비즈니스 로직을 처리하는 논리 계층이 추가되며, 중간층은 트랜잭션 처리 모니터, 메시지 서버, 응용 서버 등 다양한 방법으로 구축될 수 있다.

● 표현 계층

표현 계층은 사용자 인터페이스를 담당하는 층이다. 프론트엔드(Front-end)라고도 불린다.

● 논리 계층

논리 계층은 비즈니스 로직 계층 혹은 애플리케이션 계층으로도 불리며, 표현 계층을 통해 받은 요청을 로직에 따라 처리하고 응답하는 역할을 한다. 구현에 따라 단순 큐 역할에서 복잡한 알고리즘을 가지는 매치 메이킹 서버의 역할까지 수행할 수 있다. 미들웨어(middleware) 혹은 데이터 계층과 묶어 백엔드(Back-end)라고 불린다.

● 데이터 계층

데이터 계층은 데이터베이스와 데이터베이스를 관리하는 로직을 포함한다.

■ HTML5

HTML5는 웹 문서를 표현하는 마크업 언어인 HTML에 DOM(The Document Object Model) API 스펙을 포함하는 HTML의 가장 최신 버전이다. 또한, 부가적으로 자바스크립트 API를 활용하여 다양한 2차원 및 3차원 그래픽을 지원하며 웹 소켓을 이용하여 웹(클 라이언트)에서 서버 측과 직접적인 양방향 통신이 가능하다.

● HTML5 게임

플랫폼에 독립적인 HTML5로 개발한 게임으로 타 기종에서 구동하기 위해 빌드 변경을 할 필요가 없이 같은 빌드로 여러 서비스가 가능하므로 일각에서는 차세대 게임으로 주목하기도 한다.

■ 경로 탐색 알고리즘

<a* 알고리즘> <Dijkstra 알고리즘>

게임에서 NPC(non-player character)의 움직임은 Dijkstra 알고리즘을 확장한 a* 알고리즘을

사용해 구현한다. a* 알고리즘은 경로의 가중치만을 사용하는 Dijkstra 알고리즘과 달리 '경로 가중치 + 목적지까지의 거리'를 사용하는 알고리즘이다.

■ 게임엔진

비디오 게임의 개발에 기반이 되는 구성 요소들을 가진 필수 구성 요소들인 그래픽 엔진, 물리 엔진, 오디오 엔진, UI 시스템, 게임플레이 프레임워크 등이 잘 융합된 상태의 소스 코드와 그 기능들을 디자이너들이 사용 가능한 툴을 겸비한 게임 개발 소프트웨어를 일컫는 말이다.

● Unity

유니티는 매우 직관적인 게임엔진이다. 게임엔진을 실제로 실행시켜 보면 비쥬얼 스튜디오처럼 여러창들이 나온다. 하지만 비쥬얼 스튜디오와는 다르게 전반적인 작업이 마우스로 이용하여 눈에 보이는 대로 설계가능하다. 또한 에셋스토어가 존재하여 구현하기 힘든 부분이 에셋스토어에 존재한다면 구매하여 사용이 가능하다. 이러한 점들이 게임개발에 익숙하지 못한 개발자들에게 큰 도움이 된다.

■ 웹 소켓(Web Socket)

웹 소켓은 HTTP와 같은 하나의 컴퓨터 통신 프로토콜이다. 그러나 웹 소켓 프로토콜은 HTTP 풀링과 같은 반이중방식에 비해 더 낮은 부하를 사용한 전이중 통신을 사용하여 웹 서버 간의 통신을 가능하게 하며, 서버와의 실시간 데이터 전송을 가능하게 한다. 이는 먼저 클라이언트에 의해 요청을 받는 방식이 아닌, 서버가 내용을 클라이언트에 보내는 표준화된 방식을 제공함으로써, 또 연결이 유지된 상태에서 메시지들을 오갈 수 있게 허용함으로써 가능하게 되었다. 구글의 크롬, MS사의 엣지, 파이어폭스, 사파리, 오페라 등 브라우저들 대부분이 지원하는 프로토콜이다.

■ Node.js

Node.js는 확장성 있는 네트워크 애플리케이션 개발에 사용되는 소프트웨어 플랫폼으로 자바스크립트의 향상된 런타임이다. 구글 크롬의 자바스크립트 엔진인 V8 엔진으로 빌드되었으며 자바스크립트를 브라우저에서만 사용하는 것이 아닌 브라우저 밖에서 다양한 용도로 확장하여 사용하기 위해 만들어진 것이다.

● Node.js의 특징

- **Non-blocking I/O**

Non-blocking I/O는 데이터 전송을 마치기 전에 기타 프로세스가 계속하도록 허가하는 입출력 처리의 한 형태로 Node.js에서는 데이터의 반환을 기다리지 않고 다음 API를 실행함으로써 속도를 높였다.

- **단일스레드 이벤트 루프**

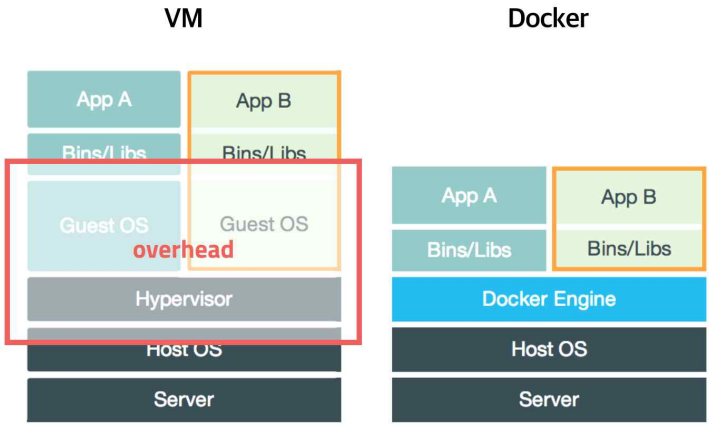
하나의 스레드에서 이벤트 루프의 반복을 이용한 작업을 처리하는 Node.js는 멀티스레드를 이용하는 것보다 요청 처리를 위한 Context Switching 비용은 비록 많이 들지만, 이벤트

큐를 이용한 작업을 통하여 많은 요청에 대해서도 순차적인 처리가 가능하다.

이와 같은 Node.js는 이미 넷플릭스, 페이팔, 우버 같은 회사들이 애용하고 있을 만큼 믿을 수 있는 성능과 안정성을 보여주고 있다.

■ 컨테이너(Container)와 도커(Docker)

컨테이너란 운영체제 수준의 가상화 기술로, 간단히 설명하자면 하나의 운영체제 커널을 공유하면서 프로세스들을 각각 격리된 환경에서 실행하도록 하는 기술이다. 이미 기존에도 가상머신(Virtual Machine)을 이용한 가상화 기술이 있지만, 이는 컴퓨터 환경 자체를 가상화하여 소프트웨어로 구현한 것이었기에 오버헤드가 존재한다.



컨테이너는 다양한 프로그램, 실행환경을 추상화할 수 있게 해주고 동일 인터페이스를 제공해 프로그램의 배포와 관리를 단순하게 만들 수 있다. 2014년 구글은 자사의 모든 서비스들(Gmail, YouTube 등)이 컨테이너로 동작하며 컨테이너를 매주 20억 개 구동한다고 발표하기도 했다.

도커(Docker)는 이 컨테이너 기술을 기반으로 하는 오픈소스 가상화 플랫폼이다. 도커는 리눅스 컨테이너를 사용하며 이미지(Image)와 레이어(Layer) 개념으로 관련 분야에서 가장 주목받는 기술이 되었다.

도커는 컨테이너 실행에 필요한 파일과 설정값 등을 포함하고 있는 이미지를 통해 프로세스를 가상화한다. 버전 관리 프로그램 git과 git 저장소 사이트 git hub처럼 도커 또한 이미지들을 업로드와 다운로드할 수 있도록 저장소 역할을 하는 Docker hub이라는 사이트가 존재한다.

■ REST(Representational State Transfer)[1)]

REST(Representational State Transfer)는 웹 기반 아키텍처로, 자원을 이름으로 구분하여 해당 자원의 상태(State)를 주고받는(Transfer) 구조이다. REST는 다음과 같은 제약조건으로 유도되었다.

- 클라이언트 – 서버 구조

1) Fielding, R. T. (2000). Architectural Styles and the Design of Network-based Software Architectures.

- 무상태성(Stateless)
- 캐시
- 통일된(Uniform) 인터페이스
- 계층화된 시스템
- 주문형 코드(Code-on-demand)

REST는 이 제약조건들을 따르는 구조를 의미하는 추상적 개념이며 그 구현에는 주로 HTTP를 사용한다. 이러한 구조를 구현한 시스템을 "RESTful" 하다고 표현한다.

HTTP를 사용한 구현은 HTTP URI(Uniform Resource Identifier)를 통해 자원(Resource)을 명시하고, HTTP Method를 통해 해당 자원에 대한 행위들(CRUD)을 구현하는 방식으로 이루어진다.

HTTP Method	의미	멱등성
POST	Create	X
GET	Read	O
PUT	Update	O
DELETE	Delete	O

■ REST API(Application Programming Interface)

REST API는 앞서 설명한 REST 구조를 적용해 구현한 API 서비스를 의미한다. API 서비스는 시스템을 분산해 확장성과 재사용성을 높여 유지보수 및 운용을 편리하게 할 수 있게 만들어준다. 구글, 카카오 등의 기업에서 이 기술을 적용하였고, 이를 통해 자사의 OpenAPI를 공개하고 있다. REST API를 설계하는 데 있어 가장 중요한 점은 자원을 URI로 어떻게 표현하느냐이다. 다음 두 기본 원칙을 지켜야 한다.

1. URI는 정보의 자원을 표현해야 한다.
2. 자원에 대한 행위는 HTTP Method로 표현한다.

즉, URI는 오직 자원을 표현하는 역할만을 해야 한다. URI에는 행위를 나타내는 동사형 단어나 get, post 등 HTTP Method를 포함하면 안 된다. 이 원칙에 따라 URI는 다음과 같은 규칙들을 따르는 것이 좋다.

- 슬래시(/)를 마지막 문자로 포함하지 않는다.
- 단어의 결합이 불가피한 경우 under bar(_)보단 dash(-)를 사용한다.
- 소문자를 사용한다.
- 파일 확장자를 URI에 포함하지 않는다.

URI 예제		
CRUD	Request	
모든 채널의 리스트를 가져온다.	GET	http://myapiserver.com/channels
특정 채널의 정보를 가져온다.	GET	http://myapiserver.com/channels/:id
채널을 생성한다.	POST	http://myapiserver.com/channels
특정 채널을 수정한다.	PUT	http://myapiserver.com/channels/:id
특정 채널을 삭제한다.	DELETE	http://myapiserver.com/channels/:id

URI 설계 이외에도 다양한 설계 지침[2]이 있으므로 참고하여 개발하여야 한다.

2) 이상학의 개발블로그, RESTful API 설계 가이드 (https://sanghaklee.tistory.com/57)

■ NoSQL(non SQL 또는 Not only SQL) 데이터베이스

NoSQL 데이터베이스는 빅데이터와 함께 등장한 개념이다. 기존의 전통적인 관계형 데이터베이스 보다 덜 제한적인 일관성 모델을 이용하는 데이터의 저장 및 검색을 위한 메커니즘을 제공한다.[3] 관계형 모델을 사용하지 않으며 스키마가 없다는 특징을 가진다.

NoSQL 데이터베이스는 일관성을 타협한 대신에 수평 확장(Scale-out)에 이점을 가지므로 빅데이터를 다루는 데에 유리해 주목받는 기술이다. 또한, NoSQL은 특정 한 대상을 지칭하는 것이 아니라 다양한 SQL이 아닌 다양한 구조의 데이터베이스들을 통칭하는 용어이다. NoSQL에 포함되는 데이터베이스 모델에는 다음이 있다.

- Column-family
- Document
- Key-value
- Graph

대표적으로는 Key-value 모델을 사용하는 Redis, Document 모델을 사용하는 MongoDB가 있으며, 이번 프로젝트에서는 MongoDB를 사용한다.

나. 개발 환경

1) 공통

- OS : Windows 10
- 형상관리 도구 : Git
- 호스팅 서비스 : 미정
- Docker

2) 클라이언트

- 소스 코드 편집기 : Visual Studio Code
- 개발 언어 : Javascript
- 게임엔진 : Unity
- 그래픽 라이브러리 : WebGL

3) 게임 서버

- 소스 코드 편집기 : WebStorm
- 개발 언어 : Node.js(Javascript)
- 웹 소켓 프레임워크 : Socket.io

4) API 서버

- 소스 코드 편집기 : Visual Studio Code
- 개발 언어 : Python
- 서버 프레임워크 : Flask
- DBMS : MongoDB(NoSQL)

---

3) Wikipedia, NoSQL 문서 (https://ko.wikipedia.org/wiki/NoSQL)

IV. 스케줄

	업무	5월	6월	7월	8월	9월	10월
1	전반적인 스터디						
2	프론트엔드의 기초 구축						
3	게임 개발						
4	게임 서버 개발						
5	API 서버 개발						
6	테스트와 유지보수						

V. 팀원의 업무

■ 201412824 이지훈

API 서버 구축 및 데이터베이스 설계

■ 201412779 강형석

게임 서버의 전반적인 구축과 프론트엔드 일부 구축

■ 201412790 김진규

게임 기획과 제작 및 조장