주간 진행 보고서

201412824 이지훈
201412779 강형석
201412790 김진규

<table>
<tr><td>주제</td><td>3계층 구조 Multi-player Online 웹 게임 시스템 구축</td></tr>
<tr><td>진행 날짜</td><td>2020.5.19 ~ 2020.5.31</td></tr>
<tr><td>김진규
(팀장)</td><td>이번 주간에는 유니티에 필요한 C#을 학습하였습니다. C#의 기본적인 흐름은 이때까지 학습했던 언어들과 큰 차이는 없었습니다. 환경은 VisualStudio2019로 진행하였습니다. 다른 언어와 약간의 차이는 있었지만 그것을 깊숙이 파고드는 것 보다는 개발에 쓰일 기본적인 부분만 학습하여 필요한 부분은 다시 조사해보는 방식으로 학습을 진행하였습니다. 그래서 C#강의 사이트(https://www.csharpstudy.com/CSharp/CSharp-intro.aspx)와 유튜브(https://www.youtube.com/watch?v=OJM-1Usv68k&list=PLUZ5gNInsv_O7XRpaNQIC9D5uhMZmTYAf)를 이용하여 학습을 진행하였습니다.

<table>
<tr><th>C</th><th>C++</th><th>C#</th></tr>
<tr><td>절차 지향</td><td>객체 지향 지원</td><td>객체 지향 지원</td></tr>
<tr><td>가장 낮은 수준의 추상성</td><td>낮은 수준의 추상성</td><td>높은 수준의 추상성</td></tr>
<tr><td>메모리 직접 관리</td><td>메모리 직접 관리</td><td>가비지 컬렉션</td></tr>
<tr><td>컴파일 했을 때 매우 가벼움</td><td>컴파일 했을 때 가벼움</td><td>바이트코드로 인터프리팅한 이후에 CLR이 컴파일을 하며, 오버헤드를 포함하고 있고, 바이너리의 사이즈가 더 큼</td></tr>
<tr><td>빠르고 최고의 성능 자랑</td><td>C와 비슷한 성능</td><td>일반적인 성능</td></tr>
<tr><td>어떤 플랫폼에도 코딩 가능</td><td>어떤 플랫폼에도 코딩 가능</td><td>윈도우 OS 위주</td></tr>
<tr><td>문법이 틀리지만 않는다면 거의 모든 것을 허용</td><td>문법이 틀리지만 않는다면 거의 모든 것을 허용</td><td>컴파일러가 경고를 보여줘서, 심각한 에러를 방지</td></tr>
<tr><td>임베디드(embeded) 장비와 시스템 수준의 코드에 적합</td><td>서버 어플리케이션, 네트워킹, 게임, 장치 드라이버에 적합</td><td>간단한 웹, 모바일, 데스크톱 어플리케이션에 적합</td></tr>
</table>
C#은 C++보다는 Java와 비슷한 구석이 많다고 느껴졌고 위에서 언급했던 것처럼 큰 차이는 없는 것 같았습니다. 왼쪽의 표는 C, C++과 비교한 표를 참고한 것입니다.
Java와의 차이점 또한 저런식으로의 차이만 존재할 뿐 크게 신경쓰일 점은 없었습니다.

다음주부터는 C#을 실제로 유니티에 접목시켜 실제이용 가능한 기능들을 부분적으로 구현하며 진행하도록 하겠습니다.</td></tr>
</table>

강형석

Node.js 웹서버를 이용해서 매치메이킹 서버를 만들어보았다.

1. 메치메이킹이란

말 그대로 게임의 매칭을 맺어주는 것을 뜻하며 상황에 따라 플레이어에 실력을 동등하게 맞추는 등 다양한 조건을 걸 수 있다.

2. 메치메이킹 서버를 만든 이유

우리가 어떤 게임을 만드는지 아직도 잘 모르기 때문에 섣불리 프로젝트를 시작하고 싶지 않았고 별도로 프론트엔드로 만들어진 것이 단 하나도 없기 때문에 우선적으로 웹 소켓과 조금의 프론트엔드로 테스트 할 수 있는 매치메이킹 서버를 구축해보았다.

3. node.js 설치 & 모듈 socket.io 설치

node.js 공식사이트에서 설치하였으며 여기서는 12.16.1 버전을 사용하였다.
또한 socket.io의 경우 node.js에 내장되어 있는 npm(node package manager)을 사용하여 아래의 코드 한 줄로 쉽게 설치 가능했다.

```
npm install socket.io
```

4. 주요 코드 설명

```
var io = require('socket.io').listen(999);
var matchMakingPool = newArray();
matchMakingPool.push('none');
var user = { socketId:'', userId:'', elo:''}
```

우선 socket.io 모듈을 require를 통해 불러오고 포트는 원하는 번호로 열어둔다.
매치메이킹 풀은 배열로 구현했으며 기본적으로 아무것도 없음을 뜻하는 한 칸을 넣어둔다.
또한 효율적으로 운용하기 위해 user라는 객체를 만들었다. 이 객체는 소켓 id, 유저 식별이 가능한 유저id, 간단하게 유저의 실력을 알 수 있는 수치 elo를 가지고 있다.

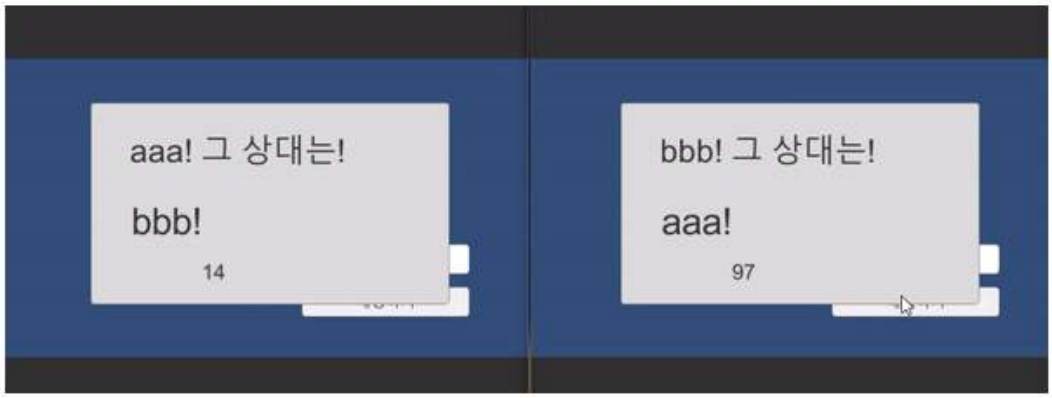
```
//새로운 유저 연결되었을 시
io.sockets.on("connection", function(socket){ /
/클라이언트에게 매치메이킹 서버에 연결되었음을 알립니다.
io.to(socket.id).emit('NoticeConnected');
```

sockets.on은 이벤트를 수신 시 동작할 행동을 적어놓으면 되는데 클라이언트에서 연결 시 자동으로 connection이 수행된다.
emit은 반대로 이벤트를 송신 시 사용하는데 to를 이용하여 특정 대상에게만 송신하였다.
방금 연결된 클라이언트에게 정상적으로 연결되었다는 'NoticeConnected' 이벤트를 송신하였다.

```
//유저의 정보를 받아옵니다.
socket.on('UpdateUserInfo',function (data){
    user={socketId: socket.id, userId: data.userId , elo: data.elo};
    //기다려야 하는가?
    var isWait=true;
    for(var i=0;i<matchMakingPool.length;i++){
     //기다리고 있는 상대가 있을 경우
     if(matchMakingPool[i]!='none'){
         var temp=matchMakingPool[i];
         console.log('상대는 이녀석',temp);
         isWait=false;
         io.to(matchMakingPool[i].socketId).emit('FoundUser',user);
         io.to(socket.id).emit('FoundUser',matchMakingPool[i]);
         matchMakingPool[i]='none'
         }
      }
     if(isWait) {
     console.log('상대가 없어..',user)
     matchMakingPool.push(user);
     }
     console.log('socketId:',socket.id,' Id:',user.userId);
     });
```

정상적으로 NoticeConnected를 수신받은 유니티 클라이언트는 다시 UpdateUserInfo 이벤트를 서버측으로 보내는데,
이 때 자신의 정보를 담은 data와 함께 emit하게 된다.
서버는 받아온 data를 이용해 새로운 user 객체를 만들어 받은 정보로 초기화한다.
그 후 매치메이킹 배열에 이미 user가 존재할 경우 그 user와 방금 연결된 user에게 서로의 정보를 넘겨주며 'FoundUser'를 emit한다.
또한 조건에 맞는 상대가 없을 경우 매치메이킹 배열에 방금 접속한 user을 추가 한 후 마친다.
이 때 굳이 유저의 정보를 받음과 동시에 매치메이킹 배열에 추가하는 이유는 받아온 data를 후에 다른 이벤트 함수에서 참조 할 수 없어서 어쩔 수 없이 정보를 받는 이벤트 처리를 하며 매치메이킹 처리도 하였고 아직 해결하지 못하였다.

5. 결과물

<table>
<tr>
<td></td>
<td>
<pre>
C:\Users\paper\Desktop\paperfrog\nodejs\matchmakingServer>node main
상대가 없어.. { socketId: 'eeL-lRWjZR_lZrjcAAAA', userId: 'aaa', elo: 97 }
socketId: eeL-lRWjZR_lZrjcAAAA   Id: aaa
상대는 이녀석 { socketId: 'eeL-lRWjZR_lZrjcAAAA', userId: 'aaa', elo: 97 }
socketId: l4F9OwEkgl9ljluVAAAB   Id: bbb
</pre>


</td>
</tr>
<tr>
<td>이지훈</td>
<td>
이번 기간에는 Python Flask 프레임워크를 사용하여 API Server 개발 방법을 학습하였다. API 서버는 본 프로젝트를 HTTP 웹 서버의 측면에서 봤을 때, MVC 패턴의 Model과 Controller를 커버한다. API 서버는 클라이언트의 요청에 대한 데이터 표현 및 관리, JWT(JSON Web Token)을 사용한 사용자 인증 등을 수행할 수 있어야 한다. 학습에는 다음 자료를 참고하였다: https://dejavuqa.tistory.com/273. 본 참고자료에서는 Flask로 RESTful API 서버를 개발할 때의 디렉터리 구성 방법, 프로젝트 설정 관리법, 기본적인 API 서버 구현, 사용자 인증, Flask에서 테스트 자동화 모듈 사용법 등을 소개한다. 참고자료는 짧은 시간 안에 학습 목표를 간단히 구현해 보기에 적합했지만, 더는 유지보수되지 않는 모듈이 일부 사용되었고 데이터베이스로 SQLite가 사용되었기 때문에 이를 보완하기 위한 추가적인 학습이 필요하였다.
개발에 사용한 핵심적인 모듈은 다음과 같다.
- flask
- unittest, flask_testing : 유닛 테스트 관련 모듈
- pymodm : Python용 MongoDB ODM(Object-Document Mapper)
- flask_bcrypt : 해시 모듈
- flask_restx : REST 구현 모듈
- jwt : Python JWT 모듈

● 디렉터리 구성
- manage.py : 프로그램 entry point
- app/test/ : 테스트 코드
- app/main/ : API Server 메인 코드
</td>
</tr>
</table>

- controller/
- model/
- service/ : MVC 패턴 구현 코드
- util/ : dto, route protection 등 보조 기능 구현 코드
- config.py : 서버 설정 정보

● JSON Web Token(JWT)

REST API 구조는 무상태성(Stateless)을 지향한다. 따라서 서버에 상태를 저장해야 하는 기존 쿠키-세션 방식은 REST API 서버에 부적합한데, 이를 해결하기 위해 등장한 방식이 토큰 방식이다. 토큰 방식은 사용자에게 인증 정보가 담긴 토큰을 발행해 사용자가 서버에게 요청할 때마다 해당 토큰을 서버에 전달하는 방식이다.

JWT는 이 웹 토큰의 구현에 JSON 형식을 사용하는 방식이며, Web 표준으로 등록되어 있다. JWT의 구조는 다음과 같다.

- Header : JWT 웹 토큰의 헤더 정보
  - typ : 토큰 타입
  - alg : 해싱 알고리즘
- Payload : 실제 토큰으로 사용하려는 데이터(Claim)가 담기는 부분
  - Reserved claims : 예약된 Claim
  - Public claims
  - Private claims
- Signature : 토큰의 무결성과 변조 방지를 위한 서명

```
{
  "iss": "velopert.com",
  "exp": "1485270000000",
  "https://velopert.com/jwt_claims/is_admin": true,
  "userId": "11028373727102",
  "username": "velopert"
}
```