**서버와 클라이언트 측 진행상황**

**Computer Software Engineering**

**Professor Name HEE-MIN PARK**

201321333 JAE-SUN HAN

201021340 YONG-RYUL WON

**10/10/2016 ~ 10/11/2016 진행 상황**

**1. 클라이언트 측 진행 상황**

**1-1. 메인 화면과 함선 간의 충돌 처리 완료**

: 서버 측에서 받은 좌표를 계속 받을 때, 즉, 클라이언트 현재 함선이 메인 화면의 최대

넓이 및 최소 넓이를 지나 갈 때의 좌표를 계산하여 그 밖으로 나가지 못하게 끔 처리

하였습니다.

**2-1. UI 디자인**

: 행성과 랭크의 대체적인 뼈대를 css를 활용하여 60%정도 완료 하였습니다.

: 미니 맵을 m키로 켜고 끌 수 있게 하였습니다.

: minimap\_ui.js 부분을 작성해 나가고 있습니다.

(게임 자체에는 중요한 사항이 아니라 판단하여 시간이 남을 때 마다 작성을 하고 있습니다.)

: 미니 맵에서 대략적인 나의 위치를 파악 가능하나 좀 더 명확한 나의 현재 위치 값을

알 수 있으면 좋겠다 싶어 나의 현재 위치를 볼 수 있는 부분을 미니맵 아래 추가하였음.

**1-3. Index.html을 컨트롤 하는 index.js 코드를 수정(서버 측 진행자가 좀 더 보기 쉽게 작성)**

**1-4. main\_layer 자체를 로컬에서 움직이는 문제**

: 이것은 main\_layer(background) 자체를 body안에서 움직일 때 마다 계속 다 그리기 때문

에 전에 말씀하신 함선의 주변만 그리거나 view\_layer를 만을 함선에 맞춰 이동해볼 것을

생각하고 있습니다.

**1-5. 행성과 함선 간의 충돌 처리 문제**

: 충돌 처리를 클라이언트 자체에서 하여 보낼 것이냐 아니면 서버에서 계산하여 일괄적으

로 보낼 것 인가에 관하여 팀원과 회의 결과 멀티 게임의 경우는 서버에서 보내는 것이 합

당하다 생각하여 서버에서 충돌 로직을 처리하여 보내기로 하였고, 테스트 시 함선과 행성

이 충돌되어 메시지를 받는 것을 확인하였습니다. 따라서 그에 따른 클라이언트 상에서의

처리 코드를 작성하고 있습니다.

: 이 충돌처리에 관하여 2d collision detection model 중 AABB(Axis-Align Bounding Box)

Collision Detection Algorithm을 참고 하였습니다.

**[테스트를 해보며 생각하고 있는 문제]**

이동 및 화면 밖으로 나가지 않는 충돌처리 구현(구현 되었음)을 하던 중 심각한 문제가 발생하였는데, 기존의 로직은 로그인 로그아웃 시 소켓 따로 연결, 키 입력 시의 해당 좌표 업데이트에 따른 화면 출력을 하여 간헐적인 통신을 통하여 부하를 줄이고 원활한 ux(user experience)를 생각하였다.

그러나 이 방법에는 크나큰 맹점이 존재한다.

그것은 로그인 로그아웃 시의 포트에서 키 입력을 임의로 받아 업데이트를 한다. 논리 자체는 문제가 없어 보인다. 그러나 이것은 하나는 해결(서버에게 나의 로그인과 로그아웃 상태를 알려주어 데이터를 받는다.)는 충족하지만 키 입력 시의 업데이트 처리(입력에서의 json데이터를 받는 순간까지는 문제가 없으나, 받고 나서의 처리가 문제(받는 즉시 태그를 생성하도록 한다. ‘나’는 태그를 한 번만 생성하게 하고 싶으나 받는 즉시 계속하여 중복 태그가 생성되어 문서 내의 태그가 무한으로 생성)가 된다.

따라서 클라이언트에서 생각한 논리적 흐름은 다음과 같다.

로그인 → 소캣 연다 → 소켓을 받으면서 그 안의 데이터를 읽어들인다(여기에서의 데이터는 유저가 로그인을 했냐/안 했냐에 따른 connect(서버의 데이터베이스에선 access(true/false) message) → 페이지에 갱신한다(이때, 응답을 받을 시 페이지의 onload시 function에서 access : true에 따른 태그 apeend를 ‘한 번’만 하게됨. 그리고 나서 다시는 요청하지 않음. 즉, 로그인 한 유저만 추린 것을 받아서 뿌림) → 추후 게임 진행 시에는 기존과 같이 key press event를 통한 서버와의 통신을 하여 지속적인 user position 갱신을 하게 됨 → 유저가 로그아웃 시에 로그인 할 시와 반대로 access : false인 db의 정보를 받아 갱신(여기 선 해당 태그를 remove하게 된다.)

따라서 이를 위한 저장소의 접근이 필요하다고 판단된다.

Logic flow is,

if(Mongodb.user.connect.access == true) {

// Connect socket and appendChild tagging

At once <= append(“<div id =’” + received.data.user[‘name’] + “’ position =‘ … + “></div>”);

// Request to server, receive data to server and update html page(reload)

}

After press key down event, connect socket that name is

‘user position(left, right, up, down) update’.

if(Mongodb.user.connect.access = false) { // that user disconnect

// Connect socket and removeChild tagging

At once <= userId.removeChild();

// Request to server, receive data to server, update html page(reload)

}

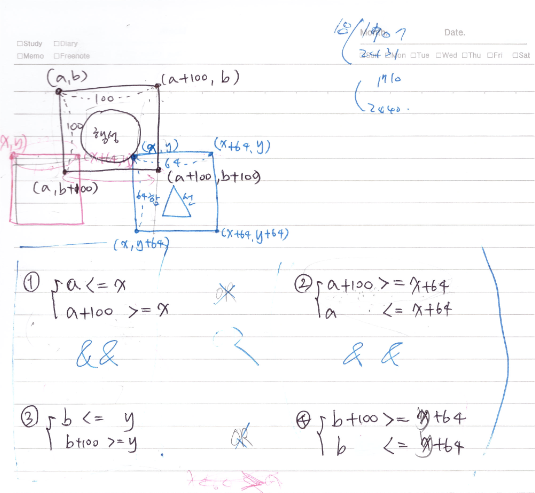
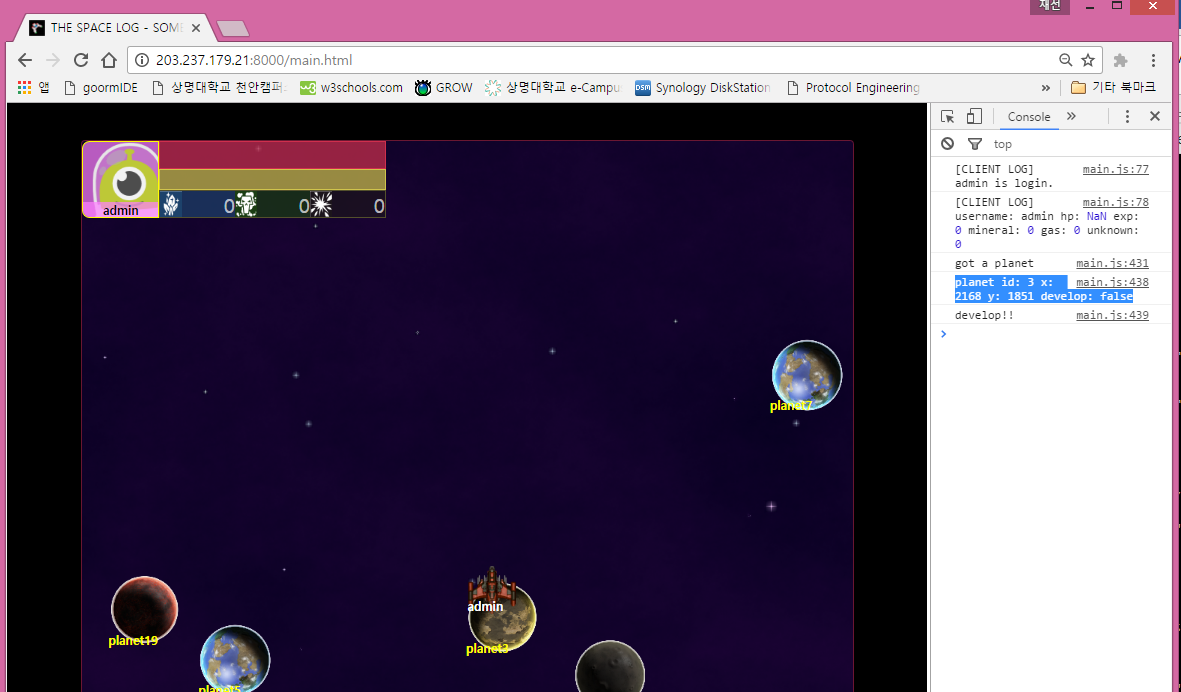
**2. 서버 측 진행 상황**

**2-1.행성과 함선 간의 충돌 처리**

**:** 행성과 함선이 충돌 했을 때 스페이스 바 입력을 받으면 해당 행성의 정보를 출력하는 부

분을 진행 중입니다. 처음에는 클라이언트에서 직접적으로 충돌 처리를 하는 방향으로 진

행하려고 했으나 서버에서 행성의 위치도 알고 유저의 함선 실시간 위치도 알기 때문에

서버에서 충돌여부를 판별하는 것이 더 나은 솔루션이라고 판단했습니다.  
  
충돌 원리를 아래의 그림과 같이 정리를 해보았습니다.  
  
 **1번 식(함선이 오른쪽에서 접근하여 행성과 충돌)과 2 번 식(함선이 왼쪽에서 접근하여 행성과 충돌)이 x 좌표의 충돌 식이고, 3번 식(함선이 아래에서 접근하여 행성과 충돌)과 4 번 식(함선이 위에서 접근하여 행성과 충돌)이 y 좌표의 충돌 식입니다. 이렇게 먼저 식을 세우고 코드를 짰습니다.**  
 **73라인에 들어가 조건문이 위의 충돌식을 구현한 것입니다. 이렇게 구현하고 나면 충돌 판별 난 행성의 정보를 getP라는 객체에 저장하여 클라이언트에게 전송합니다.  
 아래 그림은 유저가 행성과 충돌했을 때 space bar를 누르면 서버로부터 받은 행성 정보를 console.log 로 출력하는 그림입니다. 오른쪽 콘솔 창에 파란 블록지정된 부분이 space bar를 눌렀을 때 받은 정보를 출력한 부분입니다.   
  
이제 다음으로 진행할 부분은 이 정보를 바탕으로 space bar 눌렀을 때 행성의 정보를 띄우는 UI를 만들고, 유저가 개척하겠다는 요청을 서버로 보내면 서버가 1분에 10씩 유저가 개척해서 얻은 자원 량을 증가시키는 작업을 해야 합니다. 이 부분이 마무리 되고 나면 랭킹을 매기고 출력하는 부분을 작성할 예정입니다.**