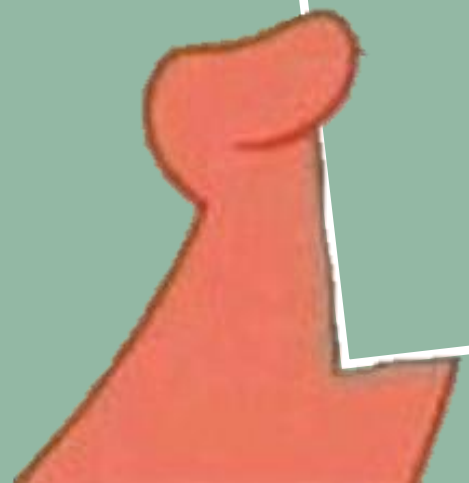
고급 웹 프로그래밍

기말 프로젝트

2015112912 김희원

INDEX

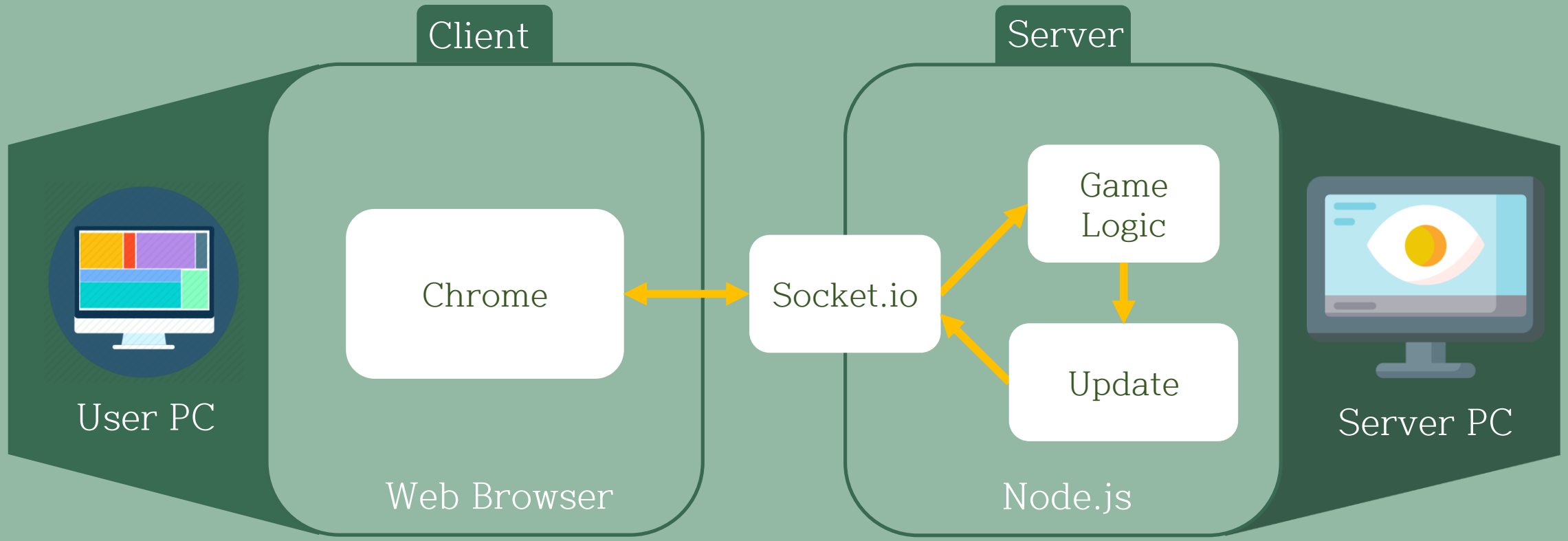


PROGRAM

Space Shooting

- WEB 에서 작동하는 우주 슈팅 게임
- 웹 소켓으로 멀티플레이 가능
- 스코어 보드 및 랭킹 시스템 구현

SYSTEM ARCITECTURE DIAGRAM



개발 / 시스템 환경

개발

- Windows + HTML / CSS / JAVA SCRIPT
- Main Editor, IDE : Atom, Chrome
- Graphic Processing: p5.js
- Server Side : Node.js
- Web Socket : Socket.io
- Graph Visualization : d3.js
- Etc : express (node.js module)

시스템 환경

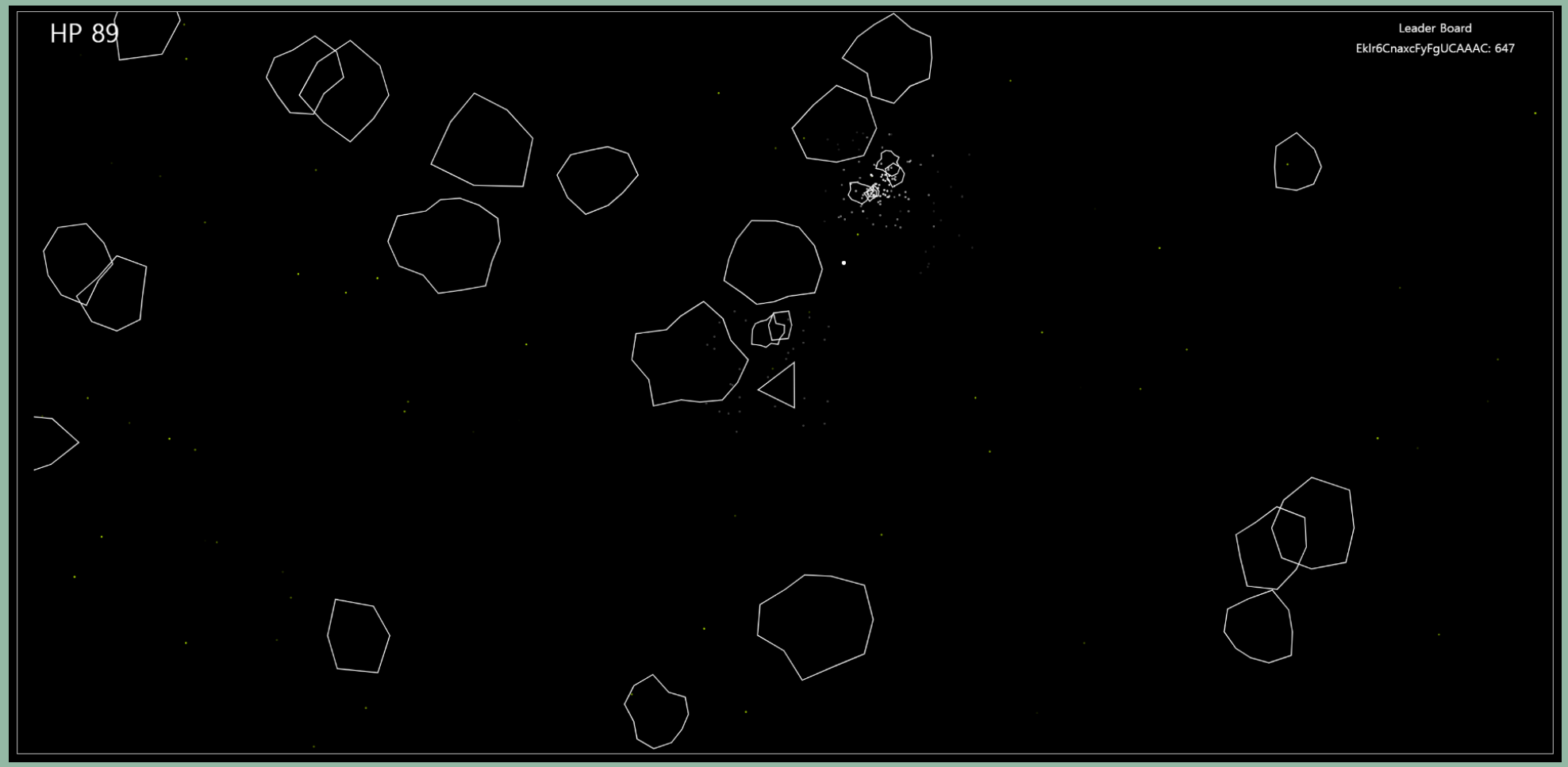
Web Browser

시작화면


```
<body style="background-image: url('space.jpg'); background-size: cover;">
<div id="myDiv" onmouseover="this.style.color = '#AABBCC'" onmouseout="this.style.color = 'white'">Advanced Web Programming</div>
<a id="myA" href="game.html"><div onmouseover="this.style.color = '#AABBCC'" onmouseout="this.style.color = 'white'">Game Start</div></a>
```

시작화면 (index.html) : 깔끔한 디자인을 위해 풀 스크린 이미지 위에 분위기 있는 폰트를 적용함
‘Game Start’ 문자열에 ‘a 태그’를 등록해서 게임 화면으로 전환 가능

게임 화면



우주 테마의 슈팅 게임,
윈도우 크기에 맞춰서 풀 스크린으로 게임 화면을 구성

게임 화면

```
C:₩Users₩kvlks₩nodejs₩multi_as>node server.js
Connected at 3000
user disconnected : Km90UCiM3Un9nhVOAAAA
[]
{ id: '8NAWDuKOspd3l9eAAAAB',
  x: 743,
  y: 364.5,
  h: 1.5707963267948966,
  l: [],
  s: 0,
  hp: 100,
  t: 0,
  d: false }
```

서버가 관리하는 정보

x,y : 우주선의 좌표
H: 우주선이 향하고 있는 방향
l : 발사한 레이저의 좌표 리스트)
s: 운석 파괴 점수
Hp : 체력
T : 생존 시간 점수
D : 데미지를 입고있는지 여부

게임 화면

```
function keyPressed(){
  if (key == ' '){
    //lasers.push(new Laser(ship.pos, ship.heading));
    var v = p5.Vector.fromAngle(ship.heading);
    var vx = v.x;
    var vy = v.y;
    var data = {px: ship.pos.x, py: ship.pos.y, vx:vx, vy:vy};
    socket.emit('laser', data);
    shoot.play();
  }
  if (keyCode == RIGHT_ARROW){
    ship.setRotation(0.1);
  }else if (keyCode == LEFT_ARROW){
    ship.setRotation(-0.1);
  }else if (keyCode == UP_ARROW){
    ship.boost();
    ship.boosting(true);
  }
}
```

P5.js의 메소드를 오버라이딩해서 키보드 입력을 구현

우주선의 행동은 클라이언트가 수행한 다음 서버에게 통지하는 식으로 구현

레이저의 발사는 다른 운석과의 충돌을 감지해야 하므로 서버에서 좌표 관리

게임 화면

```
1
2  function Ship() {
3    this.pos = createVector(width/2, height/2);
4    this.r = 20;
5    this.heading = PI/2;
6    this.angle = 0.1;
7    this.rotation = 0;
8    this.isBoosting = false;
9    this.vel = createVector(0, 0);
10
11   this.boosting = function(b){
12     this.isBoosting = b;
13   }
14
15   this.update = function(){
16     if(this.isBoosting){
17       this.boost();
18     }
19     this.pos.add(this.vel);
20     this.vel.mult(0.95);
21   }
```

Ship.js : 우주선 클래스

우주선의 속성인 좌표, 방향, 회전 여부를 프로퍼티로 두고 이를 관리하기 위한 메소드들을 구현

게임 화면

```
function Asteroid(pos, r){
  if (pos){
    this.pos = pos.copy();
  }else {
    this.pos = createVector(random(width), random(height));
  }

  if(r){
    this.r = r*0.5;
  }else{
    this.r = random(30,50);
  }

  this.vel = p5.Vector.random2D();
  this.total = floor(random(5, 15));
  this.offset = [];
  for (var i=0; i< this.total; i++){
    this.offset[i] = random(-this.r*0.5, this.r*0.5);
  }

  this.update = function(){
    this.pos.add(this.vel);
  }
```

Asteroids.js : 운석 클래스

플레이어가 닿으면 데미지를 입게 되며 플레이어는 레이저로 운석을 파괴할 수 있음

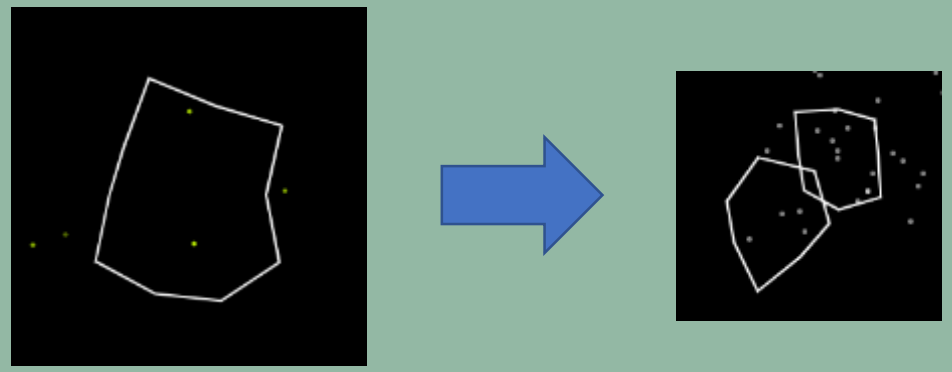
초기에 1인용 버전을 만들 때는 클라이언트 측에서 각자 관리했지만

멀티 유저용 버전을 만들기 위해 서버측에서 관리

게임 화면

```
this.breakup = function(){
  var newA = [];
  newA[0] = new Asteroid(this.pos, this.r);
  newA[1] = new Asteroid(this.pos, this.r);
  return newA;
}
```

Breakup 함수 : 운석이 파괴되면 반지름이 절반이고 각각의 행동 반경을 가지는 운석 2개가 생김

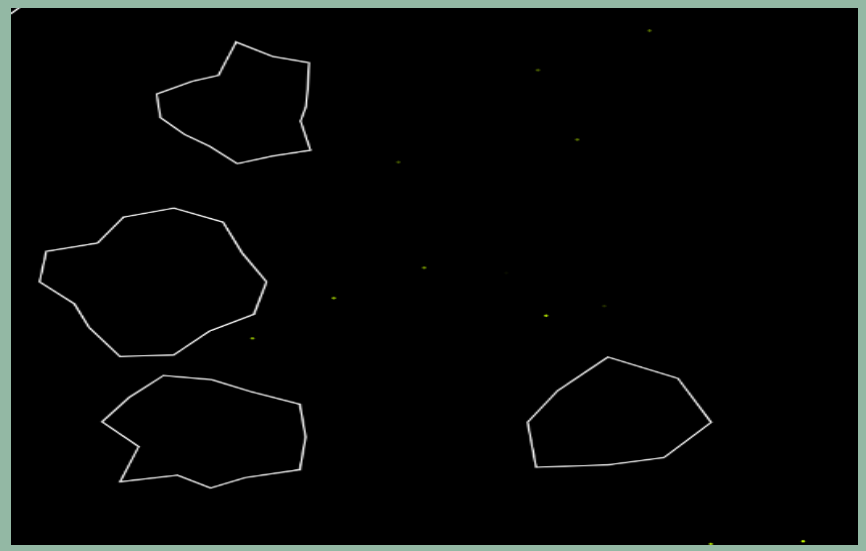


게임 화면

```
this.render = function() {
  push();
  noFill();
  stroke(255);
  translate(this.pos.x, this.pos.y);
  //ellipse(0, 0, this.r*2);
  beginShape();
  for (var i=0; i<this.total; i++){
    var angle = map(i, 0, this.total, 0, TWO_PI);
    var r = this.r + this.offset[i];
    var x = r * cos(angle);
    var y = r * sin(angle);
    vertex(x, y);
  }
  endShape(CLOSE);
  pop();
}
```

운석의 화면 구성

1. 우선 원의 형태로 시작
2. 원의 각도를 5~15의 랜덤한 각도로 N 등분 한 다음 각 점의 좌표를 직선으로 잇는 다각형 생성
3. 각 좌표에 -5~+5의 랜덤한 offset을 둬서 운석의 찌그러짐 구현

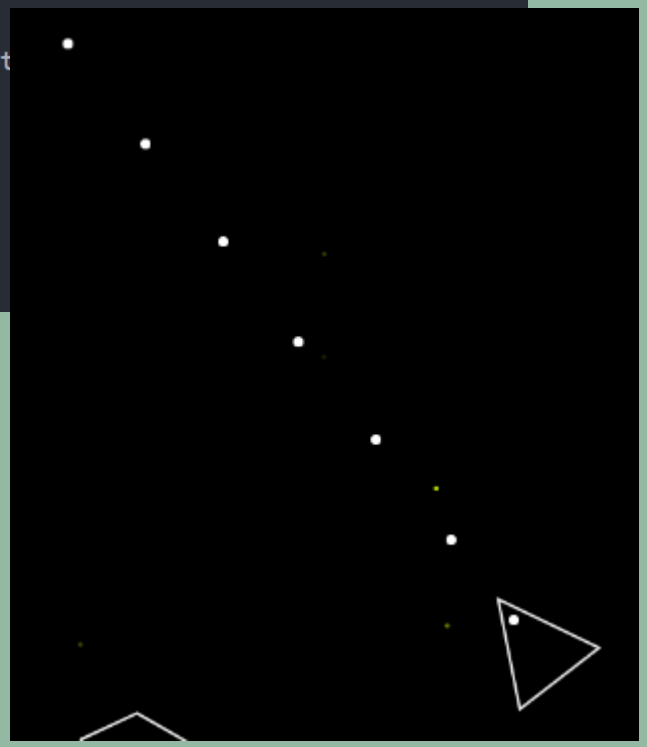


게임 화면

```
function Laser(spos, angle) {
  this.pos = createVector(spos.x, spos.y);
  this.vel = p5.Vector.fromAngle(angle);
  console.log(this.vel);
  this.vel.mult(10);

  this.update = function() {
    this.pos.add(this.vel);
  }

  this.hits = function(asteroid){
    var d = dist(this.pos.x, this.pos.y, ast
    if(d < asteroid.r){
      return true;
    }
    return false;
  }
```



laser.js : 레이저 클래스

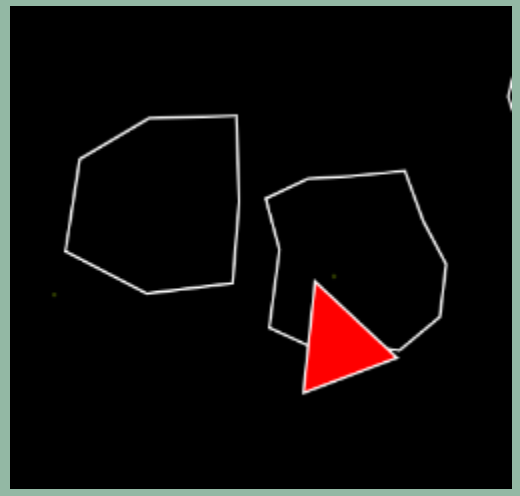
플레이어는 스페이스바 버튼을 누르면 레이저를 발사할 수 있음

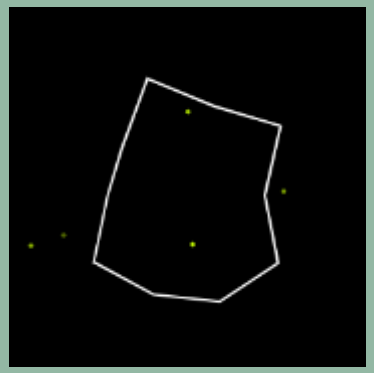
레이저 발사 요청은 웹 소켓으로 서버에게 전송됨

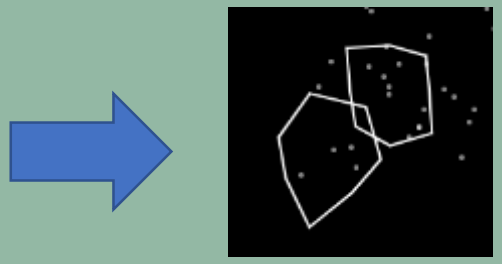
서버는 클라이언트가 부가적으로 보낸 현재 좌표와 각도를 바탕으로 레이저를 생성해줌

게임 화면

```
var d = Math.sqrt((asteroids[a].pos.x-ships[i].x)*(asteroids[a].pos.x-ships[i].x) + (asteroids[a].pos.y-ships[i].y)*(asteroids[a].pos.
//console.log(d);
if(d < 20 + asteroids[a].r)
```







수학적 distance를 이용하여 충돌 계산

레이저와 운석이 닿으면
운석이 2개로 나뉘어짐
(운석의 반지름이 10px이하라면 소멸)

우주선과 운석이 닿으면 단위시간마다
Hp가 감소함. Hp가 0이 되면 게임 종료
(결과 페이지로 이동)

충돌시 Boolean 변수를 둬서 우주선의 색을 red로 변경

게임 화면

```
class Particle{

constructor(){
  this.x = random(width);
  this.y = random(height);
  this.vx = random(-0.3, 0.3);
  this.vy = random(-0.1, 0.1);
  this.alpha = 255;
}

finished(){
  return this.alpha < 0;
}

update() {
  this.x += this.vx;
  this.y += this.vy;
  this.alpha -= 4;
}

show(){
  fill(200, 255, 0, this.alpha);
  noStroke();
  ellipse(this.x, this.y, 2);
```

Particle2.js Explode.js, firework.js

particle2는 기본적으로 우주테마의 느낌을 내기 위한 배경 이펙트 생성 클래스

Explode와 firework는 레이저와 우주선 충돌시 이펙트를 위한 클래스임

Particle은 랜덤으로 입자 생성(리스트로 관리) -> 투명도를 서서히 감소 시키는 애니메이션 -> 투명도가 일정 이하가 되면 리스트에서 제거

게임 화면

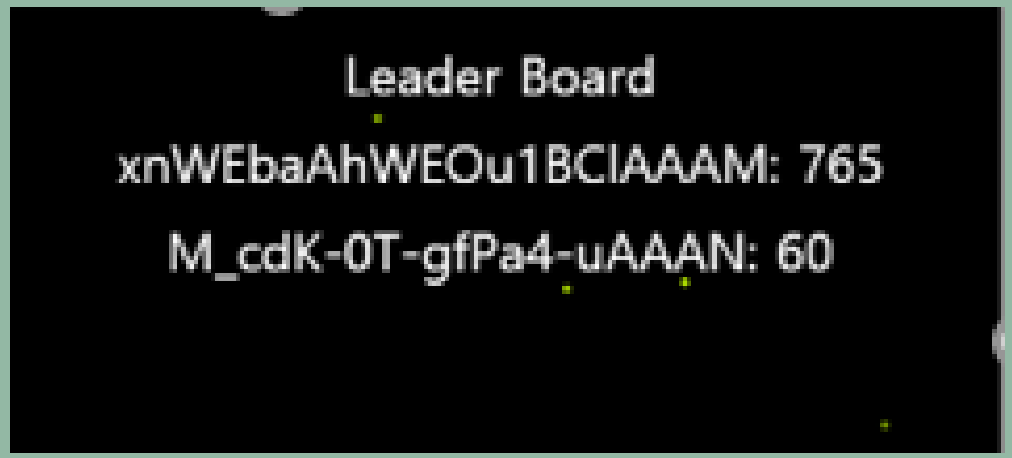
```
function preload(){
  shoot = loadSound('sound/laser.wav');
  bomb = loadSound('sound/bomb.mp3');
}
```

P5.js의 사운드 모듈을 사용하여 음향 효과 구현. 레이저 발사는 클라이언트가 스페이스바 버튼을 클릭할 때 레이저 소리가 나며

```
socket.on('explode', function(data) {
  fireworks.push(new Firework(data.x, data.y));
  bomb.play();
});
```

운석 파괴 소리는 웹 소켓으로 운석 파괴 이벤트를 감지해서 소리를 낸다.

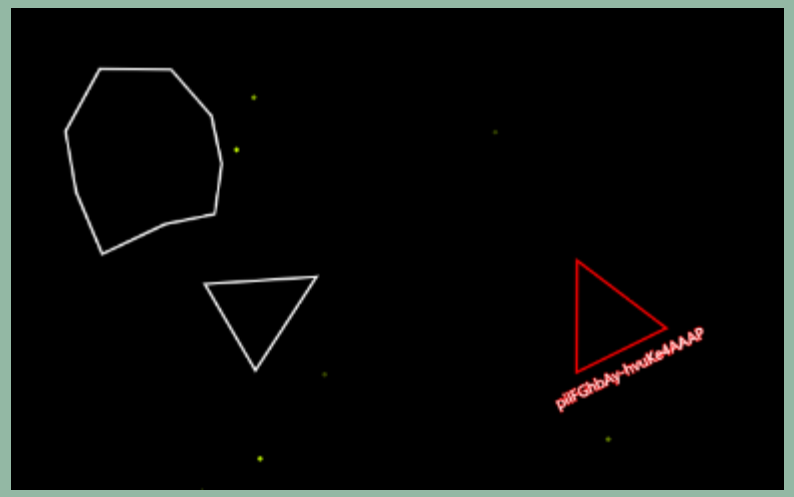
게임 화면

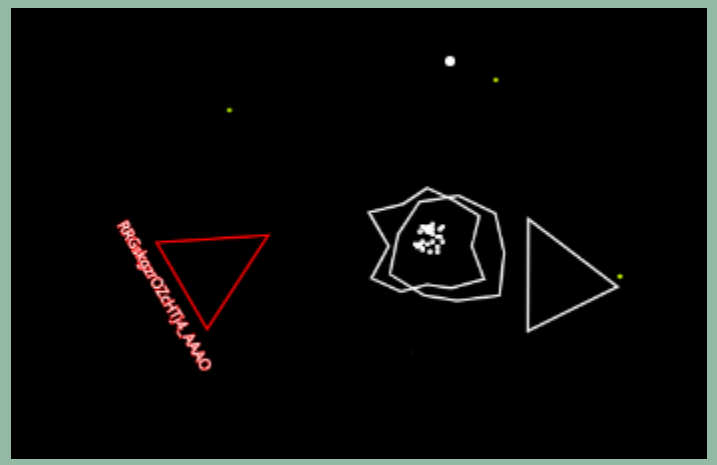


Server.js

웹 소켓으로 멀티플레이를 구현함

Leader board를 통해서 플레이어들은 모든 플레이어의 실시간 점수를 알 수 있음





다른 플레이어의 우주선은 빨간색으로 표시되며, 밑에 소켓ID정보가 나타난다

게임 화면

```
server_game_init();
setInterval(heartbeat, 30);
setInterval(asteroid_generator, 500);
```

```
    //console.log(ast_data);
    total_data[0] = ast_data;
    total_data[1] = ships;
    io.sockets.emit('heartbeat', total_data);
}
```

서버는 0.03초마다 모든 클라이언트에게 서버의 정보를 전송

운석도 0.5초마다 자동 생성되도록 함수를 등록

웹서버의 자원관리

우주선의 각 좌표는 클라이언트들이 관리한 후 서버에게 통지 (서버는 이를 모든 클라이언트들에게 에코백)

레이저는 클라이언트가 생성을 요청하고 서버가 이 조건에 맞게 생성해서 모든 클라이언트들에게 알려줌

운석은 서버가 직접 생성하며 모든 클라이언트들에게 알려줌

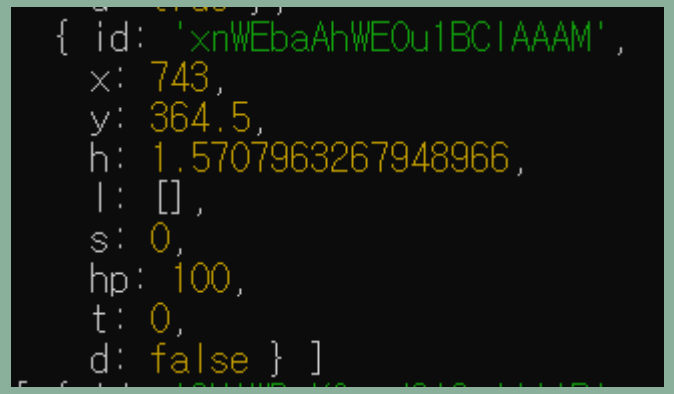
게임 화면

```
socket.on('score', function(data)
{
  score_list=data;
  console.log(score_list);
  my_id = socket.id;
  document.body.innerHTML = "<iframe src='result_page.html' width='1000px' height='1000px'></iframe>";
  socket.emit('disconnect', null);
  socket.close();
});
```

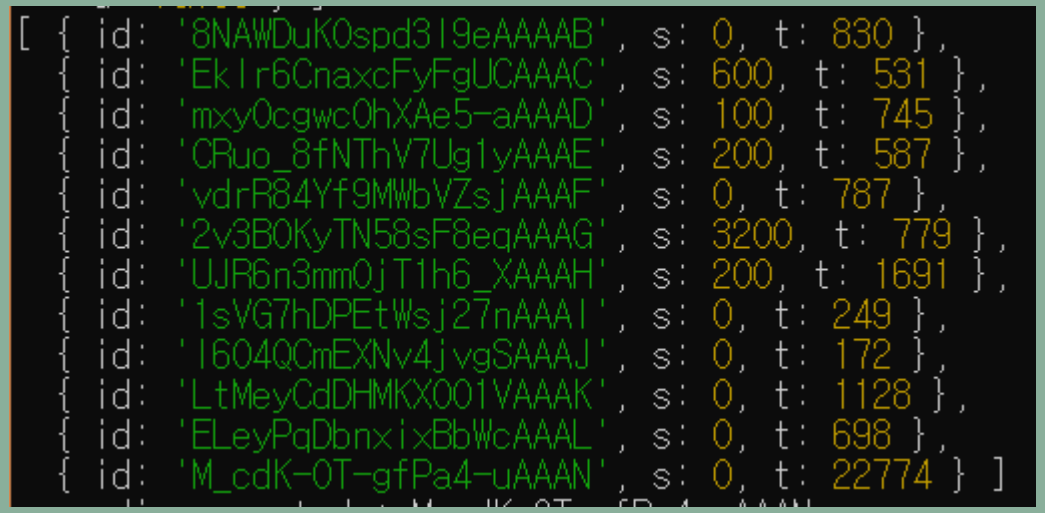
플레이어의 체력이 '0' 이 되면 서버에게 점수정보를 요청함. 서버는 서버를 연 이후로 누적된 점수 데이터들을 해당 클라이언트에게 보내줌

웹 소켓으로 점수 정보를 받으면 웹 소켓 연결을 끊고
<iframe> 태그로 결과 페이지를 보여줌

게임 화면

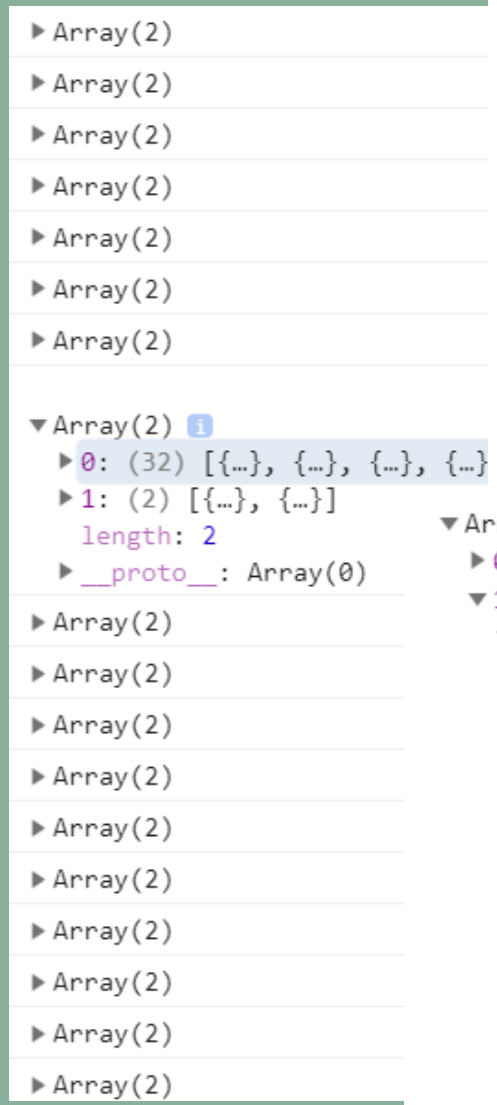


기본적으로 주고받는 데이터 형태이다 우주선 좌표, 각도, 점수, hp 등등의 정보가 있다00

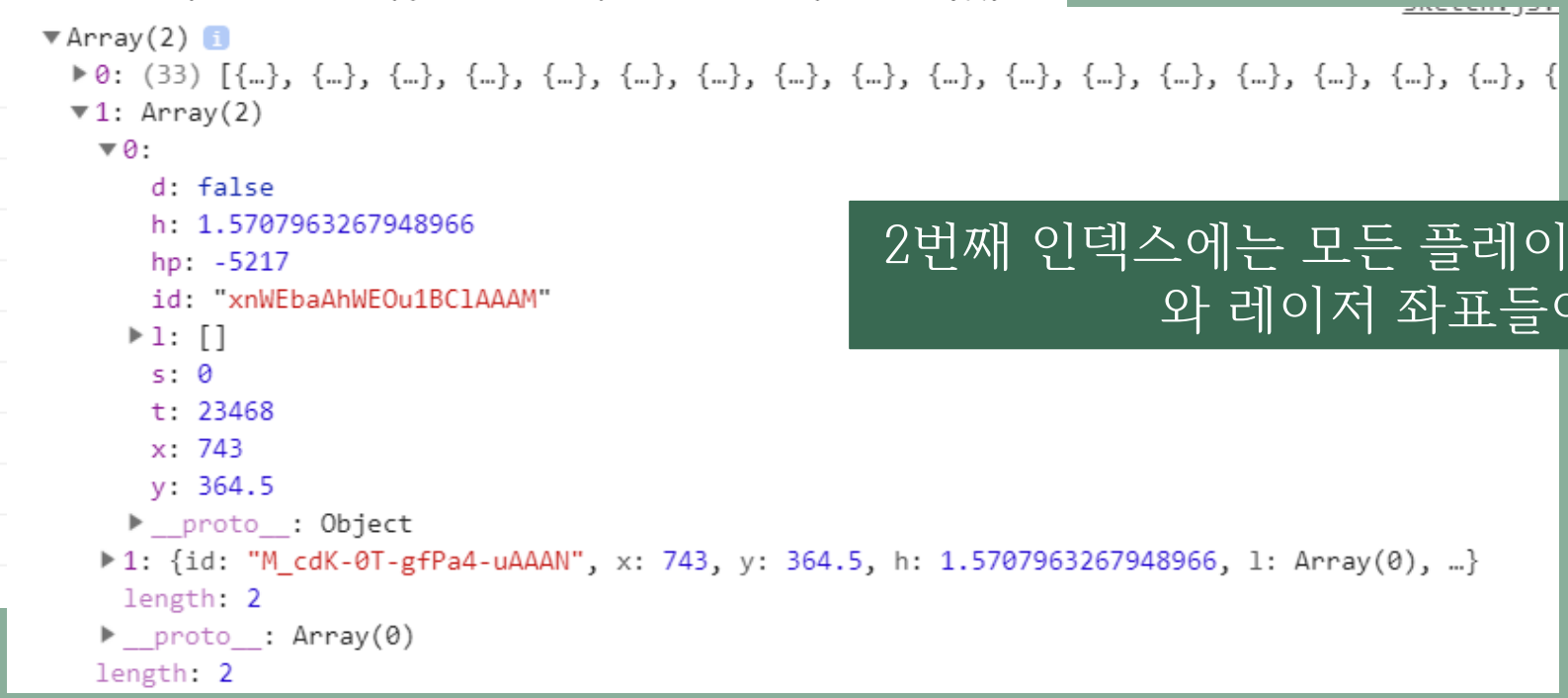


게임 종료시의 점수 정보이다
S는 운석 파괴시 마다 100씩 올라가며 (파괴점수)
T는 0.03초마다 1씩 올라간다 (생존점수)

게임 화면

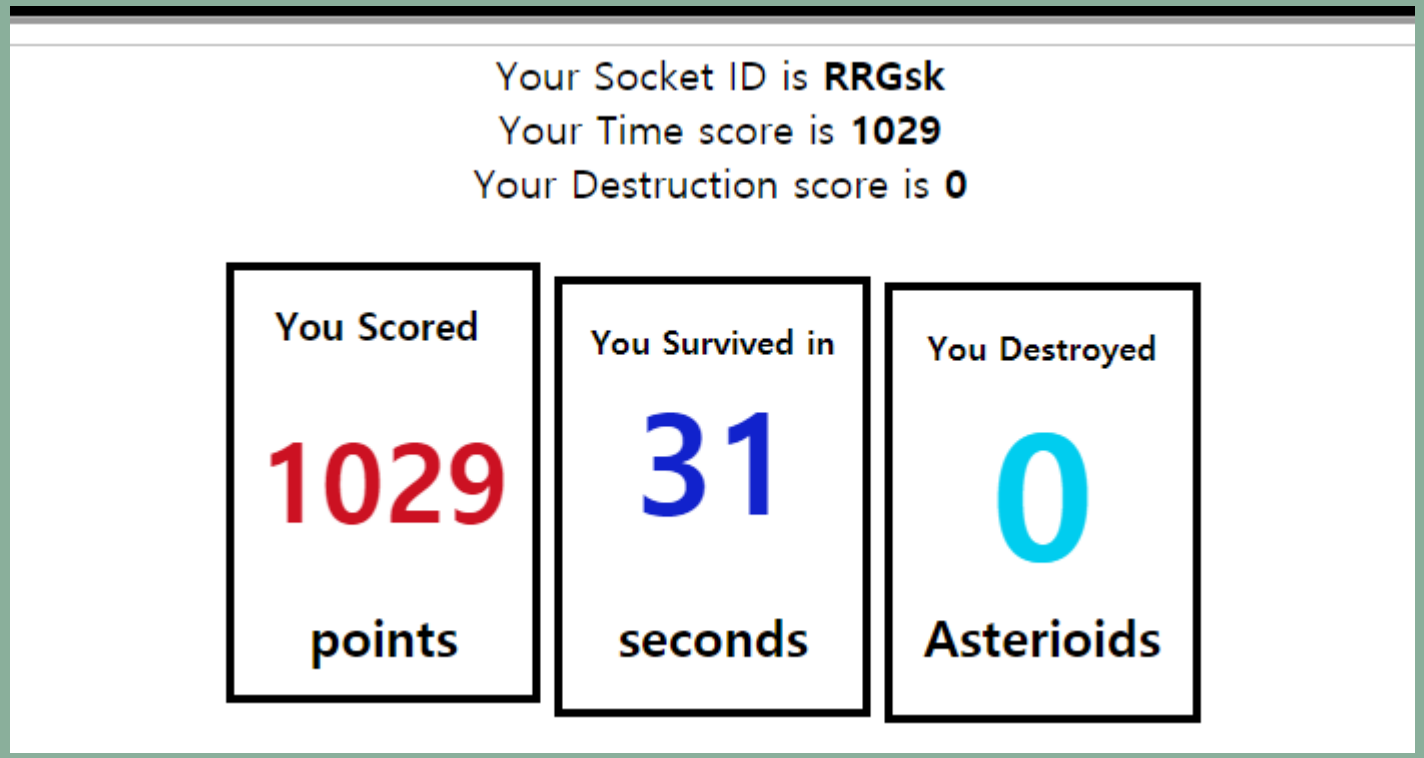


클라이언트 입장에서 받는 정보이다
받는 배열의 1번째 인덱스에는 운석들의 좌표와 offset의 배열이 있다.



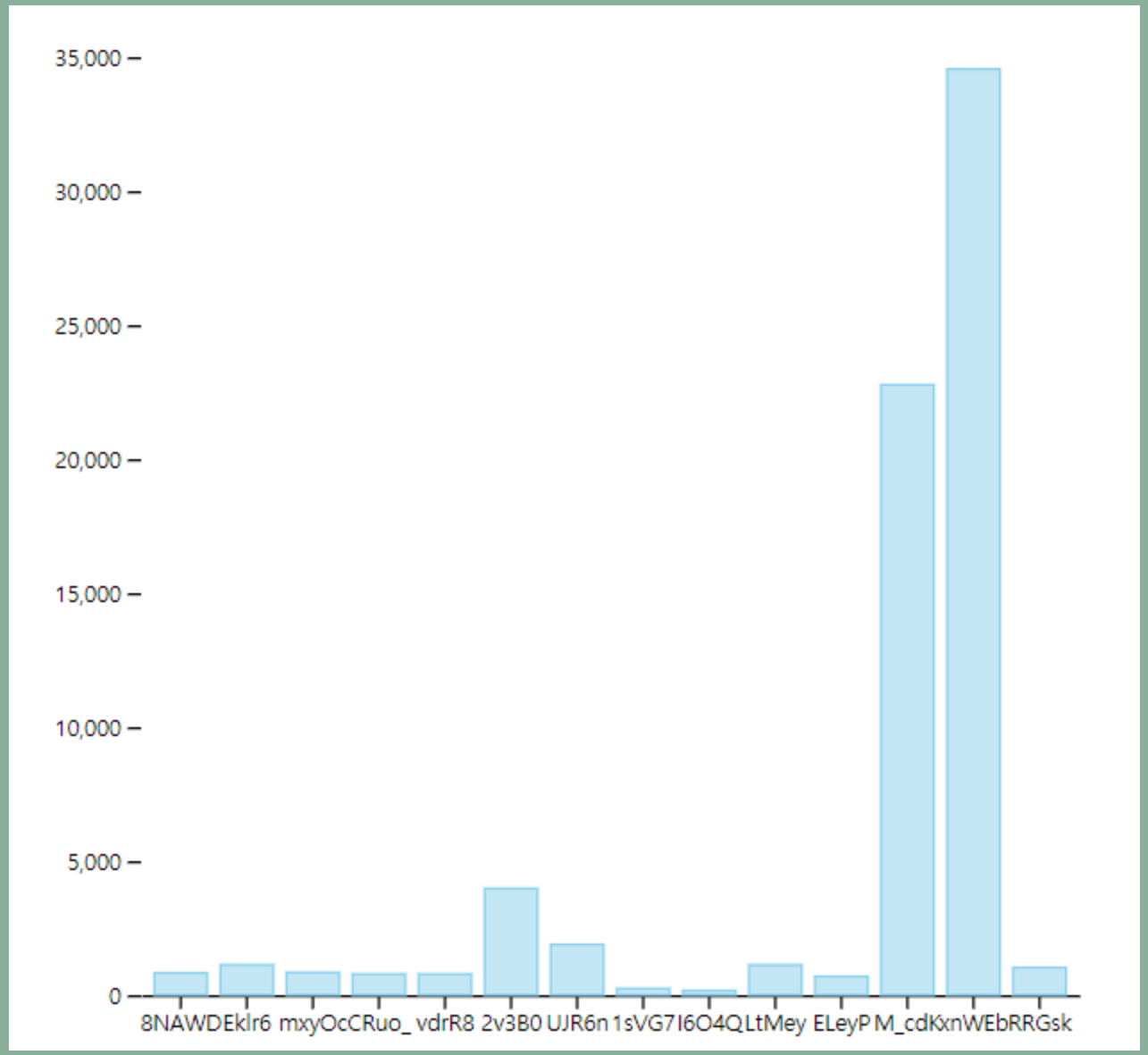
2번째 인덱스에는 모든 플레이어들의 우주선 정보와 레이저 좌표들이 있다.

게임 화면



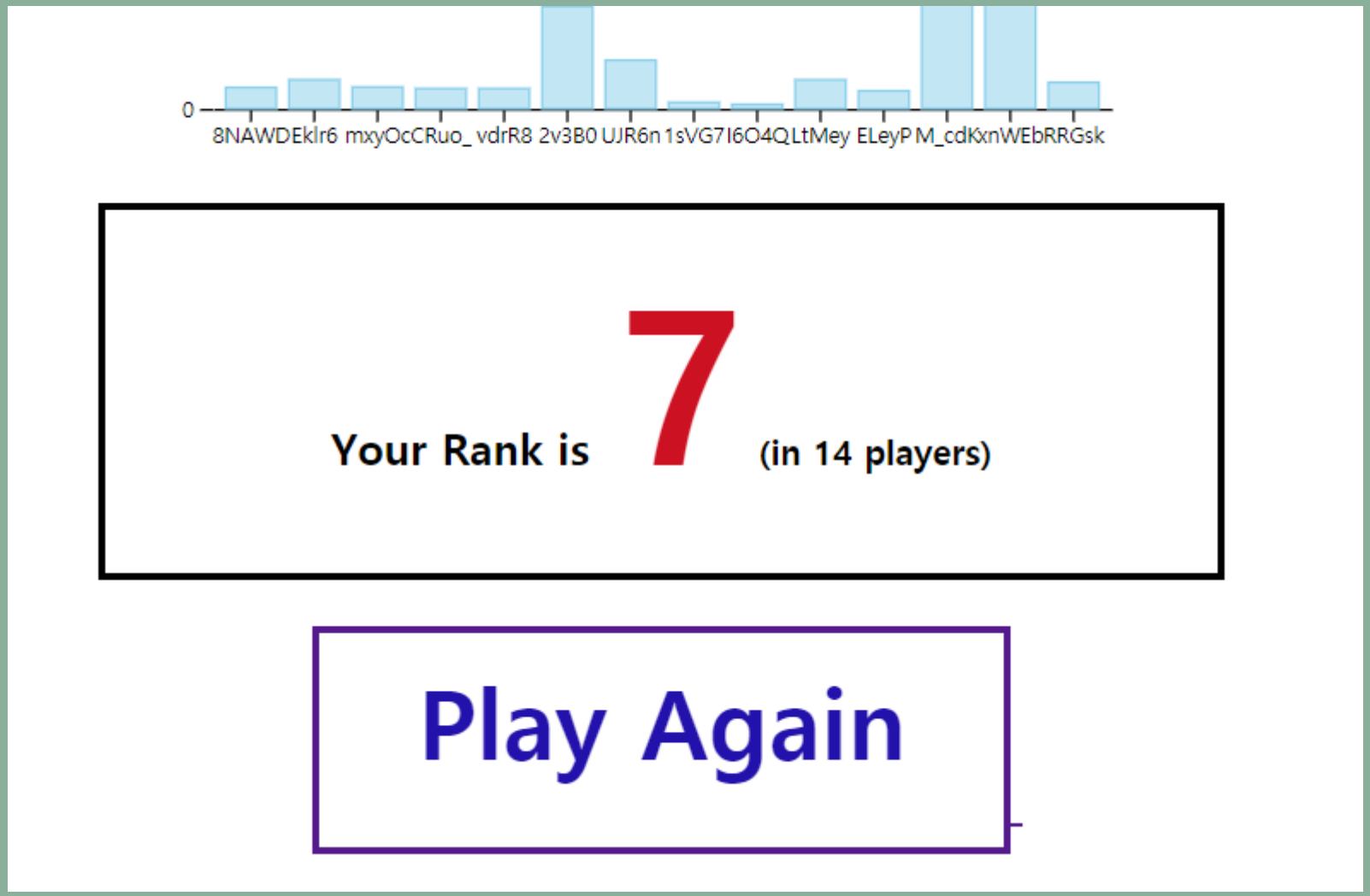
게임종료시의 결과화면이다. 웹 소켓으로 받은 결과 정보를 시각화 하였다.

게임 화면



D3.js 라이브러리를 이용하여
점수의 그래프를 시각화함

게임 화면



마지막으로 랭킹을 알려주고 다시 플레이할 수 있는 Play Again 버튼을 넣음

시연 영상

https://youtu.be/9GltodyZnU4

https://youtu.be/9GltodyZnU4
개인플레이, 멀티플레이 영상입니다.

감사합니다