



JavaScript 기반 HTML5 Platformer Game 개발 프로젝트



김진홍

Contents

1. 주제 및 기획의도
2. 사용언어 및 도구
3. 제작과정
4. 스토리보드
5. 객체관계도
6. 주요 코드
7. 느낀점
8. Github link 및 시연

1. 주제 및 기획의도



횱 스크롤 플랫폼머 러닝게임



Hero VS Vilain 구도 뒤집어보기



ECMAScript Core문법을 통한
Class와 상속 숙달

게임구현 목표

점프 한 동작으로 쉽게 즐길 수 있는 횱 스크롤 플랫폼머 캐주얼 러닝게임을 개발
충돌검사 기반 핵심기능 구현

Stage 1 개발 후 여유시간을 활용해 Stage 2로 업데이트

프로그래밍적 목표

게임에 등장하는 객체들을 Class와 상속을 통해 구현함으로써 개념을 숙달

Java에서 지원하는 Rect객체를 Javascript로 모방 구현해봄으로써

객체간 충돌을 체크하는 로직을 숙달하고, Java를 간접 학습

HTML5 / CSS3 / JavaScript를 사용함으로써 프론트엔드 개발 능력 향상

2. 사용언어 및 도구



Javascript (일부는 JQuery사용)

HTML5

CSS3

Editor : Editplus, SublimeText, Atom

ImgWork : Adobe Illustrator, Adobe Photoshop
MS PowerPoint

Version Control : SourceTree, Github

3. 제작과정

1차기간
- 18.12.07(금)
~18.12.13(목)

구분	Day1(금)	day2(토)	day3(일)	day4(월)	day5(화)	Day6(수)	day7(목)
기획							
이미지 준비							
기초마크업							
기능구현							
문서작성							

3. 제작과정

2차기간
- 19.06.03(월)
~19.06.09(일)

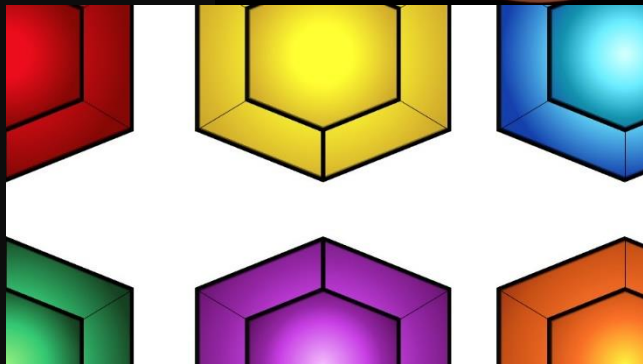
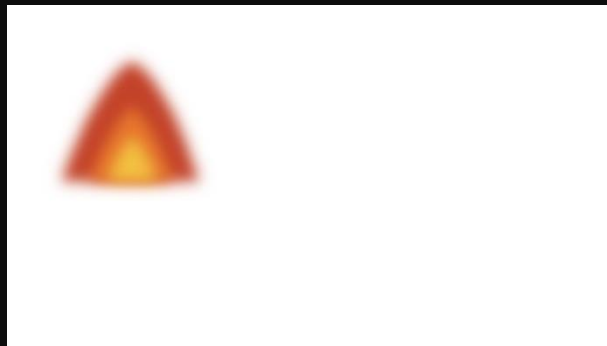
사유
- 데이터 소실로
인한 복구 및
재개발

추가마크업
- 랜딩페이지
- 스코어
- 메뉴

구분	Day1(월)	day2(화)	day3(수)	day4(목)	day5(금)	Day6(토)	day7(일)
데이터 복구 및 기획 수정							
추가마크업							
기능구현							
문서작성							

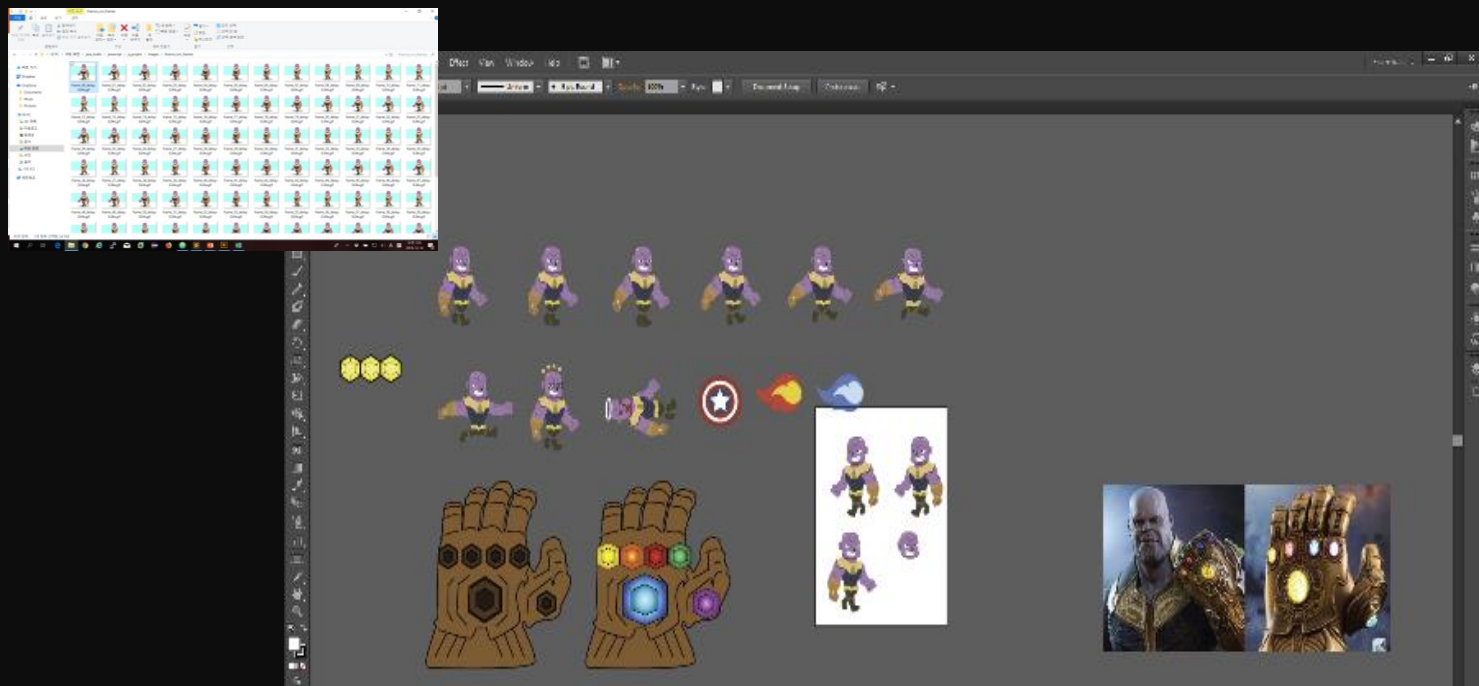
3. 제작과정

부산물



3. 제작과정

부산물




```
1 <!doctype html>
2 <html>
3   <head>
4     <title>Thanos Run</title>
5     <link rel="stylesheet" href="./css/style.css">
6     <script src="./js/lib/lib.js"></script>
7     <script src="./js/GameObject.js"></script>
8     <script src="./js/ObjectManager.js"></script>
9     <script src="./js/Sensor.js"></script>
10    <script src="./js/Hero.js"></script>
11    <script src="./js/Obstacle.js"></script>
12    <script src="./js/Stone.js"></script>
13    <script src="./js/Enemy.js"></script>
14    <script src="./js/Life.js"></script>
15    <script src="./js/Coin.js"></script>
16    <script src="./js/Floor.js"></script>
17    <script src="./js/Flame.js"></script>
18    <script type="text/javascript">
```

4. 스토리보드

Game Introduction



경제위기에 황폐해진 고향 행성
Titan의 재건을 위해 고민하던 Thanos!!
위기를 극복하고자 모험을 떠납니다.



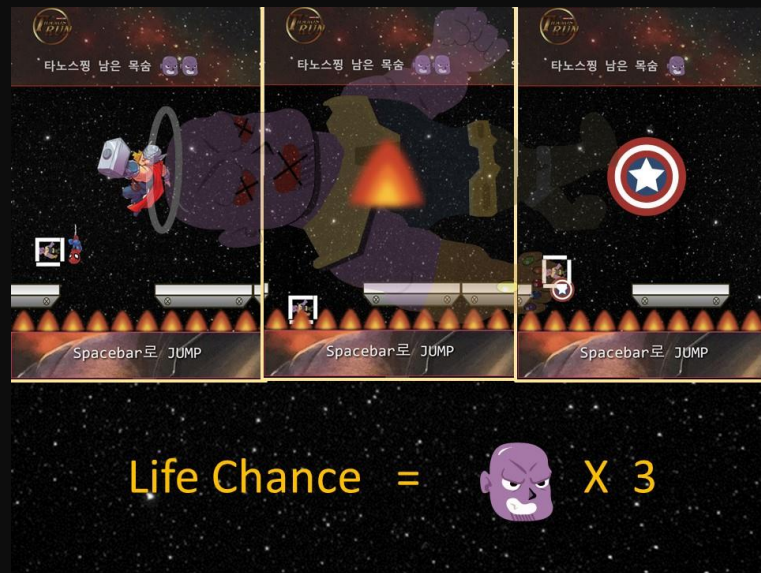
Thanos의 여행목표는 마주치는 행성마다 보유한
우주화폐 황금코인을 약탈하는 것이죠

4. 스토리보드

Game Introduction



Space Stone을 모아 finger snap을 할 때마다
황금코인이 Thanos에게로 날아듭니다.
놓치지 말고 수집하세요!!



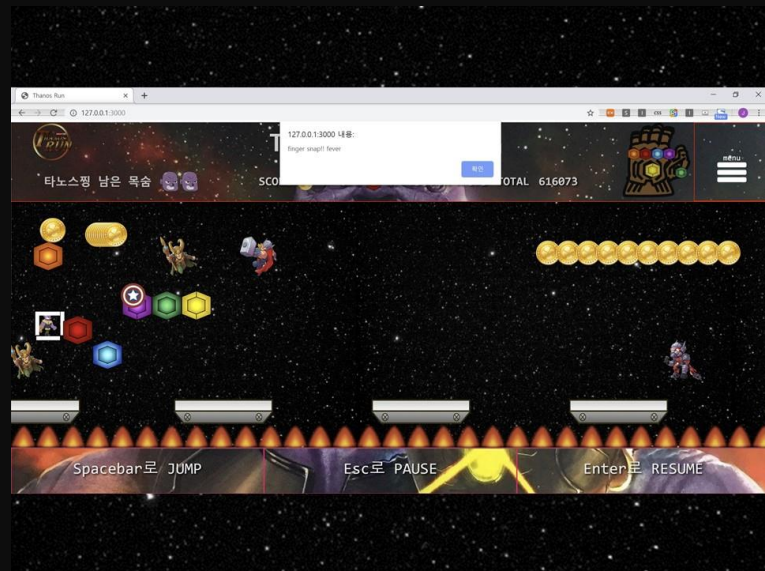
Enemy와 정면충돌하면 안돼요 ㅠㅠ
우주선 발판 밑 화염에 추락하면 안돼요 ㅠㅠ
captain의 방패에 맞으면 안돼요 ㅠㅠ
3번 밖에 없는 Thanos의 목숨을 소중히 여겨주세요

4. 스토리보드

Game Introduction



Enemy를 처치할 방법이 없냐구요?
spacebar를 눌러 점프한 후 밟아서 처치하세요

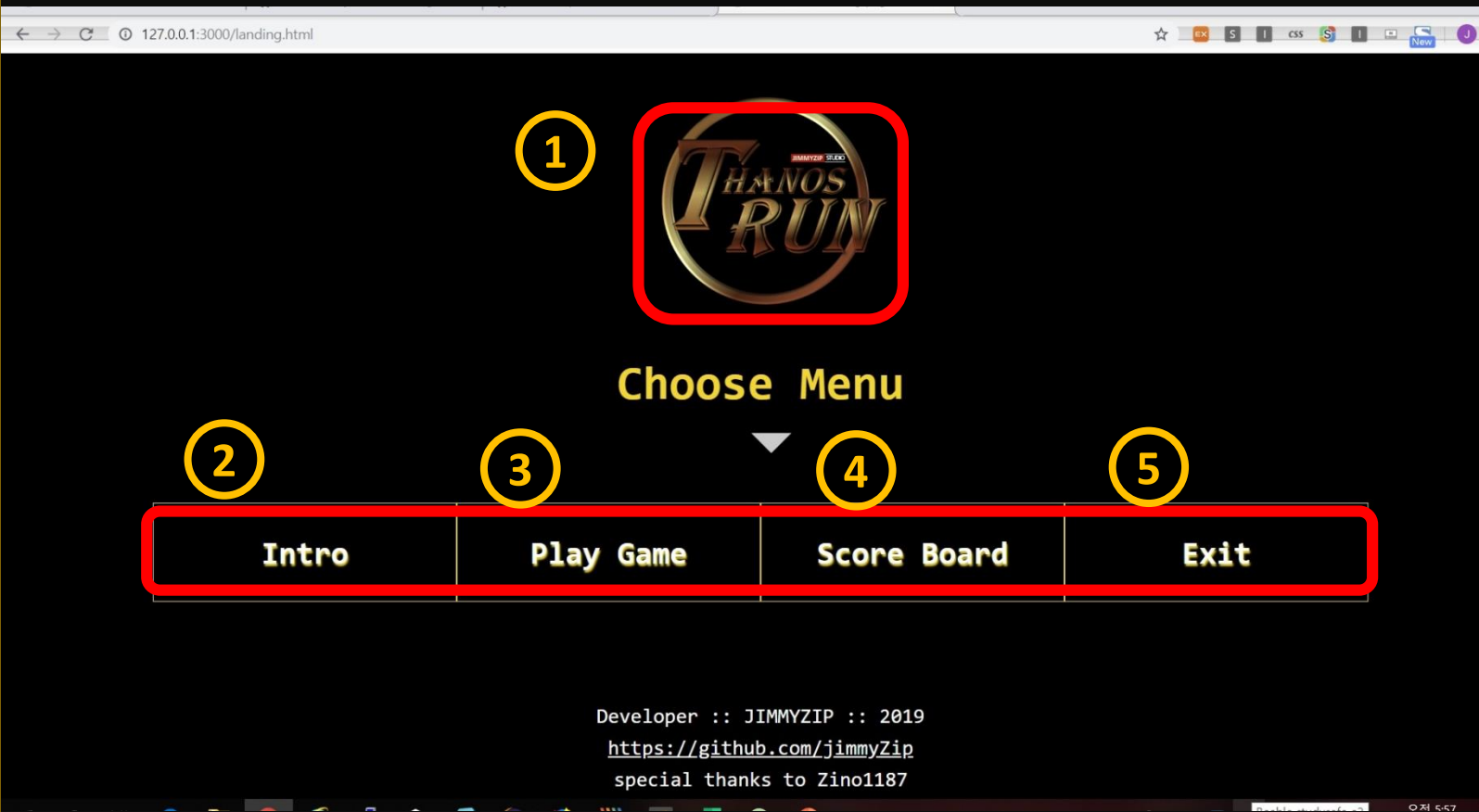


단, finger snap을 하면 Fever Mode가 발동되어
돈 앞에 보이는게 없는 무적상태가 7초간 유지됩니다.

4. 스토리보드

화면캡처 및 설명 _ landing page

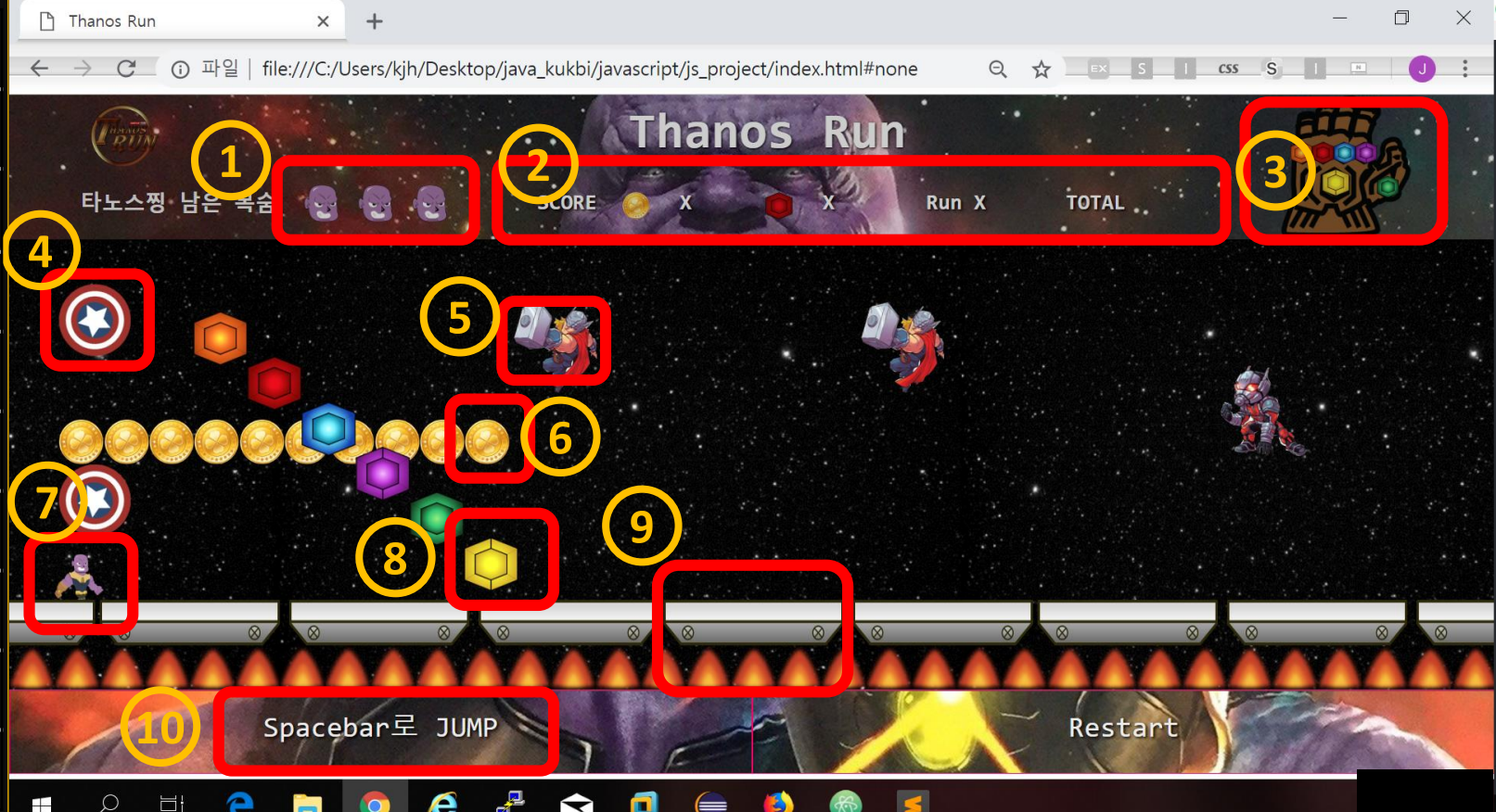
1	로고 링크
2	게임소개버튼 (모달)
3	게임시작버튼
4	점수판 버튼 (모달)
5	종료버튼



4. 스토리보드

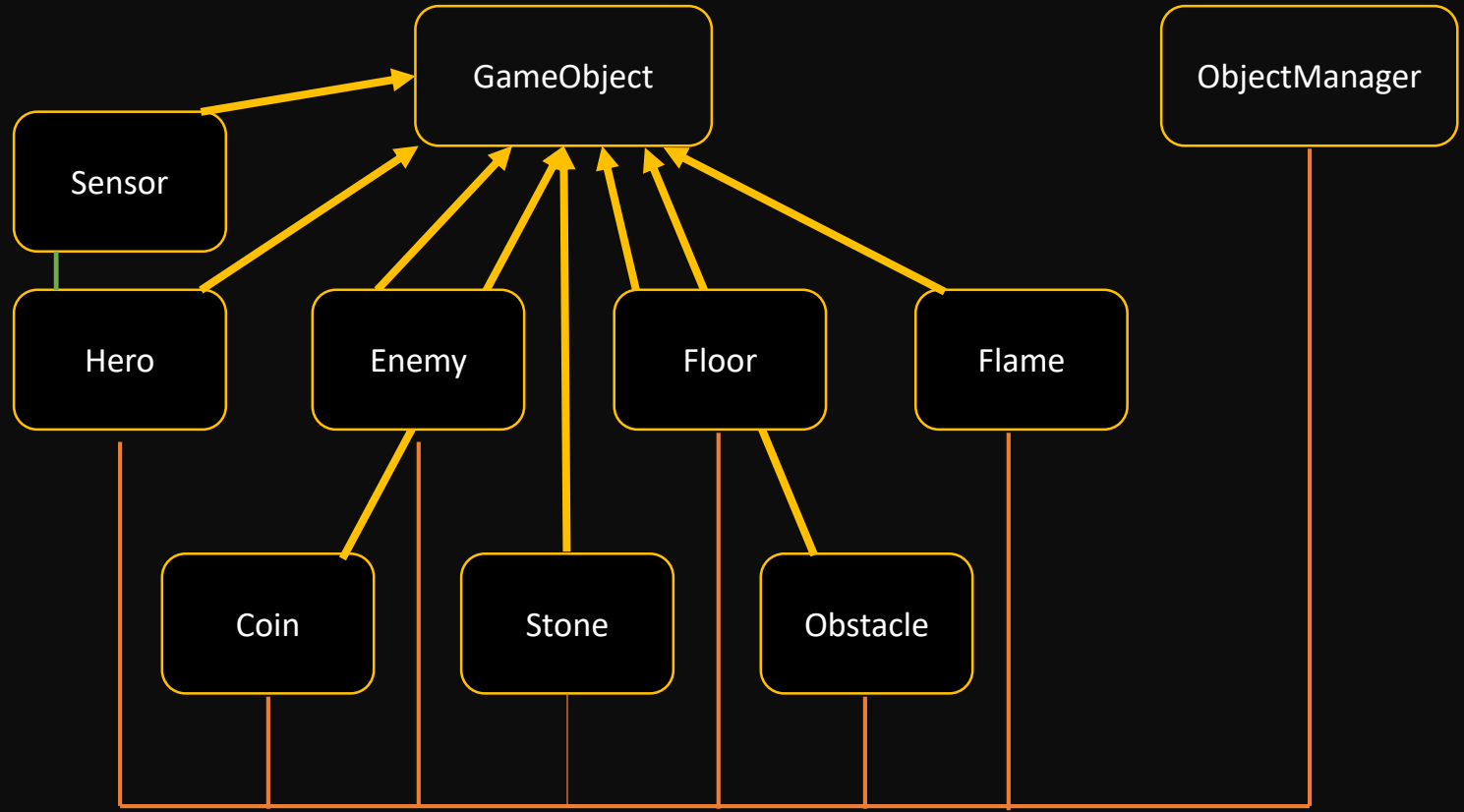
화면캡처 및 설명 _ 실제 게이밍 페이지

1	목숨
2	스코어
3	필살기 게이지
4	장애물(死)
5	빌런
6	코인
7	주인공
8	스톤(6종)
9	발판(플랫폼)
10	게임조작



5. 객체관계도

- Game에 등장하는 객체들은 GameObject를 상속받아 생성
- 생성된 객체들은 ObjectManager의 객체관리배열 objectArray[]에 담아서 관리
addObject(obj);
removeObject(obj);
- 객체들의 동작은 GameObject로 부터 상속받은
tick() : 한 단위 동작
render() : 동작 구현
메서드를 일괄 호출



6. 주요 코드

게임에 등장하는 Object 상속관계 및 생성 예시

// 게임에 등장하는 모든 객체의 상위클래스

```
class GameObject{
    constructor(typeId,container,x,y,width,height,velX,velY,targetX,targetY,bg,src){
        this.typeId=typeId;
        this.container=container;
        this.x=x;
        this.y=y;
        this.width=width;
        this.height=height;
        this.velX=velX;
        this.velY=velY;
        this.targetX=targetX;
        this.targetY=targetY;
        this.bg=bg;
        this.src=src;
        this.a=0.1;
    }
}
```

// hero 우리의 주인공 th 타노스를 정의한다.

```
class Hero extends GameObject{
    constructor(type,container,x,y,width,height,velX,velY,targetX,targetY,bg,src){
        super(type,container,x,y,width,height,velX,velY,targetX,targetY,bg,src);
    }
}
```

// 히어로 타노스 등장

```
function createHero(){
    hero = new Hero("HERO",stage,50,345,50,50,0,0,0,0,"","./images/thanos_0.png");
    objectManager.addObject(hero); // objectManager의 ObjectArray에 등록
}
```


6. 주요 코드

생성된 instance를 관리하는 ObjectManager클래스

```
class ObjectManager{
  constructor(){
    this.objectArray=[]; //게임에 등장할 모든 종류의 오브젝트들을 모아놓은 배열
  }
  //의뢰받은 객체를 objectArray라는 명단에 넣기
  addObject(obj){
    this.objectArray.push(obj);
  }
  //의뢰받은 객체를 objectArray에서 제거하기
  removeObject(obj){
    //console.log(obj);
    //console.log("지웠다!!");
    stage.removeChild(obj.div); //화면에서 지우기
    this.objectArray.splice(this.objectArray.indexOf(obj),1);
  }
}
```

ObjectManager클래스는
게임 Stage에 생성되는
객체들이

- 1) 화면에서 제거
- 2) 객체 배열에서 제거

되는 작업을 위임받아
일괄처리합니다.

addObject(obj);
removeObject(obj);

6. 주요 코드

주인공 및 게임 조작관련

```
/*-----  
게임 메인캐릭터 및 기타 동작제어( 점프, 일시정지, 재시작)  
spacebar :: jump수행  
esc | enter :: gameLoop 정지/시키는 fLag 변수 반전으로 일시정지 | 재개  
-----*/  
function control(){  
  var key=event.keyCode;  
  switch(key){  
    case 32://spacebar  
      event.preventDefault();  
      if(hero.jumping==false) hero.jump();  
      hero.jumping==true;  
      break;  
    case 27:pauseGame(); break;//esc  
    case 13:resumeGame(); break;//enter  
  }  
}
```

event.keyCode 값으로
조건 분류

32 : spacebar : jump

27 : esc : 게임 일시정지

13 : enter : 게임 재개

6. 주요 코드

객체간 충돌검사

1) Hero에 sensor 역할 element 부착

예시 flow) Hero에 부착된 bottom 방향 sensor가 floor 객체와 닿았으니 착지 및 걷는 효과를 적용한다.

```
//충돌검사를 위한 센서부착
this.sensorArray=[];
this.leftSensor=new Sensor("SENSOR",container,getSensorSize("LEFT",this.x,this.y,this.width,this.height),0,0,0,0,"#fff","");
this.rightSensor=new Sensor("SENSOR",container,getSensorSize("RIGHT",this.x,this.y,this.width,this.height),0,0,0,0,"#fff","");
this.topSensor=new Sensor("SENSOR",container,getSensorSize("TOP",this.x,this.y,this.width,this.height),0,0,0,0,"#fff","");
this.bottomSensor=new Sensor("SENSOR",container,getSensorSize("BOTTOM",this.x,this.y,this.width,this.height),0,0,0,0,"#fff","");
this.sensorArray.push(this.leftSensor);
this.sensorArray.push(this.rightSensor);
this.sensorArray.push(this.topSensor);
this.sensorArray.push(this.bottomSensor);
```

//센서를 정의

```
class Sensor extends GameObject{
    constructor(typeId,container,json,velX,velY,targetX,targetY){
        //x,y,width,height를 집단형 json으로 적는다.
        super(typeId,container,json.x,json.y,json.width,json.height);

        //왼님만 따라가자
        tick(x,y){
            this.x=x;
            this.y=y;
        }
    }
}
```

```
/*-----
어떤 대상이든, 방향만 넘겨주면 알아서 크기 정보를 생성한다
side : left, right, up, down
-----*/
function getSensorSize(side , x, y, width, height ){
    var border=8; //막대들의 두께
    var colTypePadding=2//세로형 막대 들어쓰기 퍼센트 %

    var rowTypePadding=2//가로형 막대 들어쓰기 퍼센트 %
    var rowTypeWidth=96 //가로형 막대 너비 퍼센트 %

    var json={};

    json["type"]=side;
    //좌측일 경우
    if(side == "LEFT"){ //세로형
        json["x"]=x-1; //2픽셀 바깥으로
        json["y"]=y+(height*(colTypePadding/100)); //2% 밑으로
        json["width"]=border;
```

- 주인공 element를 감싸는

4방향 센서를 생성

- Sensor객체는 JAVA에서 지원하는 Rect객체를 모방하여
JavaScript로 구현하였습니다.

6. 주요 코드

객체간 충돌검사

2) 위치값과 이동에 따른 충돌여부 판단 조건문 수행

```
/*대상별 충돌체크*/  
for(var i=0;i<objectManager.objectArray.length;i++){  
    var obj=objectManager.objectArray[i];  
    //바닥 :: 바탐센서가 플랫폼에 닿지 않으면 추락해야한다.  
    if(obj.typeId=="FLOOR"){  
        for(var a=0;a<this.sensorArray.length;a++){  
            var s = this.sensorArray[a];  
            if(hitTest(obj,s,this.velX,this.velY)){//충돌했다면  
                hitCount[a]++;  
                floorCount++;  
                bottomTarget=obj;
```

4방향 센서가
반복문 내에서
특정 대상객체와
닿았는지를 판단하고

결과 boolean값에 따라
방향별 충돌횟수,
충돌 대상을 변수에
저장합니다.

6. 주요 코드

객체간 충돌검사 3) 해당되는 작업처리

```
//바닥충돌
if(floorCount>0){
    //console.log("바닥과 닿았다. 바닥은"+bottomTarget.typeId);
    this.jumping=false;
    this.y=bottomTarget.y-this.height;
    for(var a=0;a<this.sensorArray.length-1;a++){
        this.sensorArray[a].y=this.y;
    }
    this.sensorArray[3].y=this.y+this.height;
}
```

충돌검사 결과에 따라
값이 변화된 변수를
조건비교해서

참인 경우에만

해당 동작을 수행합니다.

6. 주요 코드

게임 엔진 역할 함수, timer 함수, flag 역할 변수

```
//index.html 진입 .5초 후 게임엔진이 가동되도록  
gl = setTimeout("gameLoop()",500);  
//gameLoop();
```

```
var gameFlag = true; //게임 일시정지 및 재시작 제어용 플래그 변수
```

```
//게임엔진, Loop  
function gameLoop(){  
    if(gameFlag==true){  
        bgEffect();  
        objectManager.tick();  
        objectManager.render();  
        setTimeout("gameLoop()",20);  
    }  
}
```

```
//게임 멈춤처리 pauseGame  
function pauseGame(){  
    gameFlag=false;  
    //console.log("현재 gameFlag :: ",gameFlag);  
}  
//게임시작 또는 재개  
function resumeGame(){  
    if(lifeCount>0){  
        gameFlag=true;  
        console.log("현재 gameFlag :: ",gameFlag);  
        gameLoop();  
    }  
}
```

Index.html

>> onLoad 이벤트 핸들러에 의해 init()메서드 호출

>> init()메서드 내에서 gameLoop()메서드를 호출

>> gameLoop()내의 재귀적 호출을 통해 무한반복효과

>> 무한반복을 제어하도록 flag 변수 사용

7. 느낀점

Backup 및 형상관리의 중요성

약 3주간 javascript를 활용하는 수업의 결과물로

Thanos_run 프로젝트를 진행한 후

Java 개발 수업으로 넘어가며 Backup한 프로젝트 결과물이 상당부분 소실되었습니다.
이를 복구하는 과정에서 Backup의 수단, 주기, 버전관리의 필요성과 중요성을
뼈저리게 느꼈습니다.

보완 노력

1. 프로젝트 결과물(중간, 최종)은 SourceTree 툴을 사용해 Github에 저장
2. 특별한 경우가 아니라면 하드웨어적 저장수단 사용 지양 및 클라우드 이용
(물리적 경로 바이러스, 악성코드 차단)
3. Github 업로드 시 불필요한 부분을 체크하여 stage에서 내림
4. 단계별 구현목표를 분할하는 습관

을 가지도록 꾸준히 노력하겠습니다.

8. 개선사항

Jump이후 바닥착지

Hero 객체가 점프동작 이후 Floor와 충돌하면 착지상태를 유지하는데 있어 중력값이 누적되는 상태이다보니 시각적으로 바닥 밑으로 점점 내려감
충돌여부가 true인 동안 강제로 중력을 제거하면 이후에 부드러운 점프가 수행되지 않는 점에 대한 개선 필요

Thread 및 메모리

GameObject클래스를 상속받아 생성되는 Object들이 화면 게임 화면 밖으로 나가면 객체관리 Array와 화면에서 제거한다고 하더라도 게임이 진행되는 동안 반복적으로 생성되어 특정 시간이 지나면 전체 퍼포먼스가 저하되는 현상 개선이 필요

게임 엔진 역할을 하는 GameLoop함수를 반복적으로 수행하는 와중에 객체간 충돌검사, 충돌 결과에 따른 이벤트 처리를 Single Thread가 감당하는 상황이므로 Version upgrade 시에는 Web Worker를 도입해서 Multi Thread를 구현해볼 예정



Thanos Run



타노스징 남은 목숨



SCORE



X



X

Run X

TOTAL



Spacebar로 JUMP

Restart





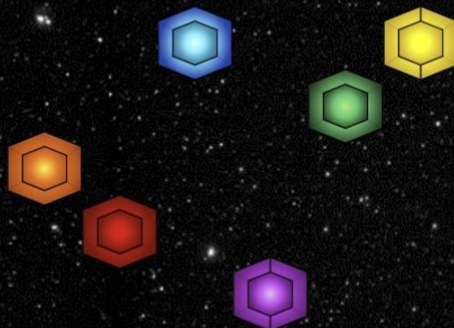
Thanos Run



menu

타노스핑 남은 목숨   

SCORE  X 0  X 0 Run X 49 TOTAL 49



Spacebar로 JUMP

Esc로 PAUSE

Enter로 RESUME

9. 시연

Github Link

<https://github.com/jimmyZip/JSPlatformerGame>

Hosting link
(변경가능성)

https://cafe24.com/jimmyzip/js_project

감사합니다.