# 휴먼 컴퓨터 인터페이스

## 과제 #2. 블록 조립 계산기

**2018203062**

**김수빈**

2019.04.13.

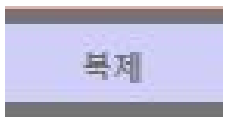
광운대학교 소프트웨어학부

# 개요

## 기능적 요구조건에 대한 구현 완성도요약

| 기능적요구조건 | | | |
|---|---|---|---|
| 기능 | 종류 | | 구현여부 |
| 수식입력 | 정수, 실수, 복소수 | + - / * % ^<br>== != > < <= >=<br>i | o |
| | 벡터, 행렬 | dot , cross | o |
| | 상수 | pi, e | o |
| | 함수 | sin, cos, tan<br>exp, log, sqrt | o |
| 결과지원 | 올바른입력 | 수식결과값 | o |
| | 잘못된 입력 | 오류메시지 | o |
| 변수, 함수 정의 | 함수 | 개수제한x | o |
| | 변수 | 개수제한x | o |
| 추가기능 | 각도계산 | deg 입력 가능 | o |
| | 추가 함수기능 | arc, inv, h, abs | o |
| | | ! | o |

# 인터페이스 요구조건

| 인터페이스 요구조건 | | | |
|---|---|---|---|
| 기능 | | 상호작용 | 구현여부 |
| 생성 | - | 키보드 입력 | o |
| 이동 | - | 마우스 드래그 | o |
| 조립 | - | 충돌검사후 조립 | o |
| 실행 |  | 실행버튼 클릭 | o |
| 분해 |  | 분해버튼 클릭 | o |
| 복제 |  | 복제버튼 클릭 | o |
| 소멸 |  | 삭제버튼 클릭<br>+전체삭제버튼 클릭시<br>모든 블럭 삭제 | o |

# 오픈소스 라이브러리 의존성 요약

## 사용 오픈소스

```
'https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js' type
"https://cdnjs.cloudflare.com/ajax/libs/fabric.js/2.4.6/fabric.min.js"
"./etc/js/math.min.js" type="text/javascript"></script>
```

jquery.js fabric.js math.js

# 본문

## 사용자 인터페이스의 구성 요소 및 사용 방법


**Canvas :** 모든 블록들이 표시되는 창
**Error Message :** 여러 오류메세지를 표시해준다.
**Menu :** 상호작용을 담당해주는 메뉴 창
**Quick button :** 조립하기 귀찮고 많이 쓰이는 수식들을 모아 놓았다.

**생성 :** 키보드 입력
**이동 :** 마우스 드래그
**조립 :** 마우스 - 블럭간 충돌
**실행 :** 수식블럭 조립 후 선택활성화 상태에서 메뉴의 실행버튼 클릭시 계산
**분해 :** 조립된 블록을 선택활성화 상태에서 분해버튼 클릭시 분해
**복제 :** 블록 선택활성화 상태에서 복제버튼 클릭시 복제
**소멸 :** 선택 활성화 블록  삭제버튼 클릭시 소멸됨.
+전체삭제 버튼 클릭시 CANVAS 내에 있는 모든 블록 삭제

# 특징적인 상호작용 방식들에 대한 세부 구현 방법

## 조립 : 마우스 - 블럭간 충돌

```
MathApp.handleMouseUp = function (window_p) {
    let canvas_p = MathApp.transformToCanvasCoords(window_p);
    let block = MathApp.findBlockOn(canvas_p);
    if (block != null) { //만약 선택된 블럭이 있으면
        MathApp.Collision();//충돌검사
        //FIXME:
    }
    if (MathApp.is_mouse_dragging) {
        MathApp.is_mouse_dragging = false;
        MathApp.mouse_drag_prev = { x: 0, y: 0 };
    }
    MathApp.canvas.requestRenderAll();
}
```

마우스 클릭을 멈추었을 때, 블록간 충돌검사하는 함수를 호출한다.

```
MathApp.Collision = function () {
    //내가 선택한 블럭이 다른 블럭들과의 거리가 설정값 이상일 때, 두 블럭을 합친다
    if(MathApp.selected_block!=null){
        let select = MathApp.selected_block;

        for (let i = 0; i < this.blocks.length; i++) {
            let block = this.blocks[i];
            let disX = Math.abs(select.position.x - block.position.x);//두개의 차이가 blocksize/2보다 작을때
            let disY = Math.abs(select.position.y - block.position.y);
            let rangeX = (select.size.width / 2) + (block.size.width / 2);
            if ((disX == 0) && (disY == 0)) continue;//자기 자신은 검사에서 제외해줘야 함!
            else if ((disX <= (rangeX + errorRange)) && (disY <= (block.size.height + errorRange))) {
                MathApp.canvas.requestRenderAll();
                MathApp.Assemble(i); //만약 거리 이내에 블럭이 있다면, 합친다
                // console.log("충돌!");
            }
            else {
                // console.log("안 충돌!");
            }
        }
        // console.log("==========================");
    }
}
```

충돌을 검사하는 함수이다. 모든 블록들과의 거리를 계산해서, 범위내에 있는 블록이라면 합친다!

```
MathApp.Assemble = function (idx) {
    // I want merge selected_block && blocks[idx];
    if(MathApp.selected_block!=null){
        let s = MathApp.selected_block;
        let b = this.blocks[idx];
        let size = {
            width: s.size.width + b.size.width,
            height: s.size.height
        };
        let position = {//block 위치 그대로
            x: b.position.x,
            y: b.position.y
        };
        let new_symbol = new MathApp.Symbol(position, size, b.name + s.name);
        b.destroy();
        s.destroy();//합친 블럭을 생성하고, 기존 두개의 블럭을 없앤다.
        MathApp.selected_block = null;
```

합치는 코드이다. 선택된 코드가 있다면,
너비는 두 블록을 합친 크기이고, 높이는 그대로.
위치는 기존에 있던 블록위치를 기준으로 설정한다.
그렇게 충돌된 블록들간의 이름을 합쳐서 새로운 블록을 만들고
충돌에 사용된 두 블록을 없앤다.

**분해 :** 조립된 블록을 선택활성화 상태에서 분해버튼 클릭시 분해

```
MathApp.Disassemble = function () {
    let select = MathApp.selected_block;
    for (let i = 0; i < select.name.length; i++) {
        let plus = 50 * i;
        let size = {
            width: SYMBOL_WIDTH,
            height: SYMBOL_HEIGHT
        };
        let position = {
            x: select.position.x - select.size.width / 2 + 5 + plus,
            y: select.position.y
        };
        let new_symbol = new MathApp.Symbol(position, size, select.name[i]);
    }
    MathApp.selected_block.destroy();//맨처음 블럭은 없애준다.
}
```

선택된 블록의 이름을 순회하면서 다시 하나하나 블록을 만들어 준다.
plus변수로 블록의 생성 위치를 계속해서 바꿔주면서 자연스러운 위치에 분해된 블록들이 위치할 수 있도록 작성하였다.
분해에 사용된 블록은 없애준다.

## 생성 : 키보드 입력 & 조립&실행시 생성되는 블록

(키보드 입력시 생성되는 코드 기본제공)

```
MathApp.Symbol = function (position, size, name) {
    // console.log("symbol 함수 실행");
    $('.error').text("");
    console.log("name : ", name);
```

블록이 생성되면, Error message는 깔끔하게 초기화 시켜준다.
생성되는 이름을 검사하기 위해 console에 출력해보면서 코딩하였다.

```
    else {
        $('.error').text("Press another key");
        console.log("press another key");
    }
```

등록된 키 이외에 키가 눌리면 error 메시지 출력

```
    if (name in MathApp.symbol_paths)
    {
```

만약 이름에 해당하는 이미지가 있다면 하나씩 생성을 하는데,
만약 그렇지 않다면 else로 넘어간다

```
let background = new fabric.Rect({
    left: position.x - size.width / 2,
    top: position.y - size.height / 2,
    width: size.width,
    height: size.height,
    fill: "rgba(255,255,255,1)",
    stroke: "rgba(0,0,0,0)",
    selectable: false
});
MathApp.canvas.add(background);
block.visual_items.push(background);
```

else가 실행되면 , 블록에 들어갈 background를 생성한 후에

```
for (let i = 0; i < name.length; i++) {//name index 별로... 하나하나 모
    console.log(name[i]);
    let path = MathApp.symbol_paths[name[i]] + ".jpg";
    fabric.Image.fromURL(path, function (img) {
        // (1) Image
        img.scaleToWidth(50);
        img.scaleToHeight(50);
        let img_w = img.getScaledWidth();
        let img_h = img.getScaledHeight();
        let plus = 50 * i;

        img.set({
            left: position.x - size.width / 2 + errorRange + plus,
            top: position.y - (img_h / 2),
            selectable: false
        });

        // (2) Boundary
        let boundary = new fabric.Rect({
            left: position.x - size.width / 2,
            top: position.y - size.height / 2,
            width: size.width,
            height: size.height + 2,
            fill: "rgba(0,0,0,0)",
            stroke: "gray",
            strokeWidth: 6,
            selectable: false
        });

        MathApp.canvas.add(img);
        block.visual_items.push(img);
        MathApp.canvas.add(boundary);
        block.visual_items.push(boundary);
    })
```

이름index 하나하나 순회하면서 그에 해당하는 이미지를 하나하나 불러들여오고, boundary도 최종 크기에 맞게 삽입해준다.

**실행 :** 수식블럭 조립 후 선택활성화 상태에서 메뉴의 실행버튼 클릭시 계산

```
MathApp.execute = function (e) {
    try{
    let parseVal = "";
    let tmpButton = MathApp.selected_block;
    //selected.name을 불러와서 parse해주기
    parseVal = parser.eval(tmpButton.name).toString();

    if(parseVal=="false"){
        parseVal="false";
    }
    else if(parseVal[0]=='f'){
        // console.log("parseVal:",parseVal);
        // console.log("this is funciton");
        parseVal="function";
    }

        SYMBOL_WIDTH * parseVal.length
        let size = {
            width: SYMBOL_WIDTH * parseVal.length,
            height: SYMBOL_HEIGHT
        };
        let position = {
            x: Math.random() * (this.canvas.width - size.width) + size.width / 2,
            y: Math.random() * (this.canvas.height - size.height) + size.height / 2
        };
        let new_symbol = new MathApp.Symbol(position, size, parseVal);
    MathApp.canvas.requestRenderAll();
}
catch(e){

    $('.error').text(e);
    console.log(e);
}
```

실행을 누르면 math.js가 알아서 계산해주고, 그에 해당하는 수식의 결과값을 블록으로 만들어준다. function과 같은 경우 수식의 결과 값이 이상하게 떠서 조건문을 추가해서 따로 처리해주었다. 만약 계산하지 못한경우에 error 창에 메시지를 띄워주도록 하였다.

**복제 :** 블록 선택활성화 상태에서 복제버튼 클릭시 복제

```
MathApp.duplicate = function () {
    let tmpButton = MathApp.selected_block;
    let size = {
        width: tmpButton.size.width,
        height: tmpButton.size.height
    };
    let position = {
        x: Math.random() * (this.canvas.width - size.width) + size.width / 2,
        y: Math.random() * (this.canvas.height - size.height) + size.height / 2
    };
    let new_symbol = new MathApp.Symbol(position, size, tmpButton.name);
}
```

복사는 키를 누르지 않고 선택된 키를 기준으로 새롭게 블록을 다시 만들어주는 방식을 채택했다.

+quick button

```
$('.quick').click(function(e){
    let text=$(this).text();
    let size = {
        width: SYMBOL_WIDTH*text.length,
        height: SYMBOL_HEIGHT
    };
    let position = {//randomposition
        x: Math.random() * (MathApp.canvas.width - size.width) + size.width / 2,
        y: Math.random() * (MathApp.canvas.height - size.height) + size.height / 2
    };
    let new_symbol = new MathApp.Symbol(position, size, text);
})
```

quick button메뉴에 있는 글자를 바로 생성해주는 코드이다.

<상호작용 메뉴>

**소멸 :** 선택 활성화 블록 삭제버튼 클릭시 소멸됨.

```
$('.menu').click(function (e) {
    // console.log("click");
    if ($(this).text() == '실행'){
        MathApp.execute();
    }
    else if ($(this).text() == '삭제') {
        MathApp.selected_block.destroy();
    }
    else if ($(this).text() == '복제') {
        MathApp.duplicate();
    }
    else if ($(this).text() == '분해') {
        MathApp.Disassemble();
    }
    else if ($(this).text() == '전체삭제') {
        let blocks=MathApp.blocks;
        for(let i=blocks.length-1;i>=0;i--)
        {
            blocks[i].destroy();
        }
    }
```

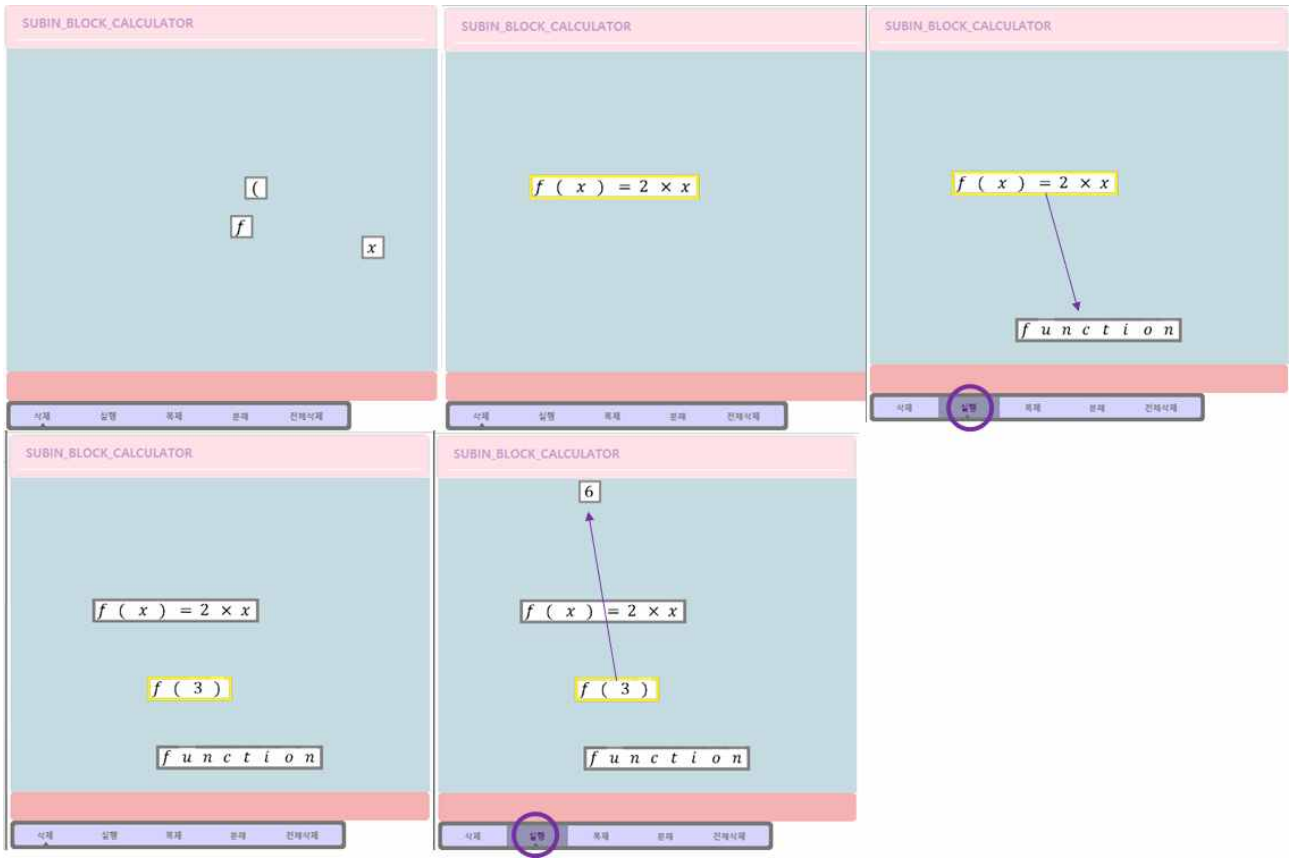
메뉴에 있는 버튼들을 누르면 다음이 실행된다. 삭제버튼을 누르면, 선택된 블록을 삭제하는 함수가 호출되고, 전체삭제를 누르면 blcoks 배열의 뒤쪽부터 모두 순회하면서 블록을 삭제한다.

**이동 :** 마우스 드래그(기본 제공코드)

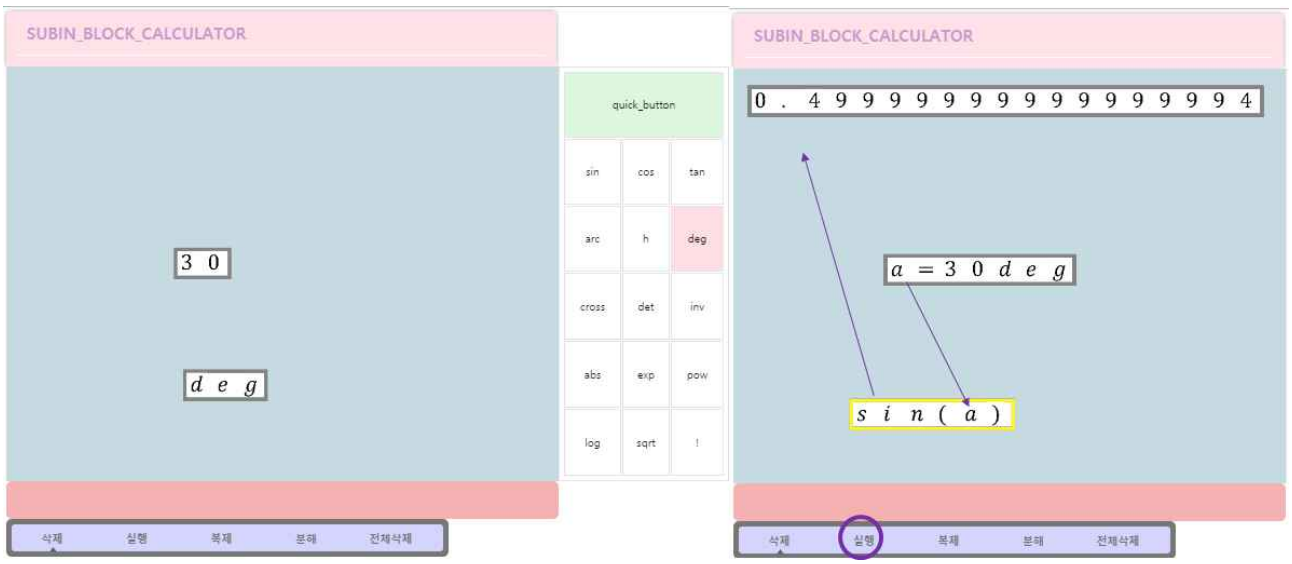
```
MathApp.Block.prototype.moveTo = function (p) {
    let tx = p.x - this.position.x;
    let ty = p.y - this.position.y;
    this.translate({ x: tx, y: ty });
}
MathApp.Block.prototype.translate = function (v) {
    this.position.x += v.x;
    this.position.y += v.y;
    this.visual_items.forEach(item => {
        item.left += v.x;
        item.top += v.y;
    });
}
```

# 실제 문제에 대한 사용 예시

## 1

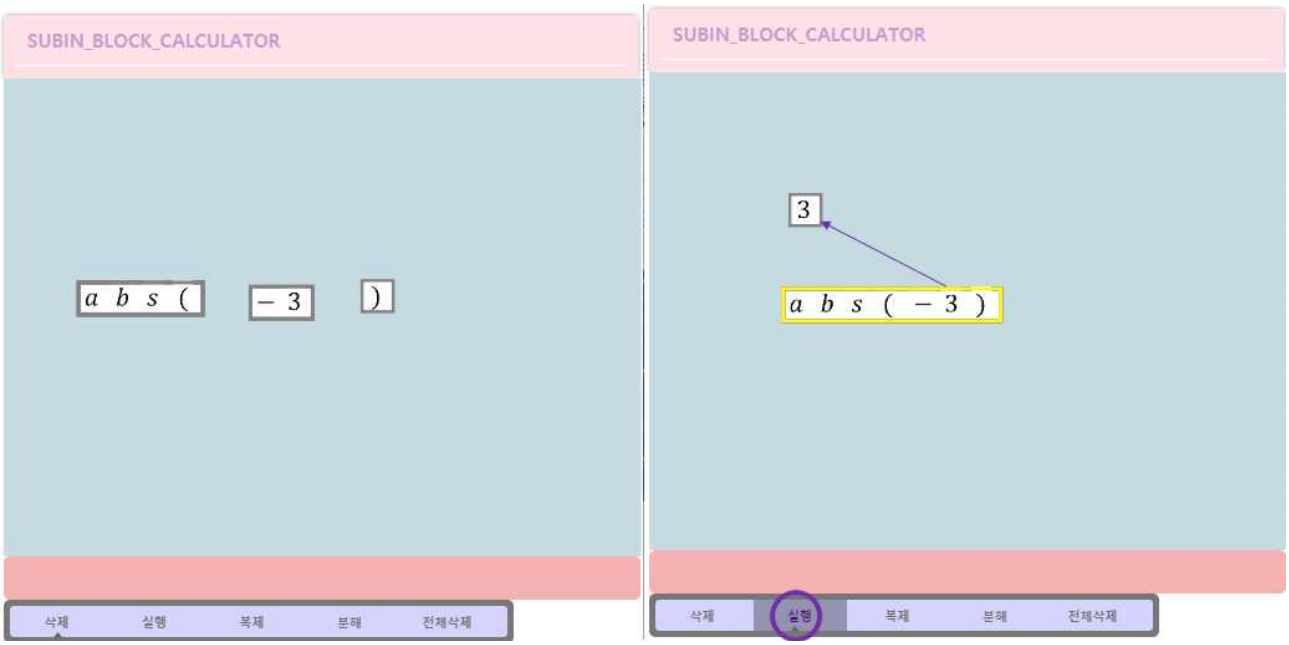



## 2



# 실제 문제에 대한 사용 예시

**3**



# 논의

### 구현 측면에서 성공적인 부분과 실패한 부분

**성공**

1. 전체적으로 보기 좋은 블록계산기를 만들었다.

2. parser로 전체적인 계산을 하는데 문제없을 정도로 잘 구현했다.

**실패**

1. 처음에 짠 코드에서 새롭게 symbols객체를 만들어서 block을 상속하려고 시도했으나, 마음대로 되지 않아 branch로 남겨두고 다시 원상복귀 시켰다.

2. 테두리부분이 깔끔하게 들어가지 않았다.

### 사용성 측면에서 긍정적인 측면과 부정적인 측면

**긍정**

quick_button으로 sin, cos등 자주 쓰이는 수식들을 하나씩 입력해서 합쳐야 하는 귀찮음을 덜어주었다. 또한, 어느 연산들이 가능한지 가시적이어서 사용할 때 가이드가 될 것 같다.

**부정**

1. 전체적으로는 블록계산기로 계산을 하는 것은 좋지 않은 일인 것 같다. 블록들을 합쳐야하는 중간과정이 생겨서 계산에 소모되는 시간이 너무 길어져서 페이지를 빨리 끄고 싶다.

2. 블록 밑에 생기는 창이 아니라 그냥 메뉴공간을 따로 만들어서 넣었는데, 마우스의 이동거리가 더 길어져서 바로 밑에 있는 것 보다는 사용할 때 귀찮을 것 같다.

### 과제 결과에 대한 전반적인 자체 평가 및 향후 개선 계획

**1**

향후 개선한다면, 먼저 branch로 남겨둔, symbols를 다시 구현하고 싶다. 지금은 else로 빼서 symbol의 코드 길이가 굉장히 길어서 후에 이 코드를 수정, 보완할 때 읽기 싫을 것 같다.

**2**

첫 번째 과제 때보다 객체에 대해서 좀 더 이해하면서 작성할 수 있었던 것 같다. 그래서 새로운 내용을 작성할 때에도 원하는 부분을 표현하는 데에 고민한 시간이 1차과제보다 많이 줄어들었다.

클릭 가능한 YouTube 링크

https://youtu.be/-lPCdxJJaPM