1. 개요

- 인터페이스 요구조건 완성도 요약

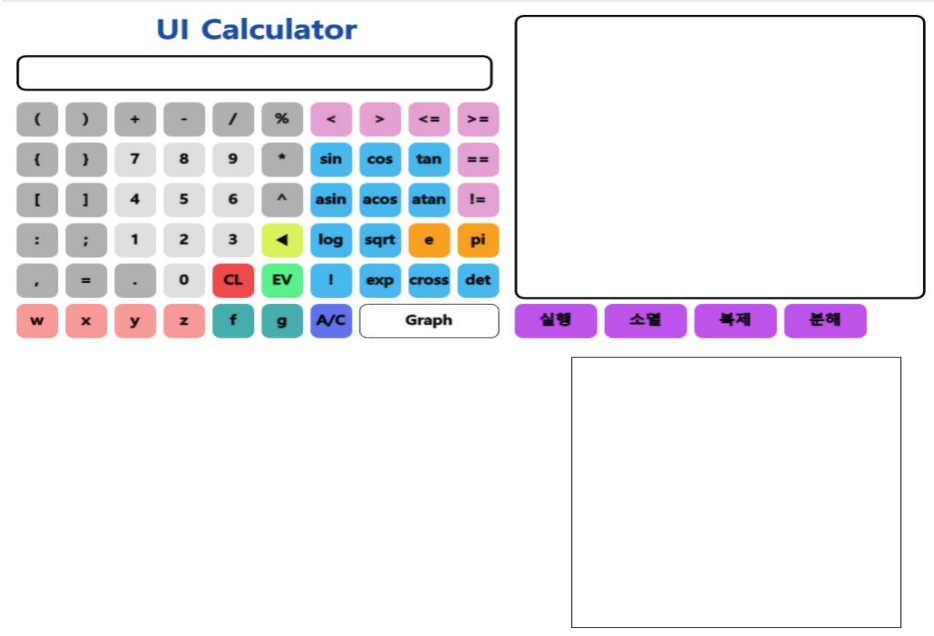
	구현사항	구현여부
시각적 표현	기호, 수, 연산자, 수식	o
	함수, 변수 등	o
대화형 상호작용	생성, 이동	o
	블록 합치기	o
	실행	o
	분해	o
	복제	o
	소멸	o

- 사용 오픈소스 라이브러리

- Plotly 라이브러리

2. 설계 및 구현

• 전체 구현 모습



- 키보드를 이용해서 숫자, 영어, 기호 등을 입력하거나, 화면의 숫자버튼이나 기호 버튼을 클릭하면, 오른쪽 빈 사각형 공간안에 해당 블록들이 생성되고, 하단의 Graph, 실행, 소멸, 복제, 분해 등의 기능을 이용할 수 있다.

• Web UI

```
WebUI.initWidgets = function() {
    WebUI.title = new WebUI.Text("UI Calculator", 40, "#194EA3")
    Resultbox1 = new WebUI.Resultbox("", {width: 580, height: 50})

    CalcButton1_1 = new WebUI.CalcButton("(", {width: 50, height: 50}, "#B0B0B
0")
    CalcButton1_2 = new WebUI.CalcButton(")", {width: 50, height: 50}, "#B0B0B
0")
    CalcButton1_3 = new WebUI.CalcButton("+", {width: 50, height: 50}, "#B0B0B
0")
    CalcButton1_4 = new WebUI.CalcButton("-
", {width: 50, height: 50}, "#B0B0B0")
    CalcButton1_5 = new WebUI.CalcButton("/", {width: 50, height: 50}, "#B0B0B
0")
    CalcButton1_6 = new WebUI.CalcButton("%", {width: 50, height: 50}, "#B0B0B
0")
    CalcButton1_7 = new WebUI.CalcButton("<", {width: 50, height: 50}, "#E4A2D
3")
    CalcButton1_8 = new WebUI.CalcButton(">", {width: 50, height: 50}, "#E4A2D
3")
    CalcButton1_9 = new WebUI.CalcButton("<=", {width: 50, height: 50}, "#E4A2
D3")
    CalcButton1_0 = new WebUI.CalcButton(">=", {width: 50, height: 50}, "#E4A2
D3")

    CalcButton2_1 = new WebUI.CalcButton("{", {width: 50, height: 50}, "#B0B0B
0")
```

- 웹UI는 지난 과제2를 응용했다.

- 대부분 PushButton을 상속한 CalcButton, GraphPlot 등을 사용.

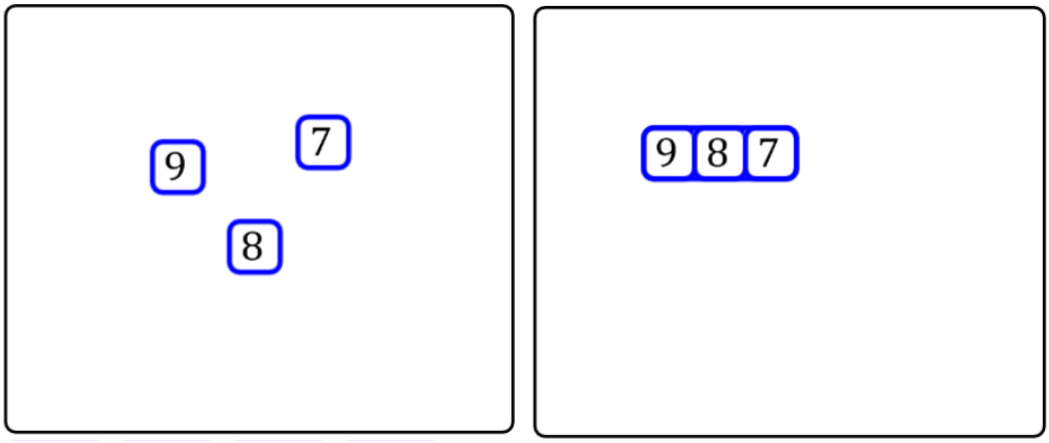
- Text, TextField도 사용하였다.

• 블록 합치기

```
MathApp.handleMouseUp = function(window_p) {
    if(MathApp.is_mouse_dragging)
```

```
    {

        if(MathApp.selected_block != null){


            let overlapBlock = MathApp.findOverlapBlock();
            if(overlapBlock != null){

                MathApp.assembleBlock(overlapBlock, MathApp.selected_block);
            }
        }

        MathApp.is_mouse_dragging = false;
        MathApp.mouse_drag_prev = {x:0, y:0};

        MathApp.canvas.requestRenderAll();
    }
}
```

- 마우스로 블록을 드래그해서, 클릭을 해제하는 순간, 겹쳐있는 블록을 체크(findOverlapBlock 함수)하고 블록이 있으면, 그 블록 오른쪽에 드래그한 블록을 붙인다(assembleBlock 함수).




```
MathApp.findOverlapBlock = function(){
    let x = MathApp.selected_block.position.x;
    let y = MathApp.selected_block.position.y;
    let width = MathApp.selected_block.size.width;
    let height = MathApp.selected_block.size.height;

    for(let i = 0; i < this.blocks.length; i++)
    {
        let block = this.blocks[i];

        if(block == MathApp.selected_block){
            continue;
```

```
        }


        // 왼쪽 위
        if( x - width/2 >= block.position.x - block.size.width/2 &&
            x - width/2 <= block.position.x + block.size.width/2 &&
            y - height/2 >= block.position.y - block.size.height/2 &&
            y - height/2 <= block.position.y + block.size.height/2 ){
            return block;
        } ……
```

- findOverlapBlock : 블록이 겹쳤는지 체크해주는 함수.

- 왼쪽 위, 왼쪽 아래, 오른쪽 위, 오른쪽 아래 범위좌표를 지정해주어 겹치는 기준으로 잡았다.

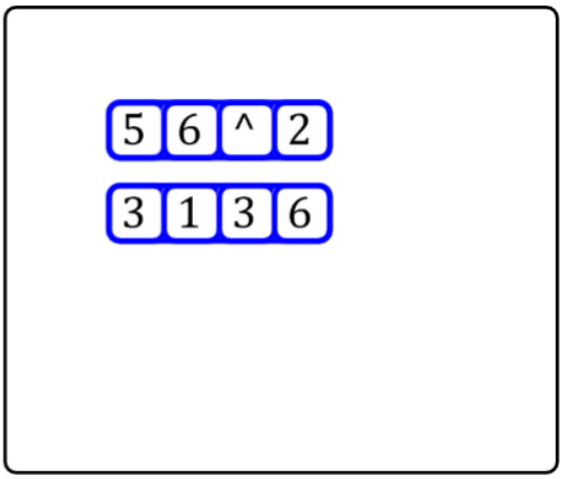
```
// 드래그해 넣는 블록을 접촉 블럭 오른쪽에 결합
MathApp.assembleBlock = function(leftBlock, rightBlock){
    let rightCnt = rightBlock.blockCnt;
    let beforeX = leftBlock.position.x;
    let beforeWidth = leftBlock.size.width;

    // 왼쪽 블록에 이름, 위치, 크기 새로 지정
    leftBlock.blockCnt += rightCnt;
    leftBlock.name += rightBlock.name;
    leftBlock.position = {
        x : leftBlock.position.x + rightBlock.size.width / 2,
        y : leftBlock.position.y
    };

    leftBlock.size = {
        width : leftBlock.size.width + rightBlock.size.width,
        height : leftBlock.size.height
    };

    // 왼쪽 블록 전체 boundary 제거
    leftBlock.visual_items.forEach(item => {
        if(item.boundary){
            MathApp.canvas.remove(item);

            let index = leftBlock.visual_items.indexOf(item);
            if(index > -1)
            {
                leftBlock.visual_items.splice(index, 1);
            }
        }
    })
…
```

- assembleBlock : 블록 두개를 합쳐주는 함수.

- **실행**

```
function excute_calc(){
    let result = 0;

    try
    {
        expression = MathApp.selected_block.name;
        result = parser.eval(expression).toString();
        var tokens = result.split(' ');
        if(tokens[0] == 'function')
        {
            result = 'function';
        }

        make_ResultBlock(result);
    }
    catch (e)
    {
        if(result != 'function')
        {
            alert(e);
        }
    }
}
```

- 선택된 블록의 식을 계산하고, makeResultBlock 함수를 통해 결과 블록 생성.




```
function make_ResultBlock(result){
    for(let i = 0; i < result.length; i++){
        key = result[i];
```

```
        if (key in MathApp.symbol_paths)
        {
            let size = {
                width : SYMBOL_WIDTH,
                height : SYMBOL_HEIGHT
            };
            let position = {    //바로 아래에 생성
                x : MathApp.selected_block.position.x - MathApp.selected_block
.size.width/2 + size.width/2 + i * size.width,
                y : MathApp.selected_block.position.y + size.height/2 + size.h
eight
            };

            resultBlockCnt++;
            var resultSymbol = new MathApp.Symbol(position, size, key);
            currentResultBlocks.push(resultSymbol);
        }
    }
}
```

- make_ResultBlock : 결과 블록 생성 함수

- result로 들어온 계산된 결과 string을 한글자씩 따로 블록을 생성한 뒤, 결합하여 생성해줌.

- **소멸**

```
function destroy_calc(){
    MathApp.selected_block.destroy();
}
```

- 선택 블록 삭제.

- **복제**

```
function duplicate_calc(){
    let str = MathApp.selected_block.name;

    for(let i = 0; i < str.length; i++){
        key = str[i];
        if (key in MathApp.symbol_paths)
        {
            let size = {
                width : SYMBOL_WIDTH,
                height : SYMBOL_HEIGHT
            };
            let position = {    //바로 아래에 생성
```

```
                x : MathApp.selected_block.position.x - MathApp.selected_block
.size.width/2 + size.width/2 + i * size.width,
                y : MathApp.selected_block.position.y + size.height/2 + size.h
eight
            };

            resultBlockCnt++;
            var resultSymbol = new MathApp.Symbol(position, size, key);
            currentResultBlocks.push(resultSymbol);
        }
    }
}
```

- 선택된 블록에 저장된 이름을 불러온 뒤, 글자별로 블록으로 만들고, 다시 결합한다.

- 결합된 블록은 바로 아래에 다시 생성된다.

- **분해**

```
function disassemble_calc(){
    let str = MathApp.selected_block.name;
    let newX = MathApp.selected_block.position.x;
    let newY = MathApp.selected_block.position.y;
    let originWidth = MathApp.selected_block.size.width;
    MathApp.selected_block.destroy();

    for(let i = 0; i < str.length; i++){
        key = str[i];
        if (key in MathApp.symbol_paths)
        {
            let size = {
                width : SYMBOL_WIDTH,
                height : SYMBOL_HEIGHT
            };
            let position = {    //원래 자리 그대로 생성
                x : newX - originWidth/2 + size.width/2 + i * size.width,
                y : newY
            };
            var resultSymbol = new MathApp.Symbol(position, size, key);
        }
    }
}
```

- 선택된 블록에 저장된 이름을 불러온 뒤, 글자별로 블록을 그자리 그대로 분리된 채 다시 생성.

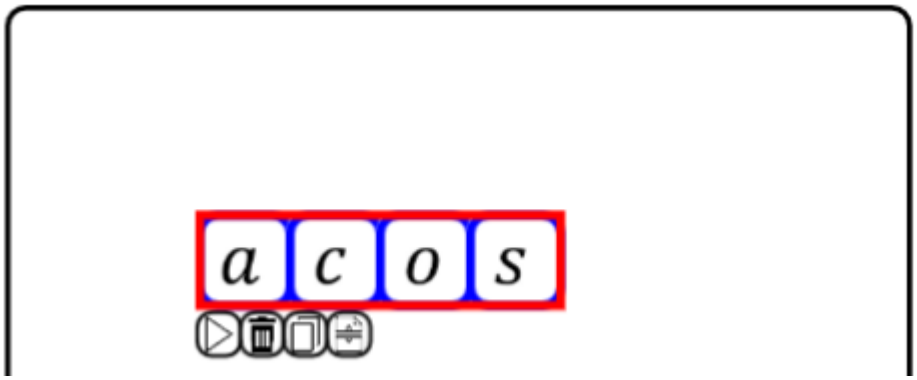
- 기존 선택된 블록은 소멸.

- **팝업 메뉴**

```
function makePopup(block, position, size){
    let left = position.x - size.width / 2;
    let top = position.y + size.height - size.height / 3 - 3;
    for(let i = 0; i < 4; i++){
        let boundary = makePopupBoundary(left, top);
        block.popup_items.push(boundary);
        left += (POPUP_WIDTH + 4);
    }

    left = position.x - size.width / 2 + 2;
    top = position.y + size.height - size.height / 3 - 1;
    for(let i = 0; i < 4; i++){
        let background = makePopupBackground(left, top);
        block.popup_items.push(background);
        left += (POPUP_WIDTH + 4);
    }

    left = position.x - size.width / 2 + 3;
    top = position.y + size.height - size.height / 3;
    makePopupImage("./source/play.png", left, top, block);
    left += (POPUP_WIDTH + 4);
    makePopupImage("./source/trash.png", left, top, block);
    left += (POPUP_WIDTH + 4);
    makePopupImage("./source/copy.png", left, top, block);
    left += (POPUP_WIDTH + 4);
    makePopupImage("./source/split.png", left, top, block);

}
```

- 블록이 선택된 상태인 경우, 블록 바로 아래에 팝업 메뉴 생성.

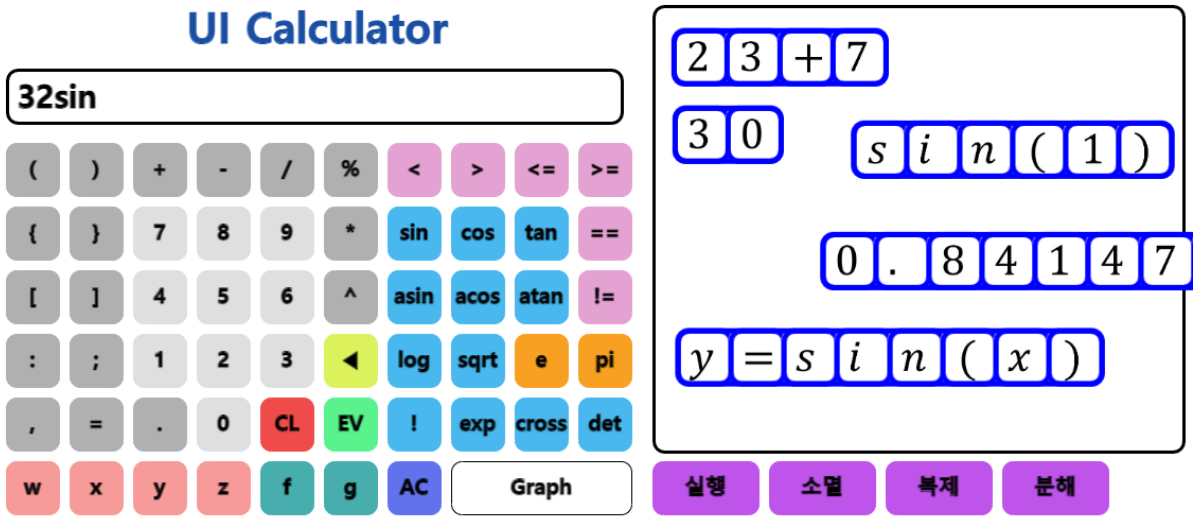
- makePopupBoundary, makePopupBackground는 boundary, background속성 부여 함수.

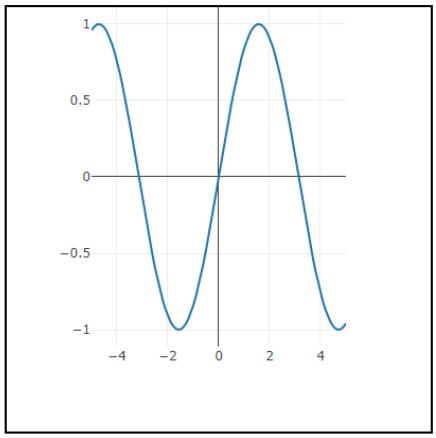


- 왼쪽부터, 실행, 소멸, 복제, 분해

3. 평가

- **실제 실행 화면**

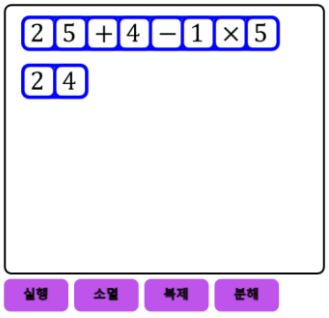




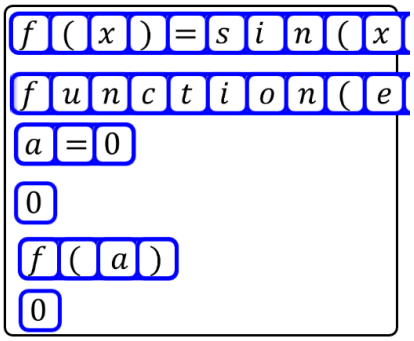
- 간단한 연산, sin 연산, sin(x)그래프 그리기 등을 시행해 보았다.

- **문제 사용 예시**

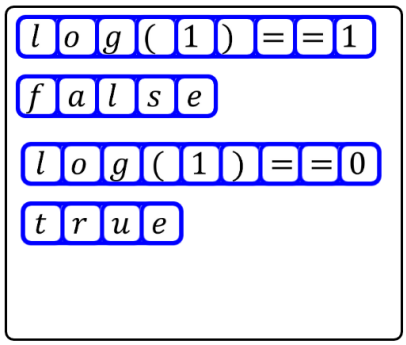
1. 25+4-1*5 = ?



2. f(x) = sin(x), a = 0, f(a) = ?



3. log(1) == 1, log(1) == 0



- **시연 동영상**

https://youtu.be/uGZBx4gb9QM

4. 논의

- **성공적인 부분**

- 생성된 블록들의 결합, 실행, 소멸, 복제, 분해 기능을 성공적으로 구현함.

- Plotly 라이브러리를 이용해 그래프를 성공적으로 구현함.

- 이전 과제를 바탕으로 클릭을 통한 블록 생성도 구현함.

- **실패한 부분**

- 선택된 블록만이 아닌 모든 블록을 한번에 소멸시킬 수 있는 기능을 구현하려고 시도했지만, 구현하지 못하였다.

- 함수가 선언되었다는 결과물 function(e,t){.....} 블록이 생성되는 것을 막고 싶었지만, 구현하지

못하였다.

• 사용성 측면에서 긍정적인 측면과 부정적인 측면

- 클릭으로도 계산이 가능하고 블록을 생성하여 계산이 가능하므로, 키보드를 사용하여 입력하지 않아도 사용이 가능하다.

- 블록이 랜덤한 자리에 생성되기 때문에 사용자가 이미 결합해 놓은 블록 위에 생성될 수도 있어 사용자가 이어서 블록 수식을 결합하는데 불편함이 있다.

- 블록 분해를 할 때, 전체가 각각 한번에 분해 되는 기능만 있지, 분해하고 싶은 블록 하나만 따로 분리해 낼 수 없어, 분해 후, 다시 결합해야 하는 불편함이 있다.

- 블록이 많아져 화면을 비우고 싶을 때, 전체 블록을 한번에 삭제할 수 있는 기능이 없어 불편함이 있다.

• 자체 평가 및 향후 개선 계획

- 전체적인 클릭형, 드래그앤드롭형 계산기로서는, 잘 구현이 되어있다고 생각이든다.

- 전체 블록 삭제 기능을 추가하면 더 편할 것이다.

- 신규 블록이 생성되는 위치를 규칙성있게 바꿔볼 예정이다.

- 분해 시, 원하는 블록만 따로 분해할 수 있도록 구현해 볼 예정이다.