휴먼컴퓨터인터페이스

WebUI 계산기 구현

과목: 휴먼컴퓨터인터페이스

담당교수: 이강훈 교수님

제출일: 2021년 5월 15일

전공: 전자공학과

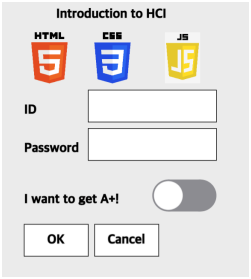
학번: 2017706056

이름:최혁순

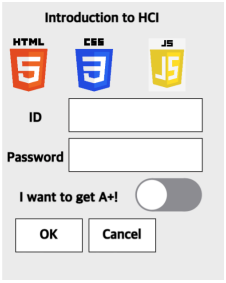
목차

요구조건 제약조건 만족도	
0단계	O
1단계	O
2단계	O
3단계	O

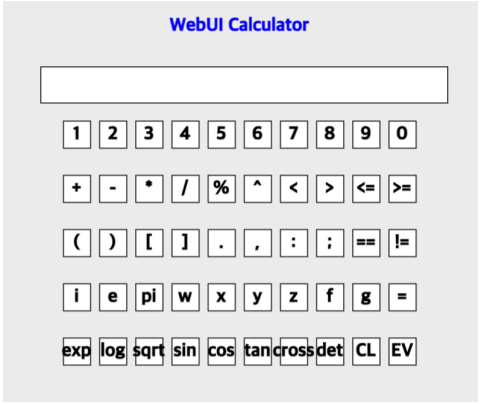
0단계



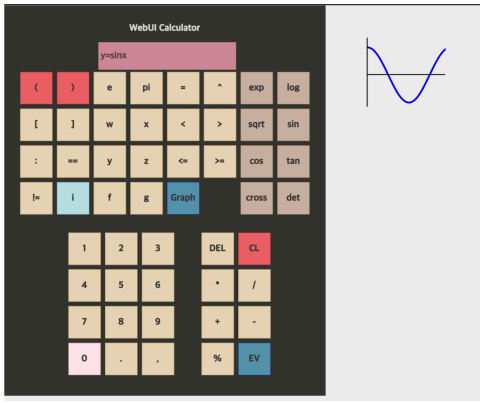
1단계



2단계



3단계



0단계: 위젯 구현

위젯기능구현을 위한 코드제시 및 의미설명

0단계는 위젯을 정의 한 후, 각각의 위젯을 생성한 후, 사용자가 직접 좌표를 설정해 위젯을 배치하는 방식이 사용되었다.

대표적인 코드들을 살펴보면,

```
WebUI.initWidgets = function() {

WebUI.title = new WebUI.Text("Introduction to HCI");
WebUI.img_html =new WebUI.Image("resources/HTML5.png", {width:
100, height: 80});
WebUI.img_css =new WebUI.Image("resources/CSS3.png", {width: 100,
height: 80});
WebUI.img_js =new WebUI.Image("resources/JS.png", {width: 100,
height: 80});
WebUI.text_id = new WebUI.Text("ID"); WebUI.text_pwd = new
WebUI.Text("Password");
WebUI.edit_id = new WebUI.TextField("", {width:200, height:50});
WebUI.edit_pwd = new WebUI.TextField("", {width:200, height:50});
WebUI.btn_ok = new WebUI.PushButton("OK", {width:100, height:50});
WebUI.btn_cancel =new WebUI.PushButton("Cancel", {width:100,
height:50});
WebUI.text_blah = new WebUI.Text("I want to get A+!");
WebUI.switch = new WebUI.Switch(false, {width:100, height:50});
```

위 코드를 통해 각각의 위젯을 생성하였다. 각각의 위젯을 생성하는데 필요한 인자를 넣으면서 생성한 모습이다.

```
WebUI.initVisualItems = function() {
    WebUI.widgets.forEach(widget => {
        widget.initVisualItems();
    });
}
```

위 코드를 통해 각각의 WebUI.widgets에 담긴 위젯들을 forEach로 돌면서 각각의 위젯 객체에 정의 되어있는 initVisualItems를 호출한다.
이때 코드 상단을 살펴보면 WebUI.widgets =[];로 비어있는것을 알 수 있는데,

```
WebUI.Widget = function() {
       WebUI.widgets.push(this);
```

다음과 같이 모든 위젯의 부모가 되는 WebUI.Widget에 보면 각 위젯들이 생성될 때, WebUI.widgets에 자기 자신을 추가해 요소를 채워나가는 것을 알 수 있다.

이제 각 위젯들을 살펴보면,

```
WebUI.Text = function(label) {
    WebUI.Widget.call(this);
```

이렇게 각 위젯들은 WebUI.Widget의 속성을 상속받는 것을 알 수 있다. 이 외에도 각 위젯마다 각기 다른 특성을 지니고 있어야 하기에 자신만의 속성이 추가됨을 알 수 있다.

```
ebUI.Text.prototype = Object.create(WebUI.Widget.prototype);
WebUI.Text.prototype.constructor = WebUI. Text;
```

또한 prototype 또한 WebUI.Widget의 것을 상속받으며, constructor는 고유의 것을 지니기 위해 다시 설정해주는 코드이며, 이 과정들은 모든 위젯이 거치는 과정들이다. 각 위젯들은 WebUI.Widget이 가지고 있는 기본 성질들을 바탕으로 만들어지기 때문이다.

각 위젯들은 대부분 비슷한 원리로 이루어져있으나, 이번 과제에서 가장 중요한 역할을 했던 PushButton을 대표적으로 살펴보면,

```
WebUI.PushButton.prototype.initVisualItems = function() {
    let background = new fabric.Rect({
        left: this.position.left,
        top: this.position.top,
        width: this.desired_size.width,
        height: this.desired_size.height,
        fill: this.fill_color,
        stroke: this.stroke_color,
        strokeWidth: 1,
        selectable: false
    });

    let text = new fabric.Text(this.label, {
```

```
        left: this.position.left,
        top: this.position.top,
        selectable: false,
        fontFamily: this.font_family,
        fontSize:   this.font_size,
        fontWeight: this.font_weight,
        textAlign:  this.text_align,
        stroke:     this.text_color,
        fill:       this.text_color,
    });

    let bound = text.getBoundingRect();
    text.left = this.position.left + this.desired_size.width/2 -
bound.width/2;
    text.top = this.position.top + this.desired_size.height/2 -
bound.height/2;

    this.size = this.desired_size;

    //
    this.visual_items.push(background);
    this.visual_items.push(text);
    this.is_resource_ready = true;
}
```

initVisualItems가 이런식으로 정의되어있는데, 먼저 fabric.Rect와 fabric.Text가 연달아 생성되어 있는것을 볼 수 있다. 이는 PushButton이 사각형의, 글씨가 있는 버튼이기에 생성된 것이다.
this.size= this.desired_size 에서 설정된 this.size는 생성된 버튼의 클릭 여부를 판단한다. 마우스의 클릭이 일어난 시점에서, this.size 범위 안에 마우스가 있었다면, 이 버튼이 클릭되었다고 판단하게 된다.
또한

```
    this.visual_items.push(background);
    this.visual_items.push(text);
```

이 코드를 통해, visual_items에 객체들을 넣어주고 이 배열에 들어가있는 객체들을 나중에 화면에 그려주는 과정을 통해 사용자에게 위젯들을 보여준다.

스위치 위젯은 직접 구현해야 했다.

```
WebUI.Switch = function(is_on, desired_size) {
    WebUI.Widget.call(this);
    this.type = WebUI.WidgetTypes.SWITCH;

    this.desired_size = desired_size;
    this.is_pushed = false;
    this.fill_colorOn = "rgba(48, 209,88)";
    this.fill_colorOff = "rgb(142,142,147)";
    this.is_on= is_on;
```

먼저 WebUI.Widget을 상속받고, 인자로 스위치가 처음 생성되었을 때, 켜져있을지 꺼져있을지 정해주는 is_on, 스위치의 크기를 지정해주는 desired_size를 받아 생성한다.
스위치가 켜질때와 꺼질때의 색이 다르기에 fill 색상을 두개로 지정해서 골라서 쓸 것을 계획했다.

```
WebUI.Switch.prototype = Object.create(WebUI.Widget.prototype);
WebUI.Switch.prototype.constructor = WebUI.Switch;
```

앞과 동일하게 프로토타입도 WebUI.Widget을 상속받고 constructor는 자기 자신의 것을 받는다.

```
WebUI.Switch.prototype.initVisualItems = function() {
    // IMPLEMENT HERE!
    let arc1 = new fabric.Circle({
        radius: this.desired_size.width/4,
        left: this.position.left,
        top: this.position.top,
        selectable: false
    });

    let rect = new fabric.Rect({
        left: this.position.left+this.desired_size.width/4,
        top: this.position.top,
```

```
        width: this.desired_size.width/2,
        height: this.desired_size.height,
                selectable: false
```

```
    let arc2 = new fabric.Circle({
        radius: this.desired_size.width/4,
        left: this.position.left+this.desired_size.width/2,
        top: this.position.top,
        selectable: false
```

initVisualItems이다. 원 두개와 사각형 하나를 테두리 없이, fill 속성으로만 겹쳐 생성하니 원하는 버튼 모양을 만들 수 있었다. 처음엔 반원,사각형,반원 이렇게 세개의 도형을 그려봤는데 각 반원과 사각형 사이에 미세한 틈이 생겨 반원이 아닌 그냥 원을 그려 넣었다. 테두리가 없기 때문에 하나의 이어진 도형처럼 보이는 효과를 얻을 수 있었다. 또한 selectable은 모두 false로 두었다.

위치와 크기는 객체를 생성할 때 주어지는 값이 변하더라도 적절한 모양이 나올 수 있도록 설정했다.

```
let arc3 = new fabric.Circle({
        radius: (this.desired_size.height/2)*0.9,
        left: this.position.left+(this.desired_size.height/2)*0.1,
        top: this.position.top+(this.desired_size.height/2)*0.1,
        fill: 'white',
    });
```

애니메이션 효과를 줄 때, 움직이는 하얀색 동그란 버튼이다. 바깥쪽 도형보다 조금 작은 크기를 갖는다.

```
if(this.is_pushed){
        arc1.set({"fill":this.fill_colorOn});
        arc2.set({"fill":this.fill_colorOn});
        rect.set({"fill":this.fill_colorOn});
    }
```

```
    else {
```

```
        arc1.set({"fill":this.fill_colorOff});
        arc2.set({"fill":this.fill_colorOff});
        rect.set({"fill":this.fill_colorOff});
    }
```

처음에 객체가 생성될 때, 켜져있는 상태면 초록색을, 꺼져있는 상태면 회색을 디폴트 색상으로 지정하기 위한 코드이다. 속성을 바꾸는 set을 사용하였다.

```
let bound = rect.getBoundingRect();
        this.position.left = bound.left;
        this.position.top = bound.top;
        this.size.width = bound.width;
        this.size.height = bound.height;
        this.visual_items.push(arc1);
        this.visual_items.push(rect);
        this.visual_items.push(arc2);
        this.visual_items.push(arc3);
        this.is_resource_ready = true;
```

객체의 위치, 크기 설정과 시각적으로 보여지는 객체들을 그려주기 위해 visual_items 배열에 추가하고, 사진이 아니기 때문에 is_resourced_ready를 true로 설정해주었다. 그러나 이런 방식으로 스위치를 구현하니 스위치 양 끝의 동그란 부분을 클릭했을 때에는 이벤트가 일어나지 않았다. 그 이유는 스위치의 범위를 스위치를 구성하는 위젯중 사각형 위젯의 범위로 받아서였다. 범위를 계산하는 알고리즘이 현재

```
if (x >= widget.position.left &&
            x <= widget.position.left + widget.size.width &&
            y >= widget.position.top &&
            y <= widget.position.top + widget.size.height) {
            return widget;
        }
```

이런방식으로 되어있다. 이는 x좌표가 위젯의 시작좌표보다 크고 (위젯의 시작좌표+위젯의 너비) 보다 작고 동시에 y좌표가 위젝의 시작좌표보다 크고 (위젯의 시작좌표+위젯의 높이)보다 작으면 마우스가 위젯 안에 있다고 판단되기 때문에, 사각형 범위를 판단한다. 따라서 이

부분을 수정하지 않으면 언제나 위젯의 범위는 사각형으로만 판정할 수 있다는 한계를 가진다. 하지만 스위치부분이 아닌 부분을 수정하는 것은 이번 과제의 범위가 아니기 때문에 현재 상황에서 낼 수 있는 최선의 결과를 도출하기로 했다.

위와 같은 식으로 스위치의 동그란 부분까지 클릭 되도록 변경하였다. 구석의 모서리 부분까지 클릭이 된다는 것이 문제이긴 하지만, 이 전의 방식보단 더 정확한 범위이고, 오차가 생기는 부분이 작기에 최선의 방법이라 생각했다.

```
    let bound = rect.getBoundingRect();
        this.position.left = bound.left-this.desired_size.width/4;
        this.position.top = bound.top;
        this.size.width = bound.width+this.desired_size.width/2;
        this.size.height = bound.height;
```

수정된 코드는 다음과 같다. 원래 left좌표에서 원의 반지름만큼 빼주고, 원래 너비에서 원의 지름만큼 더해주었다.

```
WebUI.Switch.prototype.handleMouseDown = function() {
    // IMPLEMENT HERE!
    if (!this.is_pushed) {

        this.is_pushed = true;

        if (this.onPushed != undefined) {
        this.onPushed.call(this);
        }
        return true;
    }
        else {
        return false;
```

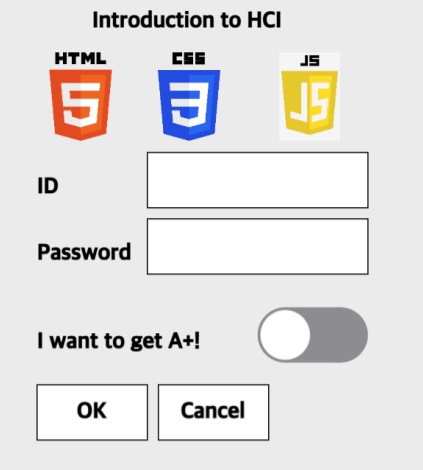
마우스가 눌렸을 때는 일반 PushButton과 다를바 없는 방식으로 구성하였다. 눌렸는지를 반환한다.

```
WebUI.Switch.prototype.handleMouseUp = function() {
    if (this.is_pushed) {
       if(!this.is_on){
```

```
        this.visual_items[3].animate('left', '+=50', { onChange:
WebUI.canvas.renderAll.bind(WebUI.canvas) });
        this.visual_items[0].set({"fill":this.fill_colorOn});
        this.visual_items[1].set({"fill":this.fill_colorOn});
        this.visual_items[2].set({"fill":this.fill_colorOn});
    }
        if(this.is_on){
        this.visual_items[3].animate('left', '-=50', { onChange:
WebUI.canvas.renderAll.bind(WebUI.canvas) });
        this.visual_items[0].set({"fill":this.fill_colorOff});
        this.visual_items[1].set({"fill":this.fill_colorOff});
        this.visual_items[2].set({"fill":this.fill_colorOff});
    }
        this.is_on=!this.is_on
        this.is_pushed = false;
        return true;
    }
        else {
        return false;}
```

마우스를 눌렀다 뗐을때, 만약 스위치가 꺼져있었다면 하얀 원을 50픽셀만큼 오른쪽으로 서서히 이동시키는 애니메이션을 발동시키고, 바탕 도형을 초록색으로 바꾼다. 반대로 켜져있었다면, 하얀 원을 왼쪽으로 서서히 이동시키고, 바탕을 회색으로 바꿔 꺼지는듯한 모션을 만들어준다. 두 경우 모두 이러한 이벤트가 일어난 후엔 this.is_on= !this.is_on 코드로 상태를 반대로 만들어준다.

로그인페이지 구성을 위한코드 제시 및 결과분석



0단계의 방법은 위젯의 생성과 위젯의 배치가 매우 직관적이었다는 생각이 든다.
좌표를 직접 찍는 일은 원하는 정확한 위치에 위젯을 배치할 수 있다는 점과, 세부적인 위치까지 조정할 수 있단 점이 장점이라고 생각되었다. 다만 위젯의 갯수가 많아진다면, 많은 위젯들의 위치를 일일이 찍는것은 매우 번거로울 것으로 예상된다.

1단계: 레이아웃 구현
레이아웃기능 구현을 위한 코드 제시 및 의미설명

1단계는 0단계에서 구현한 로그인 화면과 동일하지만, 절대적인 좌표를 찍어 위치시키는것이 아닌, 레이아웃 기능을 통해 상대적인 위치를 계산해서 자동으로 배치해주는 과정을 거쳤다.

먼저 레이아웃 위젯으로는 Container, Column, Row 세가지가 있는데 세가지가 비슷한 구조를 가진다.

먼저 Container를 살펴보면,

```
new WebUI.Container({
                    desired_size: {width: 300, height: 60},
                    horizontal_alignment: WebUI.Alignment.CENTER,
                    vertical_alignment: WebUI.Alignment.CENTER,
                    children: [ new WebUI.Text("Introduction to
HCI") ],
                    }),
```

생성될 때 인자로 여러 프로퍼티를 받게 되는데, 컨테이너의 크기, 컨테이너 안에 들어갈 요소의 위치, 그 안에 들어갈 자식 객체들 을 가지게 된다.
horizontal_alignment는 컨테이너 내부에서 컨테이너의 자식 객체가 어떤 수평적인 위치를 가지는지 정해준다. 지금같은 경우 CENTER로 되어있기 때문에, 컨테이너의 중앙에 위치하고, vertical_alignment는 반대로 수직적인 위치를 정해준다. 이 경우에 horizontal_alignment와 vertical_alignment 둘 다 CENTER로 되어있기 때문에 정 중앙에 자식 객체들이 위치할 것이다.

```
WebUI.Container.prototype.extendSizeChildren = function(size,
child_size) {
    // IMPLEMENT HERE!
    if (size.width < child_size.width) size.width =
child_size.width;
if (size.height < child_size.height) size.height =
child_size.height; return size;
}
WebUI.Container.prototype.calcNextPosition = function(position,
size) {
    // IMPLEMENT HERE!
    let next_left = position.left;
```

```
let next_top = position.top;
return {left: next_left, top: next_top};
}
```

컨테이너 자식의 크기와 각 요소의 위치를 정의하는 메소드이고 크기를 측정하는 메소드는 컨테이너의 자식 요소들을 하나씩 돌면서 이 메소드가 호출되는데, 단순히 자식들 가장 큰 것의 크기를 따른다는 코드이다.

```
WebUI.Column = function(properties) {
    WebUI.Widget.call(this, properties);

    this.type = WebUI.WidgetTypes.COLUMN;
}

WebUI.Column.prototype = Object.create(WebUI.Widget.prototype);
WebUI.Column.prototype.constructor = WebUI.Column;

WebUI.Column.prototype.extendSizeChildren = function(size,
child_size) {
    // IMPLEMENT HERE!
    size.width += child_size.width;
if (size.height < child_size.height) size.height =
child_size.height;
return size;
}

WebUI.Column.prototype.calcNextPosition = function(position, size)
{
    // IMPLEMENT HERE!
    let next_left = position.left + size.width;
let next_top = position.top;
return {left: next_left, top: next_top};
}
```

가로로 자식들을 나열하는 위젯이다. 가로 사이즈는 각 자식들의 너비를 더해가고, 세로 사이즈는 각 자식들의 높이중 가장 높이가 큰 것의 세로 사이즈를 따라간다. 가로 위치는 각 자식의 너비만큼 더해가고, 가로로 나열하기때문에 세로 위치는 변하지 않는다.

```
WebUI.Row = function(properties) {
    WebUI.Widget.call(this, properties);

    this.type = WebUI.WidgetTypes.ROW;
}

WebUI.Row.prototype = Object.create(WebUI.Widget.prototype);
WebUI.Row.prototype.constructor = WebUI.Row;

WebUI.Row.prototype.extendSizeChildren = function(size,
child_size) {
    // IMPLEMENT HERE!
    if (size.width < child_size.width) size.width =
child_size.width; size.height += child_size.height;
return size;
}

WebUI.Row.prototype.calcNextPosition = function(position, size) {
    // IMPLEMENT HERE!
    let next_left = position.left;

let next_top = position.top + size.height;
return {left: next_left, top: next_top};
}
```

반대로 세로로 자식을 나열하는 위젯이다. 세로 사이즈는 각 자식들의 너비를 더해가고, 가로 사이즈는 각 자식들의 너비중 가장 너비가 큰 것의 가로 사이즈를 따라간다. 세로 위치는 각 자식의 높이만큼 더해가고, 세로로 나열하기 때문에 가로 위치는 변하지 않는다.

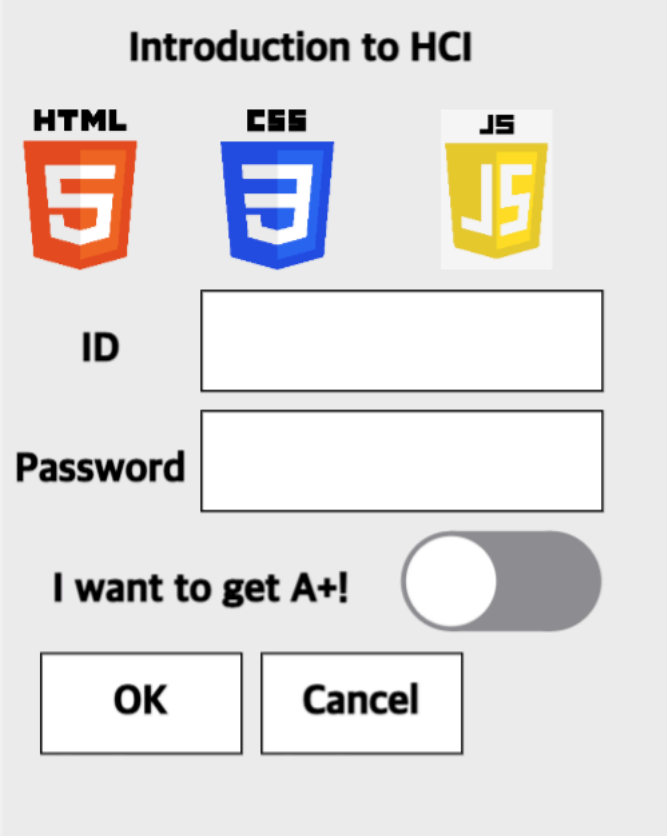
로그인페이지재구성을위한코드제시및결과분석

```
new WebUI.Column({
                    children: [
                        new WebUI.Image("resources/HTML5.png",
                                                {width: 100, height:
80}),
                        new WebUI.Image("resources/CSS3.png",
```

```
                                                {width: 100, height:
80}),
                            new WebUI.Image("resources/JS.png",
                                                {width: 100, height:
80})
                        ]
                    }),
```

다음과 같이 Column의 자식에 Image 위젯 세개를 넣어주면 세 위젯들이 가로로 나열되어 있는 모습을 확인할 수 있다. 위젯의 크기만 지정해주었지 위치는 Container, Column, Row 레이아웃 위젯들이 상대적인 위치를 계산해 자동으로 배치해준다.

레이아웃 위젯을 사용하여 배치한 결과는 다음과 같다.



여러 시행착오끝에 완성은 하였지만, 초반엔 컨테이너 위젯이 어떤 크기를 가지고 있고, horizontal_alignment와 vertical_alignment에 따라 어떻게 위치해 있는지 눈으로 확인할 수 없어 힘들었다. 따라서

```
WebUI.Container.prototype.initVisualItems = function() {
    let background = new fabric.Rect({
        left: this.position.left,
        top: this.position.top,
        width: this.desired_size.width,
```

```
        height: this.desired_size.height,
        fill: this.fill_color,
        stroke: this.stroke_color,
        strokeWidth: 1,
        selectable: false
    });

    //
    this.visual_items.push(background);
    this.is_resource_ready = true;
}
```

임시로 위와 같은 코드를 작성해 컨테이너의 테두리를 살펴보았다.

위와 같은 화면을 볼 수 있었으며, 이 결과를 토대로 컨테이너가 어떤 역할을 하는지 코드뿐만 아니라, 시각적으로 학습함으로써 더 자유자재로 사용할 수 있었으며 원하는 화면을 만들 수 있었다. 현재 모든 컨테이너의 `horizontal_alignment`와 `vertical_alignment` 속성이 CENTER여서 모든 컨테이너 안의 위젯이 컨테이너 정중앙에 위치한다는 것을 시각적으로 알 수 있었다.

컨테이너위젯 사용 전에는 “ID” 텍스트가 상대적으로 모니터 왼쪽에 치우쳐있었는데, 컨테이너로 가상의 공간을 만들고 horizontal_alignment와 vertical_alignment 모두 CENTER로 설정해 ID주변에 여유를 두게 만들었다. 만약 horizontal_alignment가 RIGHT이었다면 TextField와 Text가 붙어있는 형태가 되었을 것이다. OK버튼과 Cancel 버튼 또한 너무 왼쪽으로 치우쳐져 있어서 컨테이너를 사용하여 왼쪽 벽과 거리를 두었다.
지금의 구조는 서로 관계를 갖고 있는 위젯들을 Column을 사용하여 가로로 나열했고, Column으로 묶인 객체들을 다시 Row의 자식들로 두어 세로로 나열하였다.

자세히 설명하자면, “Introduction to HCI”가 컨테이너 안에 있게 함으로써 다른 위젯들과 크기를 맞춰주었으며 코드는 다음과 같다.

```
new WebUI.Container({
                    desired_size: {width: 300, height: 60},
                    horizontal_alignment: WebUI.Alignment.CENTER,
                    vertical_alignment: WebUI.Alignment.CENTER,
                    children: [ new WebUI.Text("Introduction to
HCI") ],
                    })
```

앞에 설명한 ID 텍스트의 크기와 Paddword 텍스트의 크기 조정 과정은 동일하며 다음과 같은 코드로 이루어져있다.

```
new WebUI.Container({
                        desired_size: {width: 100, height: 60},
                        horizontal_alignment:
WebUI.Alignment.CENTER,
                        vertical_alignment:
WebUI.Alignment.CENTER,
                        children: [
                            new WebUI.Text("ID") ],
new WebUI.Container({
                        desired_size: {width: 100, height: 60},
                        horizontal_alignment:
WebUI.Alignment.CENTER,
```

```
                        vertical_alignment:
WebUI.Alignment.CENTER,
                        children: [
                            new WebUI.Text("Password"),
```

둘 다 너비 100, 높이 60의 컨테이너 정 중앙에 위치한다는 코드이다.
I want to get A+ 코드와 스위치또한 위치가 원하는 바가 아닌 벽에 붙어있는 형태여서

```
 new WebUI.Container({
                    desired_size: {width: 200, height: 60},
                    horizontal_alignment: WebUI.Alignment.CENTER,
                    vertical_alignment: WebUI.Alignment.CENTER,
                    children: [

                new WebUI.Text("I want to get A+!")
                    ],
                    }),
```

```
                    new WebUI.Container({
                        desired_size: {width: 100, height: 60},
                        horizontal_alignment:
WebUI.Alignment.CENTER,
                        vertical_alignment:
WebUI.Alignment.CENTER,
                        children: [

                            new WebUI.Switch(false, {width: 100,
height: 50})                        ],
                        }),
```

다음과 같이 각각 너비 200 높이 60, 너비 100 높이 60의 컨테이너로 묶어주었다.

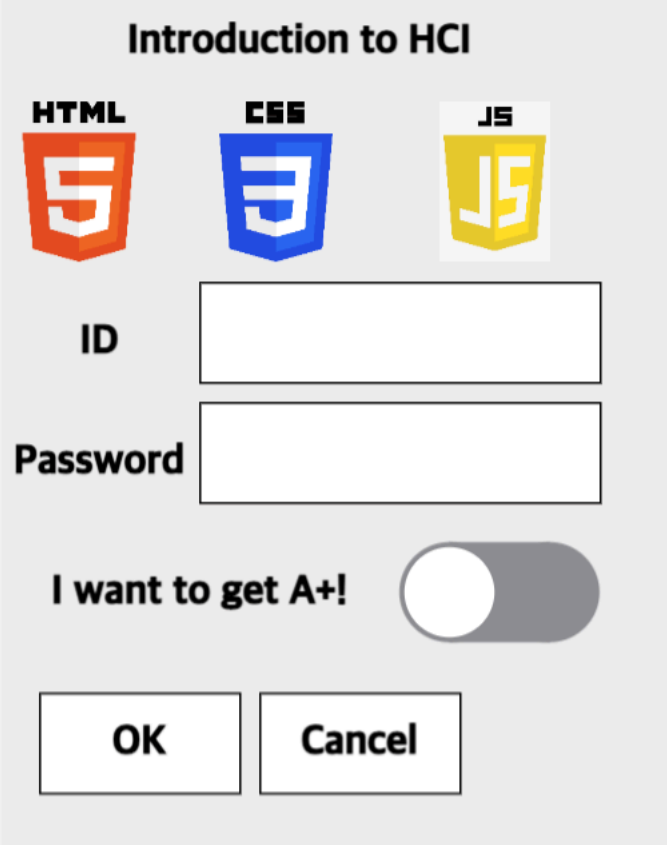
OK버튼과 Cancel 버튼 또한 왼쪽에 너무 치우쳐져있어서 왼쪽에 패딩을 덧붙일 역할을 해줄 컨테이너 안에 적재하였다. 코드는 밑과 같다.

```
new WebUI.Container({
        desired_size: {width: 250, height: 60},
        horizontal_alignment: WebUI.Alignment.CENTER,
        vertical_alignment: WebUI.Alignment.CENTER,
        children: [

    new WebUI.Column({
```

```
        children: [
        new WebUI.PushButton("OK", {width: 100, height: 50}),
        new WebUI.PushButton("Cancel", {width: 100, height: 50}) ]
        }),
```

하나의 컨테이너 안에 버튼 두개를 가로정렬로 나열하였다. column안에 들어간 두 위젯이 묶여서 컨테이너의 정중앙에 위치한다.

결과 분석



레이아웃 위젯을 사용한 위젯 배치는 하나 하나 좌표를 정해줄 필요없이, 각각의 상대적인 위치, 어떤것들이 묶이고, 서로의 상하관계 정도만 지정해주면 레이아웃 위젯이 자동으로 위치를 계한해준다는 점에서 빠르게 위젯을 배치할 수 있다. 특히 위젯이 늘어나면 더욱 효과를 얻을 수 있을 것이다. 또한 앞에서 제시한 컨테이너에 rect를 그려서 컨테이너를 시각적으로 보이게 하는 방법은 디버깅에 있어도 굉장한 효과를 주었다.

2단계: 기본 계산기

구체적인 구현 방법

2번 요구사항은 앞서 진행한 1단계에서 완성한 WebUI를 가지고 기본 계산기를 만드는 일이었다.

initWidget 부분을 살펴보면 전체적인 구조는 각 요소들을 Row에 넣어 정렬하는 것이다. Row 안의 요소들은 컨테이너 안에 적재되어있거나, Column안에 이미 정렬되어 있는것들의 묶음이다.

```
 WebUI.app = new WebUI.Row({
            children: [
```

다음과 같이 Row로 시작한다.
나머지 위젯들은 모두 이 Row의 자식 요소들이다.

```
new WebUI.Container({
                                desired_size: {width: 800, height:
60},
                               horizontal_alignment:
WebUI.Alignment.CENTER,
                               vertical_alignment:
WebUI.Alignment.CENTER,
                               children: [ new WebUI.Text("WebUI
Calculator","blue"),]}),
```

첫 번째 요소는 맨 위의 텍스트 부분이다. 이것은 텍스트 단독 요소이므로 다른 위젯과 맞추기 위해 컨테이너 안에 넣었다. web_ui_v1과 다른 점이 있는데 바로 Text위젯의 생성과정에서 “blue”라는 속성이 추가되어 텍스트 위젯이 생성되었다는 점이다. 이번 과제에서 상단의 제목은 파란색, 계산기의 글씨가 디스플레이 되는 부분은 검은색 글씨로 같은 텍스트 위젯이지만 색이 다르다. 따라서 텍스트 위젯을 조금 수정하여 텍스트의 색상을 인자로 받아 텍스트 객체를 생성하도록 하였다.

변경전

```
WebUI.Text = function(label) {
    this.text_color = 'black';
```

위와 같이 원래의 텍스트 위젯은 무조건 검은색 색상의 텍스트가 출력되게 되어있다. 하지만 변경한 Text 위젯은 color 값을 받아서 각 텍스트마다 색상을 바꿀 수 있게 하였다. 바꾼 코드는 밑과 같다.

변경후

```
WebUI.Text = function(label,color) {
    this.text_color = color;
```

텍스트 위젯은 너비가 800이었다. 모든 요소들이 같은 Row에 포함되는 구조이므로 같은 너비를 800으로 맞춰주었다.

```
 new WebUI.Container({
                                desired_size: {width: 800, height:
60},
                                horizontal_alignment:
WebUI.Alignment.CENTER,
                                vertical_alignment:
WebUI.Alignment.CENTER,

                                children: [

                                new WebUI.Container({
                                    desired_size: {width: 450,
height: 40},
                                    horizontal_alignment:
WebUI.Alignment.LEFT,
                                    vertical_alignment:
WebUI.Alignment.CENTER,
                                    border:true,

                                    children: [

                                new
WebUI.Text(displayValue,"black")]})
```

사용자의 입력이 표시되는 부분이다. 먼저 입력이 표시되는 필드는 계산기의 너비인 800보다 작은 450이다. 하지만 450을 그대로 넣어버리면 다른것들과 너비가 맞지 않아 알맞은 외형이 나오지 않으므로 너비 800의 컨테이너 안의 정중앙에 위치시켰다.

또 컨테이너 위젯생성에서도 이 전의 단계와 다른 변화가 생겼는데
border:true이다.

입력을 나타내주는 부분이 사각형으로 표시가 되어야하는데 일반적인 컨테이너는 테두리가 없기 때문에 나타나지 않는다. 따라서 1단계에서 디버깅 목적으로 만든 컨테이너 테두리에 rect를 그려주는 코드를 변형해서 추가하였다.

앞에서 설명한 코드는 모든 컨테이너에 테두리가 표시되었지만, 이번엔 출력이 표시되는 컨테이너에만 테두리가 나타나면 되기 때문에 다음과 같이 코드를 변경하였다.

```
WebUI.Container.prototype.initVisualItems = function() {
    if(this.border){
    let background = new fabric.Rect({
        left: this.position.left,
        top: this.position.top,
        width: this.desired_size.width,
        height: this.desired_size.height,
        fill: this.fill_color,
        stroke: this.stroke_color,
        strokeWidth: 1,
        selectable: false
    });

    this.visual_items.push(background);
    this.is_resource_ready = true;
}
else     this.is_resource_ready = true;

}
```

Border 속성에 true or false를 받아서 true일 때만 테두리가 표시되고 false일 경우에는 is_resource_ready=true를 반환한다.

다시 돌아와서 위젯 생성 코드를 보면

```
                                        border:true,
```

이처럼 border가 true로 되어있는 것을 알 수 있다. 이 파일에서 이 컨테이너를 제외한 나머지 컨테이너는 border 값을 넣어주지 않는다.
텍스트 위젯 부분은 원래 “”로 감싸져서 출력할 텍스트를 나타내주었지만, displayValue라는 변수를 인자로 주고 색상은 black으로 설정하였다 displayValue변수에 대한 설명은 밑에서 할 예정이다.

버튼 부분은 한 과정을 반복한다.

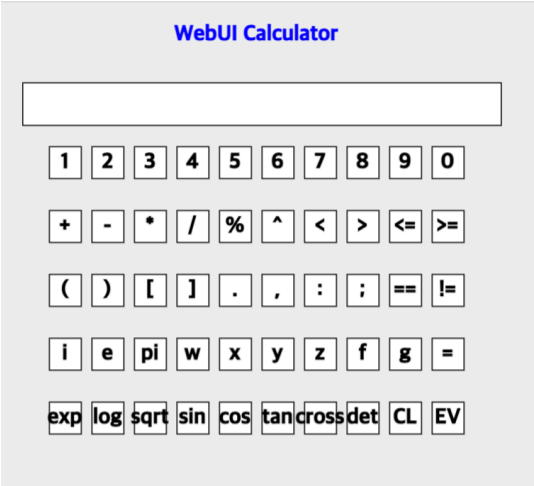
```
 new WebUI.Container({
                    desired_size: {width: 800, height: 60},
                    horizontal_alignment: WebUI.Alignment.CENTER,
                    vertical_alignment: WebUI.Alignment.CENTER,

                    children: [
                new WebUI.Column({ children: [
                    new WebUI.CalcButton("1", {width: 30, height:
30}),
                    new WebUI.CalcButton("2", {width: 30, height:
30}),
                    new WebUI.CalcButton("3", {width: 30, height:
30}),
                    new WebUI.CalcButton("4", {width: 30, height:
30}),
                    new WebUI.CalcButton("5", {width: 30, height:
30}),
                    new WebUI.CalcButton("6", {width: 30, height:
30}),
                    new WebUI.CalcButton("7", {width: 30, height:
30}),
                    new WebUI.CalcButton("8", {width: 30, height:
30}),
                    new WebUI.CalcButton("9", {width: 30, height:
30}),
```

```
                    new WebUI.CalcButton("0", {width: 30, height:
30}),
                ]
            })]}),
```

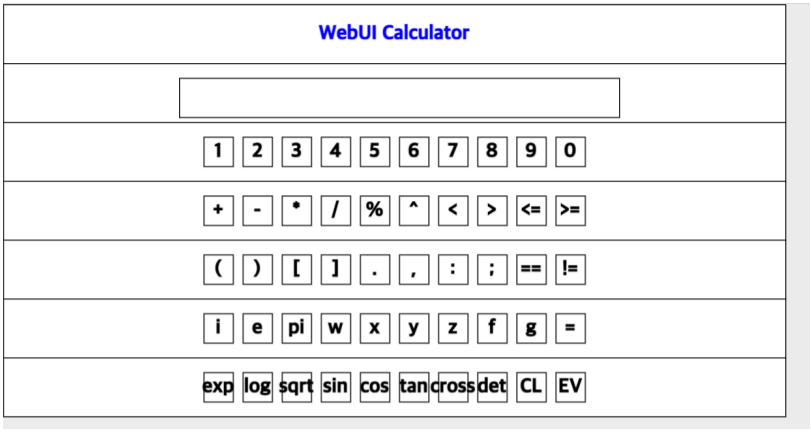
위와 같이 너비 800의 컨테이너 안에 가로로 버튼들을 정렬한 묶음을 넣었다.

이와 같은 너비 800 높이 60의 컨테이너들이 최상위 부모 Row의 자식으로 들어가 세로로 나열된다.

전체적인 모습은 다음과 같다.



위치의 계층구조가 어떻게 되어있는지 보기위해 모든 컨테이너에 테두리를 입혀본 모습은 다음과 같다.



7개의 너비 800, 높이 60의 컨테이너가 세로로 정렬되어 있고, 각 컨테이너 안에 위젯들이 위치한 모습이다. 결과값을 출력해주는 디스플레이 부분은 큰 컨테이너 안에 작은 컨테이너가 있는 모습이다. 버튼 부분은 컨테이너안에 가로배치된 묶음들이 적재되어있다.

전체적인 레이아웃 구조는 다음과 같다.

```
Row {
      Container {
                     Text
                  },
      Container {
                     Container(결과값 출력부분)
                  },
      Container {
                     Column (버튼)
                  },
      Container {
                     Column (버튼)
                  },
      Container {
                     Column (버튼)
                  },
      Container {
                     Column (버튼)
                  },
      Container {
                     Column (버튼)
                  }

}
```

다음은 PushButton을 상속받아 만든 CalcButton이다.

```
WebUI.CalcButton = function(label, desired_size) {
    WebUI.PushButton.call(this);

    this.type = WebUI.WidgetTypes.CALC_BUTTON;

    this.label = label;
    this.desired_size = desired_size;
```

```
    this.onPushed=this.handleButtonPushed;

    }
WebUI.CalcButton.prototype =
Object.create(WebUI.PushButton.prototype);
WebUI.CalcButton.prototype.constructor = WebUI.CalcButton;
```

PushButton의 속성을 상속받고, onpushed에 handleButtonPushed라는 멤버 함수를 넣어주었다.

```
WebUI.CalcButton.prototype.handleButtonPushed=  function(){

    if(displayValue == '0') displayValue = '';

                if(this.label == 'EV')
                {
                    try
                    {
                        displayValue =
parser.eval(displayValue).toString();
                        var tokens = displayValue.split(' ');
                        if(tokens[0] == 'function')
                        {
                            displayValue = tokens[0];
                        }

WebUI.widgets[2].visual_items[0].text=displayValue;

                        displayValue = '0';
                    }
                    catch (e)
                    {

                        if(displayValue != 'function')
                        {displayValue = e.toString();
```

```
WebUI.widgets[2].visual_items[0].text=displayValue;
```

```
                        }displayValue = '0';
                    }
                }
                else
                {
                    if(this.label == 'CL')
                    {
                        displayValue = '0';
```

```
WebUI.widgets[2].visual_items[0].text=displayValue;
                    }
                    else
                    {
                        var temp=displayValue;
                        displayValue += this.label;
```

```
if(displayValue.length>32)displayValue=temp;
```

```
WebUI.widgets[2].visual_items[0].text=displayValue;
                    }

                }
```

```
};
```

handleButtonPushed 함수이다.

```
var displayValue = ''
var parser = math.parser();
```

파일 최상단에 이와같이 계산에 필요한 math.parser()객체와 값을 저장할 displayValue를 정의해주었다.

```
    if(displayValue == '0') displayValue = '';
```

만약 저장된 값이 0이면 아무것도 표현하지 않기 위해 ‘’로 변수를 비워주었다.

```
 if(this.label == 'EV')
                {
                    try
                    {
                        displayValue =
parser.eval(displayValue).toString();
                        var tokens = displayValue.split(' ');
                        if(tokens[0] == 'function')
                        {
                            displayValue = tokens[0];
                        }

WebUI.widgets[2].visual_items[0].set("text",displayValue);
                        displayValue = '0';
                    }
                    catch (e)
                    {
                        if(displayValue != 'function')
                        {displayValue = e.toString();

WebUI.widgets[2].visual_items[0].set("text",displayValue);
                        }displayValue = '0';
                    }
                }
```

값을 계산하는 ‘EV’버튼을 클릭하면 계산을 진행한다. 수식이 저장된 displayValue 변수를 파싱하여 계산하는 과정을 거친다. displayValue에 수식을 저장하는 내용은 이 다음에 나온다.

```
WebUI.widgets[2].visual_items[0].set("text",displayValue);
```

출력을 표시하는 텍스트 위젯이 WebUI.widgets[2]에 위치하였으며 텍스트 위젯의 visual_item[0]이 텍스트 객체이다. 따라서 그 안의 text요소에 접근하여 그 값에 displayValue 변수를 넣어주면 화면에 displayValue에 저장된 값이 표시된다.

```
if(this.label == 'CL')
                    {
                        displayValue = '0';

WebUI.widgets[2].visual_items[0].set("text",displayValue);
```

'CL'버튼은 클리어 버튼이다. displayValue 변수에 있는 값을 0으로 초기화해준다. 표시하는 부분은 앞과 동일하다.

```
else
                    {
                        var temp=displayValue;
                        displayValue += this.label;

if(displayValue.length>32)displayValue=temp;

WebUI.widgets[2].visual_items[0].set("text",displayValue);
            }
                }
```

'EV'와 'CL'을 제외한 나머지 버튼에 대한 부분이다.
나머지 버튼들은 누르는 즉시 버튼의 label이 출력 텍스트에 반영되야 한다.
그 부분이 바로 displayValue += this.label; 이다. 비어있던 스트링 변수에 버튼의 label을 추가해나가는 방식이다. 예를들어 '1', 'sin' ,'2' 버튼을 차례로 누르면 displayValue 는 초기값인 '' 에서 '1', '1sin' , '1sin2' 로 변해간다.

이때 추가로 버튼을 눌렀을 때, 그 길이가 길어짐에 따라 디스플레이 바깥으로 텍스트가 나가는 현상을 고려하여 코드를 구성하였다.

```
var temp=displayValue;
```

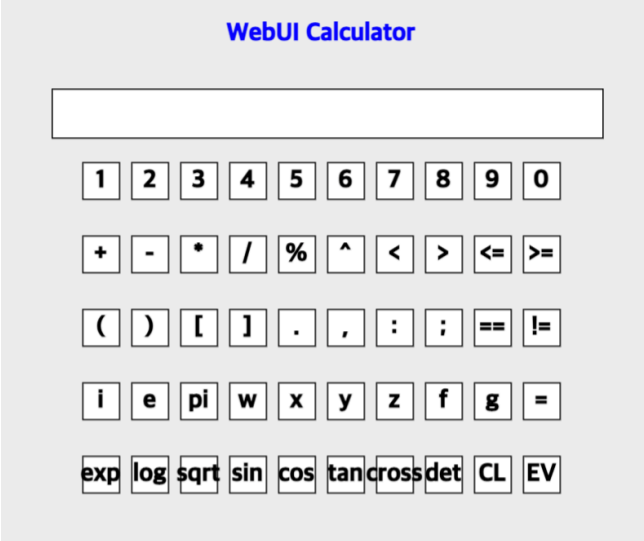
먼저 이 코드로 버튼이 눌리기 이전의 displayValue 값을 저장해둔다.

```
                displayValue += this.label;
```

```
if(displayValue.length>32)displayValue=temp;
```

그리고 displayValue 변수에 눌린 버튼의 label이 반영되어 값이 갱신된다.
이 후, 그 길이가 너무 길면 새로 갱신된 값이 아닌, 이전에 임시로 저장해 두었던 temp를 불러와 displayValue에 다시 저장하면서 값을 되돌린다. 이런 방법으로 텍스트가 디스플레이를 넘어가는 것을 방지하였다.

실행과정 및 결과 분석



각 버튼을 누르면 상단의 디스플레이 바에 표시가 되고 “EV” 을 누르면 입력된 수식이 계산되고, 만약 잘못된 수식을 넣으면 에러가 발생하고 그 에러를 출력해준다. “CL” 버튼을 누르면 수식이 지워진다.
전체적으로 웬만한 계산기의 기능이 전부 구현이 되었지만, 버튼의 위치가 사용자에게 익숙하지 않은 모습이고, 지우기 버튼이 없어, 한번만 실수하더라도 “CL” 버튼으로 모든 수식을 지워야 하므로 큰 부담을 줄 수 있다.

3단계: 확장 계산기

3단계는 2단계의 계산기를 보완하여 계산기를 만드는 단계이다.

새로운 인터페이스/기능 개요

기존의 계산기에서 추가한 기능은 지우기(delete), 그래프 표현이다.
지우기 버튼을 누르면 직전에 입력된 값이 지워지면서 한번 실수하면 수식 전체를 지워야 했던 아쉬움이 있던 2단계 계산기에서 보완했다.
또한 기본적인 사용자 정의 함수를 그래프로 나타낼 수 있다. y=sinx같은 간단한 함수를 그려낼 수 있다.
또한 버튼들의 위치도 사용자 친화적으로 배치했고, 색도 최대한 미적으로 편안함을 느끼도록 디자인하였다.
사용법은 기본 계산기와 동일하나, DEL키를 누르면 직전 입력을 지울 수 있고,
Graph버튼을 한번 누른 후, sin,cos,tan,exp 버튼을 누르고 다시 Graph버튼을 누르면 그래프를 볼 수 있다. 이후 다시 Graph버튼을 누르면 그래프가 지워진다.

새롭게 추가된 위젯들의 구현 방법

먼저 새롭게 추가한 위젯은 GridView, NewCamBtn, GraphPlot,Line 이렇게 총 네개이다.
먼저 그리드 뷰이다.
그리드 뷰는 가로열의 크기를 지정해주면 자동으로 행렬로 위젯들을 배치해준다. 예를들어 그리드뷰의 자식으로 10개의 자식을 넣어주고 가로 크기를 3으로 지정하면
X X X
X X X
X X X
X
이런식으로 3*3행렬과 하나 남는 요소는 그 다음줄의 첫번째 요소로 표현된다.

```
WebUI.GridView = function(properties) {
    WebUI.Widget.call(this, properties);

    this.type = WebUI.WidgetTypes.GRIDVIEW;


}

WebUI.GridView.prototype = Object.create(WebUI.Widget.prototype);
```

```
WebUI.GridView.prototype.constructor = WebUI.GridView;
```

그리드뷰는 기본적으로 WebUI.Widget을 상속받는다.

```
WebUI.GridView.prototype.extendSizeChildren = function(size,
child_size) {
    // IMPLEMENT HERE!
    if(this.colsizenuminput2 <=this.colposnum-1){
    size.width += child_size.width;
if (size.height < child_size.height) size.height =
child_size.height; }

this.colsizenuminput2+=1;
return size;
}
```

그리드뷰의 extendSizeChildren 멤버함수는 기본적으로 Row와 Column의 것과 비슷하다. 하지만 인자로 받은 가로의 크기만큼만 extendSizeChildren를 돌고 카운트를 담당하는 전역변수 colsizenuminput2를 함수가 호출될때 마다 1씩 증가시키고, 변수가 가로 크기에 도달하면 더이상 크기를 변경하는 연산을 수행하지 않는다.

```
WebUI.GridView.prototype.calcNextPosition = function(position,
size) {
    // IMPLEMENT HERE!
    let next_left;
    let next_top;

if(this.colsizenuminput==0){
        firstPosLeft=position.left;
    }
```

다음은 calcNextPosition 함수이다. 이 역시 카운트를 담당하는 전역변수 colsizenuminput를 이용하여 계산한다. 처음 이 함수가 호출 될 때, colsizenuminput 변수는 0이다. 그 때의 position.left를 firstPosLeft에 담는다. 이는 행렬의 왼쪽의 x좌표를 나타낸다. 만약 인자로 넣어준 크기가 3이면 가로로 세개의 위젯을 배치시키고 그 다음줄로 넘어가고 다시 3개의 위젯을 배치시키고 또 다음줄로 넘어가는 과정을 반복하는데, 이때 각 줄의 3개의 위젯이 시작하는 위치는 모두 같아야하고, 그 위치가 firstPosLeft가 된다.

```
this.colsizenuminput+=1;
```

colsizenuminput는 함수가 호출될 때마다 1씩 커진다.

```
  if(this.colsizenuminput <=this.colposnum-1){
            next_left = position.left + size.width;
             next_top = position.top;
```

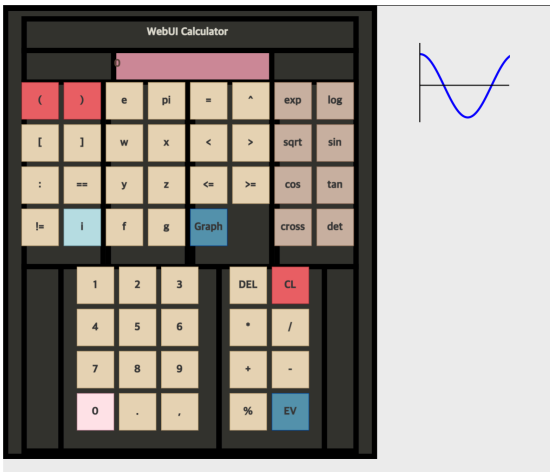
다음 위치를 나타내는 next_left는 그 다음 위젯의 너비만큼 증가하고, 가로로 배치시키는 동안 y좌표는 동일하다. 이제 colsizenuminput가 인자로 넘겨준 행렬의 가로 크기에 도달하면, 다음줄로 넘어가는 과정을 거친다.

```
 else if(this.colsizenuminput >this.colposnum-1){
             next_left=firstPosLeft;
             next_top = position.top + size.height;
            this.colsizenuminput=0;
return {left: next_left, top: next_top};
```

줄을 넘기는 과정이므로 left좌표는 처음에 저장한 firstPosLeft로 돌아가며, top좌표는 여태까지의 값에서 높이를 더해주었다.

이제 WebUI.initWidgets의 전체적인 구조를 먼저 살펴보면 앞에서 정의한 그리드뷰를 컨테이너 안에 넣은것들을 1,2 단계에서 했던것 처럼 정렬했다.

2단계에서 했던것처럼 컨테이너 위젯을 생성할 때, border:true를 선언해주면 각 컨테이너에 테두리가 생긴다. 또한 “fillColor:"rgb(50,50,49)",



처럼 색상을 지정해주면 그 색으로 컨테이너가 채워지고 값을 지정해주지 않으면 채워지지 않는다. 다음은 각 위젯들의 상대적인 정렬 계층구조를 살펴보기위해, 모든 컨테이너에 테두리를 그려준 모습이다.

위 그림으로부터 각 위젯들이 어떤 관계를 가지고 있는지 알 수 있다. 현재 첫번째 줄에는 가로크기가 2인 GridView 위젯 4개가 Column으로 나열되어있고 밑의 줄에 가로크기가 3인 GridView와 2인 GridView가 Column으로 나열되어있는 구조이다.

다음은 새로 정의한 NewCalcBtn이다. 2단계에서 정의한 CalcButton은 PushButton을 상속하고 NewCalcBtn은 CalcButton을 상속받는다.
CalcButton은 색을 아이보리색으로 변경했다. NewCalcBtn은 CalcButton에서 객체 생성시에 배경색, 테두리색, 글자색을 설정할 수 있는 위젯이다.

```
    WebUI.CalcButton.call(this);

    this.type = WebUI.WidgetTypes.NEW_CALCBTN;
    this.label = label;
    this.desired_size = desired_size;

    this.stroke_color = strokeColor;
    this.fill_color = fillColor;

    this.text_color = textColor;
}

WebUI.NewCalcBtn.prototype =
Object.create(WebUI.CalcButton.prototype);
WebUI.NewCalcBtn.prototype.constructor = WebUI.NewCalcBtn;
```

위 코드는 생성자 코드이다.

```
WebUI.NewCalcBtn.prototype.initVisualItems = function() {
    let background = new fabric.Rect({
        left: this.position.left,
```

```
        top: this.position.top,
        width: this.desired_size.width,
        height: this.desired_size.height,
        fill: this.fill_color,
        stroke: this.stroke_color,
        strokeWidth: 1,
        selectable: false
    });


    let text = new fabric.Text(this.label, {
        left: this.position.left,
        top: this.position.top,
        selectable: false,
        fontFamily: this.font_family,
        fontSize:   this.font_size,
        fontWeight: this.font_weight,
        textAlign:  this.text_align,
        stroke:     this.text_color,
        fill:       this.text_color,
    });

    let bound = text.getBoundingRect();
    text.left = this.position.left + this.desired_size.width/2 -
bound.width/2;
    text.top = this.position.top + this.desired_size.height/2 -
bound.height/2;

    this.size = this.desired_size;

    //
    this.visual_items.push(background);


    this.visual_items.push(text);
    this.is_resource_ready = true;
}
```

Rect 객체와 text객체를 생성하여 버튼을 표현하였다.

```
WebUI.CalcButton.prototype.handleButtonPushed=function(){
 else if(this.label == 'DEL'){

displayValue=displayValue.replace(displayTemp[displayTemp.length-
1],"");
                        displayTemp.pop();
                        console.log(displayValue);

WebUI.widgets[2].visual_items[0].set("text",displayValue);
```

CalcButtn에서 HandleButtonPushed 함수가 변경되어 다시 재정의되었다.
del버튼의 추가를 위해서였는데,

버튼을 눌렀을 때,

```
                        displayTemp.push(this.label);
```

코드가 추가되었는데, 이는 각 입력되는 버튼들의 label을 순서대로 displayTemp라는 배열에 넣기 위함이다. DEL버튼을 누르면 displayTemp에서 마지막 요소를 뺀 값들을 다시 displayValue에 넣는다 그러면 가장 마지막에 입력된 값이 지워지는 효과를 얻는다. 이렇게 지운 값은 displayTemp.pop()을 통해 displayTemp에서도 지워준다.

다음은 GraphPlot 위젯이다.

그래프 기능은 모든 그래프를 그리는 것은 힘들다고 판단되어 공학계에서 대표적으로 많이 사용하는 sin, cos, tan, exp를 구현하였다. “Graph” 버튼을 누르면 디스플레이에 “y=“이라고 표시된다. 이후 원하는 함수를 클릭하고 다시 “Graph”버튼을 누르면 그래프가 보여진다.

```
var Arr = new Array();
var xArr = new Array();
var yArr = new Array();
```

순서대로 그래프의 정보를 담는 배열, x축의 정보를 담는 배열, y축 정보를 담는 배열이다.
평소와는 다르게 생성자로 배열 객체를 생성하였다.

```
 children:[new WebUI.Line(),new WebUI.Line(),new WebUI.Line()
```

Line이라는 위젯 세개가 생성되었고 차례로 함수, x축, y축을 담당한다.

```
WebUI.Line = function(label,color) {
    WebUI.Widget.call(this);


}
WebUI.Line.prototype = Object.create(WebUI.Widget.prototype);
WebUI.Line.prototype.constructor = WebUI. Line;

//WebUI.Text.prototype = /* IMPLEMENT HERE! */;
//WebUI.Text.prototype.constructor = /* IMPLEMENT HERE! */;

WebUI.Line.prototype.initVisualItems = function() {
    var line = new fabric.Path('M 650 90 L 700 100');
        line.set({ left: 0, top:
0 ,fill:null,stroke:"black",strokeWidth:0});


        this.visual_items.push(line);
        this.is_resource_ready = true;


}
```

마지막 Line위젯이다. Fabric.path를 시각적으로 보여주는 위젯이고 처음 처음엔 strokeWidth가 0이어서 보이지 않는다.

다음은 “Graph” 버튼을 눌렀을 때의 처리이다.

```
 if(graphClk==0){
                              displayValue = '';

WebUI.widgets[2].visual_items[0].set("text","y=");
                           graphClk=1;

                            }
```

만약 눌렀을 때, 상태를 표시하는 변수인 graphClk가 0이면 처음 눌렸다고 간주한 뒤 y=을 출력하고 상태변수를 1로 변경한다.

```
else if(graphClk==1){
                                graphClk=0;
```

두번째 눌르기 전에는 사용자가 함수를 입력한 후이며 두번째 누르면 상태변수를 0으로 되돌리고 사용자가 입력한 함수를 받아와서 캔버스에 출력한다.

```
else if(graphClk==1){

                                func.push(this.label);
```

그래프 버튼을 처음 누른 다음, 값들을 입력하면 func 배열에 저장된다.

```
 if(func[0]=='sin'){
                                    for(x=500; x<=920; x+=1){
                                        y = 90.0 - Math.sin(1*((x-
590)*Math.PI/90))*60;
                                        if(x==500){
                                            Arr.push(["M", x, y]);
                                        }
                                        else {
                                            Arr.push(["L", x, y]);
                                        }
                                    }
```

다음은 func 배열에 담긴 값이 sin일 때 싸인 함수를 그리는 과정이다. 먼저 싸인함수의 각 xy 좌표값을 계산해서 2차원 배열로 담는다.

cos, tan, exp 모두 동일한 과정을 거친다.

```
  WebUI.widgets[73].visual_items[0].set("path",Arr);
```

```
WebUI.widgets[73].visual_items[0].set("strokeWidth",4);

WebUI.widgets[73].visual_items[0].set("stroke","blue");
```

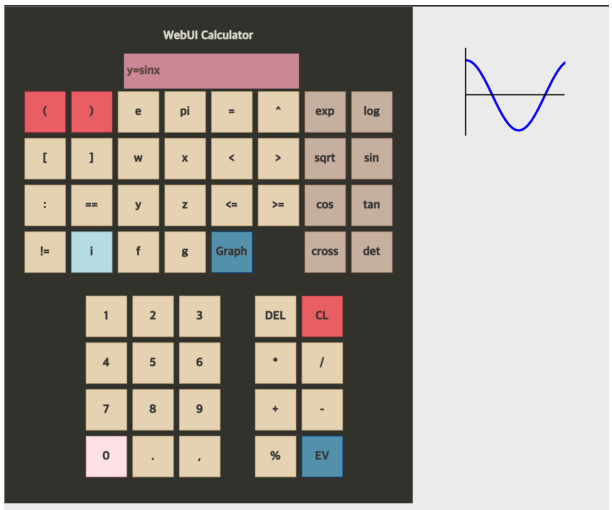
이후 visual_items[0]의 속성값들을 바꿔준다. 특히 path 속성에 함수 좌표들을 담은 배열을 넣어주면 함수가 그려진다. 굵기는 4로 맞췄고 색은 파란색이다.

이후 함수좌표 배열을 초기화한다.

```
 else if(graphClk==2){
                                graphClk=0;

WebUI.widgets[73].visual_items[0].set("strokeWidth",0);

WebUI.widgets[74].visual_items[0].set("strokeWidth",0);

WebUI.widgets[75].visual_items[0].set("strokeWidth",0);

                              }
```

마지막으로 그래프 버튼을 다시 누르면 그래프가 사라진다.

실행과정제시및결과분석



최종 모습은 다음과 같다. 기능별로 위치를 모아놨으며 버튼의 색깔도 중요하다고 생각되는 버튼의 색깔을 따로 표시하였다. 2단계와 차이는 사용자 친화적인 인터페이스, 지우기 기능, 간단한 그래프등이 있다.

논의

평소 우리가 사용하던 웹,어플리케이션 등의 위젯이 어떻게 배치되고, 어떻게 클릭이 이루어지고, 드래그는 어떤 방식으로 이루어지는지 알 수 있었던 과제였다.

먼저 아쉬웠던 점은 0단계에서 스위치 클릭의 정확한 테두리를 잡아내지 못한 점이다. 이전의 과제에서 canvas 클래스의 멤버함수를 통해 테두리를 계산한 알고리즘을 사용하면 추후 보완을 통해 가능할 것이나, 과제에서 제공된 코드를 크게 변경해야하는 점이어서 구현하지 않았다.

두번째는, 계산기에 그래프를 보다 사용자가 원하는대로 그려낼 수 없었다. 이는 먼저 각 함수들의 x축 y축 좌표가 다르기에 모든 경우의 수를 생각했어야 했다는 점이 문제였다. 예를들면 exp 함수같은 경우, x=2~3지점 사이에서 y값이 무한대로 치솟아 오른다. 이 경우 원하는 그래프 모양을 얻기 위해선 그래프를 굉장히 확대해야했고, 삼각함수의 경우 x축과 y축이 정수가 아니다. 그래서 모든 경우에 대비해 단위 변환을 해야한다는 점이 과제에 주어진 시간에 비해 무척 어려운 부분이었다. 하지만 그래프를 그리기 위해 line 위젯의 자식의 속성에 접근하는 방법을 찾아낸 것은 매우 뿌듯한 일이었다.

또한 계산기에 추가 기능들을 더 넣고 싶었으나 시간 관계로 그러지 못한 점이 아쉬웠다.

성공한 부분은 생각보다 뛰어난 디자인을 구현했다는 점이다. 또한 버튼의 눌림에 따라 그래프를 숨기고 나타나게 하는 것을 구현한 점, delete버튼의 기능을 구현하는 과정들에서 프로그래밍적으로 한단계 성장했다는 것을 느낄 수 있었다.