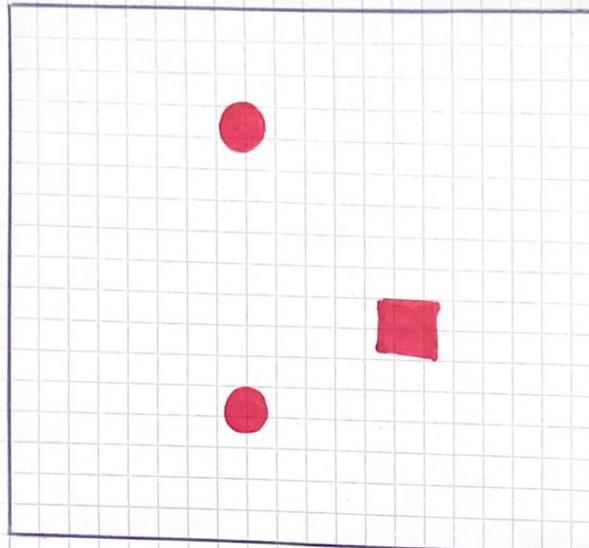


→ id= canvas



→ id= small

400x400 px

→ id= kreis

hinzufügen

→ id= medium

600x600 px

→ id= cube

hinzufügen

→ backgroundcolor

Hintergrund bearbeiten

Canvas spezif.

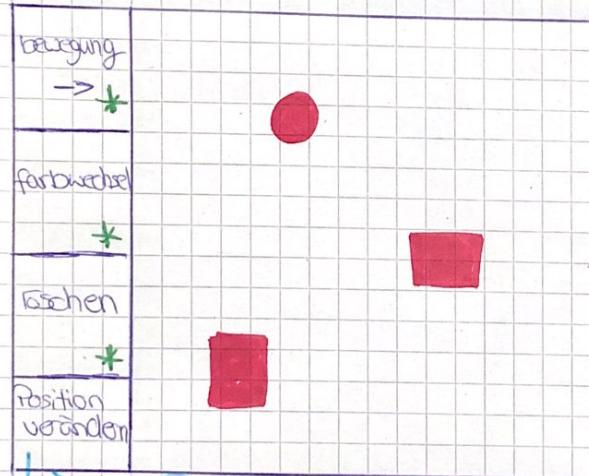
→ id= save

gespeicherte Bilder:

- 1
- 2
- 3

### Interaktion:

1. Bearbeiten der animierten Objekte durch Mausclick auf jeweiliges Objekt



\* Klick fügt dem Element gewählte Animation hinzu oder löscht dieses

↳ Um Position zu ändern, button drücken & Maus gedrückt halten & auf gewünschte Position ziehen

2. Bearbeiten der Hintergrundfarbe

rot



grün

lila

blau

→ bei click auf Button erscheinen im Hintergrund bearbeiten Canvas die zu wählenden Farben

## 1. Konzept Aufbau des Canvas & Buttons

EventListener ("DOMContentLoaded", init)  
EventListener ("mousedown", changeColor)  
EventListener ("mouseup", SetPosition)  
EventListener ("mousemove", MoveObject)

```
export let crc : CanvasRenderingContext2D  
export let SwitchColor : string = "green"  
export let ClientX : number = 0  
export let ClientY : number = 0  
export let CanvasWidth : number = 600  
export let CanvasHeight : number = 600
```

```
let canvas : HTMLCanvasElement;  
let CircleArray : Kreis[] = []  
let NeutralArray : Kreis[] = []  
let AnimatedLeftRight : Kreis[] = []  
let NewPosition : Kreis[] = []  
let AnimatedColor : Kreis[] = []  
let fps : number = 30  
let farbZähler : number = 0  
let isMoving : boolean = false  
let rot : String = red  
let green : String = green  
let purple : String = purple  
let blue : String = blue  
let backgroundColor = blue  
let ChangeBackgroundColor : boolean = false  
let ObjektBearbeiten : boolean = false  
let NeuePosition : boolean = false
```

init

```
Canvas = document.getElementsByTagName("canvas")[0]  
Canvas.width = CanvasWidth  
Canvas.height = CanvasHeight  
crc = Canvas.getContext("2d")  
crc.clearRect(0, 0, CanvasWidth, CanvasHeight)  
crc.rect(0, 0, CanvasWidth, CanvasHeight)  
crc.fillStyle = backgroundColor  
crc.fill()
```

Add Event Listener ()

Update ()

## add Event Listener

```
button.getElementById = "backgroundColor"
addCircle.getElementById = "kreis"
sCanvas.getElementById = "small"
mCanvas.getElementById = "medium"
addCube.getElementById = "cube"
saveImage.getElementById = "save"
```

```
button.addEventListener("click", changeBackgroundColor)
addCircle.addEventListener("click", addNewCircle)
sCanvas.addEventListener("click", smallCanvas)
mCanvas.addEventListener("click", mediumCanvas)
addCube.addEventListener("click", addNewCube)
saveImage.addEventListener("click", saveCanvasImage)
```

## update

```
if (changeBackgroundColor || ObjektBearbeiten || NeuePosition)
```

```
if (switchColor == "green" && farbzähler == geradeZahl)
```

```
switchColor = "red"
```

```
if (switchColor == "red" && farbzähler == ungeradeZahl)
```

```
switchColor = "green"
```

```
window.setTimeout(update, 1000 / fps)
crc.ClearRect
crc.rect(0, 0, canvasWidth, canvasHeight)
crc.fillStyle = backgroundColor
crc.fill
```

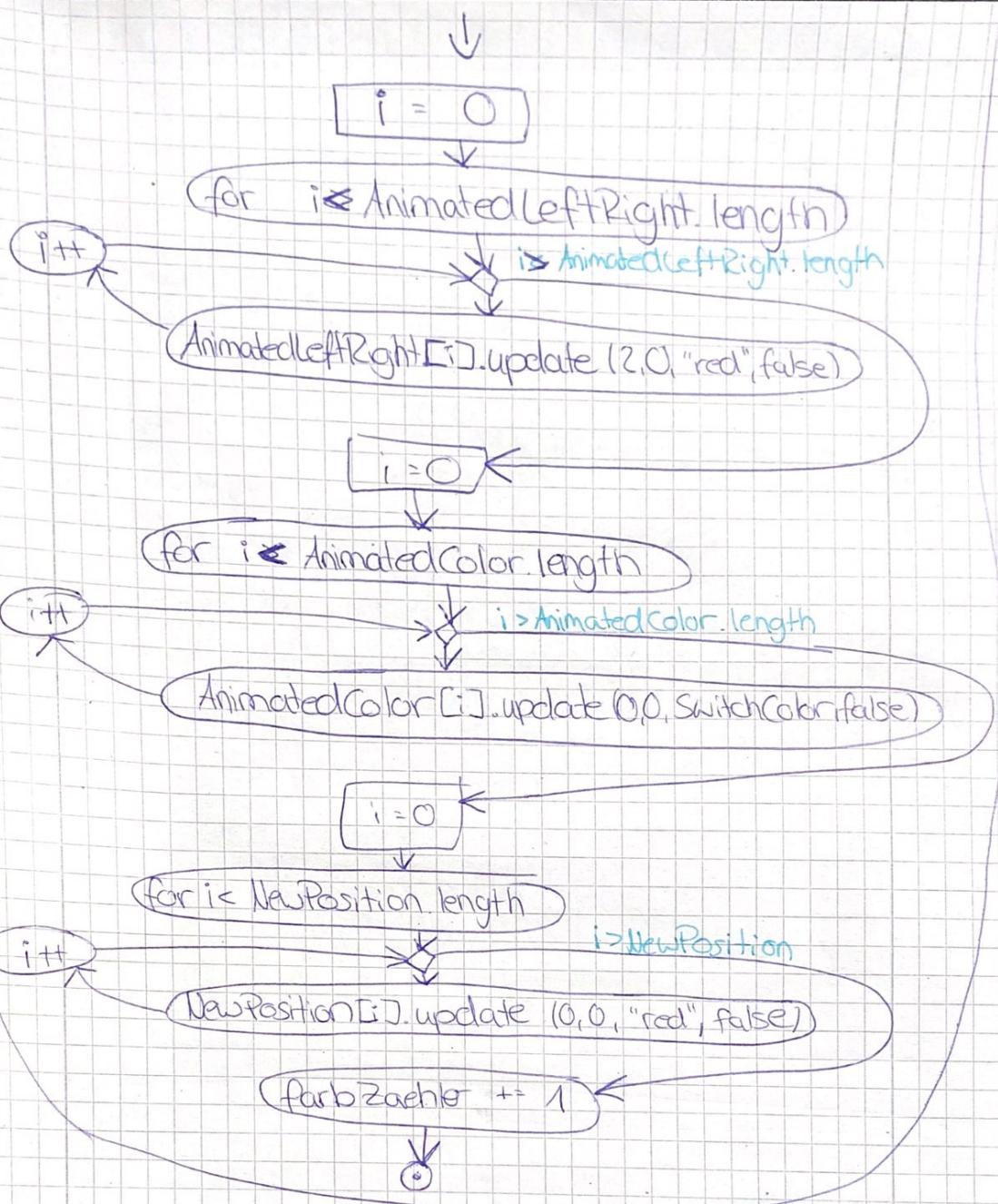
```
i = 0
```

```
for (i < CircleArray.length)
```

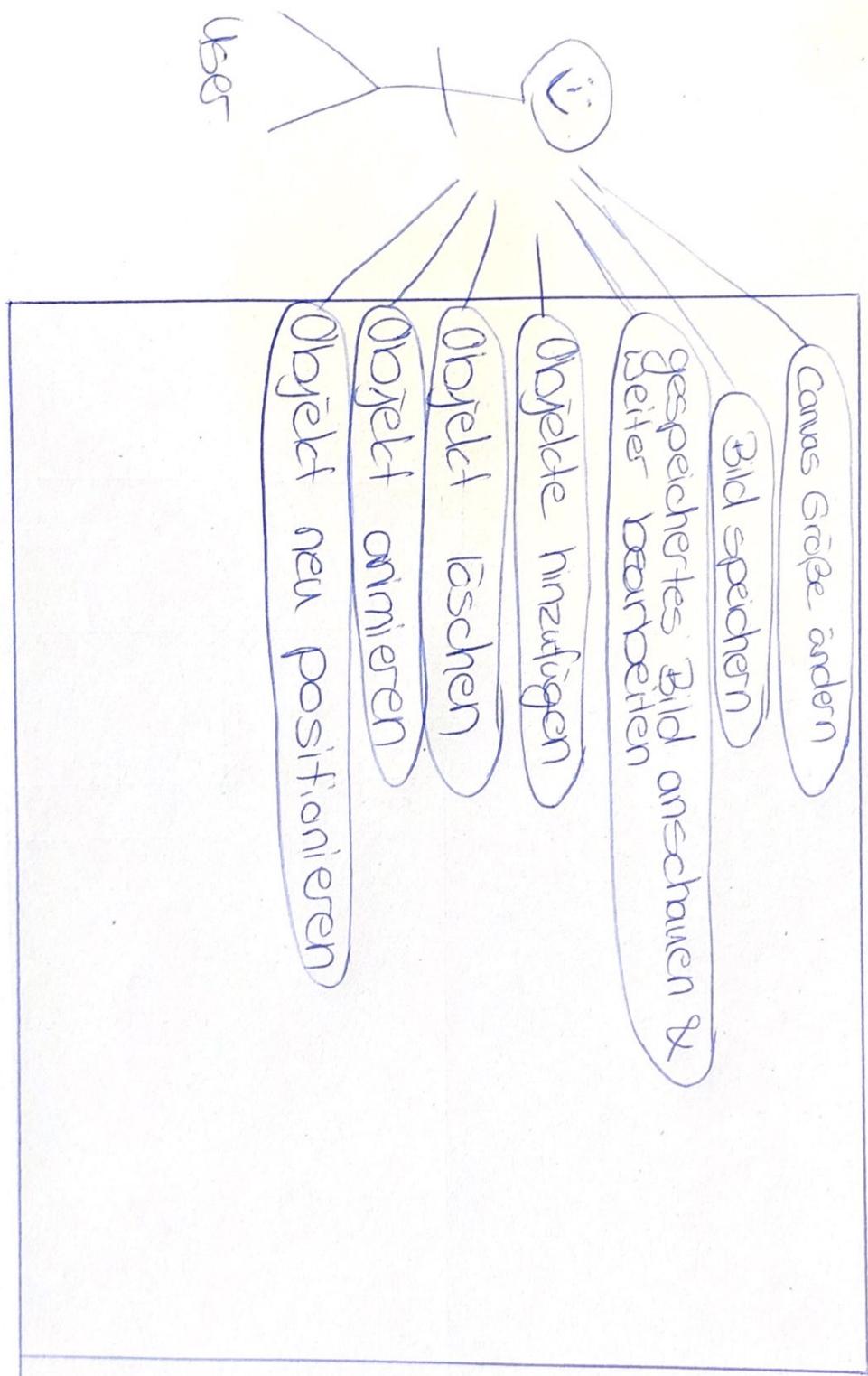
```
i++
```

```
CircleArray[i].update(0, 0, "red", false)
```

\* Weiter auf  
nächster Seite



## Anwendungstalldiagramm

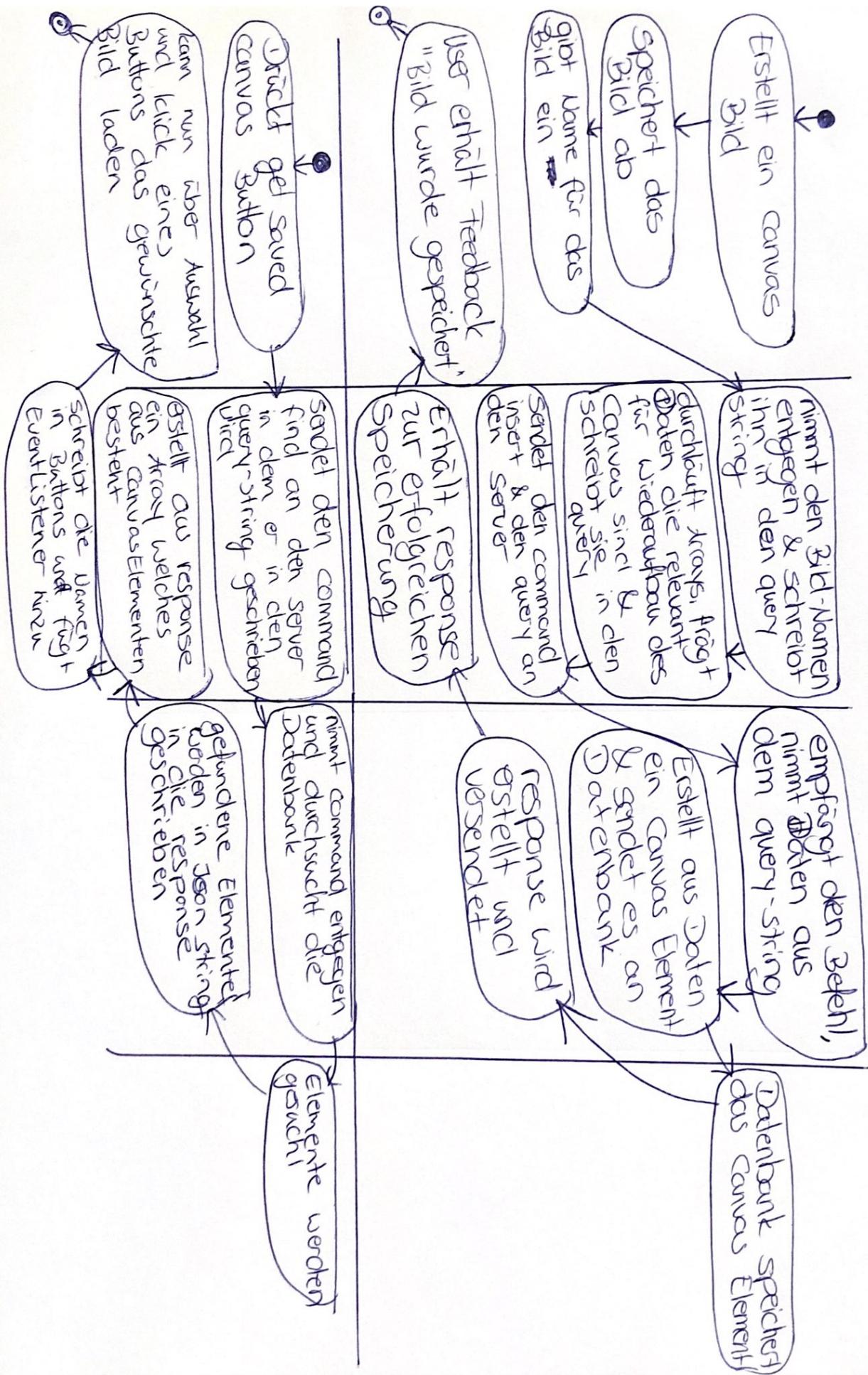


User

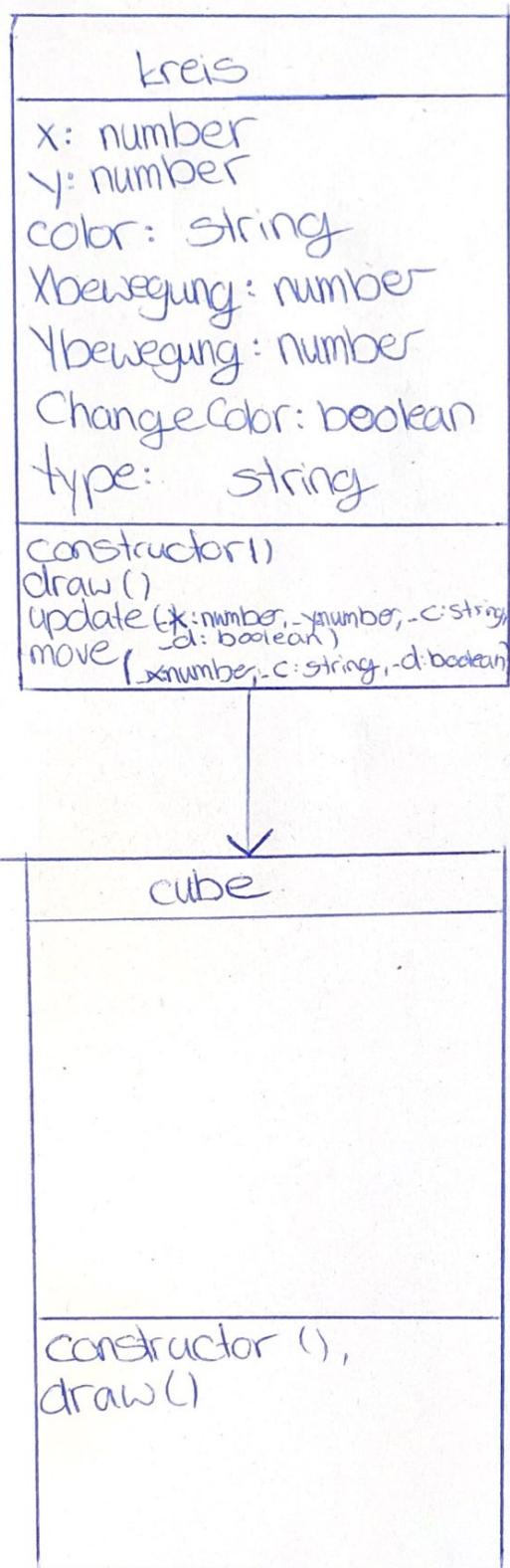
Client

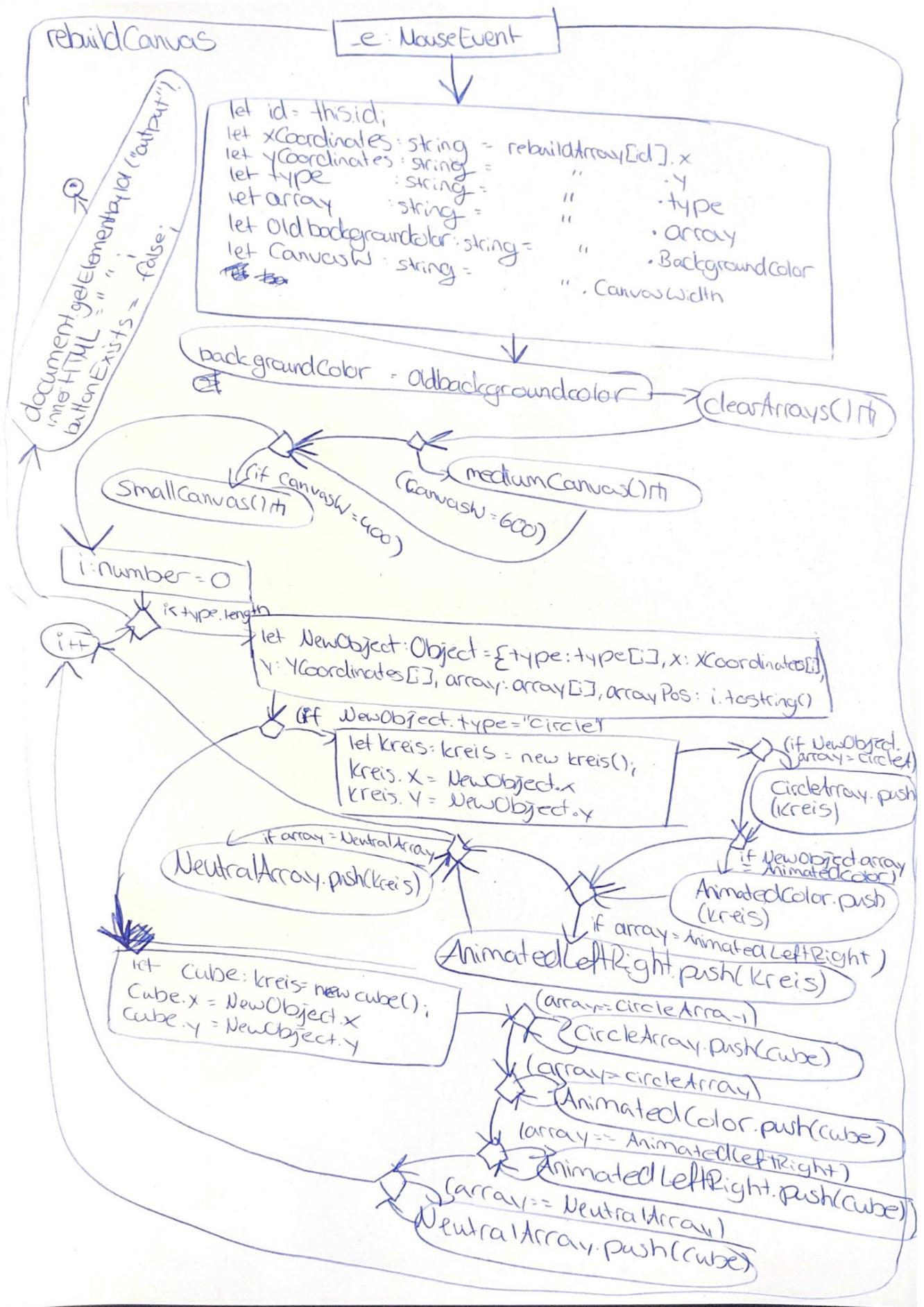
Server

Database



## Klassendiagramme





find

```
let query: string = "command=find"
```

```
sendRequest(query, handleFindResponse) →
```

Der Server liefert uns alle Bilder in einem json string zurück, welchen ich in der handleFindResponse entgegennehme.

handleFindResponse

- event: ProgressEvent

```
let xhr: XMLHttpRequest = (XML... > event.target);
```

(if xhr.readyState == XMLHttpRequest.DONE)

```
rebuildArray = JSON.parse(xhr.response);
```

(if buttonExists == false)

```
let i = 0
```

buttonExists = true;

< rebuildArray.length

```
let button = createElement("button")  
button.innerText = rebuildArray[i].name  
button.addEventListener("click", rebuildCanvas)  
button.setAttribute("id", i)  
document.getElementById("output").  
appendChild(button)
```

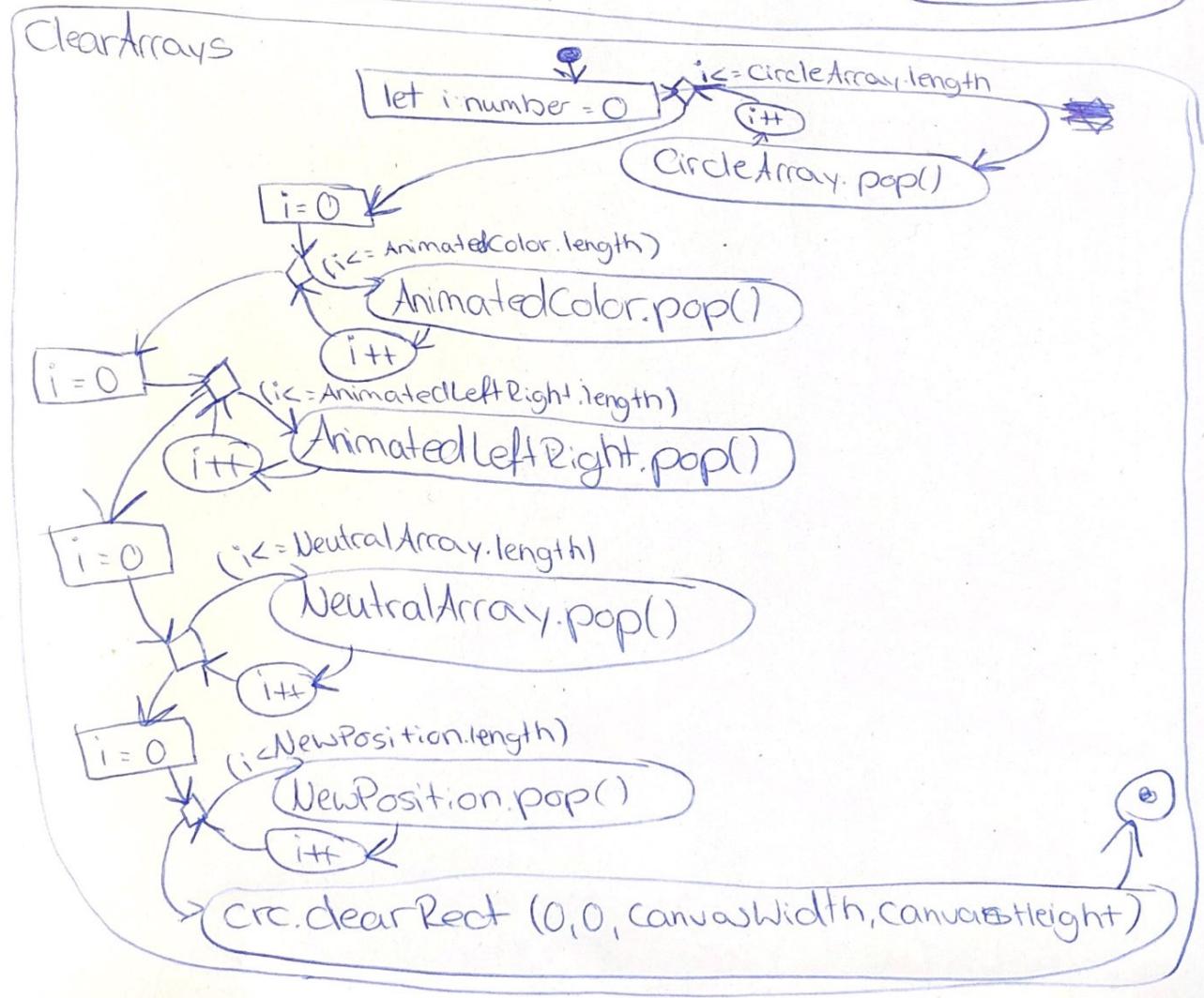
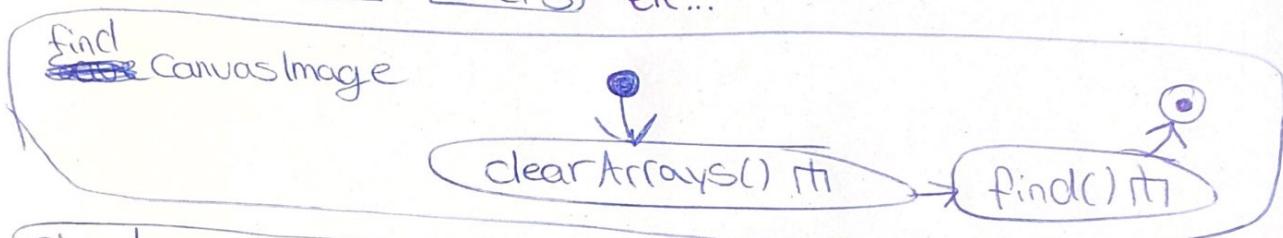
- Die Buttons werden im HTML dargestellt an das div mit der id="output"
- bei Klick wird die Funktion rebuildCanvas aufgerufen

## Konzept Wiederherstellen des Canvas

Dem User steht der Button get saved canvas zur Verfügung. Wenn er diesen klickt, werden die Bilder geladen, welche in der Datenbank gespeichert sind & können ausgewählt werden.

get saved canvas | → id = "save"  
→ eventlistener ruft die Funktion ~~find~~ ~~CanvasImage()~~

2. nach Klick werden die gespeicherten Bilder anhand von Buttons für den User geladen  
→ id = "0"      → id = "1"      → id = "2"  
Bild 1      Bild 2      Bild 3      etc...



## Speichern von Canvas Bildern

Globale Variablen werden angelegt:

```
interface Object {  
    type: string;  
    x: string;  
    y: string;  
    array: string;  
    arrayPos: string; }
```

```
interface CanvasElement {  
    name: string;  
    backgroundColor: string;  
    canvasWidth: string;  
    type: string;  
    x: string;  
    y: string;  
    array: string;  
    arrayPos: string; }
```

```
interface AssocStringString {  
    [key: string]: string; }
```

```
let rebuildArray: CanvasElement[];  
let ElementNum: number = 0;  
let buttonExists: boolean = false;
```

## Konzept zum Speichern von Canvas Bildern

- Dem User steht ein Button zur Verfügung, mit welchem er jederzeit sein Bild abspeichern kann und einen Namen geben kann.

\* Globale Variablen auf Seite: 1

Save canvas

→ id: "save"

→ Der EventListener ruft die Funktion saveCanvasImage auf

SaveCanvasImage

```
let bildName: String = prompt("geben sie einen  
Namen für ihr Bild ein")
```

changeBackgroundColor = true

insert(bildName)

- In der Insert-Funktion bau ich meinen String so zusammen, dass ich alle nötigen Infos über mein Canvas Bild an den Server senden kann

export insert

-name: String

```
let query: String = "Command=insert"
```

```
query += "&name=" + _name;  
query += "&bc" + backgroundColor;  
query += "&cw" + canvas.width.toString()
```

i.number = 0

i++

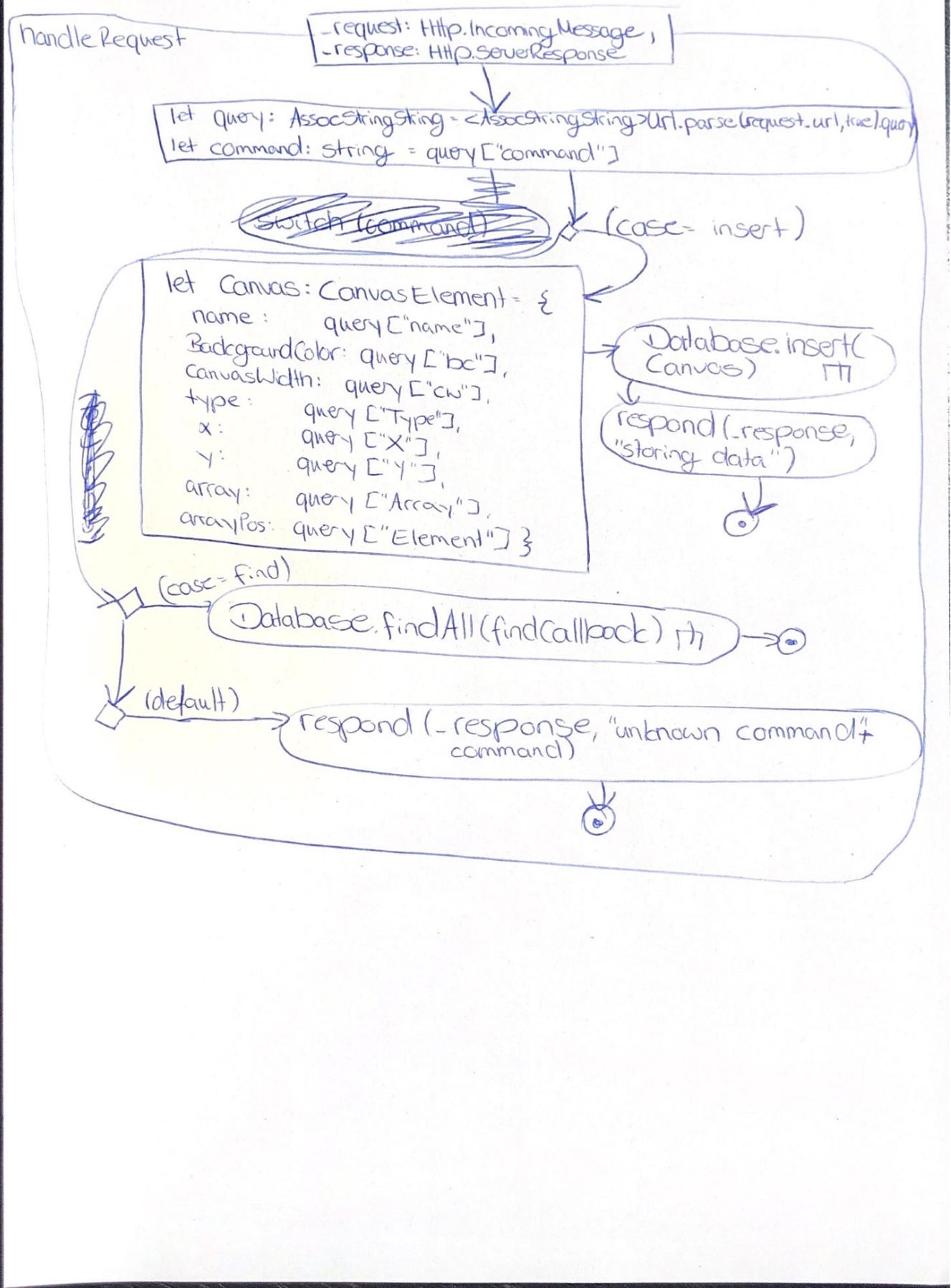
ElementNum += 1;

(is AnimatedLeft+Right, Length)

```
let Element: Object = {  
    type: AnimatedLeftRightCJ.type,  
    x: " " .x.toString(),  
    y: " " .y.toString(),  
    array: "AnimatedLeftRight"  
    arrayPos: " " .ElementNum.toString()}
```

```
Query += "&Element=" + Element.arrayPos + "&Array=" + Element.array + "&Type=" + Element.type + "&x=" +  
Element.x + "&y=" + Element.y;
```

Der Server nimmt in diesem Fall in der handleRequest den command "insert" entgegen, wie es in der URL steht.



export insert

①

i:number=0

(i < circledarray.length)

i > circledarray.length  
i++

```
let Element: Object = {  
    type: CircleArray[i].type,  
    x: " " .x.toString(),  
    y: " " .y.toString(),  
    arrayPos:  
    array: "circlearray" .ElementNum.toString()
```

Element Num += 1

query += "&Element=" + Element.arrayPos + "&Array=" + Element.array + "&Type=" +  
Element.type + "&X=" + Element.x + "&Y=" + Element.y;

i:number=0

(i < NeutralArray.length)

i++

```
let Element: Object = {  
    type: NeutralArray[i].type,  
    x: " " .x.toString(),  
    y: " " .y.toString(),  
    arrayPos: " " .ElementNum.toString(),  
    array: "Neutral Array"
```

ElementNum+=1

query += "&Element=" + Element.arrayPos + "&Array=" + Element.array + "&Type=" +  
Element.type + "&X=" + Element.x + "&Y=" + Element.y;

i:number=0

(i < AnimatedColor.length)

i++

```
let Element: Object = {  
    type: AnimatedColor[i].type,  
    x: " " .x.toString(),  
    y: " " .y.toString(),  
    arrayPos: " " .ElementNum.toString(),  
    array: "AnimatedColor"
```

ElementNum+=1

query += "&Element=" + Element.arrayPos + "&Array=" + Element.array + "&Type=" +  
Element.type + "&X=" + Element.x + "&Y=" + Element.y;

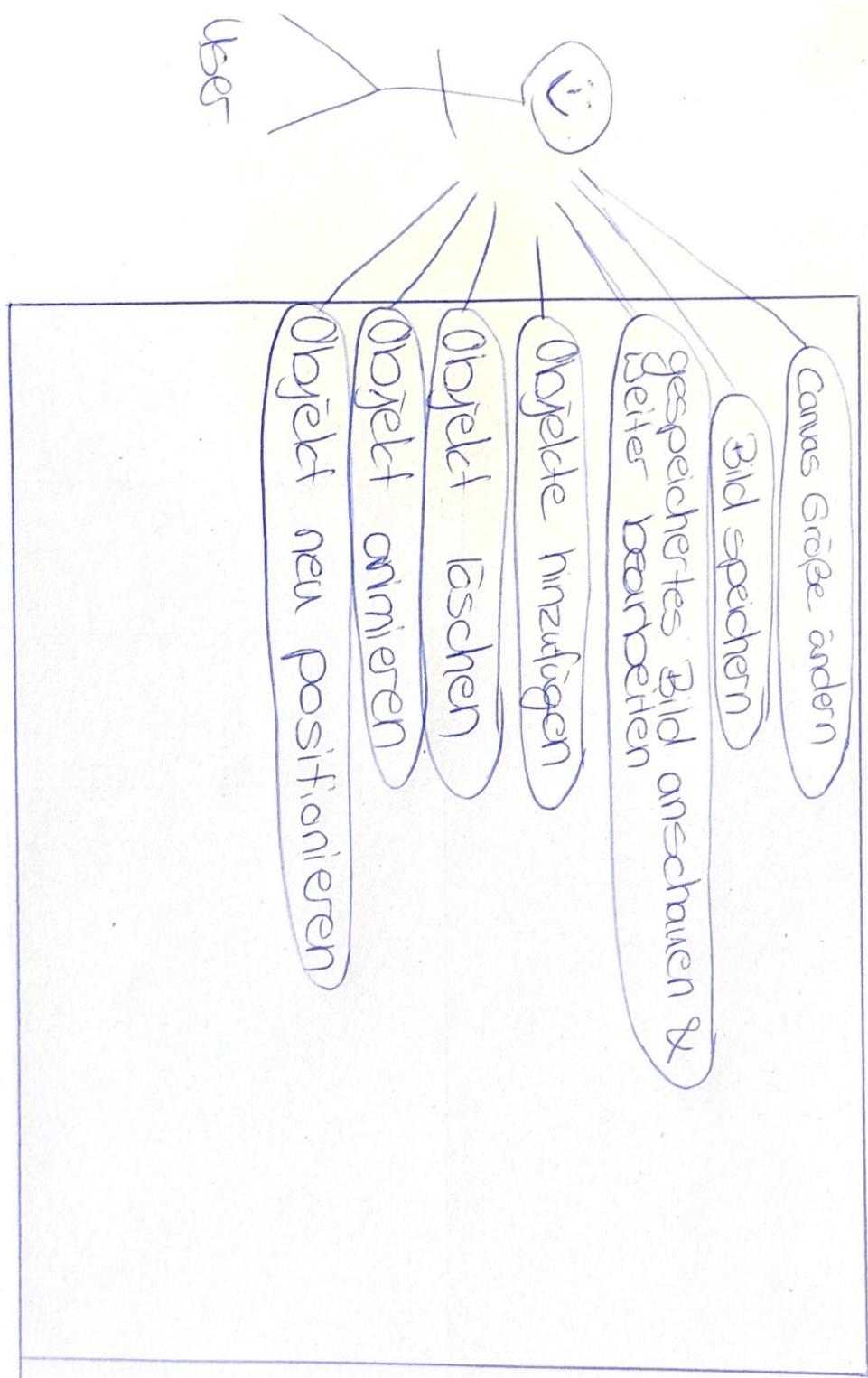
query += "&Anzahl=" + ElementNum

SendRequest(query, handleInsertResponse) ②

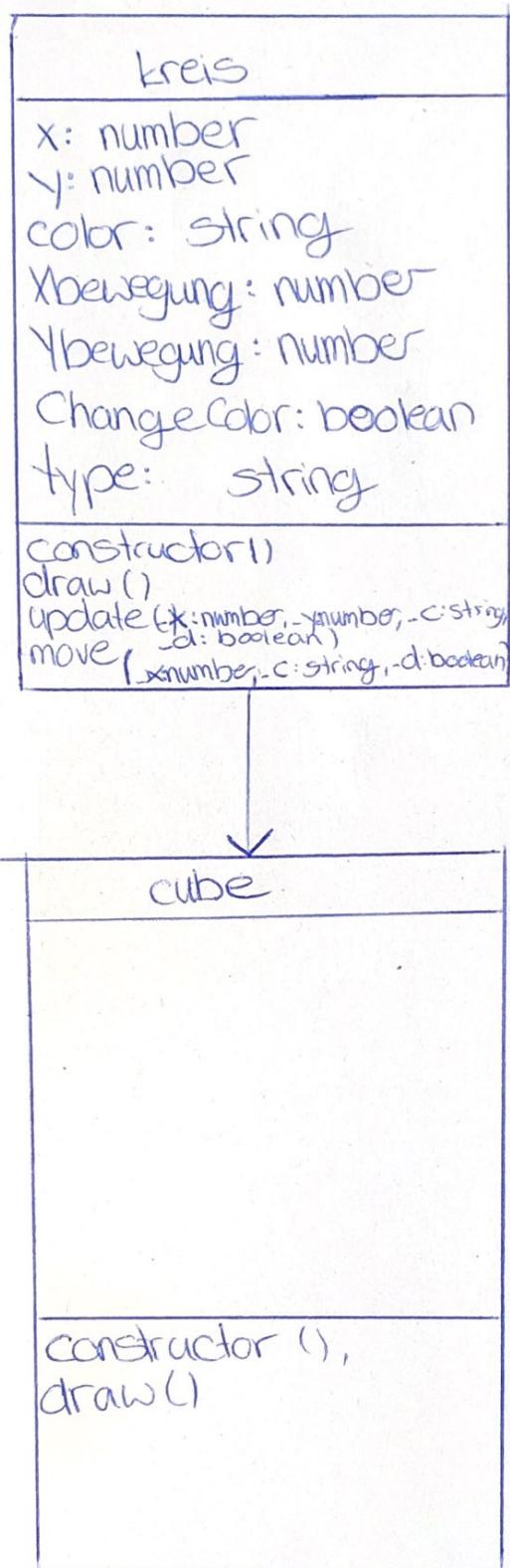
? hier wird der String (query) mit der GET Methode  
an den Server gesendet

③

## Anwendungstalldiagramm



## Klassendiagramme



## Konzept zum Ändern der Hintergrundfarbe

1. Der User kann über den Button Hintergrund bearbeiten die Hintergrundfarbe des Canvas ändern.

Hintergrund  
bearbeiten

→ id = "backgroundColor"

→ Der EventListener ruft die Funktion Change Background auf

ChangeBackground



ChangeBackgroundColor = trace  
farbauswahl () ↗

farbauswahl



```
let redRec: Path2D = new Path2D()  
let greenRec: Path2D = new Path2D()  
let purpleRec: Path2D = new Path2D()  
let blueRec: Path2D = new Path2D()
```

redRec.rect (0,0, canvaswidth/4, canvas Height)  
crc.fillStyle = rot;  
crc.fill (redRec)

greenRec.rect (CanvasWidth/4,0, canvasWidth/4, canvas Height)  
crc.fillStyle = gruen  
crc.fill (greenRec)

purpleRec.rect ((CW/4)+2,0, CW/4, CH)  
crc.fillStyle = purple  
crc.fill (purpleRec)

blueRec.rect ((CW/4)+3,0, CW/4, CH)  
crc.fillStyle = blue  
crc.fill (blueRec)

CW = canvasheight  
CH = canvaswidth

## Konzept zum hinzufügen eines neuen Objekts

- Der User bekommt 2 Buttons mit denen er je einen Kreis oder Vierck hinzufügen kann

 hinzufügen

↳ id = "kreis"  
Der Event-Listener ruft die Funktion addNewCircle() auf

 hinzufügen

↳ id = "cube"  
Der EventListener ruft addNewCube auf

addNewCircle

```
let kreis: Kreis = new Kreis();
CircleArray.push(kreis);
kreis.update(0,0,"red",false);
```

addNewCube

```
let cube: Kreis = new cube();
CircleArray.push(cube);
Cube.update(0,0,"red",false);
```

## Konzept zum Ändern der Canvas Größe

1. Der User bekommt 2 Buttons zur Verfügung gestellt, mit welchen er die Canvas Größe von  $400 \times 400\text{px}$  und  $600 \times 600\text{px}$  ändern kann.

Canvas  
 $400 \times 400\text{px}$

↳ id = "small"  
Hier ruft der EventListener  
smallCanvas auf

Canvas  
 $600 \times 600\text{px}$

↳ id = "medium"  
Hier ruft er mediumCanvas  
auf

SmallCanvas

CanvasWidth = 400  
Canvas Height = 400

canvas.Width = CanvasWidth  
canvas.height = CanvasHeight +

ChangeCanvasSize(CanvasWidth, CanvasHeight) if  
init() if

mediumCanvas

CanvasWidth = 600  
Canvas Height = 600

canvas.width = canvasWidth  
canvas.height = canvasHeight

ChangeCanvasSize(CanvasWidth, CanvasHeight) if  
init() if

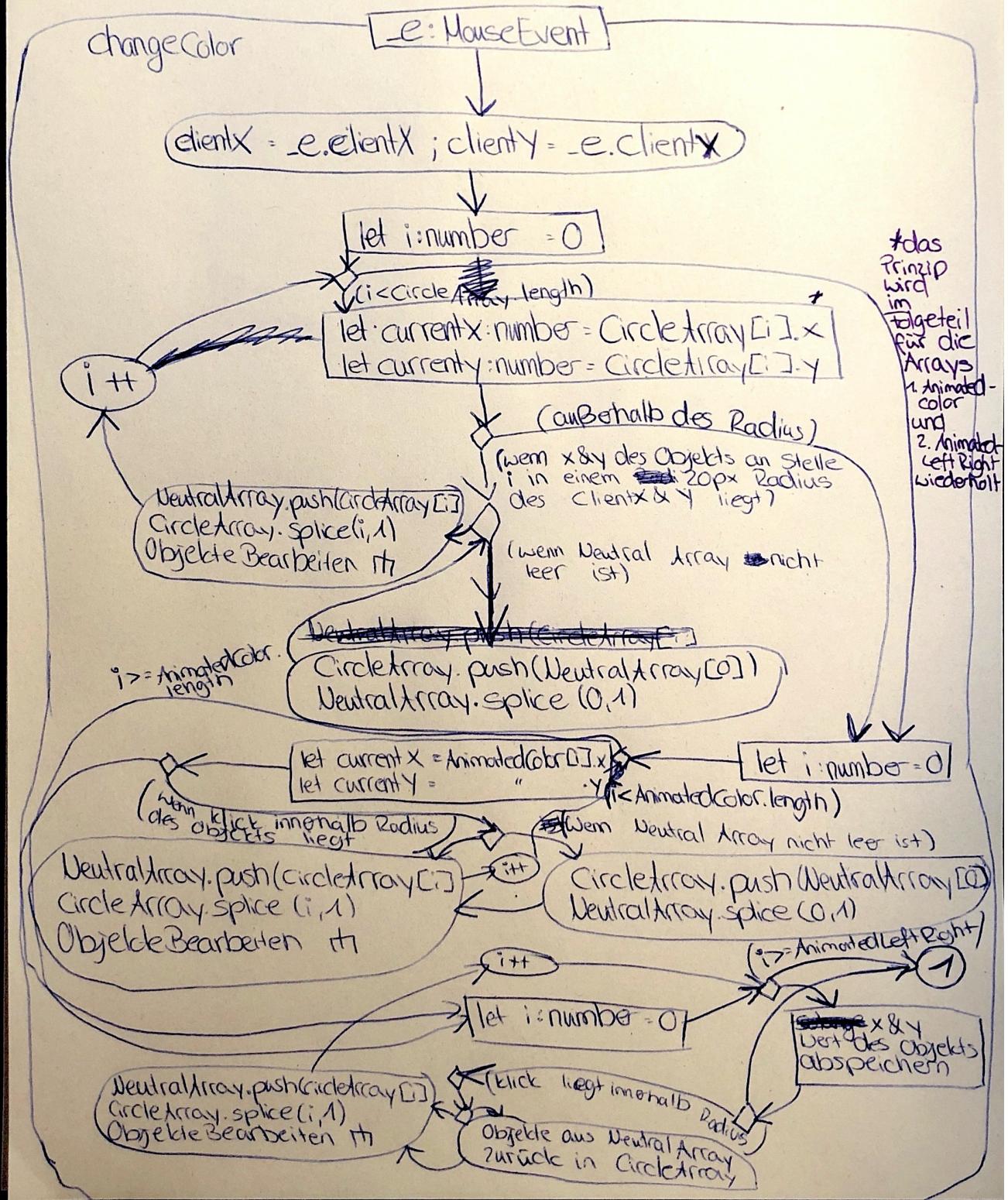
ChangeCanvasSize

-w:number, -h:number

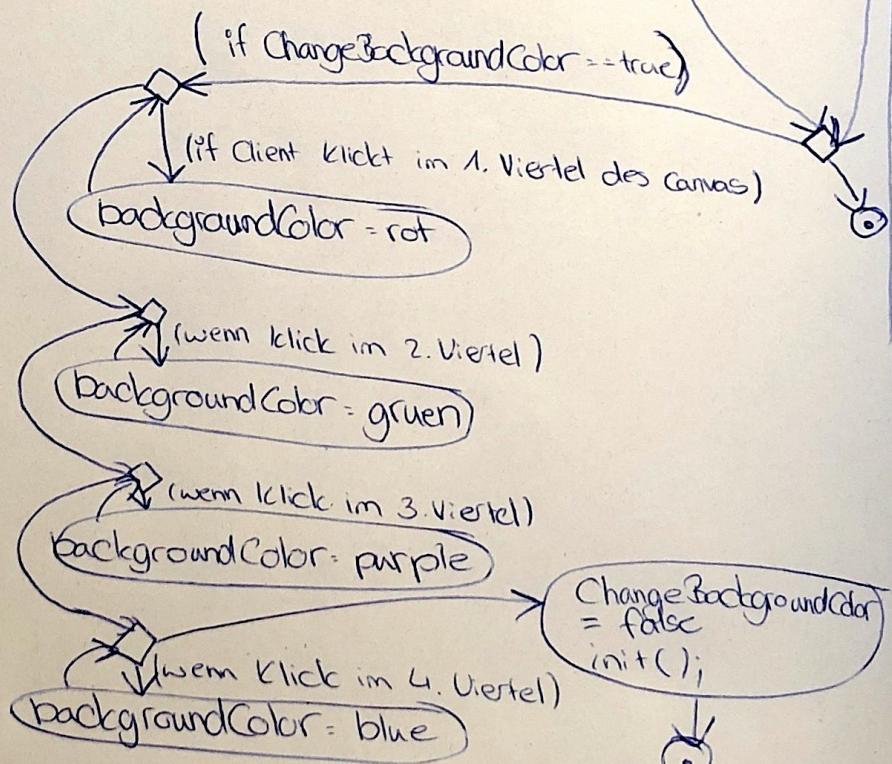
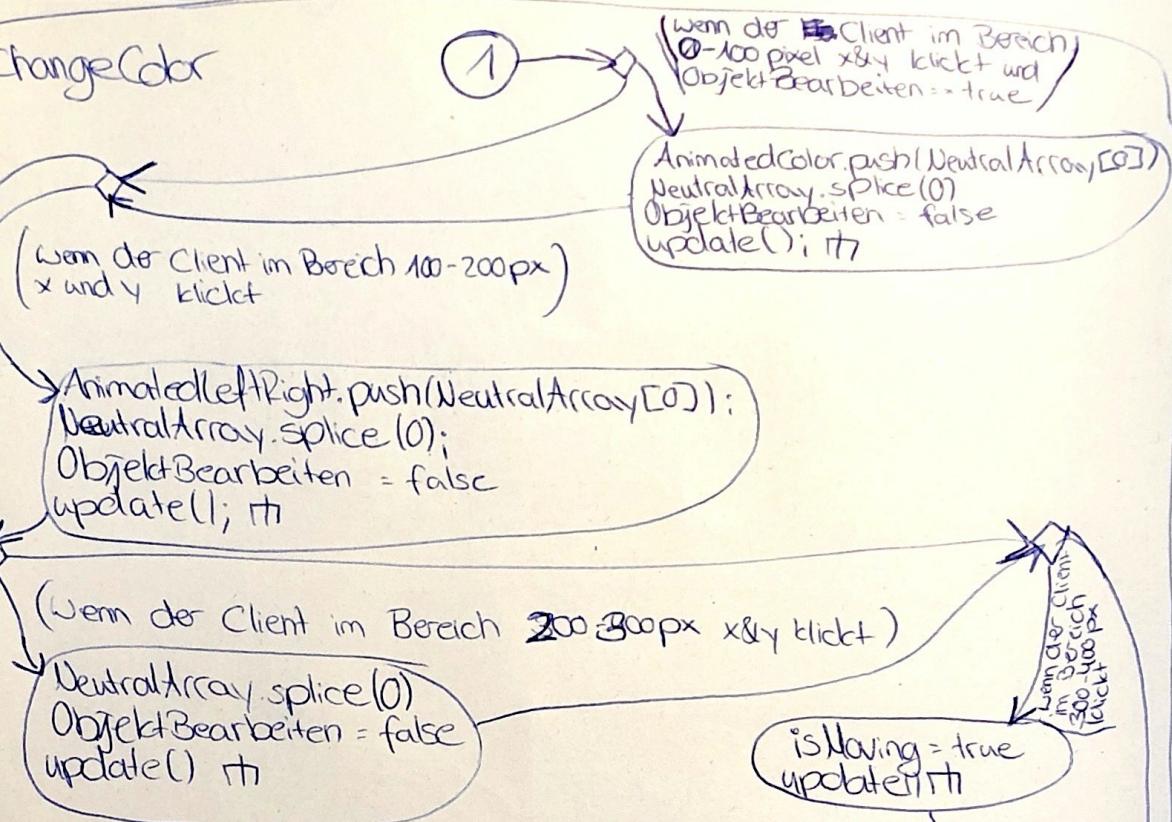
Canvas.getElementById("canvas")[0];  
canvas.Width = -w;  
canvas.height = -h

## Konzept: Objekte bearbeiten

1. Der Mousedown Event-Listener ruft die Funktion changeColor auf, in welcher geschaut wird, welches Element ausgewählt wurde.



## ChangeColor



## Objekte bearbeiten

ObjektBearbeiten = true;

```
let bewegung = new Image();
let farbe = new Image();
let delete = new Image();
let newPosition = new Image();
```

bewegung.src = "move.png"  
crc.drawImage(bewegung, 0, 0, 100, 100)

farbe.src = "farbe.png"  
crc.drawImage(farbe, 0, 100, 100, 100)

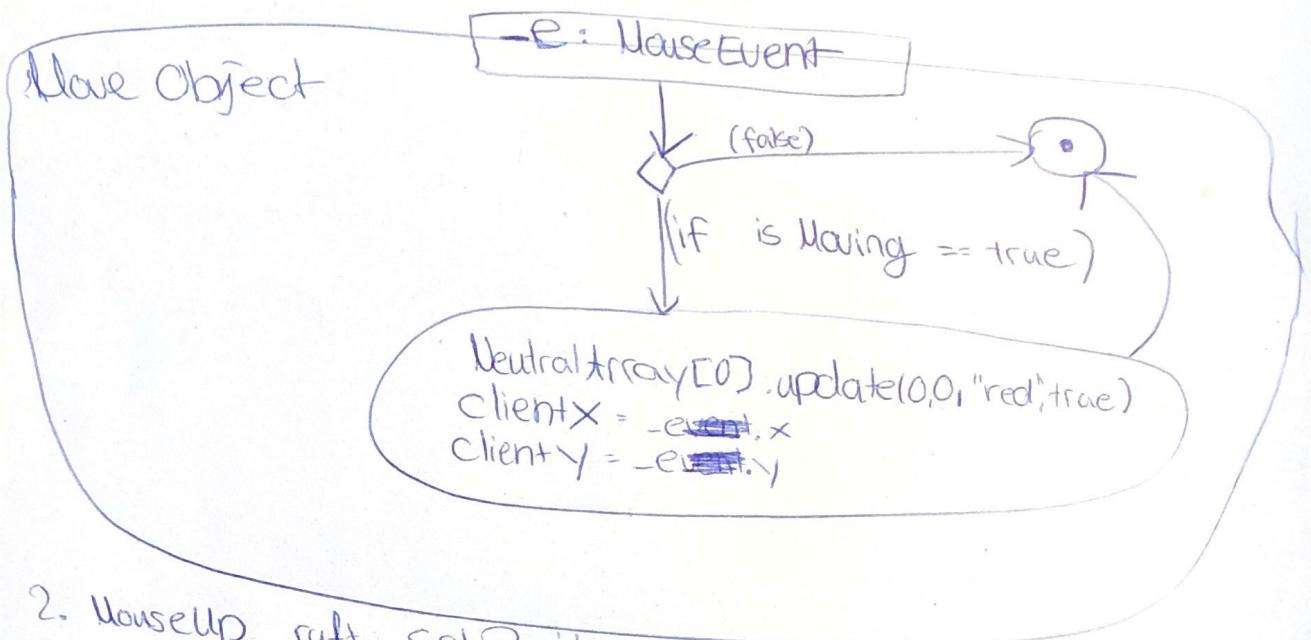
delete.src = "delete.png"  
crc.drawImage(delete, 0, 200, 100, 100)

newPosition.src = "pos.png"  
crc.drawImage(newPosition, 0, 300, 100, 100)

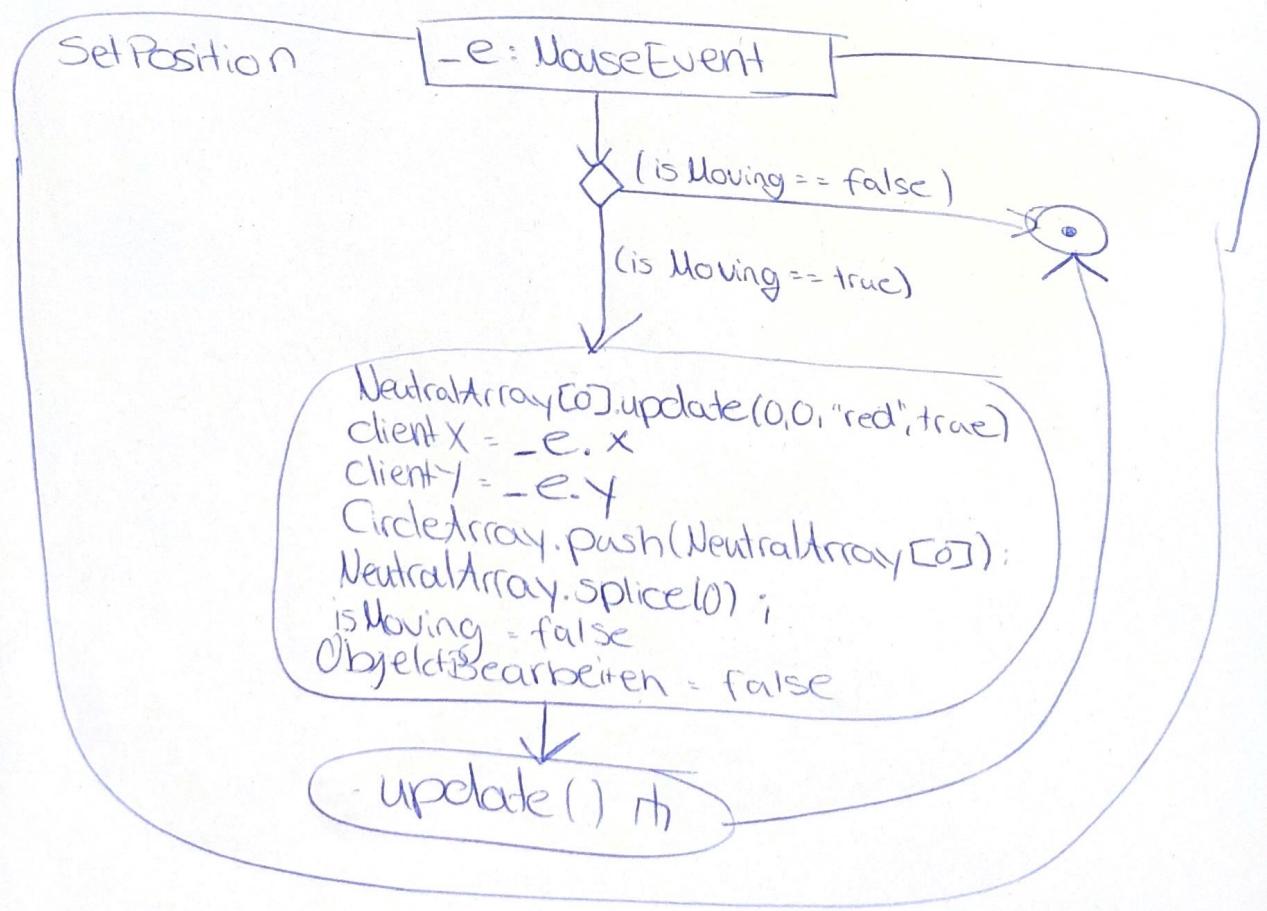
OK

Funktionen um Objekte zu verschieben:

1. Der EventListener MouseMove ruft die Funktion MoveObject auf



2. MouseUp ruft SetPosition auf



## Speichern von Canvas Bildern

Globale Variablen werden angelegt:

```
interface Object {  
    type: string;  
    x: string;  
    y: string;  
    array: string;  
    arrayPos: string; }
```

```
interface CanvasElement {  
    name: string;  
    backgroundColor: string;  
    canvasWidth: string;  
    type: string;  
    x: string;  
    y: string;  
    array: string;  
    arrayPos: string; }
```

```
interface AssocStringString {  
    [key: string]: string; }
```

```
let rebuildArray: CanvasElement[];  
let ElementNum: number = 0;  
let buttonExists: boolean = false;
```

## Konzept zum Speichern von Canvas Bildern

- Dem User steht ein Button zur Verfügung, mit welchem er jederzeit sein Bild abspeichern kann und einen Namen geben kann.

\* Globale Variablen auf Seite: 1

Save canvas

→ id: "save"

→ Der EventListener ruft die Funktion saveCanvasImage auf

SaveCanvasImage

```
let bildName: String = prompt("geben sie einen  
Namen für ihr Bild ein")
```

changeBackgroundColor = true

insert(bildName)

- In der Insert-Funktion bau ich meinen String so zusammen, dass ich alle nötigen Infos über mein Canvas Bild an den Server senden kann

export insert

-name: String

```
let query: String = "Command=insert"
```

```
query += "&name=" + _name;  
query += "&bc" + backgroundColor;  
query += "&cw" + canvas.width.toString()
```

i.number = 0

i++

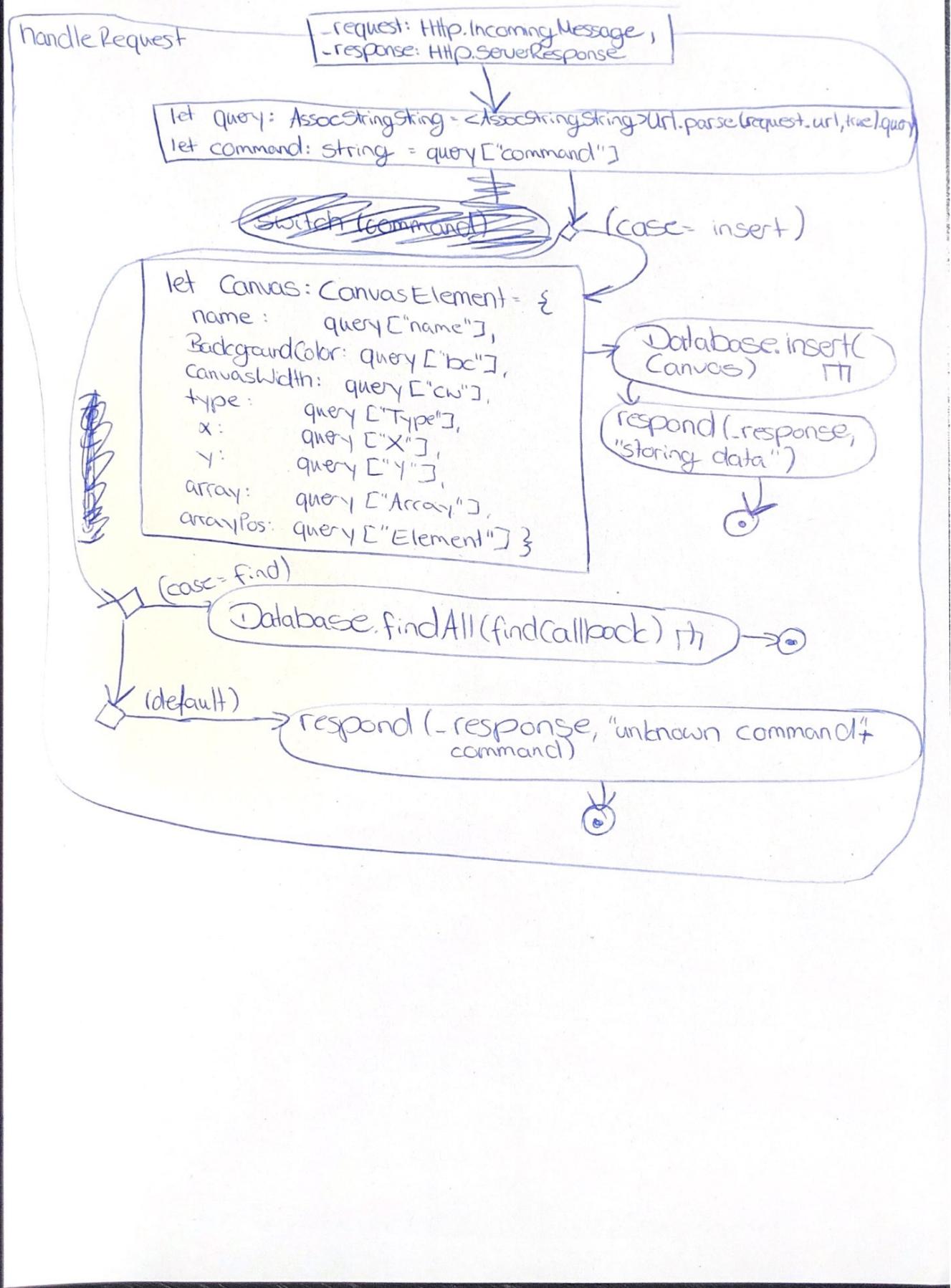
ElementNum += 1;

(is AnimatedLeft+Right, Length)

```
let Element: Object = {  
    type: AnimatedLeftRightCJ.type,  
    x: " " .x.toString(),  
    y: " " .y.toString(),  
    array: "AnimatedLeftRight"  
    arrayPos: " " .ElementNum.toString()}
```

```
Query += "&Element=" + Element.arrayPos + "&Array=" + Element.array + "&Type=" + Element.type + "&x=" +  
Element.x + "&y=" + Element.y;
```

Der Server nimmt in diesem Fall in der handleRequest den command "insert" entgegen, wie es in der URL steht.



export insert

①

i:number=0

(i < circledarray.length)

i > circledarray.length  
i++

```
let Element: Object = {  
    type: CircleArray[i].type,  
    x: " " .x.toString(),  
    y: " " .y.toString(),  
    arrayPos:  
    array: "circlearray" .ElementNum.toString()
```

Element Num += 1

query += "&Element=" + Element.arrayPos + "&Array=" + Element.array + "&Type=" +  
Element.type + "&X=" + Element.x + "&Y=" + Element.y;

i:number=0

(i < NeutralArray.length)

i++

```
let Element: Object = {  
    type: NeutralArray[i].type,  
    x: " " .x.toString(),  
    y: " " .y.toString(),  
    arrayPos: " " .ElementNum.toString(),  
    array: "Neutral Array"
```

ElementNum+=1

query += "&Element=" + Element.arrayPos + "&Array=" + Element.array + "&Type=" +  
Element.type + "&X=" + Element.x + "&Y=" + Element.y;

i:number=0

(i < AnimatedColor.length)

i++

```
let Element: Object = {  
    type: AnimatedColor[i].type,  
    x: " " .x.toString(),  
    y: " " .y.toString(),  
    arrayPos: " " .ElementNum.toString(),  
    array: "AnimatedColor"
```

ElementNum+=1

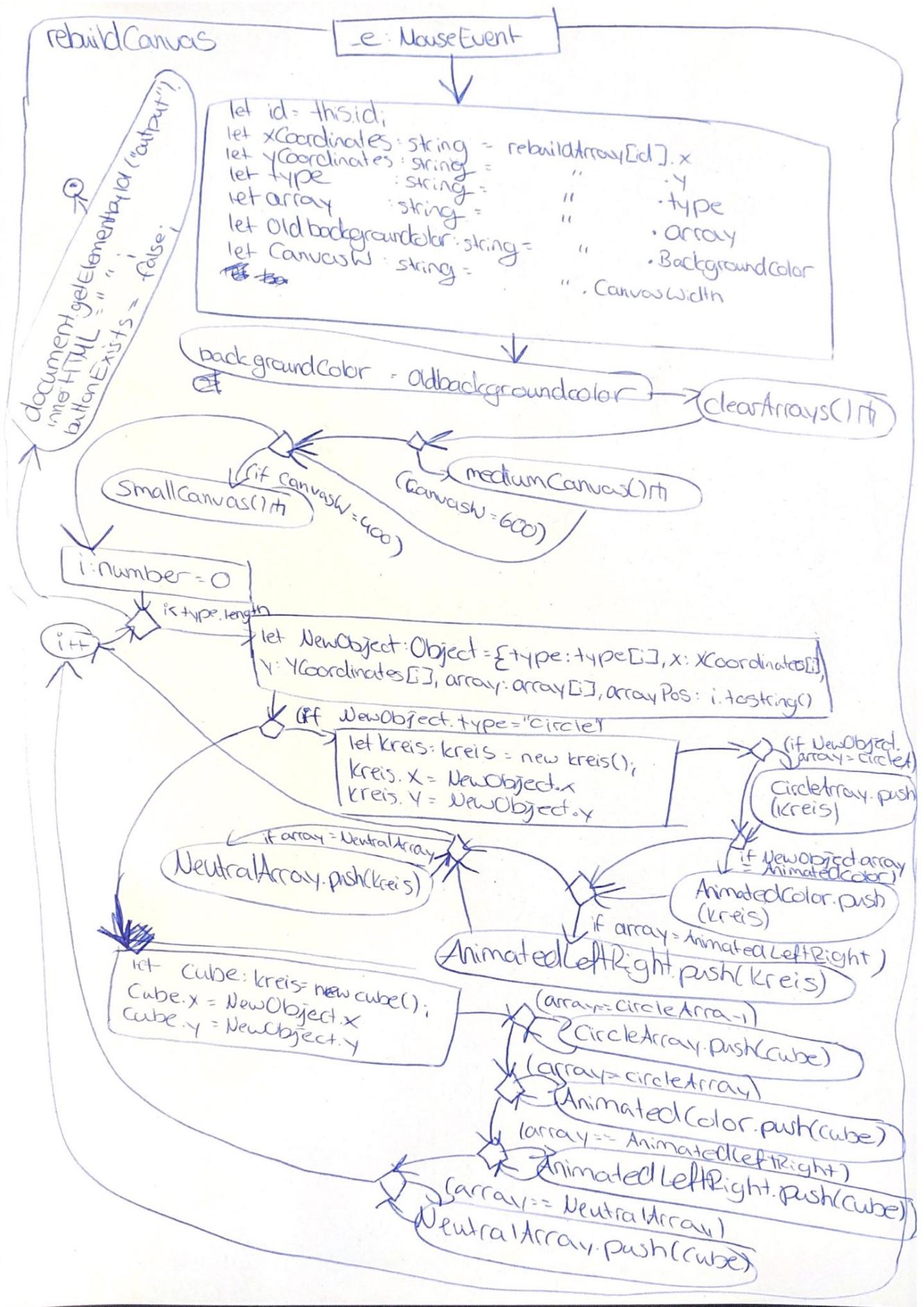
query += "&Element=" + Element.arrayPos + "&Array=" + Element.array + "&Type=" +  
Element.type + "&X=" + Element.x + "&Y=" + Element.y;

query += "&Anzahl=" + ElementNum

SendRequest(query, handleInsertResponse) // ②

? hier wird der String (query) mit der GET Methode  
an den Server gesendet

②



find

```
let query: string = "command=find"
```

```
sendRequest(query, handleFindResponse) →
```

Der Server liefert uns alle Bilder in einem json string zurück, welchen ich in der handleFindResponse entgegennehme.

handleFindResponse

- event: ProgressEvent

```
let xhr: XMLHttpRequest = (XML... > event.target);
```

(if xhr.readyState == XMLHttpRequest.DONE)

```
rebuildArray = JSON.parse(xhr.response);
```

(if buttonExists == false)

```
let i = 0
```

buttonExists = true;

< rebuildArray.length

```
let button = createElement("button")  
button.innerText = rebuildArray[i].name  
button.addEventListener("click", rebuildCanvas)  
button.setAttribute("id", i)  
document.getElementById("output").  
appendChild(button)
```

- Die Buttons werden im HTML dargestellt an das div mit der id="output"
- bei Klick wird die Funktion rebuildCanvas aufgerufen

## Konzept Wiederherstellen des Canvas

Dem User steht der Button get saved canvas zur Verfügung. Wenn er diesen klickt, werden die Bilder geladen, welche in der Datenbank gespeichert sind & können ausgewählt werden.

get saved canvas | → id = "save"  
→ eventlistener ruft die Funktion ~~find~~ ~~CanvasImage()~~

2. nach Klick werden die gespeicherten Bilder anhand von Buttons für den User geladen  
→ id = "0"      → id = "1"      → id = "2"  
Bild 1           Bild 2           Bild 3      etc...

