backgroundColor

id="myCanvas"

400 || 600 || 800 width

id="blue" | id=white | id=grey | id=green } button

editable

⊙ id="edit" ⊙ id="x" } radiobuttons / gleiche Radiogruppe

canvas

⊙ id="small" ⊙ id="medium" ⊙ id="large" } radiobuttons / gleiche Radiogruppe

circle

placeCircle
id="circle"

} button → platziert Kreise auf Canvas

Color Change

⊙ id="purple" ⊙ id="pink" ⊙ id="blue" } radiobuttons / gleiche Radiogruppe

Rectangle

placeRectangle
id="rect"

} button → platziert Vierecke auf Canvas

Color Change

⊙ id="purple1" ⊙ id="pink1" ⊙ id="blue1" } radiobuttons / gleiche Radiogruppe

animations

} in dieses Div werden die buttons von der function "createAnimationButtons" geschrieben

save and load

id="save" } Button

saved pictures

} buttons

id = "myCanvas"

canvas

600px

600px

click

background-color

blue | white | gray | green

} fieldset } buttons

edit selected shape

○ edit ○ not edit

} radiobutton } fieldset

size of canvas

○ small ○ medium ○ large

} radiobutton } fieldset

Circle

place circle

drag | delete | rainbow

○ purple ○ pink ○ blue

} button Form erstellen
} buttons "animation"
} radiobuttons Farbe Form

} fieldset

Rectangle

place rect

drag | delete | move

○ purple ○ pink ○ blue

} button Form erstellen
} buttons "animation"
} radiobuttons Farbe Form

} fieldset

## Mainclass

x: number
y: number
dx: number
dy: number
a: number
size: number
color: number
rainbow: boolean
type: string
moving: boolean
newColor: boolean

draw()
update()
move()
constructor()

## Cube

x = Math.random()*crccanvas.width.
y = Math.random()*crccanvas.height.
dx = 0
dy = 0
a = 1
size = 8
color = "SteelBlue" || "pink" || "purple"
rainbow = false
newColor = false
type = "circle"
moving = false

constructor()
draw()
move()
update()

## Rect

x=Math.random()*canvas.width.
y=Math.random()*canvas.height
dx = 0
dy = 0
a = 1
size = 8
color = "SteelBlue" || "pink" || "purple"
rainbow = false
type = "rec"
moving = false

constructor()
draw()
move()
update()

User/Player

Client

Server

Datenbank

Bild wird durch verschiedene Buttons erstellt, Canvas wird gestaltet

save-Button wird gedrückt

storing data ok drücken

(name eingegeben)

input-Daten werden entgegengenommen

Input-Daten werden in query-string gespeichert

request + query an Server schicken

request wird empfangen

Daten aus query-string verarbeitet

wird in Datenbank abgespeichert

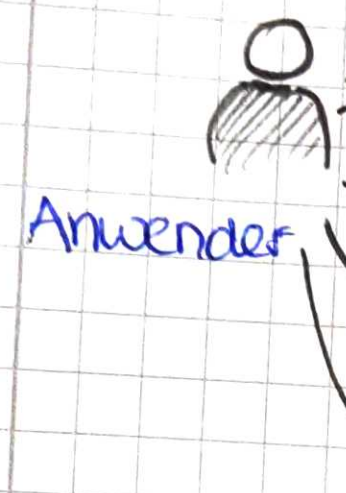
response wird erstellt

response wird abgeschickt

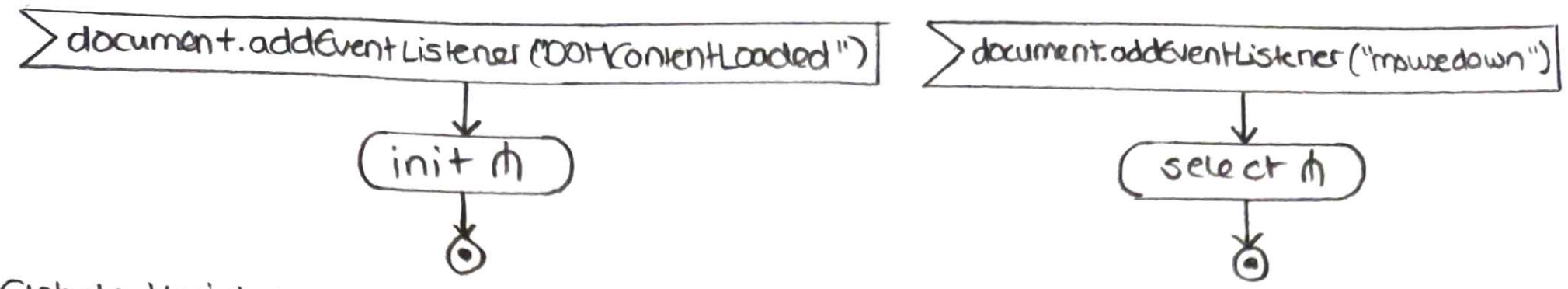
name des Bildes auf Button geschrieben

auf Knopfdruck kann das gespeicherte Bild abgerufen und weiterbearbeitet werden

# Anwendungsfalldiagramm




- Anwender kann Kreise/Formen platzieren
- Formen können bewegt werden (dragging)
- Formen können animiert werden (farbdurchlauf oder Bewegung nach rechts)
- Farbe des Hintergrunds und der Kreise bzw. Rechtecke kann durch Farbauswahl angepasst werden
- man kann die Größe der Malfläche einstellen
- man kann Bilder abspeichern und sie zu einem späteren Zeitpunkt weiterbearbeiten




Globale Variabeln + Arrays

```
export let crc: CanvasRenderingContext2D;
export let circleArray: Circle[] = []
export let changedArray: Circle[] = []
export let colorArray: Circle[] = []
export let moveArray: Circle[] = []
export let deleteArray: Circle[] = []
export let backgroundColor: string = "white"
export let canvas: HTMLCanvasElement

let pushedObject: number
let fps: number = 30
let imageData: ImageData
let draggedObject: boolean = false
let boolean: boolean = false
let buttonsAreCreated: boolean = false
```

init

find() ⇑

canvas = document.getElementByTagName("canvas")[0];

crc.canvas.getContext("2d")

imageData = crc.getImageData(0,0, canvas.width, canvas.height

let buttonBlue: HTMLButtonElement=<HTMLButtonElement>
document.getElementById("blue")

buttonBlue.addEventListener("click", background ⇑)

let buttonWhite: HTMLButtonElement=<HTMLButtonElement>
document.getElementById("white")

buttonWhite.addEventListener("click", background01 ⇑)

let buttonGrey: HTMLButtonElement=<HTMLButtonElement>
document.getElementById("grey")

buttonGrey.addEventListener("click", background02 ⇑)

let buttonGreen: HTMLButtonElement=<HTMLButtonElement>
document.getElementById("green")

buttonGreen.addEventListener("click", background03 ⇑)

A

hinzufügen der EventListener für schon im HTML erstellte Buttons

init

(A)

let onchange : HTMLButtonElement = <HTMLButtonElement>
document.getElementById("small")

onchange.addEventListener("change", resizeCanvas_01)

Radiobuttons zur Festlegung der Größe des Canvas

let onchange01 : HTMLButtonElement = <HTMLButtonElement>
document.getElementById("medium")

onchange01.addEventListener("change", resizeCanvas_02)

let onchange02 : HTMLButtonElement = <HTMLButtonElement>
document.getElementById("large")

onchange02.addEventListener("change", resizeCanvas_03)

Radiobutton check ob man Objekte bearbeiten darf

let edit : HTMLInputElement = <HTMLInputElement>
document.getElementById("edit")

edit.addEventListener("change", select)

buttons zum erstellen von Kreisen/ Vierecken

let createCircle : HTMLButtonElement = <HTMLButtonElement>
document.getElementById("circle")

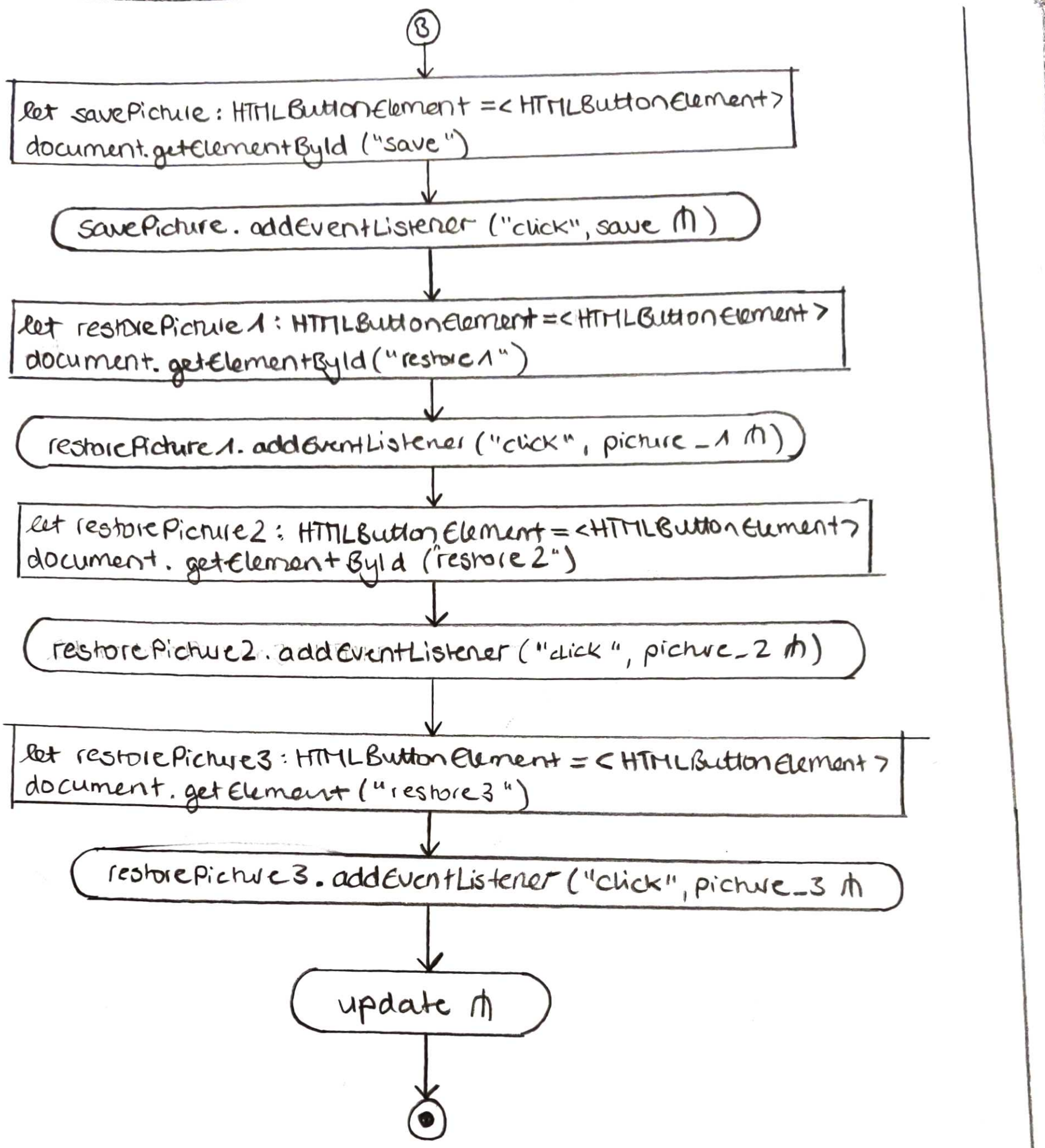
createCircle.addEventListener("click", drawCircle)

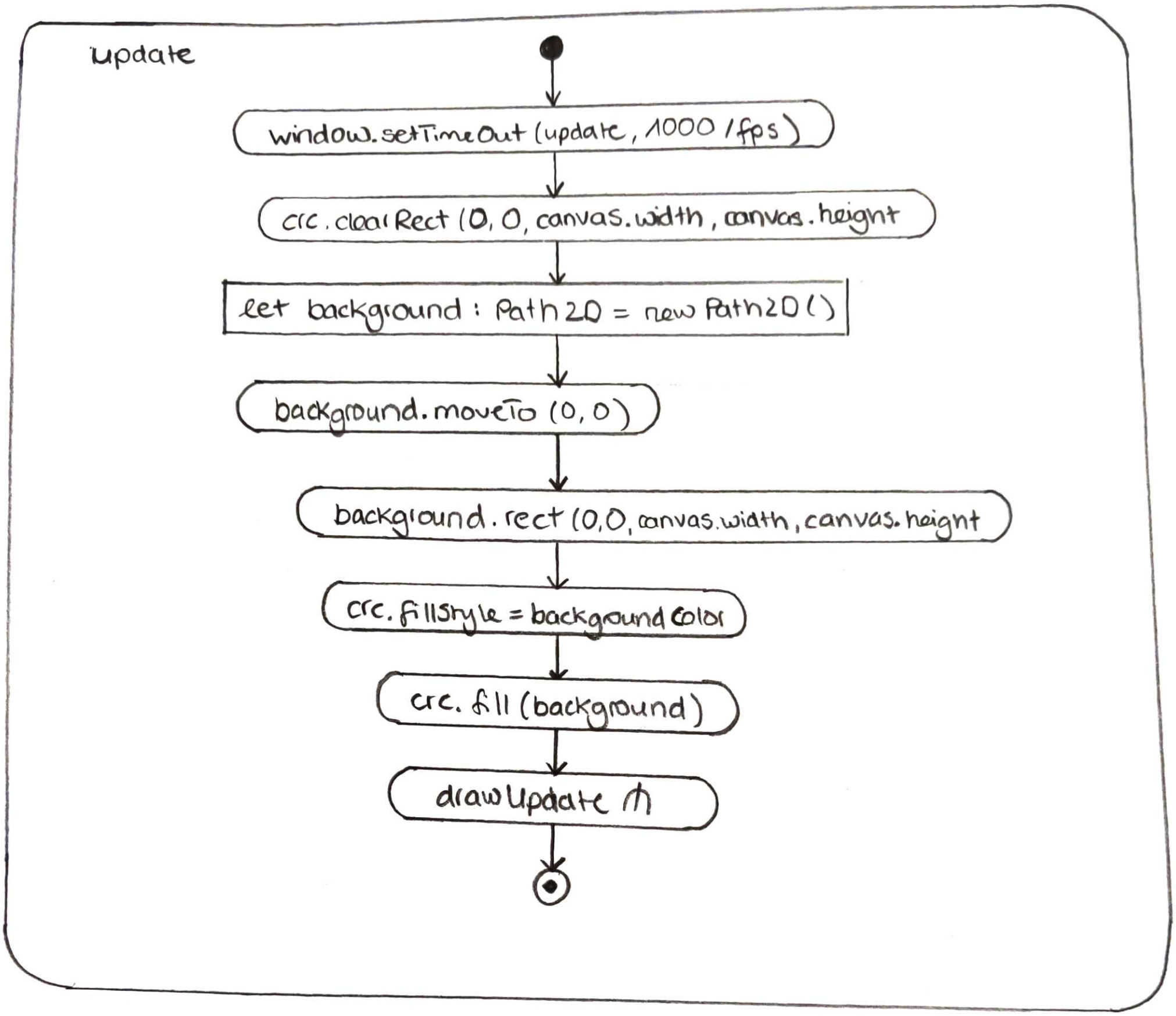
let createRect : HTMLButtonElement = <HTMLButtonElement>
document.getElementById("rect")

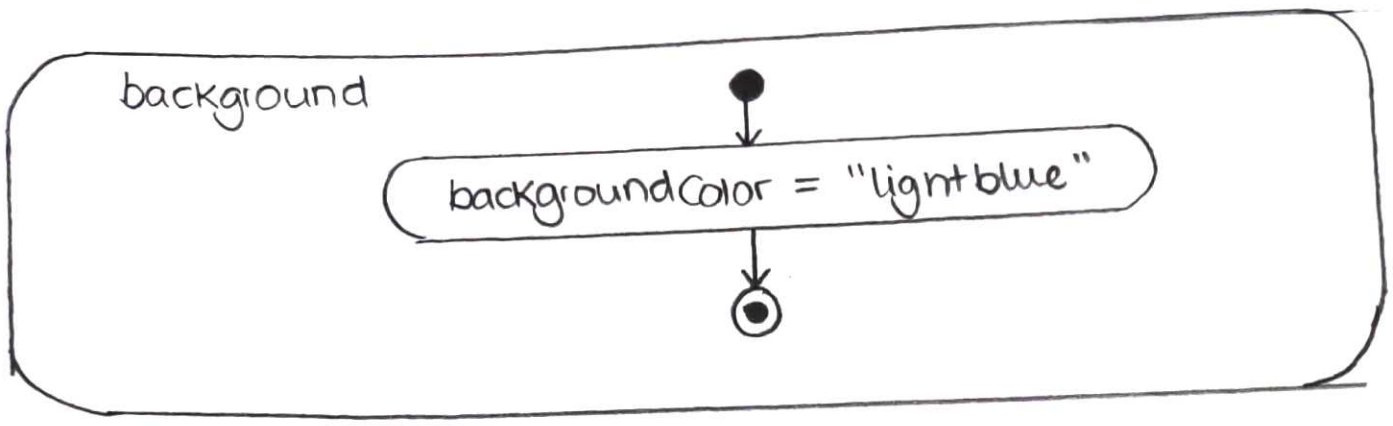
createRect.addEventListener("click", drawRect)

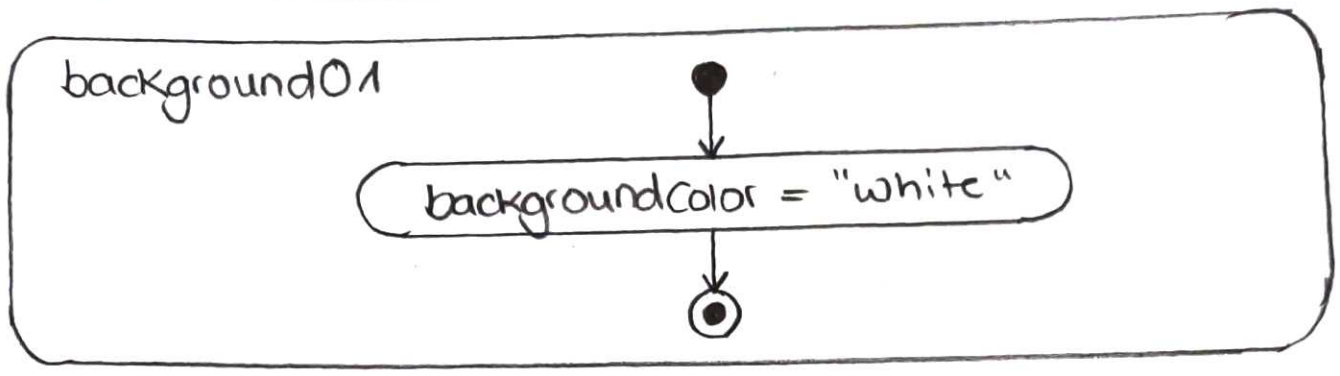
(B)

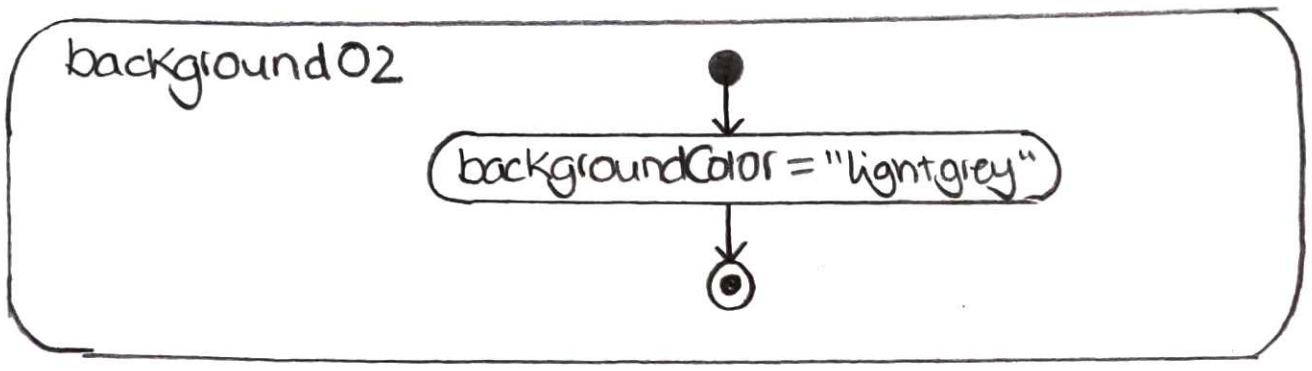
init

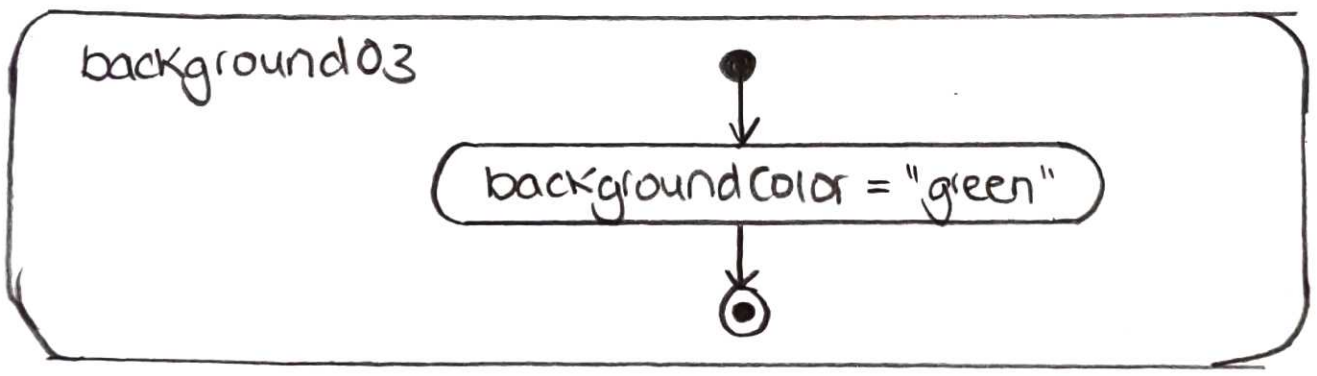

let savePicture : HTMLButtonElement = <HTMLButtonElement>
document.getElementById ("save")
savePicture . addEventListener ("click", save ⇑)
let restorePicture1 : HTMLButtonElement = <HTMLButtonElement>
document. getElementById ("restore1")
restorePicture1. addEventListener ("click", picture_1 ⇑)
let restorePicture2 : HTMLButtonElement = <HTMLButtonElement>
document . getElement+Byld ("restore2")
restorePicture2 . addEventListener ("click", picture_2 ⇑)
let restorePicture3 : HTMLButtonElement = <HTMLButtonElement>
document . getElement ("restore3")
restorePicture3 . addEventListener ("click", picture_3 ⇑
update ⇑


update
window.setTimeOut(update, 1000/fps)
crc.clearRect(0, 0, canvas.width, canvas.height
let background : Path2D = new Path2D()
background.moveTo(0, 0)
background.rect(0,0, canvas.width, canvas.height
crc.fillStyle = backgroundColor
crc.fill(background)
drawUpdate()



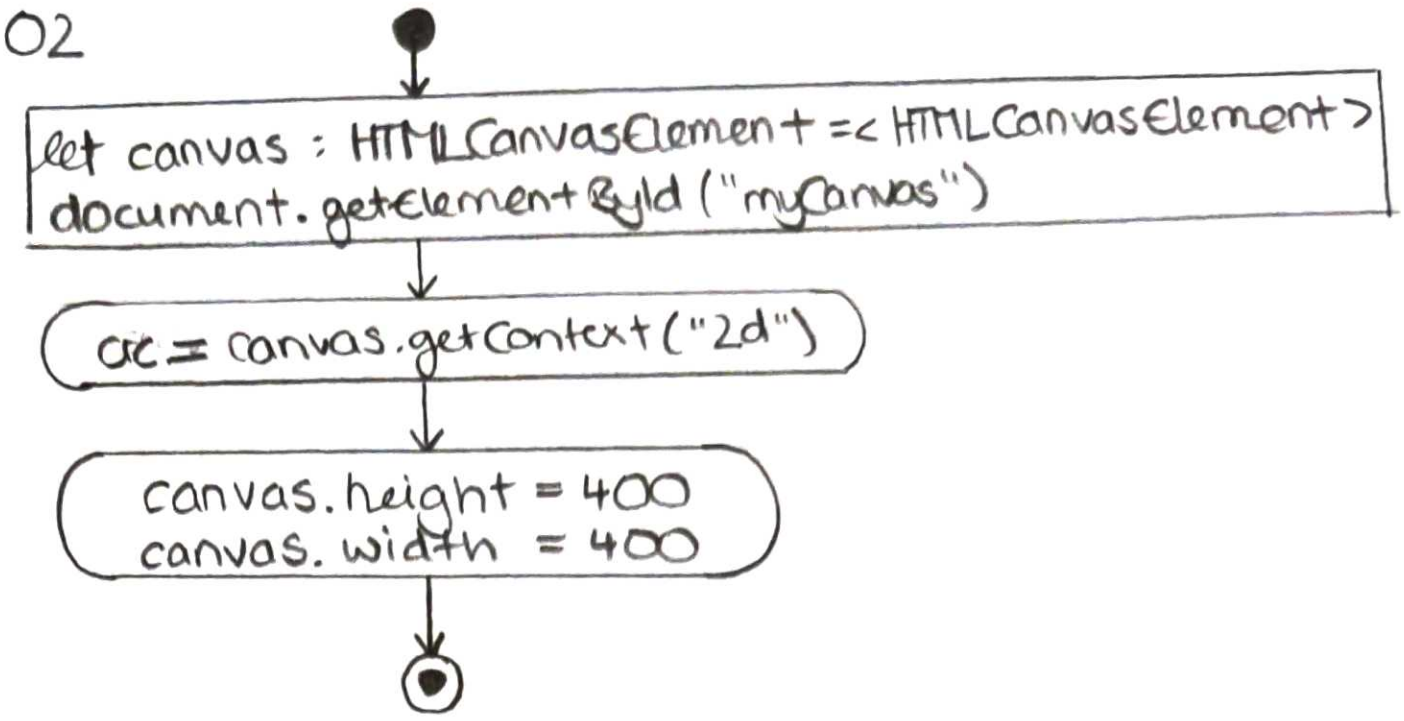



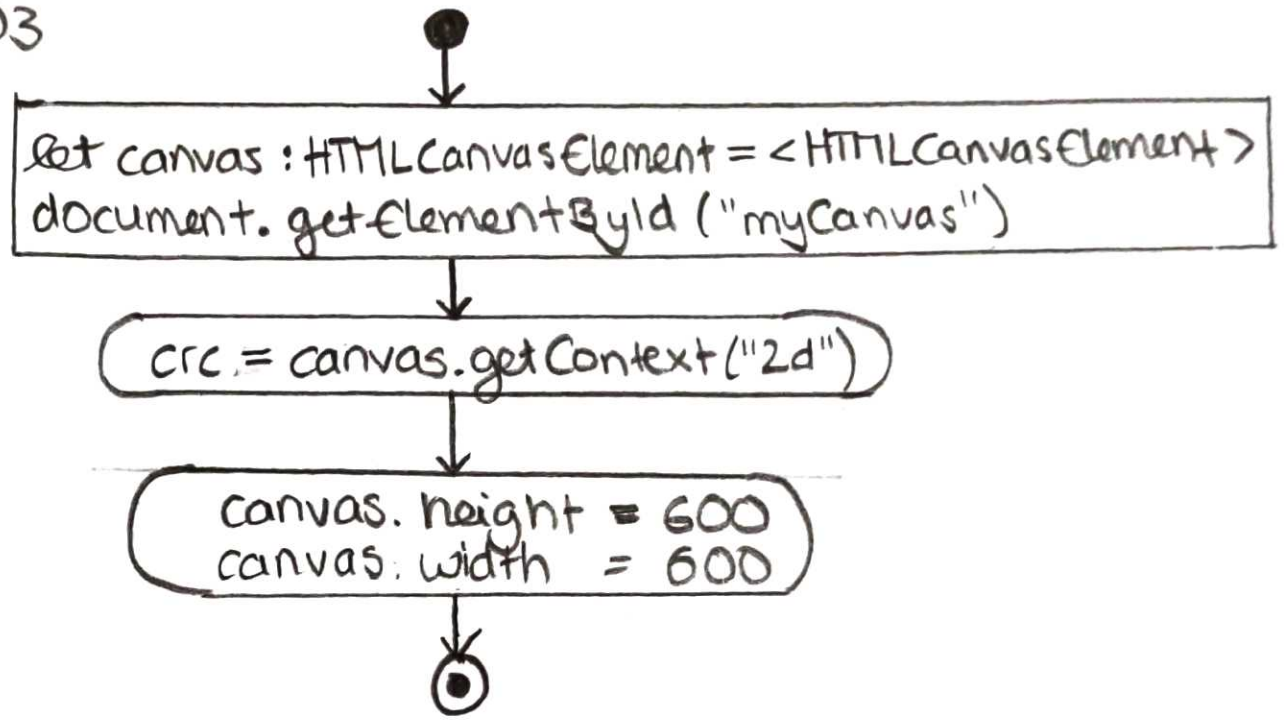




backgroundColor functions / 4 Auswahlmöglichkeiten

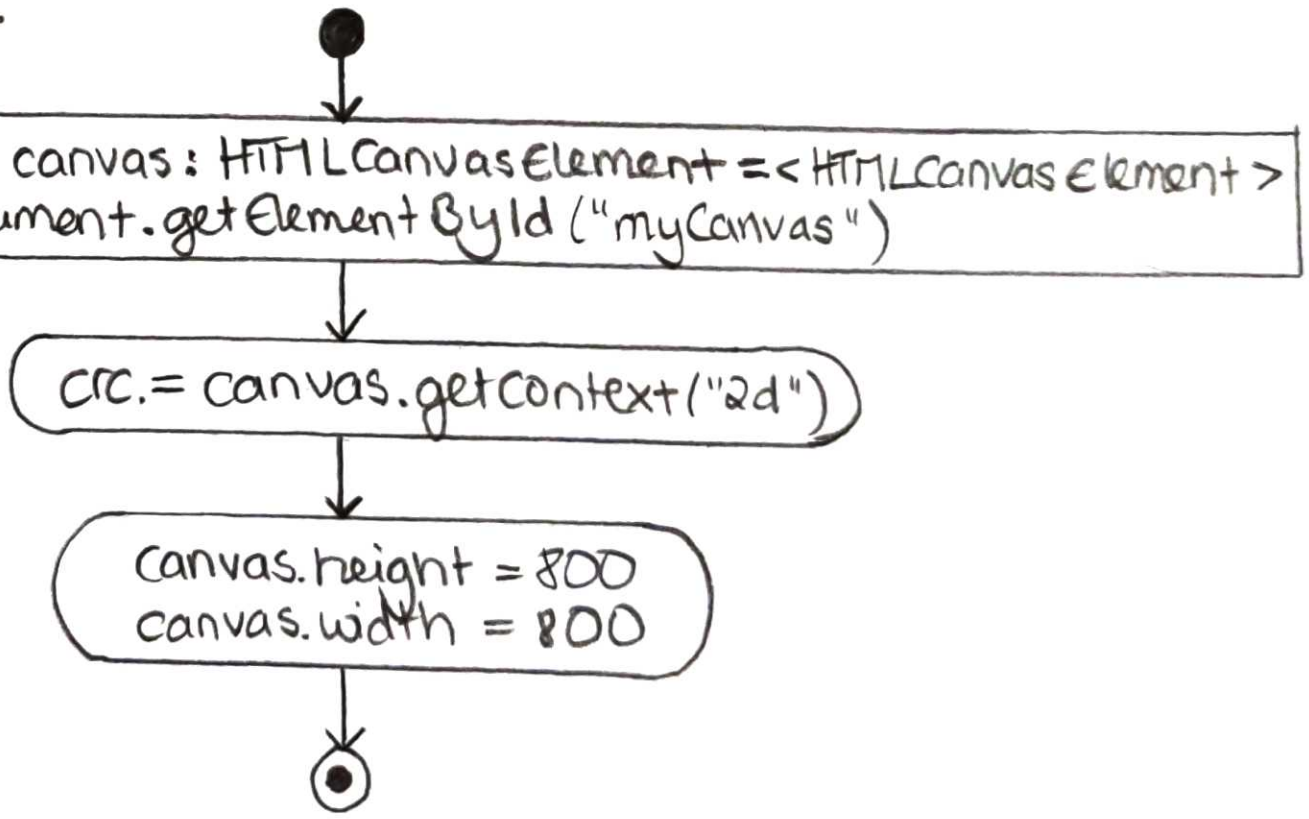
resizeCanvas_02



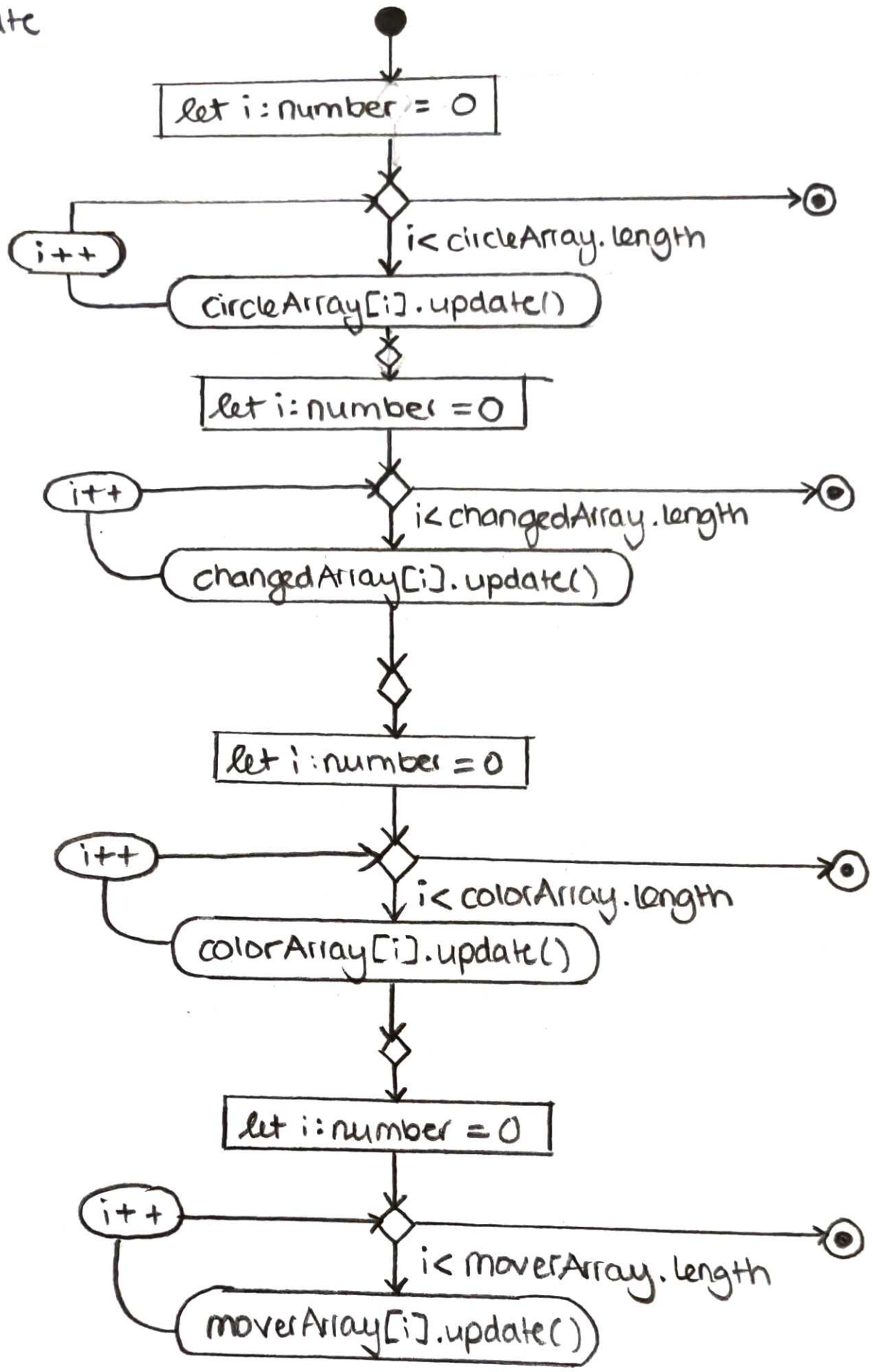

resizeCanvas_03

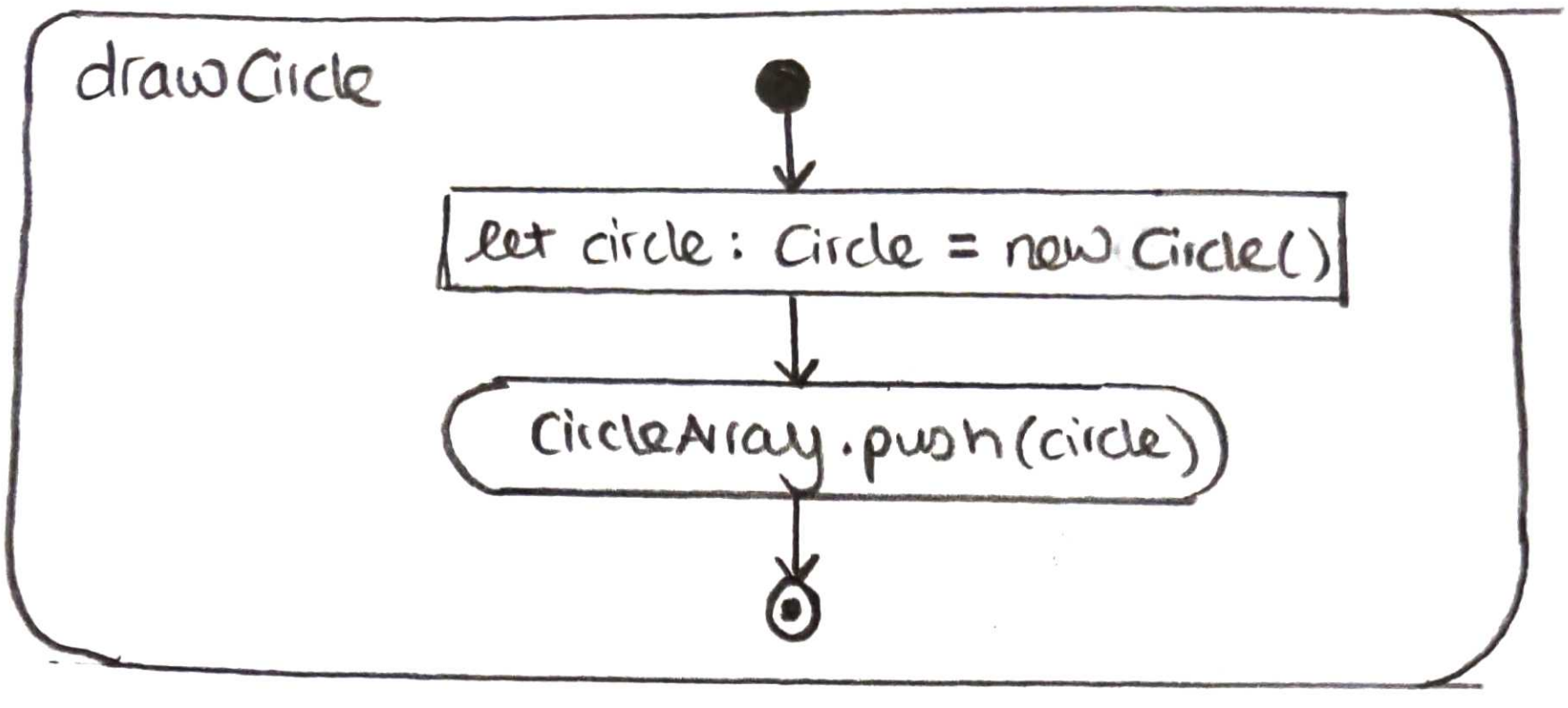


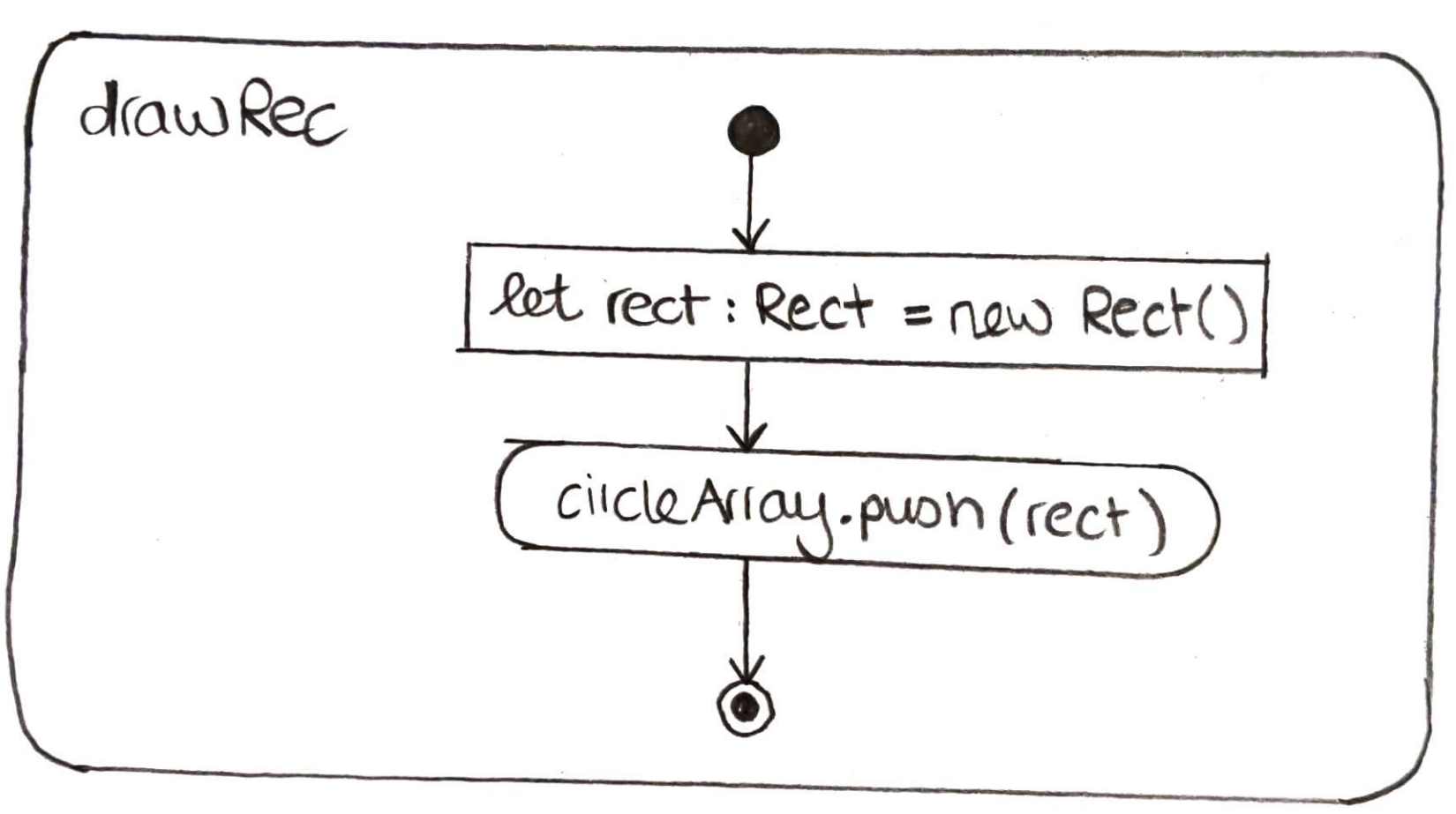

resizeCanvas_04



drawUpdate


let i:number = 0
i<circleArray.length
i++
circleArray[i].update()
let i:number =0
i<changedArray.length
i++
changedArray[i].update()
let i:number =0
i<colorArray.length
i++
colorArray[i].update()
let i:number =0
i<moverArray.length
i++
moverArray[i].update()


drawCircle
let circle : Circle = new Circle()
CircleArray.push(circle)


drawRec
let rect : Rect = new Rect()
circleArray.push(rect)

select

_e : MouseEvent

let clientX : number = _e.clientX
let clientY : number = _e.clientY

getnewPosition (clientX, clientY)

entweder hier edit aufrufen/definieren
oder global angelegt haben (per ID)
let edit : HTMLInputElement = <HTMLInputElement>
document.getElementById ("edit")

checkt ob radiobutton "edit" selektiert ist um zu bearbeiten

if

edit.checked = true

let i : number = 0

i++

for

i < circleArray.length

let currentX : number = circleArray[i].x
let currentY : number = circleArray[i].y

wenn nicht übereinstimmt

if

clientX < currentX + 20

4 solcher Argumente um Radius festzulegen

boolean = true

if

boolean == false

changedArray.push (circleArray[i])
pushedObject = i

= true

if

buttonsAreCreated = false

createAnimationsButtons ()

buttonsAreCreated = true

boolean = false

createAnimationButtons

```
let buttonDelete : HTMLButtonElement = <HTMLButtonElement>
document.createElemente ("button")
```

```
buttonDelete.innerHTML = "delete"
```

```
document.getElementById("animations").appendChild(buttonDelete)
buttonDelete.addEventListener ("click", deleteObjects fn)
```

```
let buttonMove: HTMLButtonElement = <HTMLButtonElement>
document.createElement("button")
```

```
buttonMove.innerHTML = "move"
document.getElementById("animations").appendChild(buttonMove)
buttonMove.addeventListener("click", moverObjects fn)
```

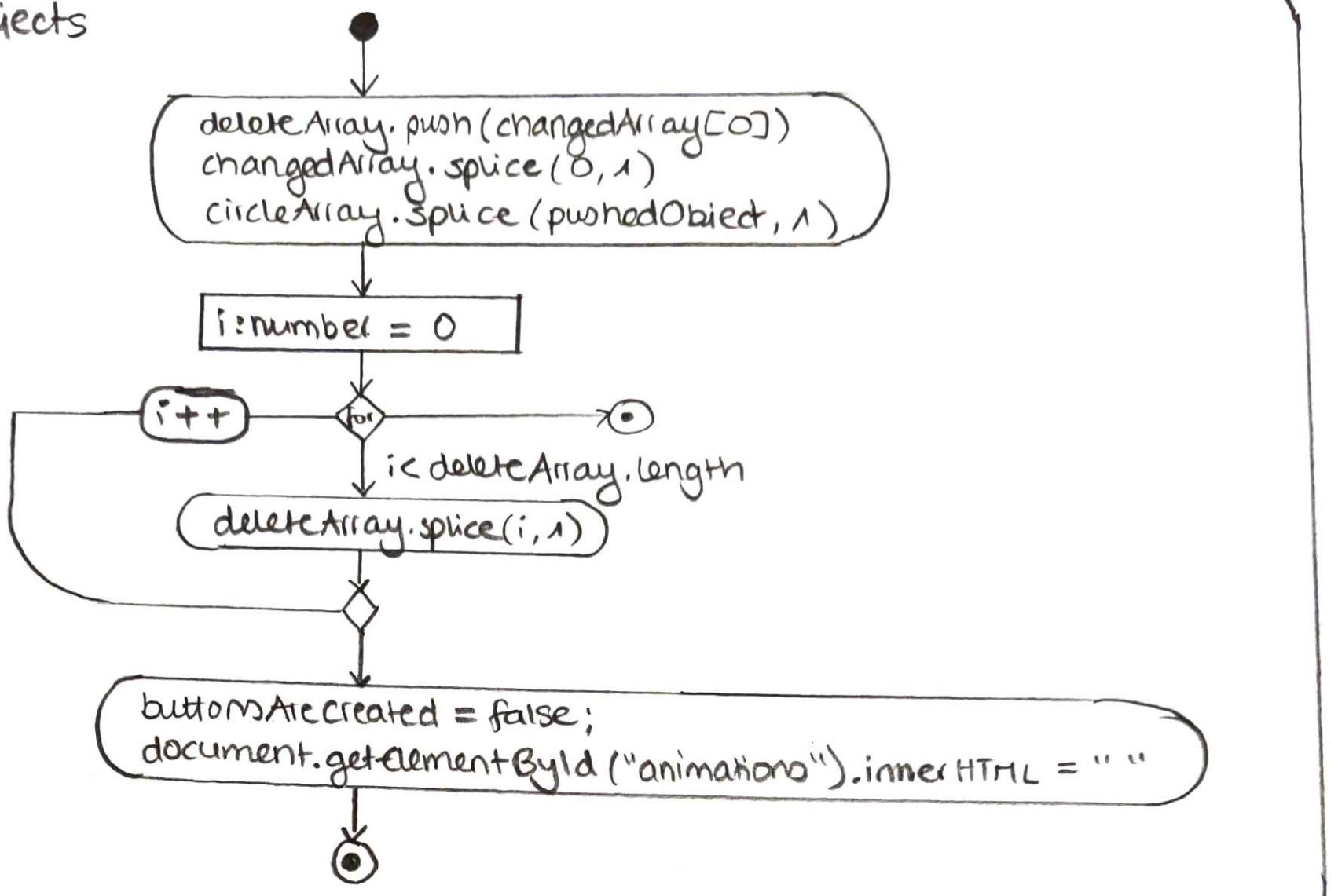
```
let buttonDrag : HTMLButtonElement = <HTMLButtonElement>
document.createElement("button")
```

```
buttonDrag.innerHTML = "drag"
document.getElementById("animations").appendchild(buttonDrag)
buttonDrag.addEventListener("click", dragObjects fn)
```

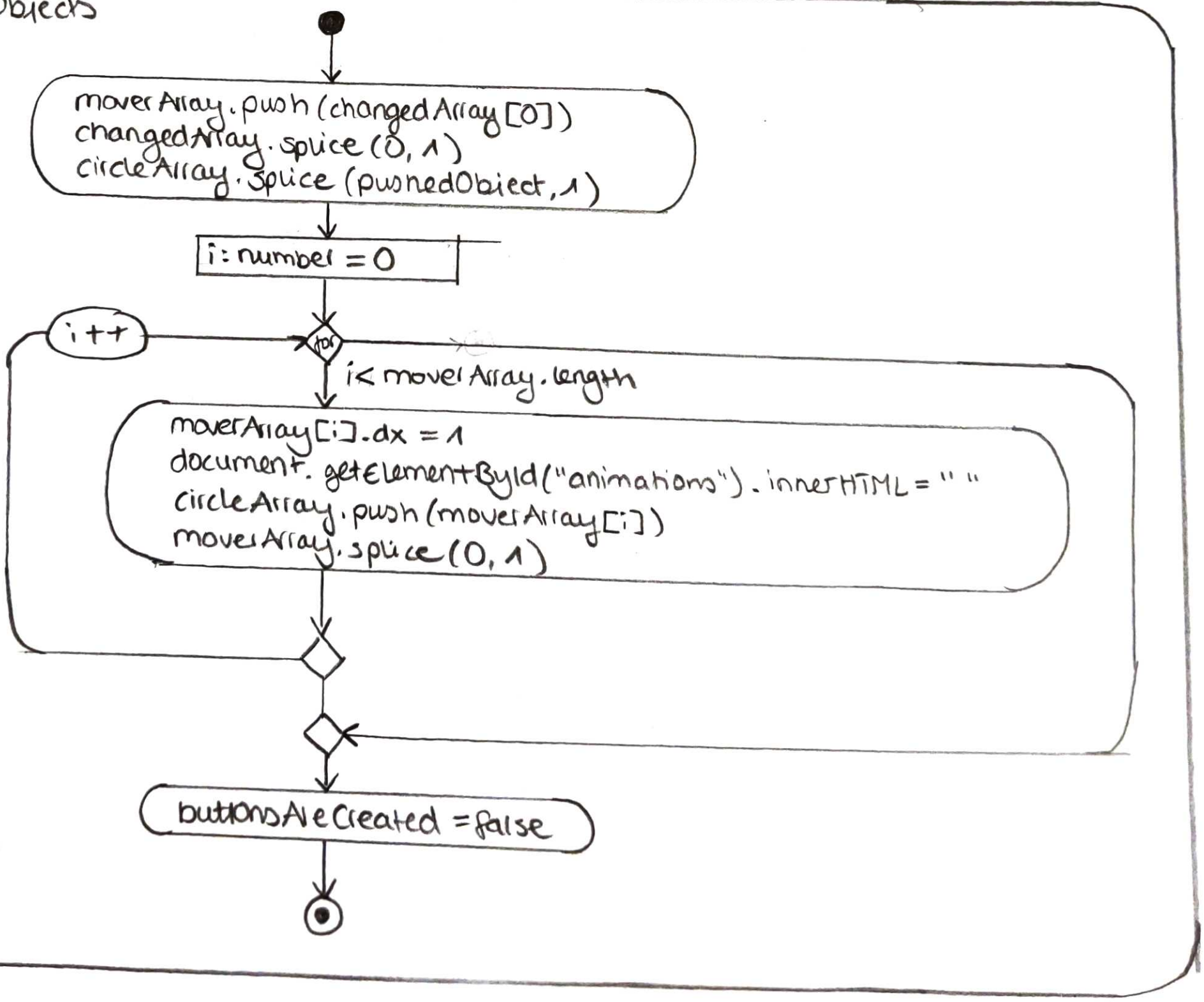
```
let buttonColor : HTMLButtonElement = <HTMLButtonElement>
document.createElement("button")
```

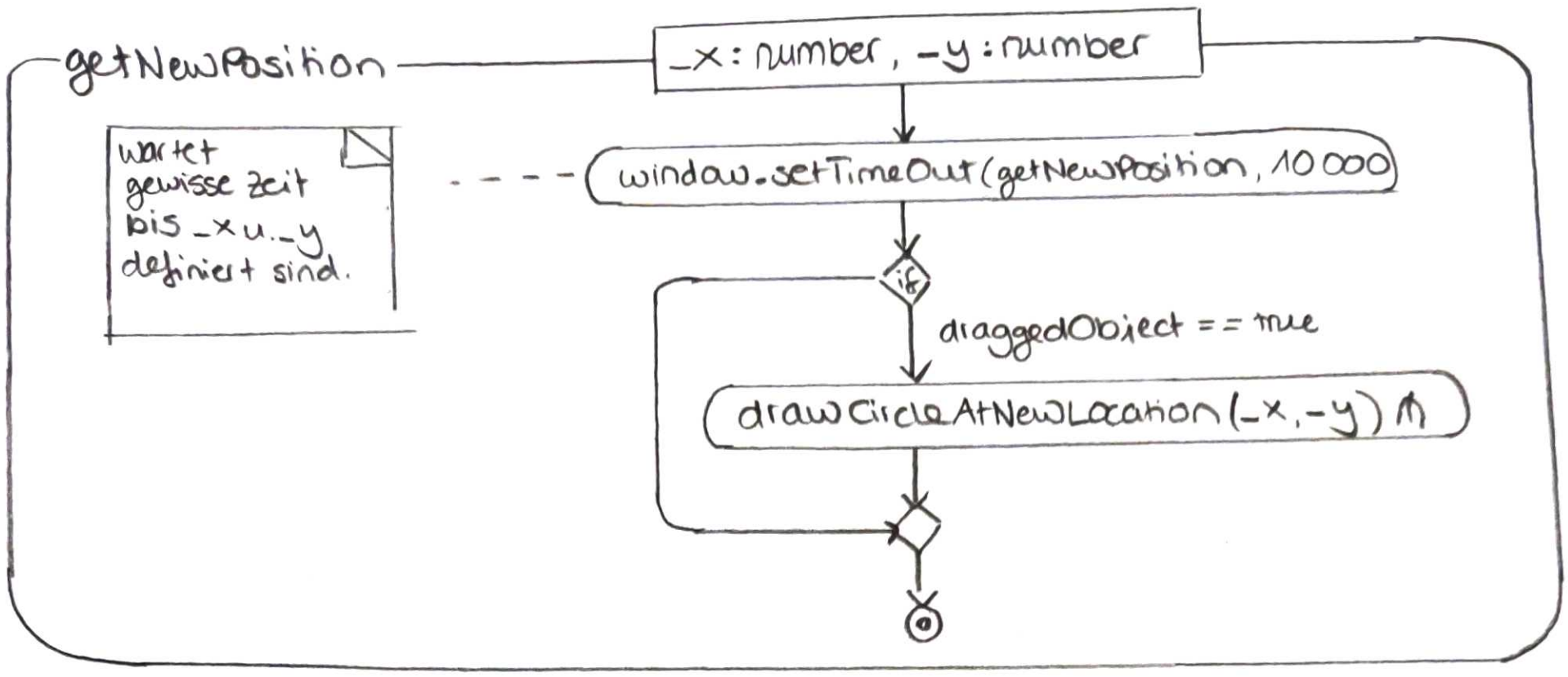
```
buttonColor.innerHTML = "rainbow"
document.getElementById("animations").appendChild(buttonColor)
buttonColor.addEventListener("click", colorObjects fn
```

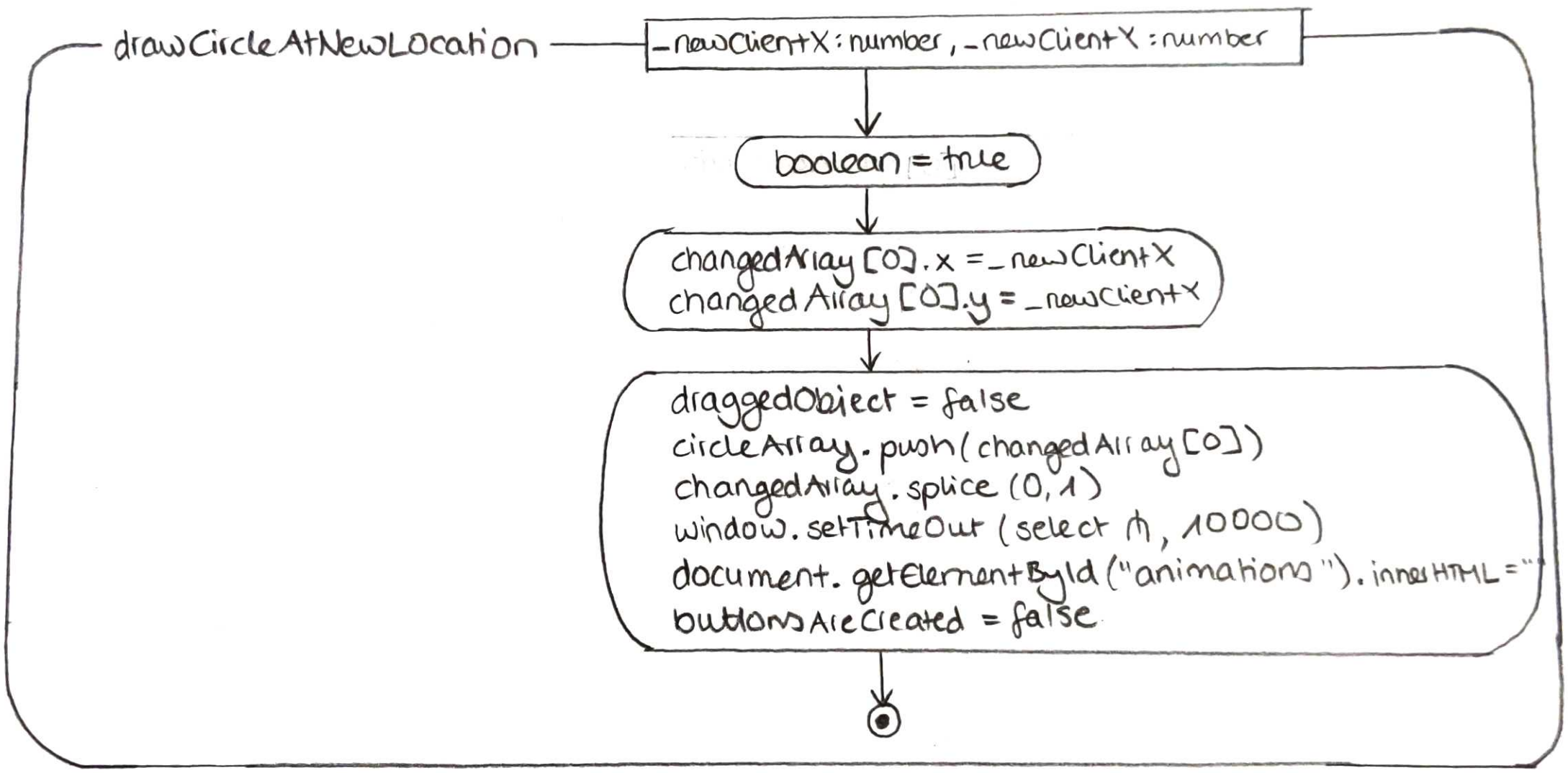
deleteObjects



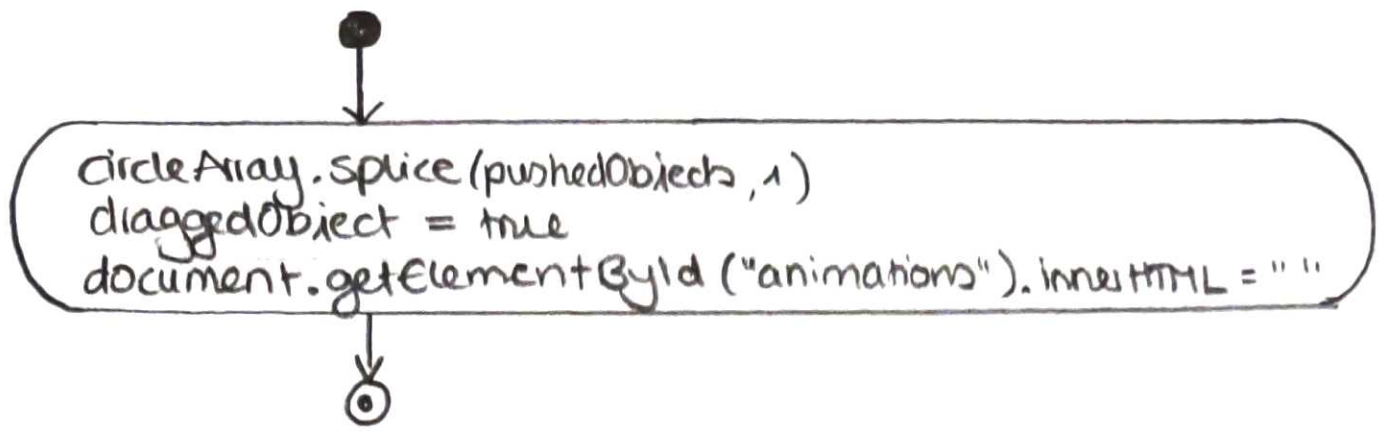

moverObjects



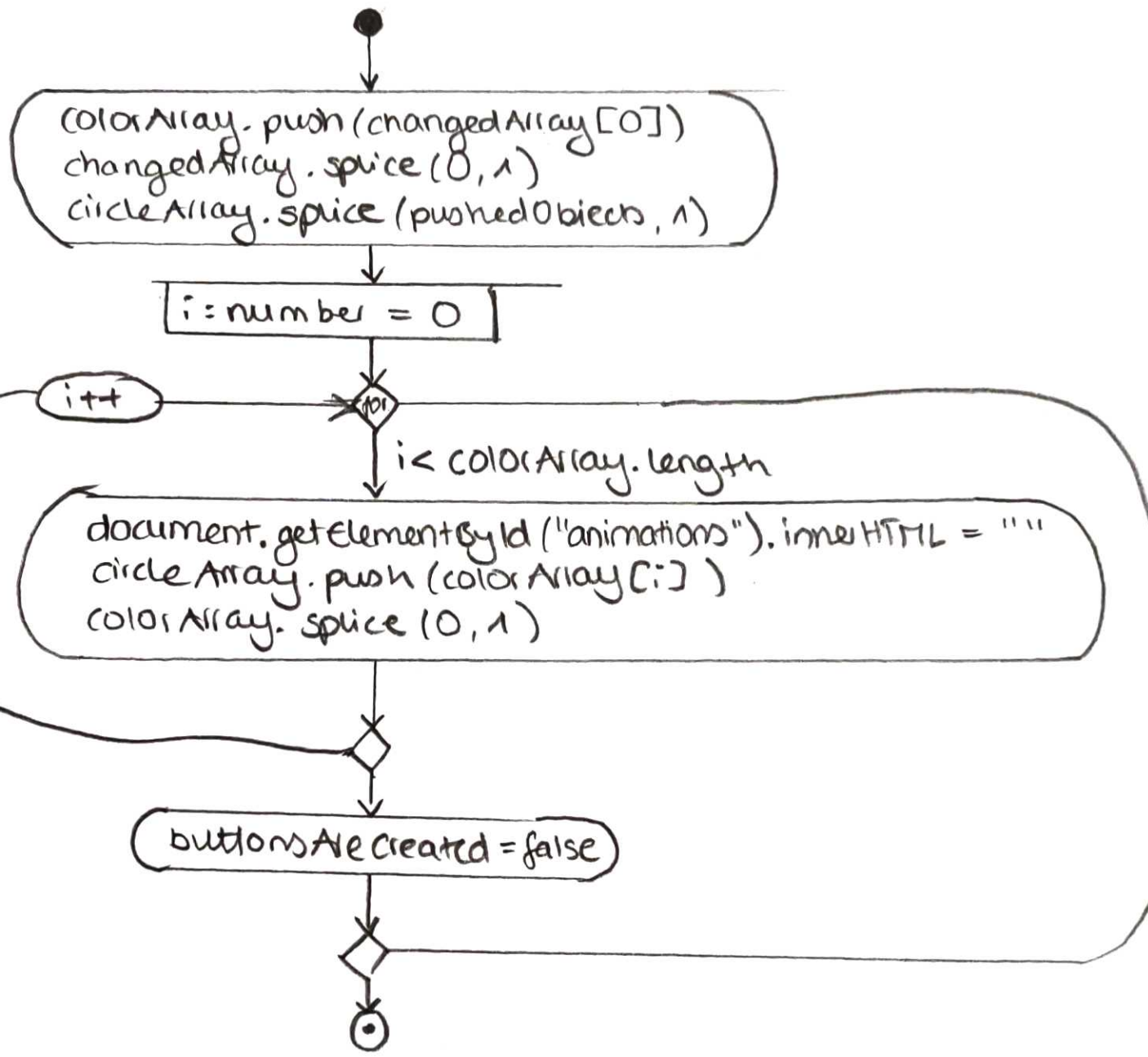

getNewPosition
_x : number , -y : number
wartet gewisse Zeit bis _x u._y definiert sind.
window.setTimeOut(getNewPosition, 10 000
if
draggedObject == true
drawCircleAtNewLocation(_x,-y)


drawCircleAtNewLocation
-newClientX : number , -newClientY : number
boolean = true
changedArray[0].x = _newClientX
changedArray[0].y = _newClientY
draggedObject = false
circleArray.push(changedArray[0])
changedArray.splice(0,1)
window.setTimeOut(select (), 10000)
document.getElementById("animations").innerHTML =""
buttonsAreCreated = false

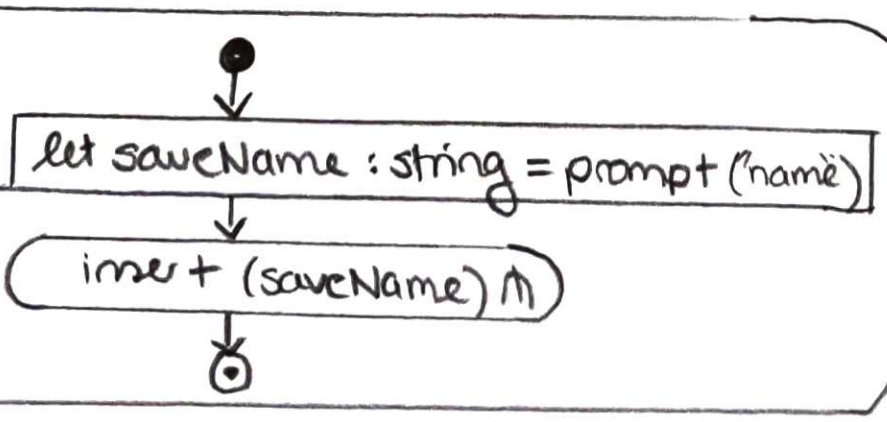
dragObjects



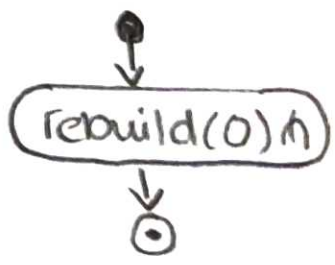

colorObjects



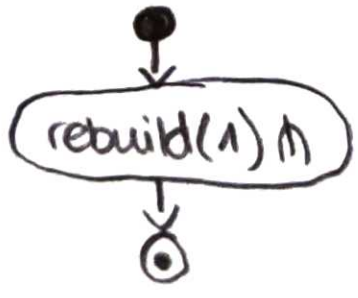

save



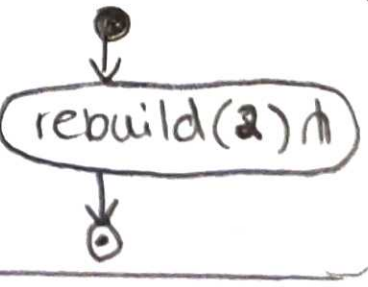

picture_1

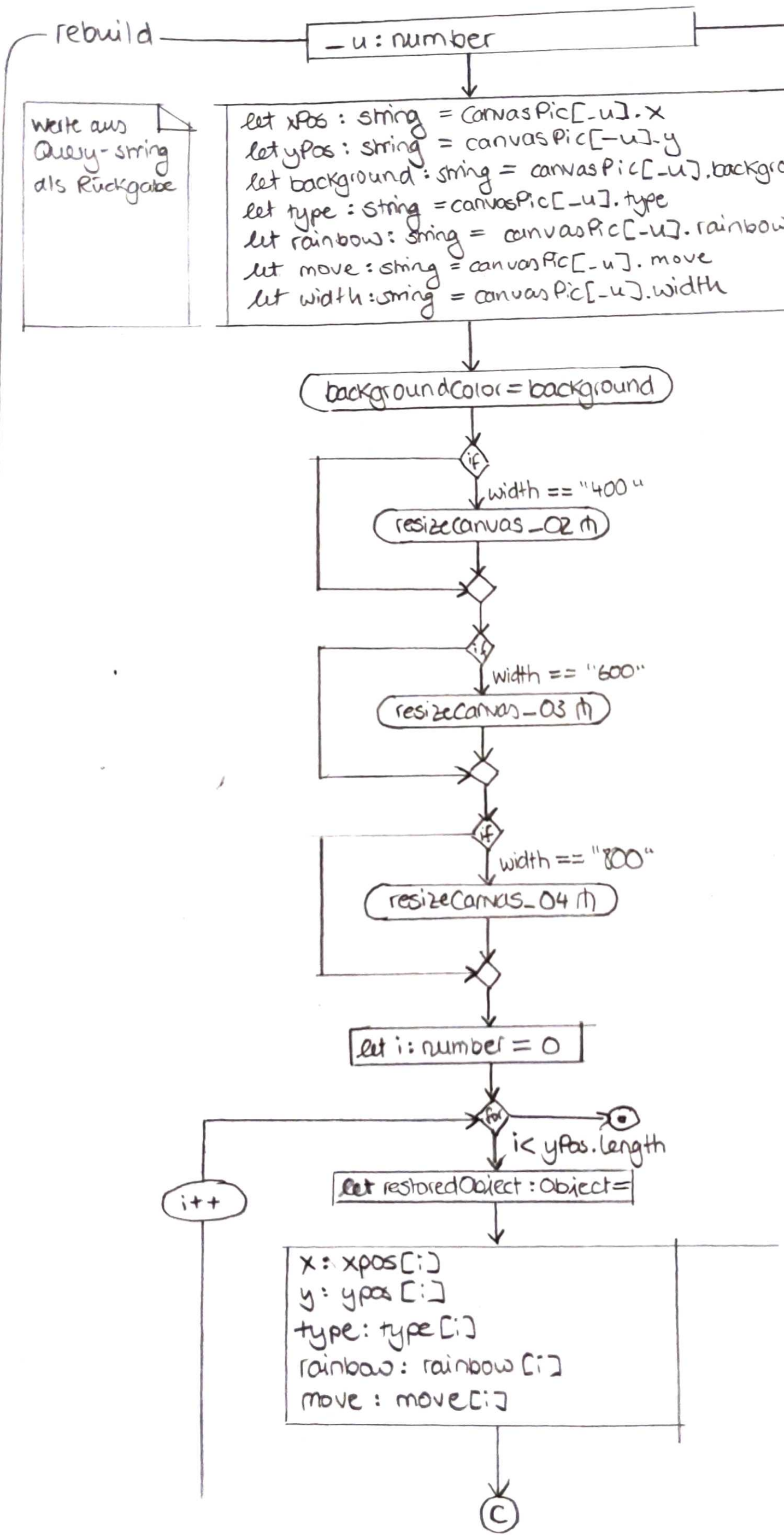


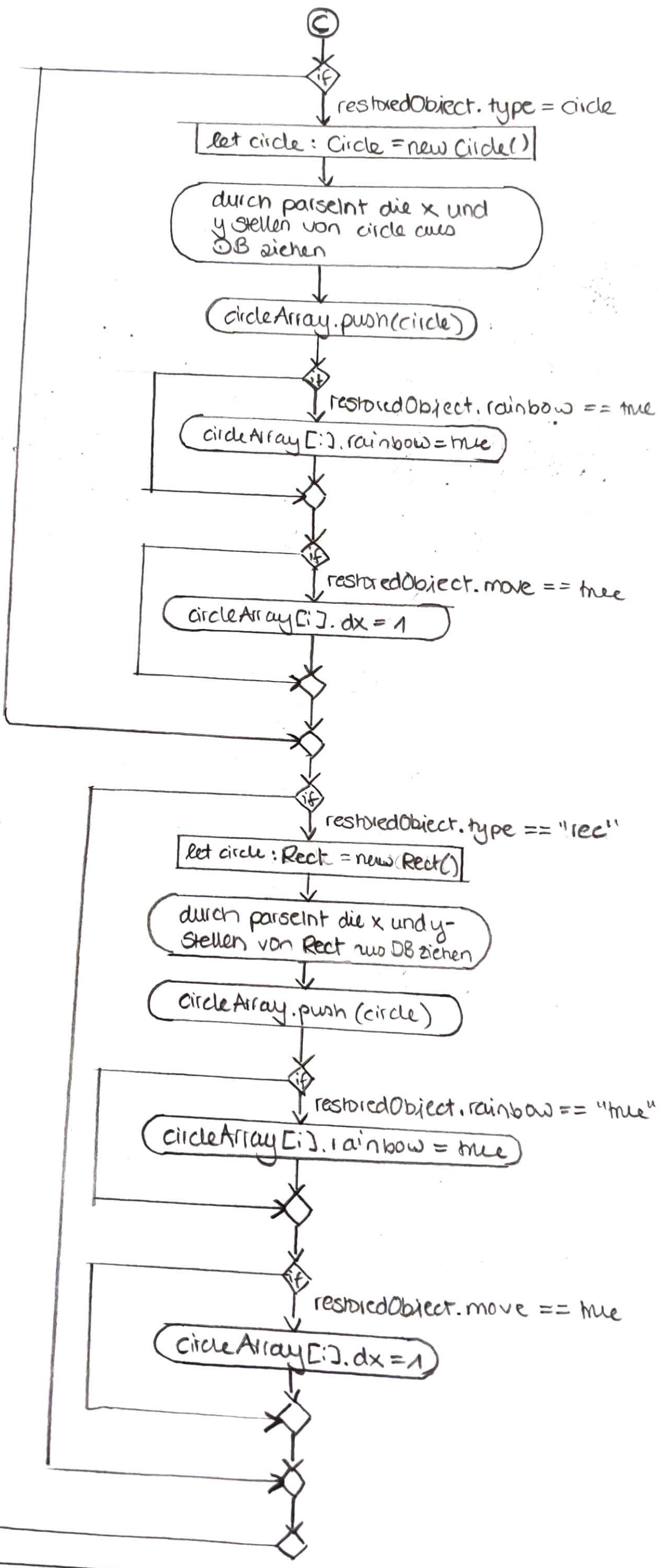

picture_2




picture_3




rebuild
_u: number
Werte aus Query-string als Rückgabe
let xPos : string = canvasPic[_u].x
let yPos : string = canvasPic[_u].y
let background : string = canvasPic[_u].backgroundcolor
let type : string = canvasPic[_u].type
let rainbow: string = canvasPic[_u].rainbow
let move : string = canvasPic[_u].move
let width : string = canvasPic[_u].width
backgroundColor = background
if
width == "400"
resizeCanvas_02 ()
if
width == "600"
resizeCanvas_03 ()
if
width == "800"
resizeCanvas_04 ()
let i: number = 0
for
i< yPos.length
i++
let restoredObject : Object =
x: xpos[i]
y: ypos[i]
type: type[i]
rainbow: rainbow[i]
move: move[i]
C


C
if
restoredObject. type = circle
let circle : Circle = new Circle()
durch parseInt die x und y stellen von circle aus DB ziehen
circleArray.push(circle)
if
restoredObject. rainbow == true
circleArray[:]. rainbow = true
if
restoredObject. move == true
circleArray[i]. dx = 1
if
restoredObject. type == "rec"
let circle : Rect = new Rect()
durch parseInt die x und y-Stellen von Rect aus DB ziehen
circleArray.push (circle)
if
restoredObject. rainbow == "true"
circleArray[:]. rainbow = true
if
restoredObject. move == true
circleArray[:]. dx = 1

# DB Client - veränderte Funktionen

```
export interface Object {
    x: string;
    y: string;
    type: string;
    rainbow: string;
    move: string;
}

export let canvasPic: CanvasElement[];
```

export