

# **Entwicklung Interaktiver Anwendungen**

**Jana Krämer**

**260716**

## **Spielanleitung Zauberbild:**

Dem Spieler stehen drei verschiedene Canvas-Größen zur Verfügung: Klein, Mittel und Groß.

Die Hintergrundfarbe des Canvas kann durch die Buttons „Gelb“, „Blau“ und „Lila“ geändert werden.

Der Spieler hat die Möglichkeit 2 verschiedene Elemente auszuwählen: einen Kreis und ein Dreieck.

Der Kreis kann sowohl von links nach rechts animiert werden, als auch auf der Stelle pulsieren.

Für das Dreieck stehen ebenfalls zwei Animationsmuster zur Auswahl: eine Bewegung von links nach rechts, sowie von oben nach unten.

Jederzeit kann ein beliebiges Element ausgewählt werden. Daraufhin erscheinen zwei weitere Buttons, die entweder die Option bieten, das Objekt zu löschen, oder es in seine Position zu verändern. Für die zweite Interaktion kann der Spieler das Element mit den Pfeiltasten verschieben. Anschließend verschwinden die Buttons wieder.

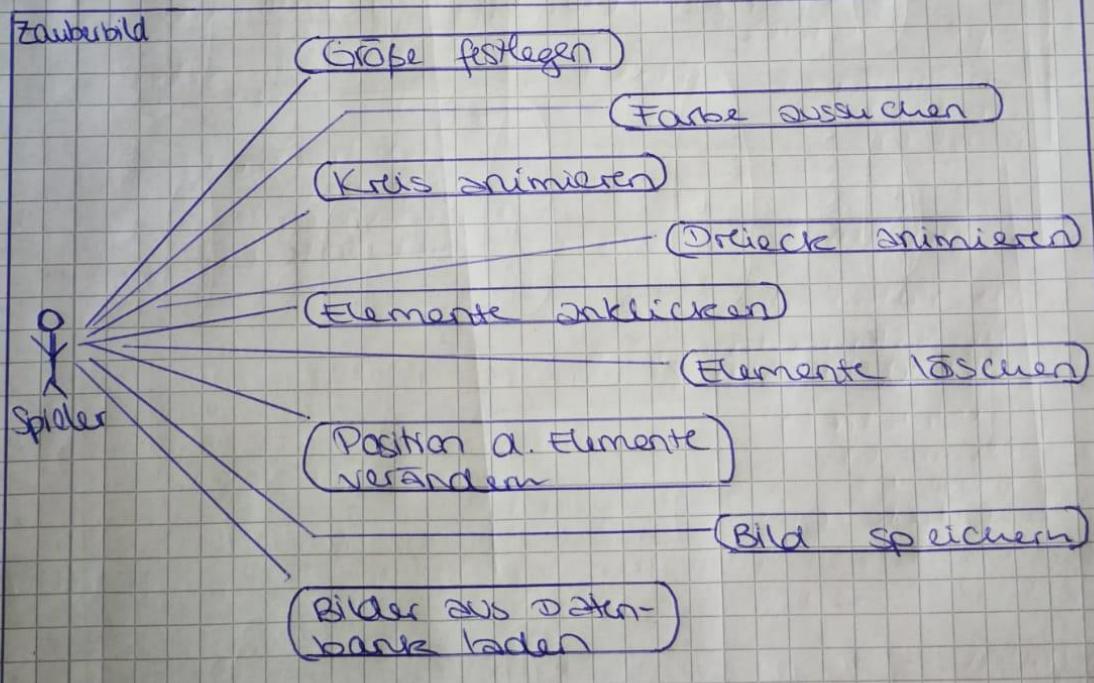
Am Ende kann der Spieler sein Zauberbild abspeichern. Er erhält eine kurze Rückmeldung, dass sein Bild gespeichert wurde.

Beim Klick auf den Button „Lade Bilder“ können bereits in der Datenbank gespeicherte Zauberbilder wieder hergestellt und weiterbearbeitet werden.

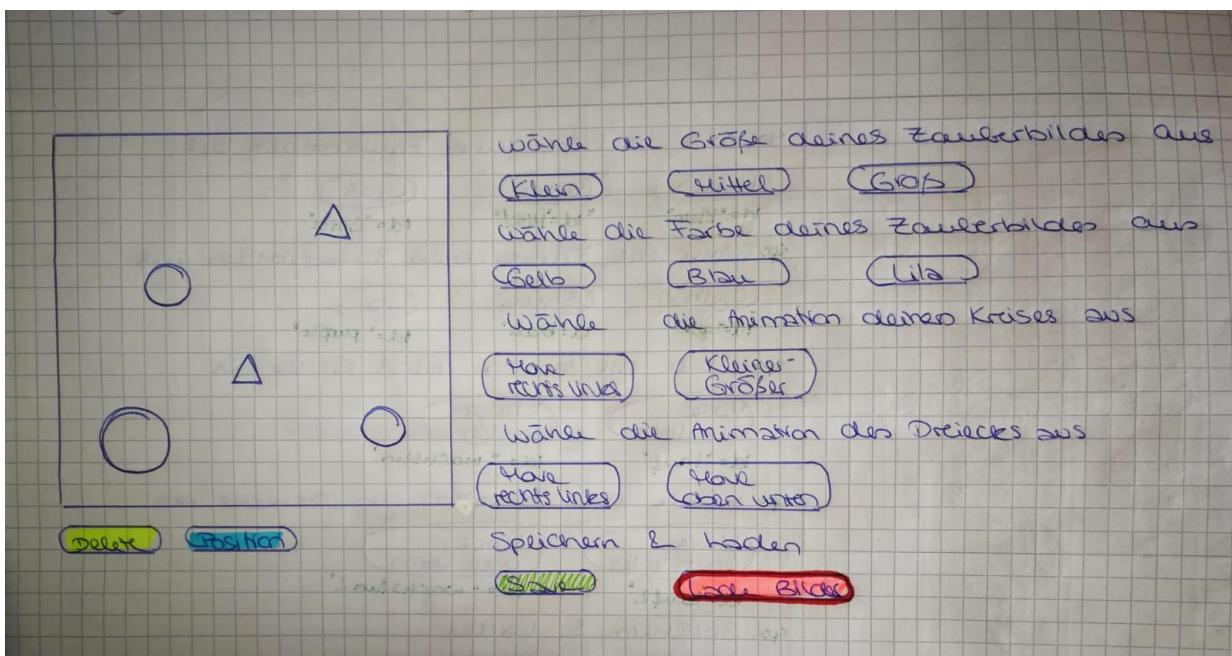
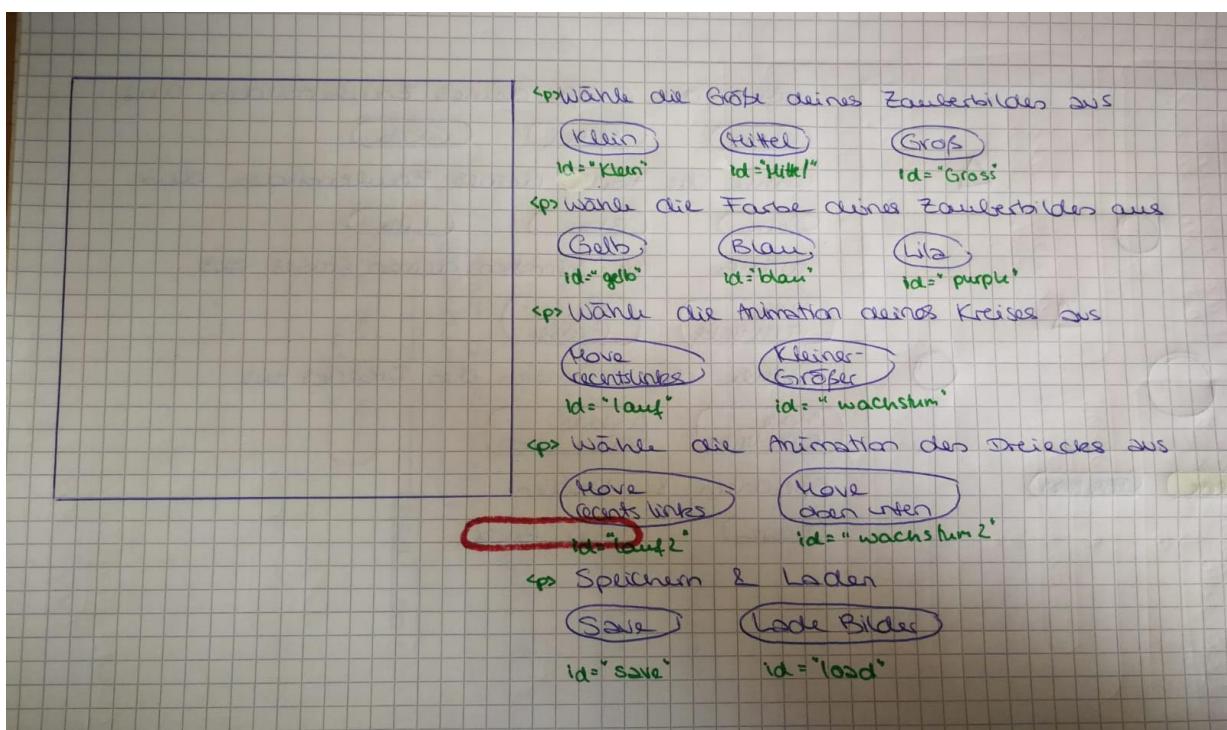
### Ablaufdiagramm

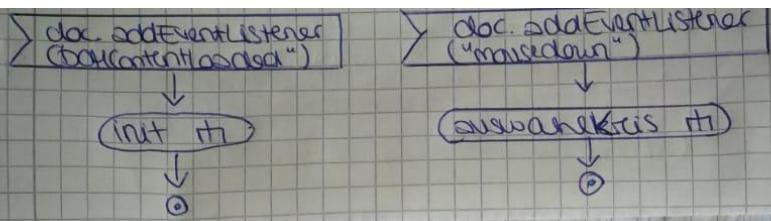
- ① Größe festlegen
- ② Farbe aussuchen
- ③ Animation für Kreis auswählen
- ④ Animation für Dreieck auswählen
- ⑤ Elemente anklicken ;
- ⑥ Löschen / Position verändern
- ⑦ Bild speichern
- ⑧ gespeicherte Bilder aus Datenbank wieder herstellen und weiter bearbeiten

### Anwendungsfalldiagramm



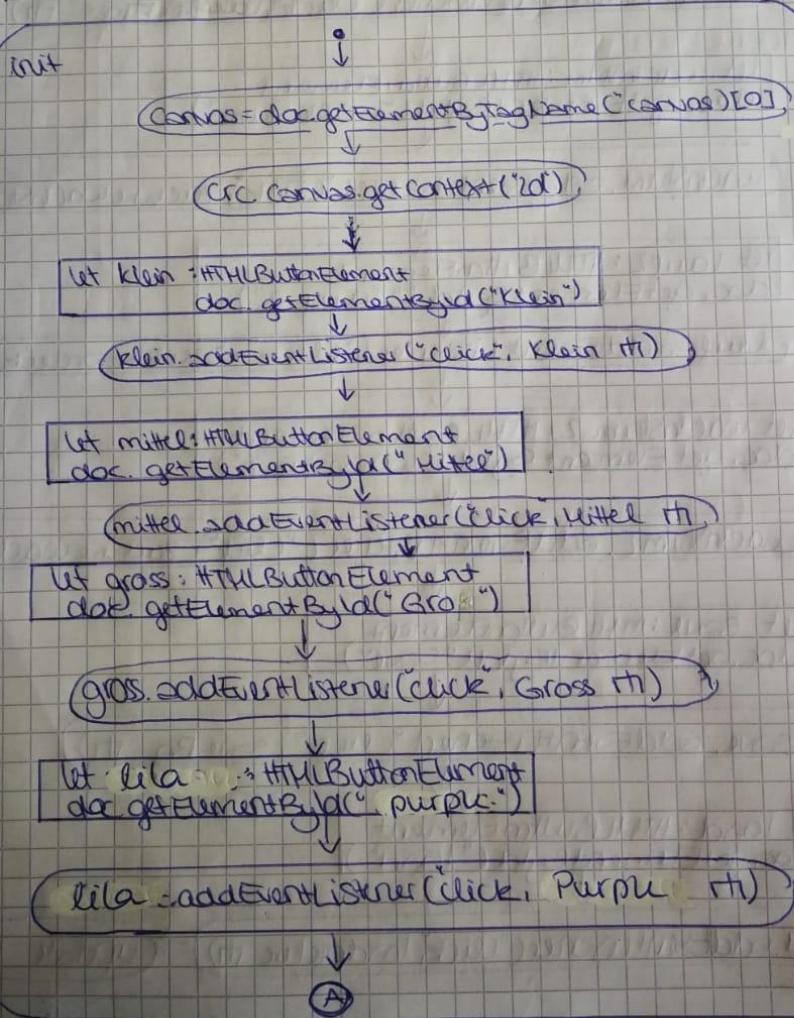
## Skizzen:



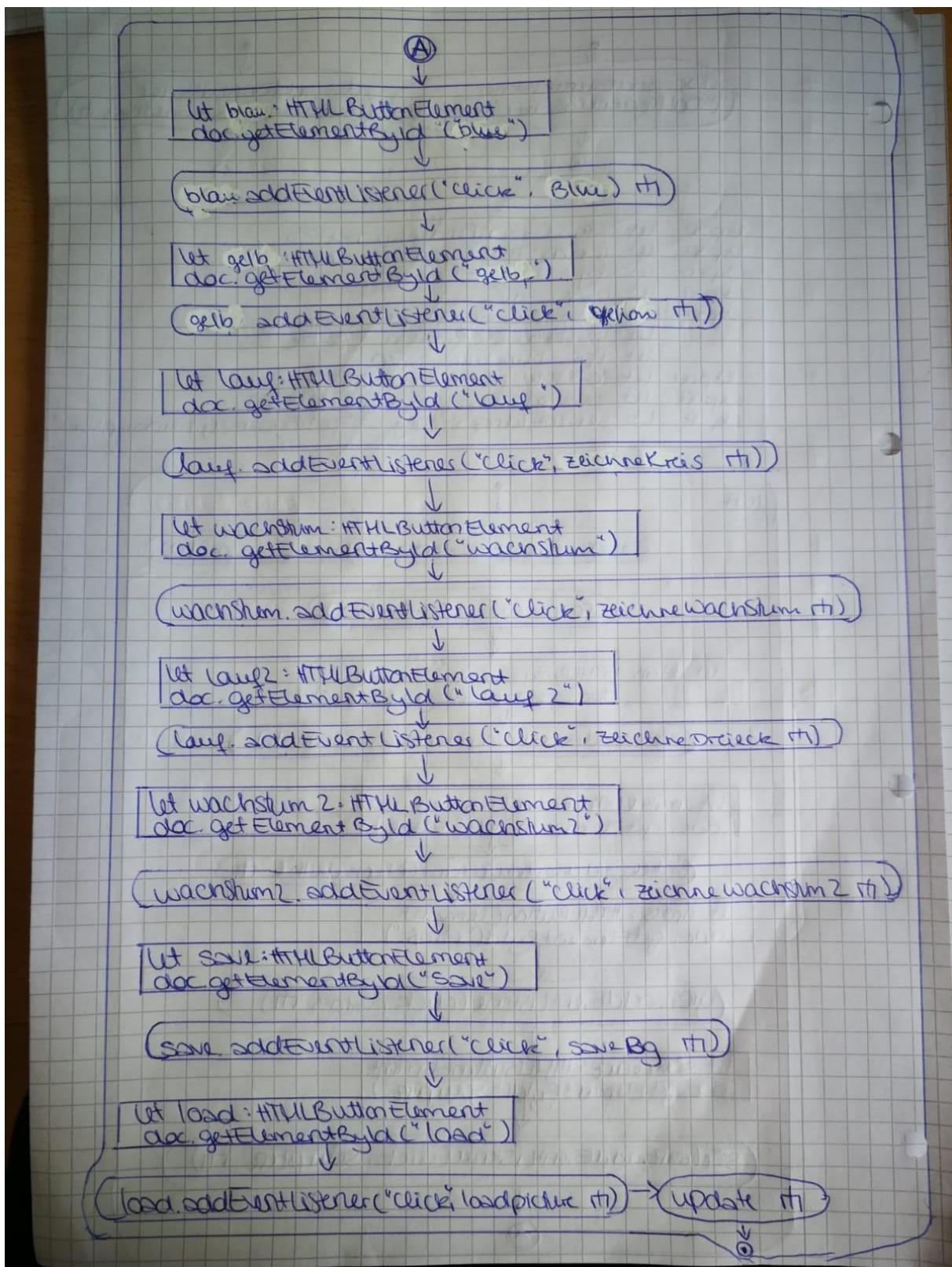


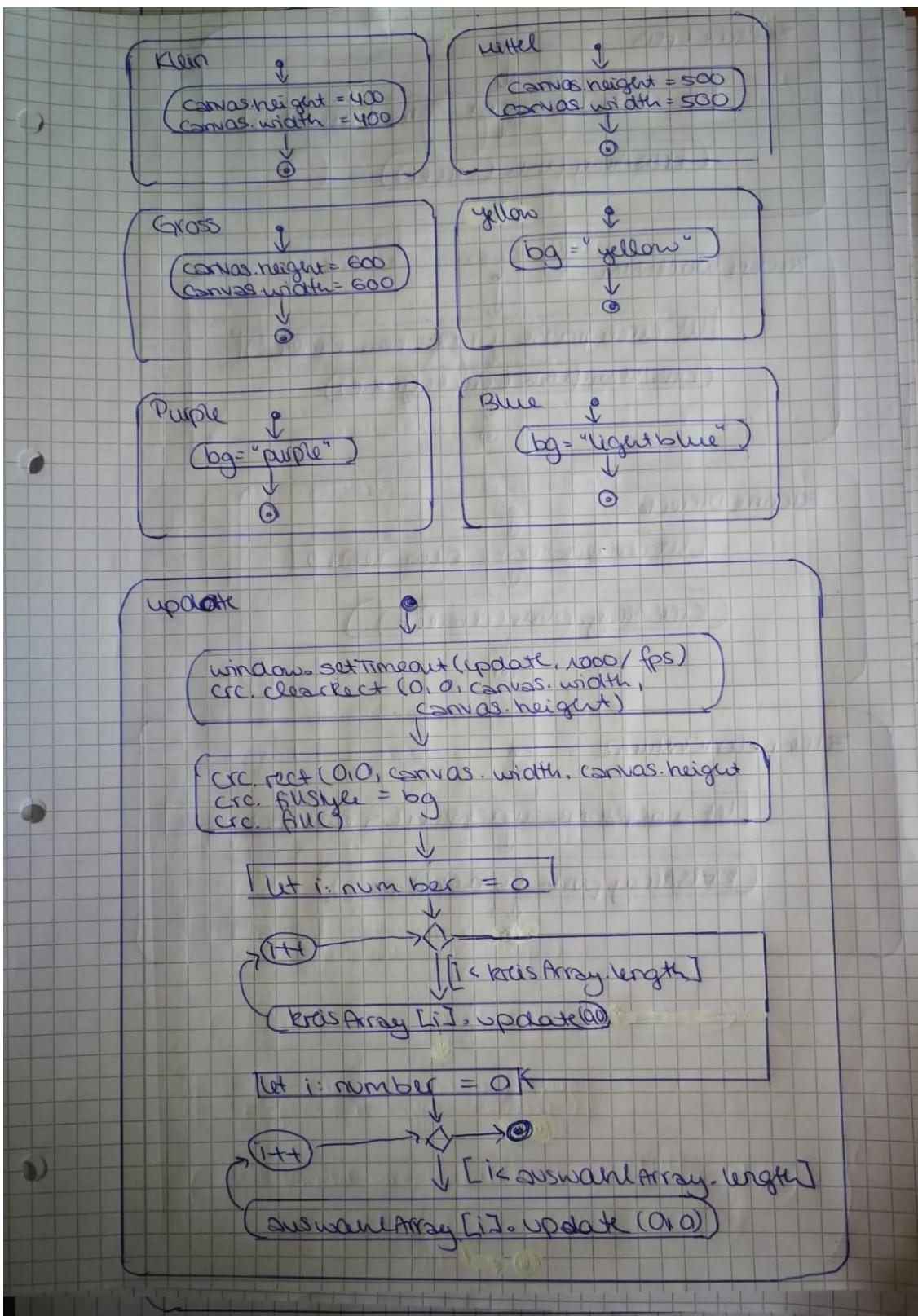
```

export let laedebildes : Boolean = true
export let crc : CanvasRenderingContext2D
export let canvas : HTMLCanvasElement
export let bg : string = "white"
export let color : string
export let ausgewaehltesElement : number
export let auswahle : Boolean = false
let fps : number = 30
export let kreisArray : Element[] = []
export let auswahlArray : Element[] = []
let serverAddress : string = "https://kraemerj.hackerrank.com"
let buttons2 : Boolean = false
  
```



## Buttons werden erstellt + Eventlistener angefügt:





zeichne Kreis

```
let circle: Element = new Element()
```

```
(kreisArray.push(circle))
```

zeichne Wachstum

```
let circlegroesse: Groesse = new Groesse()
```

```
(kreisArray.push(circlegroesse))
```

zeichne Dreieck

```
let dreieckTest: Test = new Test()
```

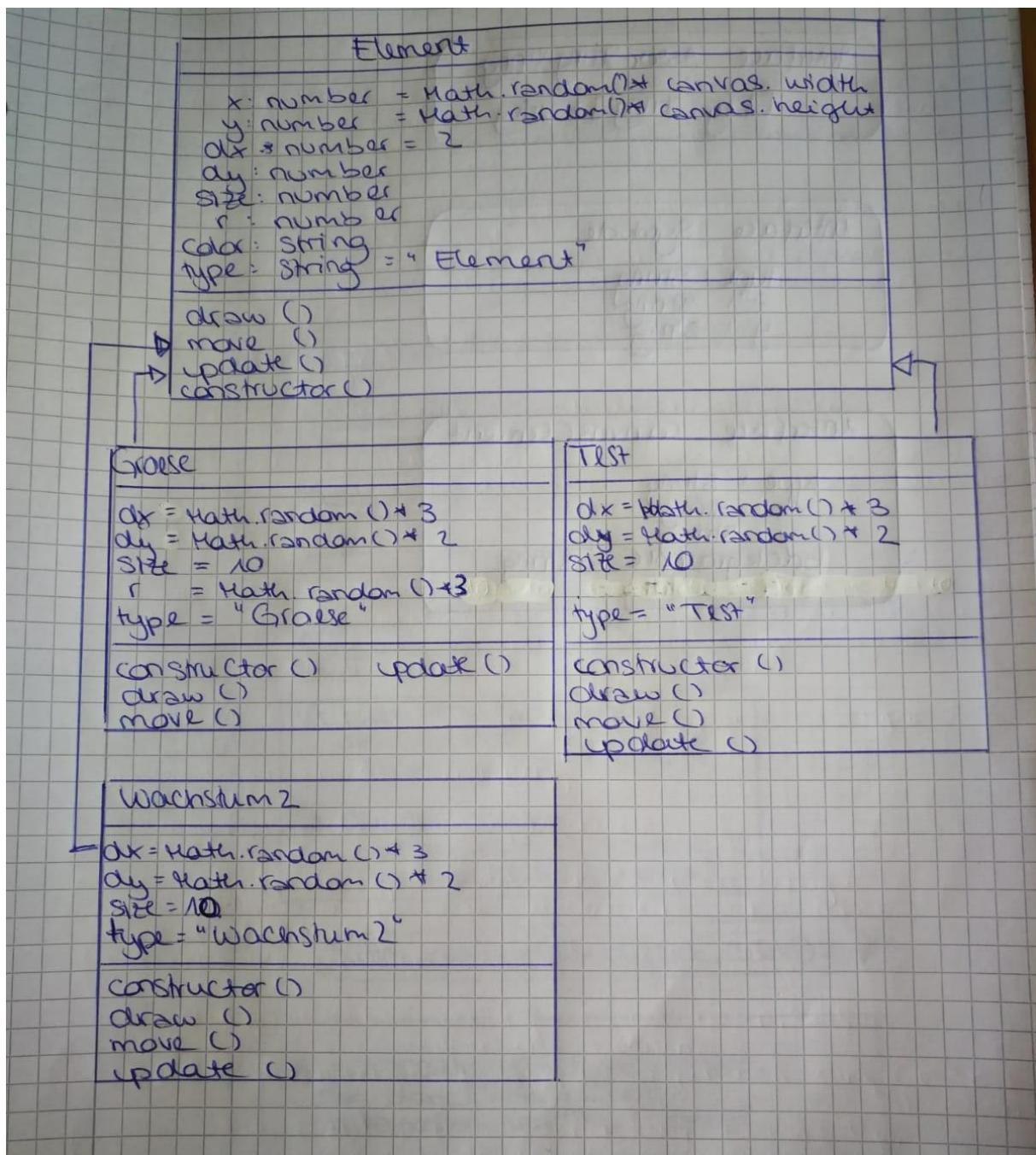
```
(kreisArray.push(dreieck))
```

zeichneWachstum2

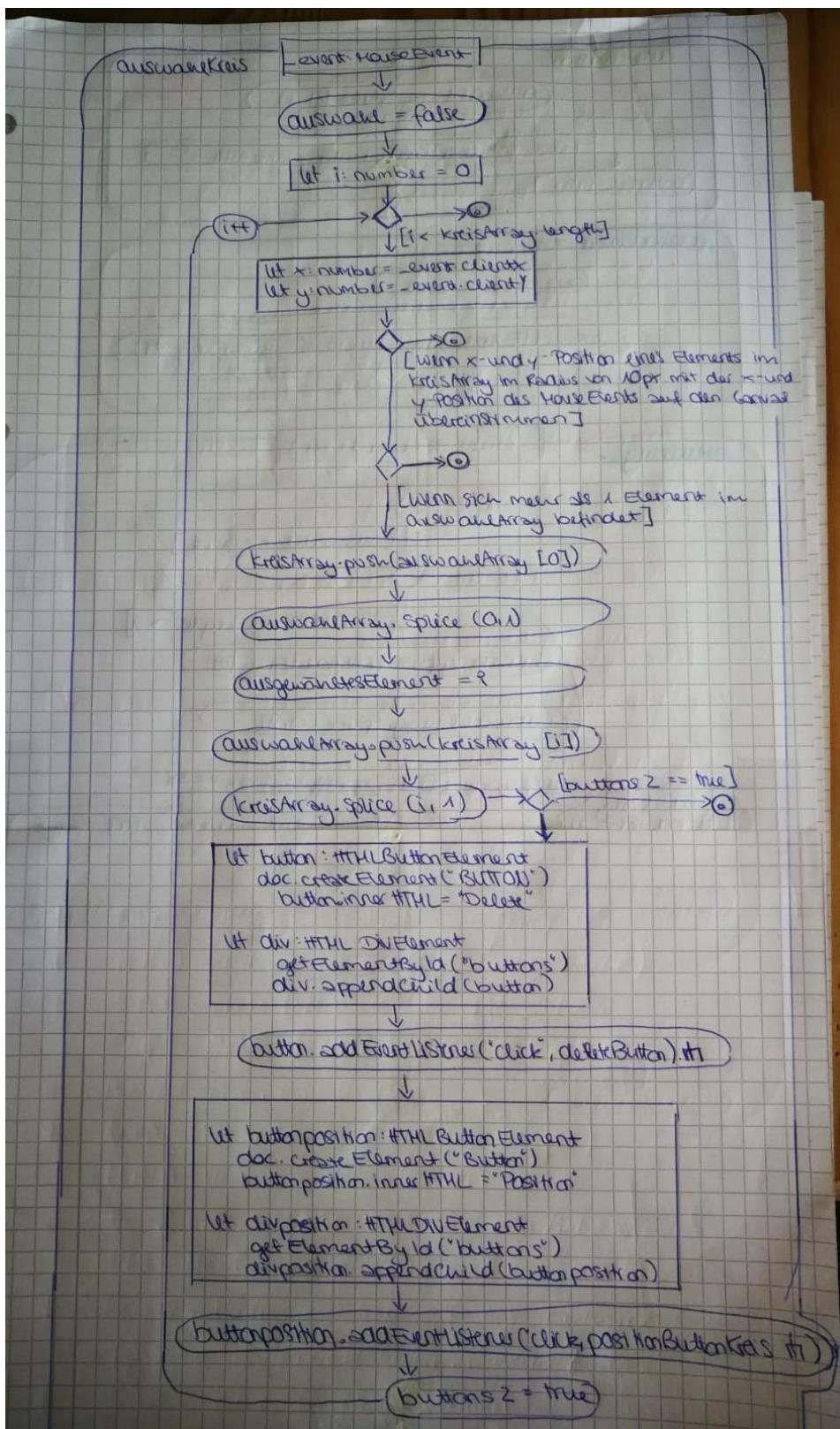
```
let wachstum2: Wachstum2 = new Wachstum2()
```

```
(kreisArray.push(wachstum2))
```

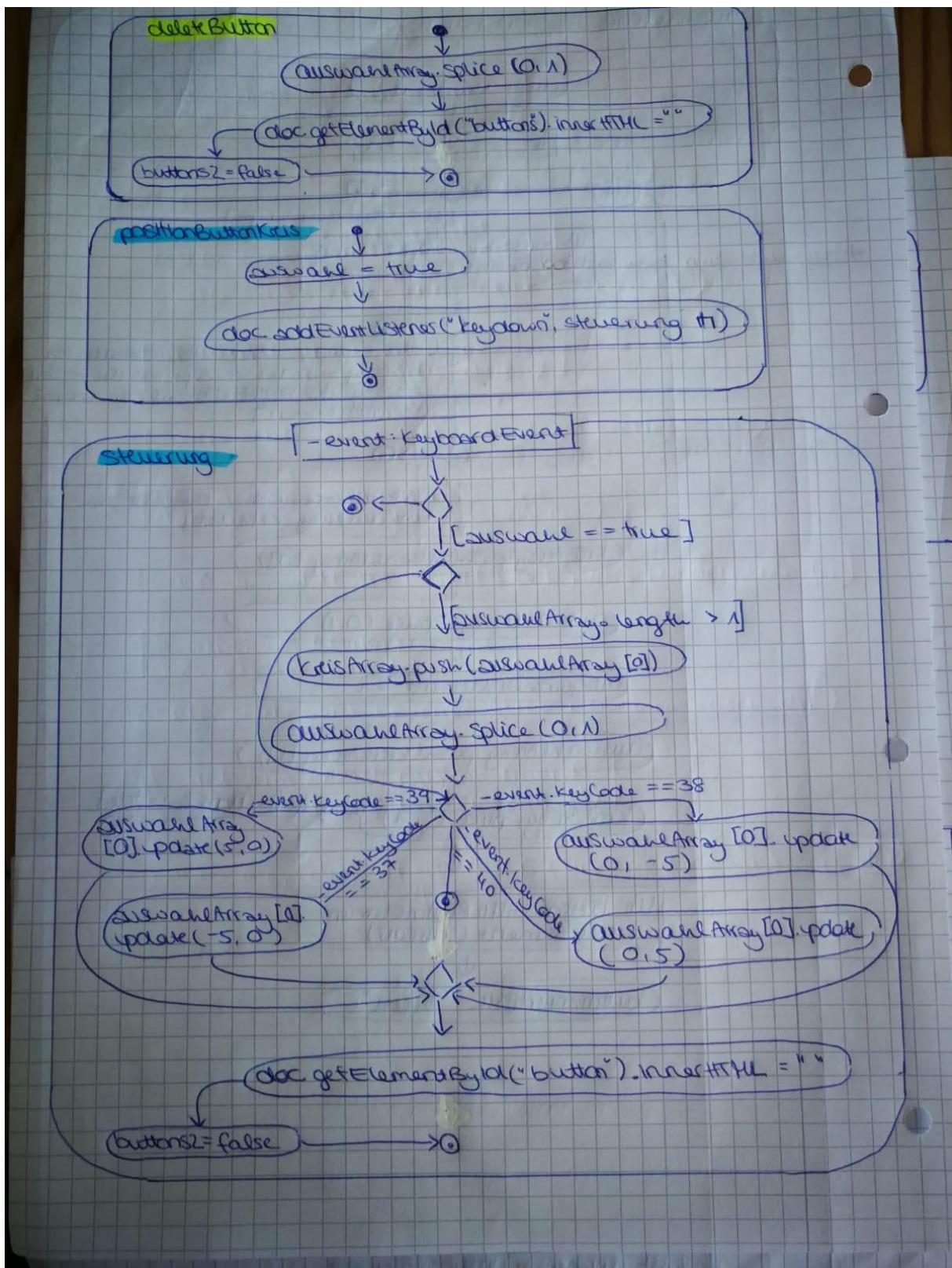
### Klassendiagramm:



## Elemente auswählen + Buttons zum Löschen und verschieben erstellen:

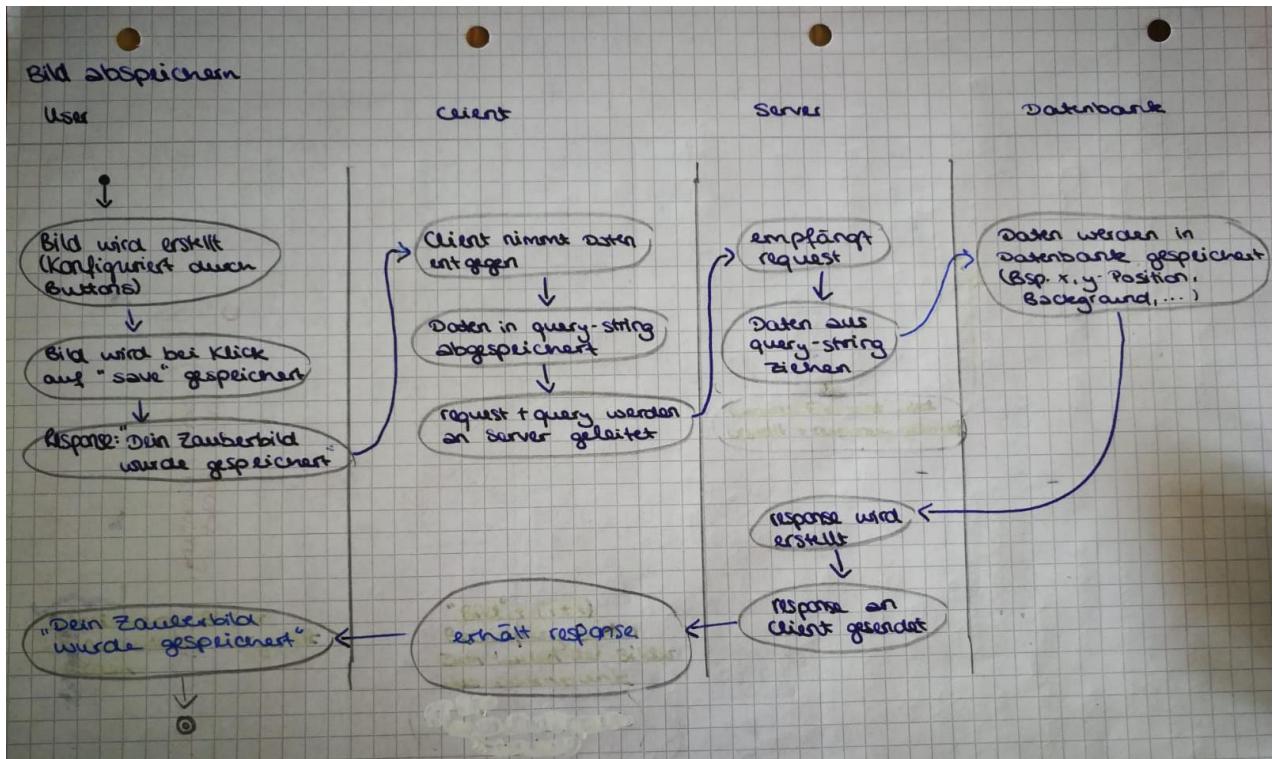


## Funktionen zum Löschen der Elemente und verschieben:

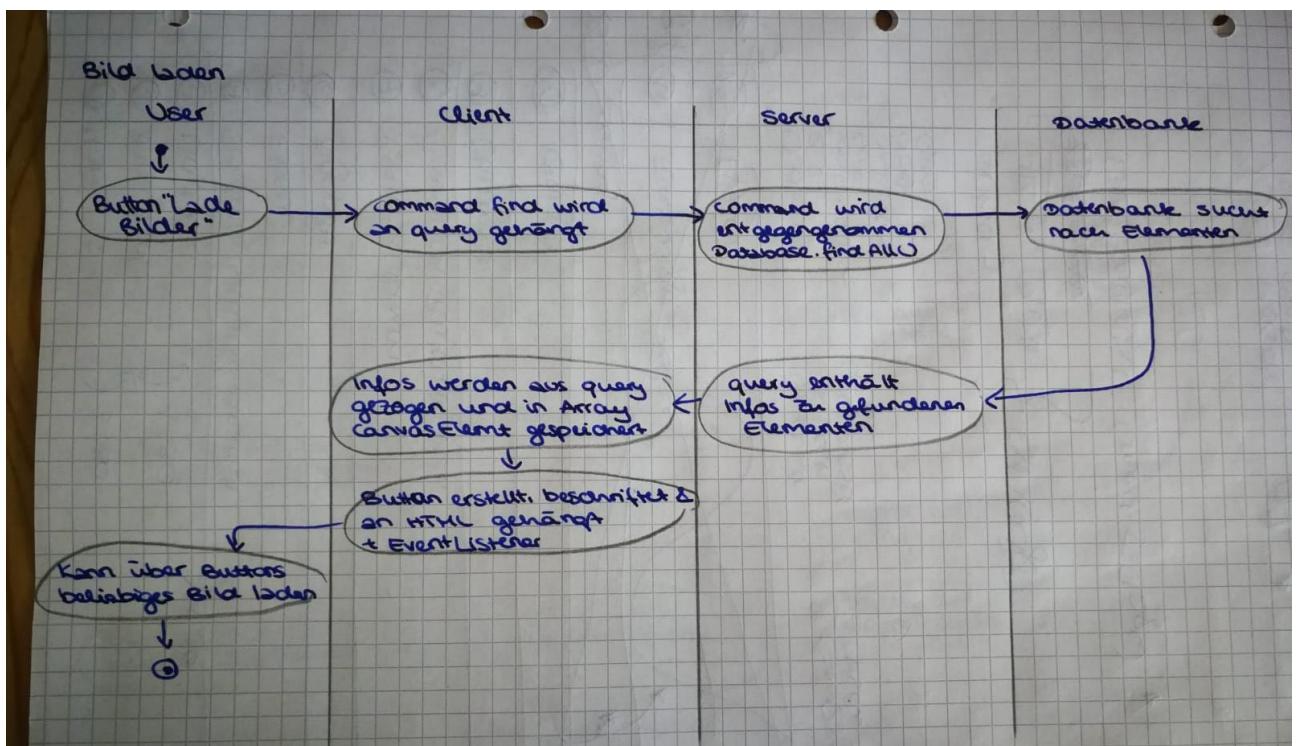


### Domänenübergreifendes Aktivitätsdiagramm:

„Save“



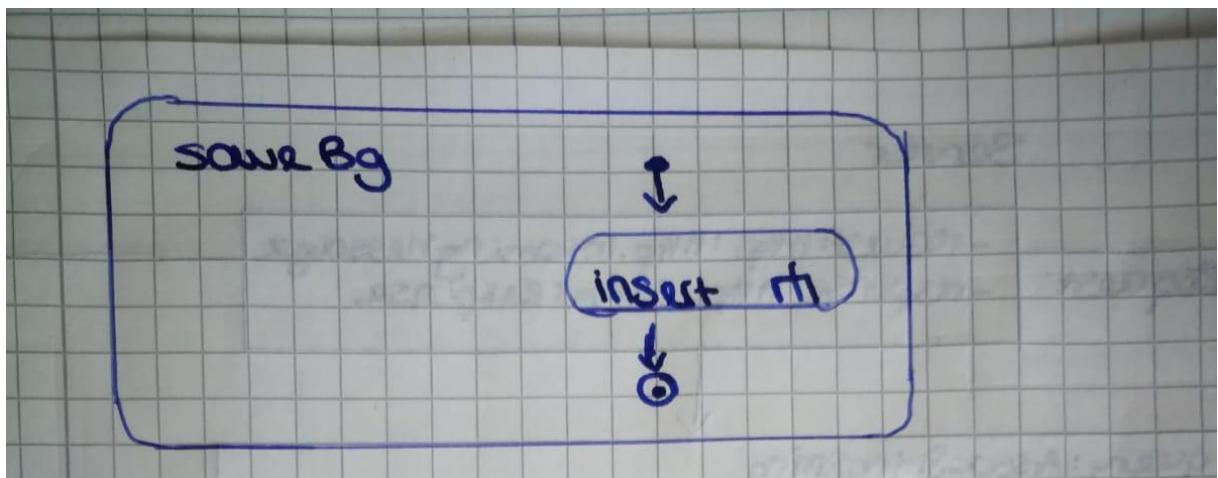
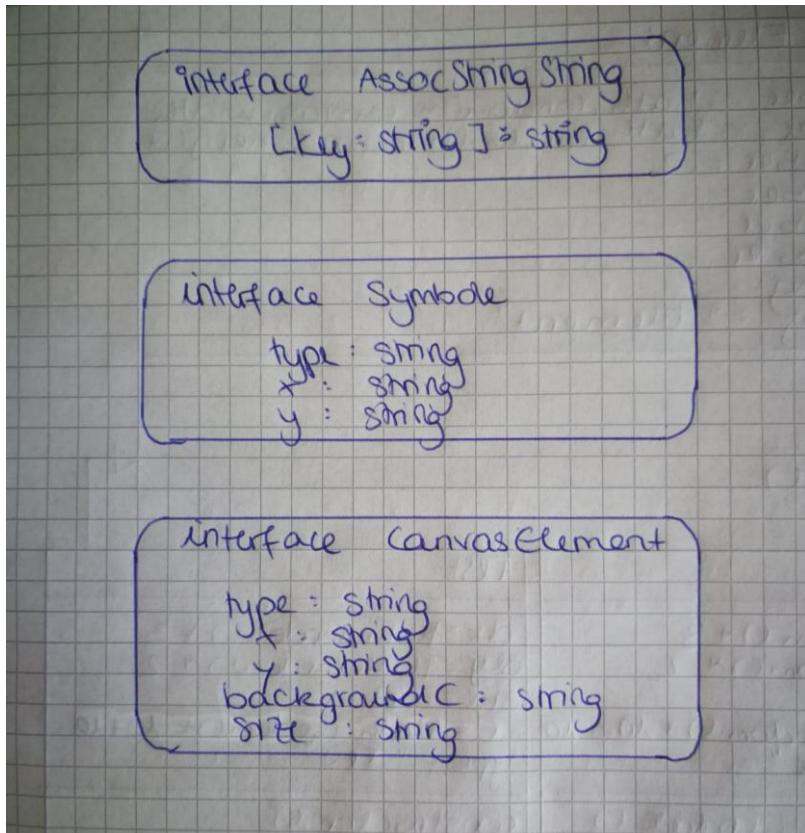
„Bild laden“



Interfaces anlegen:

Symbole: für jedes Element wird ein symbol vom Typ Symbol erstellt

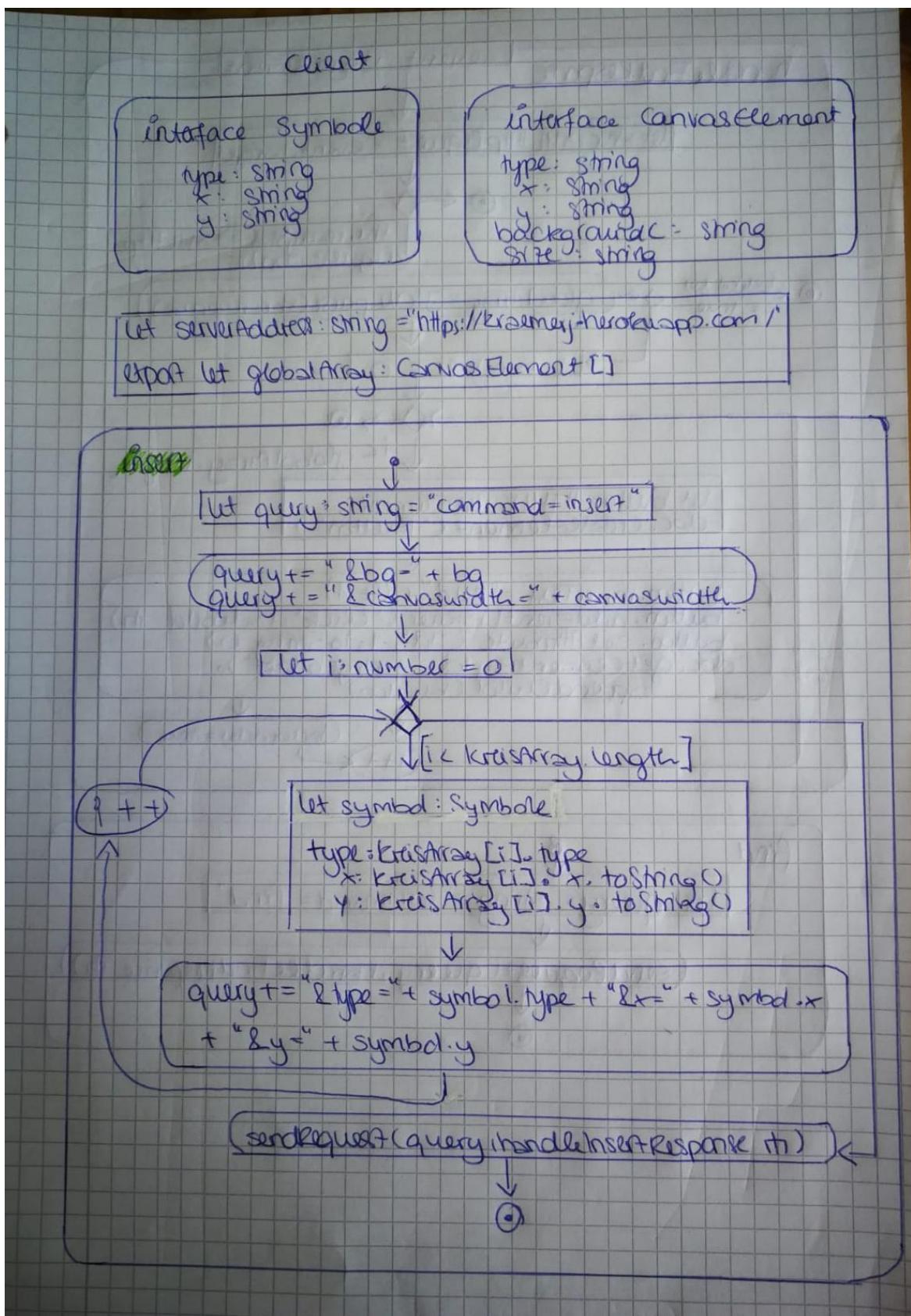
CanvasElement: alle Symbole werden zusammengenommen in „CanvasElement“ in der Datenbank gespeichert



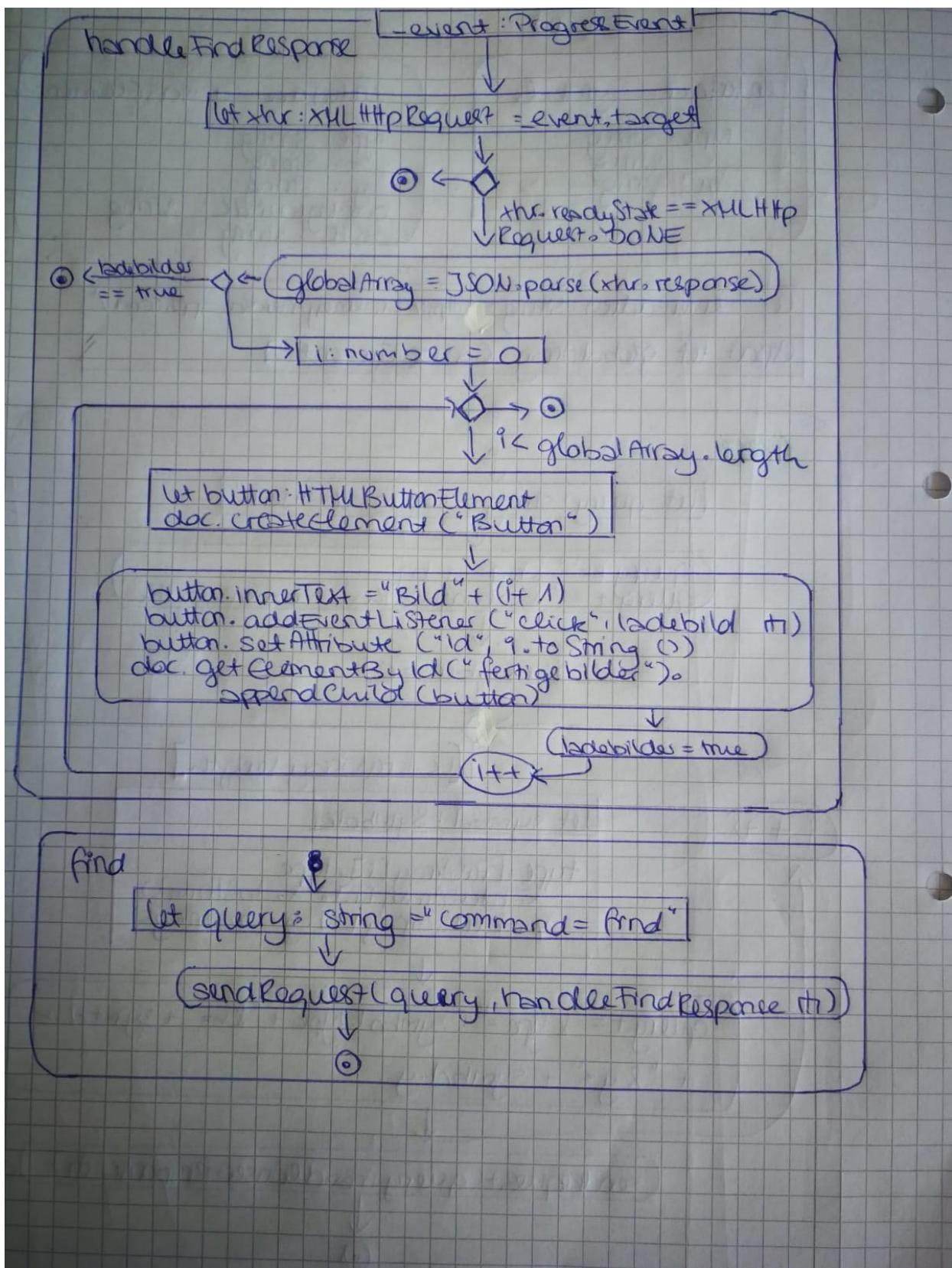
(„Safe“-Button ruft funktion „saveBg“ auf und die wiederrum „insert“:

for-Schleife läuft durch Array und speichert Type, y, und x in query ab: ein symbol vom Typ: „Symbole“ wird angelegt. Zusätzlich werden Farbe und Größe abgespeichert.)

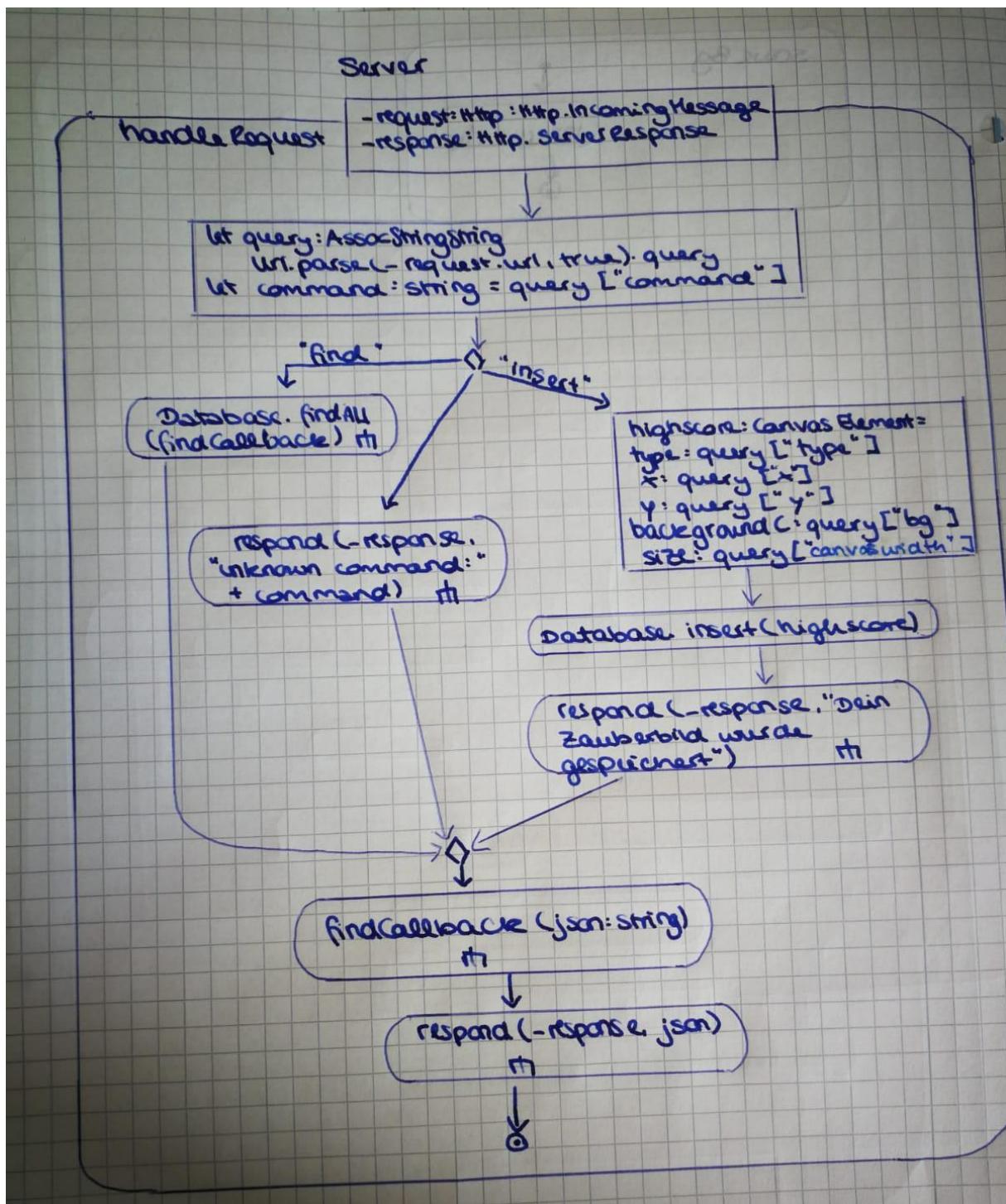
Client: speichert Daten in query



**Buttons werden erstellt, um Bilder zu laden:**



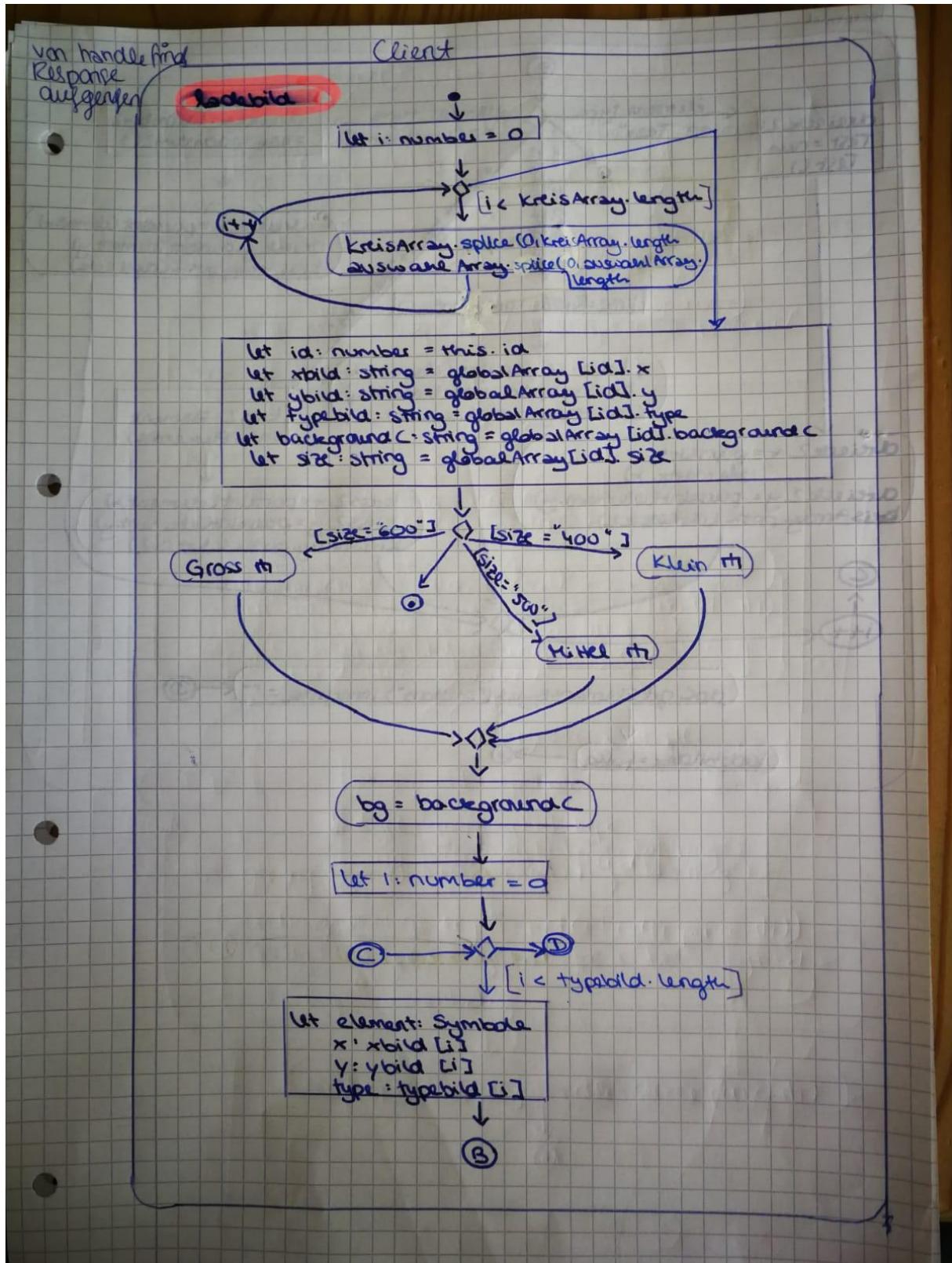
**Server:**



Client: Bilder werden wieder hergestellt:

id = Position des CanvasElement im globalArray -> welcher Button wurde geklickt

(pro Bild ein Array: CanvasElement + Button), (im Array CanvasElement befinden sich mehrere „Symbole“)



Je nach gespeicherter Typ des Symbols wird Element neu gezeichnet:

