# Funktionale Analyse

Plattform: PC

> Bei einer mobilen Version müssen sich zu viele Elemente den Platz teilen. Icons für Farbwahl, Stiftgröße, Scores oder ähnliches sind nicht gut auf einen Blick einsehbar.

Objektinteraktionen:

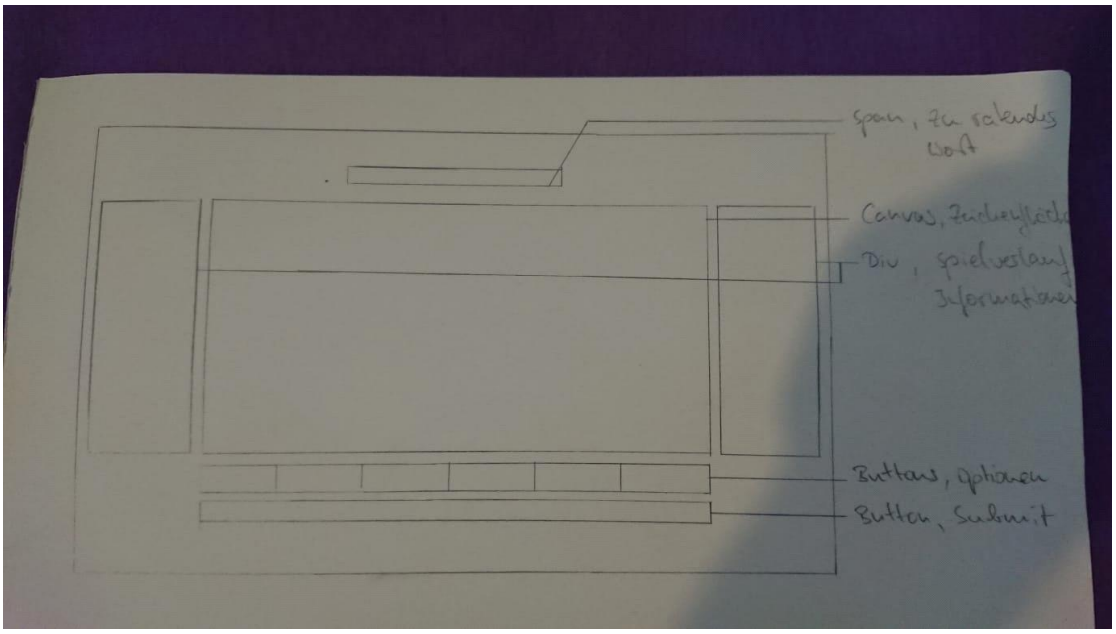
> Es gibt an sich keine beweglichen oder interagierenden Objekte. Die Interfaces variieren je nach Zug, also ob gerade gezeichnet oder geraten wird und werden dynamisch angepasst.

Aufbau & Ablauf:

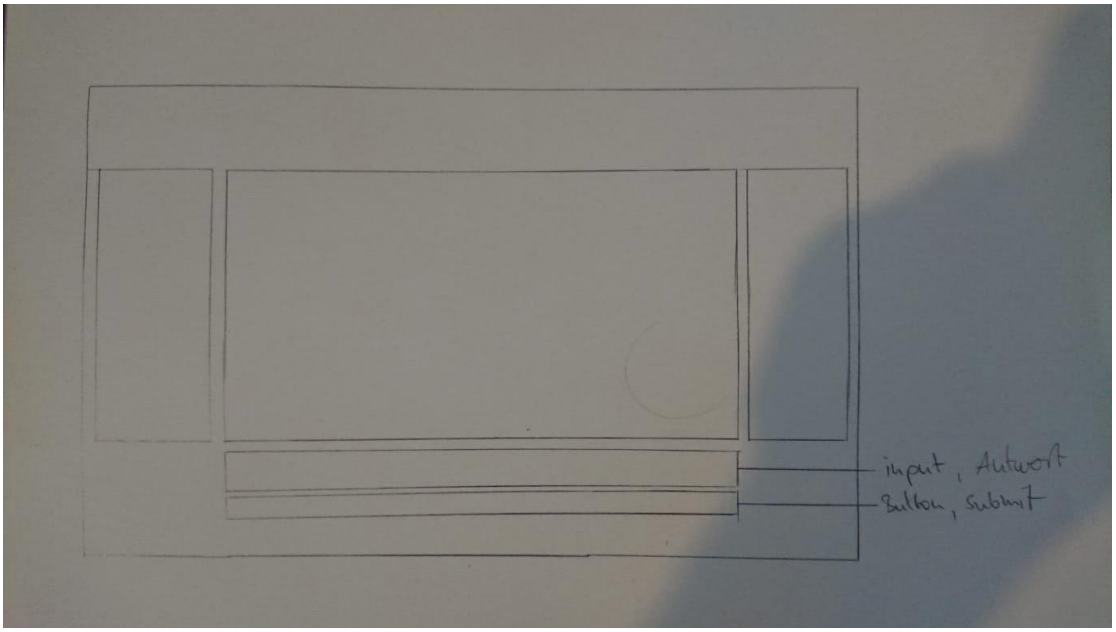
> Der Nutzer startet die Anwendung. Ihm wird ein zufälliges Wort aus der Datenbank angezeigt. Unterhalb des zu zeichnenden Wortes befindet sich eine Zeichenfläche, auf der der Nutzer per Mausklick zeichnen kann. Zudem sind unterhalb der Zeichenfläche Optionen für Interaktionsmöglichkeiten mit der Zeichnung.
>
> Links und rechts von der Zeichenfläche befinden sich Informationen zum Spielverlauf. So zum Beispiel die bereits erratenen Worte aus vergangenen Runden und die letzten Rateversuche.
>
> Der Nutzer ist in der Hinsicht eingeschränkt, dass er nur die zur Verfügung gestellten Farben und Stiftgrößen verwenden kann. Sollte das zu zeichnende Wort sich als zu schwierig erweisen, so kann er dieses Ueberspringen.

Skizzen:

Zeichnen




Raten




Nutzerinteraktion - Reaktion des Systems

Klick auf "NEXT"-Button: Neues Wort wird aus Datenbank gelesen

Klick auf "CLEAR"-Button: canvas wird gecleart

Klick auf "M"-Button: lineWidth wird angepasst

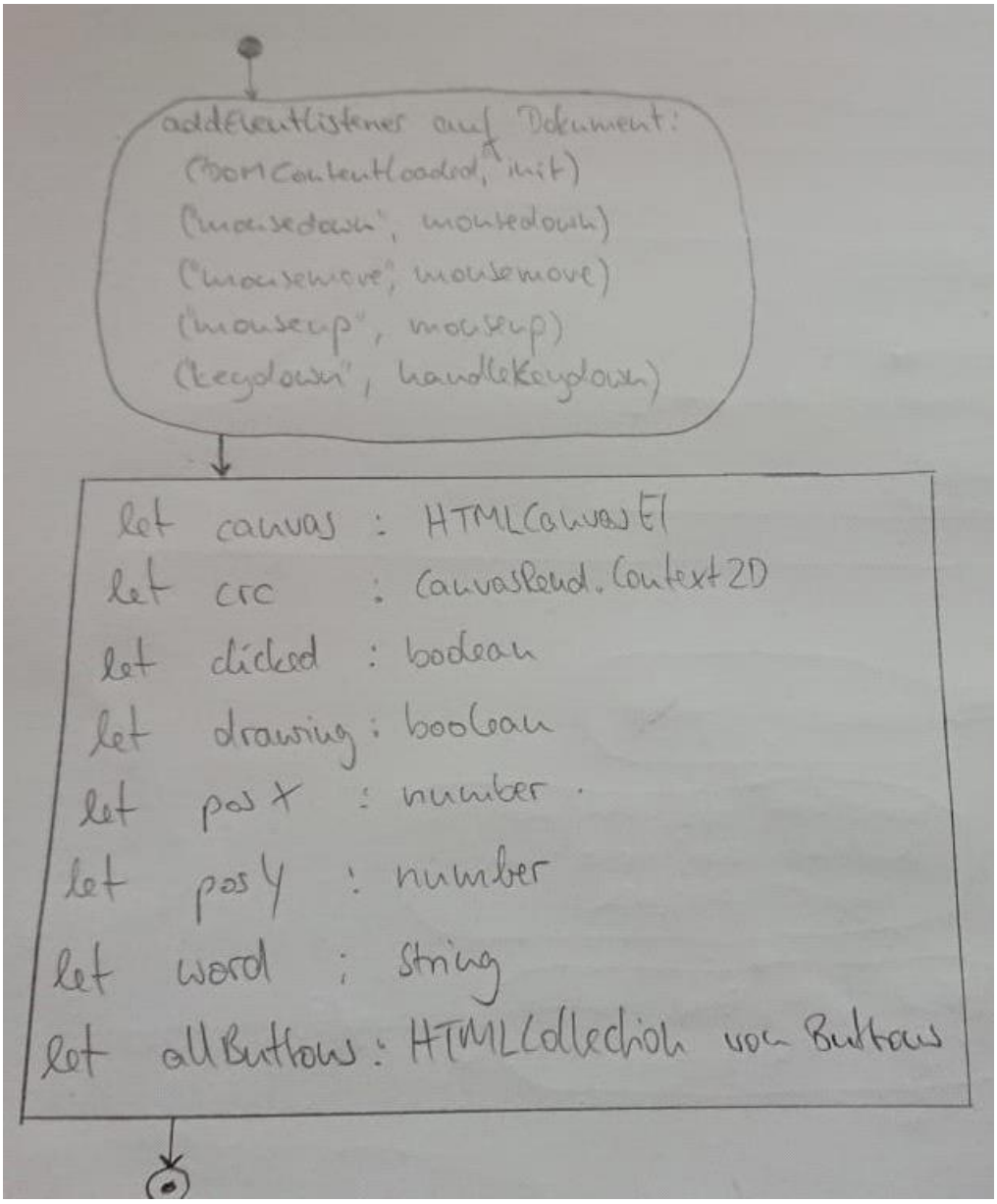
Klick auf roten Button: strokeStyle wird angepasst

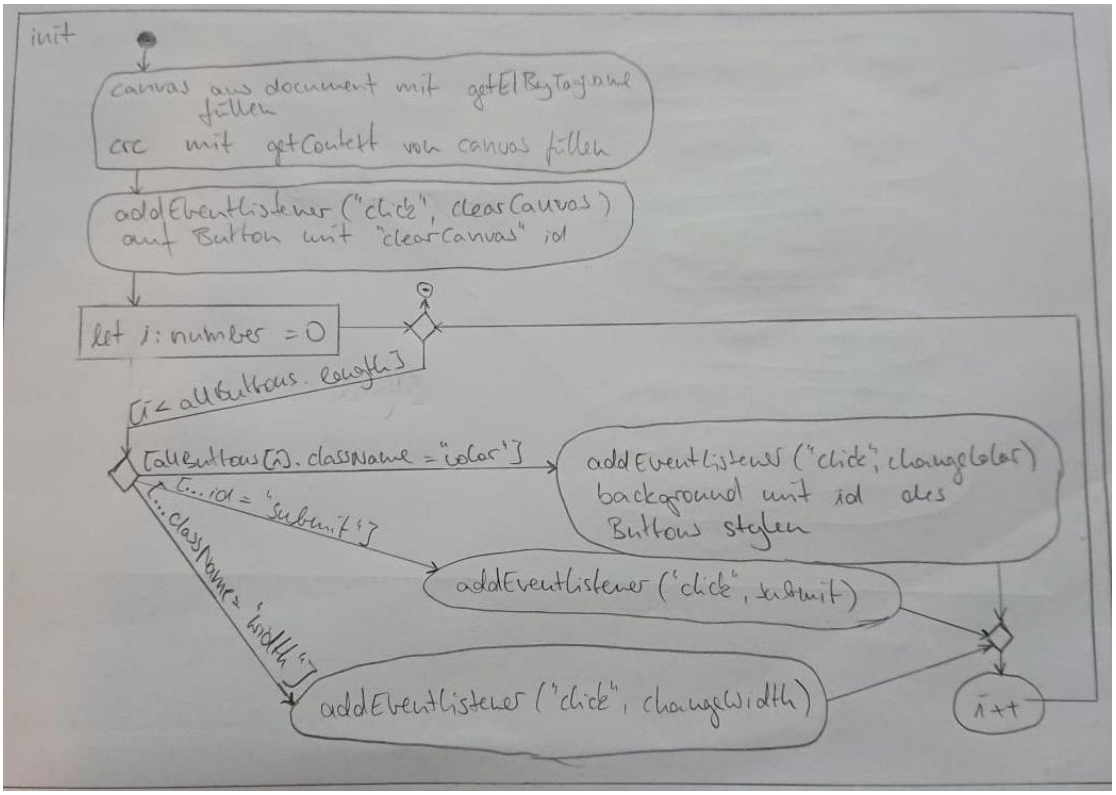
Klick auf Canvas: Mausposition wird ausgelesen und auf Canvas abgebildet

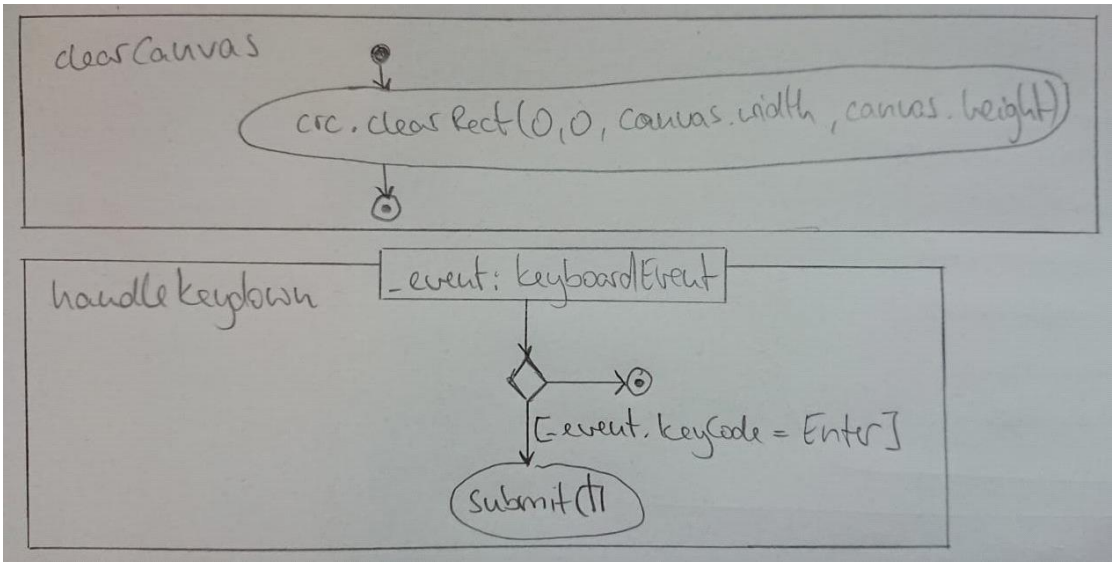
[ENTER]: input.innerText wird mit span.innerText verglichen

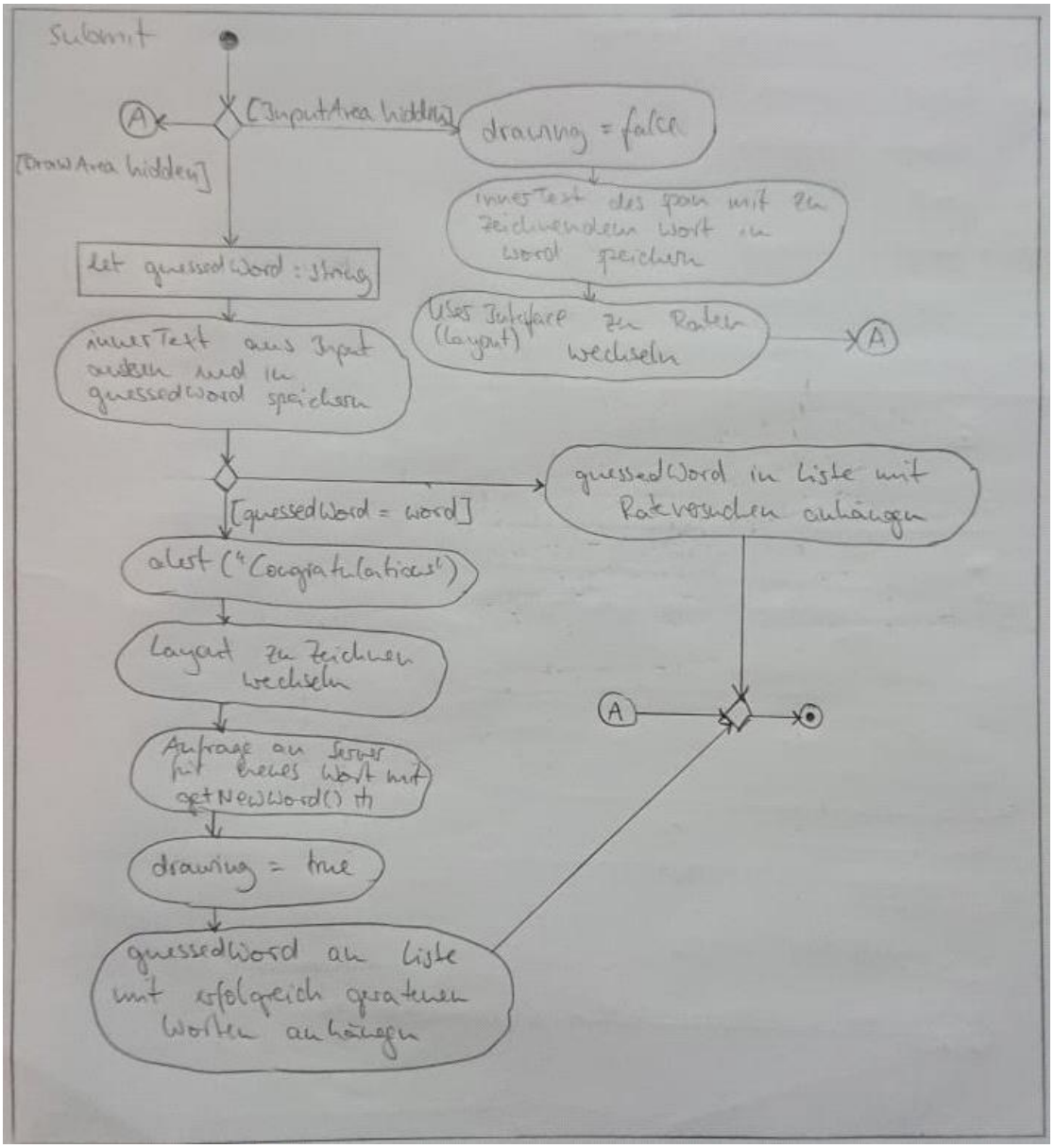
# Technische Analyse

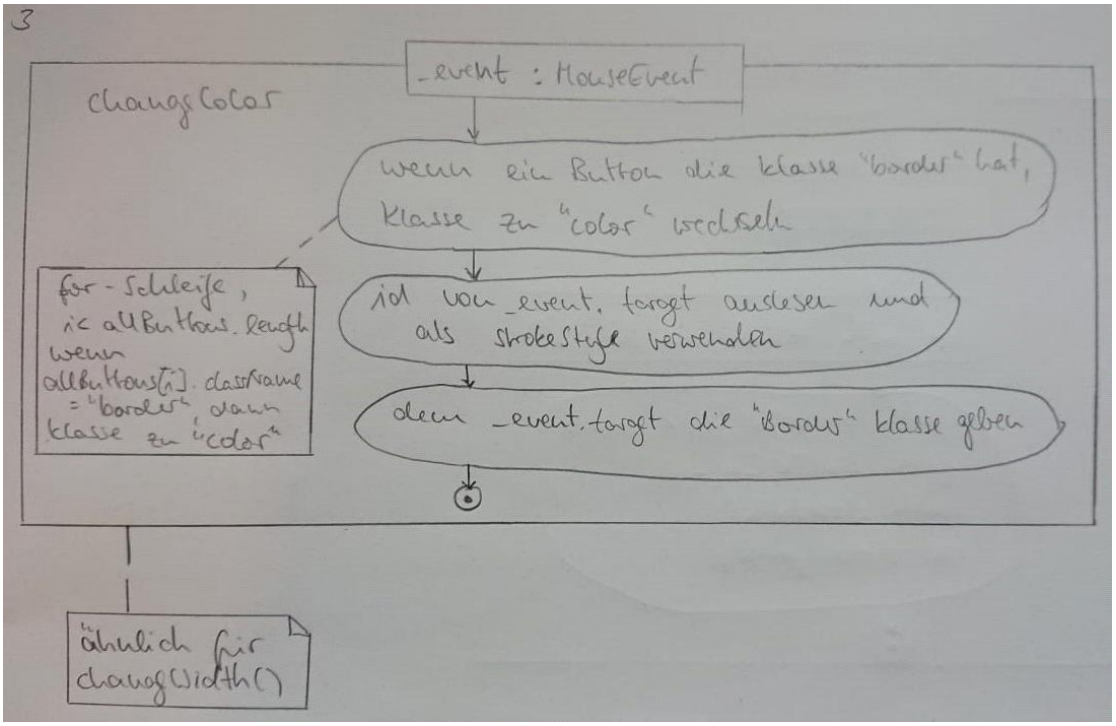
Aktivitätsdiagramme:

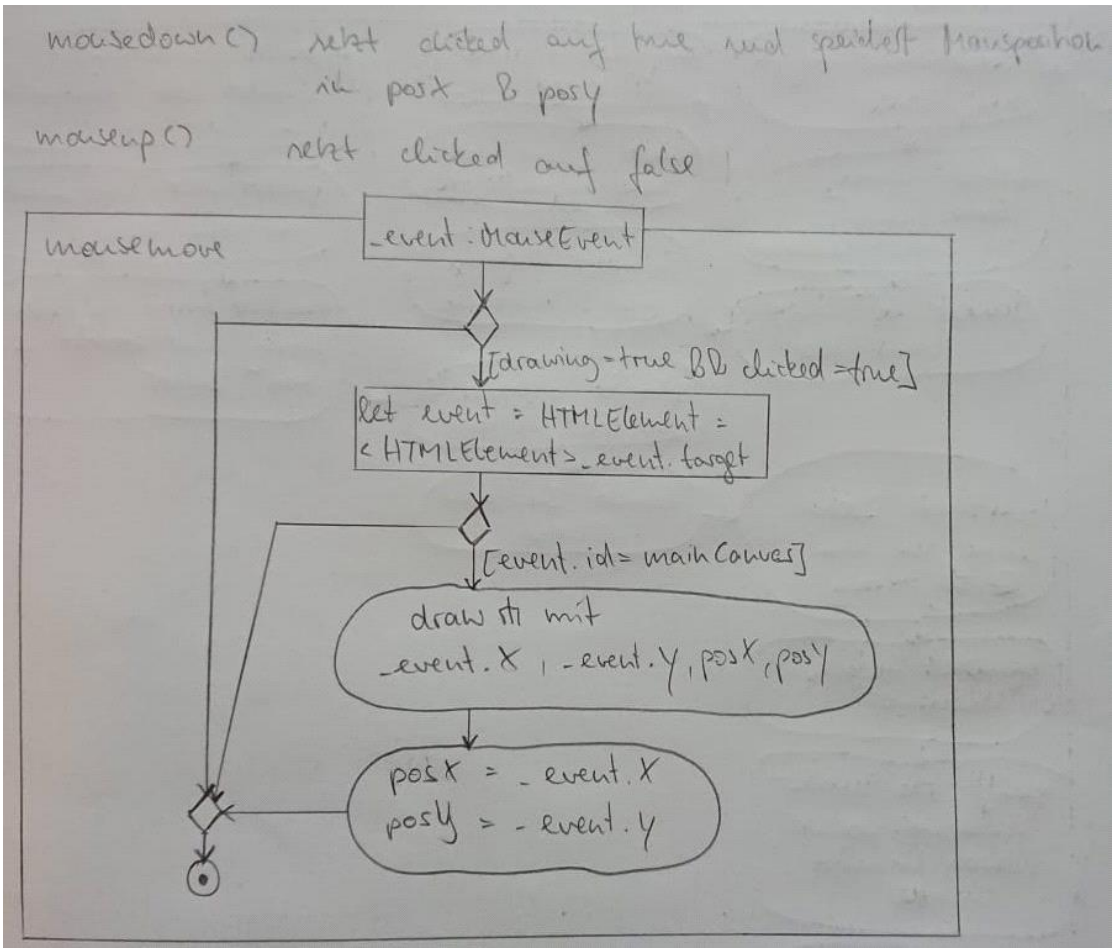


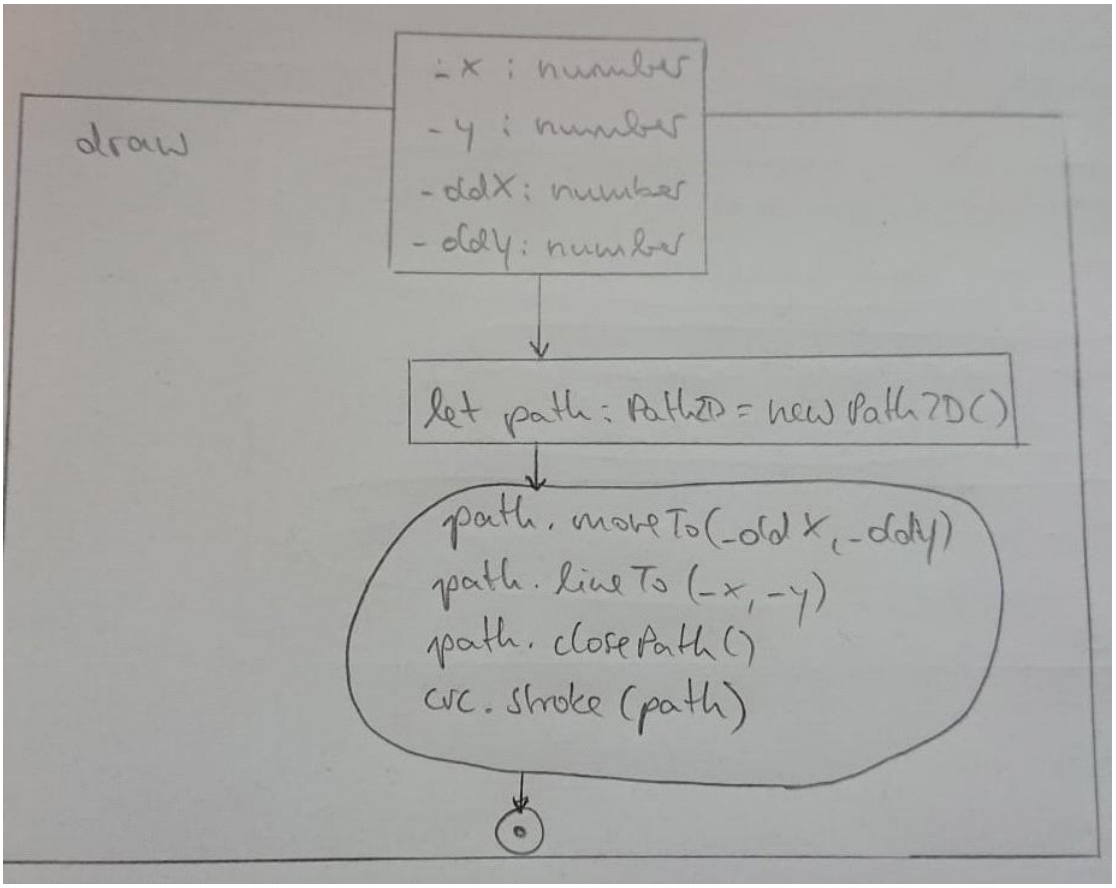

init
canvas aus document mit getElBeyTagName füllen
crc mit getContext von canvas füllen
addEventListener ("click", clearCanvas) auf Button mit "clearCanvas" id
let i: number = 0
[i < allButtons.length]
[allButtons[i].className = "color"]
addEventListener ("click", changeColor) background mit id des Buttons stylen
[... id = 'submit']
addEventListener ("click", submit)
[... className = 'width']
addEventListener ("click", changeWidth)
i++

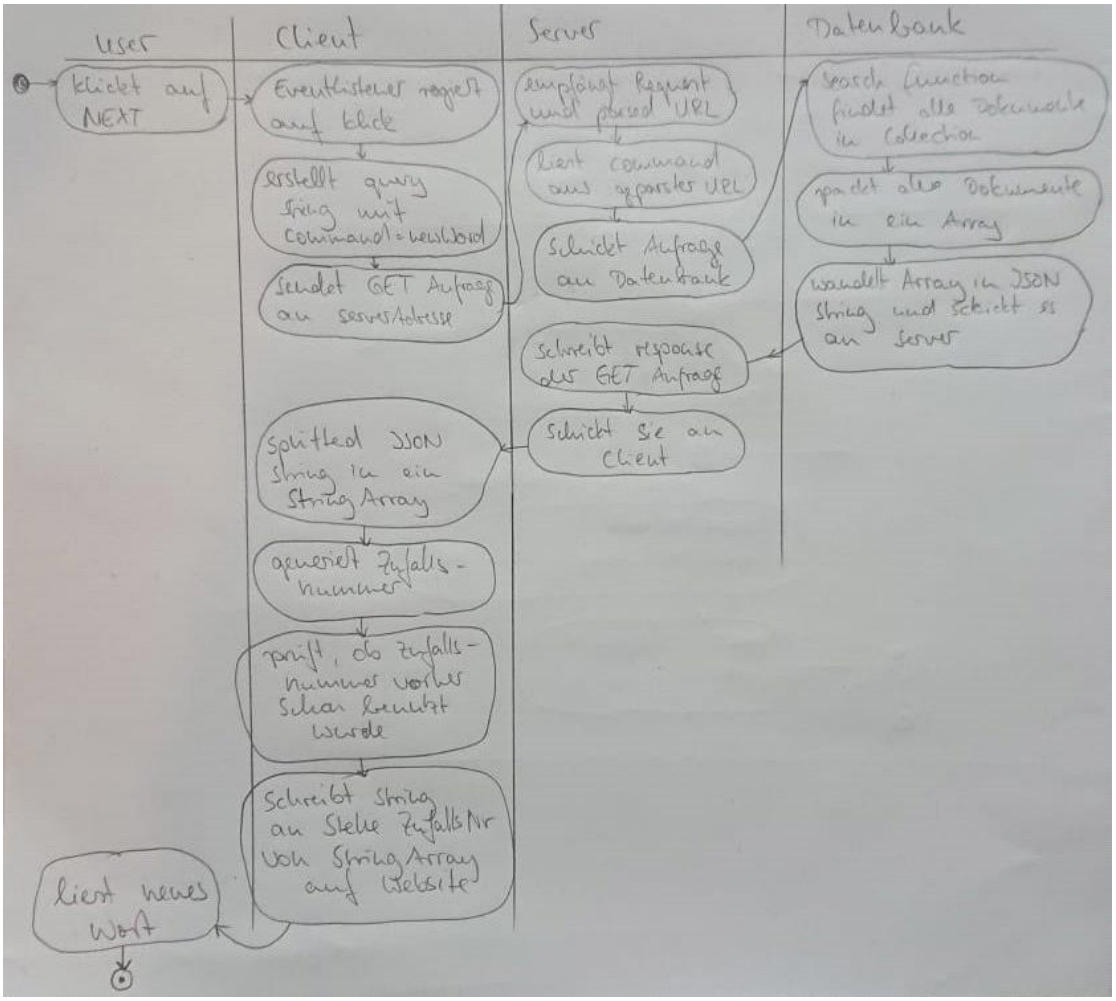

clearCanvas
crc.clearRect(0,0, canvas.width, canvas.height)
handleKeydown
_event: keyboardEvent
[_event.keyCode = Enter]
submit()

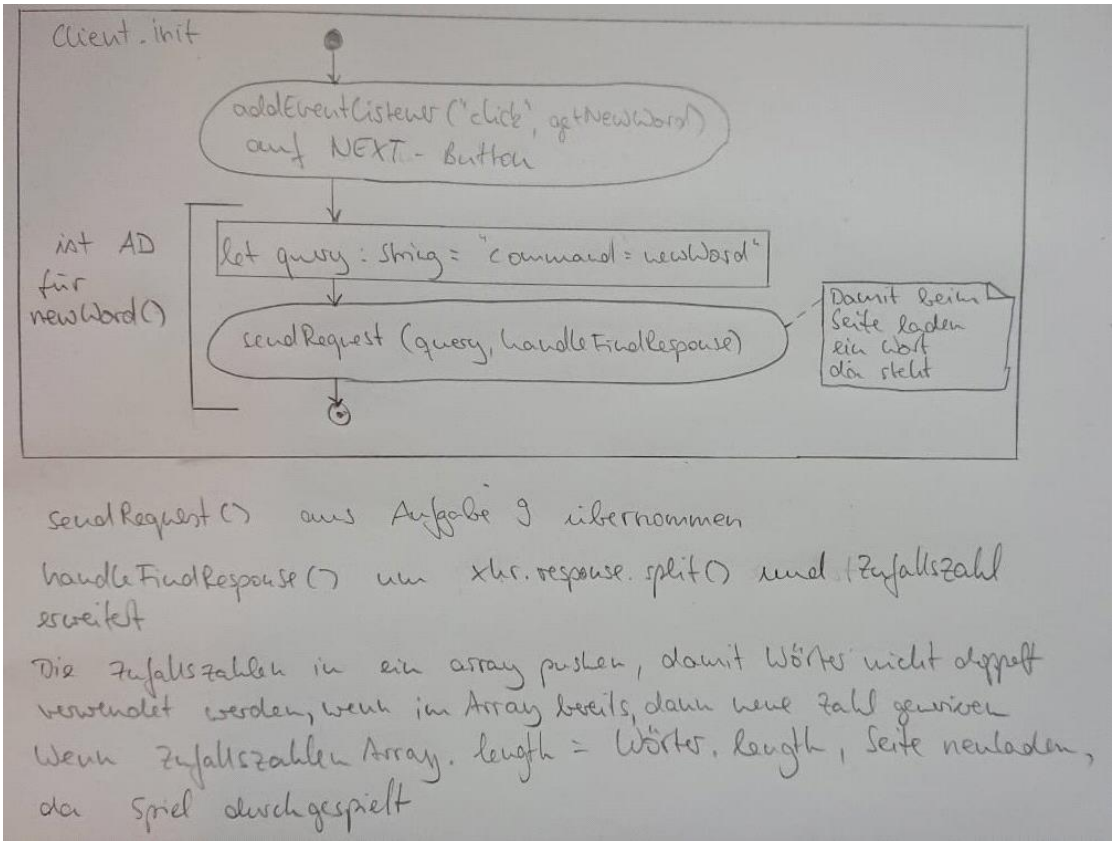

Submit
A
[InputArea hidden]
drawing = false
[DrawArea hidden]
innerText des Span mit zu zeichnendem Wort in word speichern
let guessedWord : String
Ui.setInterface zu Raten (Layout) wechseln
A
innerText aus Input auslesen und in guessedWord speichern
guessedWord in Liste mit Rateversuchen anhängen
[guessedWord = word]
alert ("Congratulations!")
Layout zu Zeichnen wechseln
A
Anfrage an Server mit neuem Wort mit getNewWord() th
drawing = true
guessedWord an Liste mit erfolgreich geratenen Worten anhängen


3
changeColor
_event : MouseEvent
wenn ein Button die Klasse "border" hat, Klasse zu "color" wechseln
for-Schleife, ic allButtons.length wenn allButtons[i].className = "border", dann Klasse zu "color"
id von _event.target auslesen und als strokeStyle verwenden
dann _event.target die "border" Klasse geben
ähnlich für changeWidth()


mousedown() setzt clicked auf true und speichert Koordinaten in posX & posY
mouseup() setzt clicked auf false
mousemove
_event : MouseEvent
[drawing=true && clicked=true]
let event = HTMLElement = <HTMLElement>_event.target
[event.id= mainCanvas]
draw sth mit _event.X, _event.Y, posX, posY
posX = _event.X
posY = _event.Y


draw
- x : number
- y : number
- oldX : number
- oldY : number
let path : Path2D = new Path2D()
path.moveTo(-oldX, -oldY)
path.lineTo(-x, -y)
path.closePath()
ctx.stroke(path)


User
Client
Server
Datenbank
klickt auf NEXT
Eventlistener reagiert auf klick
erstellt query String mit command = newWord
sendet GET Anfrage an server/Adresse
empfängt Request und parsed URL
liest command aus geparstes URL
schickt Anfrage an Datenbank
Search function findet alle Dokumente in Collection
packt alle Dokumente in ein Array
wandelt Array in JSON String und schickt es an Server
schreibt response als GET Anfrage
schickt sie an Client
splittet JSON String in ein String Array
generiert Zufalls-nummer
prüft, ob Zufalls-nummer vorher schon benutzt wurde
schreibt String an Stelle Zufallsnr von String Array auf Website
liest neues Wort




sendRequest() aus Aufgabe 9 übernommen

handleFinalResponse() um xhr.response.split() und Zufallszahl erweitert

Die Zufallszahlen in ein array pushen, damit Wörter nicht doppelt verwendet werden, wenn im Array bereits, dann neue Zahl generiert

Wenn Zufallszahlen Array.length = Wörter.length, Seite neuladen, da Spiel durchgespielt