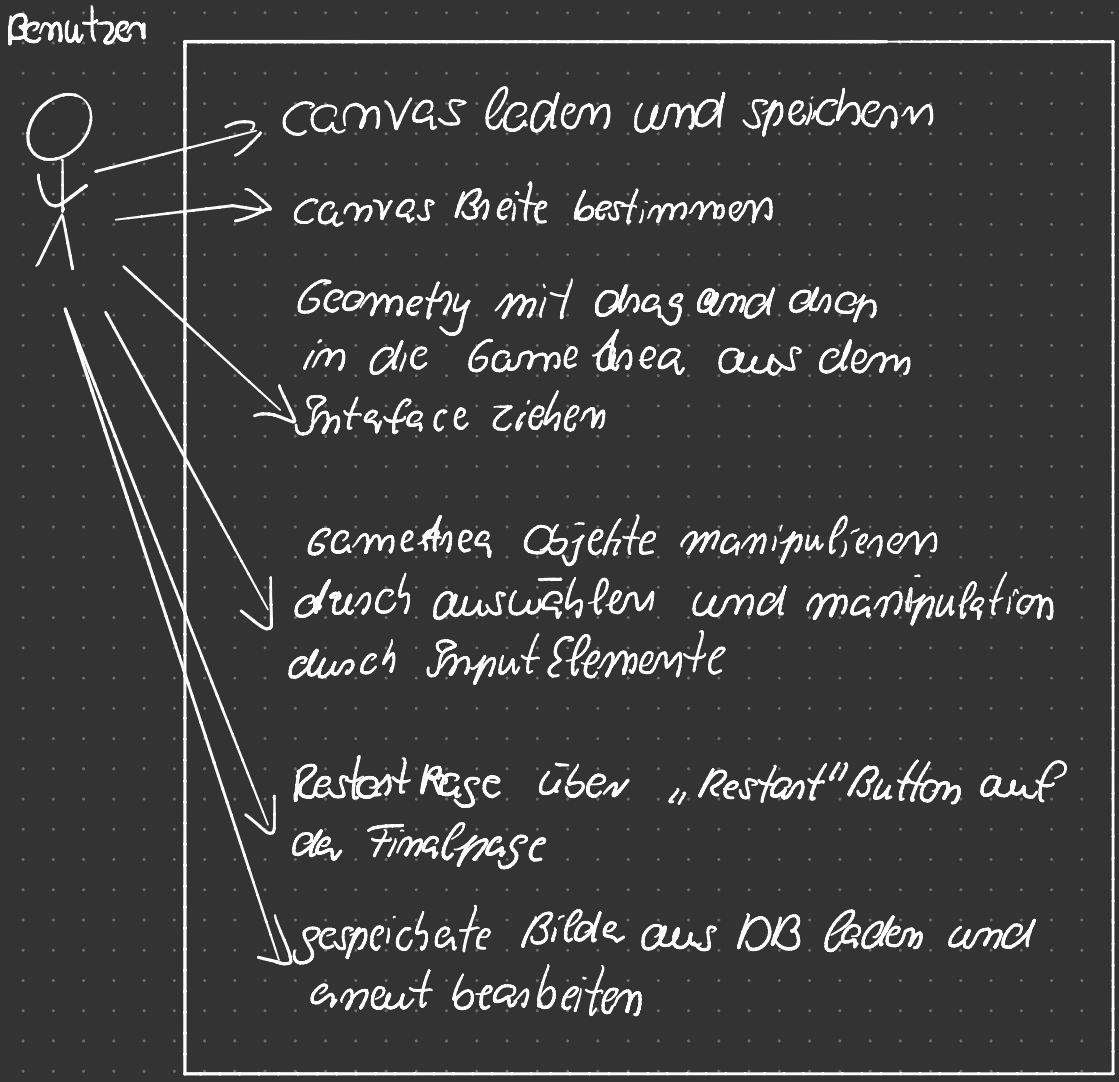
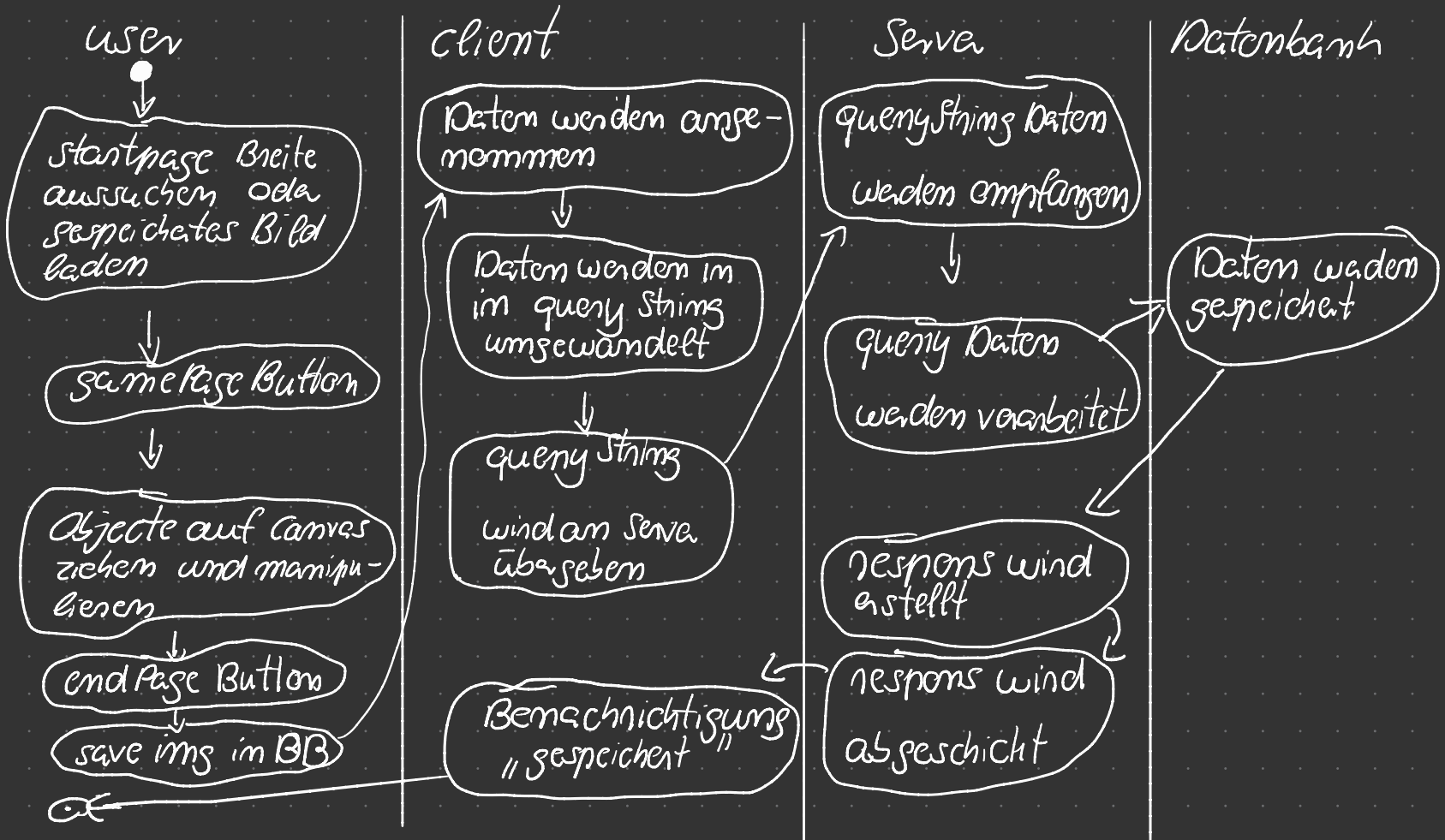
Use Case Diagramm:



Dom übergreifendes Aktivitätsdiagramm



StartPage

wähle ein gespeichertes Bild

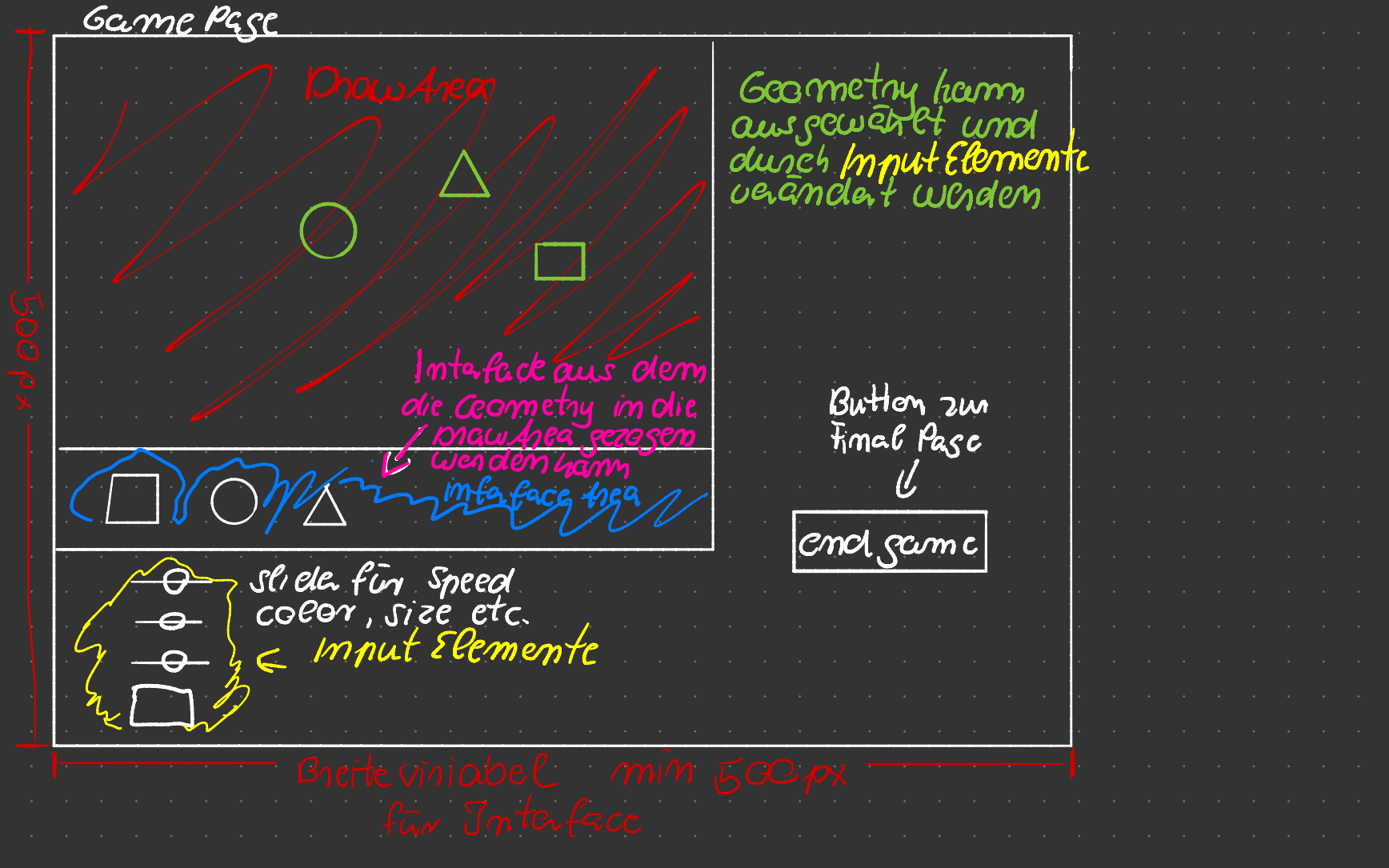
- ○ Bild 1
- ○ Bild 2
- ○ Bild 3

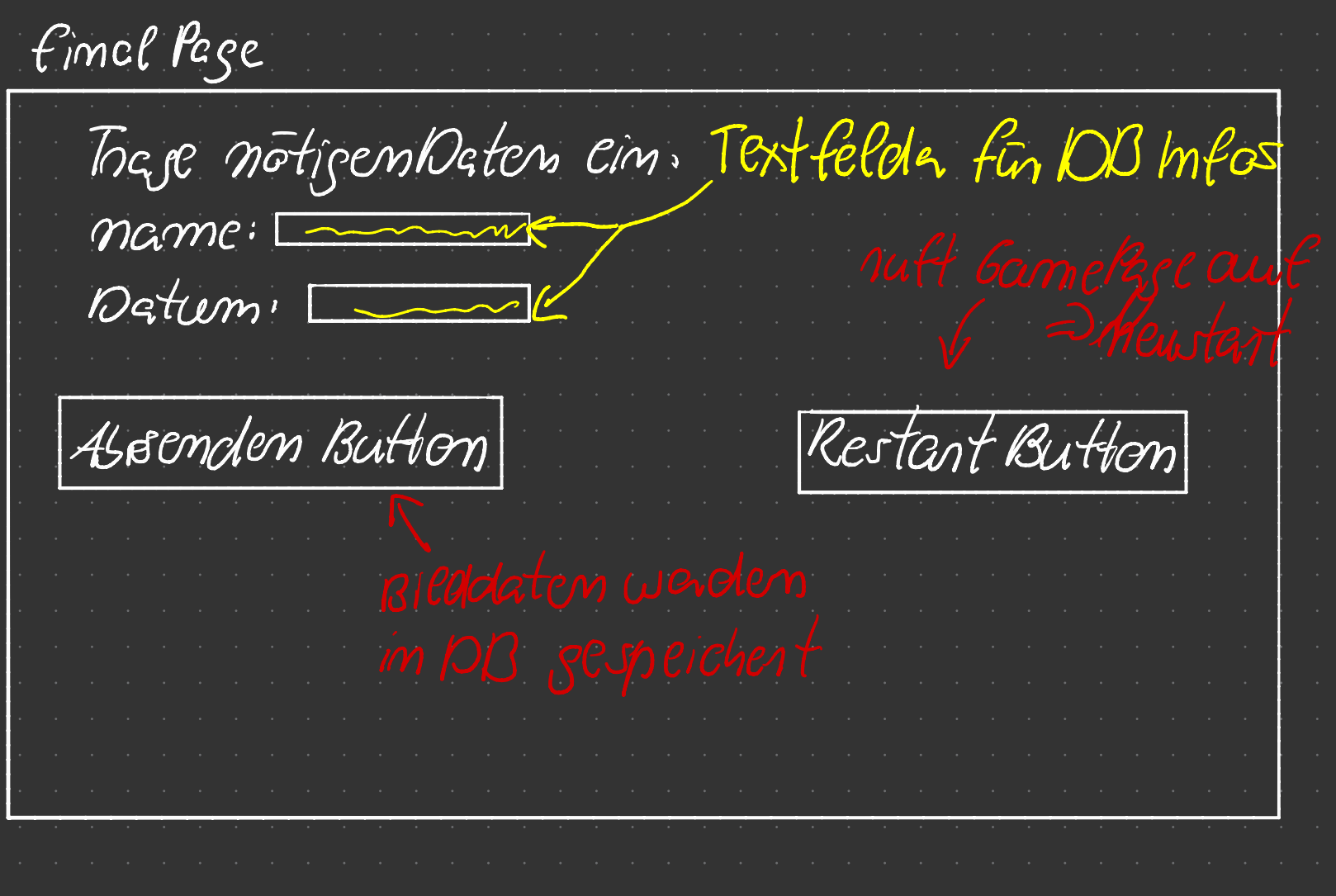
oder die Breite der neuen Leinwand mit dem Slider

Breite:

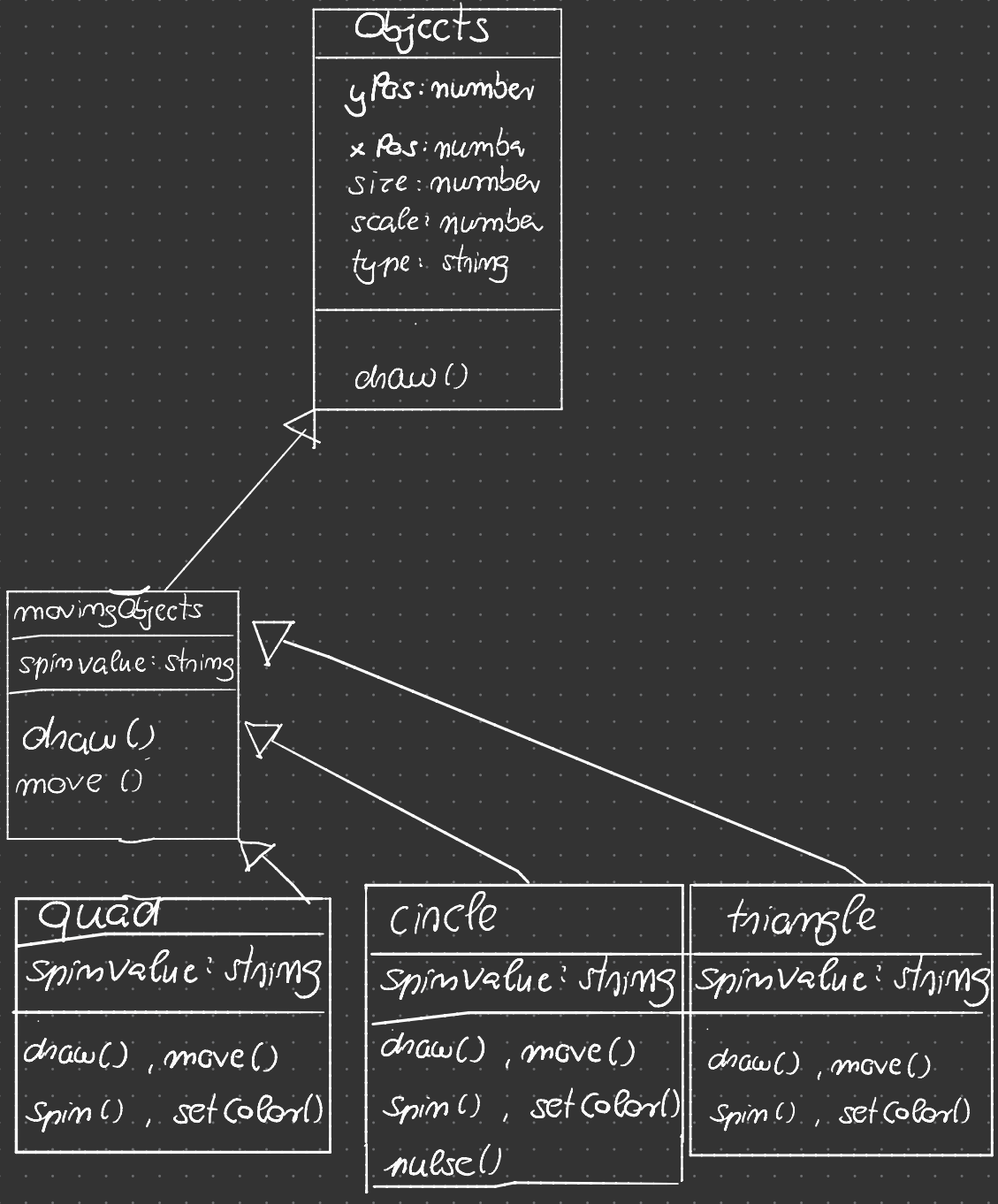
Button To Gamepage

Speichert die Auswahl des Users und startet GamePage


Game Page
Draw Area
Geometry kann ausgewählt und durch Input Elemente verändert werden
Interface aus dem die Geometry im die Draw Area gezogen werden kann
Interface Area
Button zur Final Page
end game
slide für Speed color, size etc.
Input Elemente
500 px
Breite variabel min 500 px
für Interface


final Page
Trage nötigen Daten ein:
Textfelder für DB Infos
name:
Datum:
ruft GamePage auf
↓ ⇒ Neustart
Absenden Button
Restart Button
Bilddaten werden
im DB gespeichert

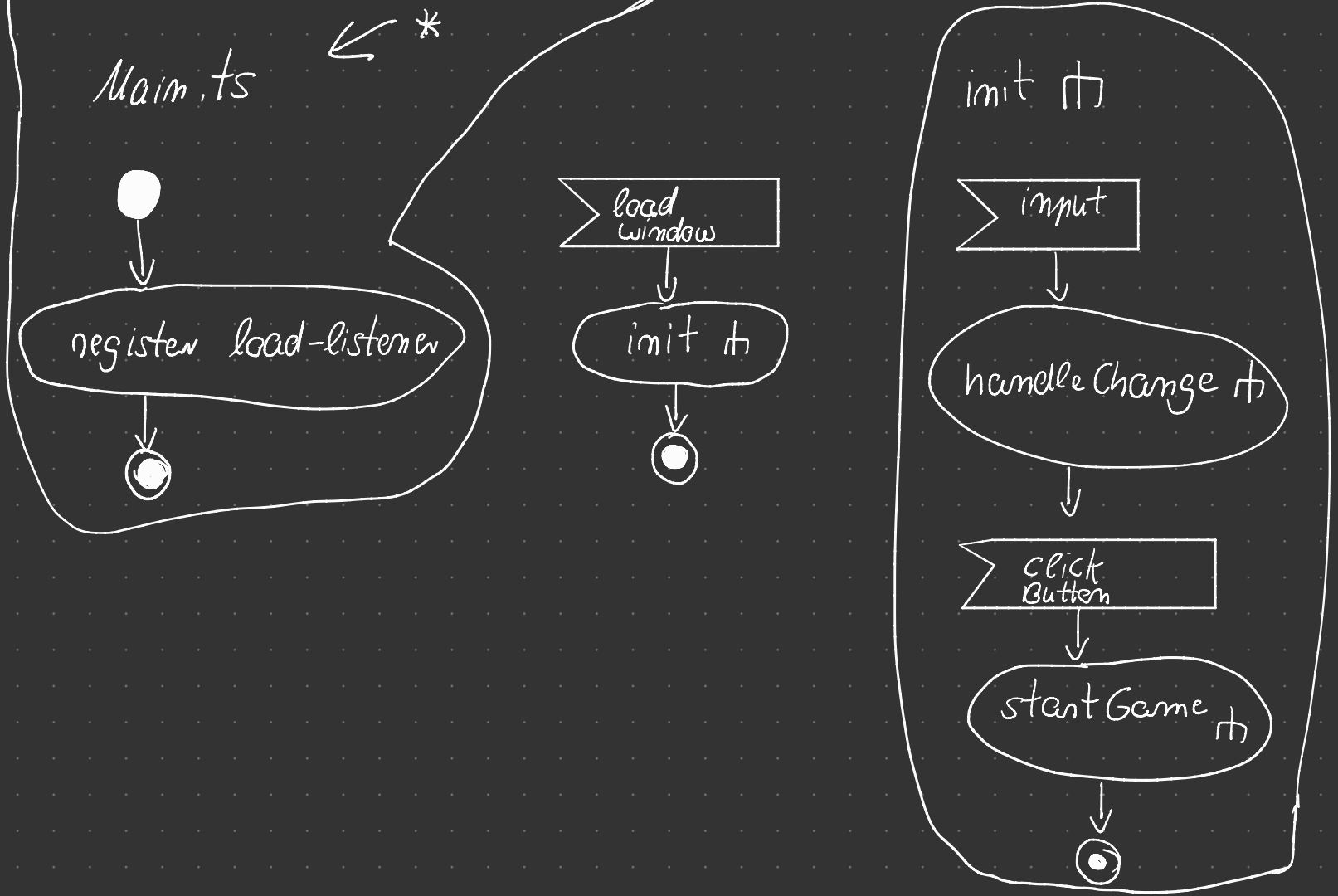
Klassen diagram

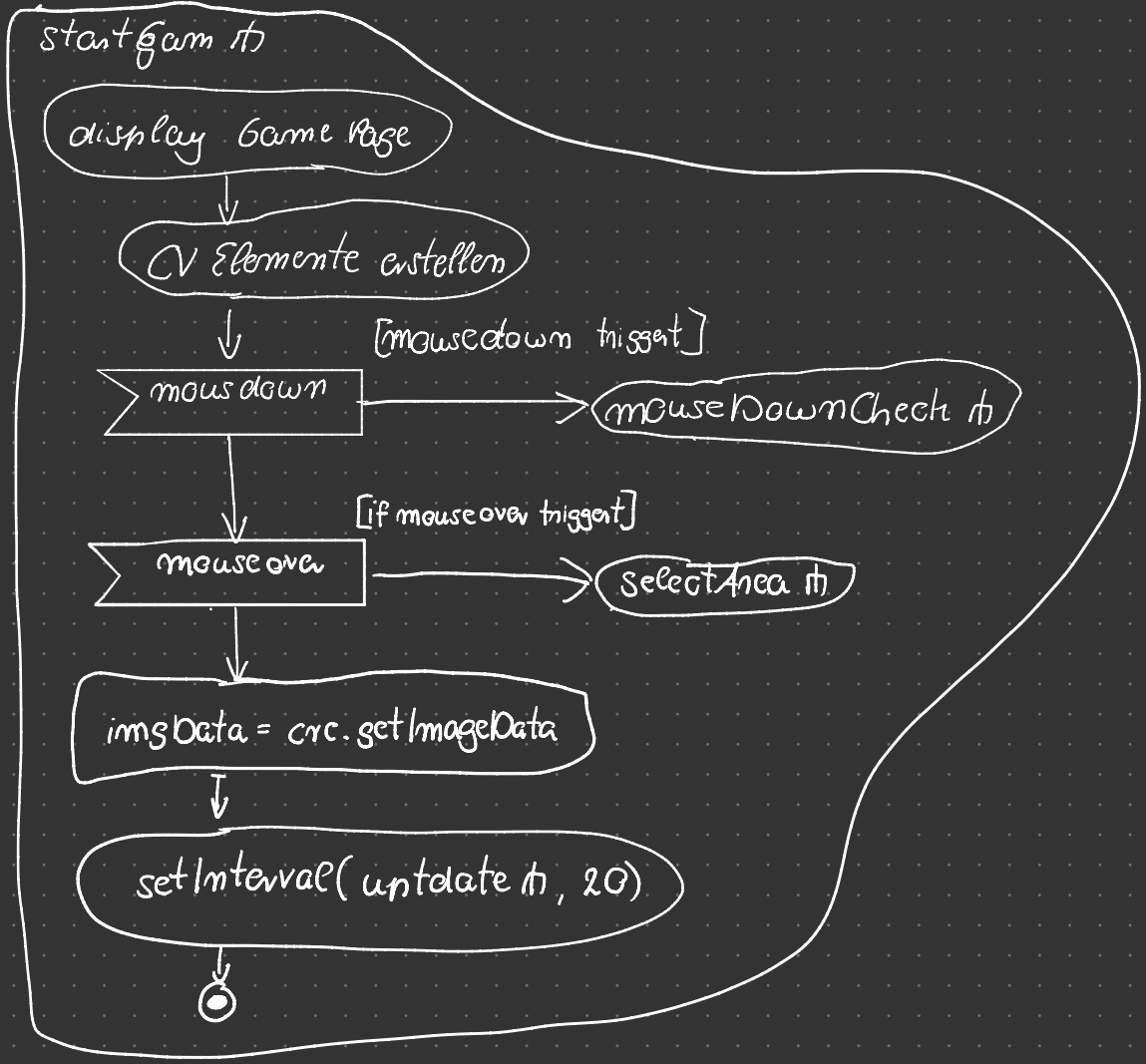


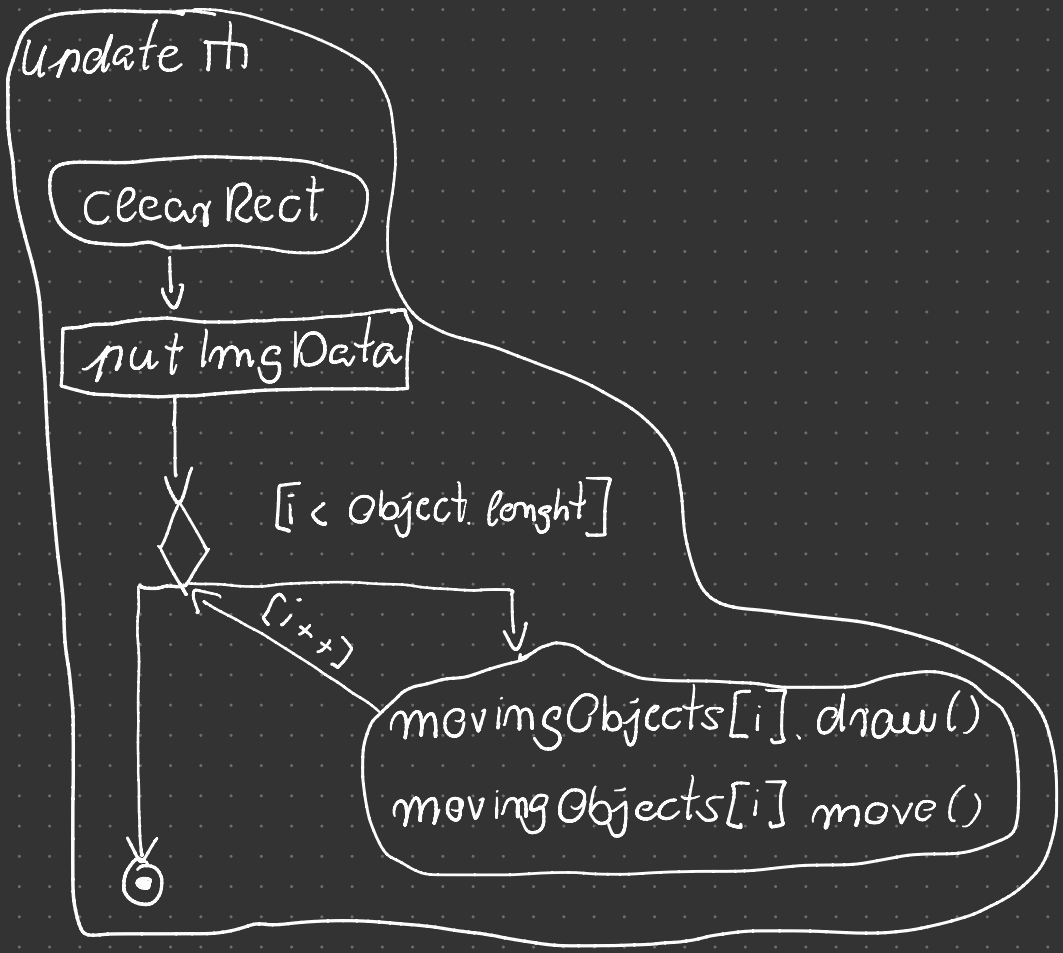
```
export let crc: canvasRenderingContext2D
expat let canvas: HTMLCanvasElement
let imgData
expat let interface: HTMLCanvasElement
export let canvasWidth: numbe
expat let canvasHeight: numbe
expat let quads: Quad[]
"  "  circles: Circle[]
"  "  triangles: Triangle[]
"  "  movingObjects: MovingObject[]
"  "  objects: Object[]
"  "  interfaceObjects: MovingObject[]
```

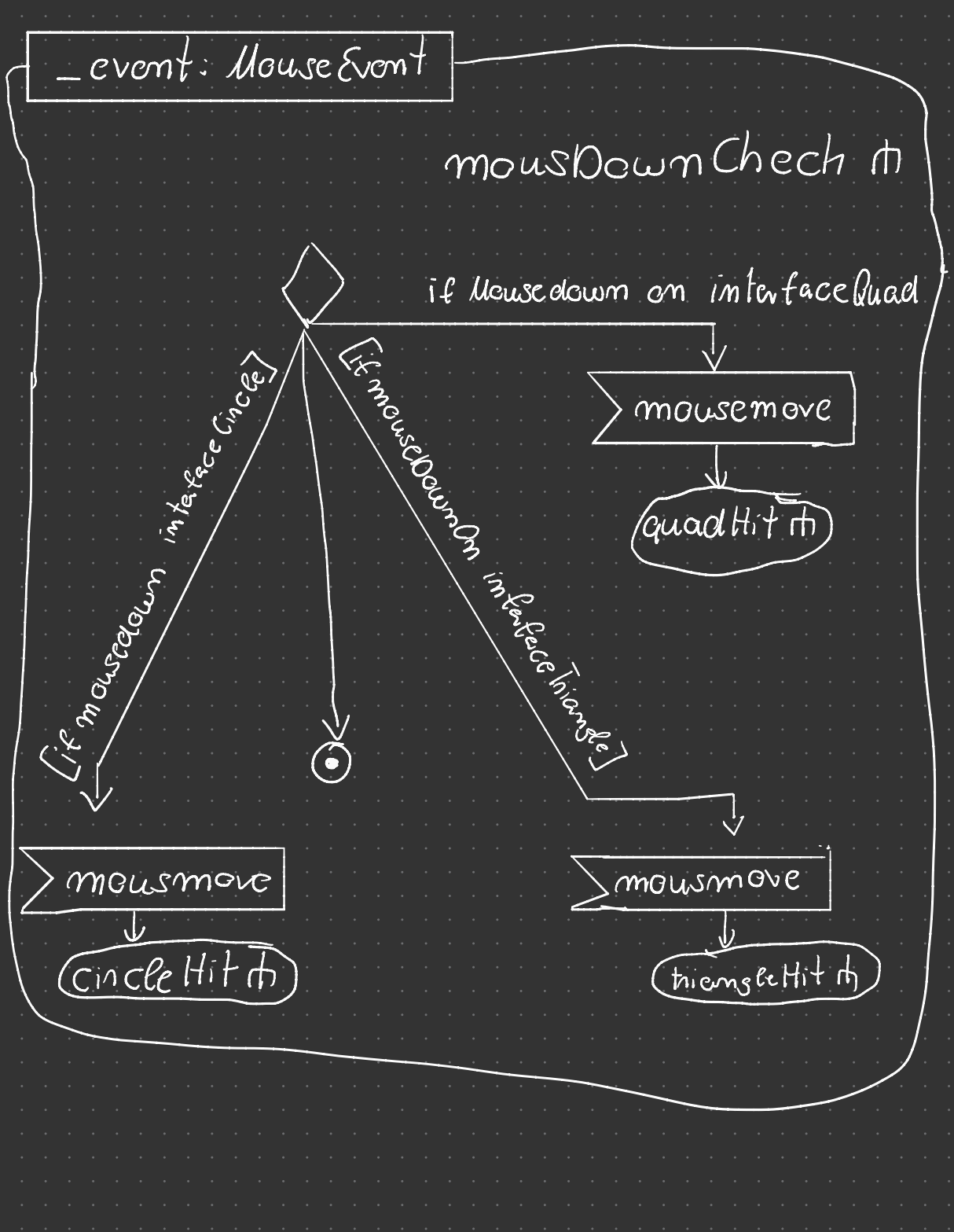
```
let startPage: HTMLElement
let gameOverPage: "
let finalPage: "
let startButton: HTMLButtonElement
let finalButton: "
let slider_xspeed: HTMLInputEle
let slider_yspeed: "
let scale: "
let color: "
let spin: "
```

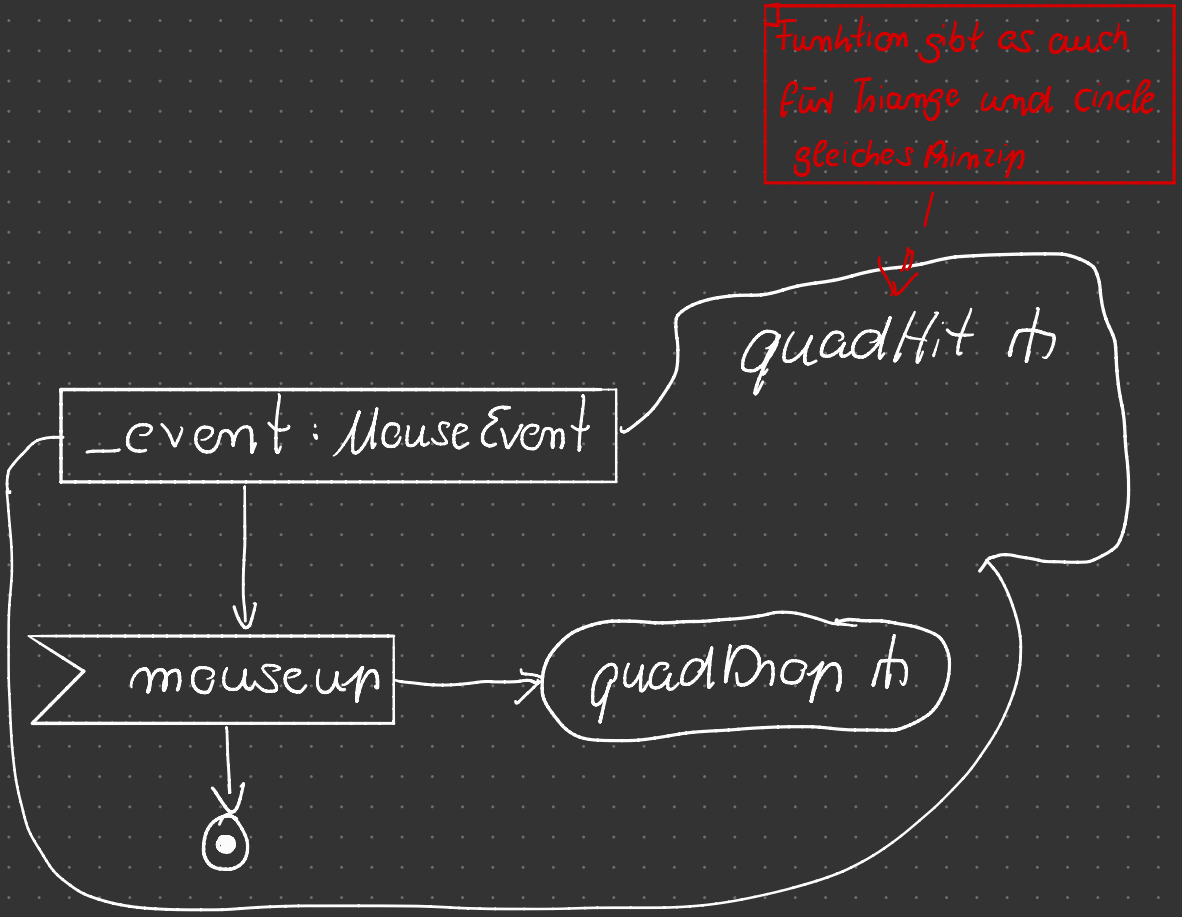
*

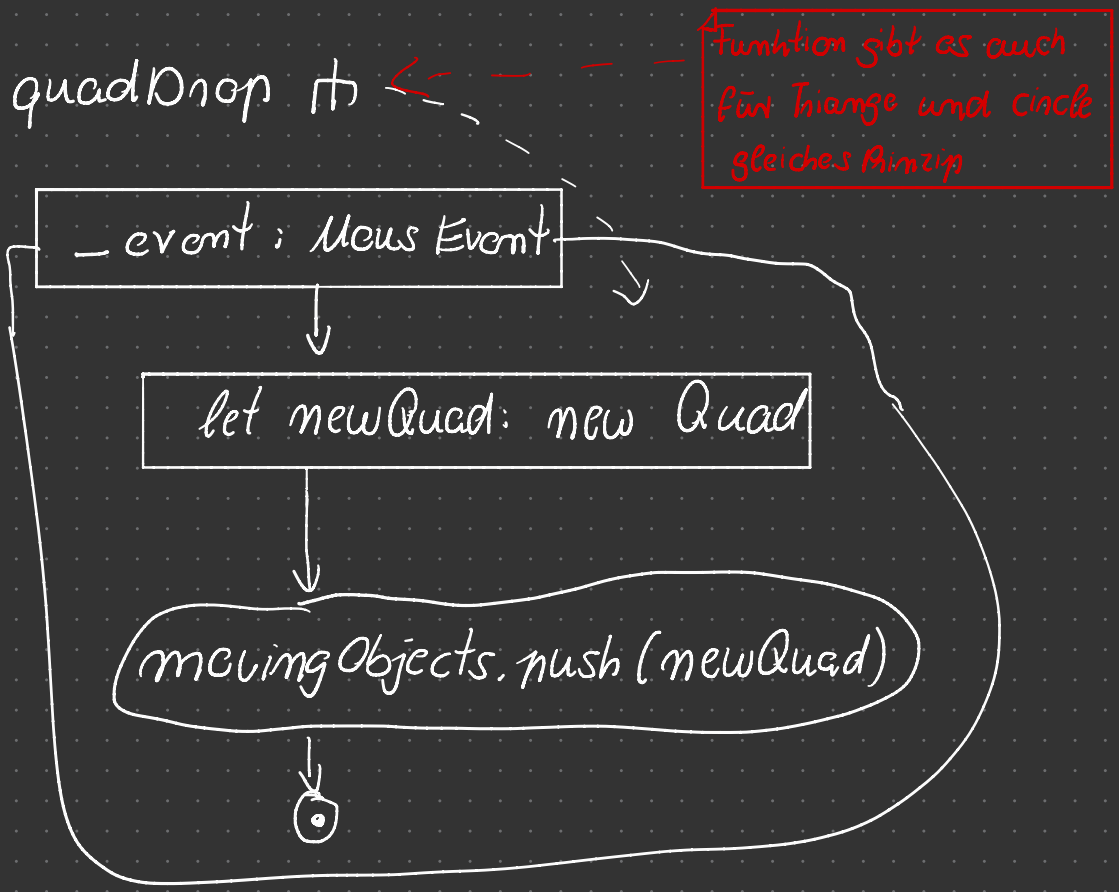

*
Main.ts
register load-listener
load window
init
init
input
handle Change
click Button
start Game

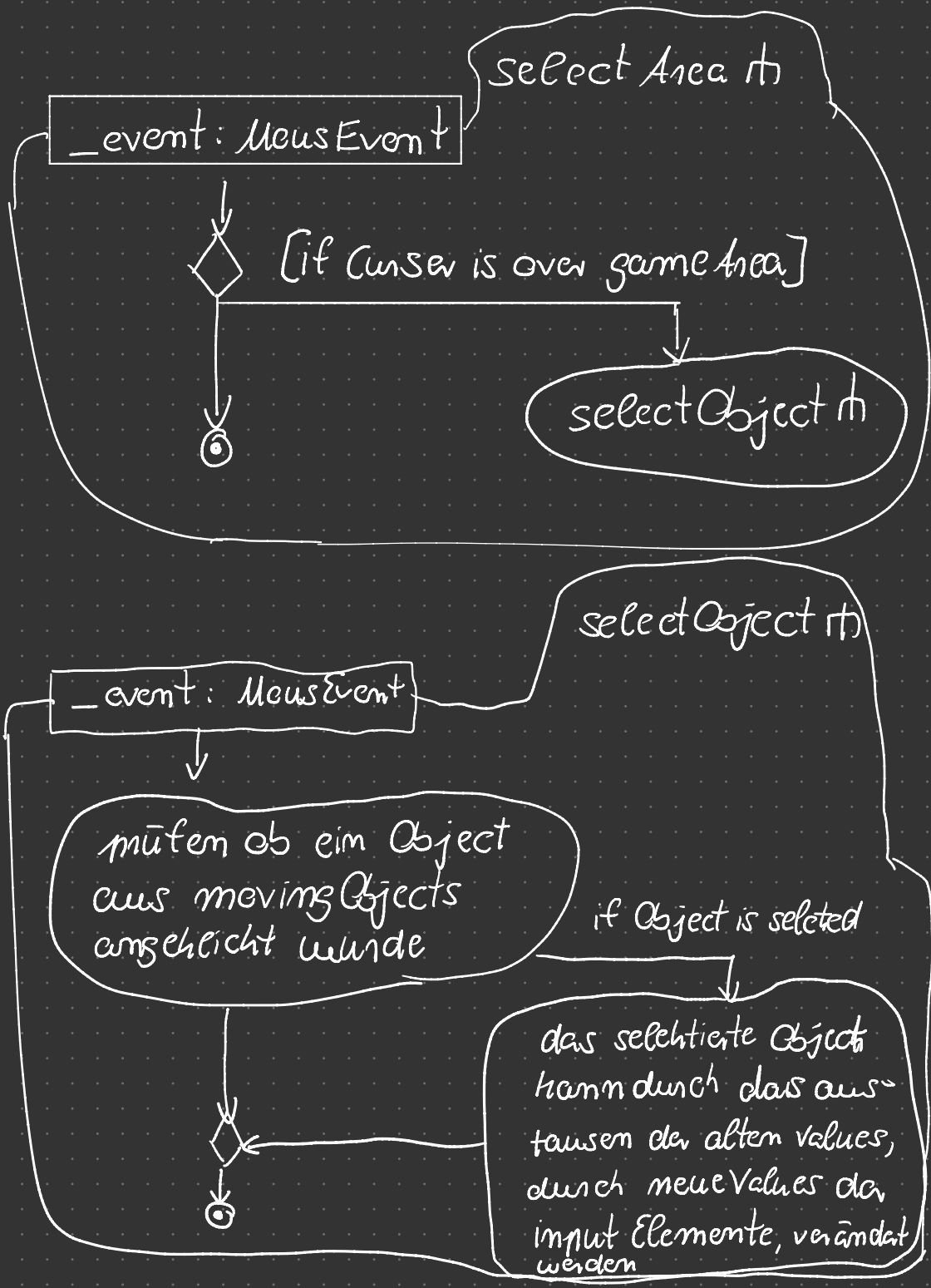

startGame fn
display Game Page
CV Elemente erstellen
[mousedown triggert]
mousdown
mouseDownCheck fn
[if mouseover triggert]
mouseover
selectArea fn
imgData = crc.getImageData
setInterval(uptdate fn , 20)


Update th
clearRect
putImgData
[i < object.lenght]
[i++]
movingObjects[i].draw()
movingObjects[i].move()


_event: MouseEvent
mousDownChech m
if Mousedown on interfaceQuad
mousemove
quadHit m
[if mousedown interfaceCircle]
[if mouseDownOn interfaceTriangle]
mousmove
circleHit m
mousmove
triengleHit m


Funktion gibt es auch für Triangle und circle gleiches Prinzip
quadHit th
_event : MouseEvent
mouseup
quadDrop th


quadDrop
Funktion gibt es auch für Triange und Circle gleiches Prinzip
_event : MouseEvent
let newQuad: new Quad
movingObjects.push(newQuad)


selectArea fkt
_event: MausEvent
[if Cursor is over gameArea]
selectObject fkt
selectObject fkt
_event: MausEvent
prüfen ob ein Object aus movingObjects angeklickt wurde
if Object is seleted
das selektierte Object kann durch das aus-tauschen der alten values, durch neue Values der input Elemente, verändert werden

Server.ts

```
import url from "url"
import Http from "Http"
import Mongo from "mongoDb"
let savedImg: Mongocollection
let Database: string
```

startServer fn

_port

createServer

open_port

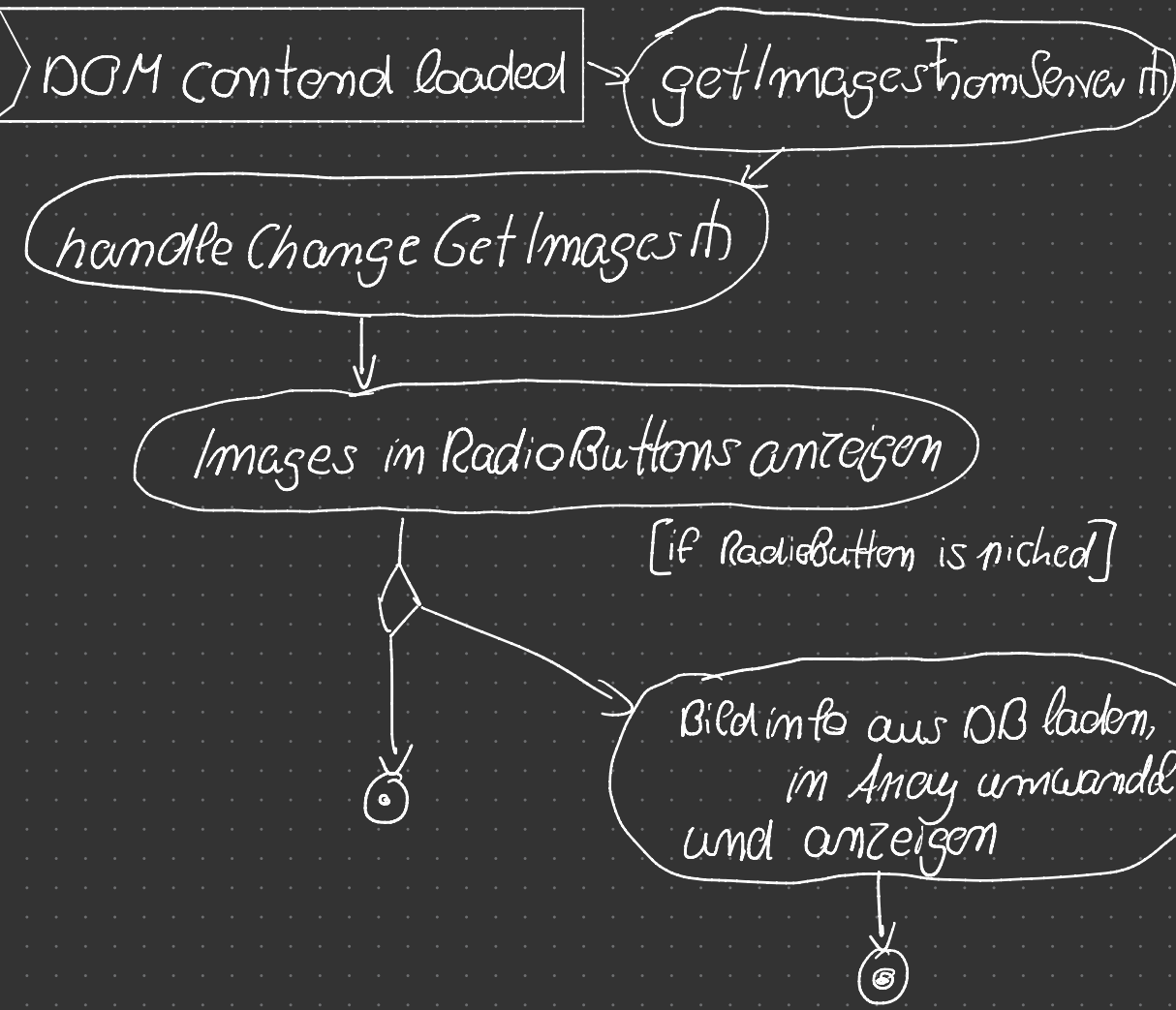
request

handleRequest fn

_request
_response

query = parse_request

split url in Arrays


DOM content loaded
getImagesFromServer()
handleChangeGetImages()
Images in RadioButtons anzeigen
[if RadioButton is niched]
Bildinfo aus DB laden,
in Array umwandeln
und anzeigen