Konzept zur Endabgabe in EiA2 von Felix Brunn

Funktionale Analyse:

Funktionale Analyse

- Das Spiel ist eine Art Bullet-Hell-Spiel bei dem man sich mit dem Raumschiff durch viel Gegner schießen muss. Das Ziel ist es möglichst lange zu überleben und so einen hohen Highscore zu erzielen.
- Als Plattform ist der PC gewählt, da so mehr Übersicht gewährleistet ist und der Highscore immer einsehbar ist.
- Interaktion der Objekte im Spiel:
  - → wenn ein Schuss auf einen gegnerischen Raumschiff trifft, explodiert der Gegner und der Schuss und der Gegner verschwindet
  - → trifft der Spieler auf einen Gegner explodiert der Gegner und der Spieler verliert ein Leben
  - → die Wolken interagieren mit nichts und dienen nur der Optik
  - → die Herzen interagieren nur mit dem Spieler, sammelt der Spieler ein Herz ein verschwindet das Herz und der Spieler bekommt ein Leben
  - →

BRUNNEN

- Aufbau und Ablauf aus Sicht des Nutzers

→ nachdem die Website geöffnet wurde erscheint die Spielanleitung. Nach Bestätigung startet das Spiel.

→ Der Spieler kann sein Raumschiff mit "w,a,s,d" steuern und mit "Leertaste" schießen

→ an der Rechten Seite kann der Spieler die Top 10 Highscores ansehen

→ Links oben im Canvas sieht der Spieler seine Leben und rechts oben seinen Score

→ Der Spieler kann den Canvas mit seinem Raumschiff nicht verlassen

→ berührt das Spielerraumschiff ein Herz bekommt er ein Extraleben und das Herz spawnt am rechten Rand erneut (außer der Spieler hat zum Zeitpunkt des Einsammelns bereits 4 Leben)

→ berührt das Spielerraumschiff einen Gegner verliert er ein Leben, 50 Punkte und der Gegner wird zerstört.

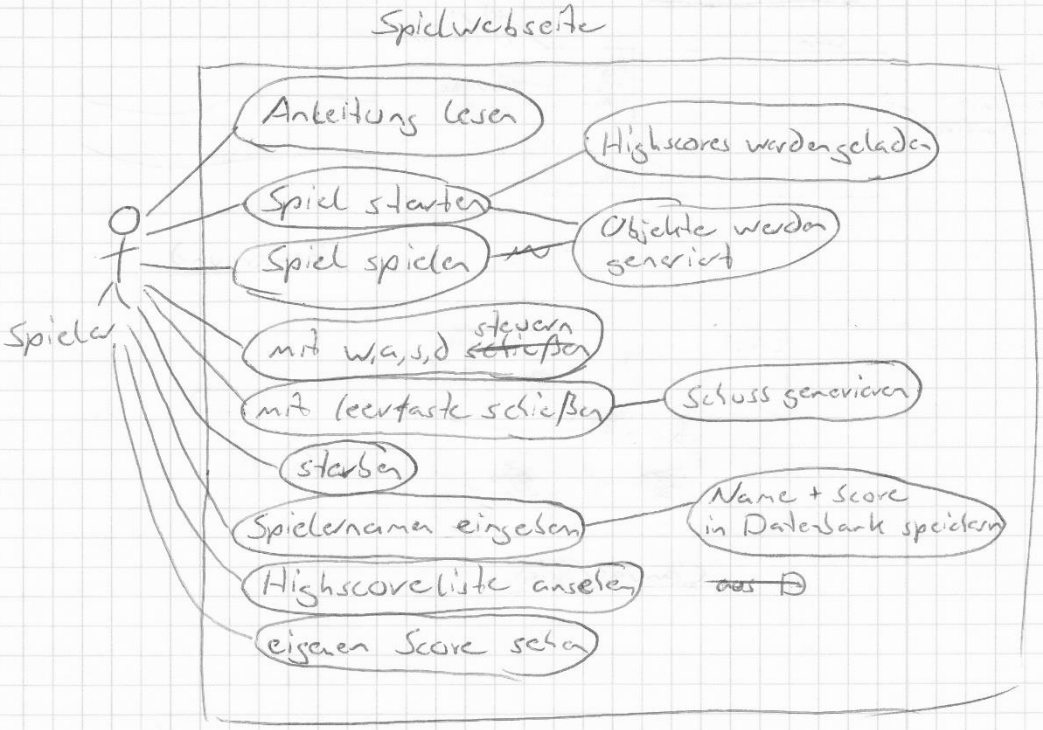
→ trifft ein Schuss des Spielers einen Gegner wird der Gegner zerstört und der Spieler erhält 50 Punkte

→ das Spielerraumschiff wächst mit jedem eingesammelten Leben und schrumpft mit jedem Leben das der Spieler verliert

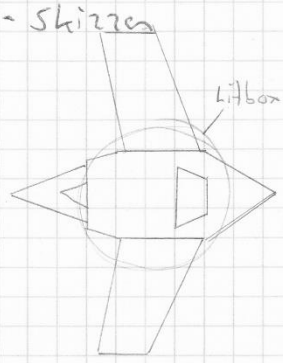
→ fallen die Leben des Spielers auf 0 hat der Spieler verloren, der Spieler kann nun seinen Spielernamen eingeben

→ ist der Score hoch genug wird er nach Bestätigung im Scoreboard angezeigt

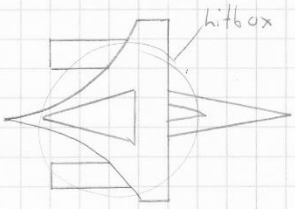
- Nutzerinteraktionen



- Skizzen

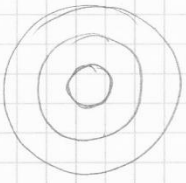


Spieler

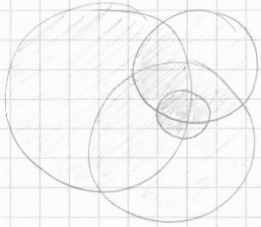


Gegner

Herz

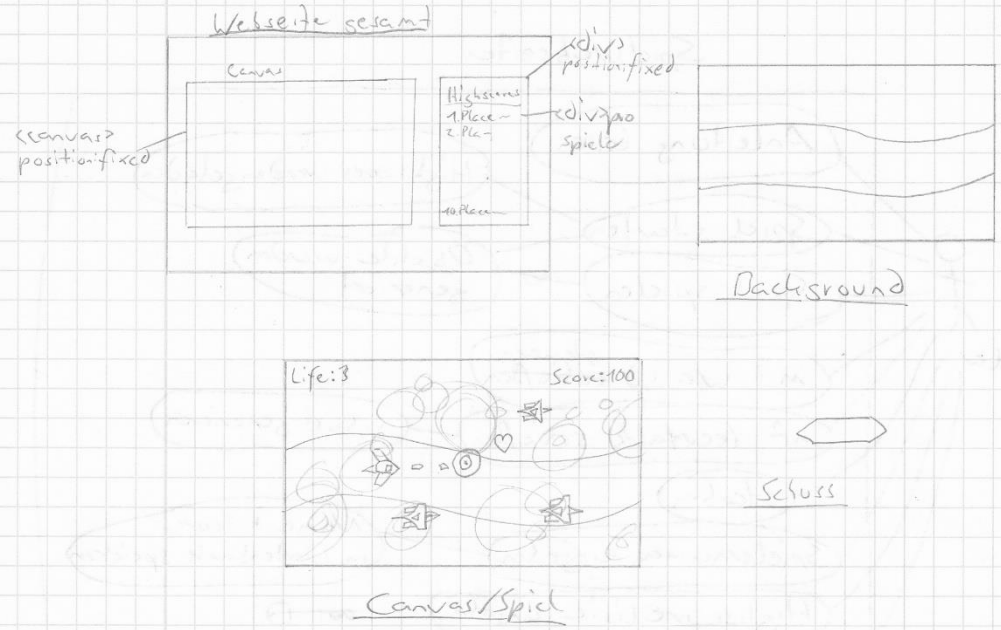


Explosion



Wolken

- Skizzen 2

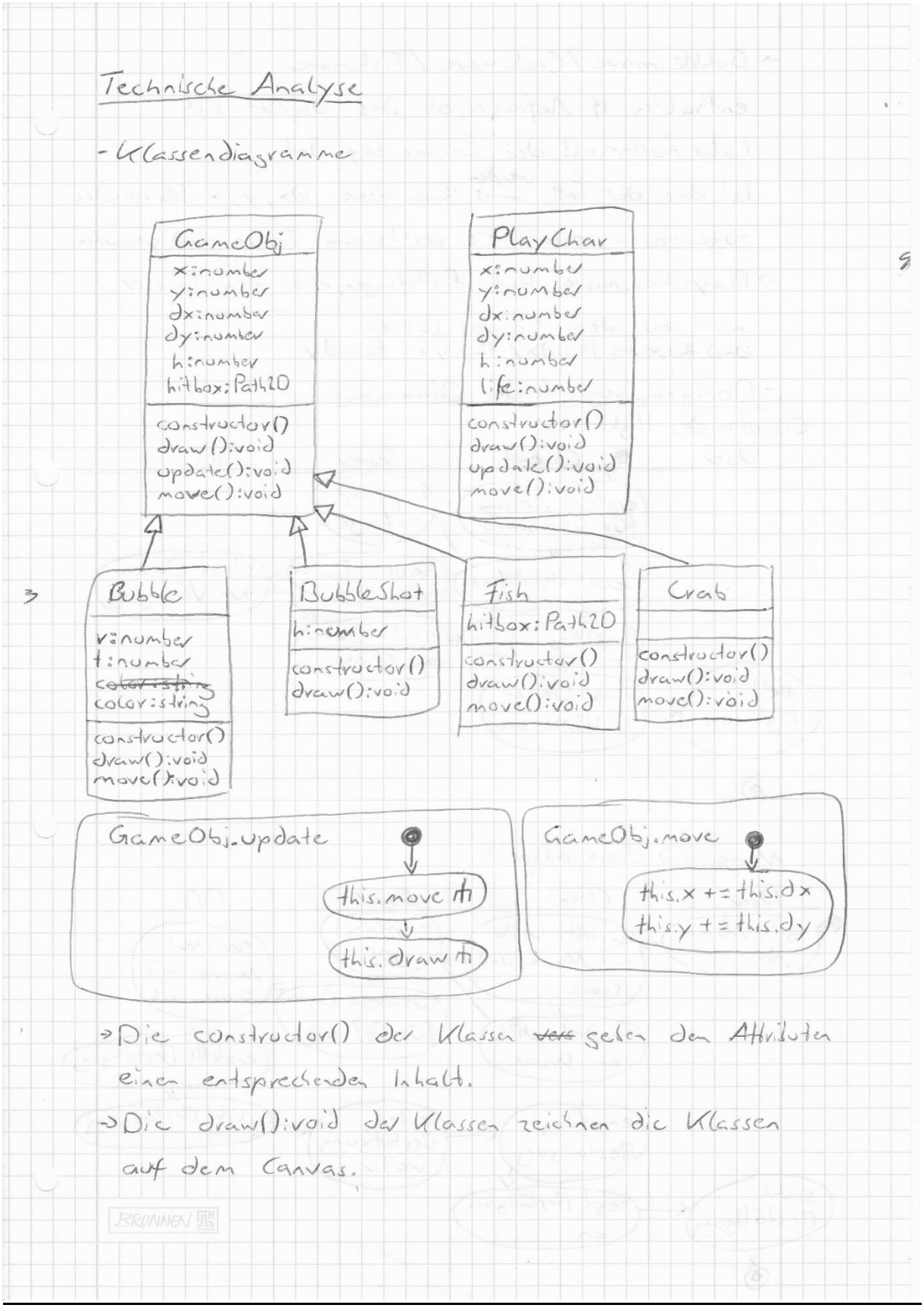


Background

Canvas/Spiel

Schuss

Technische Analyse:



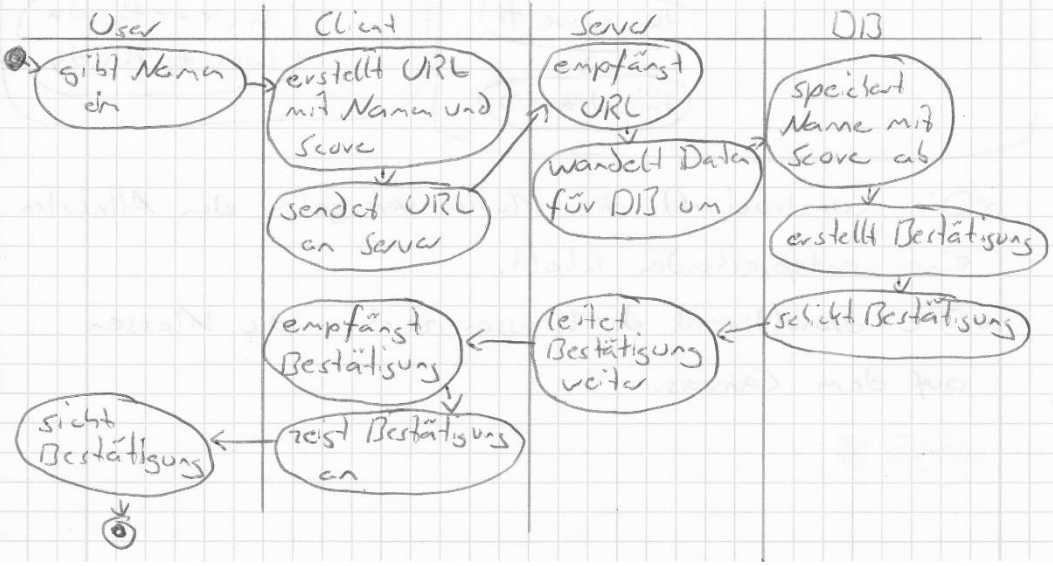
→ Bubble.move / Crab.move / Fish.move
enthalten if-Abfragen, ob das Objekt sich links außerhalb des Canvas befindet.
Ist das der Fall, ~~wird~~ werden ihnen neue dx, y, x-Koordinaten zugewiesen, sodass sie rechts am Canvasrand spawnen.

→ PlayChar.move hat if-Abfragen, die den Spieler nicht aus dem Canvas lassen.
und this.x += this.dx / this.y += this.dy

- Domänenübergreifendes Diagramm

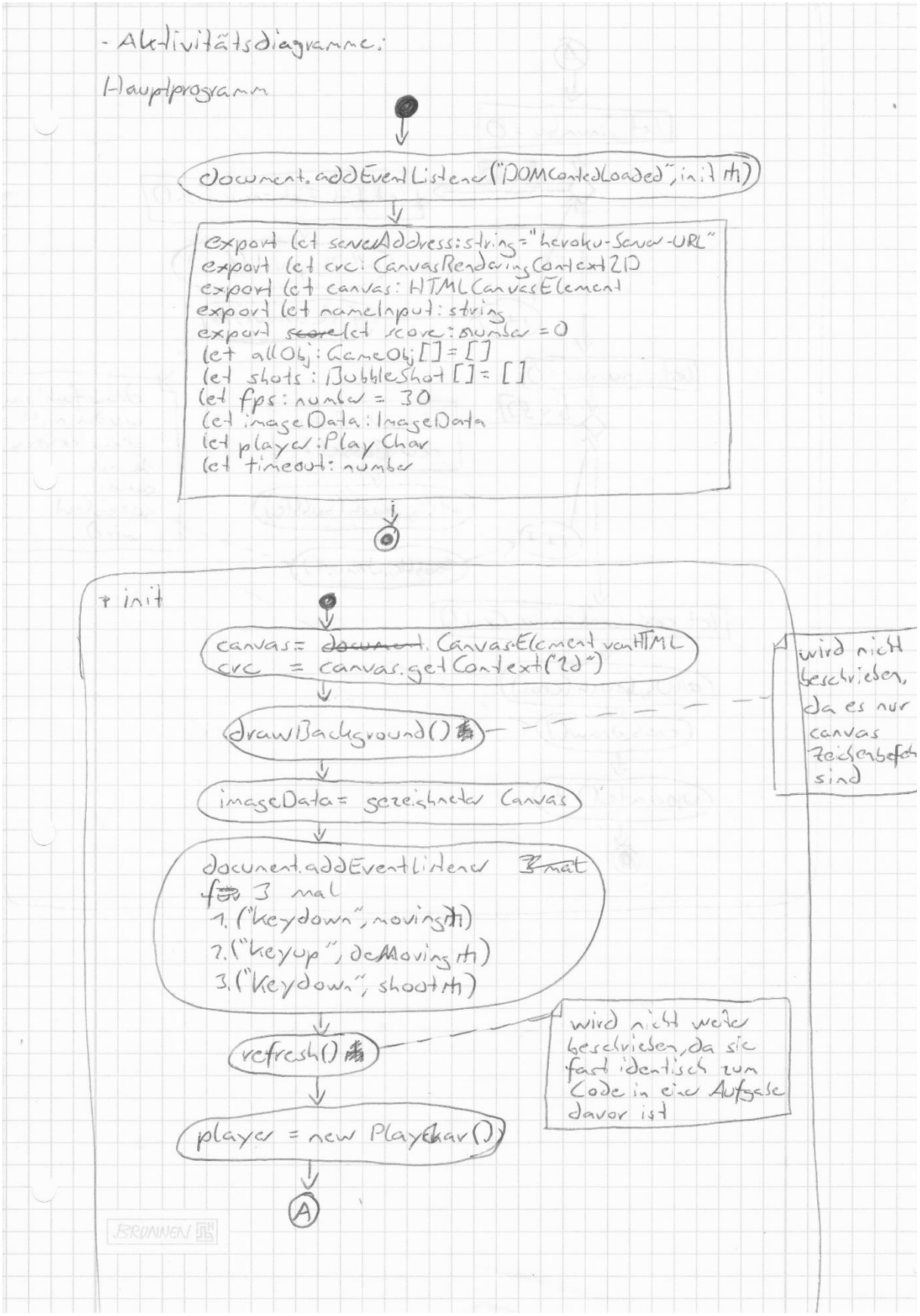
Erstellen des Highscores:

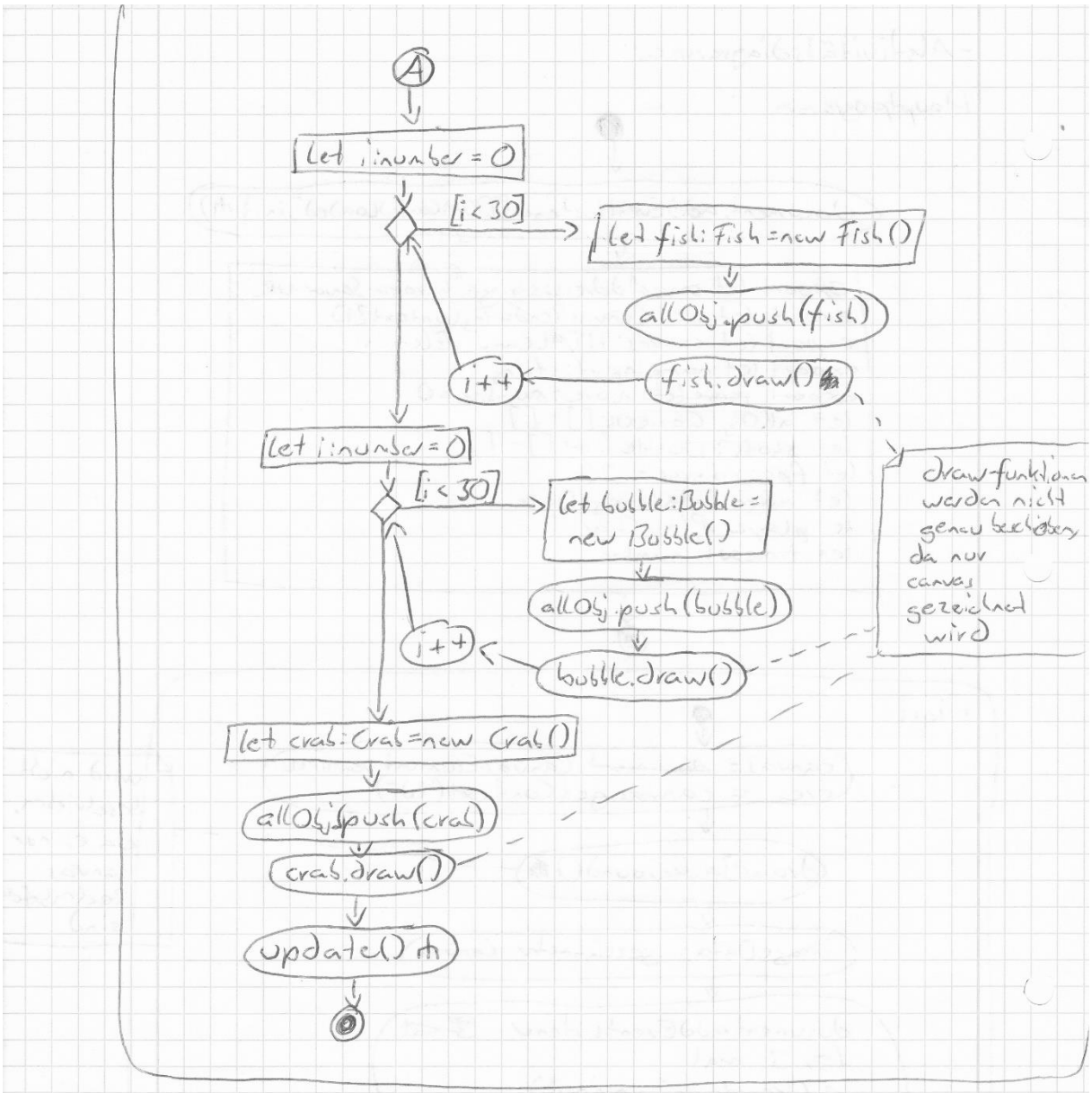
Name und Score abspeichern:

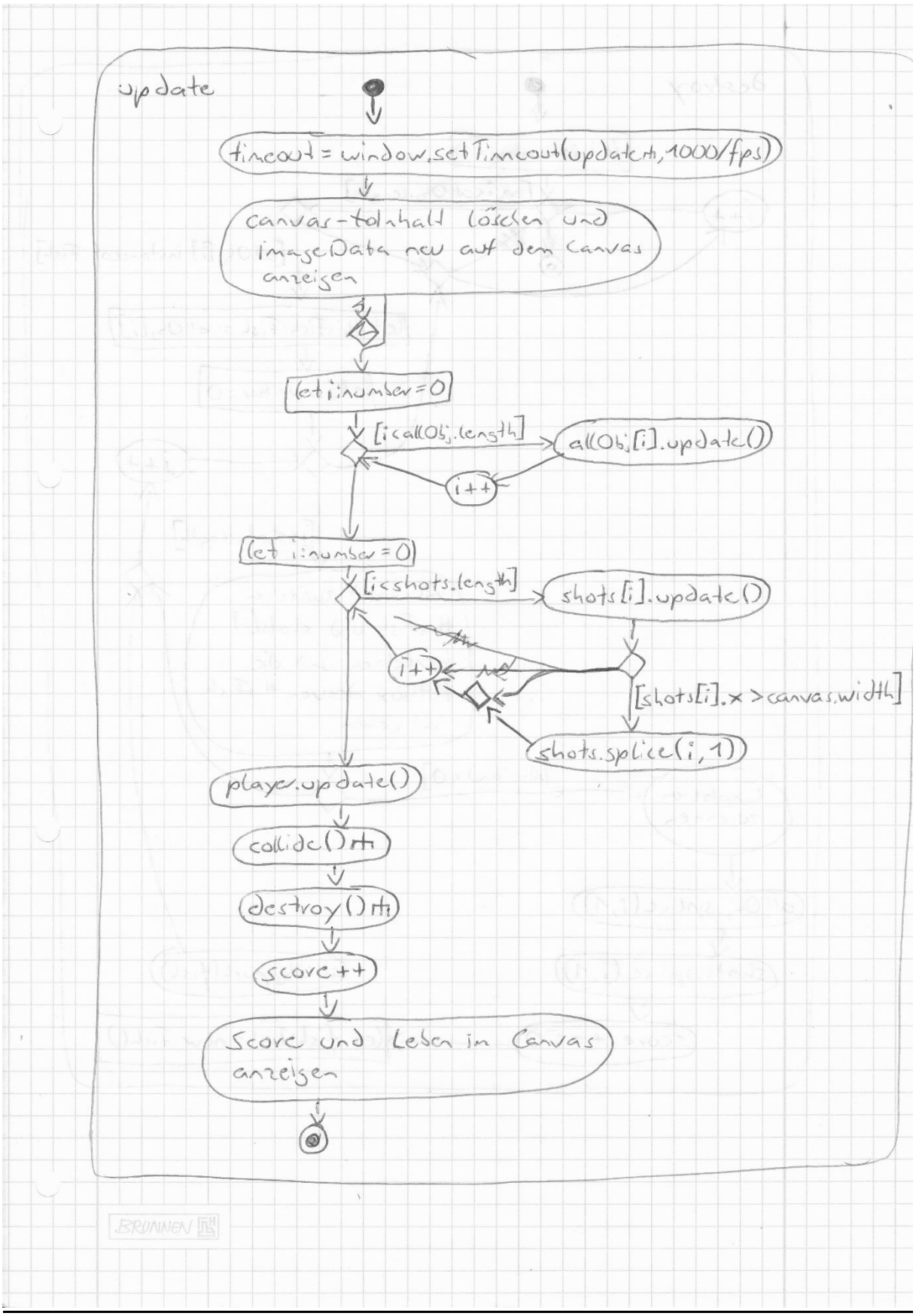


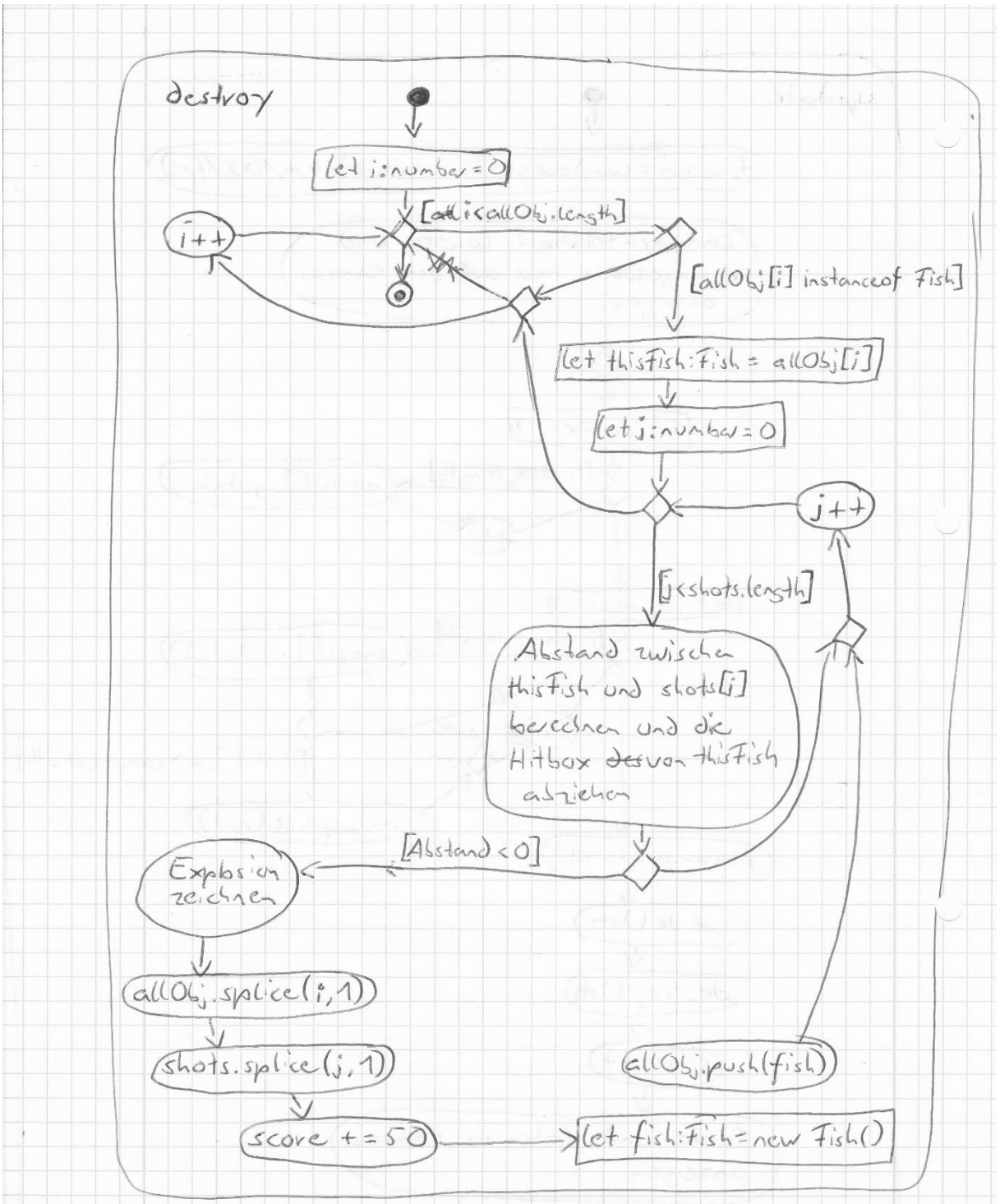
- Aktivitätsdiagramme:

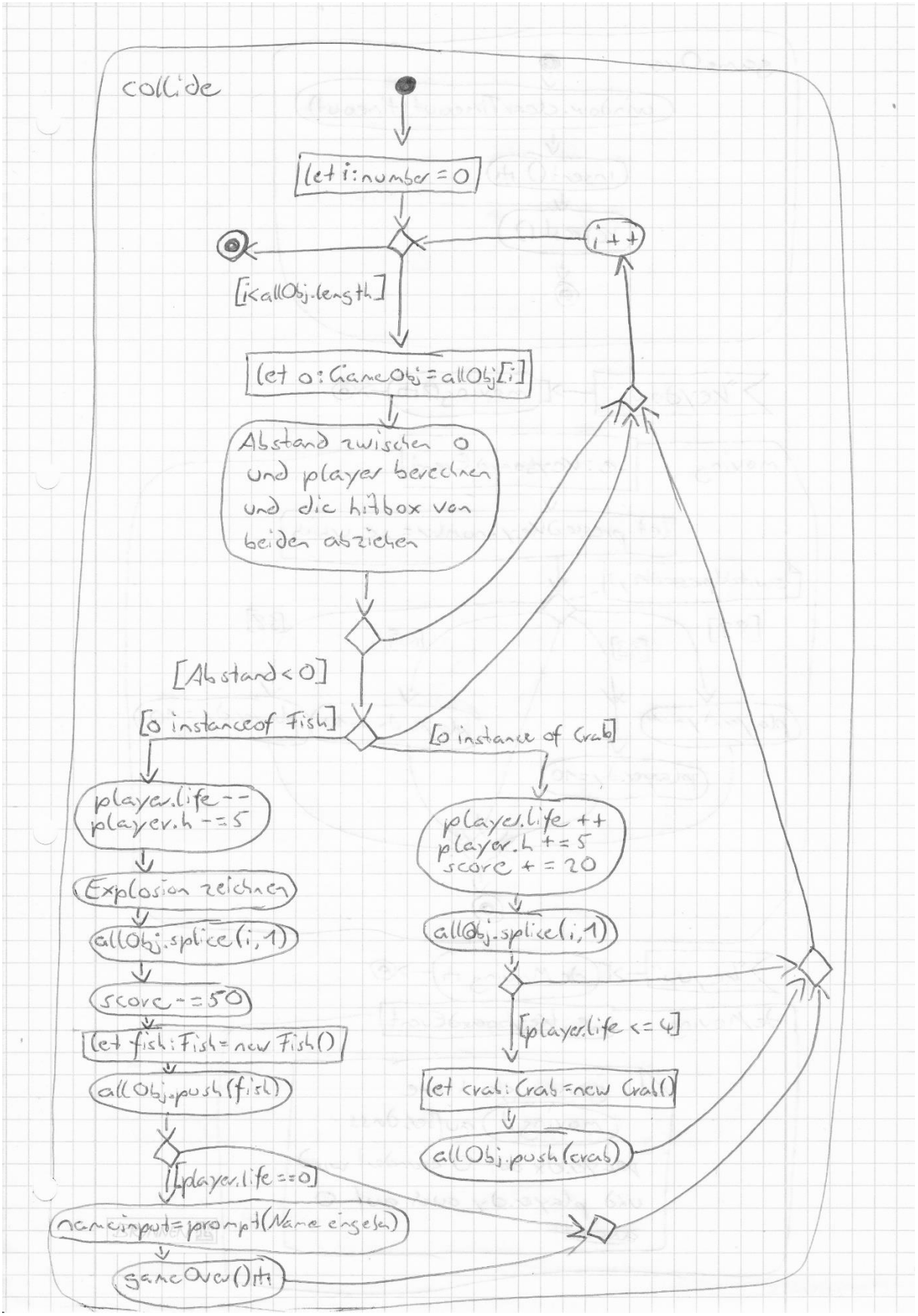
Hauptprogramm

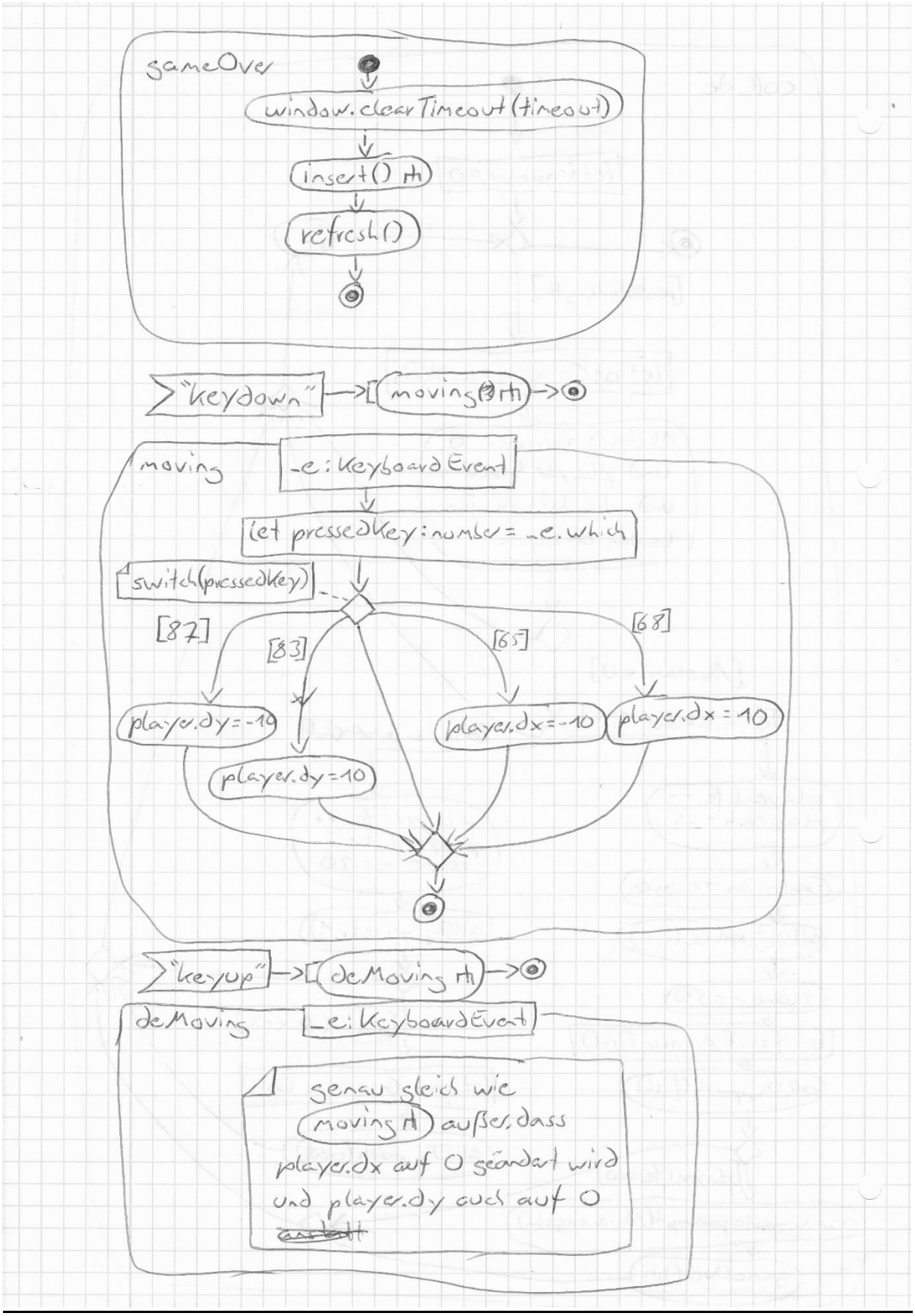


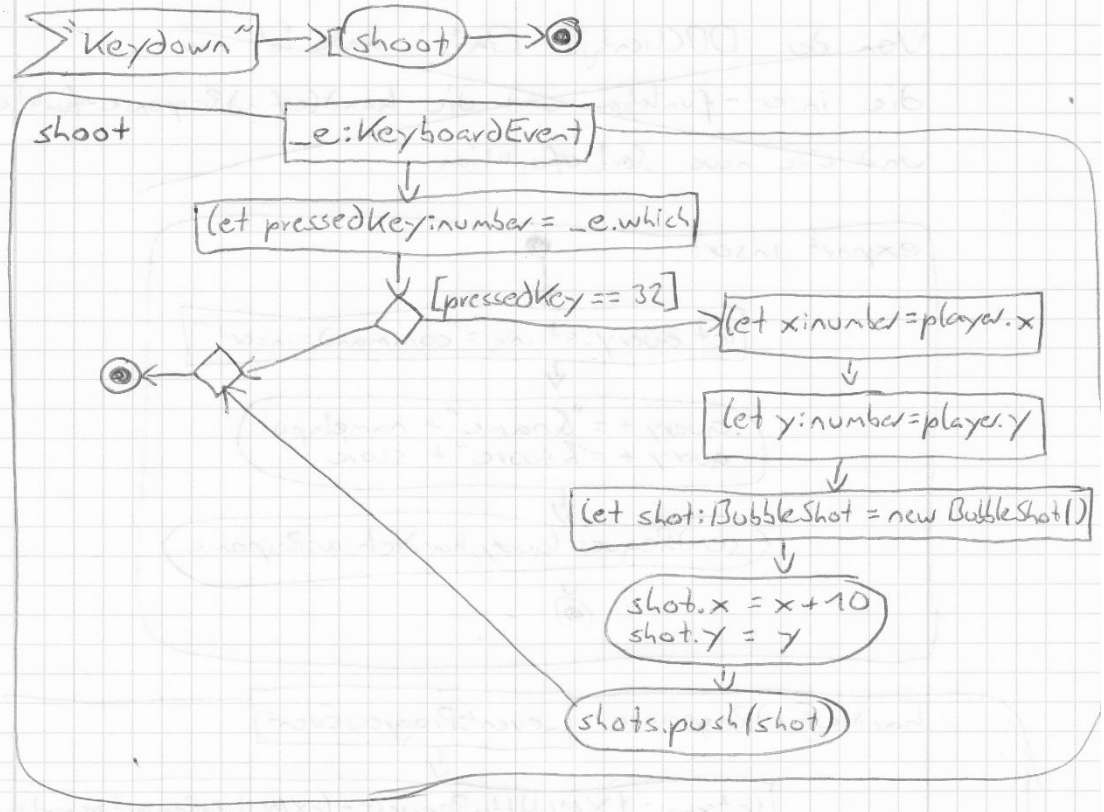

A
let i:number = 0
[i < 30]
let fish: Fish = new Fish()
allObj.push(fish)
fish.draw()
i++
let i:number = 0
[i < 30]
let bubble:Bubble = new Bubble()
allObj.push(bubble)
bubble.draw()
i++
draw-funktionen werden nicht genau beschrieben, da nur canvas gezeichnet wird
let crab: Crab = new Crab()
allObj.push(crab)
crab.draw()
update() th


update
timeout = window.setTimeout(update, 1000/fps)
canvas-Inhalt löschen und imageData neu auf dem Canvas anzeigen
let i:number=0
[i<allObj.length]
allObj[i].update()
i++
let i:number=0
[i<shots.length]
shots[i].update()
i++
[shots[i].x > canvas.width]
shots.splice(i, 1)
player.update()
collide()
destroy()
score++
Score und Leben im Canvas anzeigen


destroy
let i:number=0
[i<allObj.length]
i++
[allObj[i] instanceof Fish]
let thisFish:Fish = allObj[i]
let j:number=0
j++
[j<shots.length]
Abstand zwischen thisFish und shots[j] berechnen und die Hitbox von thisFish abziehen
[Abstand<0]
Explosion zeichnen
allObj.splice(i,1)
shots.splice(j,1)
score +=50
allObj.push(fish)
let fish:Fish=new Fish()

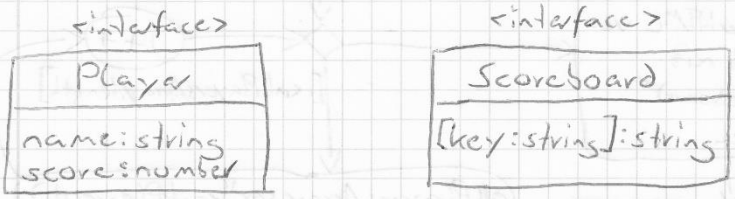

collide
let i:number=0
i++
[i<allObj.length]
let o:GameObj=allObj[i]
Abstand zwischen o und player berechnen und die hitbox von beiden abziehen
[Abstand<0]
[o instanceof Fish]
[o instance of Crab]
player.life--
player.h-=5
Explosion zeichnen
allObj.splice(i,1)
score-=50
let fish:Fish=new Fish()
allObj.push(fish)
[player.life==0]
nameinput=prompt(Name eingeben)
gameOver()
player.life++
player.h+=5
score+=20
allObj.splice(i,1)
[player.life<=4]
let crab:Crab=new Crab()
allObj.push(crab)


gameOver
window.clearTimeout(timeout)
insert() H
refresh()
"keydown" -> [moving H] ->
moving
_e: KeyboardEvent
let pressedKey: number = _e.which
switch(pressedKey)
[87]
[83]
[65]
[68]
player.dy = -19
player.dy = 10
player.dx = -10
player.dx = 10
"keyup" -> [deMoving H] ->
deMoving
_e: KeyboardEvent
genau gleich wie
moving H außer, dass
player.dx auf 0 geändert wird
und player.dy auch auf 0

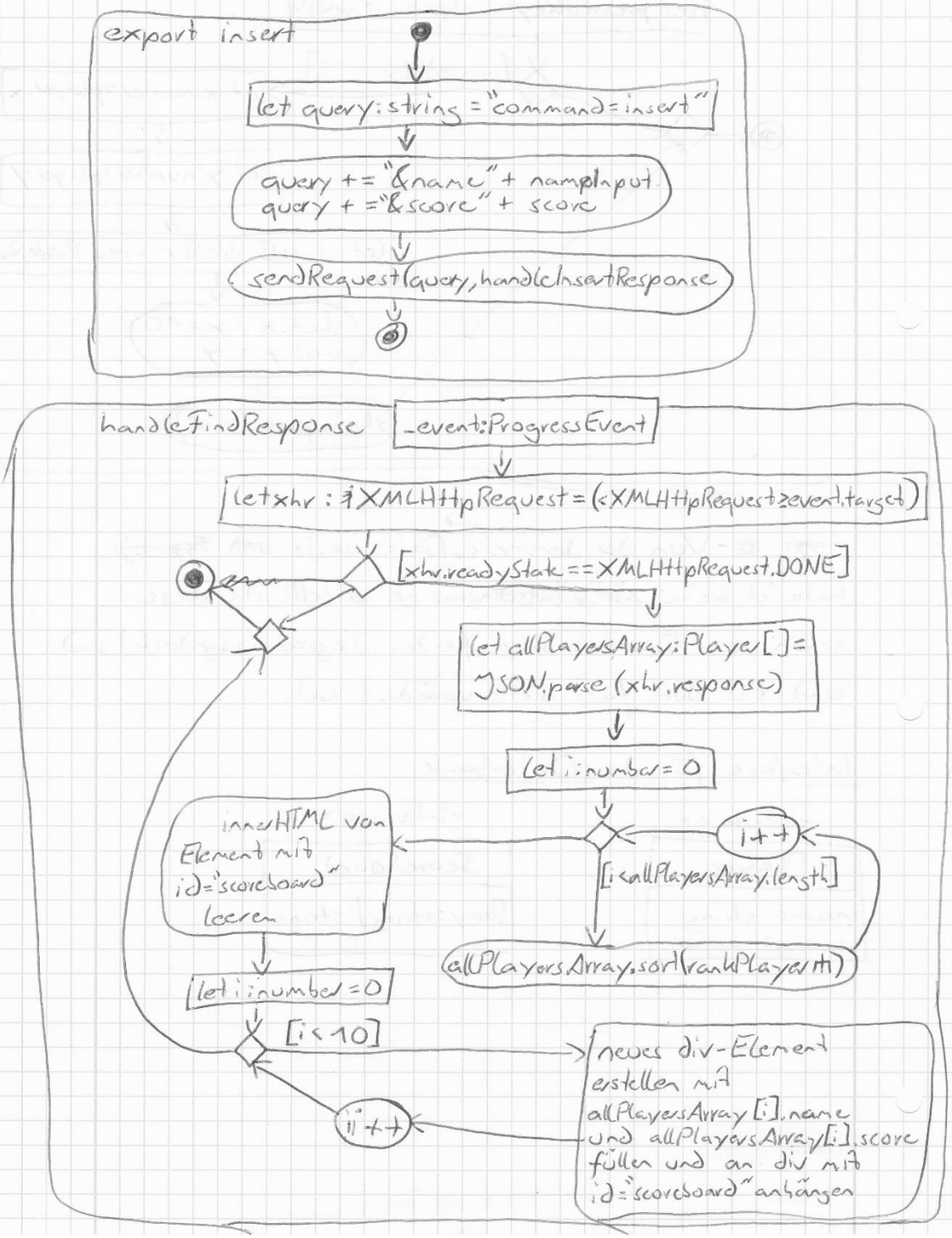


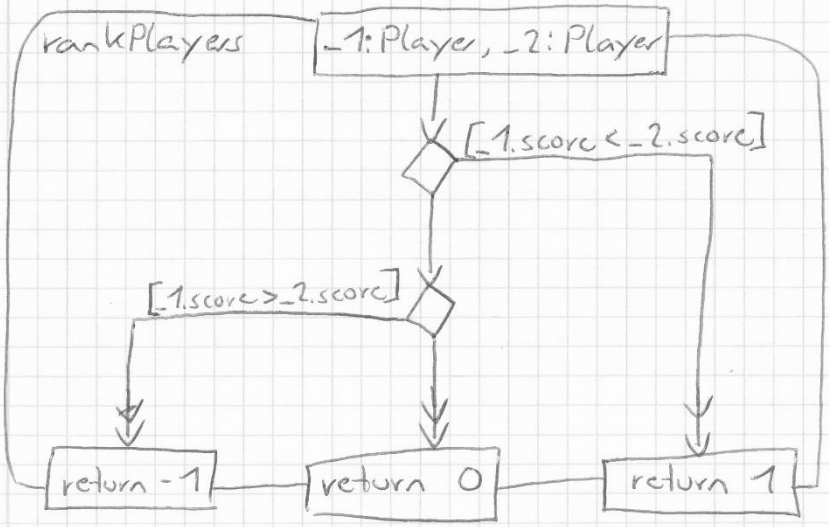
~~types.ts~~ - Von der server.js und Database.js ~~und~~ ~~Types.js~~ habe ich keine Aktivitätsdiagramme erstellt, da diese schon vom Domänenübergreifenden Diagram abgedeckt sind und sich dort fast nichts verändert hat.

Interface für die Datenbank



Von der DBClient.js-Datei habe ich nur die insert-funktion und die handleFindResponse-funktion und eine neue Sortierfunktion




rankPlayers
_1: Player, _2: Player
[_1.score < _2.score]
[_1.score > _2.score]
return -1
return 0
return 1