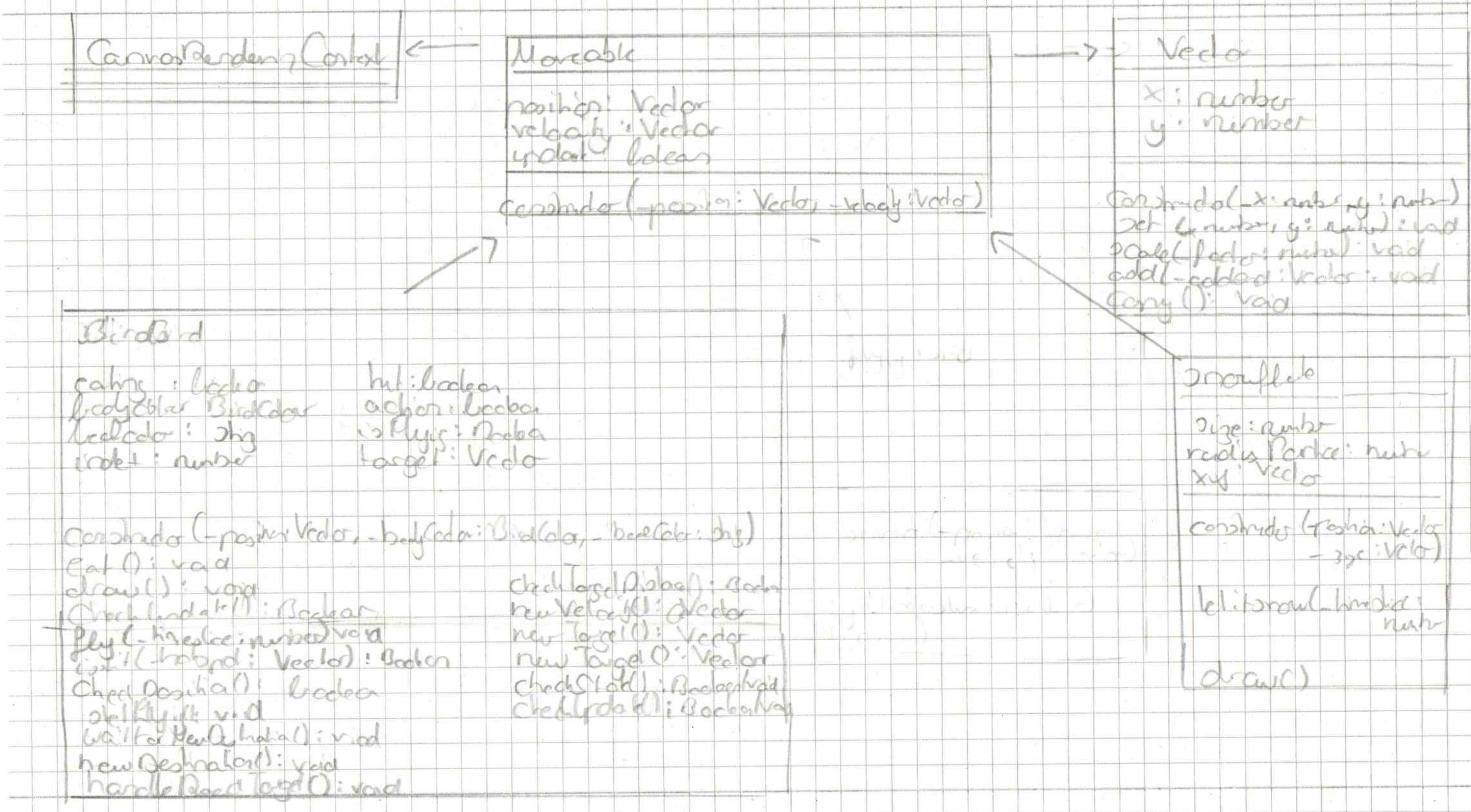
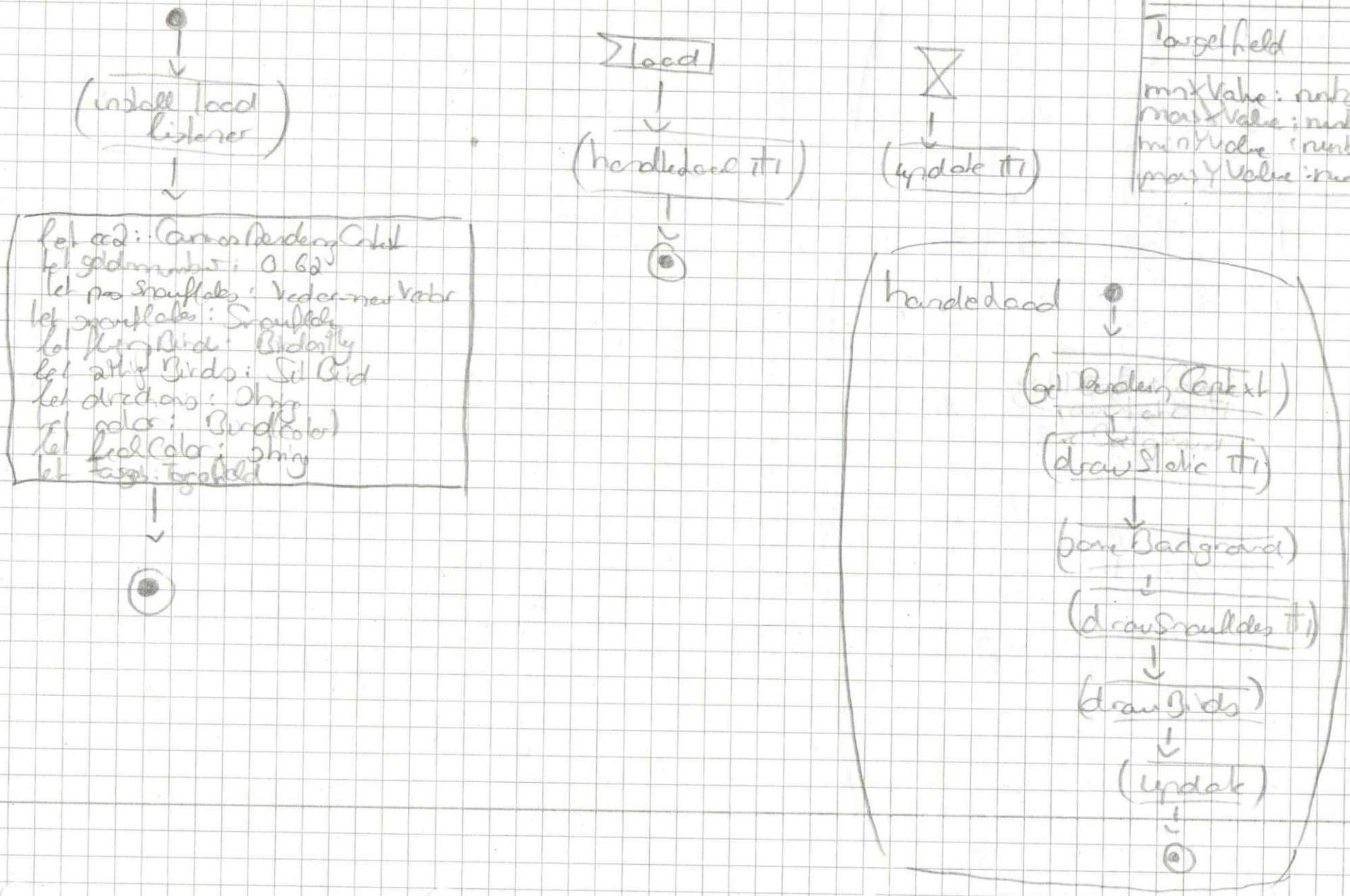
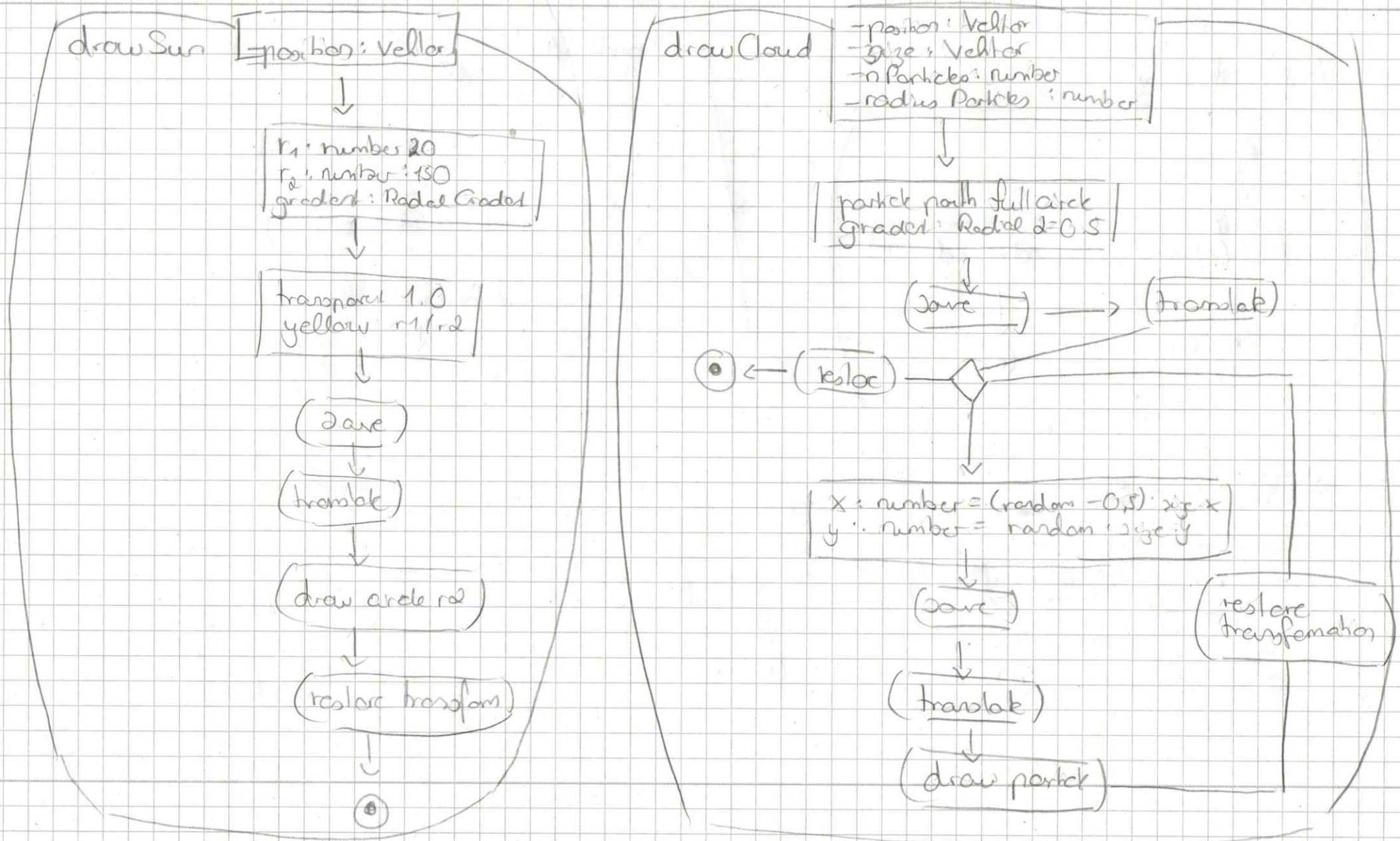
1

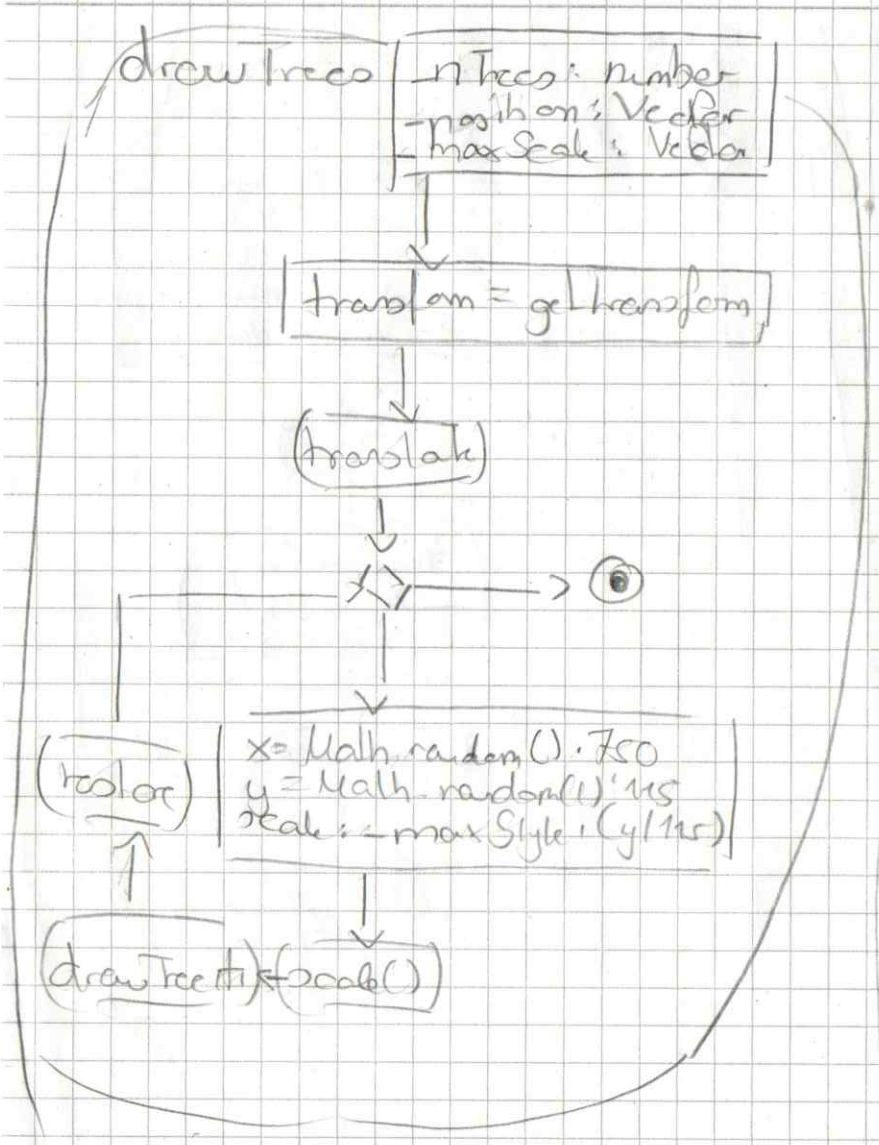

CanvasRenderingContext
Moveable
position: Vector
velocity: Vector
updated: boolean
constructor(_position: Vector, _velocity: Vector)
Vector
x: number
y: number
constructor(_x: number, _y: number)
set(x: number, y: number): void
scale(_factor: number): void
add(_added: Vector): void
copy(): void
Birdbird
eating: boolean
bodyColor: BirdColor
beakColor: string
index: number
hit: boolean
action: boolean
isFlying: boolean
target: Vector
constructor(_position: Vector, _bodyColor: BirdColor, _beakColor: string)
eat(): void
draw(): void
fly(_timeslice: number): void
checkPosition(): boolean
waitForNewDestination(): void
newDestination(): void
handleReachTarget(): void
checkTargetReached(): boolean
newVelocity(): Vector
newTarget(): Vector
newTarget(): Vector
Snowflake
size: number
radiusParticle: number
xy: Vector
constructor(_position: Vector, _size: Vector)
draw()

2

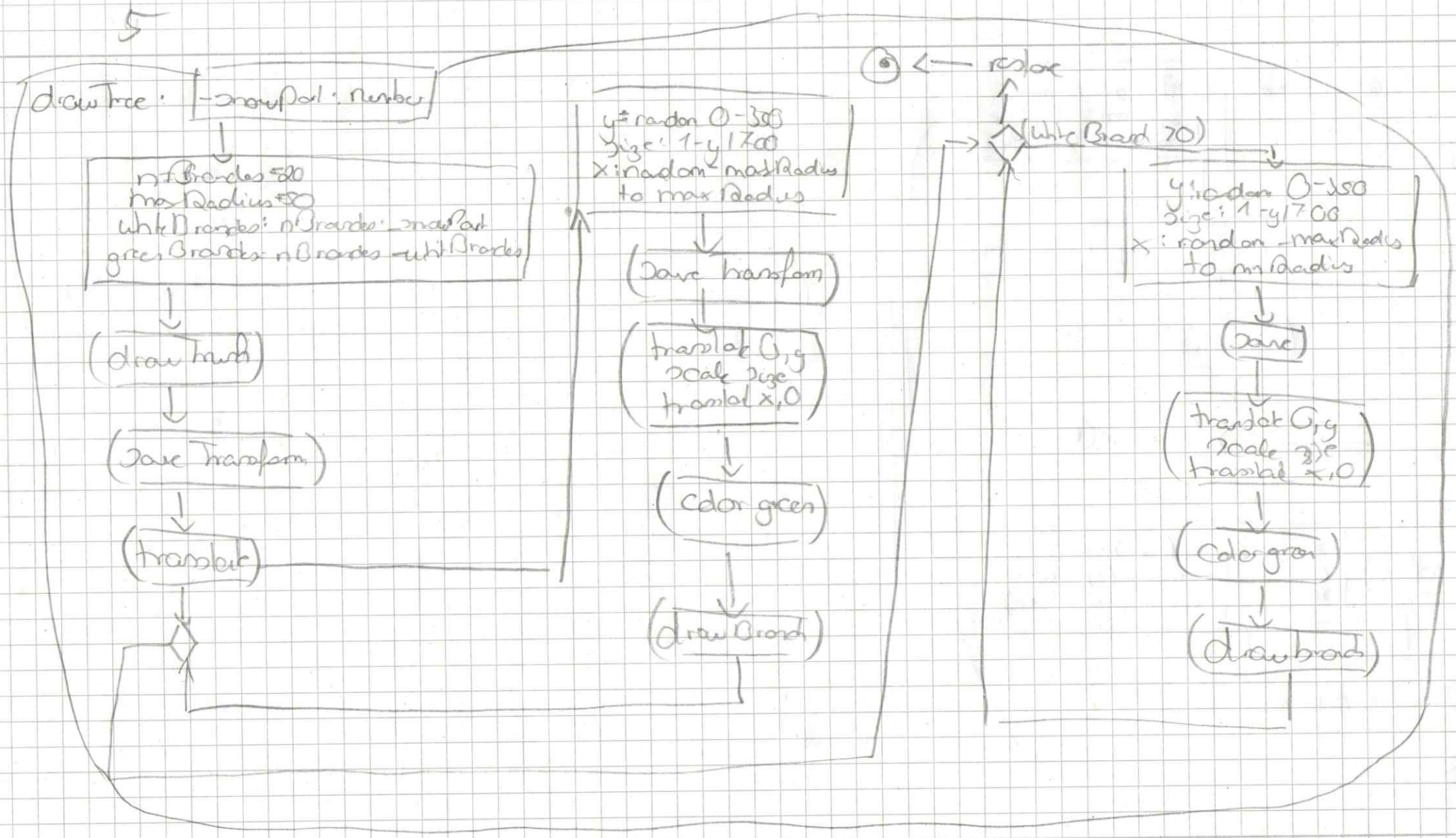
Aktivitätsdiagramm

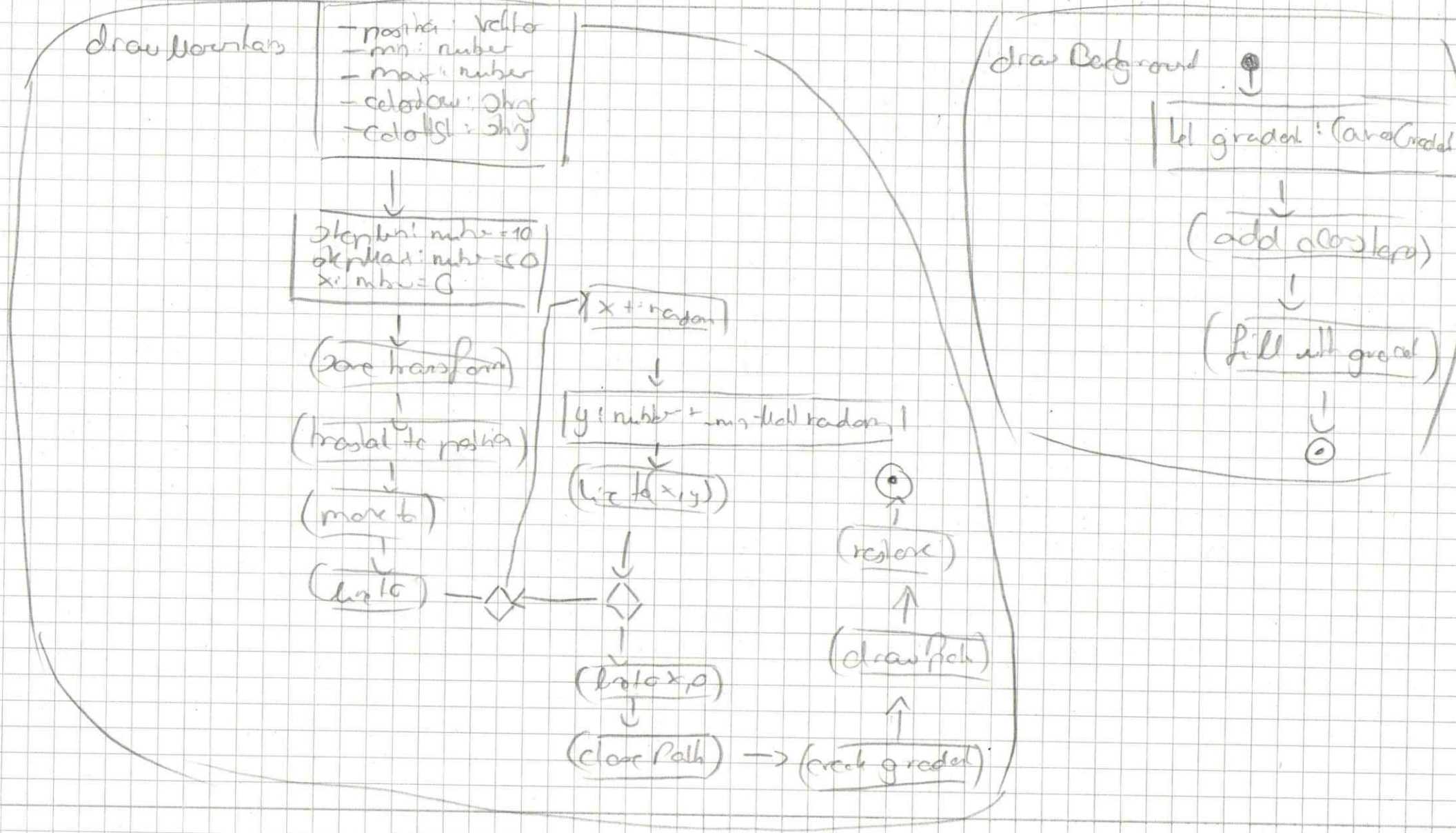


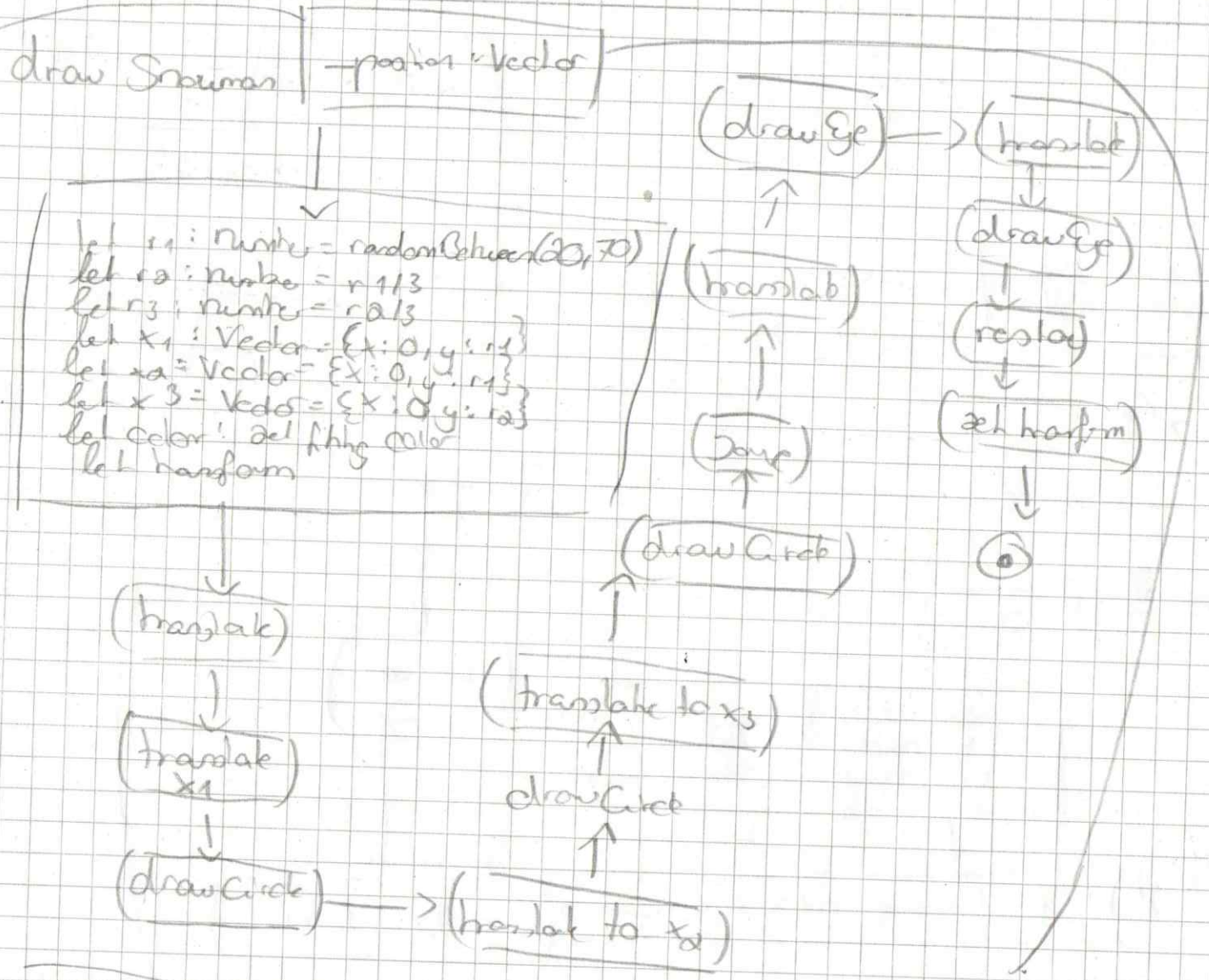

drawSun
position : Vektor
r1 : number 20
r2 : number 150
gradient : Radial Gradient
transparent 1.0
yellow r1/r2
save
translate
draw circle r2
restore transform
drawCloud
-position : Vektor
-size : Vektor
-nParticles : number
-radius Particles : number
particle path full circle
gradient : Radial d=0.5
save
translate
restore
x : number = (random -0,5) * size.x
y : number = random * size.y
save
translate
draw particle
restore transformation

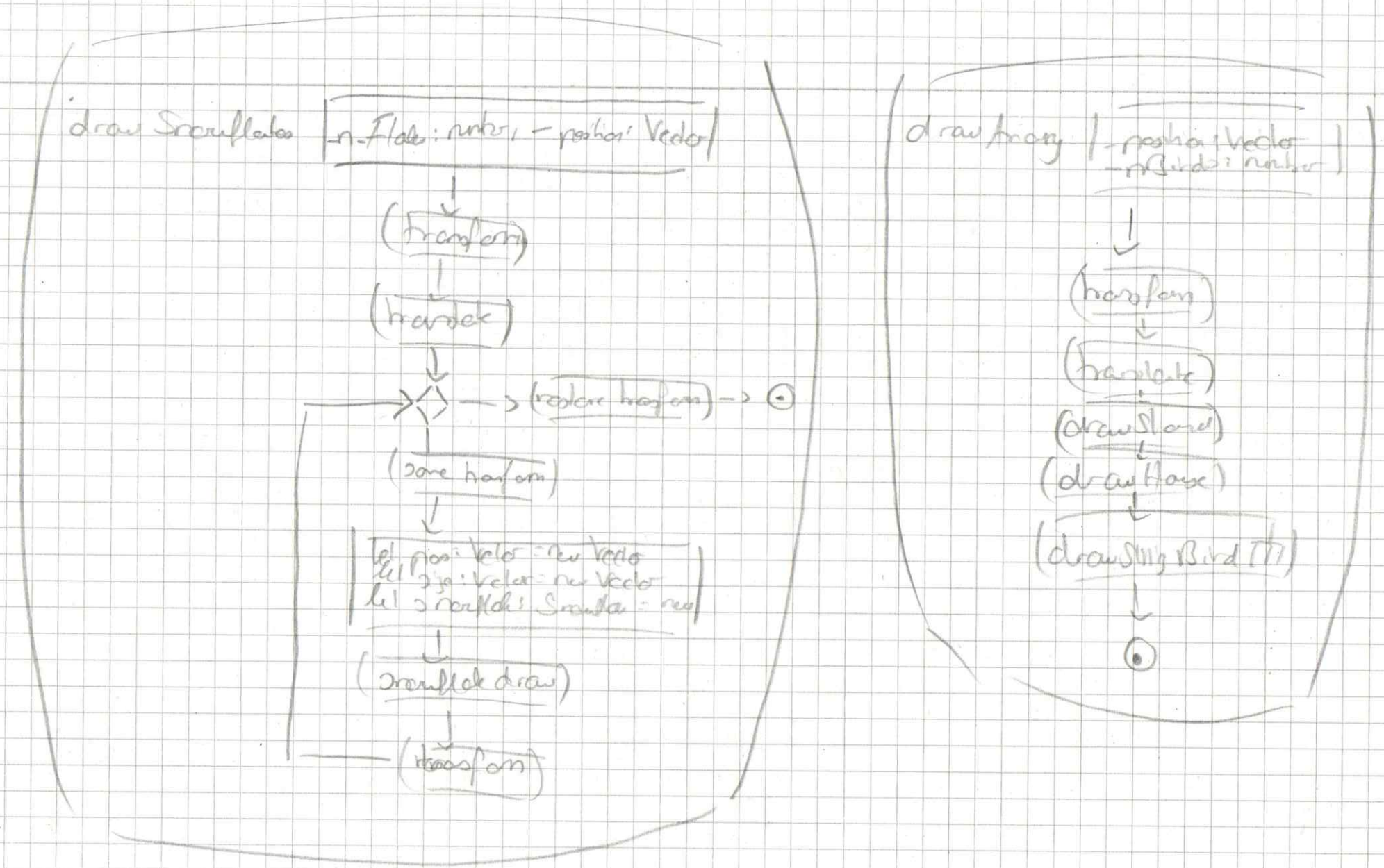

drawTrees
nTrees : number
position : Vector
maxScale : Vector
transform = getTransform
translate
x = Math.random() * 750
y = Math.random() * 145
scale = maxScale * (y / 145)
color
drawTree(n-1, scale())

5

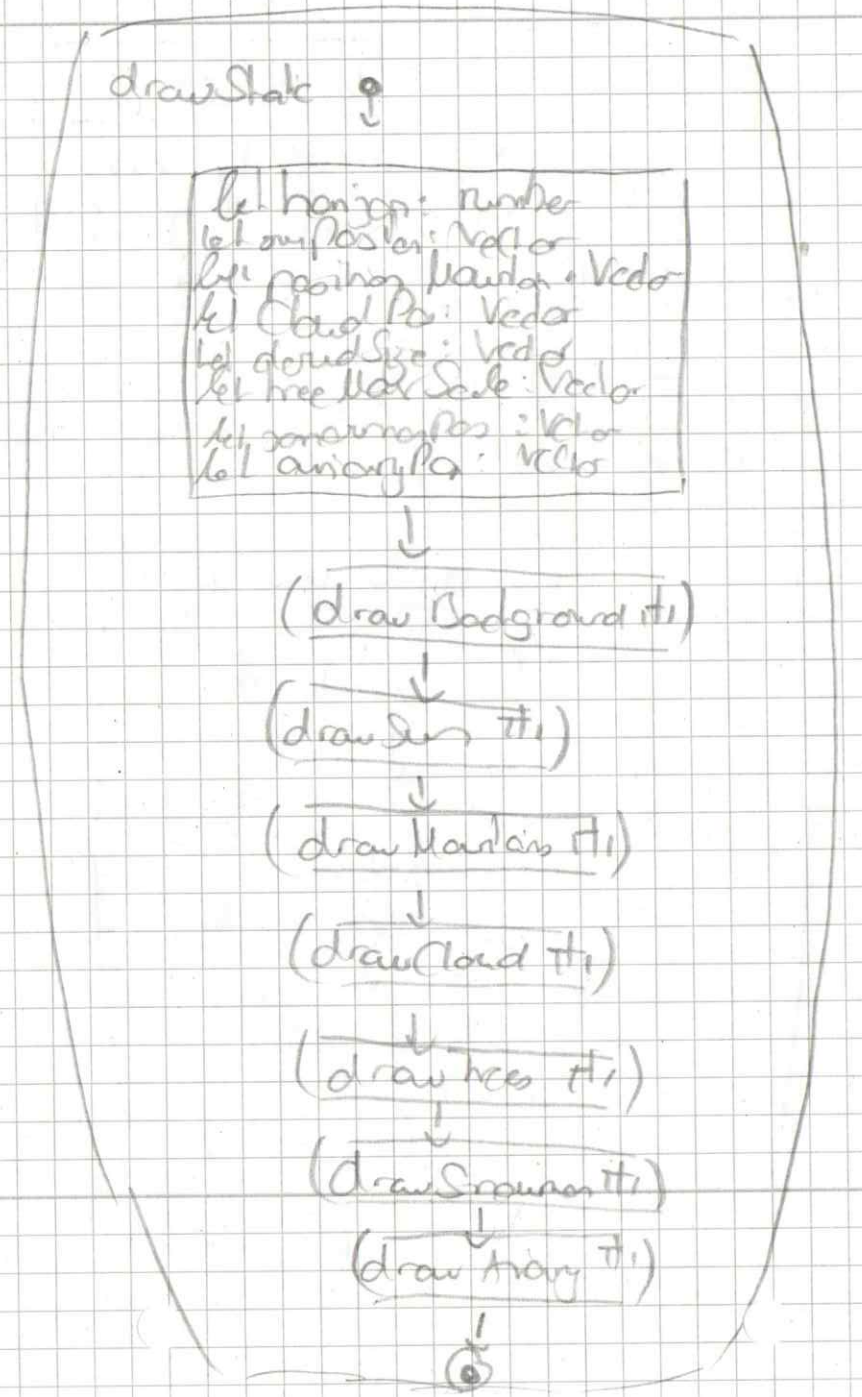

drawTree : snowPart : Number
nBranches=20
maxRadius=50
whiteBranches: nBranches * snowPart
greenBranches: nBranches - whiteBranches
(drawTrunk)
(save Transform)
(translate)
y= random 0-350
size: 1-y/700
x: random maxRadius to maxRadius
(save transform)
(translate 0,y scale size translate x,0)
(color green)
(drawBranch)
restore
(whiteBranch 70)
y: random 0-350
size: 1-y/700
x: random -maxRadius to maxRadius
(save)
(translate 0,y scale size translate x,0)
(color green)
(drawbranch)

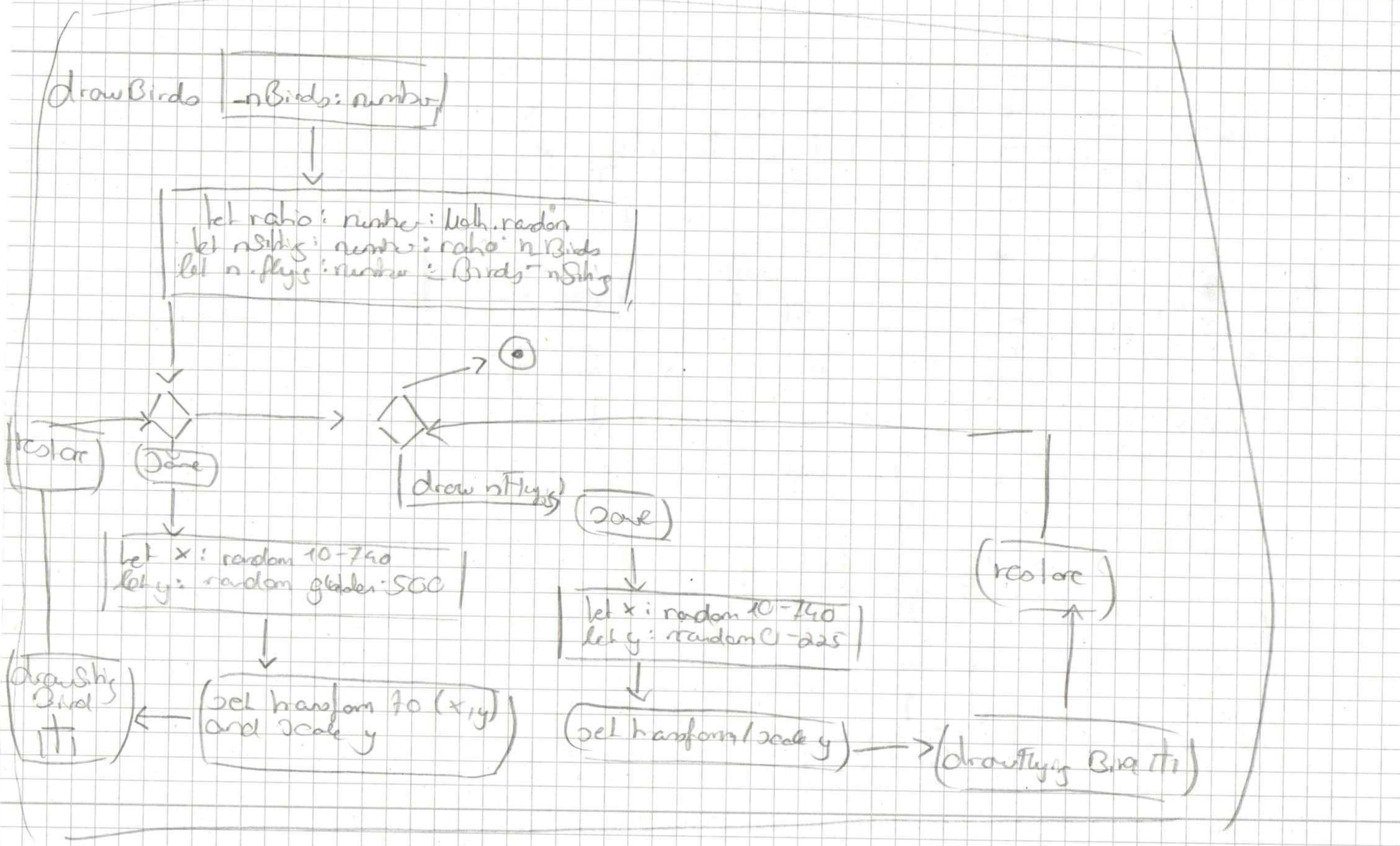

drawMountains
- position : Vector
- min : number
- max : number
- colorDraw : String
- colorFill : String
lengthi : number = 10
direction : number = 50
x : number = 0
save transform
translate to position
move to
line to
x = random
y : number = min + Math.random
line to (x, y)
restore
draw Path
color Path
check gradient
drawBackground
let gradient : CanvasGradient
add color stop
Fill with gradient

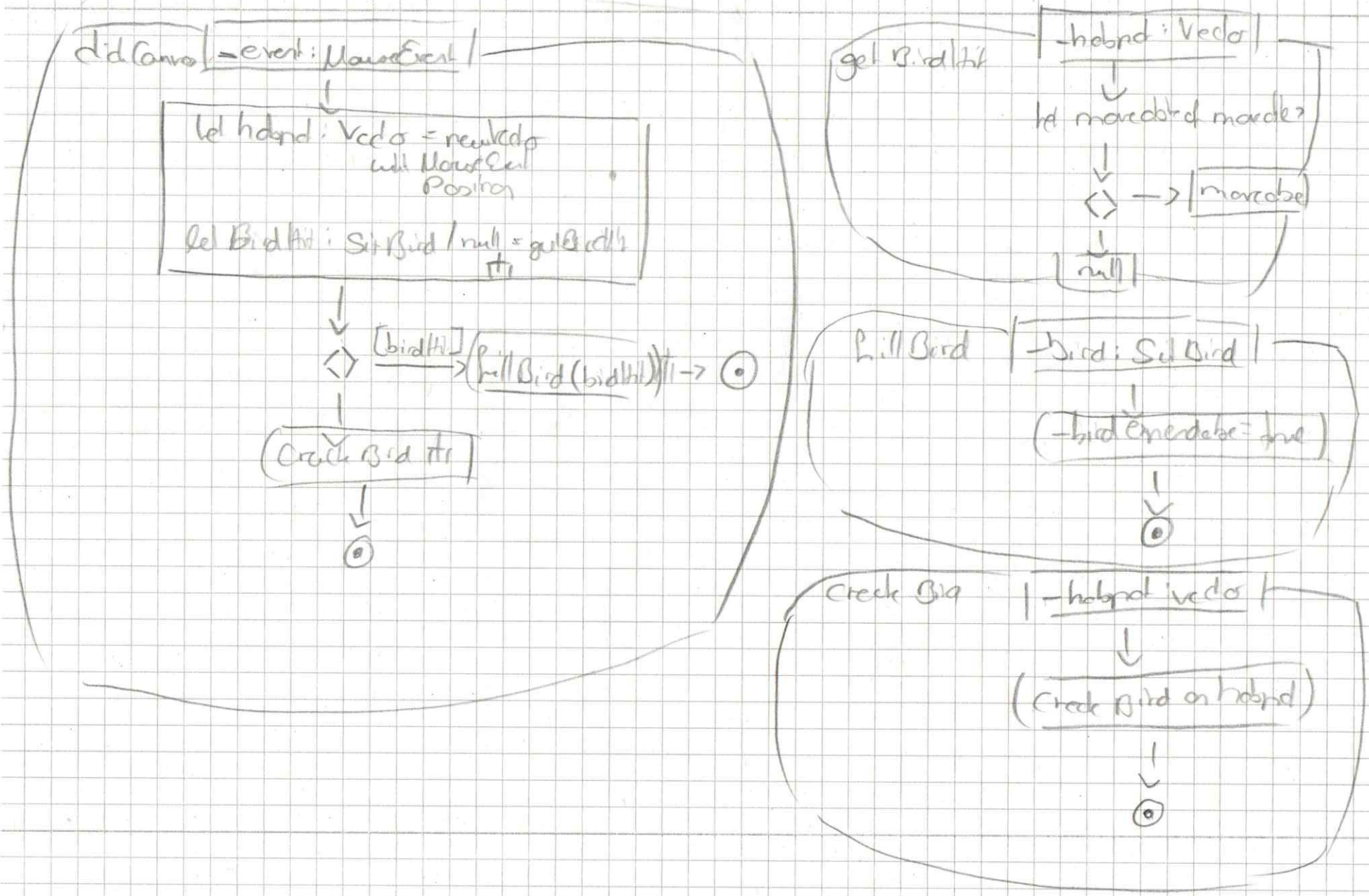

drawSnowman
position: Vector
let r1: number = randomBetween(20, 70)
let r2: number = r1/3
let r3: number = r2/3
let x1: Vector = {x: 0, y: r1}
let x2: Vector = {x: 0, y: r1}
let x3 = Vector = {x: 0, y: r2}
let color: all things color
let transform
translate
translate x1
drawCircle
translate to x2
drawCircle
translate to x3
drawCircle
translate
drawEye
drawEye
translate
drawEye
reset
set transform


drawScrollables
-n_Floor: number
-position: Vector
transform
translate
<
return transform
save transform
let pos: Vector = new Vector
let bg: Vector = new Vector
let scrollObj: Scrolla = new
scrollObj draw
transform
drawImg
-position: Vector
-nBirds: number
transform
translate
drawSland
drawHouse
drawSmallBird(n)

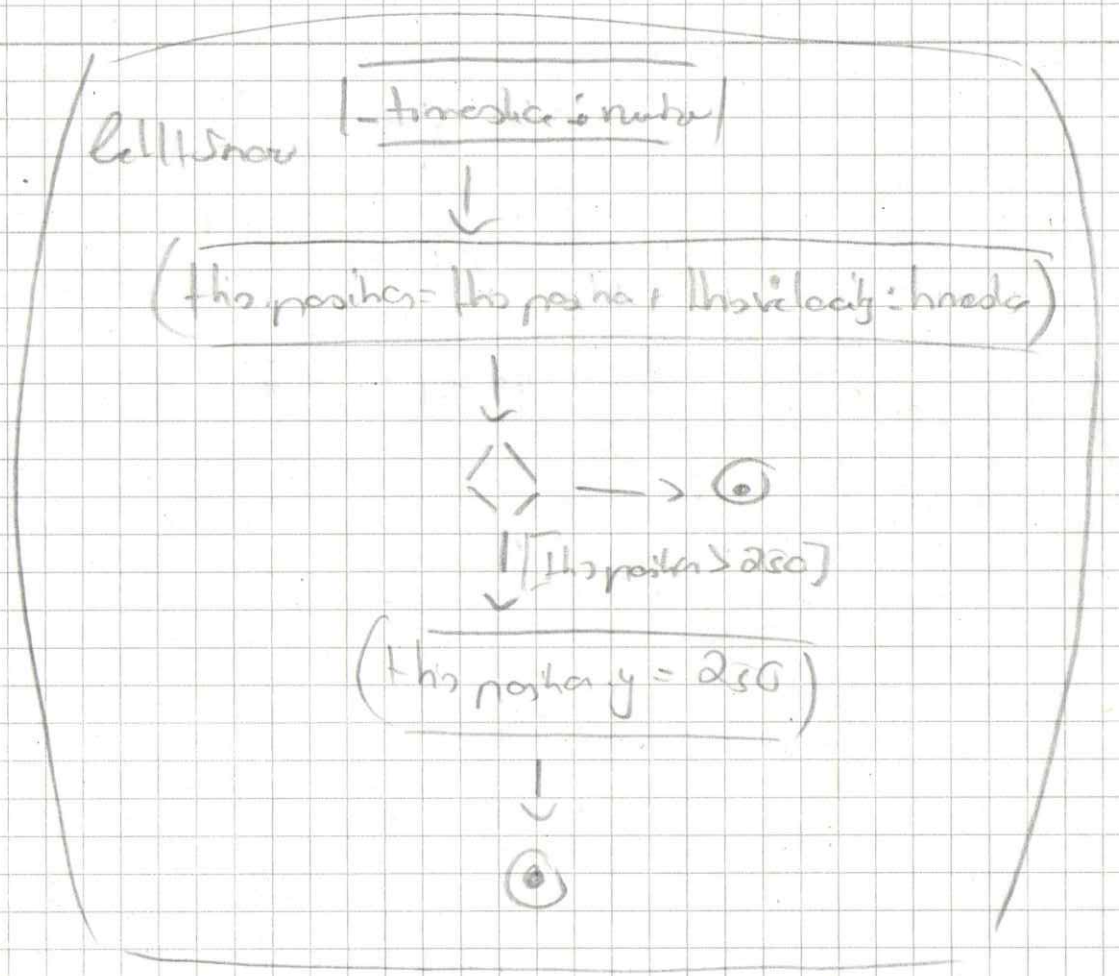
2

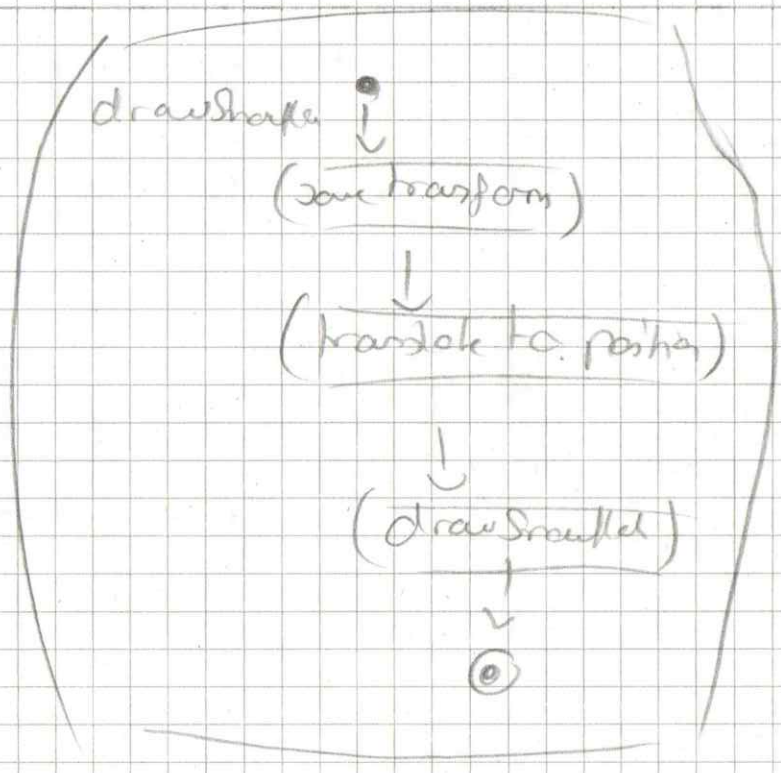

drawState
let horizon: number
let sunPosition: Vector
let mountain1Vector: Vector
let cloudPos: Vector
let cloudSize: Vector
let treeVarScale: Vector
let sunburstPos: Vector
let arrowPos: Vector
(drawBackground #1)
(drawSun #1)
(drawMountains #1)
(drawCloud #1)
(drawTrees #1)
(drawSnowman #1)
(drawArrow #1)


drawBirds
_nBirds: number
let ratio: number: Math.random
let nSitting: number: ratio · nBirds
let n.flying: number = Birds - nSitting
restore
Done
let x: random 10-740
let y: random golden: 500
set transform to (x, y) and scale y
drawSitting Bird (ti)
draw nFlying
Done
let x: random 10-740
let y: random 0-225
set transform/scale y
drawFlying Bird (ti)
restore

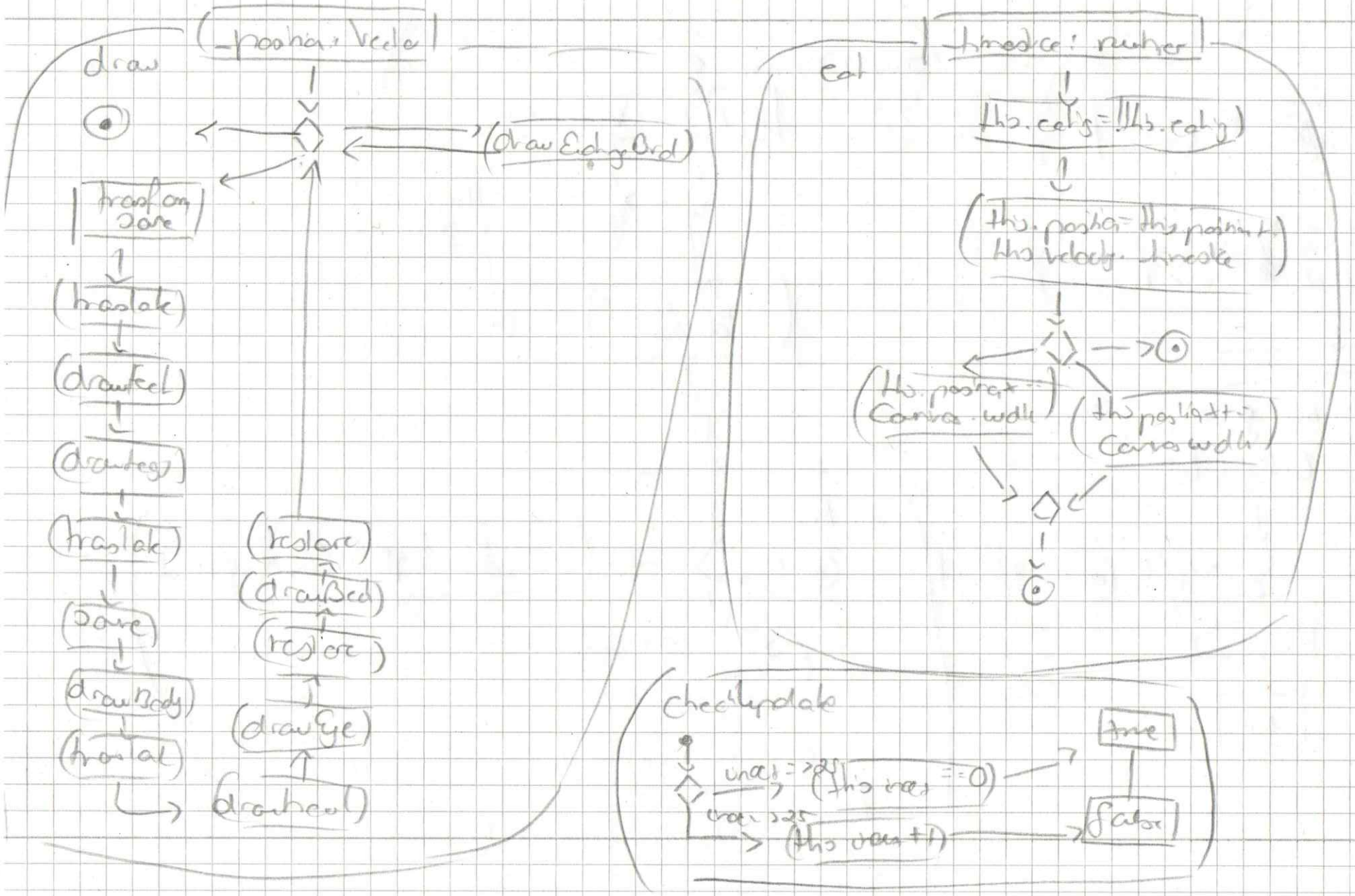

didConvo
event: MouseEvent
let hdand: Vec2 = new Vec2
with MouseEvent
Position
let BirdHit: Si Bird | null = getBirdAt
[birdHit]
killBird(birdHit)
CreateBird

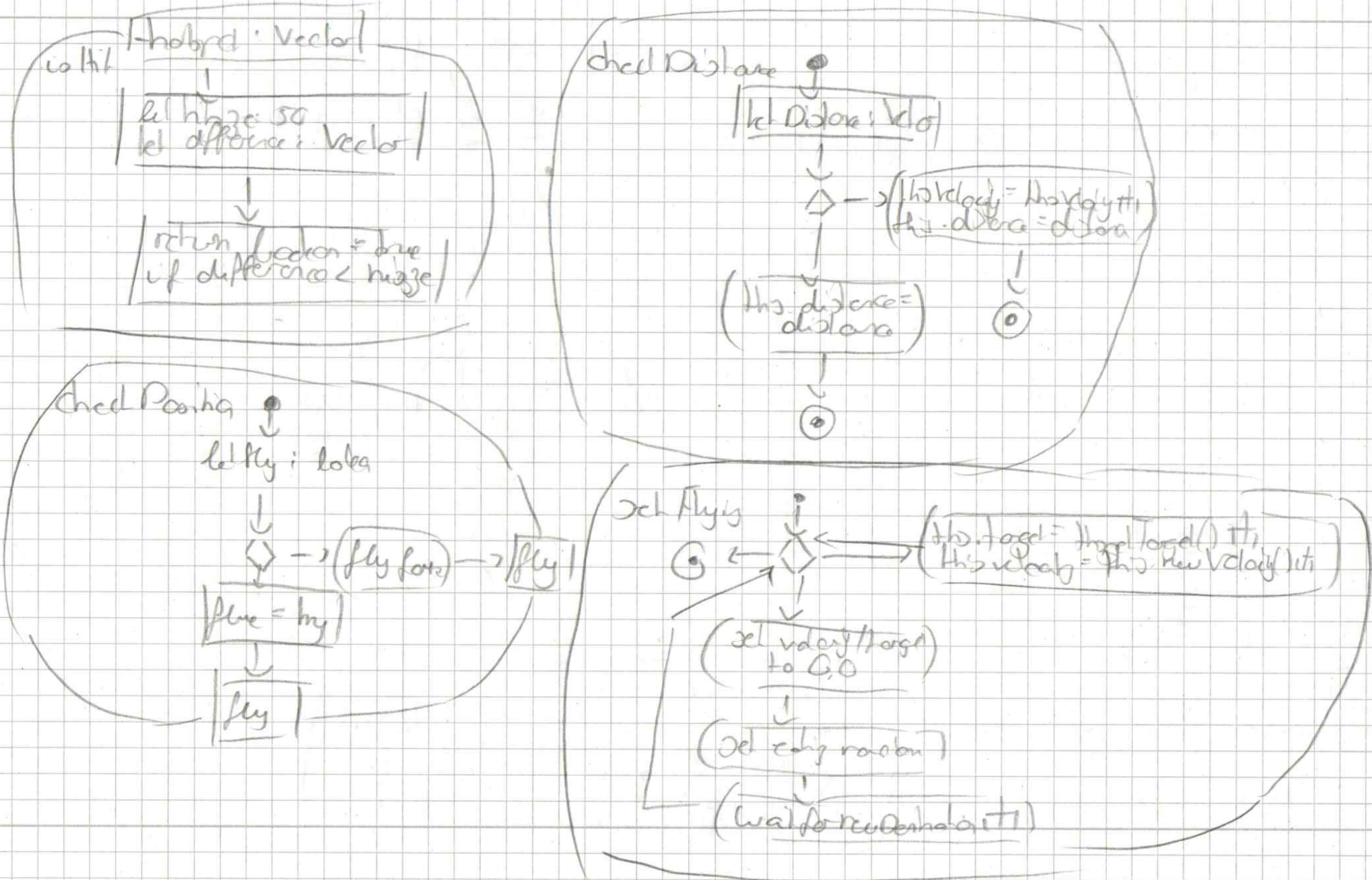
getBirdAt
-hdand: Vec2
let moveBird: maybe
null
killBird
-bird: SiBird
(-bird.removable = true)
CreateBird
-hdand: Vec2
(CreateBird on hdand)


fallSnow
|_timeSinceLastSnow
this.position = this.position + this.velocity * timeSinceLastSnow
this.position.y > 250
this.position.y = 250

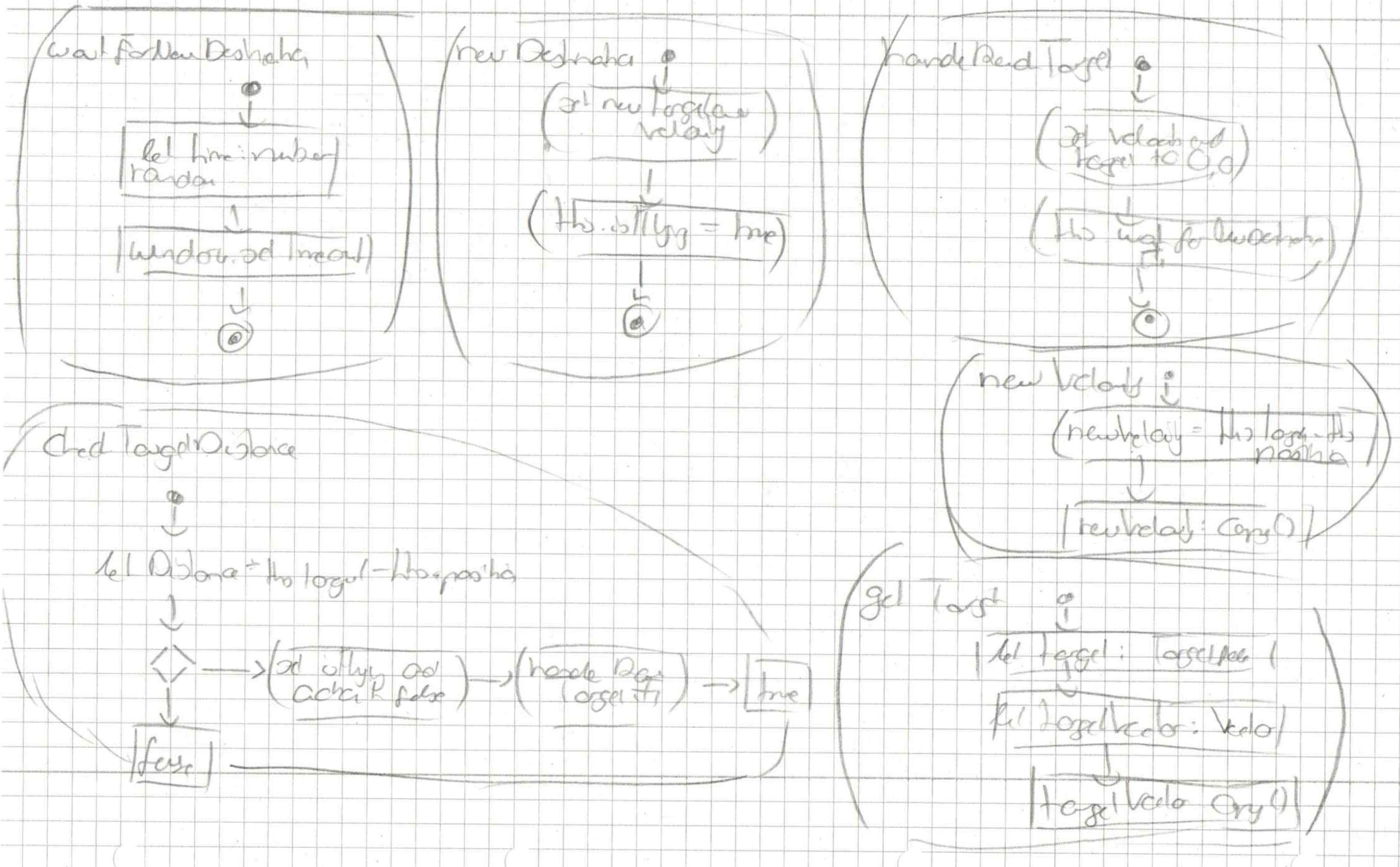

drawSnowflake
save transform
translate to position
drawSnowflake

Activity diagram: Birds





collide
let hitSize 50
let difference : Vector
return boolean true if difference < hitSize
checkDistance
let Distance : Vector
this.distance = distance
checkPosition
let fly : boolean
fly = true
fly false
fly
setFlying
set velocity/target to 0,0
wait for newPosition



waitForNewDestination
window.setTimeout()
newDestination
this.isMoving = true
handleReachTarget
newVelocity
newVelocity.copy()
checkTargetDistance
true
false
getTarget
targetVector.copy()