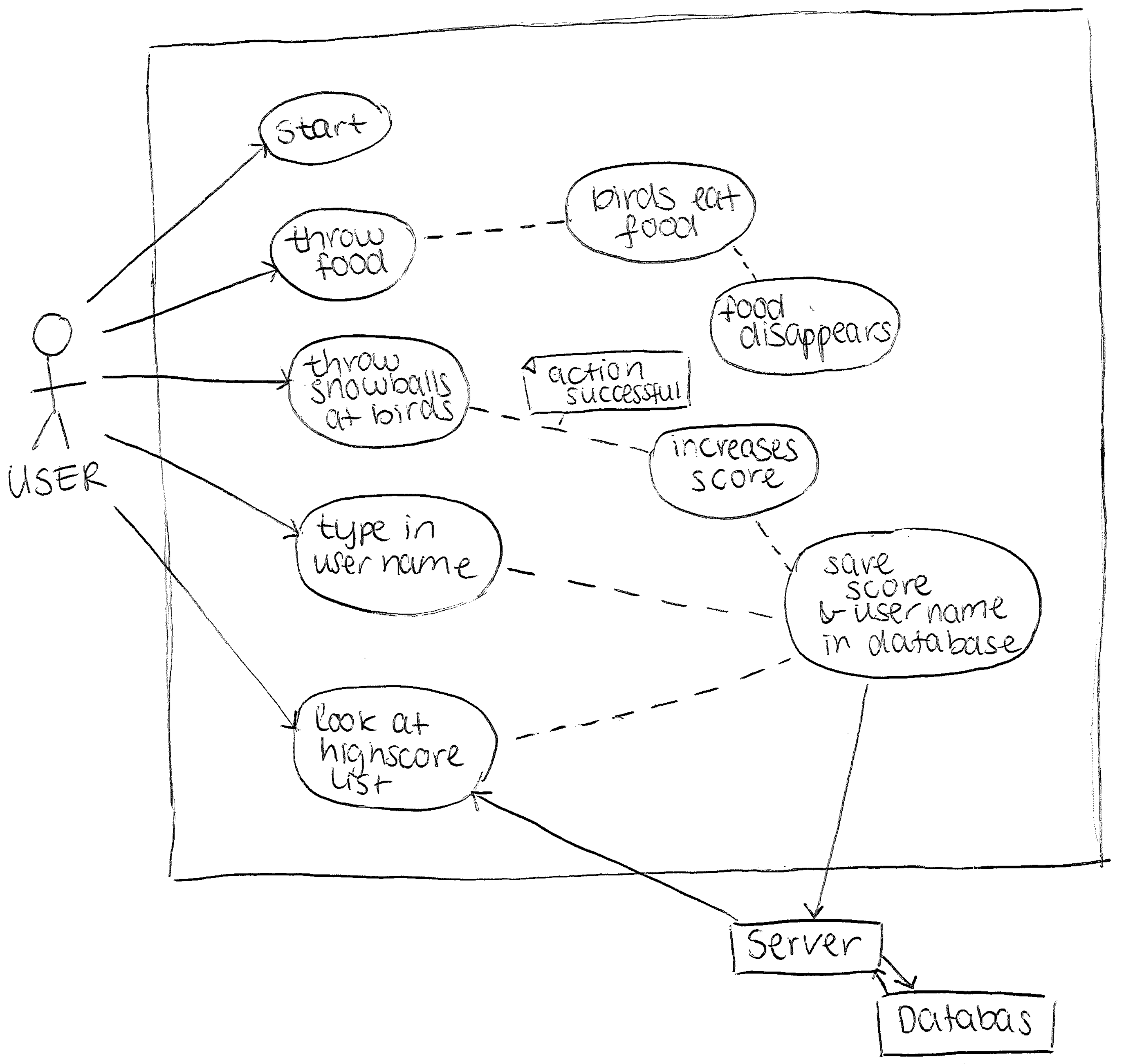
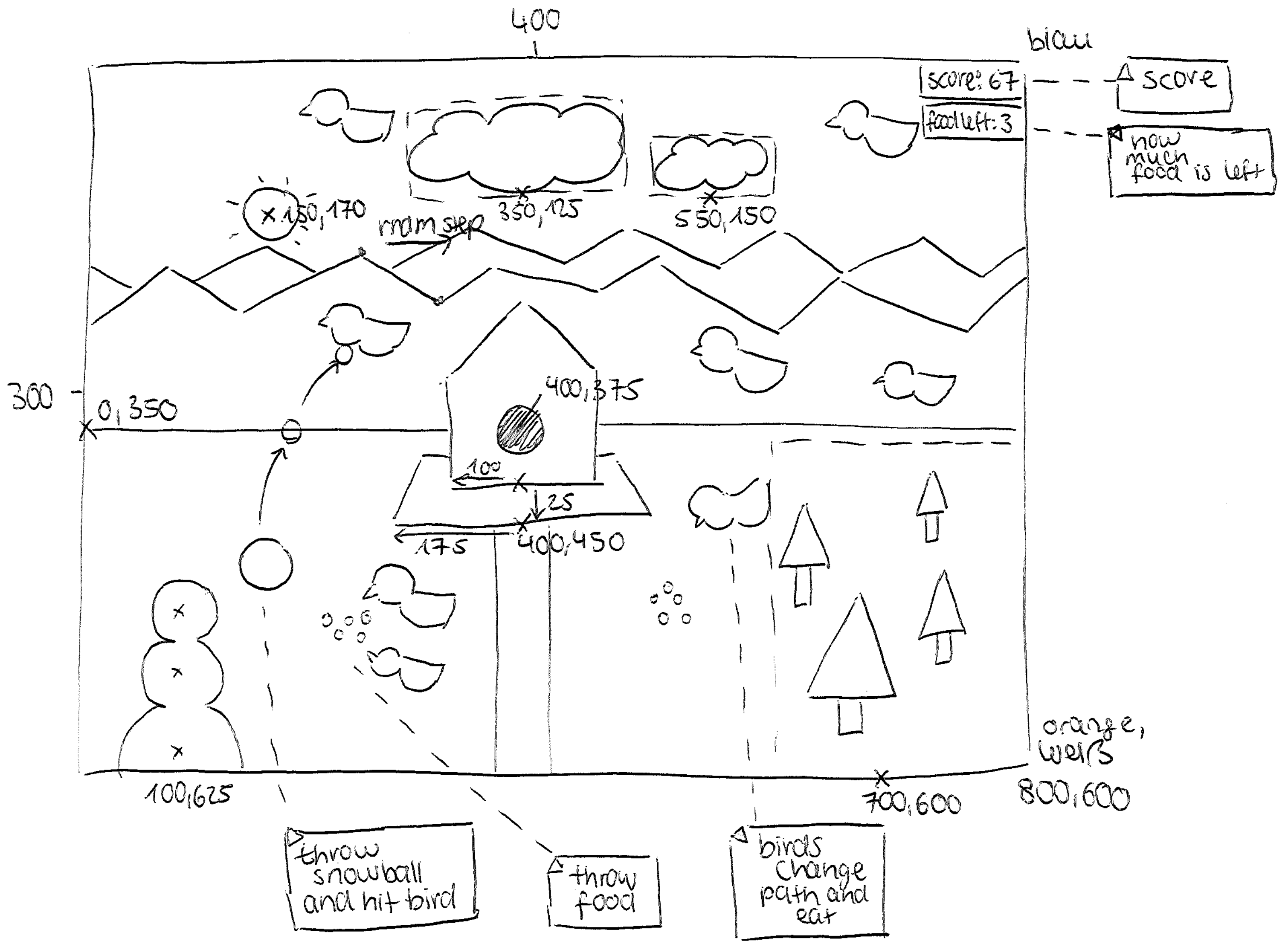
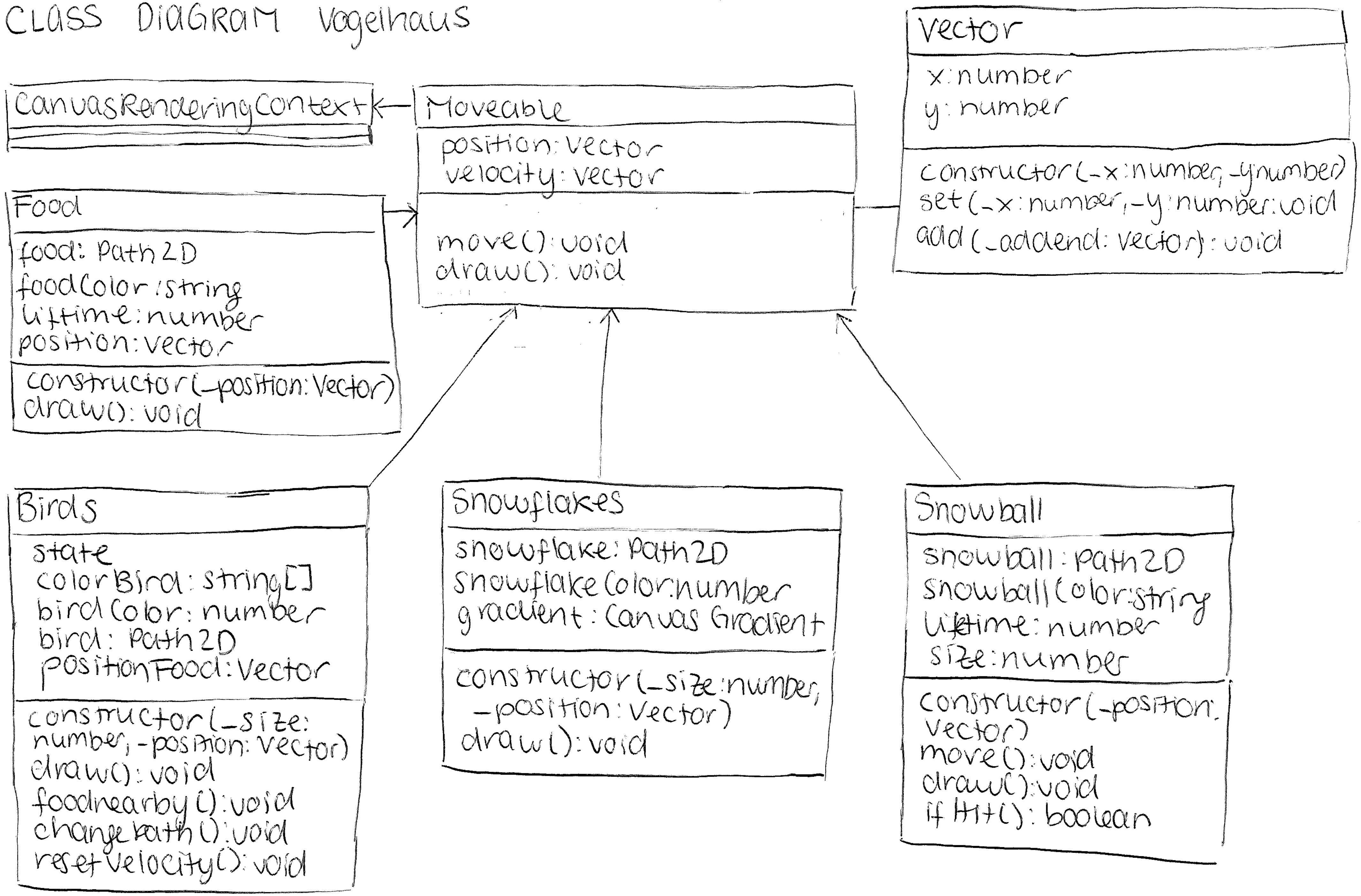
USE CASE DIAGRAM Vogelhaus



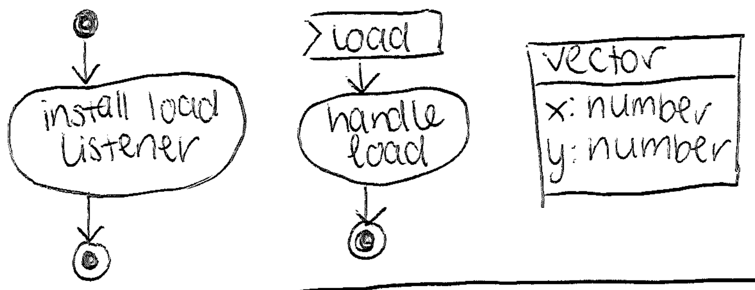
USER INTERFACE Vogelhaus

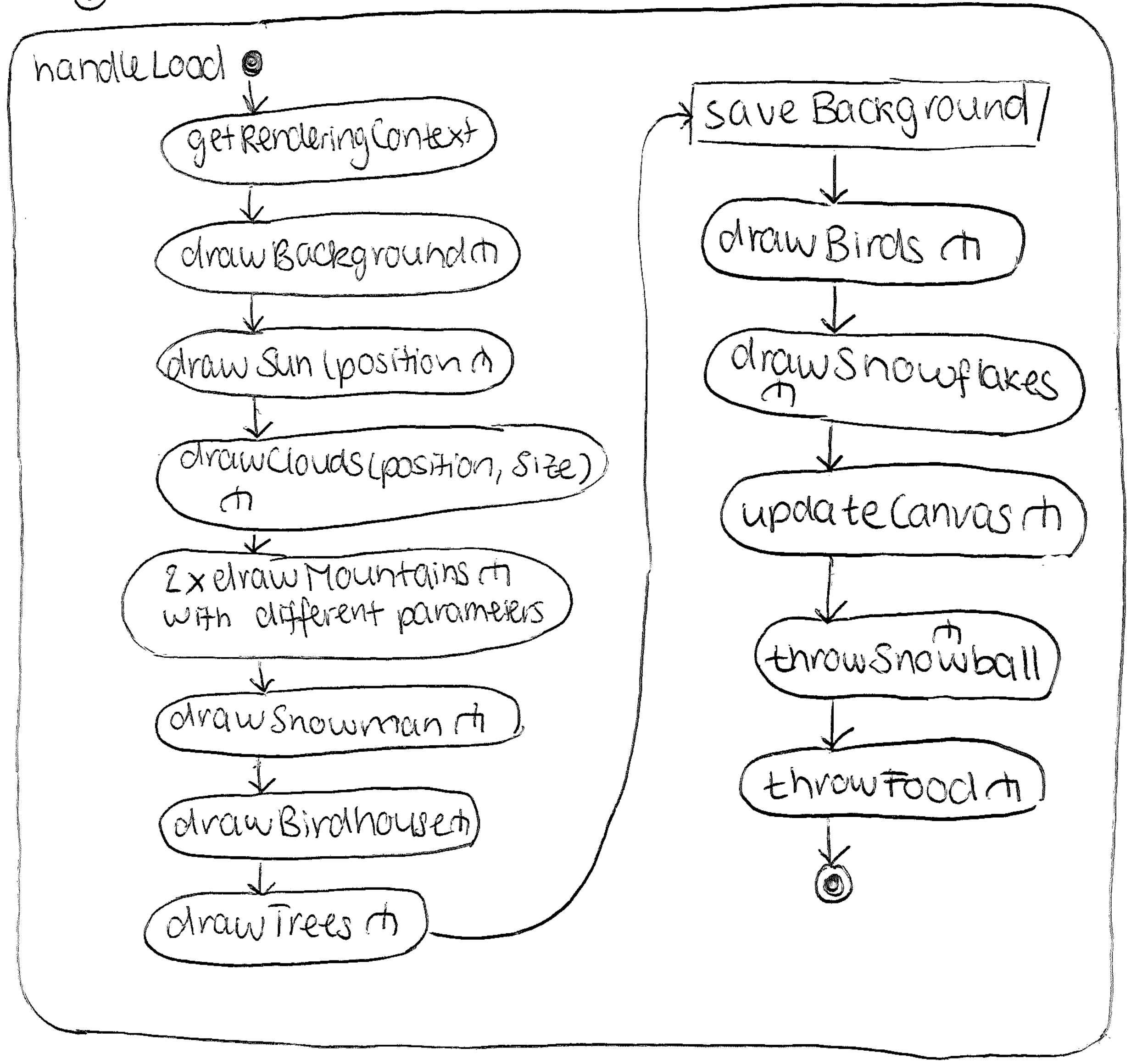


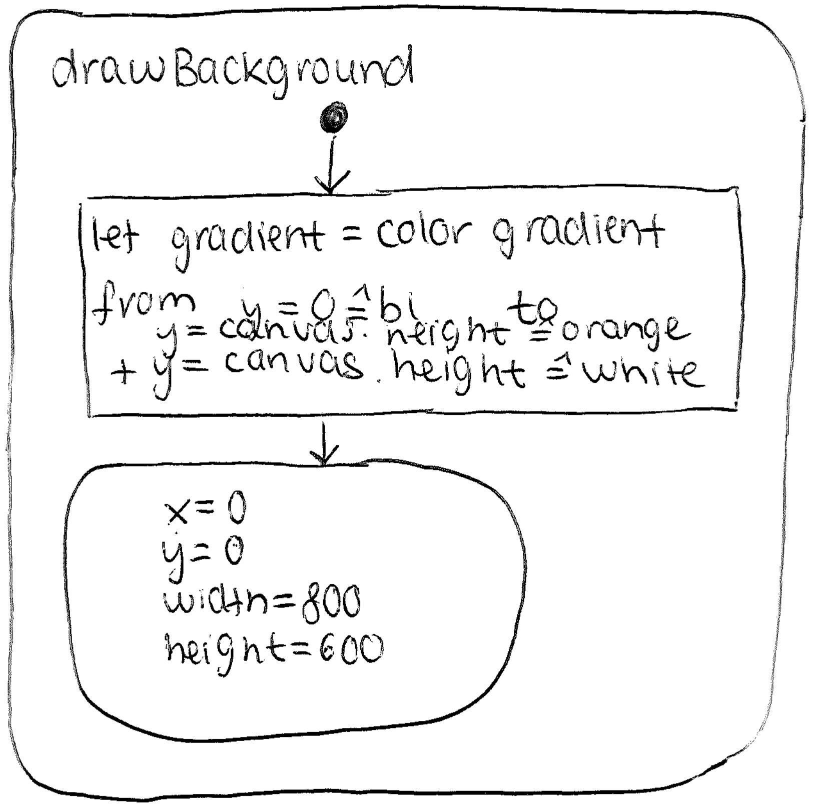
CLASS DIAGRAM Vogelhaus

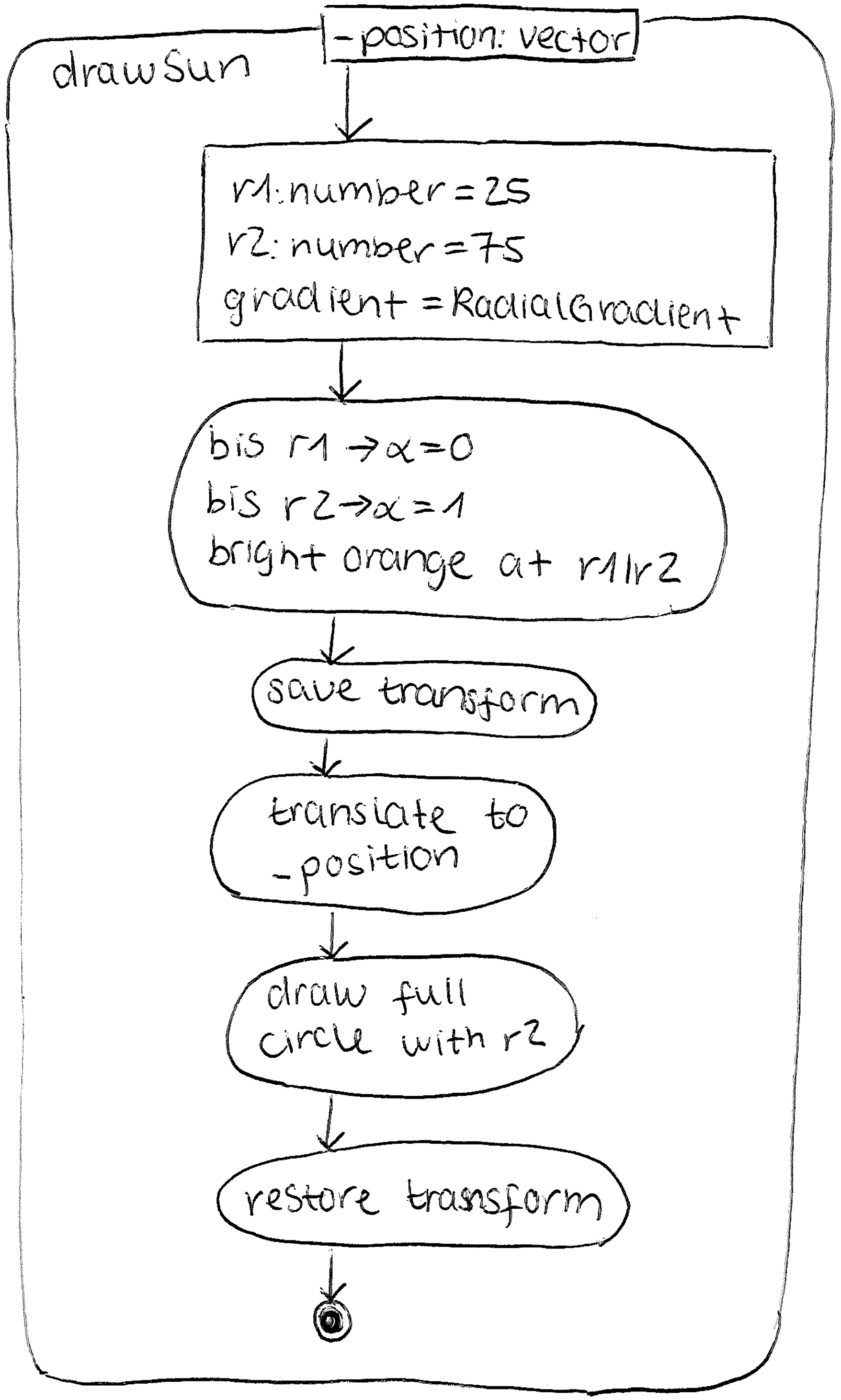


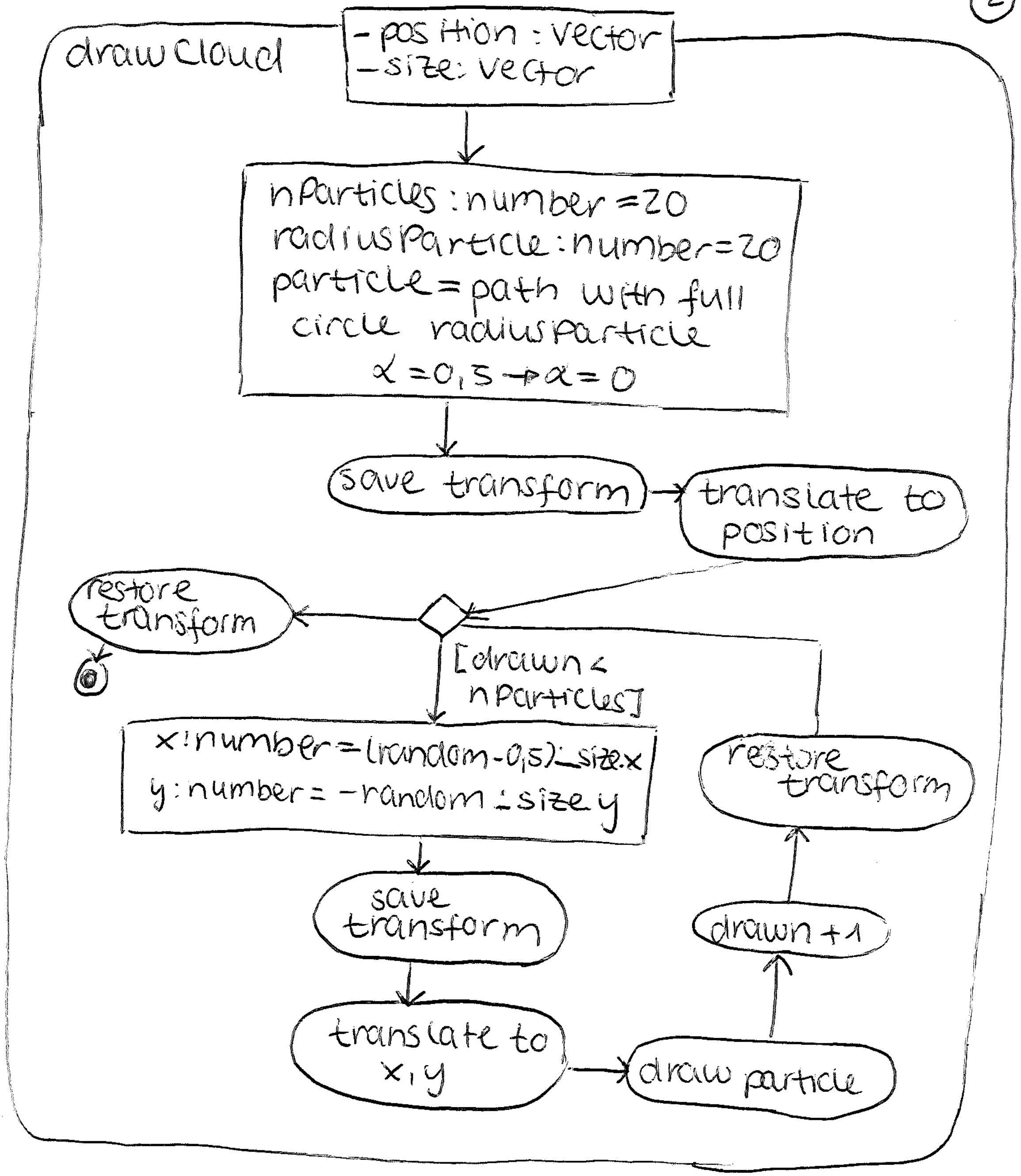
ACTIVITY DIAGRAMS Vogelhaus

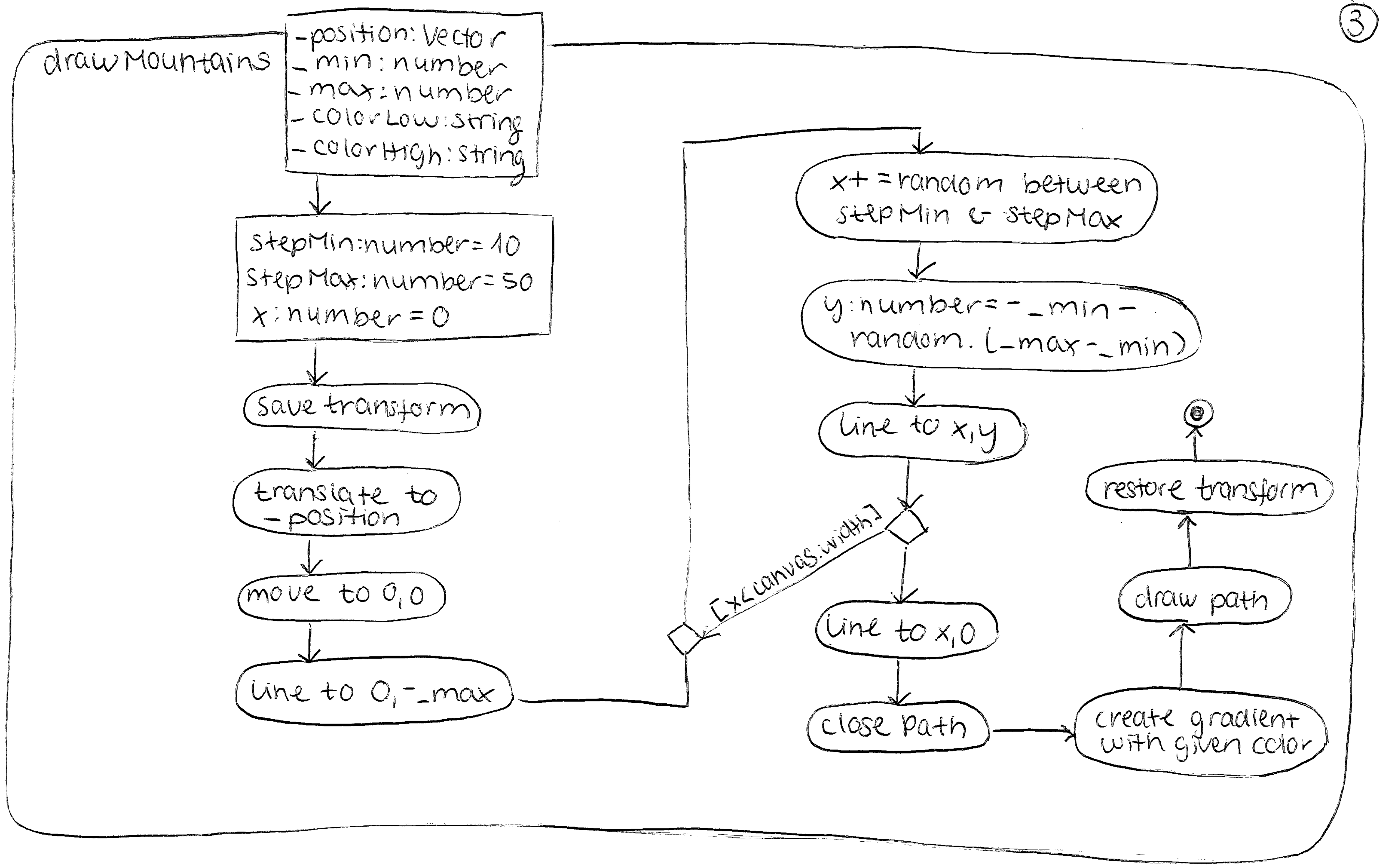


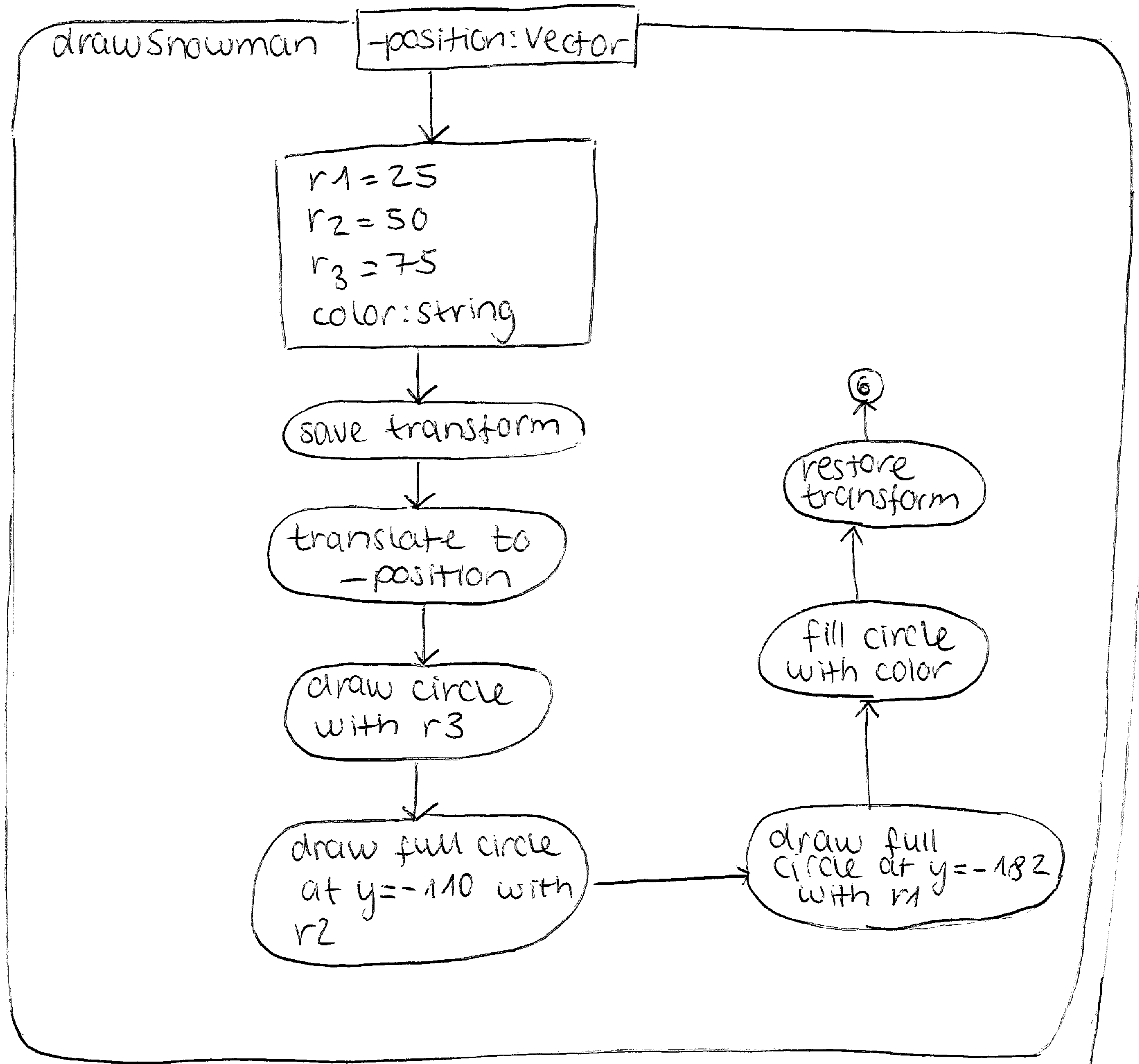


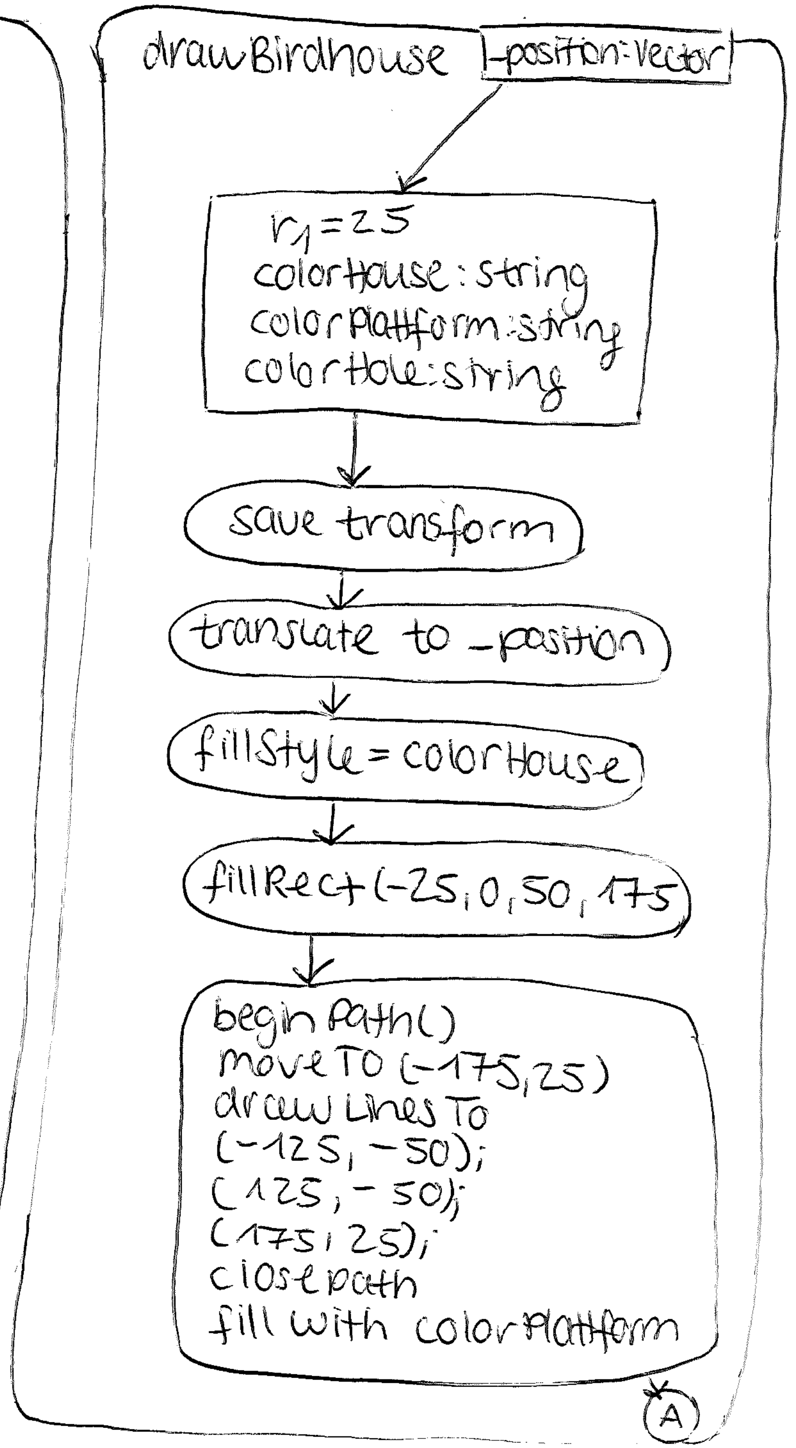

drawBackground
let gradient = color gradient
from y = 0 ≙ bl
y = canvas. height to ≙ orange
+ y = canvas. height ≙ white
x = 0
y = 0
width = 800
height = 600

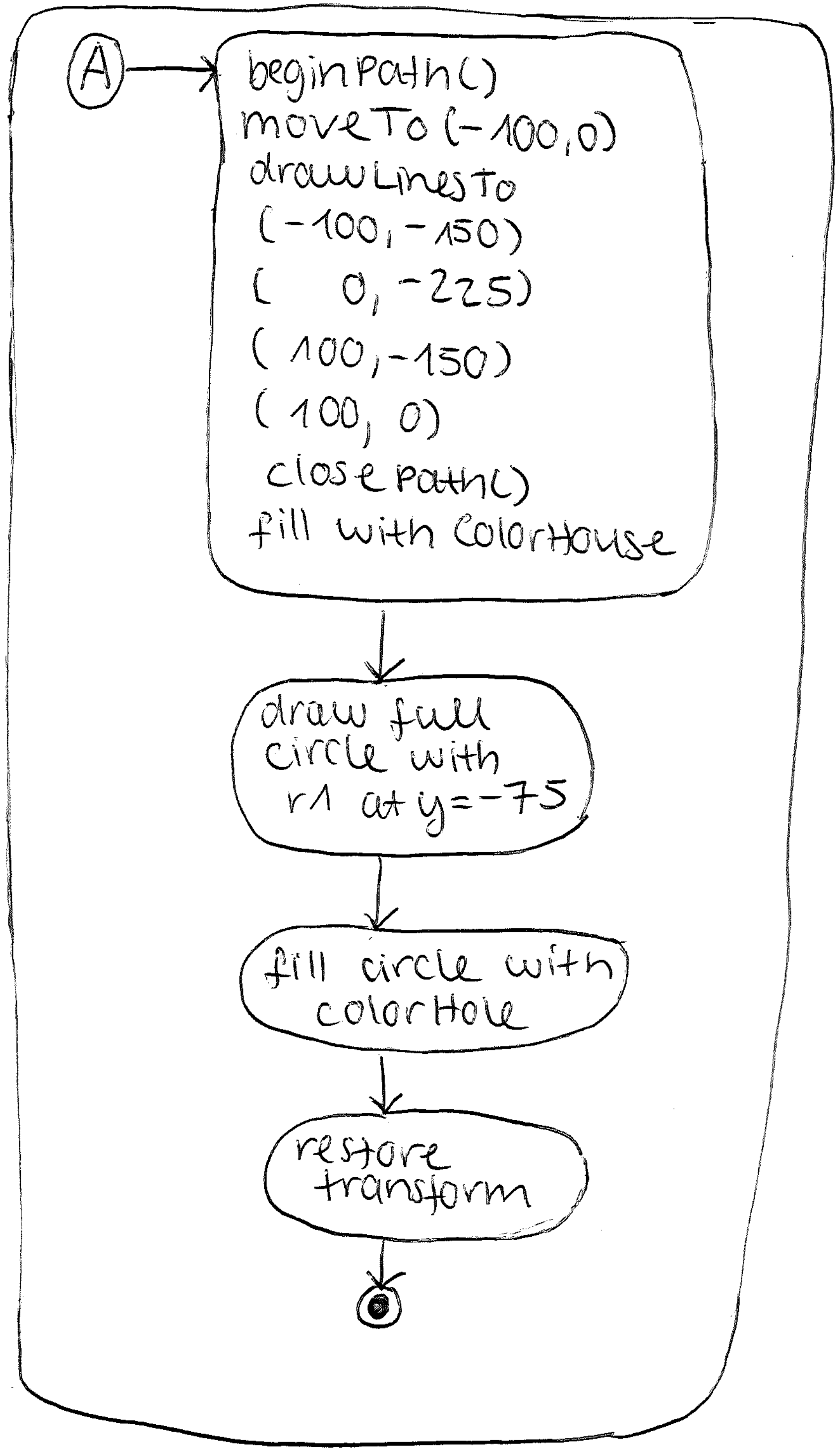

drawSun
-position: vector
r1:number=25
r2: number=75
gradient = RadialGradient
bis r1 → α=0
bis r2 → α=1
bright orange at r1|r2
save transform
translate to _position
draw full circle with r2
restore transform

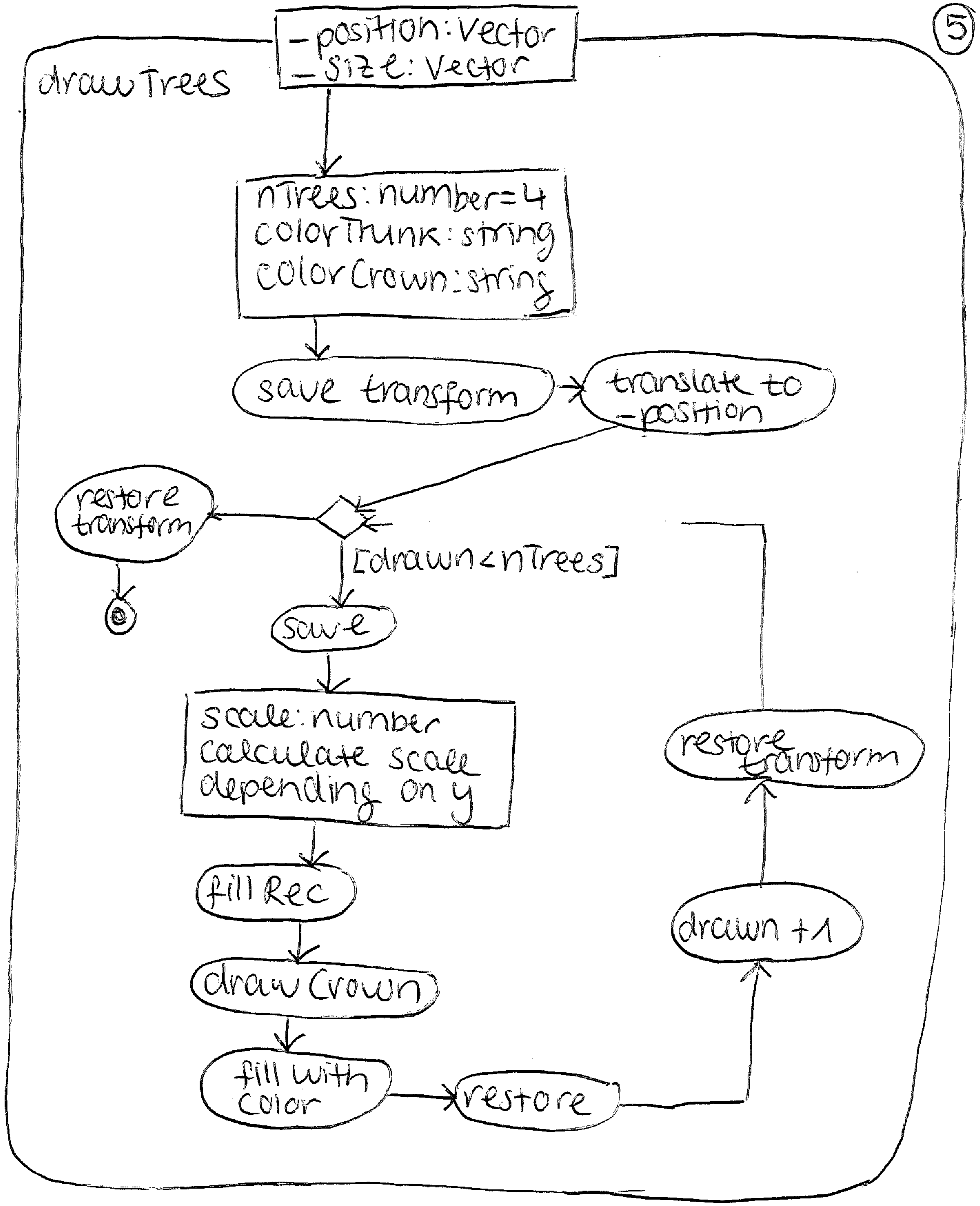

drawCloud
-position : vector
-size: vector
nParticles : number =20
radiusParticle : number=20
particle = path with full circle radiusParticle
α = 0,5 → α = 0
save transform
translate to position
restore transform
[drawn < nParticles]
x: number = (random-0,5) * size.x
y: number = -random * size.y
save transform
translate to x, y
draw particle
drawn +1
restore transform

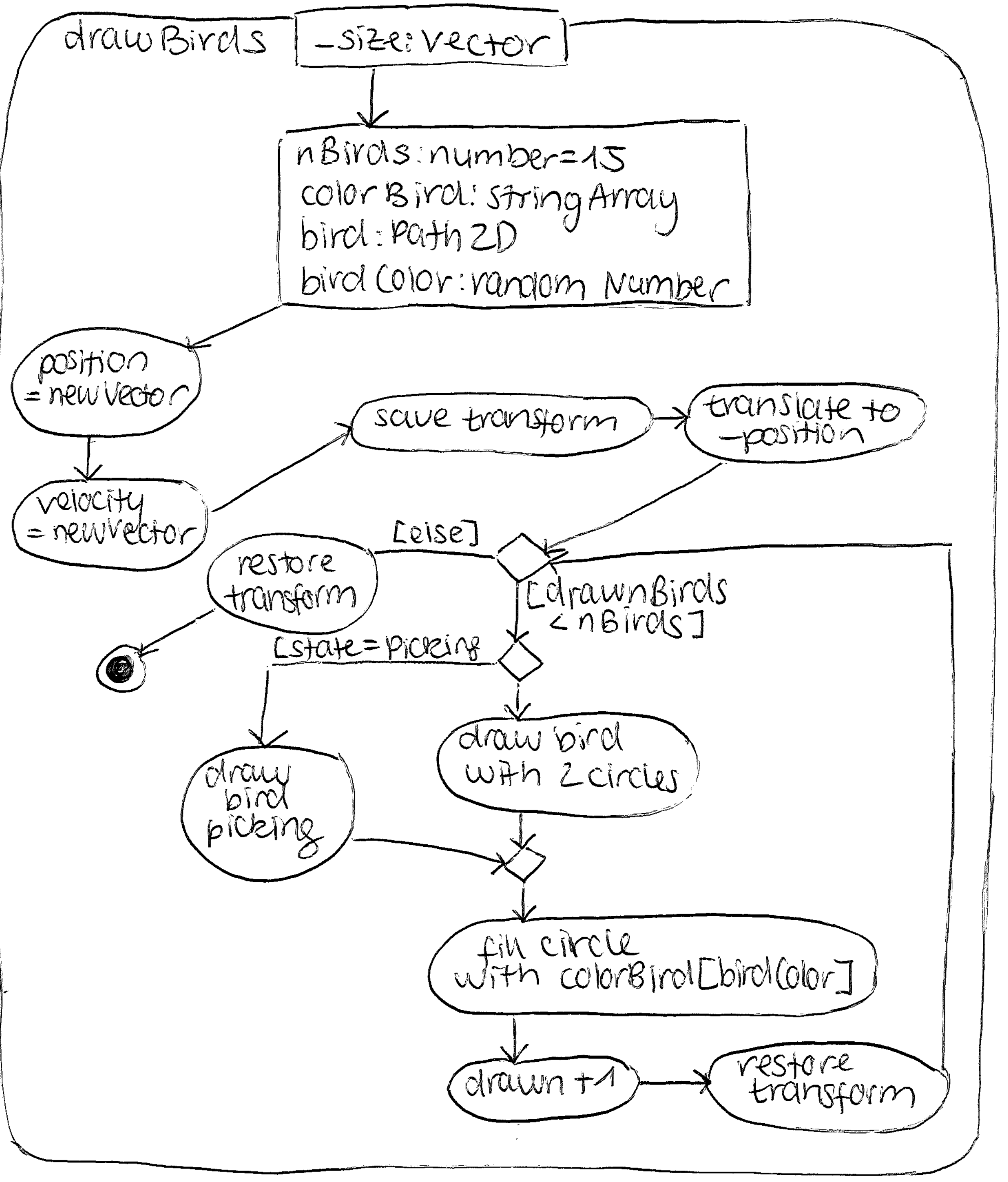

drawMountains
-position: Vector
_min: number
_max: number
_colorLow: string
_colorHigh: string
stepMin: number = 10
stepMax: number = 50
x: number = 0
save transform
translate to _position
move to 0,0
line to 0,-_max
[x<canvas.width]
x+ = random between stepMin & stepMax
y: number = -_min - random. (_max-_min)
line to x,y
line to x,0
close Path
create gradient with given color
draw path
restore transform

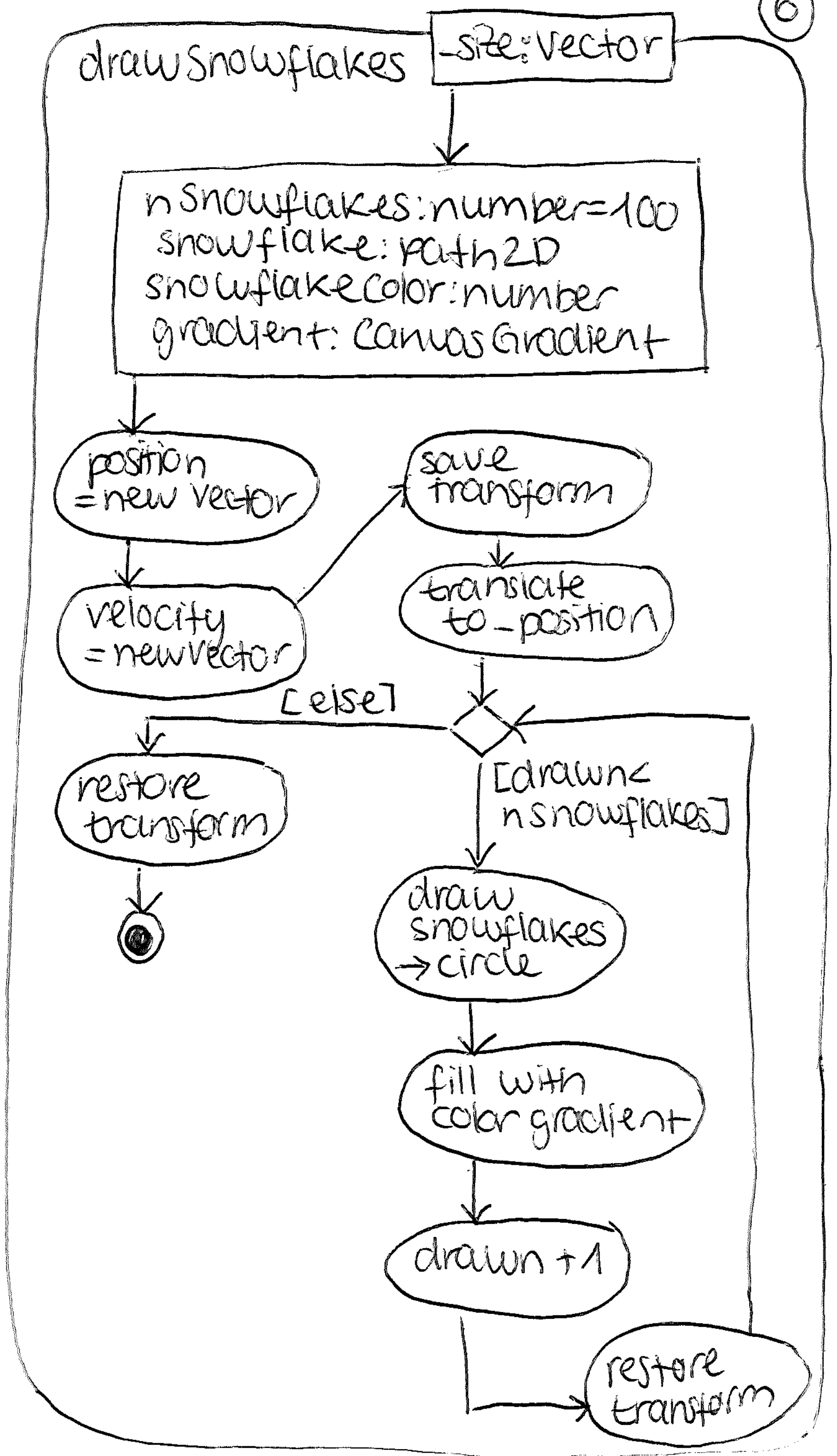

drawSnowman
-position: Vector
r1 = 25
r2 = 50
r3 = 75
color: string
save transform
translate to _position
draw circle with r3
draw full circle at y=-110 with r2
draw full circle at y=-182 with r1
fill circle with color
restore transform
6

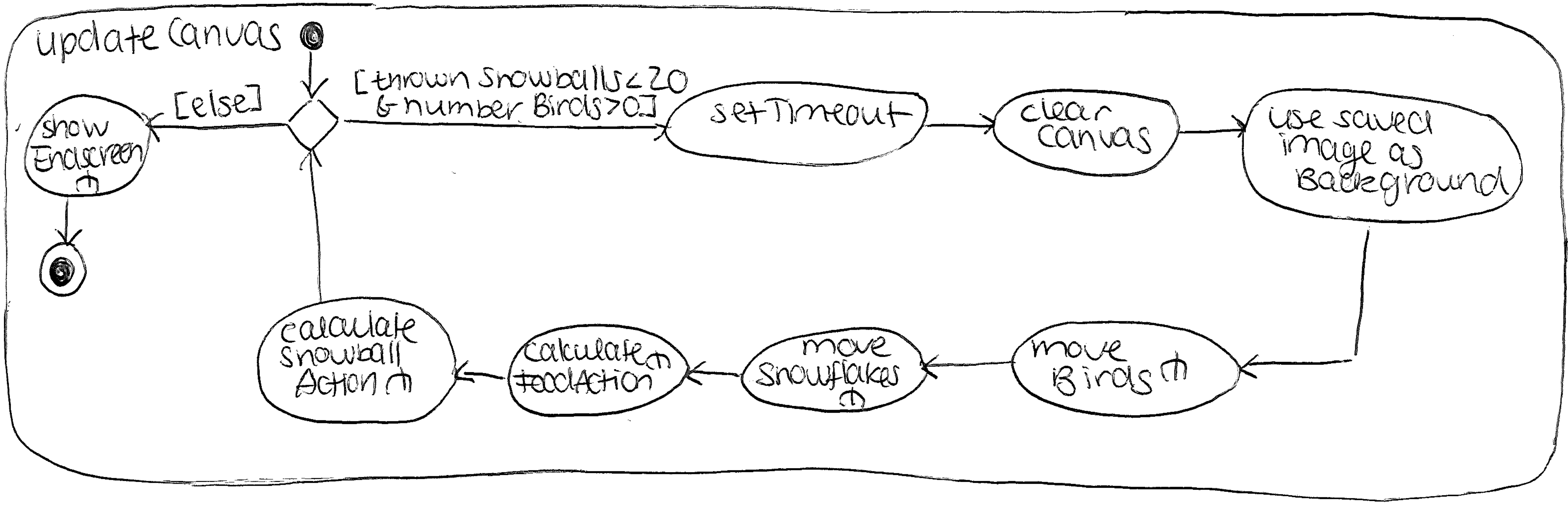

drawBirdhouse
-position: Vector
r1 = 25
colorHouse: string
colorPlattform: string
colorHole: string
save transform
translate to _position
fillStyle = colorHouse
fillRect(-25, 0, 50, 175)
beginPath()
moveTO(-175, 25)
drawLinesTo
(-125, -50);
(125, -50);
(175, 25);
closePath
fill with colorPlattform
A

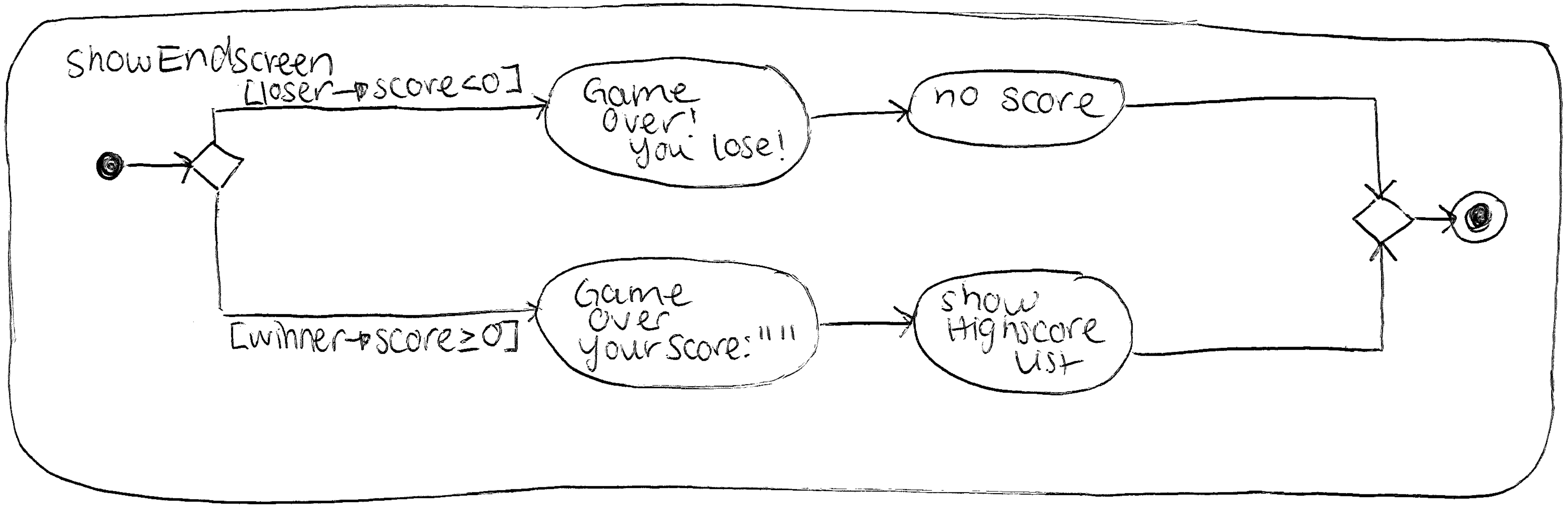

A
beginPath()
moveTo(-100,0)
drawLinesTo
(-100,-150)
(0,-225)
(100,-150)
(100, 0)
closePath()
fill with ColorHouse
draw full circle with r1 at y=-75
fill circle with colorHole
restore transform

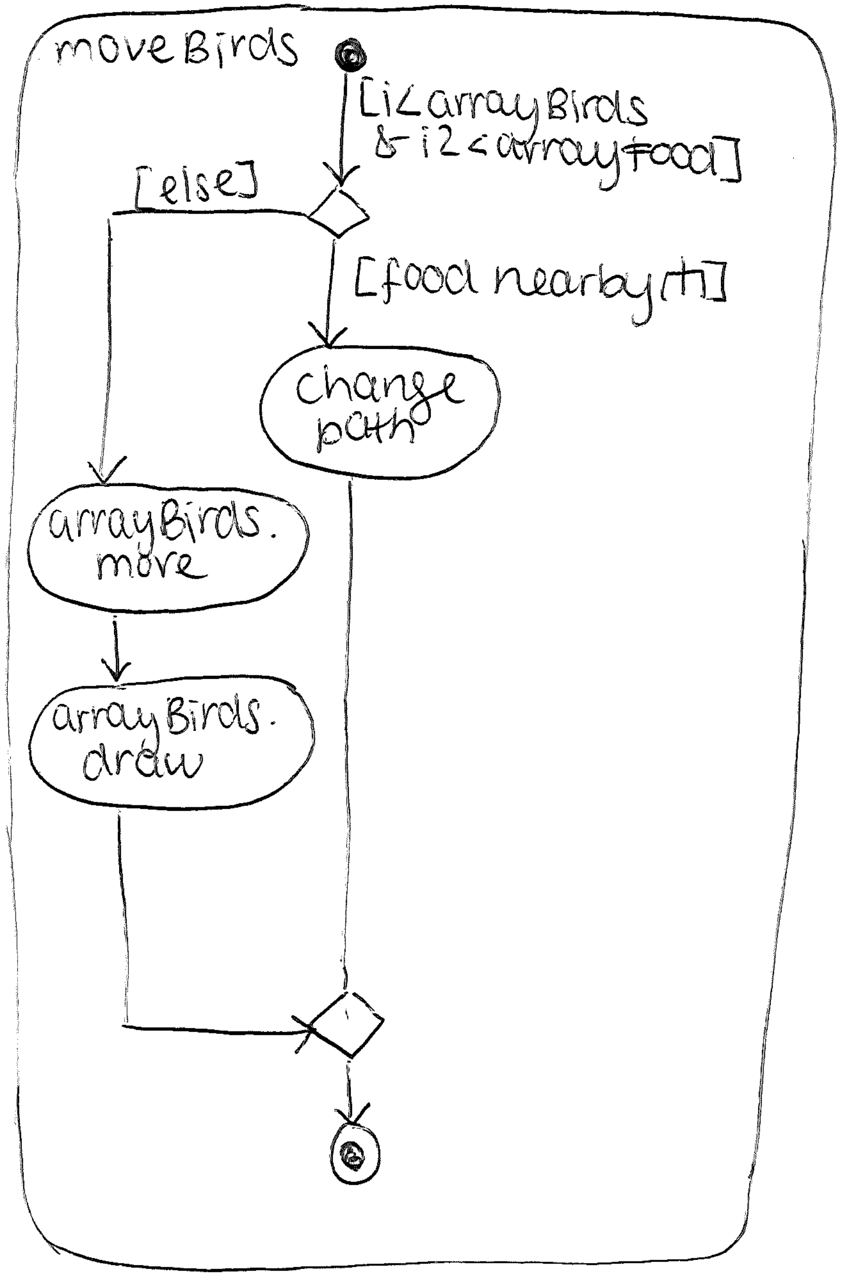

drawTrees
_position: Vector
_size: Vector
nTrees: number=4
colorTrunk: string
colorCrown: string
save transform
translate to _position
restore transform
[drawn<nTrees]
save
scale: number
calculate scale depending on y
fill Rec
draw Crown
fill with color
restore
drawn +1
restore transform

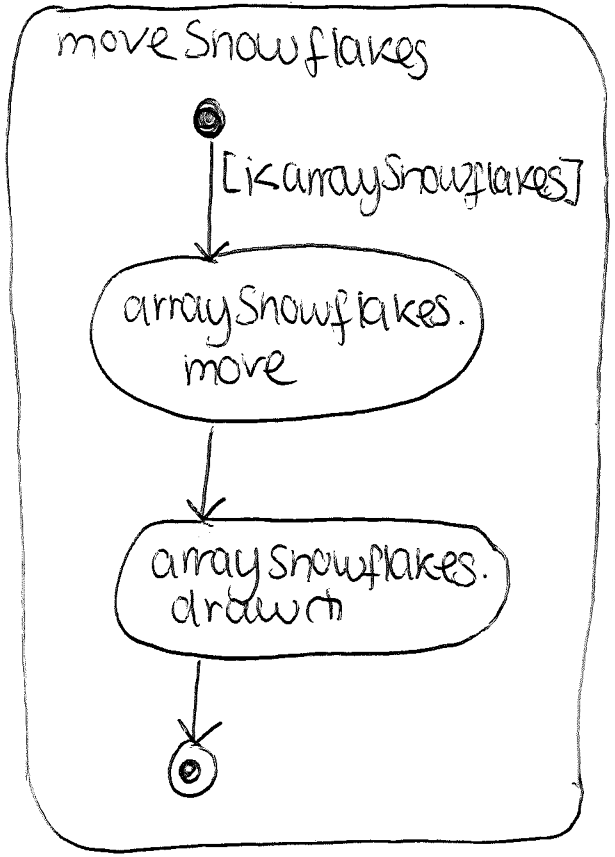

drawBirds
_size: vector
nBirds: number=15
colorBird: String Array
bird: Path2D
birdColor: random Number
position = newVector
velocity = newVector
save transform
translate to _position
[else]
restore transform
[drawnBirds < nBirds]
[state=Picking
draw bird picking
draw bird with 2 circles
fill circle with colorBird[birdColor]
drawn +1
restore transform

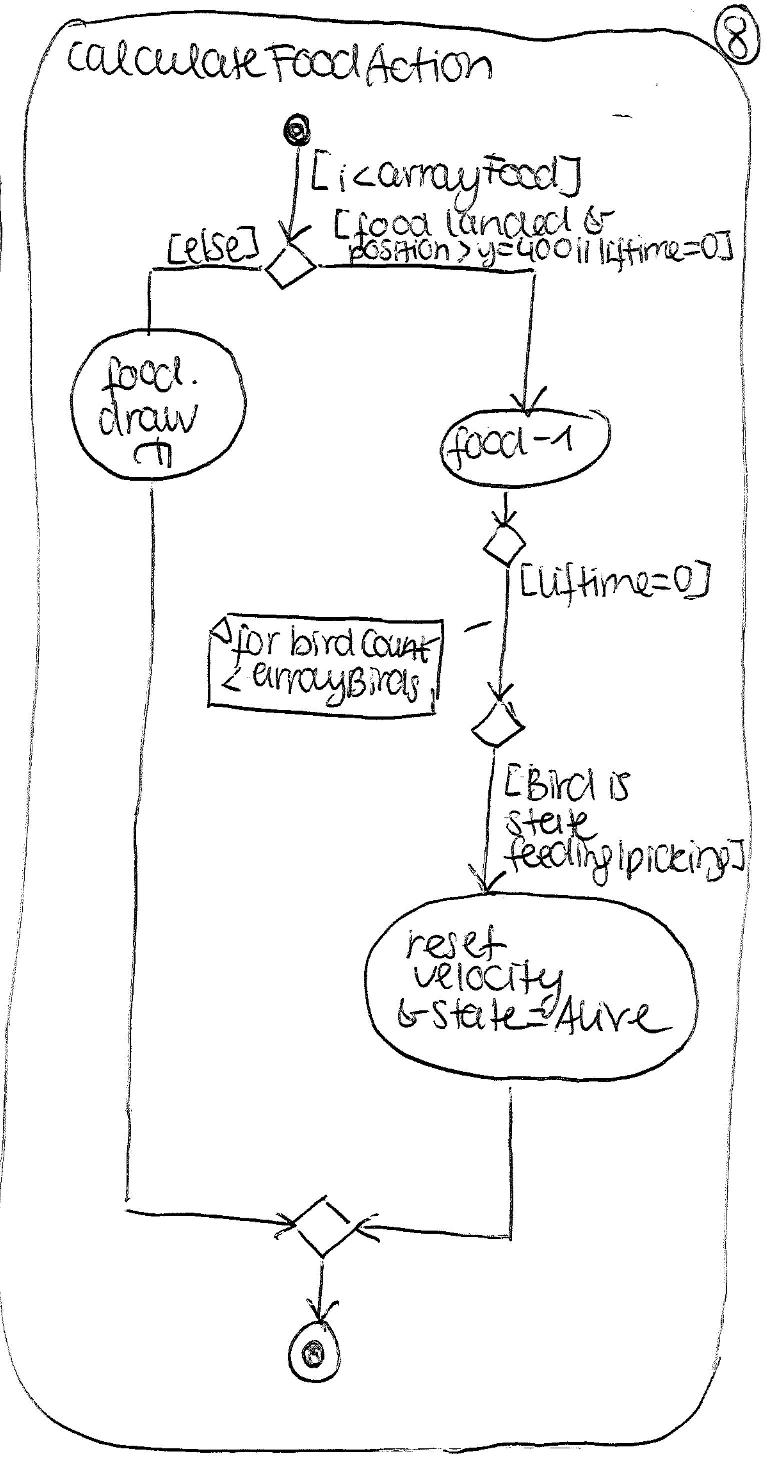

drawSnowflakes
_size: vector
nSnowflakes: number=100
snowflake: Path2D
snowflakeColor: number
gradient: CanvasGradient
position = new Vector
velocity = newVector
save transform
translate to _position
[else]
restore transform
[drawn< nSnowflakes]
draw snowflakes → circle
fill with color gradient
drawn +1
restore transform


updateCanvas
[else]
[thrown Snowballs<20 & number Birds>0]
show Endscreen
setTimeout
clear canvas
use saved image as Background
move Birds
move Snowflakes
Calculate ~~Food~~ FoodAction
calculate snowball Action

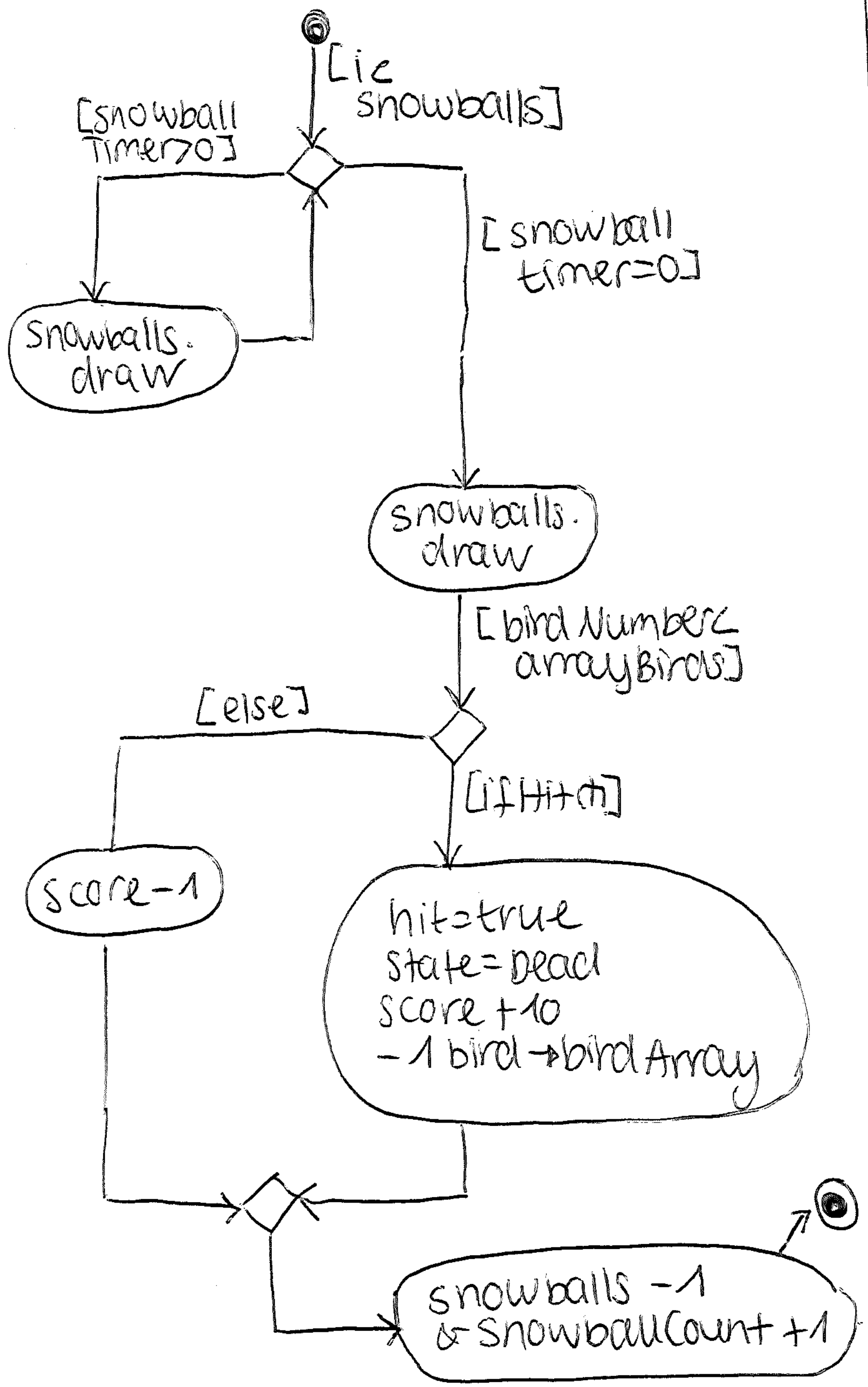

showEndscreen
[loser→score<0]
Game Over! You lose!
no score
[winner→score≥0]
Game over Your Score: ""
show Highscore List


moveBirds
[i<arrayBirds & i2<arrayFood]
[else]
[food nearby()]
change path
arrayBirds. move
arrayBirds. draw

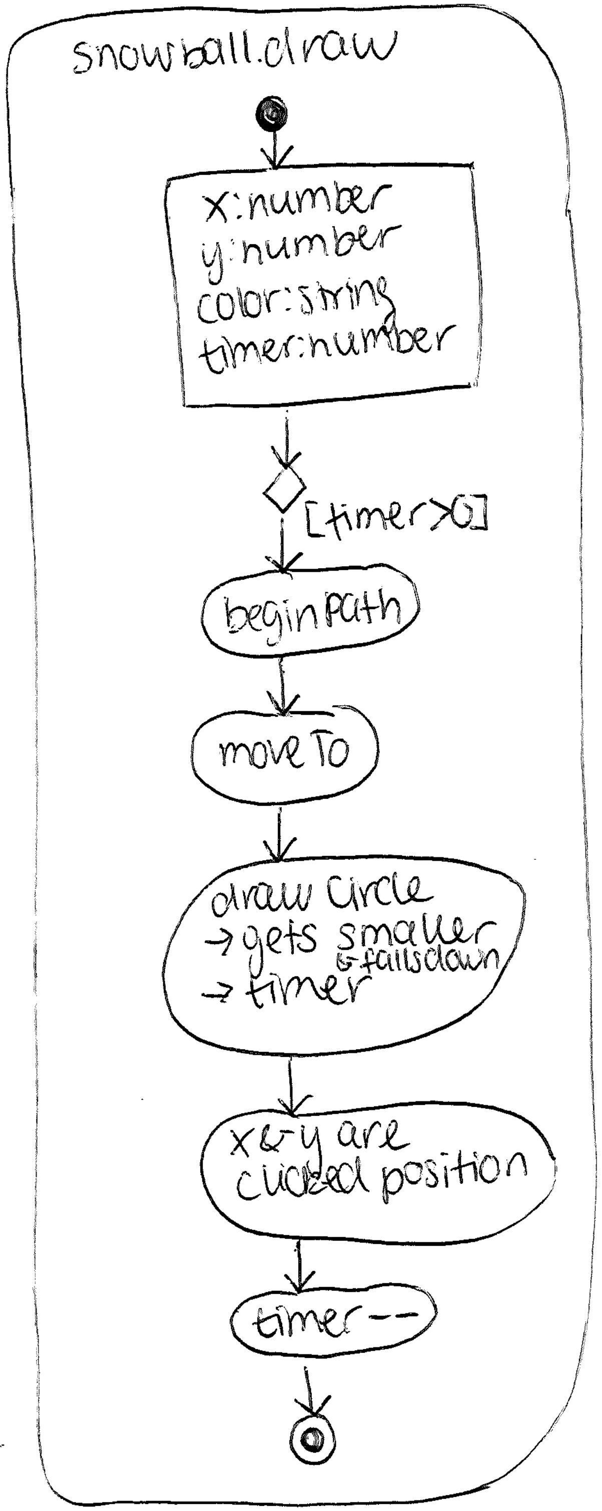

moveSnowflakes
[i<arraySnowflakes]
arraySnowflakes. move
arraySnowflakes. draw()


calculateFoodAction
[i<arrayFood]
[else]
[food landed & position>y=400 || lifetime=0]
food. draw ()
food-1
[lifetime=0]
for bird count < arrayBirds
[Bird is state feeding|picking]
reset velocity & state=Alive

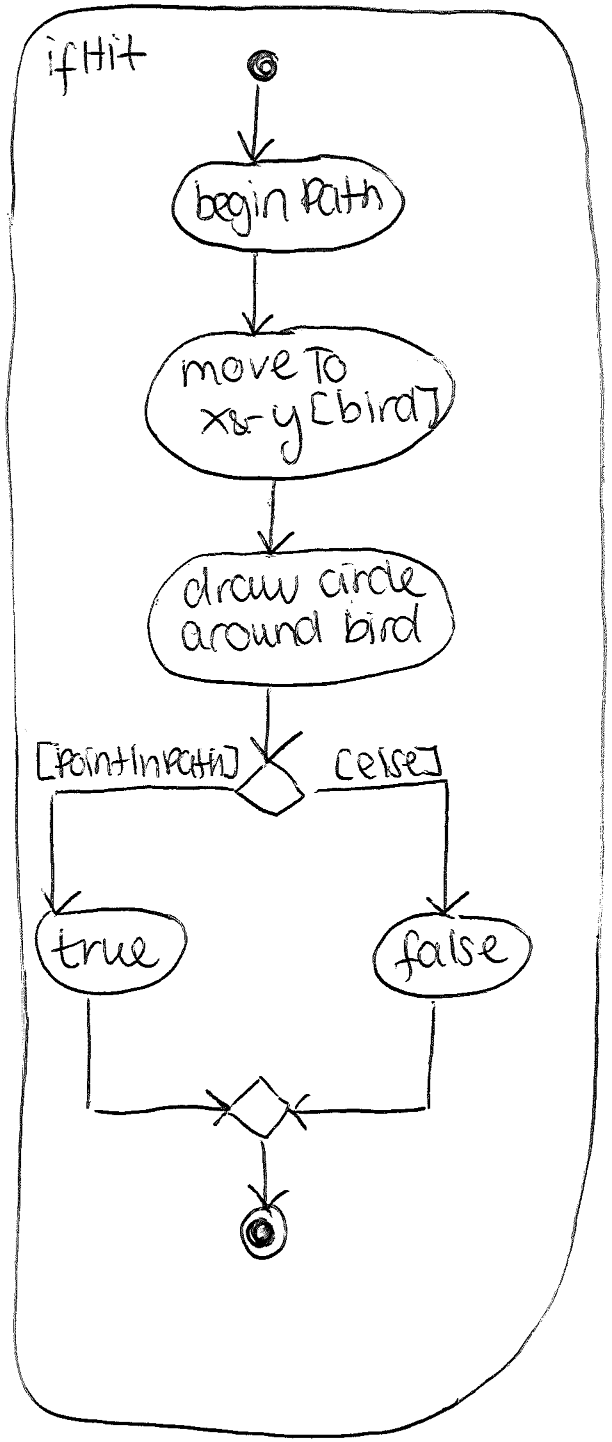
calculateSnowballAction

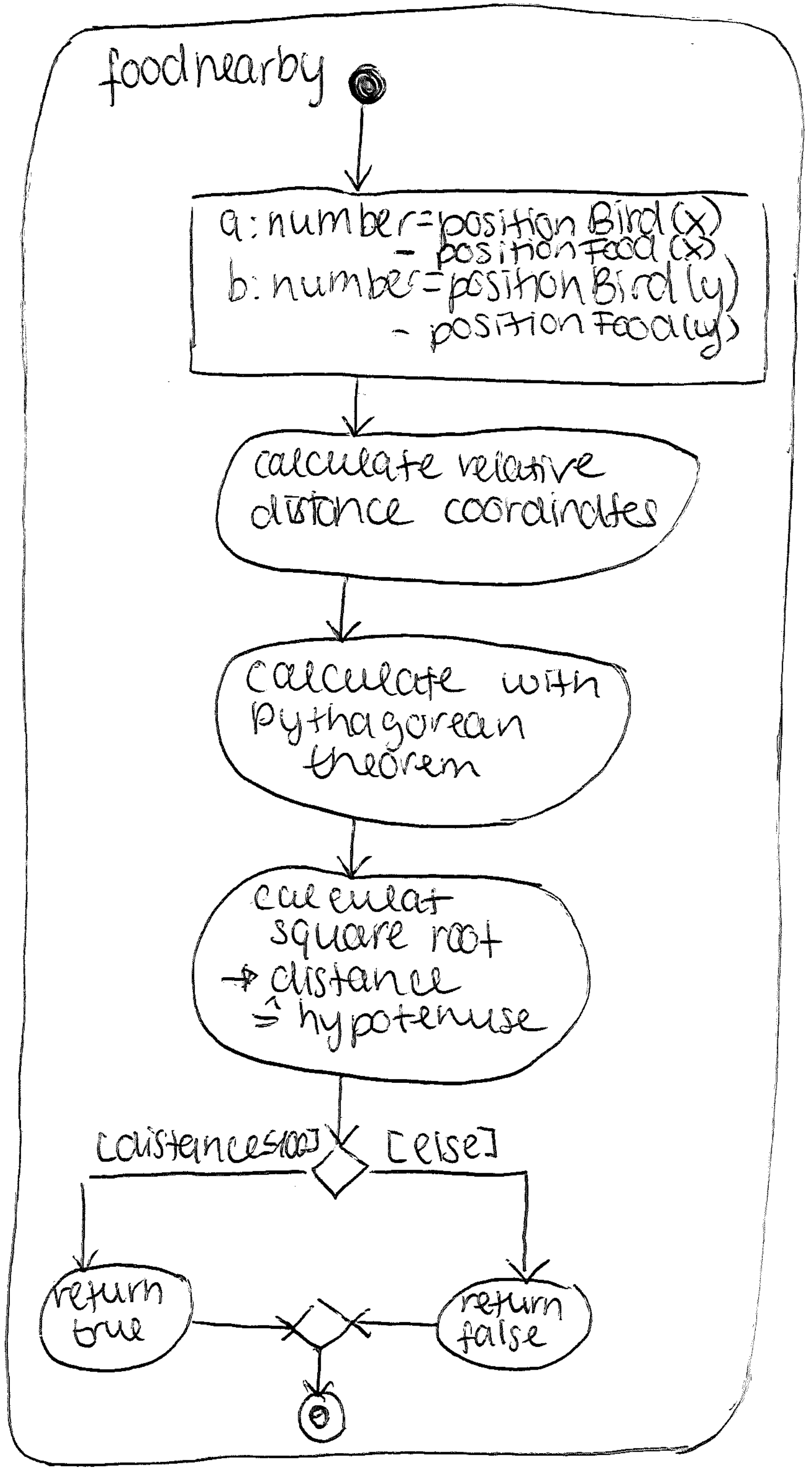


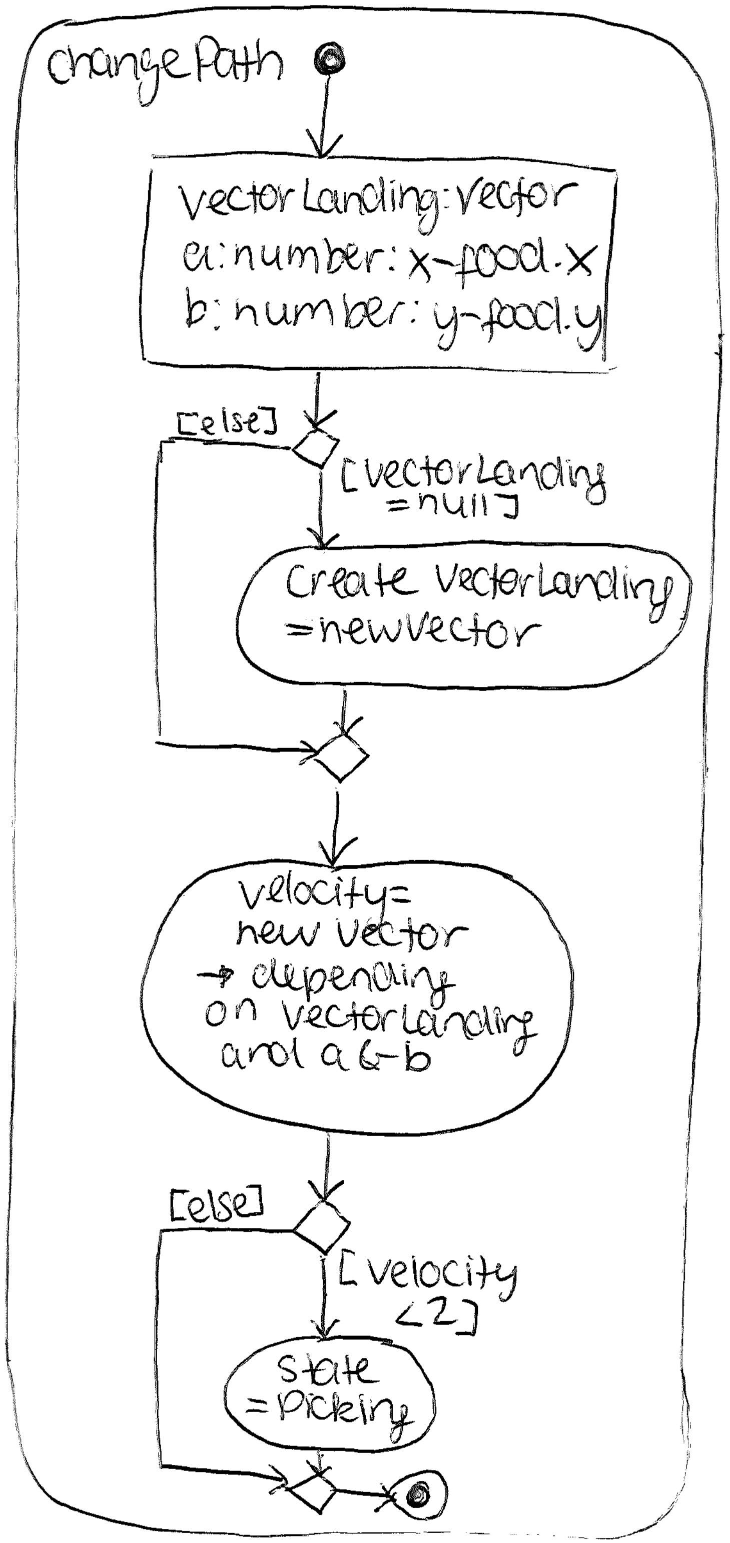
snowball.draw

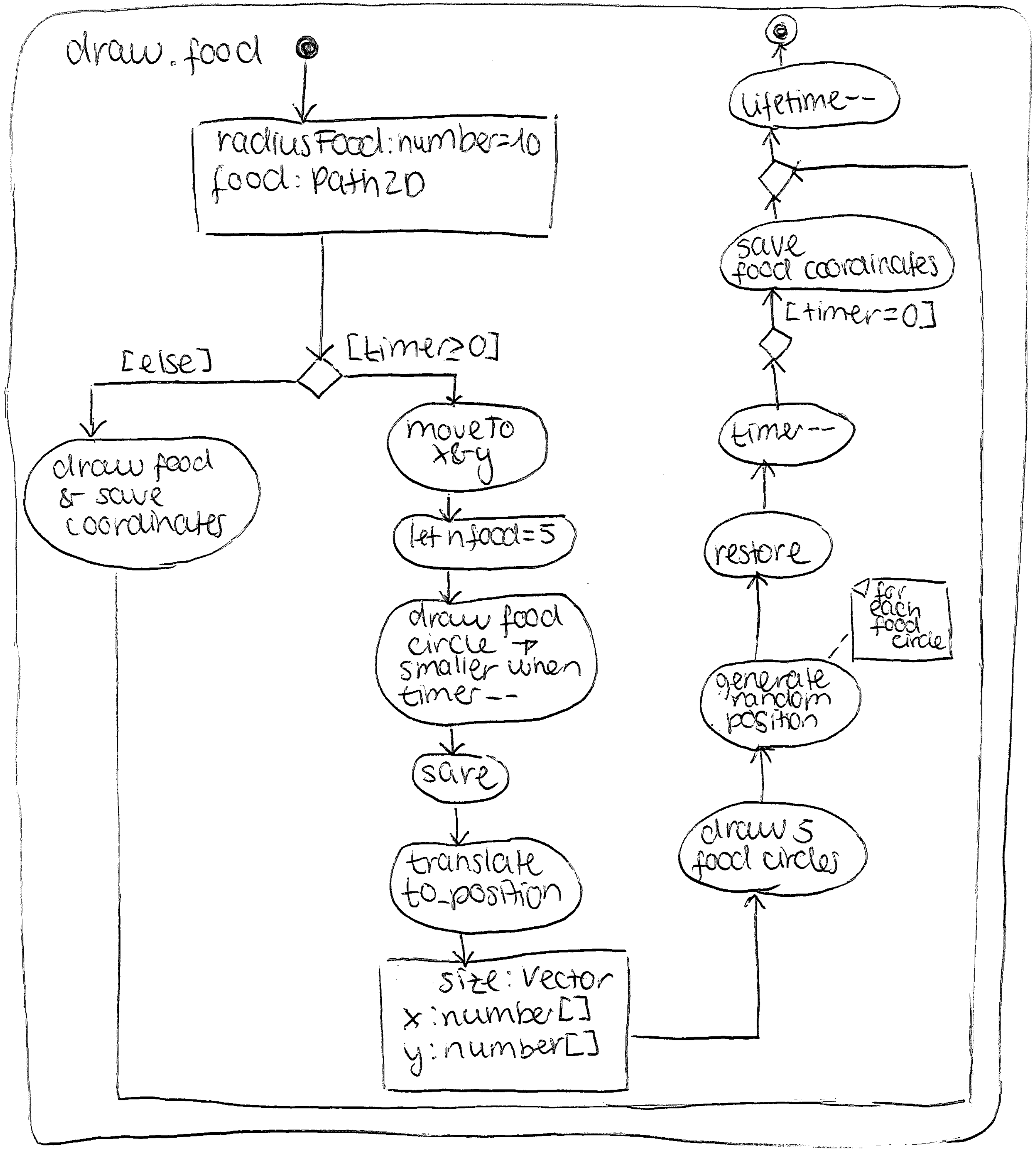


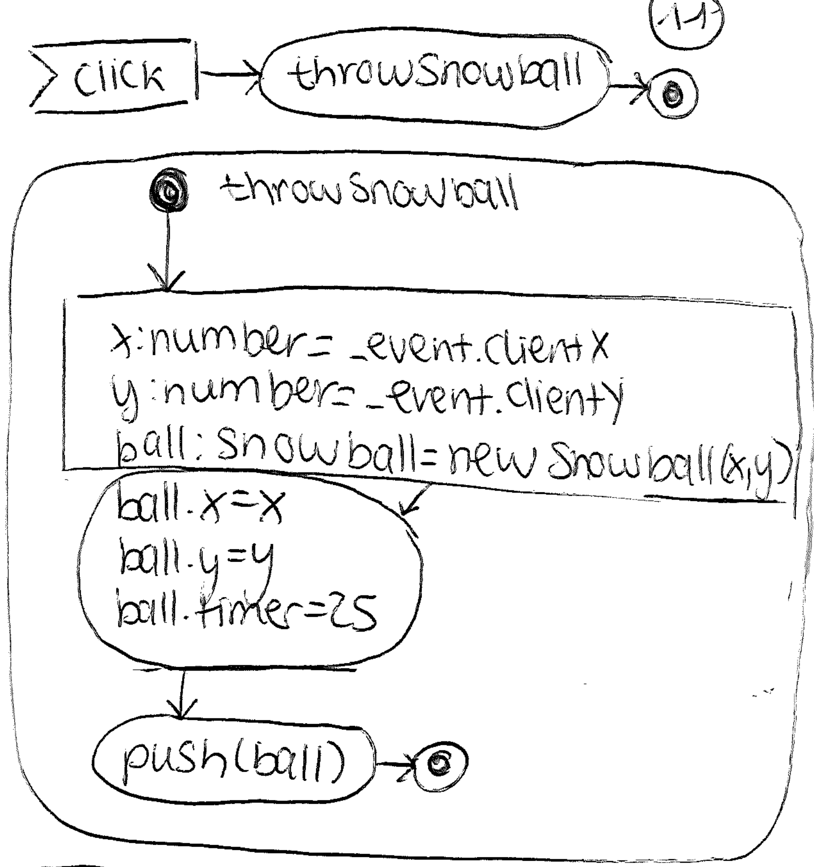
ifHit

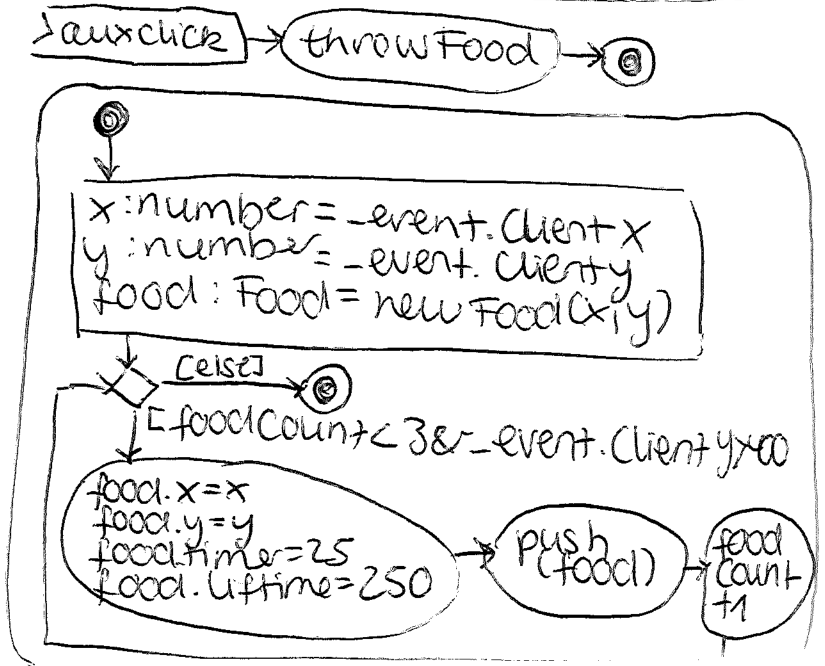



foodnearby
a: number = positionBird(x) - positionFood(x)
b: number = positionBird(y) - positionFood(y)
calculate relative distance coordinates
calculate with pythagorean theorem
calculat square root → distance ≙ hypotenuse
[distance≤100]
[else]
return true
return false


changePath
vectorLanding: vector
a: number: x-food.x
b: number: y-food.y
[else]
[vectorLanding =null]
Create vectorLanding =newVector
velocity= new Vector → depending on vectorLanding and a&b
[else]
[velocity <2]
State =Picking


draw.food
radiusFood: number=10
food: Path2D
[else]
[timer≥0]
draw food & save coordinates
moveTo x&y
let nfood=5
draw food circle → smaller when timer--
save
translate to_position
size: Vector
x: number[]
y: number[]
draw 5 food circles
generate random position
for each food circle
restore
timer--
[timer=0]
save food coordinates
lifetime--


click
throwSnowball
throwSnowball
x: number=_event.clientX
y: number=_event.clientY
ball: Snowball=new Snowball(x,y)
ball.x=x
ball.y=y
ball.timer=25
push(ball)


auxclick
throwFood
x: number=_event.ClientX
y: number=_event.ClientY
food: Food=new Food(x,y)
[else]
[foodCount<3&_event.ClientY>400]
food.x=x
food.y=y
food.timer=25
food.lifetime=250
push (food)
foodCount +1