Nutzererlebnis

Dieses Spiel ist eines für Jung und Alt. Man braucht keine bestimmten Skills, es ist einfach zu verstehen und es macht Spaß sich mit anderen Spielern zu messen. Erzielt man einen hohen Score bzw. findet sich in der Highscore-Liste weit oben, hat der Spieler ein positives Erlebnis. Allgemein sich einfach Mal ein bisschen ablenken und hat dabei Spaß ein lustiges Game zu spielen.

Festlegung der Plattform

Für das Spiel wurde ein Grundkonzept entwickelt. Bei der Überlegung wie man dieses umsetzen möchte, war es die einfachste Lösung den PC als Plattform zu wählen. Dieser hat mehr Eingabemöglichkeiten, die während dem Spiel benutzt werden können. Deswegen mussten keine extra Buttons oder Ähnliches erstellt werden, sondern es können einfach die Maustasten für das Spielen der Anwendung verwendet werden.

Informationen für den Nutzer

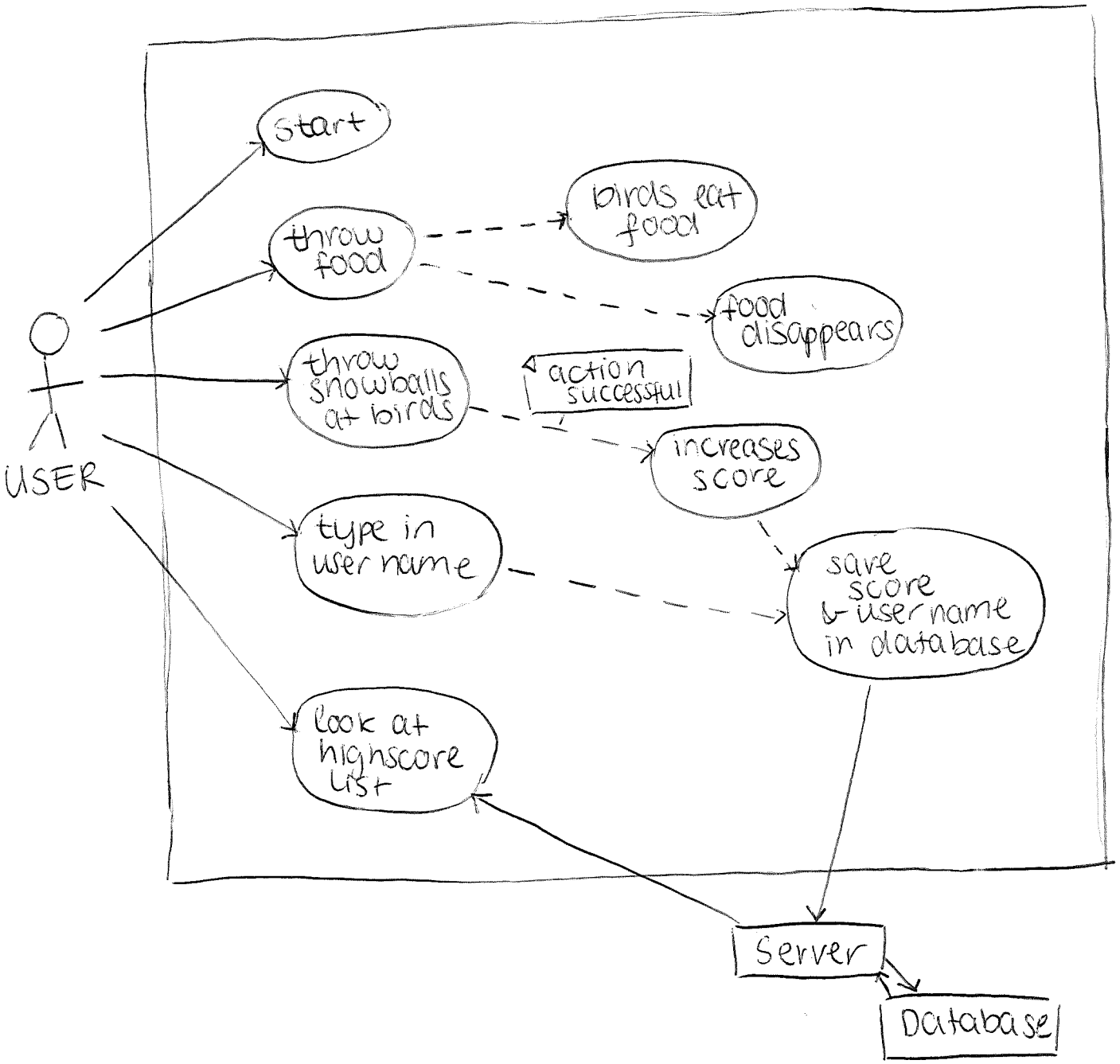
Vor und nach Ende des Spiels kann der Nutzer die Highscore-Liste einsehen. Man kann sich mit anderen Spielern messen und es ist ersichtlich welchen Platz man in der Rangliste einnimmt. An dieser Stelle kann er das Spiel auch (neu) starten. Während des Spiels kann er seinen Score, die Anzahl von zur Verfügung gestellten Futteranzahl und die Anzahl noch verfügbarer Schneebälle.

Allgemeine Nutzung des Spiels

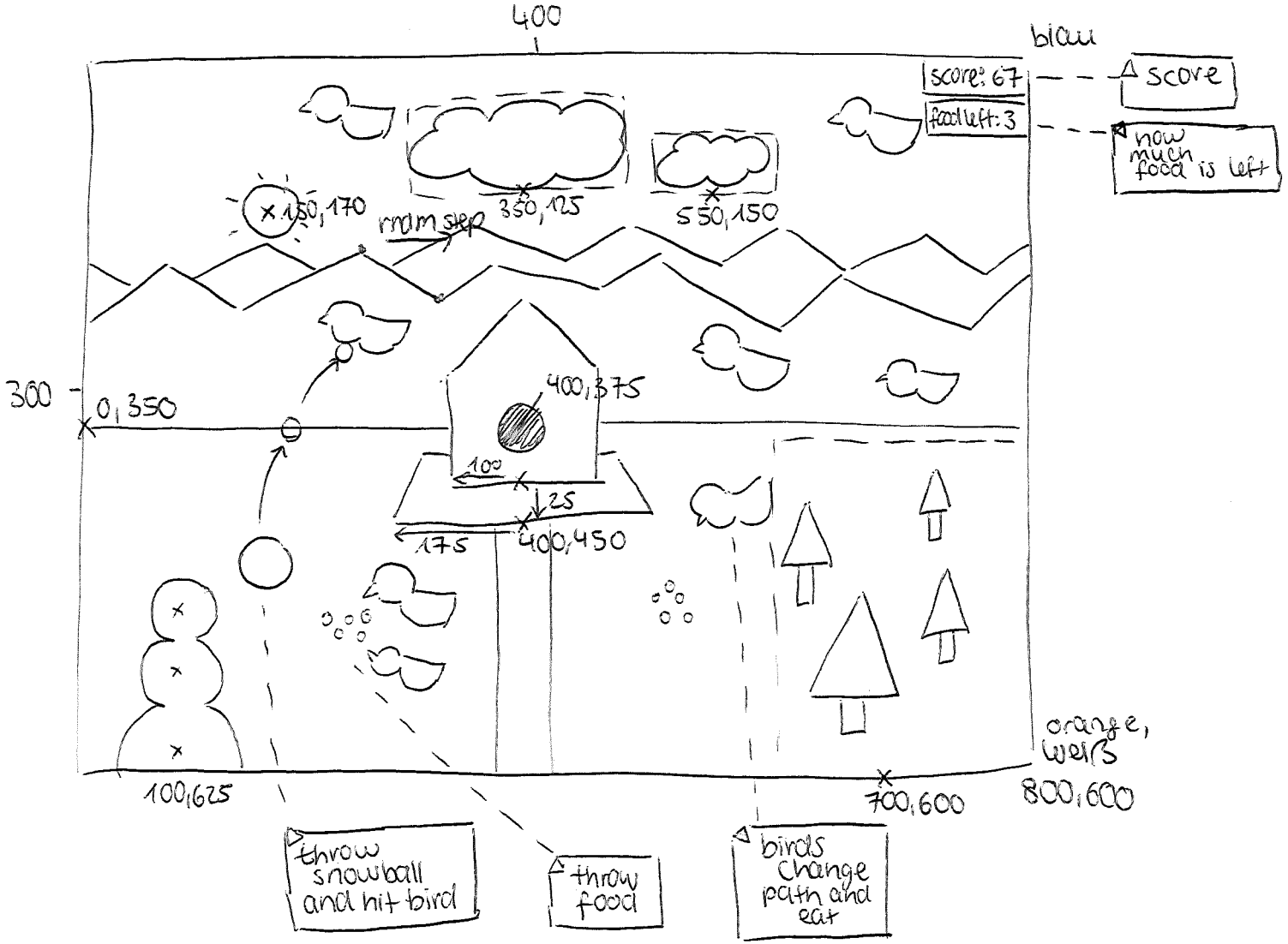
Der Spieler hat die Möglichkeit durch das Werfen eines Schneeballs Vögel zu treffen. Dies ist möglich mit einem Klick der linken Maustaste. Bei erfolgreicher Aktion verschwinden diese und 10 Punkte wandern auf den Score des Users. Bei Misserfolg wird ein Punkt abgezogen. Ist der Score nicht negativ wandert dieser auf eine Highscore-Liste auf dem alle erfolgreichen Spieler eingetragen sind.

Um Vögel anzulocken und damit einen höheren Score zu erlangen, kann man Futter werfen. Dies passiert durch Klick auf die mittlere oder rechte Maustaste. Insgesamt hat der Spieler drei Futterwürfe zur Verfügung. Das Futterwerfen ist auch nur im unteren Bereich der Canvas möglich.

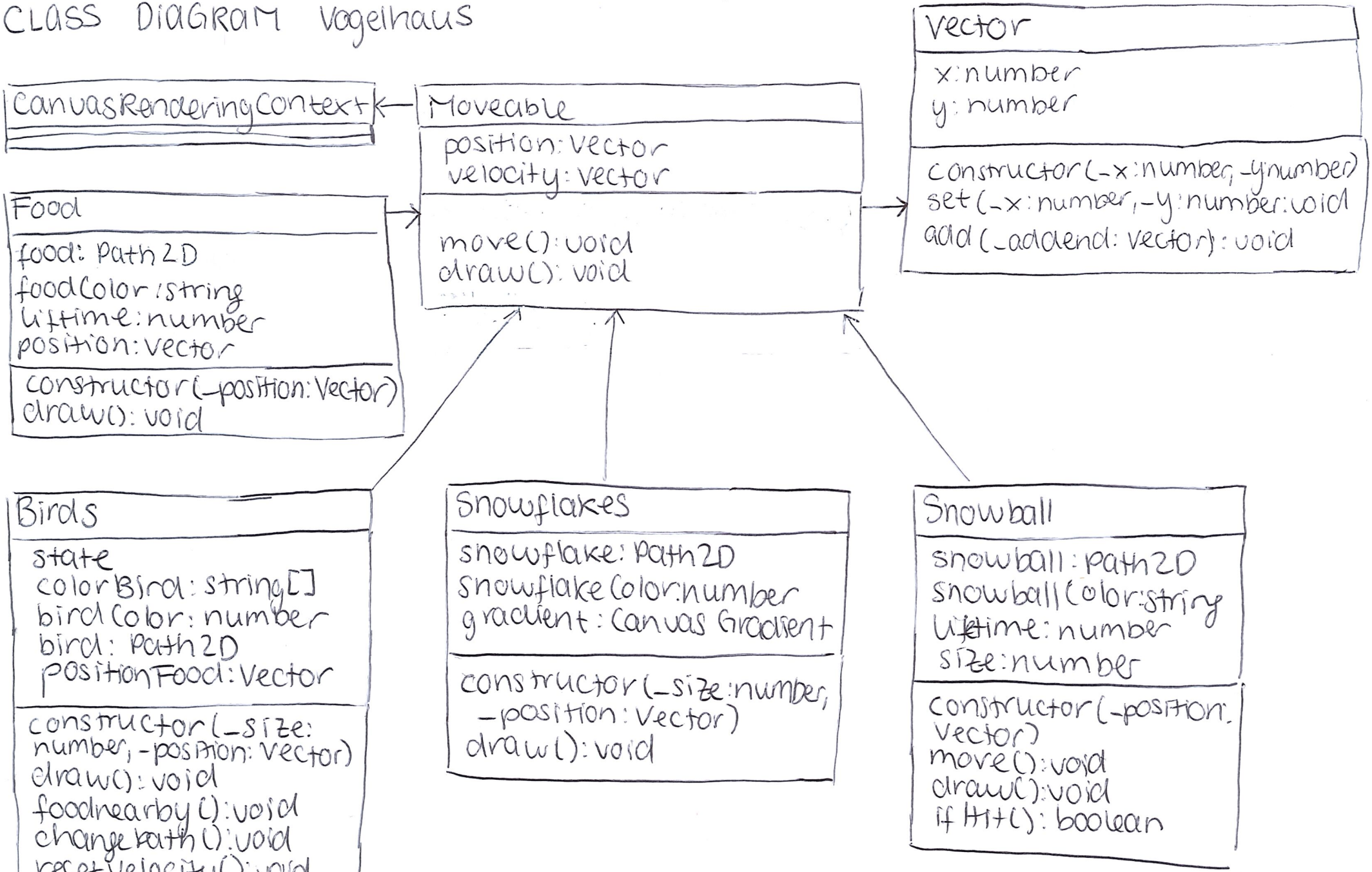
USE CASE DIAGRAM Vogelhaus



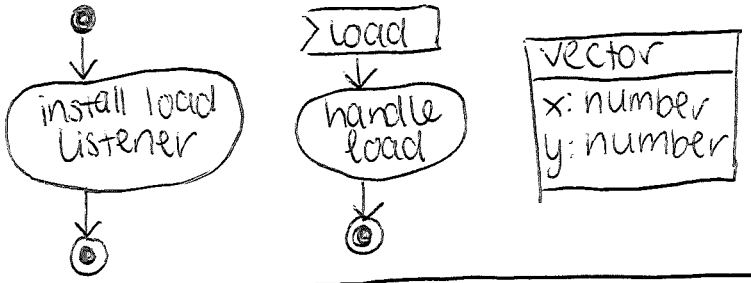
USER INTERFACE Vogelhaus



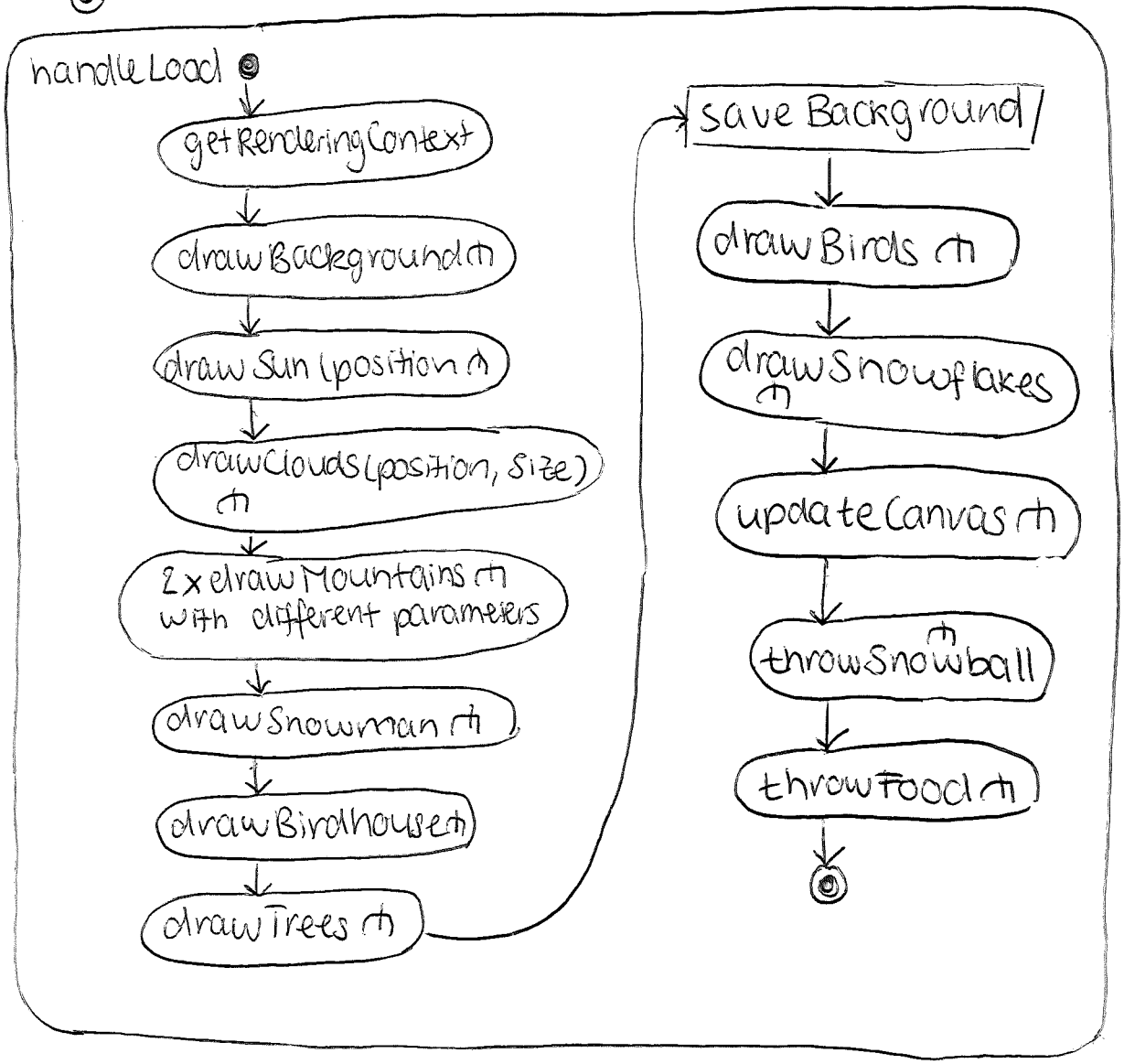
CLASS DIAGRAM Vogelhaus

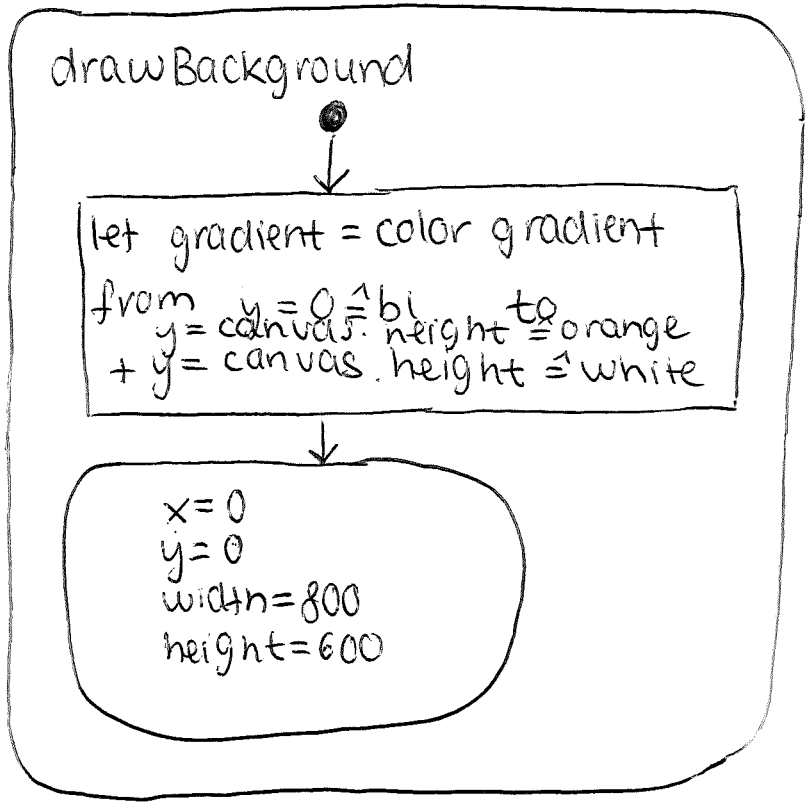


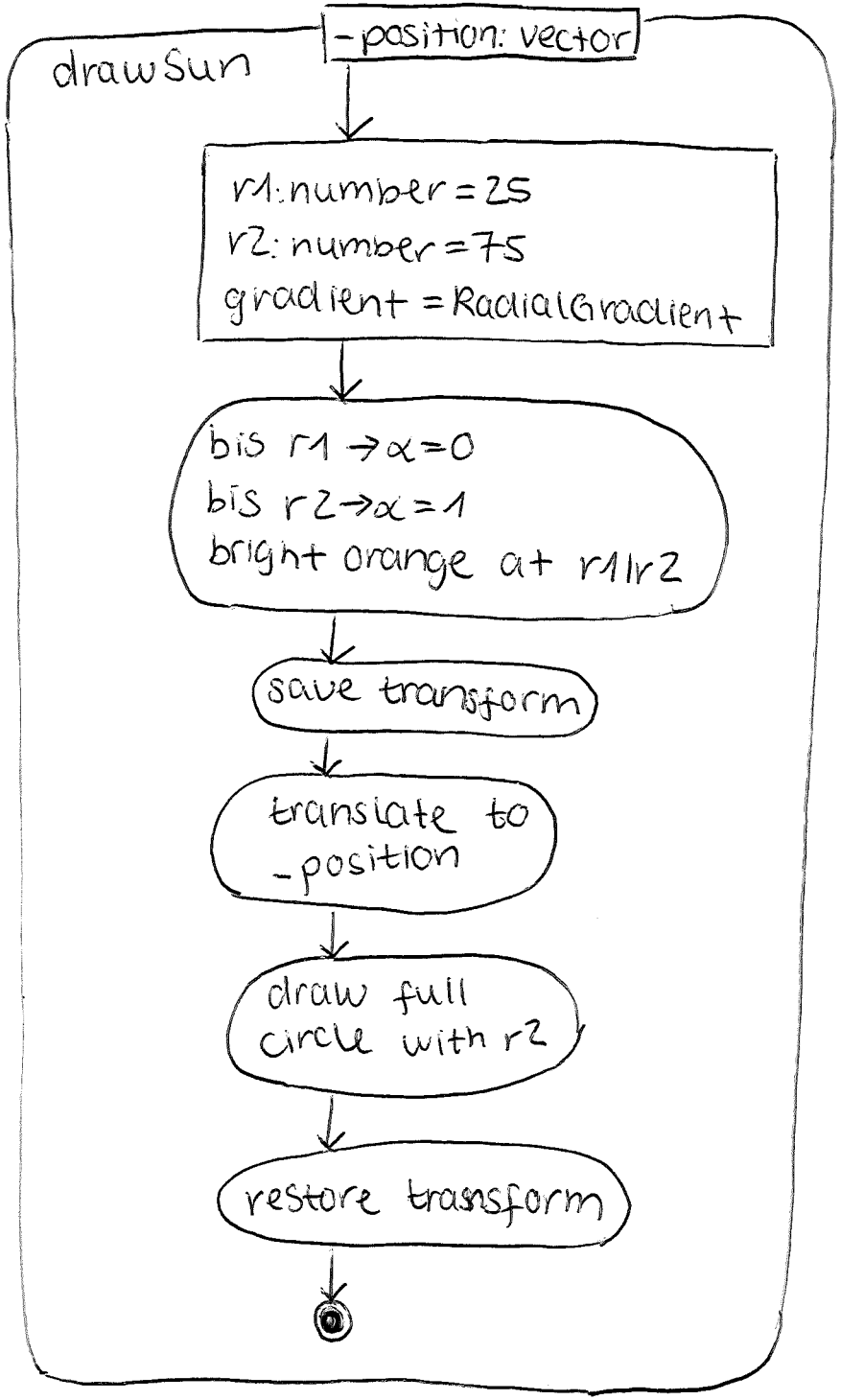
ACTIVITY DIAGRAMS Vogelhaus

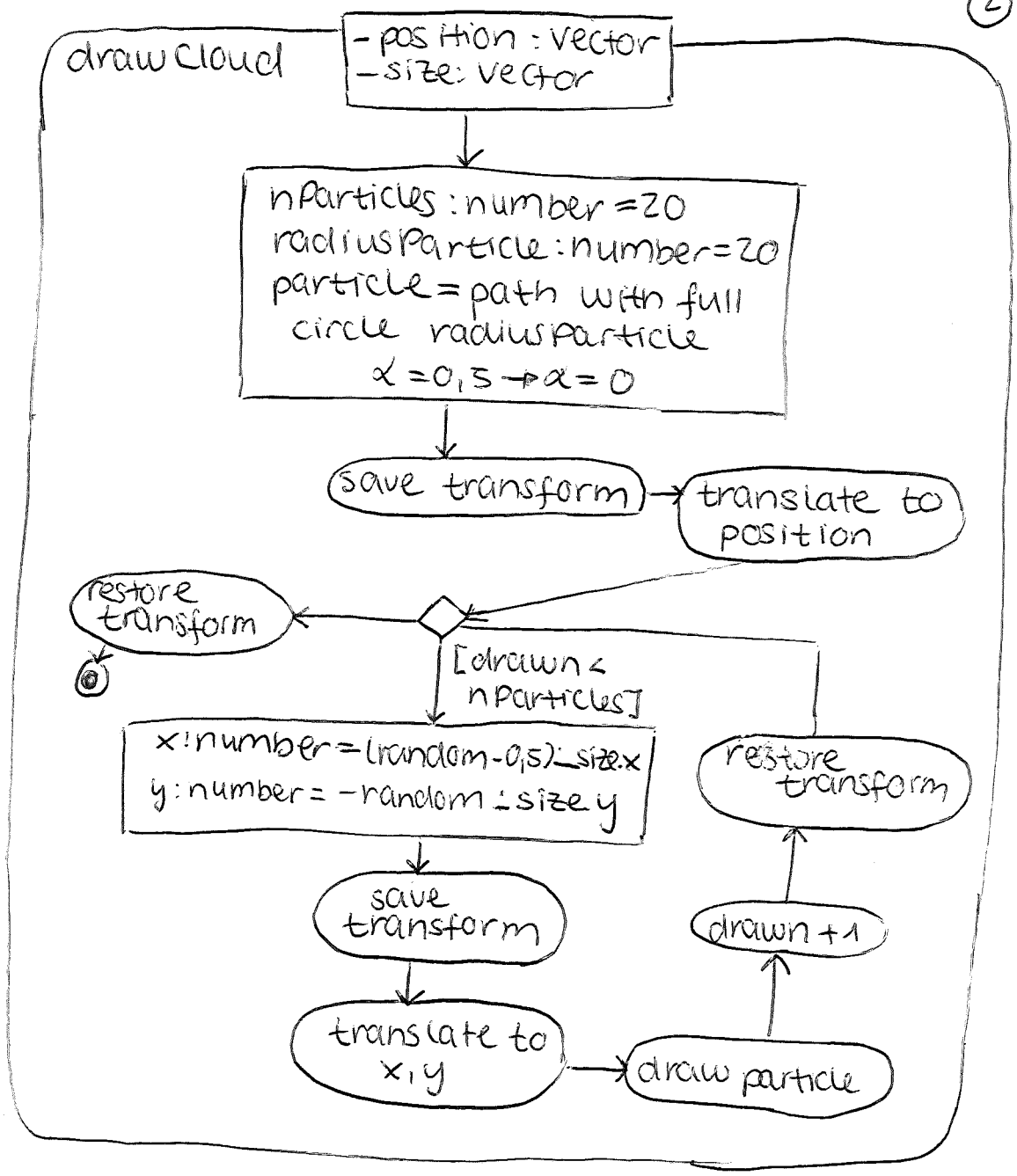


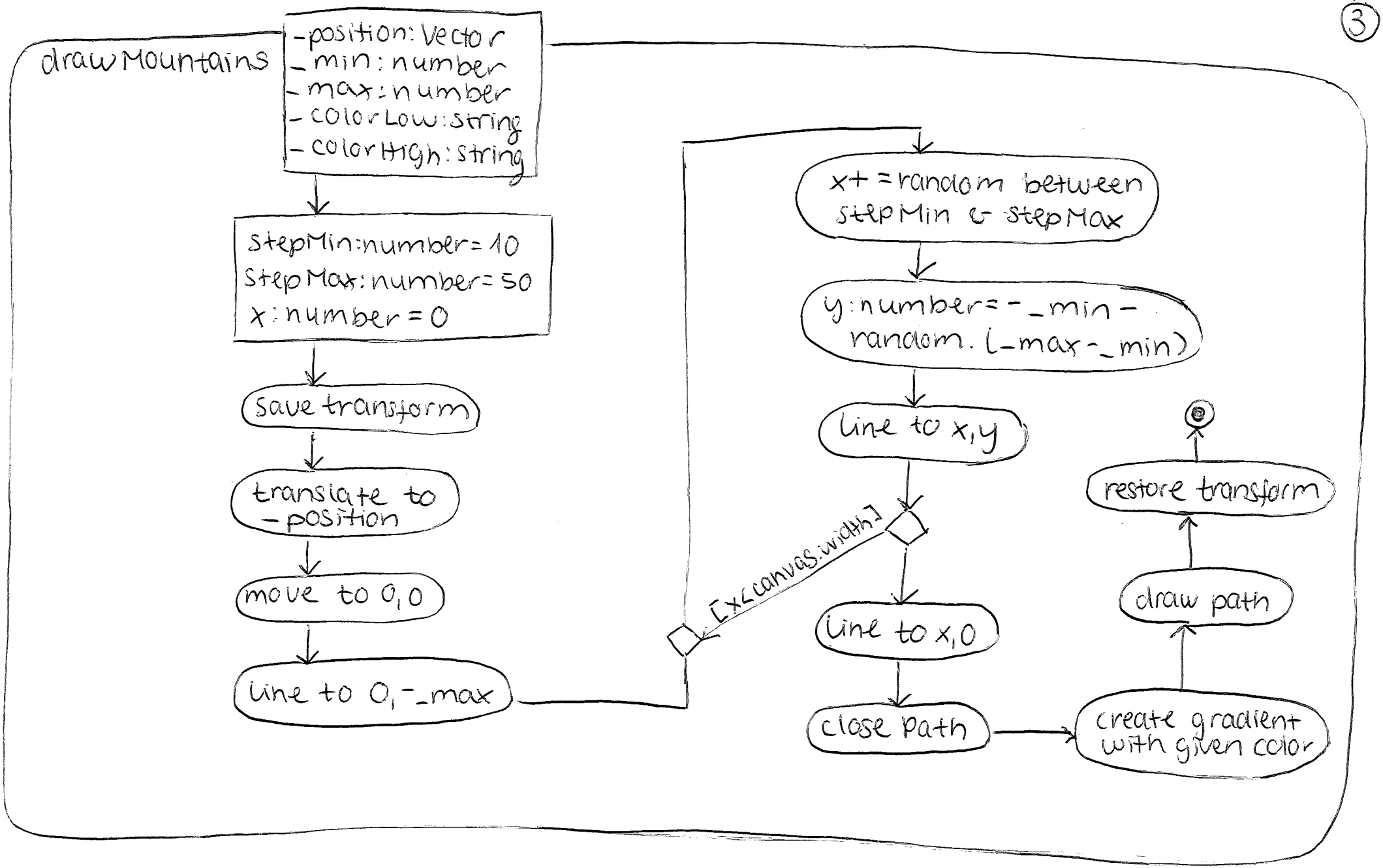
vector
x: number y: number

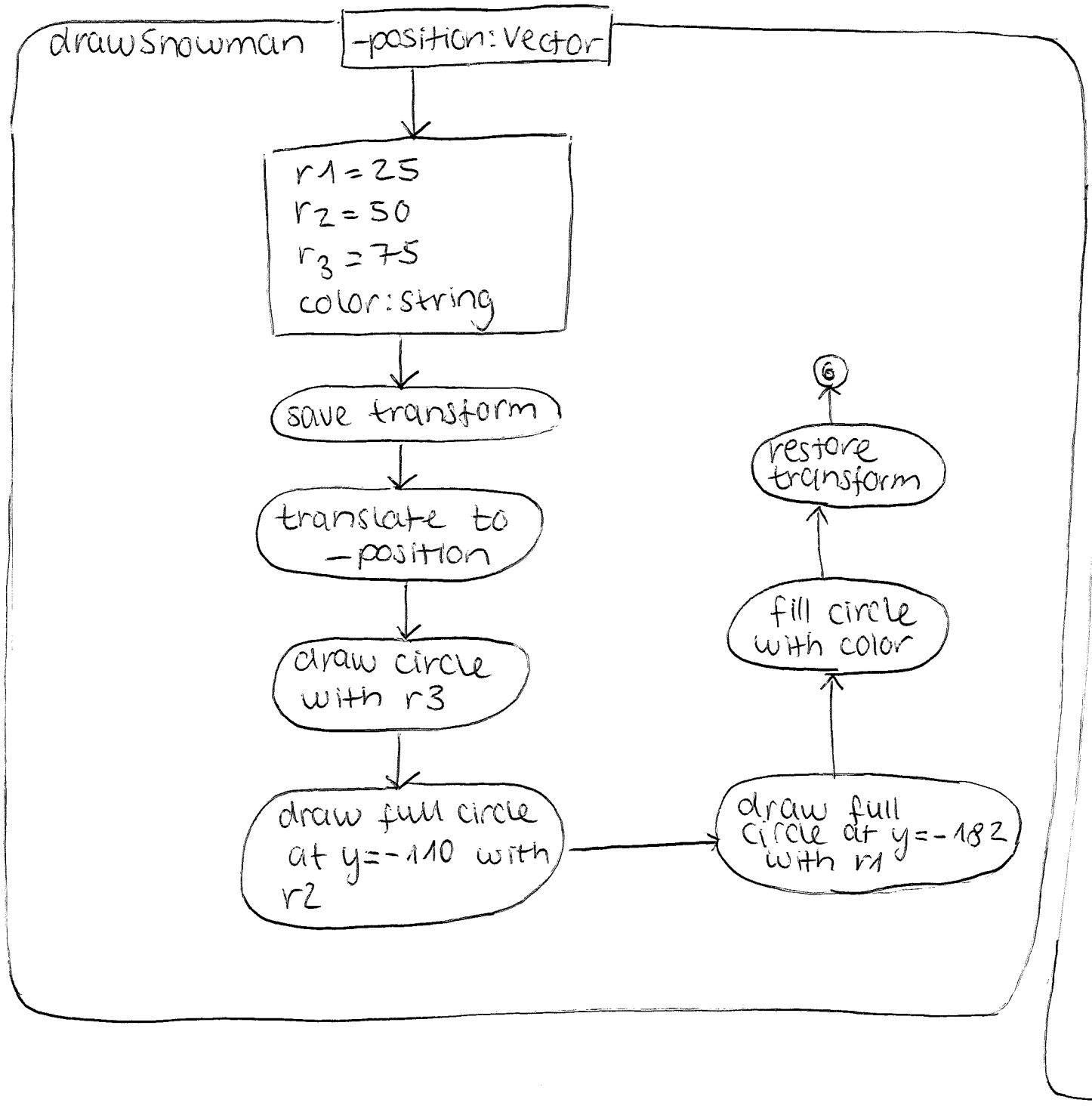


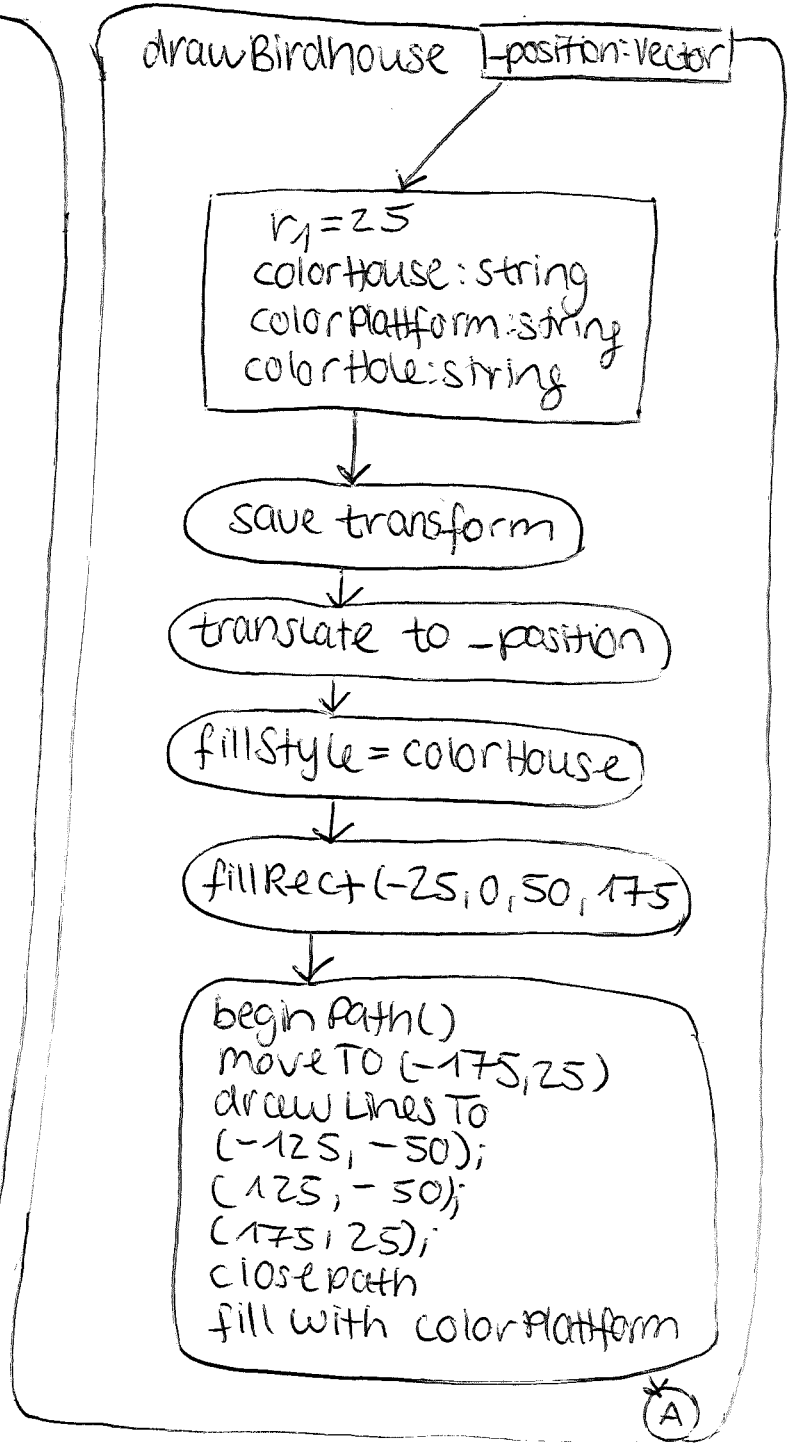



drawSun
-position: vector
r1:number=25
r2: number=75
gradient=RadialGradient
bis r1 →α=0
bis r2→α=1
bright orange at r1|r2
save transform
translate to -position
draw full circle with r2
restore transform

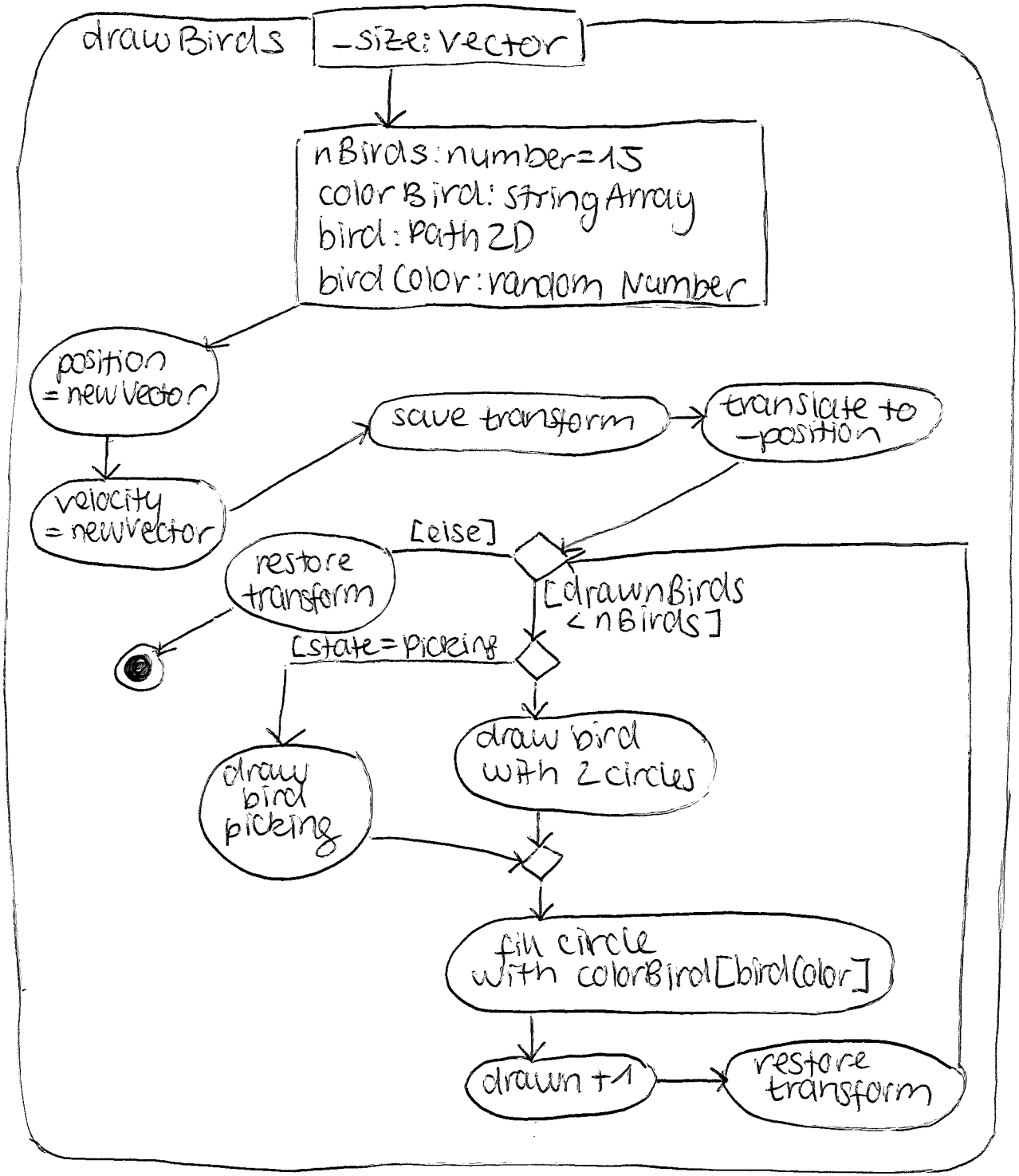

drawCloud
-position: vector
-size: vector
nParticles: number=20
radiusParticle: number=20
particle=path with full circle radiusParticle
α=0,5→α=0
save transform
translate to position
restore transform
[drawn< nParticles]
x:number=(random-0,5)*size.x
y:number=-random*size.y
save transform
translate to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

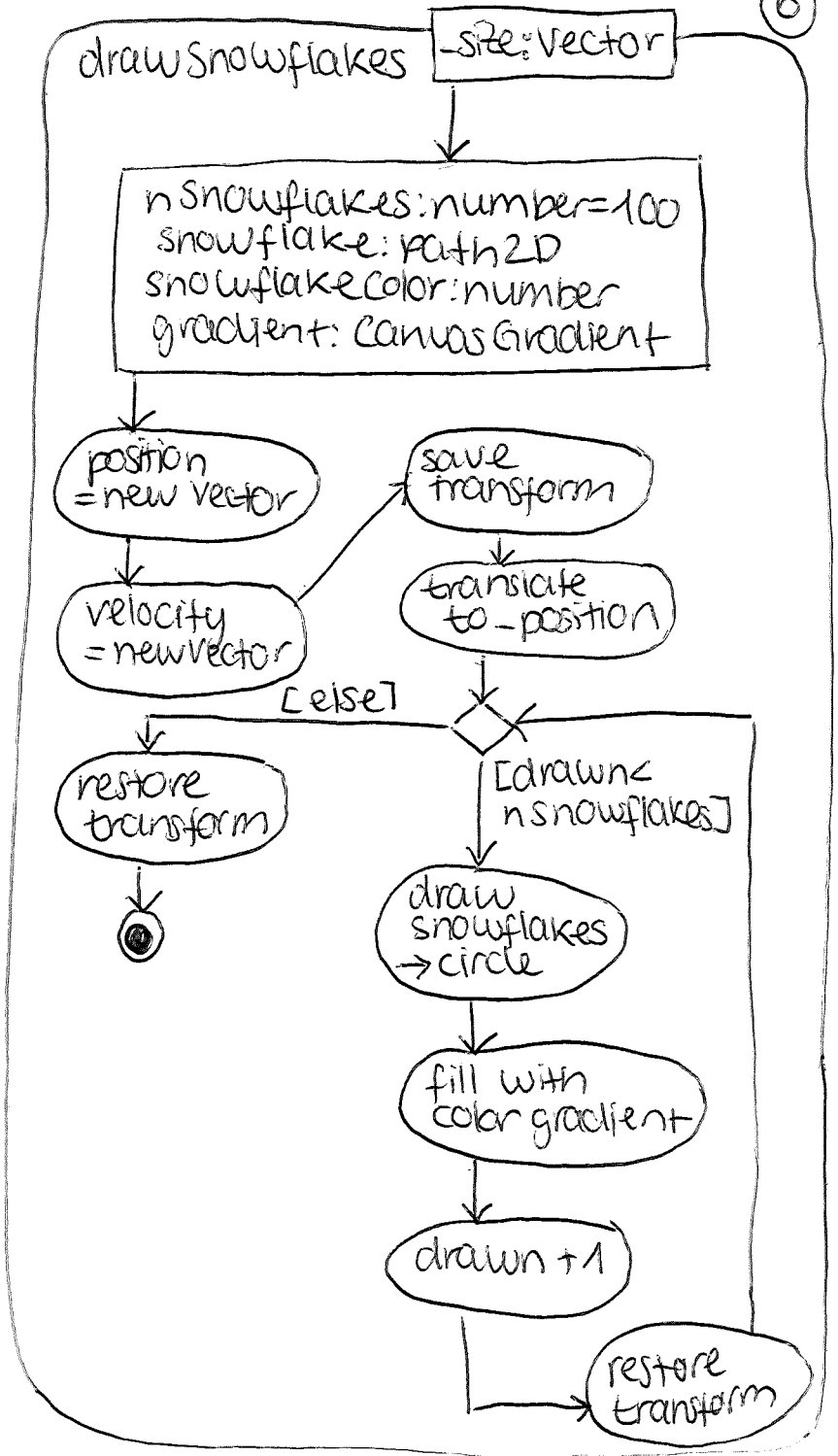

drawMountains
-position: Vector
-_min: number
-max: number
-colorLow: string
-colorHigh: string
stepMin:number=10
stepMax:number=50
x:number=0
save transform
translate to -position
move to 0,0
line to 0,-_max
[x<canvas.width]
x+=random between stepMin & stepMax
y:number=-_min-random.(_max-_min)
line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

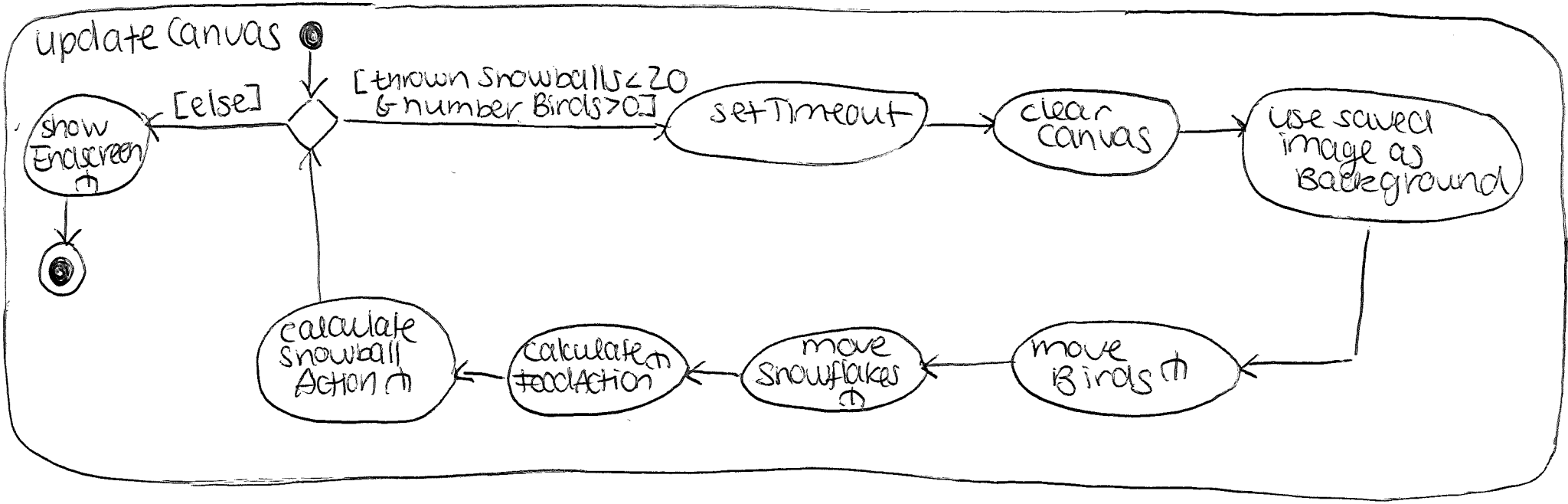

drawSnowman
-position: Vector
r1 = 25
r2 = 50
r3 = 75
color: string
save transform
translate to -position
draw circle with r3
draw full circle at y=-110 with r2
draw full circle at y=-182 with r1
fill circle with color
restore transform
6

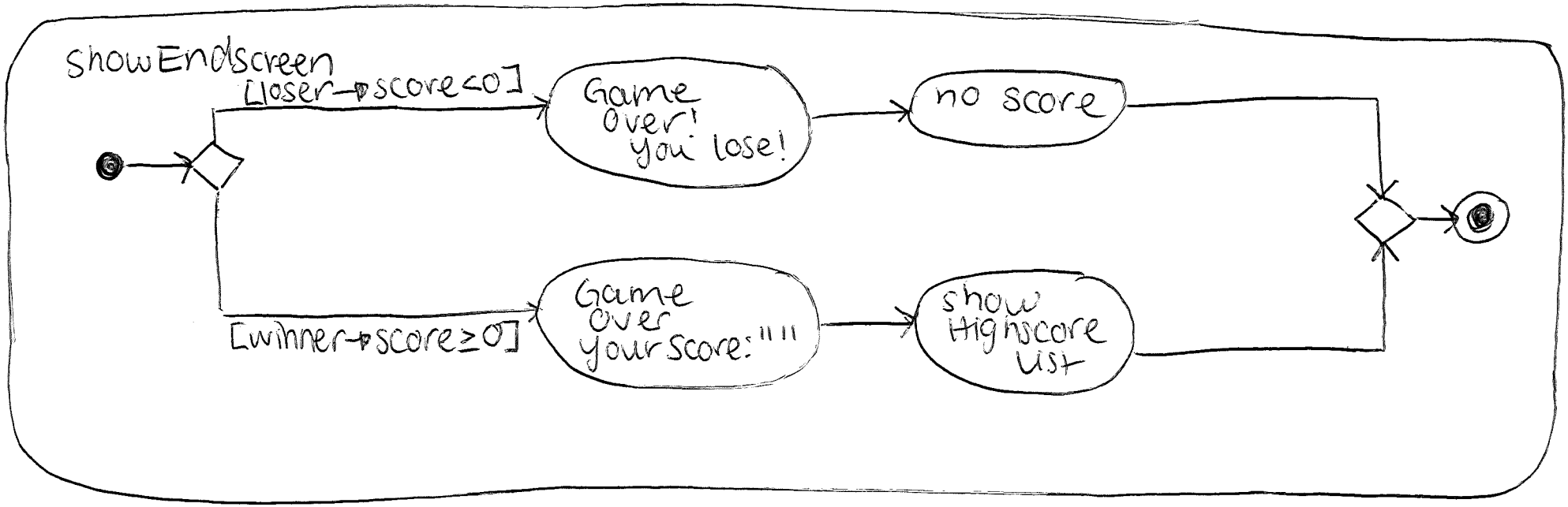

drawBirdhouse
-position: Vector
r1 = 25
colorHouse: string
colorPlatform: string
colorHole: string
save transform
translate to -position
fillStyle = colorHouse
fillRect(-25, 0, 50, 175
beginPath()
moveTo(-175, 25)
drawLinesTo
(-125, -50);
(125, -50);
(175, 25);
closePath
fill with colorPlatform
A

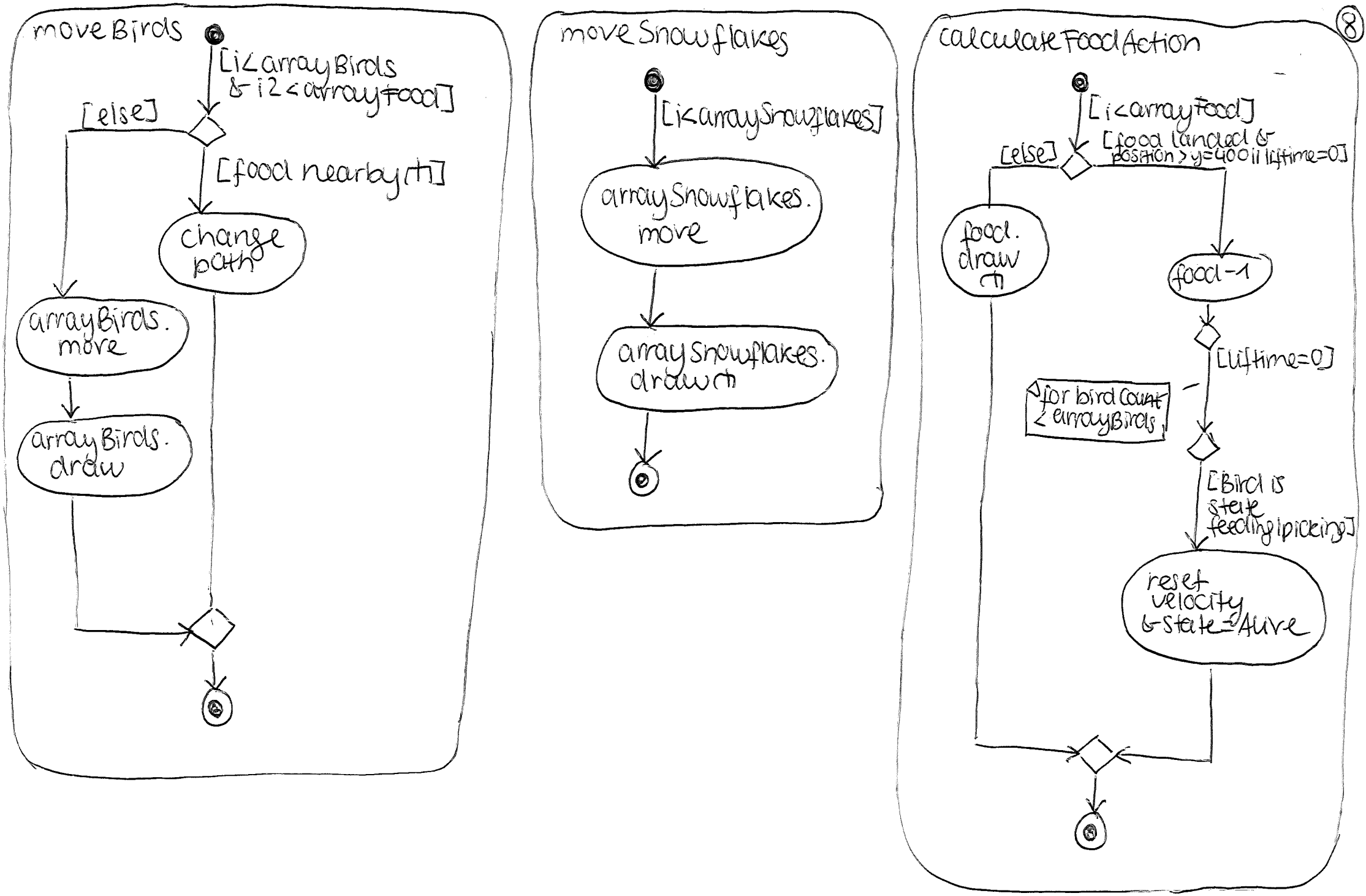

A
beginPath()
moveTo(-100,0)
drawLinesTo
(-100,-150)
(0,-225)
(100,-150)
(100, 0)
closePath()
fill with colorHouse
draw full circle with r1 at y=-75
fill circle with colorHole
restore transform
5
drawTrees
-position: Vector
-size: Vector
nTrees: number=4
colorTrunk: string
colorCrown: string
save transform
translate to -position
restore transform
[drawn<nTrees]
save
scale: number
calculate scale depending on y
fillRec
drawCrown
fill with color
restore
drawn +1
restore transform

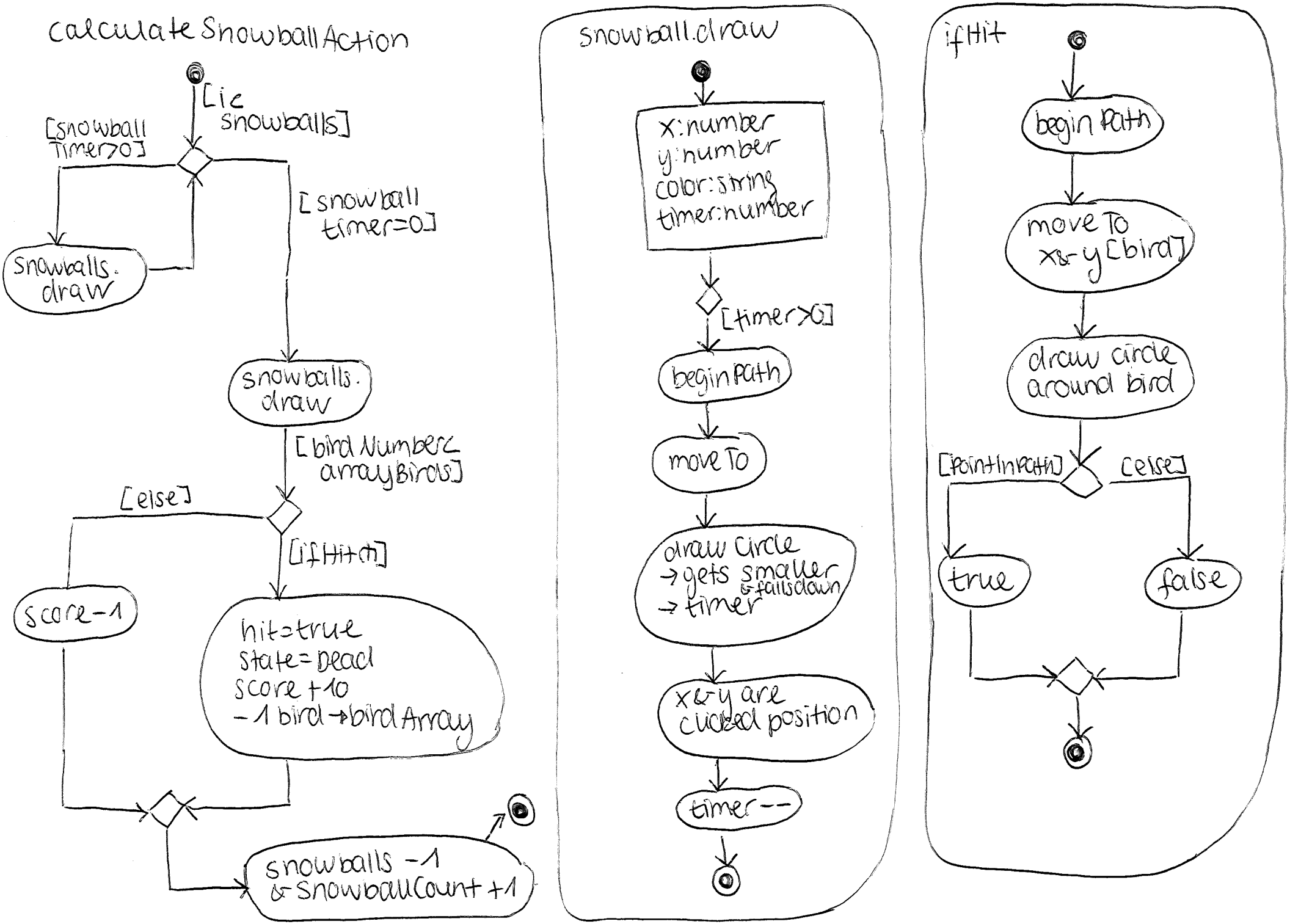

drawBirds
_size:vector
nBirds:number=15
colorBird:stringArray
bird:Path2D
birdColor:randomNumber
position =newVector
velocity =newVector
save transform
translate to -position
[else]
restore transform
[drawnBirds <nBirds]
[state=picking
draw bird picking
draw bird with 2 circles
fill circle with colorBird[birdColor]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

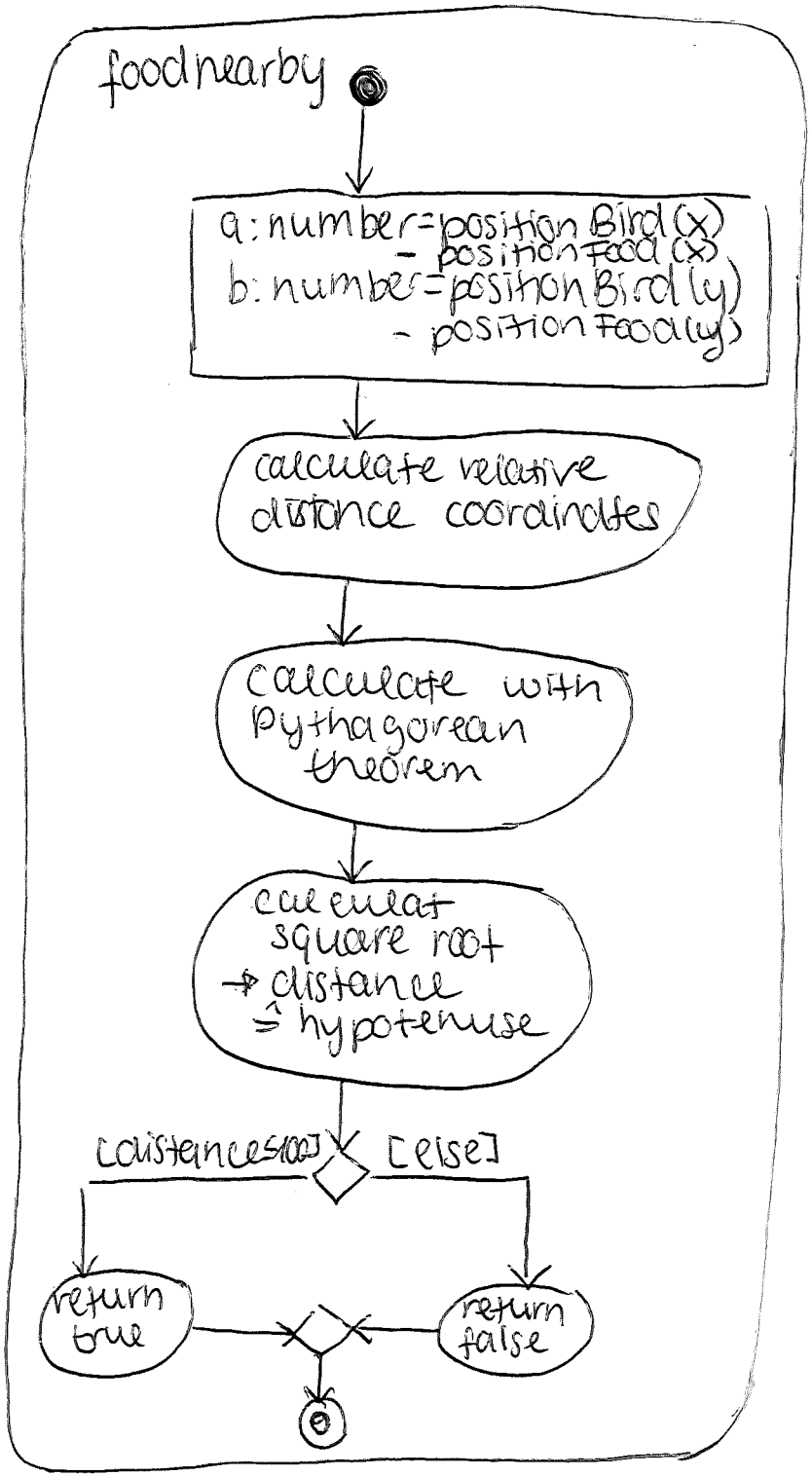

drawSnowflakes
_size:vector
nSnowflakes:number=100
snowflake:Path2D
snowflakeColor:number
gradient:CanvasGradient
position =new vector
velocity =newVector
save transform
translate to_position
[else]
restore transform
[drawn< nsnowflakes]
draw snowflakes →circle
fill with color gradient
drawn+1
restore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

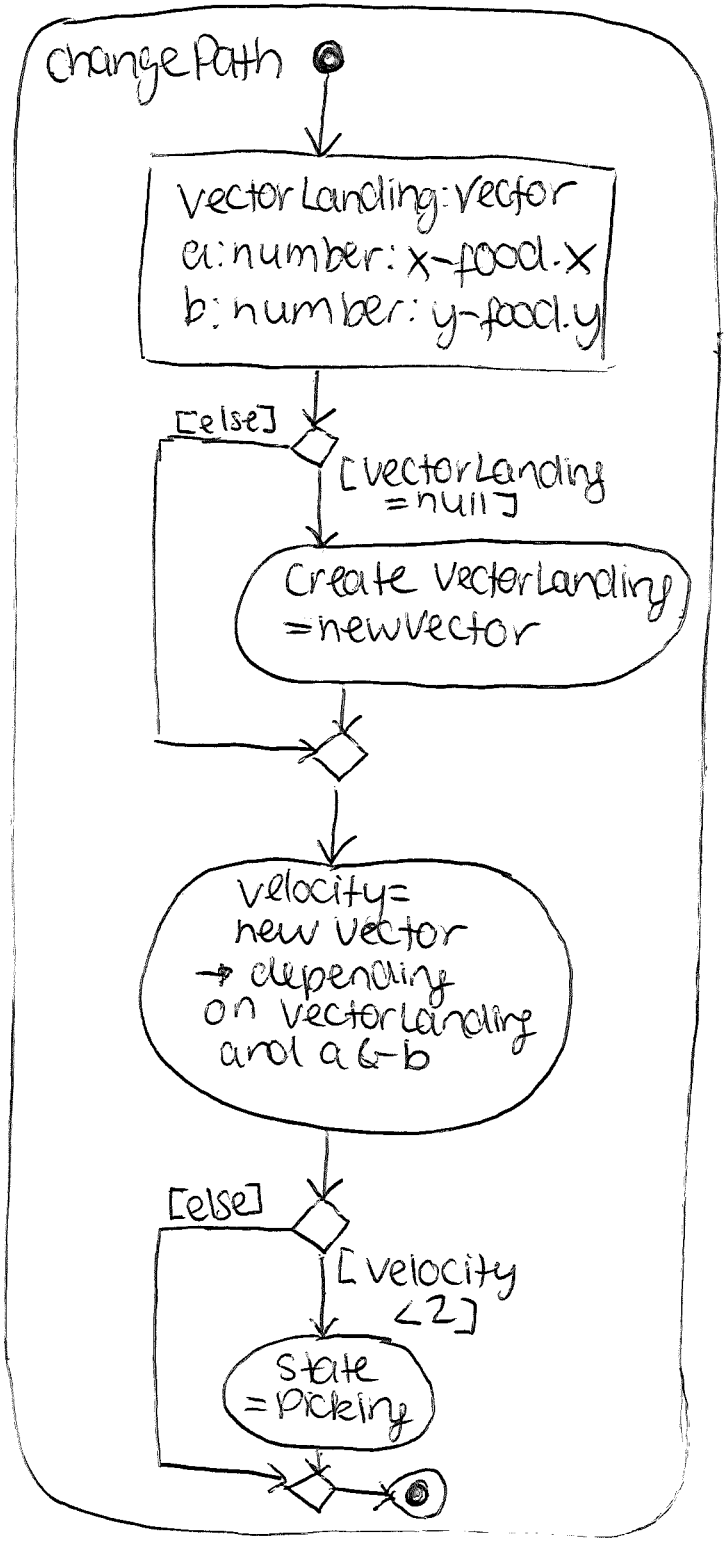

updateCanvas
[else]
[thrownSnowballs<20 & numberBirds>0]
showEndscreen
setTimeout
clearCanvas
use saved image as Background
move Birds
move Snowflakes
calculate foodAction
calculate snowballAction

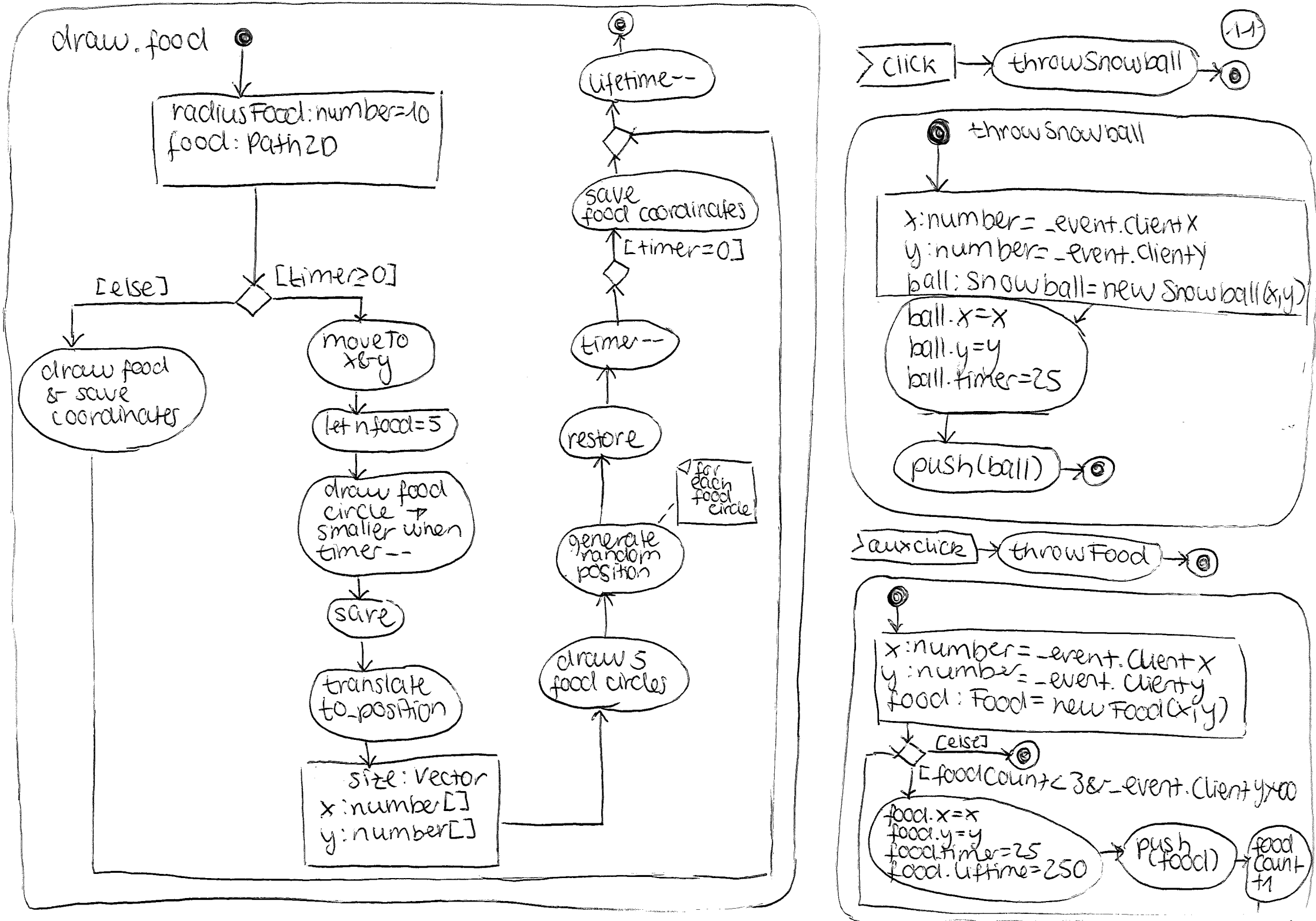

showEndscreen
[loser→score<0]
Game over! you lose!
no score
[winner→score≥0]
Game over your score: !!!!
show Highscore list


moveBirds
[i<arrayBirds & i2<arrayFood]
[else]
[food nearby(r)]
changePath
arrayBirds.move
arrayBirds.draw
moveSnowflakes
[i<arraySnowflakes]
arraySnowflakes.move
arraySnowflakes.draw
calculateFoodAction
8
[i<arrayFood]
[else]
[food landed & position>y=400 || liftime=0]
food.draw
food-1
[liftime=0]
for bird count < arrayBirds
[Bird is state feeding|picking]
reset velocity & state=Alive


calculate SnowballAction
[ic snowballs]
[snowball Timer>0]
[snowball timer=0]
snowballs. draw
snowballs. draw
[bird Number< arrayBirds]
[else]
[ifHit=n]
score-1
hit=true
state=dead
score+10
-1 bird→birdArray
snowballs -1
& SnowballCount+1
snowball.draw
x:number
y:number
color:string
timer:number
[timer>0]
beginPath
moveTo
draw circle
→gets smaller & fallsdown
→timer
x&y are
clicked position
timer--
ifHit
beginPath
moveTo
x&y[bird]
draw circle
around bird
[pointInPath]
[else]
true
false


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
[distances100]
[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--
save food coordinates
[timer=0]
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>0]
food.x=x
food.y=y
food.timer=25
food.lifetime=250
push(food)
food count +1