Designdokument

Für eine Abgabe an der

Hochschule Furtwangen University

im Kurs Entwicklung Interaktiver Anwendungen II

Wintersemester 21/22

Prof. Jirka Dell´ Oro-Friedl

Endabgabe – Döner-Trainer

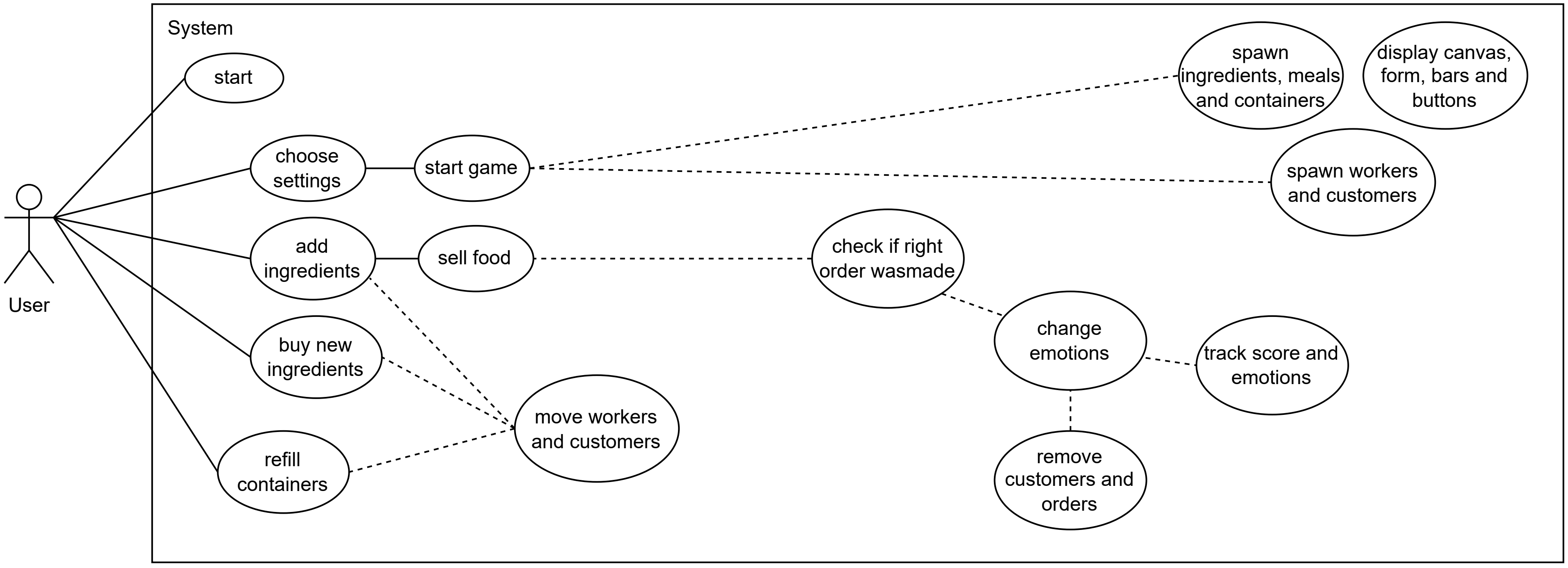
Jasmin Basler

Matrikel-Nr.: 265114

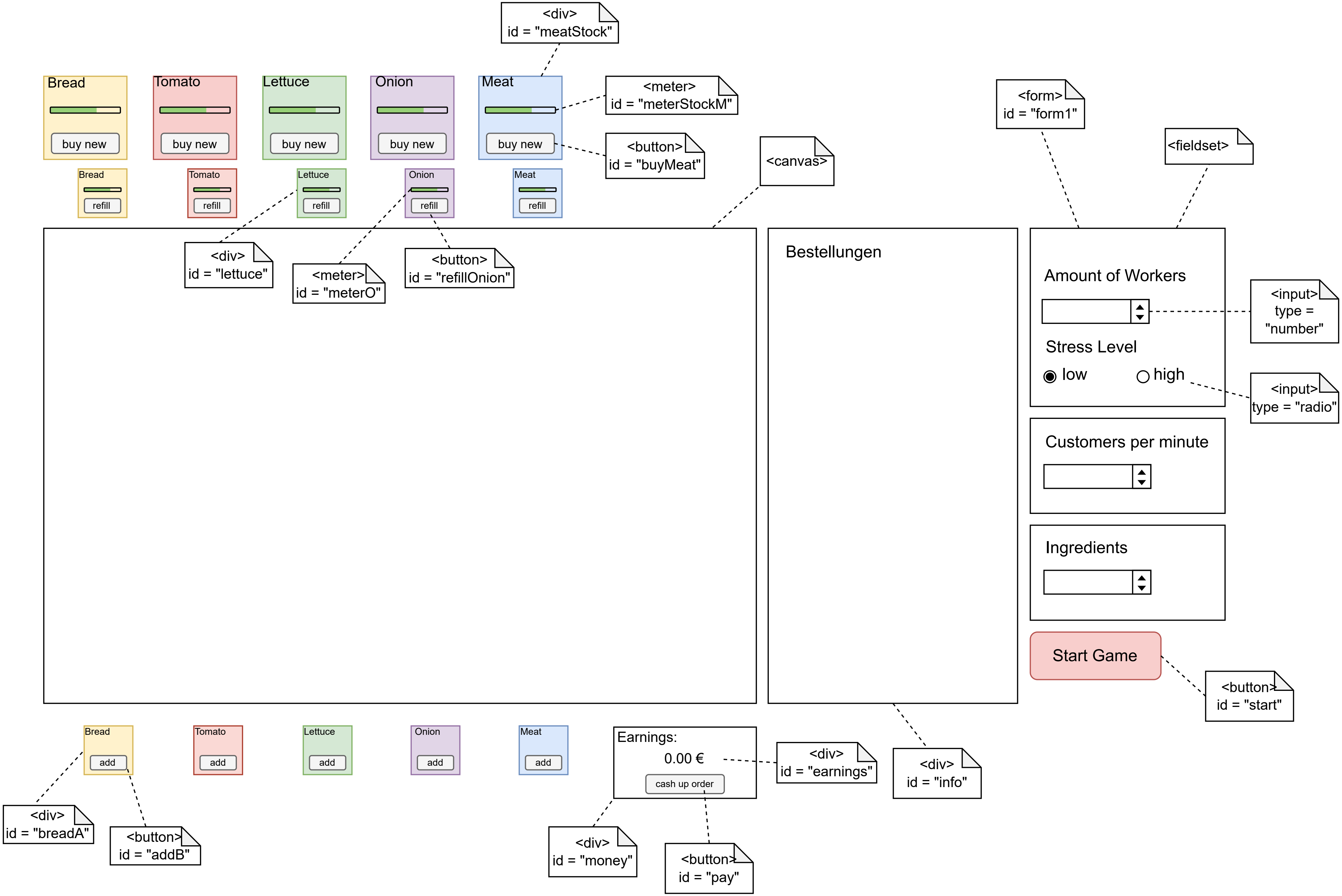
In Zusammenarbeit mit Fiona Virnich

Stand: 15.02.2022

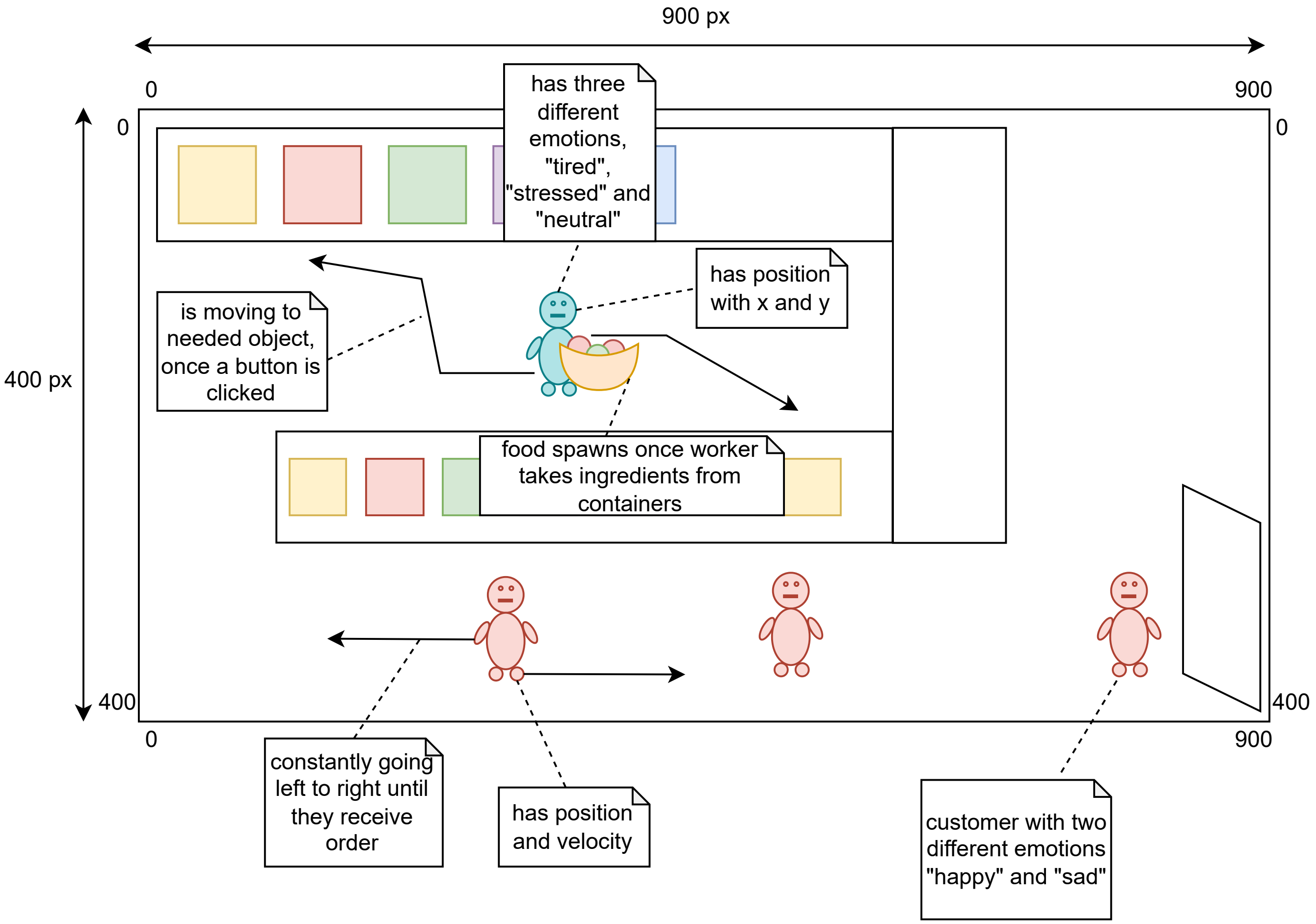
Döner-Trainer: Use-Case-Diagram


System
User
start
choose settings
start game
add ingredients
sell food
buy new ingredients
refill containers
move workers and customers
spawn ingredients, meals and containers
display canvas, form, bars and buttons
spawn workers and customers
check if right order wasmade
change emotions
track score and emotions
remove customers and orders

Döner-Trainer: UI-Scribble



Döner-Trainer: UI-Scribble Canvas



has three different emotions, "tired", "stressed" and "neutral"

has position with x and y

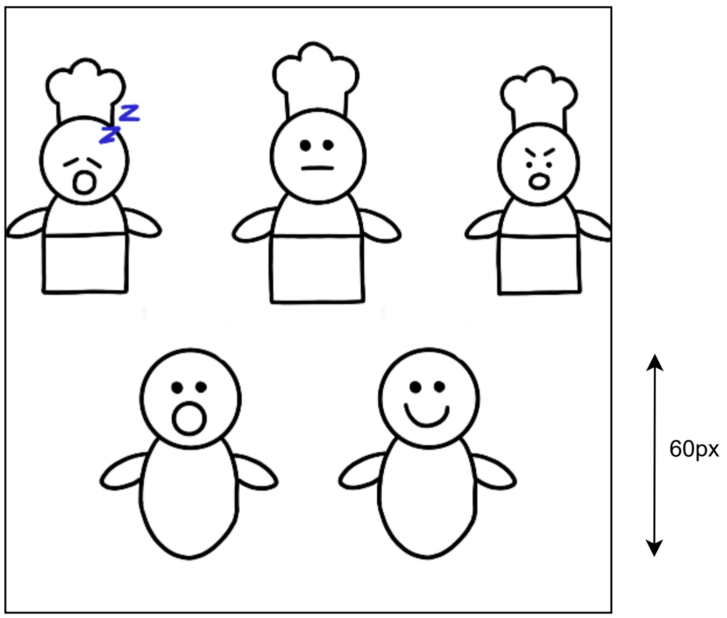
is moving to needed object, once a button is clicked

food spawns once worker takes ingredients from containers

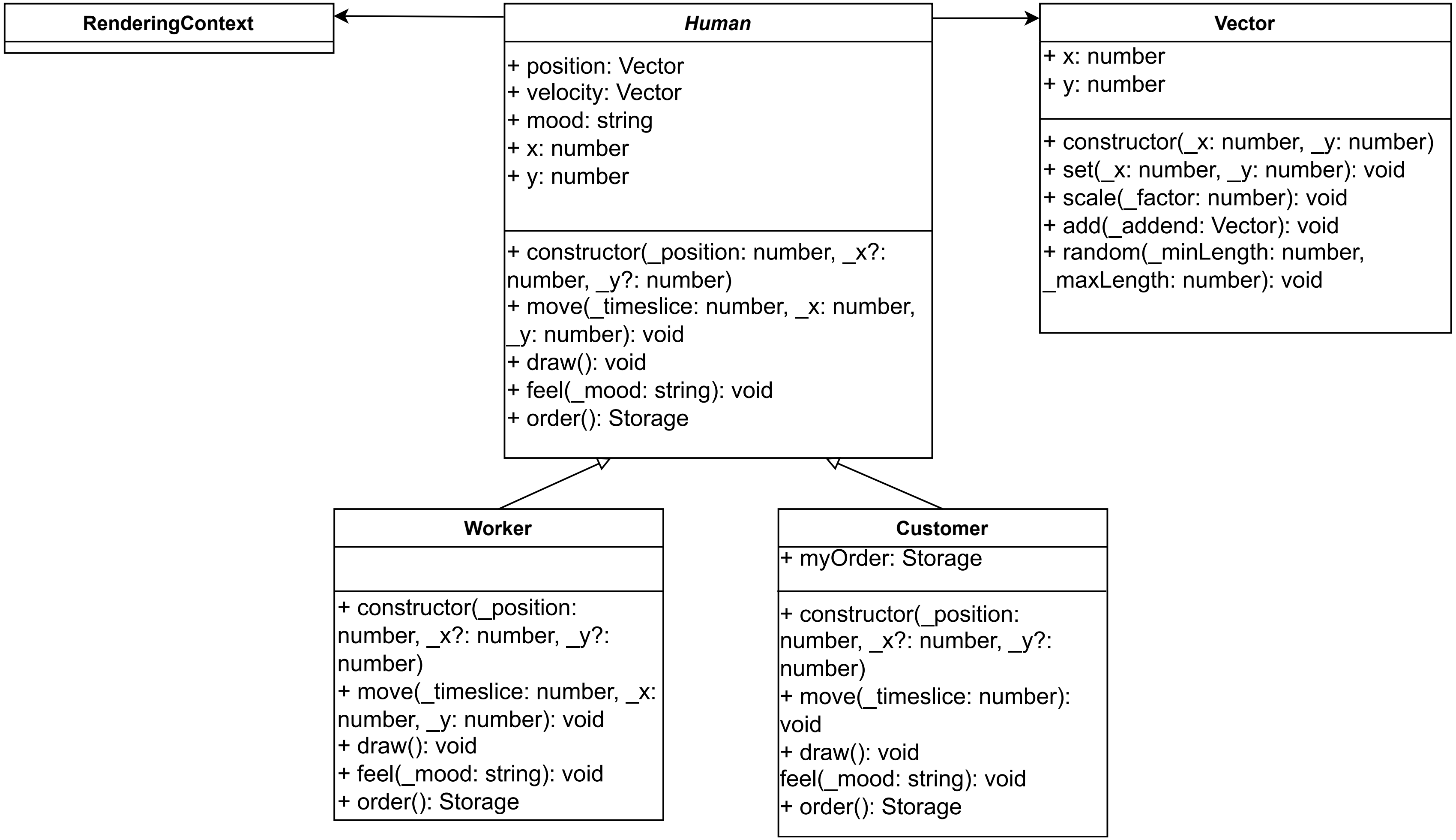
constantly going left to right until they receive order

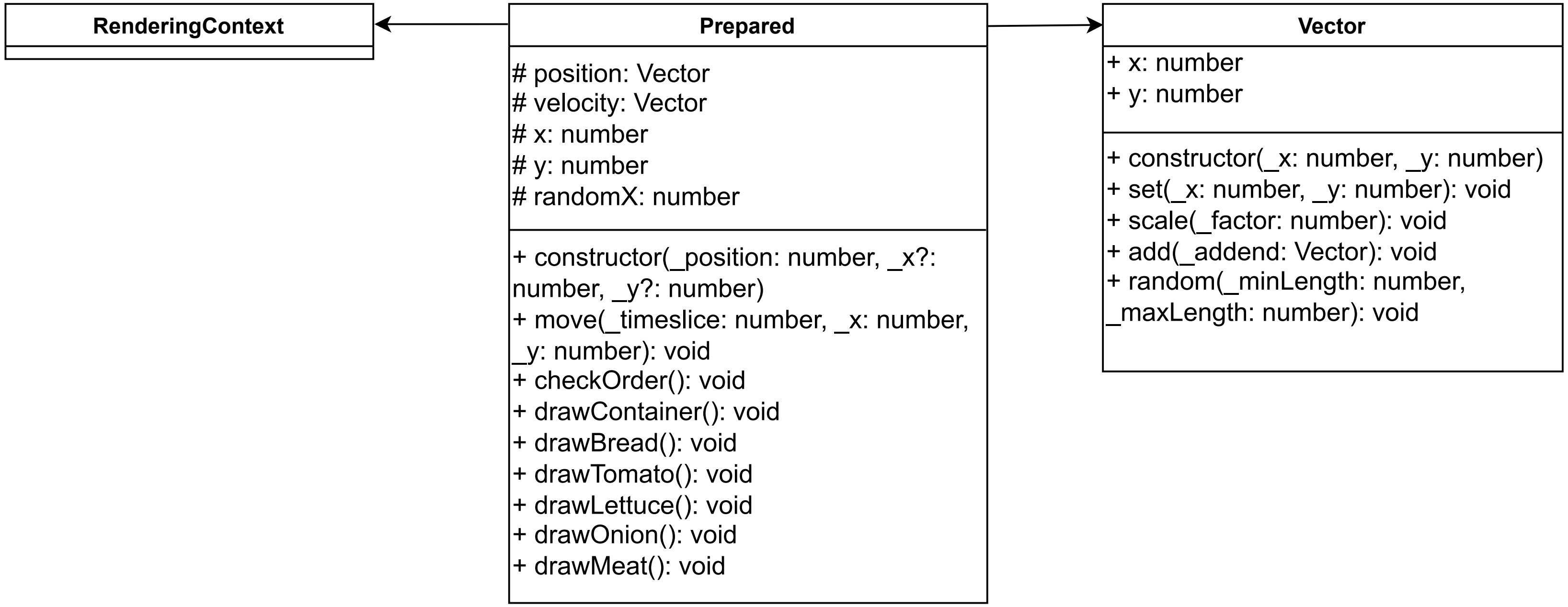
has position and velocity

customer with two different emotions "happy" and "sad"



Döner-Trainer: Class Diagram


RenderingContext
Human
+ position: Vector
+ velocity: Vector
+ mood: string
+ x: number
+ y: number
+ constructor(_position: number, _x?: number, _y?: number)
+ move(_timeslice: number, _x: number, _y: number): void
+ draw(): void
+ feel(_mood: string): void
+ order(): Storage
Vector
+ x: number
+ y: number
+ constructor(_x: number, _y: number)
+ set(_x: number, _y: number): void
+ scale(_factor: number): void
+ add(_addend: Vector): void
+ random(_minLength: number, _maxLength: number): void
Worker
+ constructor(_position: number, _x?: number, _y?: number)
+ move(_timeslice: number, _x: number, _y: number): void
+ draw(): void
+ feel(_mood: string): void
+ order(): Storage
Customer
+ myOrder: Storage
+ constructor(_position: number, _x?: number, _y?: number)
+ move(_timeslice: number): void
+ draw(): void
feel(_mood: string): void
+ order(): Storage



RenderingContext

Prepared

# position: Vector
# velocity: Vector
# x: number
# y: number
# randomX: number

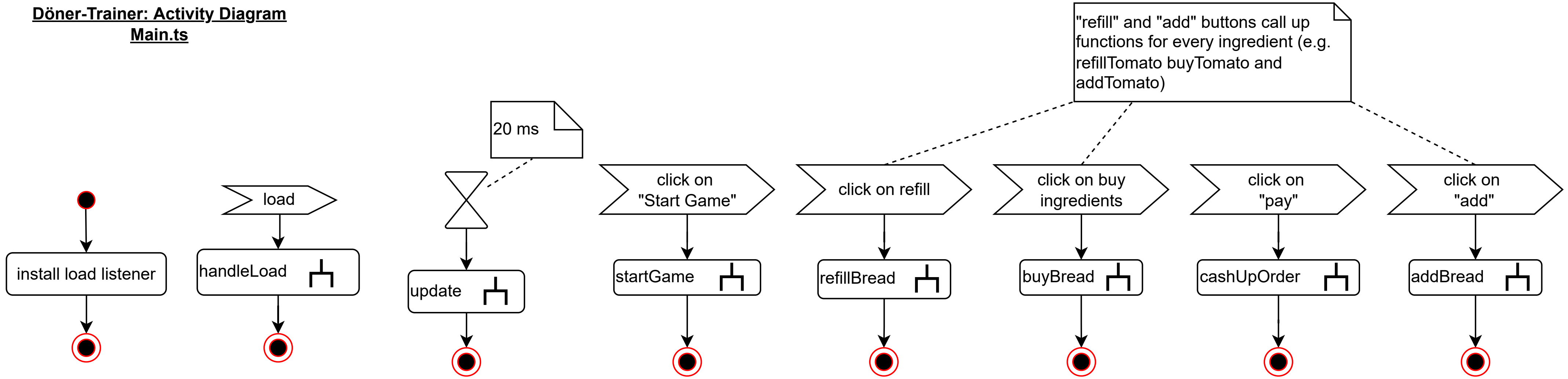
+ constructor(_position: number, _x?: number, _y?: number)
+ move(_timeslice: number, _x: number, _y: number): void
+ checkOrder(): void
+ drawContainer(): void
+ drawBread(): void
+ drawTomato(): void
+ drawLettuce(): void
+ drawOnion(): void
+ drawMeat(): void

Vector

+ x: number
+ y: number

+ constructor(_x: number, _y: number)
+ set(_x: number, _y: number): void
+ scale(_factor: number): void
+ add(_addend: Vector): void
+ random(_minLength: number, _maxLength: number): void

Döner-Trainer: Activity Diagram
Main.ts



Globals:

crc2: CanvasRenderingContext2D
let info: HTMLElement
imageData: ImageData

workers: Worker[]
customers: Customer[]
orders: Storage[]
ordersMade: Storage[]
displayOrders : string[]
drawOrders: Prepared[]
currentCustomerAmount: number = 0
earnings: number 0
happyScore: number = 0
orderCorrect: boolean
moodWorker: string
moodCustomer: string

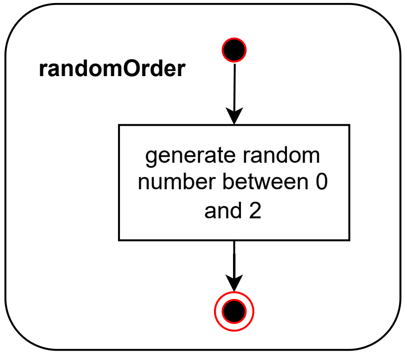
xOfWorker: number
yOfWorker: number
xOfWorker2: number
yOfWorker2: number
xOfCustomer: number
yOfCustomer: number

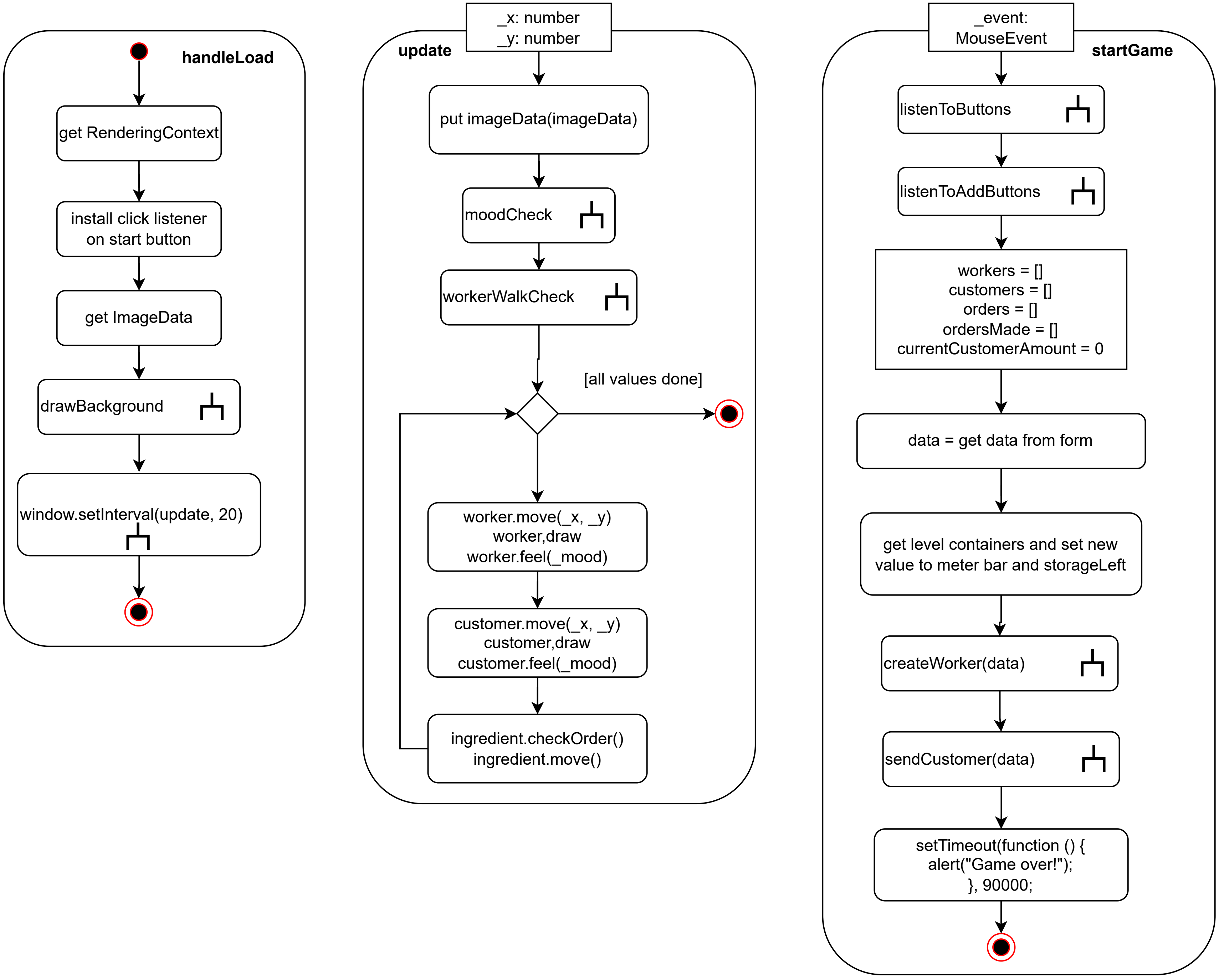
Storage
bread: number tomato: number lettuce: number onion: number meat: number

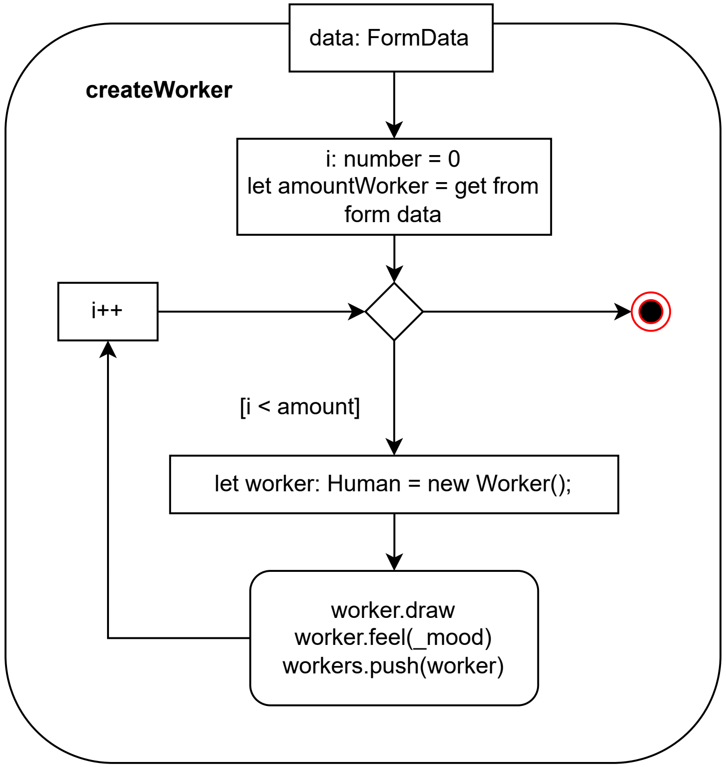
storageLeft
bread: 1000 tomato: 1000 lettuce: 1000 onion: 1000 meat: 1000

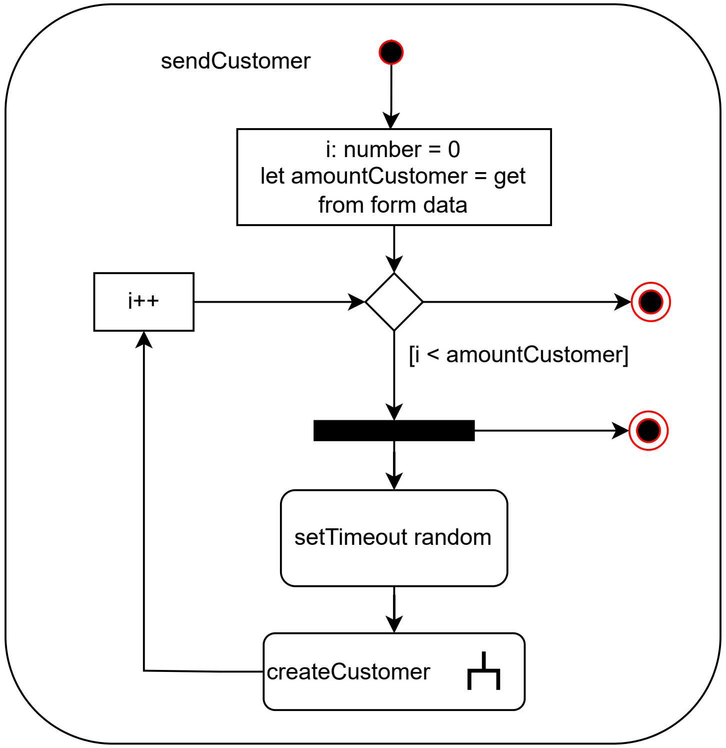
counterLeft
bread: 80 tomato: 80 lettuce: 80 onion: 80 meat: 80

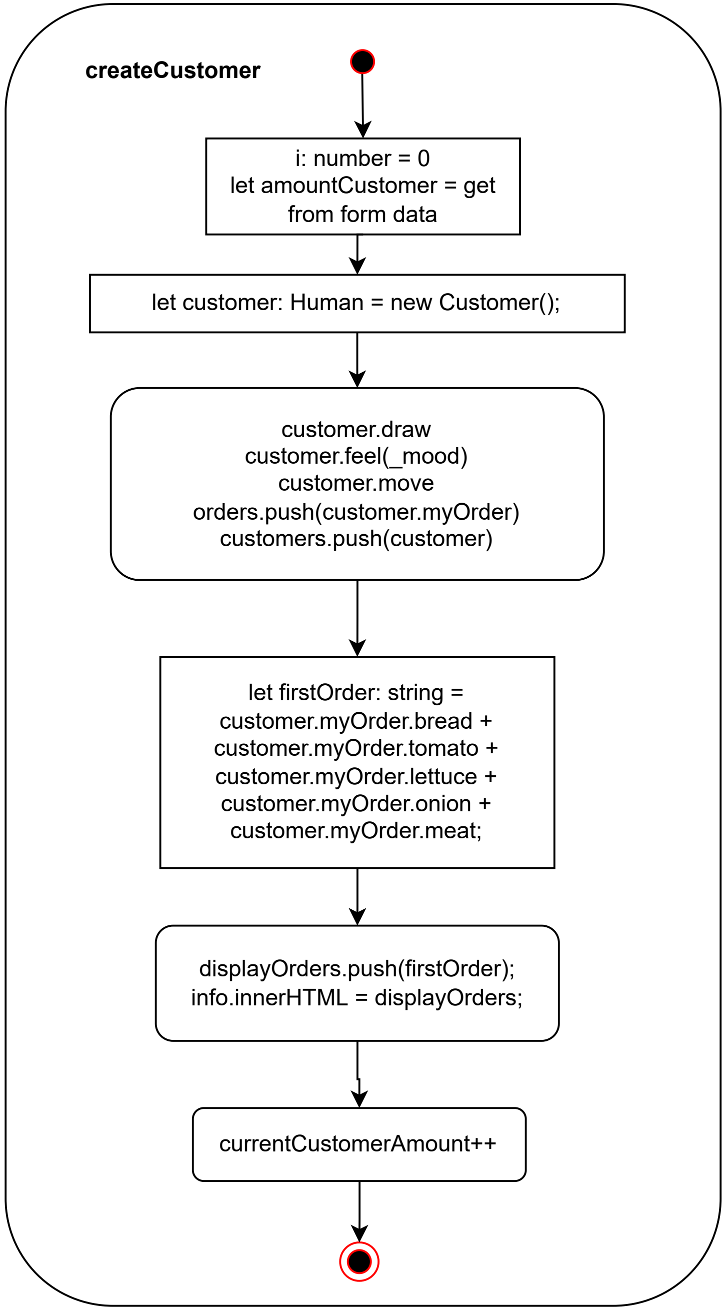
currentOrder
bread: 0 tomato: 0 lettuce: 0 onion: 0 meat: 0

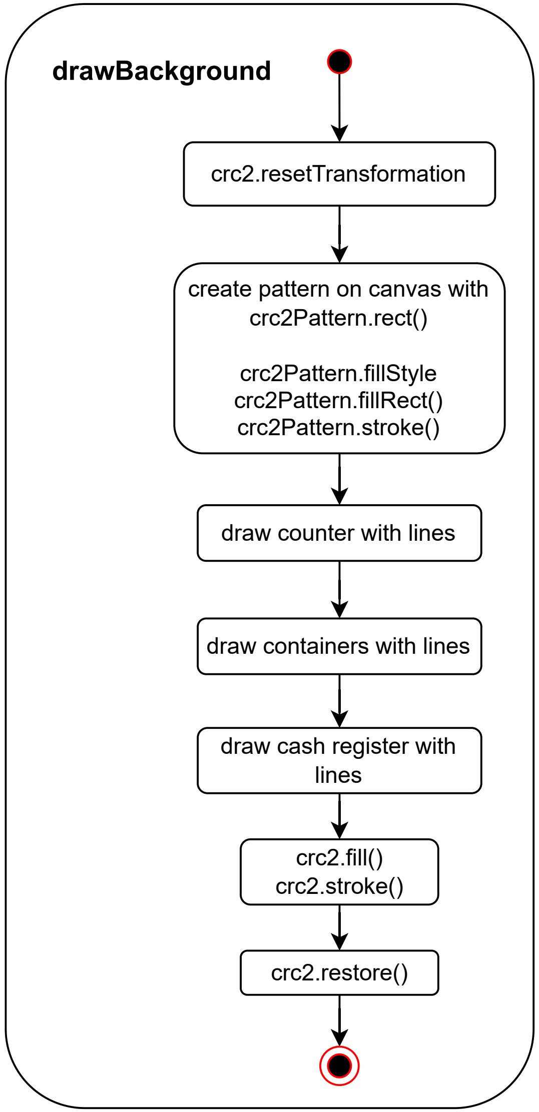


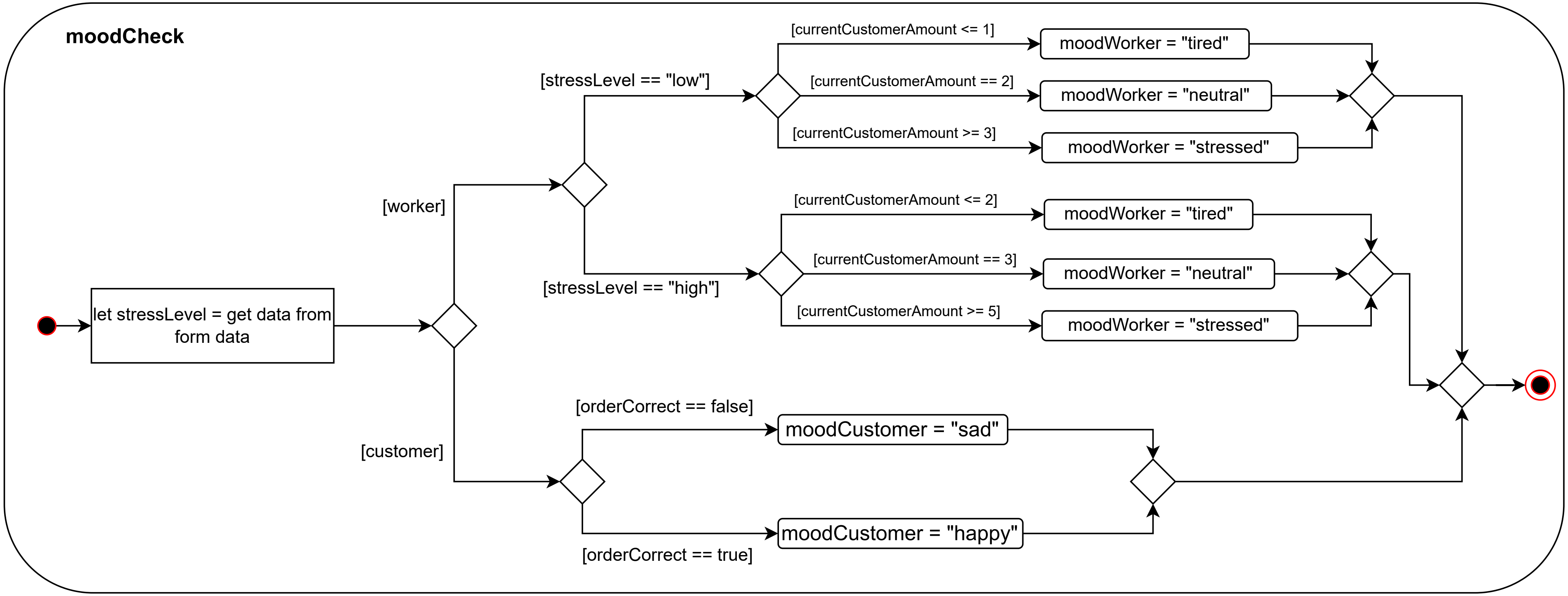

handleLoad
get RenderingContext
install click listener on start button
get ImageData
drawBackground
window.setInterval(update, 20)
update
_x: number
_y: number
put imageData(imageData)
moodCheck
workerWalkCheck
[all values done]
worker.move(_x, _y)
worker,draw
worker.feel(_mood)
customer.move(_x, _y)
customer,draw
customer.feel(_mood)
ingredient.checkOrder()
ingredient.move()
_event: MouseEvent
startGame
listenToButtons
listenToAddButtons
workers = []
customers = []
orders = []
ordersMade = []
currentCustomerAmount = 0
data = get data from form
get level containers and set new value to meter bar and storageLeft
createWorker(data)
sendCustomer(data)
setTimeout(function () {
alert("Game over!");
}, 90000;

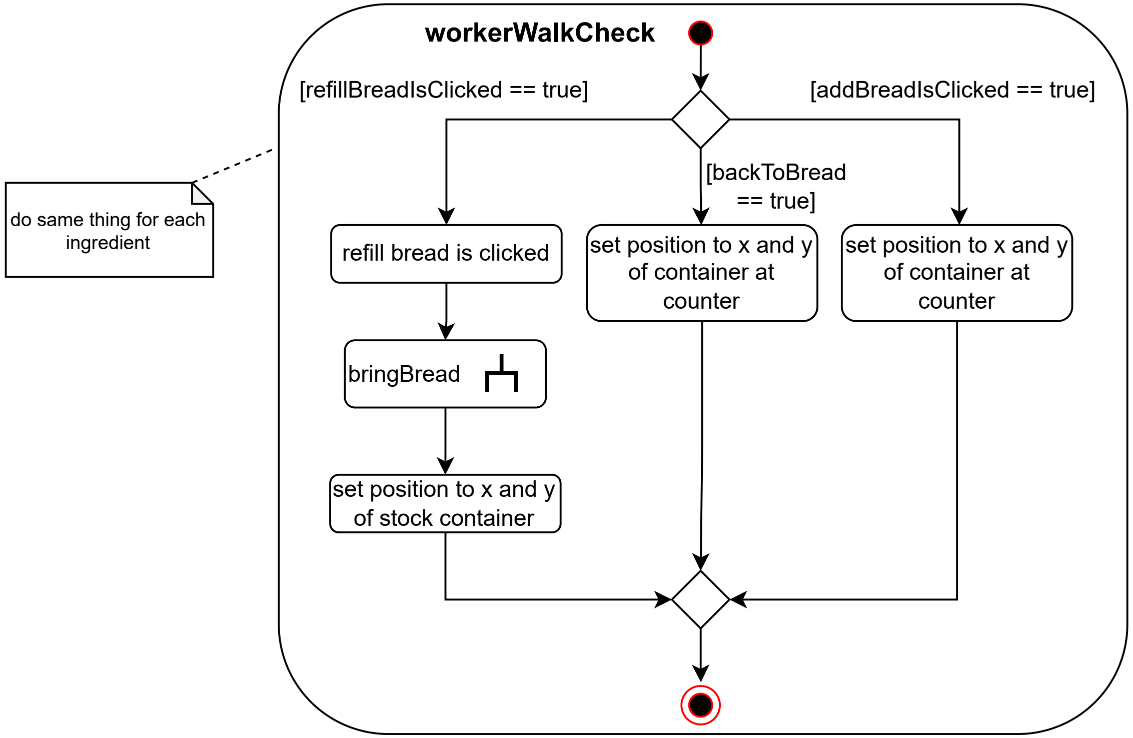

data: FormData
createWorker
i: number = 0
let amountWorker = get from form data
i++
[i < amount]
let worker: Human = new Worker();
worker.draw
worker.feel(_mood)
workers.push(worker)


sendCustomer
i: number = 0
let amountCustomer = get from form data
i++
[i < amountCustomer]
setTimeout random
createCustomer


createCustomer
i: number = 0
let amountCustomer = get from form data
let customer: Human = new Customer();
customer.draw
customer.feel(_mood)
customer.move
orders.push(customer.myOrder)
customers.push(customer)
let firstOrder: string =
customer.myOrder.bread +
customer.myOrder.tomato +
customer.myOrder.lettuce +
customer.myOrder.onion +
customer.myOrder.meat;
displayOrders.push(firstOrder);
info.innerHTML = displayOrders;
currentCustomerAmount++


drawBackground
crc2.resetTransformation
create pattern on canvas with
crc2Pattern.rect()
crc2Pattern.fillStyle
crc2Pattern.fillRect()
crc2Pattern.stroke()
draw counter with lines
draw containers with lines
draw cash register with lines
crc2.fill()
crc2.stroke()
crc2.restore()

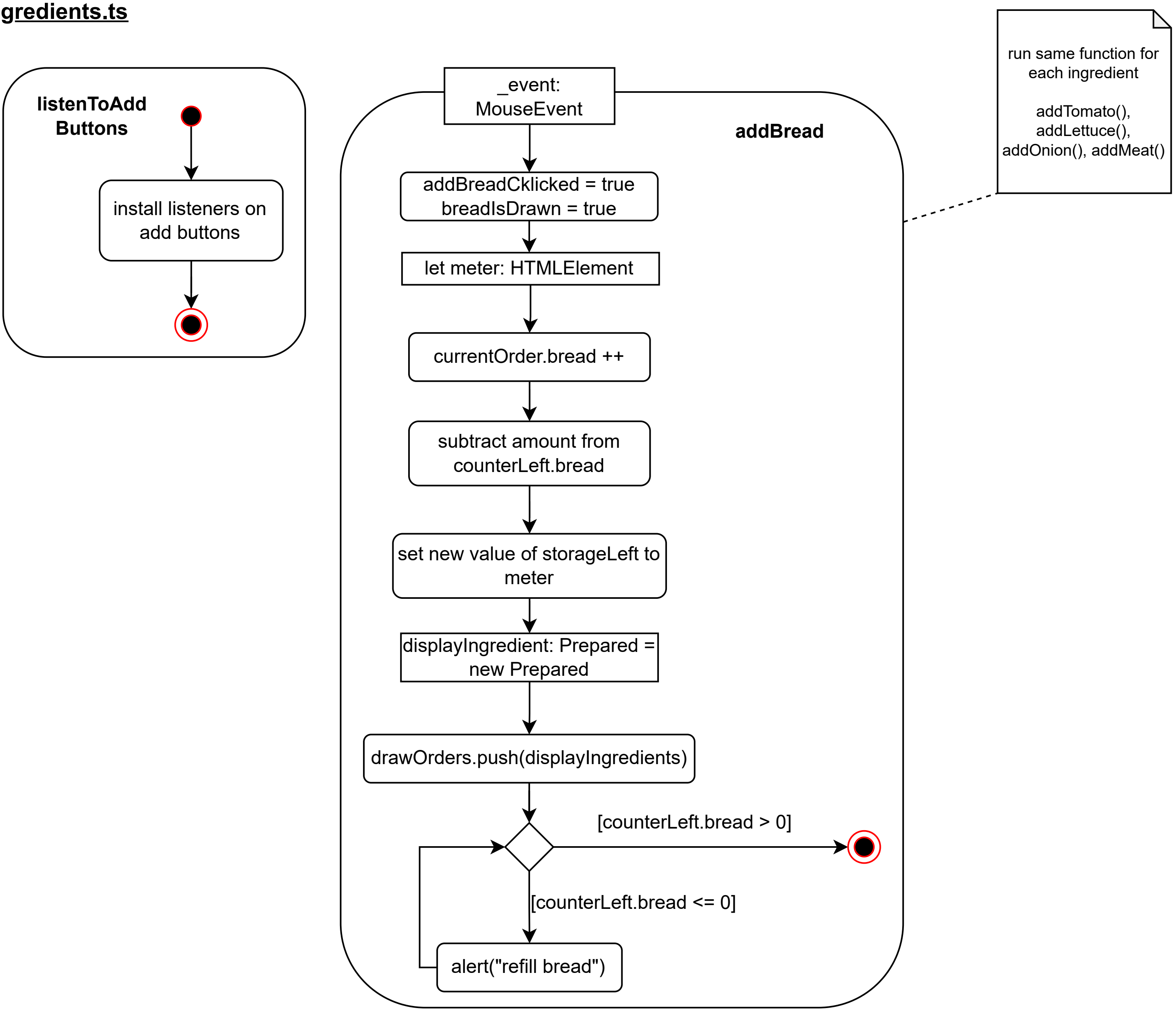

moodCheck
let stressLevel = get data from form data
[worker]
[customer]
[stressLevel == "low"]
[stressLevel == "high"]
[currentCustomerAmount <= 1]
moodWorker = "tired"
[currentCustomerAmount == 2]
moodWorker = "neutral"
[currentCustomerAmount >= 3]
moodWorker = "stressed"
[currentCustomerAmount <= 2]
moodWorker = "tired"
[currentCustomerAmount == 3]
moodWorker = "neutral"
[currentCustomerAmount >= 5]
moodWorker = "stressed"
[orderCorrect == false]
moodCustomer = "sad"
[orderCorrect == true]
moodCustomer = "happy"


workerWalkCheck
do same thing for each ingredient
[refillBreadIsClicked == true]
[addBreadIsClicked == true]
[backToBread == true]
refill bread is clicked
bringBread
set position to x and y of stock container
set position to x and y of container at counter
set position to x and y of container at counter

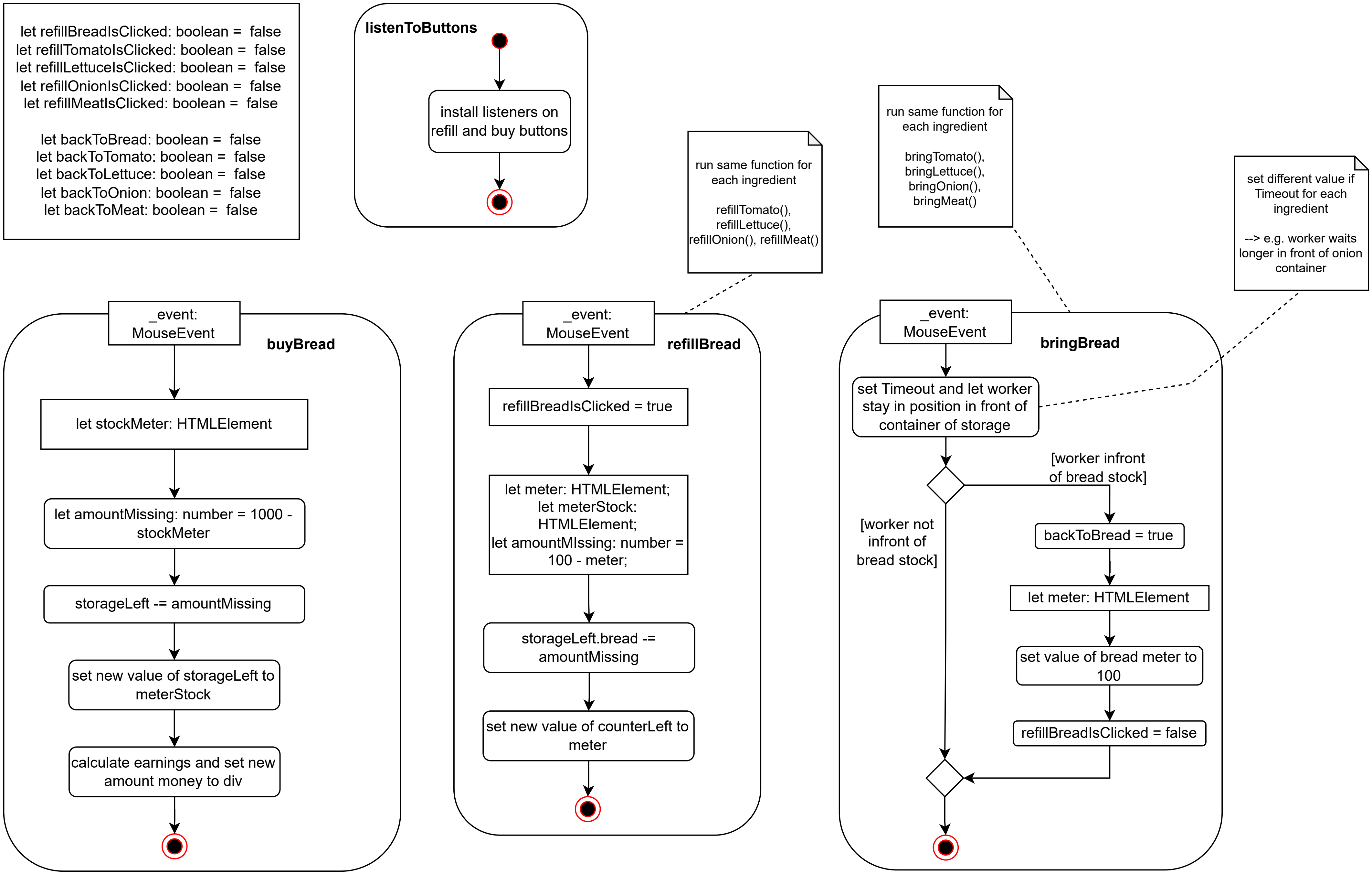
Döner-Trainer: Activity Diagram addIngredients.ts

let addBreadIsClicked: boolean = false
let addTomatoIsClicked: boolean = false
let addLettuceIsClicked: boolean = false
let addOnionIsClicked: boolean = false
let addMeatIsClicked: boolean = false

let addBreadIsDrawn: boolean = false
let addTomatoIsDrawn: boolean = false
let addLettuceIsDrawn: boolean = false
let addOnionIsDrawn: boolean = false
let addMeatIsDrawn: boolean = false

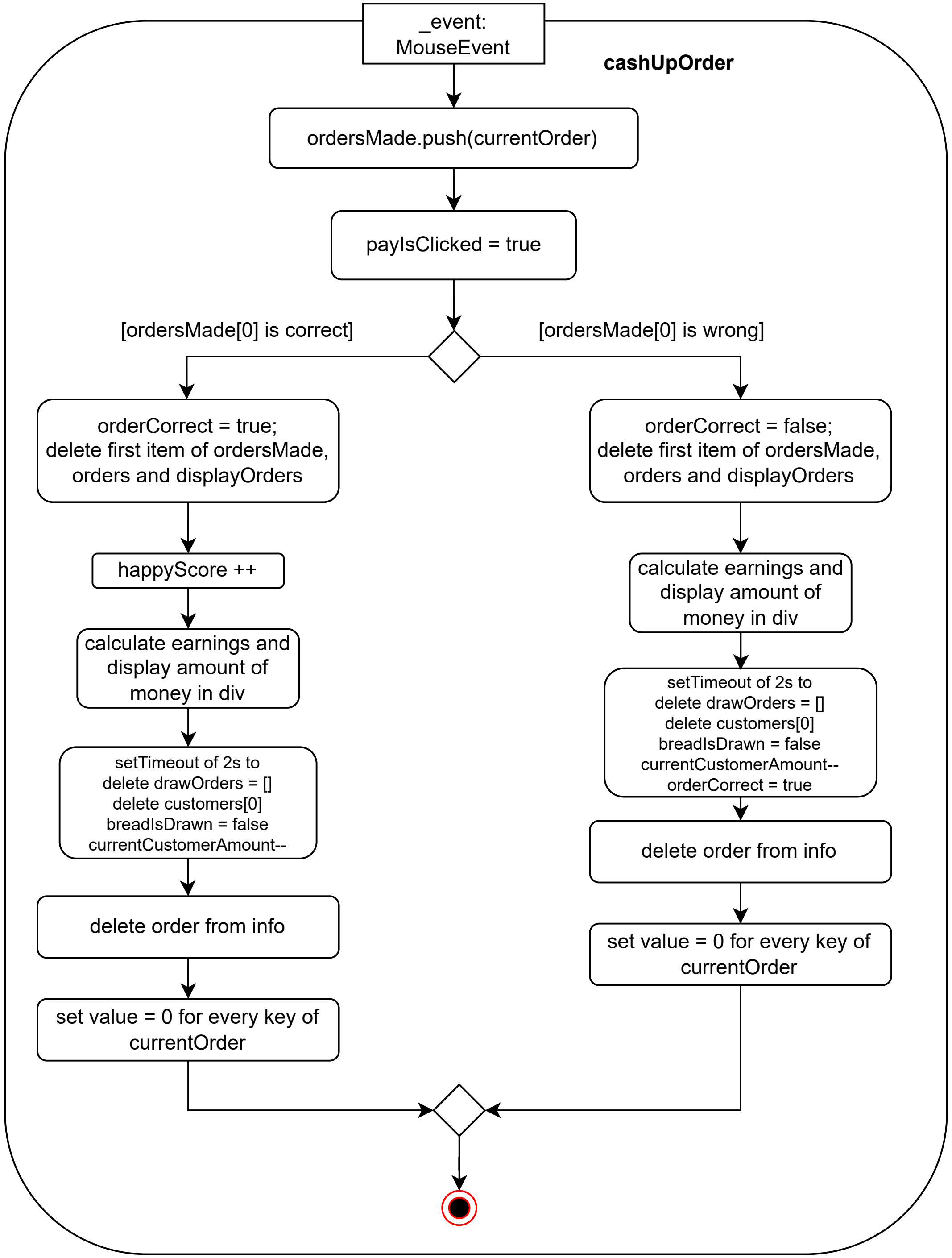

listenToAdd Buttons
install listeners on add buttons
_event: MouseEvent
addBread
addBreadCklicked = true
breadIsDrawn = true
let meter: HTMLElement
currentOrder.bread ++
subtract amount from counterLeft.bread
set new value of storageLeft to meter
displayIngredient: Prepared = new Prepared
drawOrders.push(displayIngredients)
[counterLeft.bread > 0]
[counterLeft.bread <= 0]
alert("refill bread")
run same function for each ingredient
addTomato(), addLettuce(), addOnion(), addMeat()

Döner-Trainer: Activity Diagram container.ts

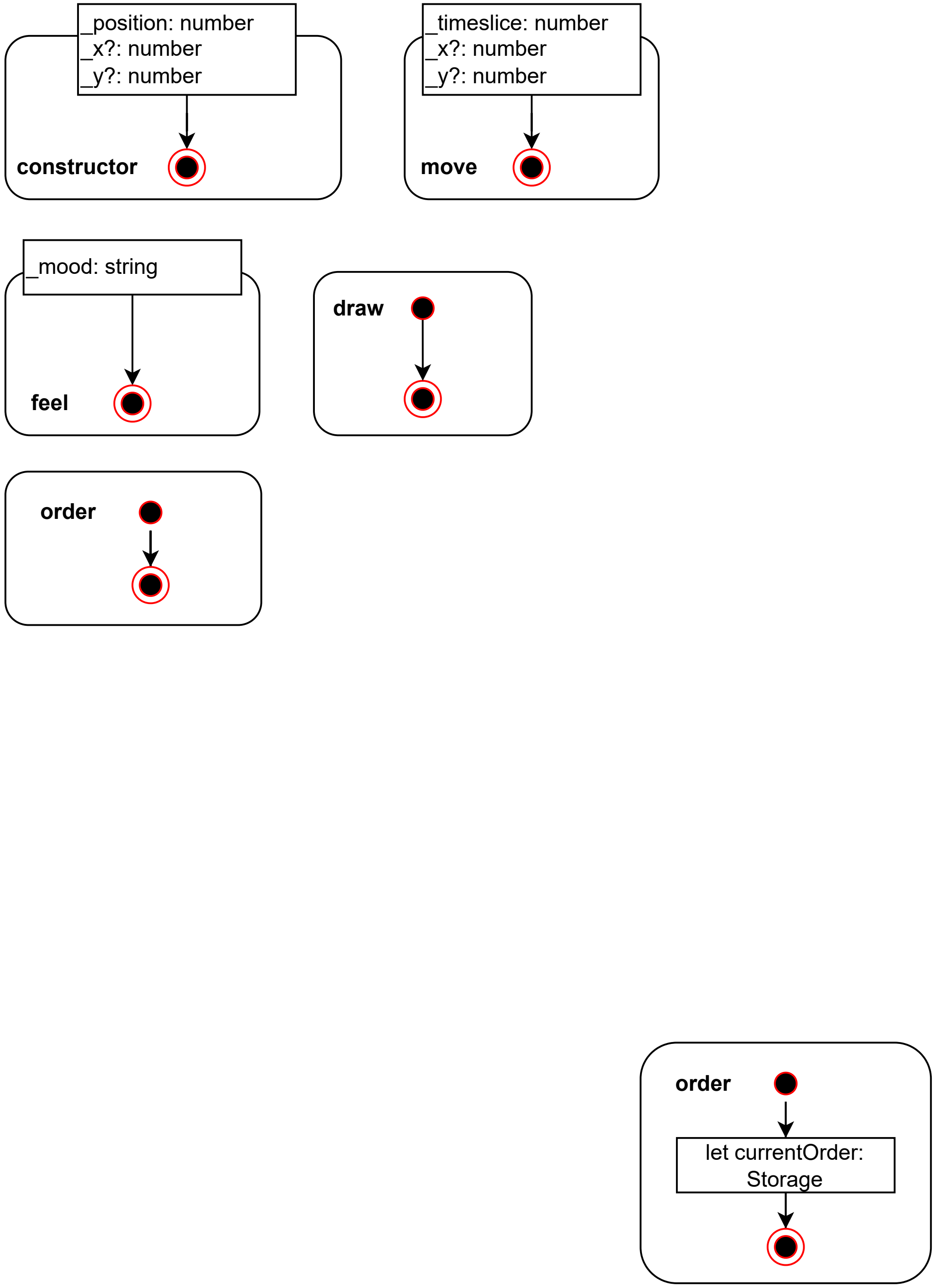

let refillBreadIsClicked: boolean = false
let refillTomatoIsClicked: boolean = false
let refillLettuceIsClicked: boolean = false
let refillOnionIsClicked: boolean = false
let refillMeatIsClicked: boolean = false
let backToBread: boolean = false
let backToTomato: boolean = false
let backToLettuce: boolean = false
let backToOnion: boolean = false
let backToMeat: boolean = false
listenToButtons
install listeners on refill and buy buttons
run same function for each ingredient
refillTomato(), refillLettuce(), refillOnion(), refillMeat()
run same function for each ingredient
bringTomato(), bringLettuce(), bringOnion(), bringMeat()
set different value if Timeout for each ingredient
--> e.g. worker waits longer in front of onion container
_event: MouseEvent
buyBread
let stockMeter: HTMLElement
let amountMissing: number = 1000 - stockMeter
storageLeft -= amountMissing
set new value of storageLeft to meterStock
calculate earnings and set new amount money to div
_event: MouseEvent
refillBread
refillBreadIsClicked = true
let meter: HTMLElement; let meterStock: HTMLElement; let amountMIssing: number = 100 - meter;
storageLeft.bread -= amountMissing
set new value of counterLeft to meter
_event: MouseEvent
bringBread
set Timeout and let worker stay in position in front of container of storage
[worker infront of bread stock]
[worker not infront of bread stock]
backToBread = true
let meter: HTMLElement
set value of bread meter to 100
refillBreadIsClicked = false

Döner-Trainer: Activity Diagram pay.ts

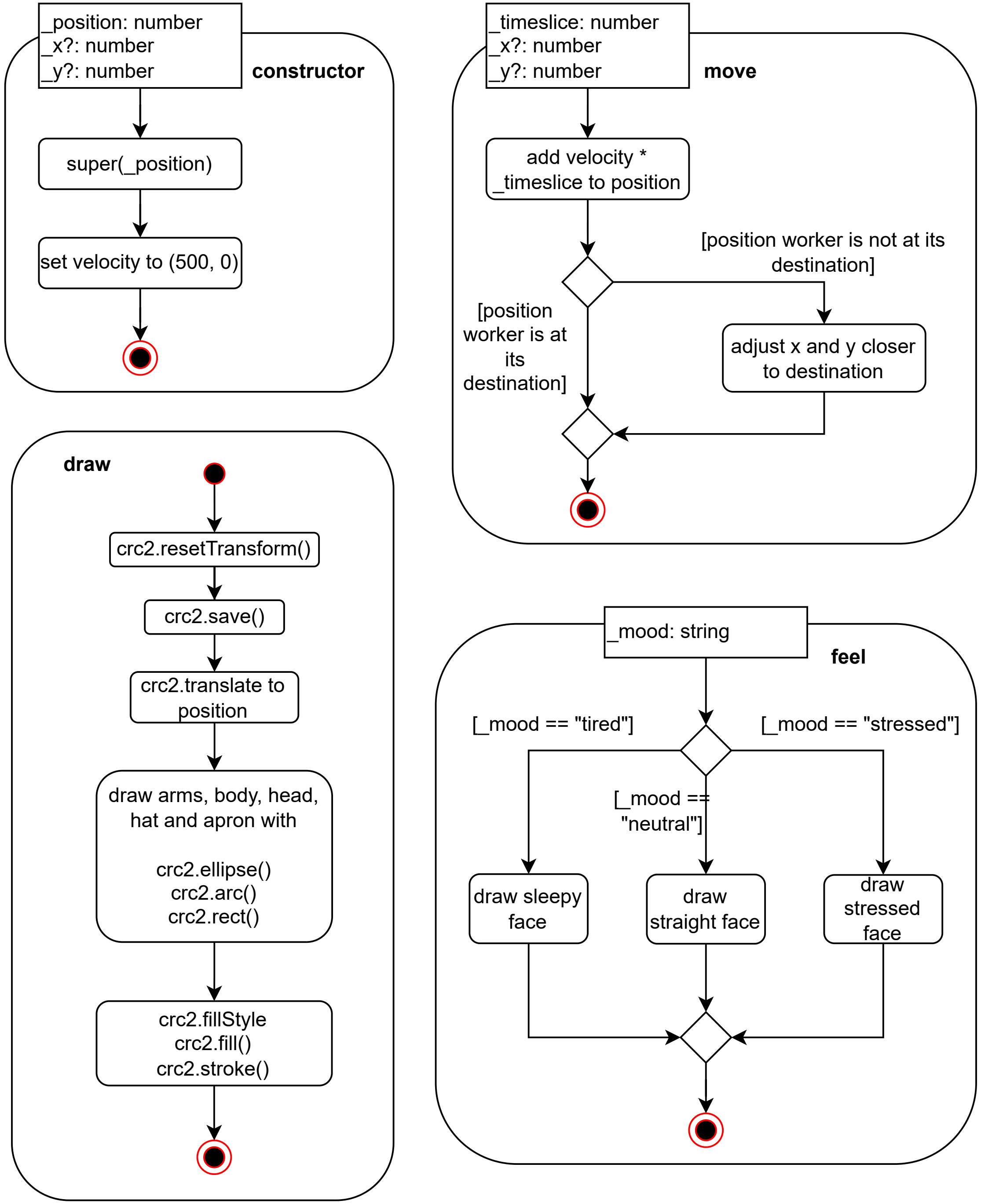
let payIsClicked : boolean = false


_event: MouseEvent
cashUpOrder
ordersMade.push(currentOrder)
payIsClicked = true
[ordersMade[0] is correct]
[ordersMade[0] is wrong]
orderCorrect = true; delete first item of ordersMade, orders and displayOrders
happyScore ++
calculate earnings and display amount of money in div
setTimeout of 2s to delete drawOrders = [] delete customers[0] breadIsDrawn = false currentCustomerAmount--
delete order from info
set value = 0 for every key of currentOrder
orderCorrect = false; delete first item of ordersMade, orders and displayOrders
calculate earnings and display amount of money in div
setTimeout of 2s to delete drawOrders = [] delete customers[0] breadIsDrawn = false currentCustomerAmount-- orderCorrect = true
delete order from info
set value = 0 for every key of currentOrder

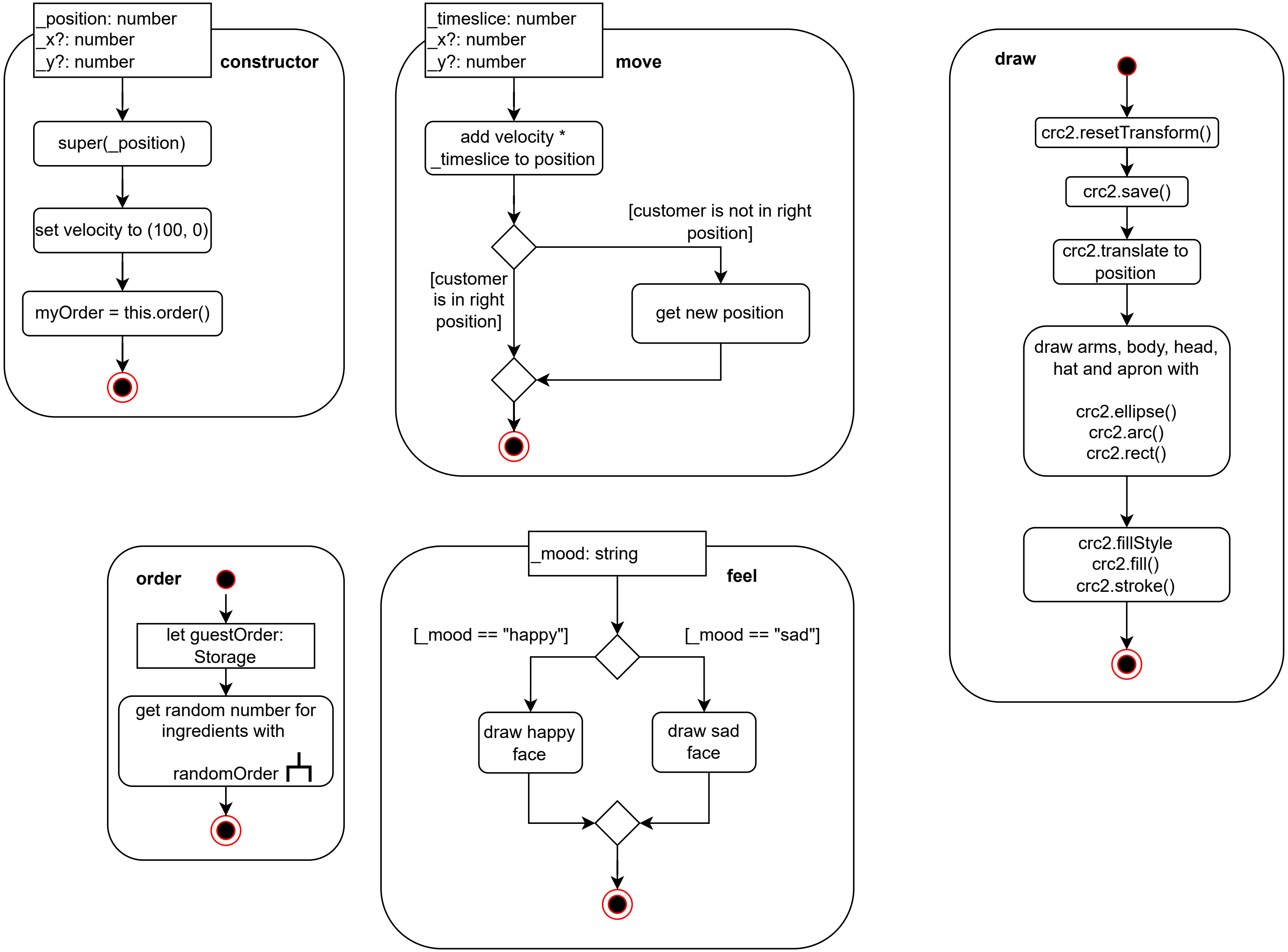
Döner-Trainer: Activity Diagram - Human



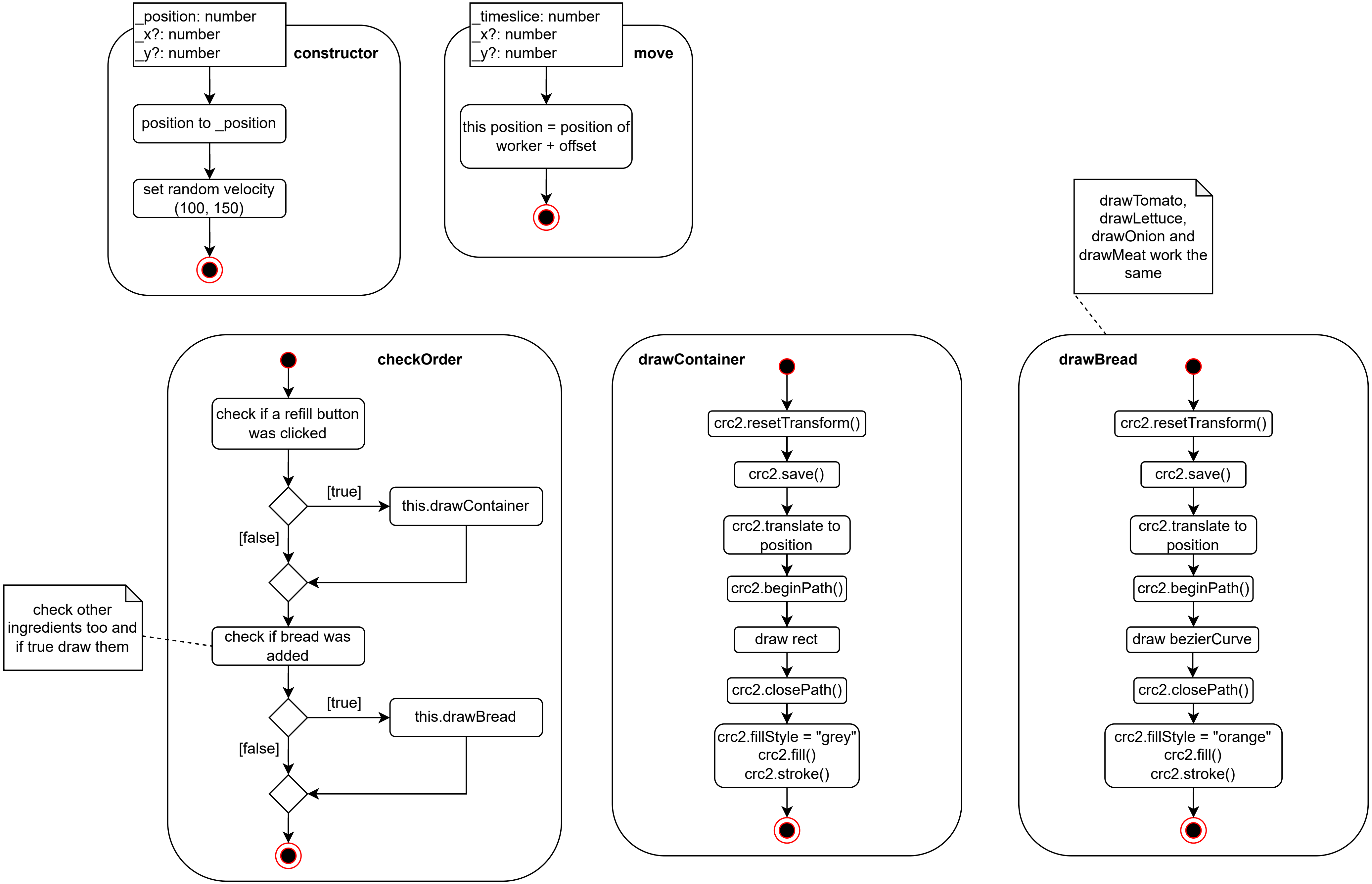
Döner-Trainer: Activity Diagram - Worker



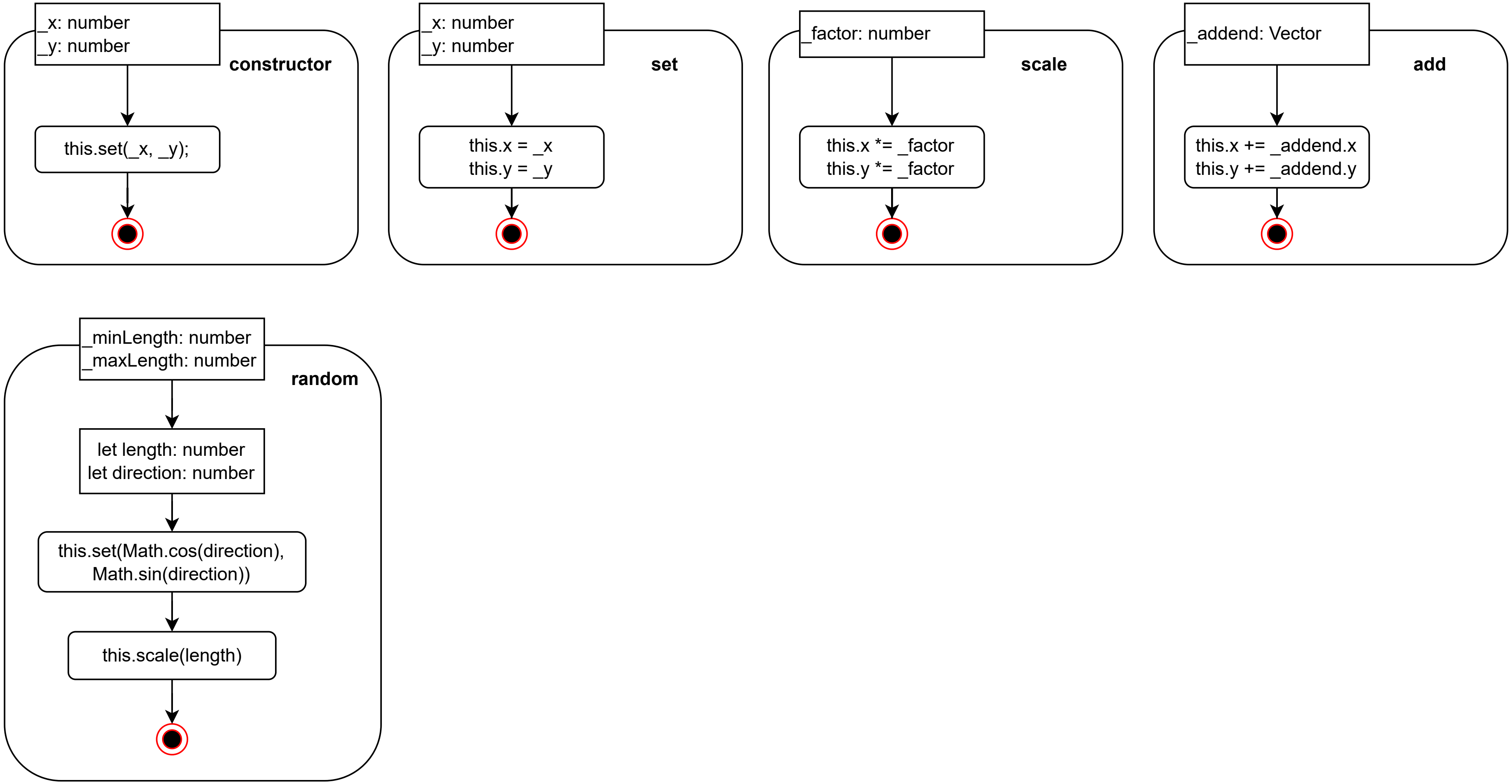
Döner-Trainer: Activity Diagram - Customer



Döner-Trainer: Activity Diagram - Prepared



Döner-Trainer: Activity Diagram - Vector


_x: number
_y: number
constructor
this.set(_x, _y);
_x: number
_y: number
set
this.x = _x
this.y = _y
_factor: number
scale
this.x *= _factor
this.y *= _factor
_addend: Vector
add
this.x += _addend.x
this.y += _addend.y
_minLength: number
_maxLength: number
random
let length: number
let direction: number
this.set(Math.cos(direction),
Math.sin(direction))
this.scale(length)