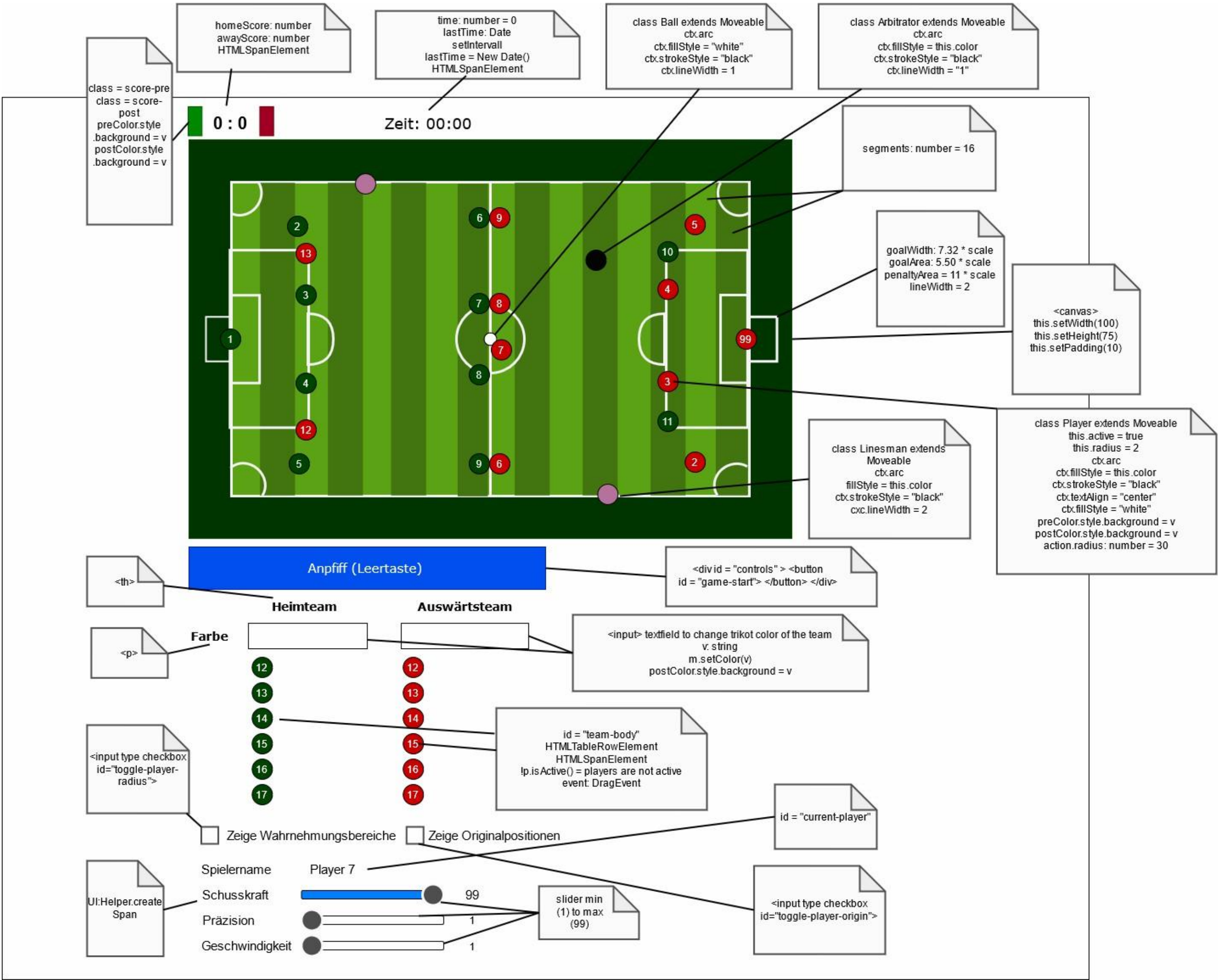
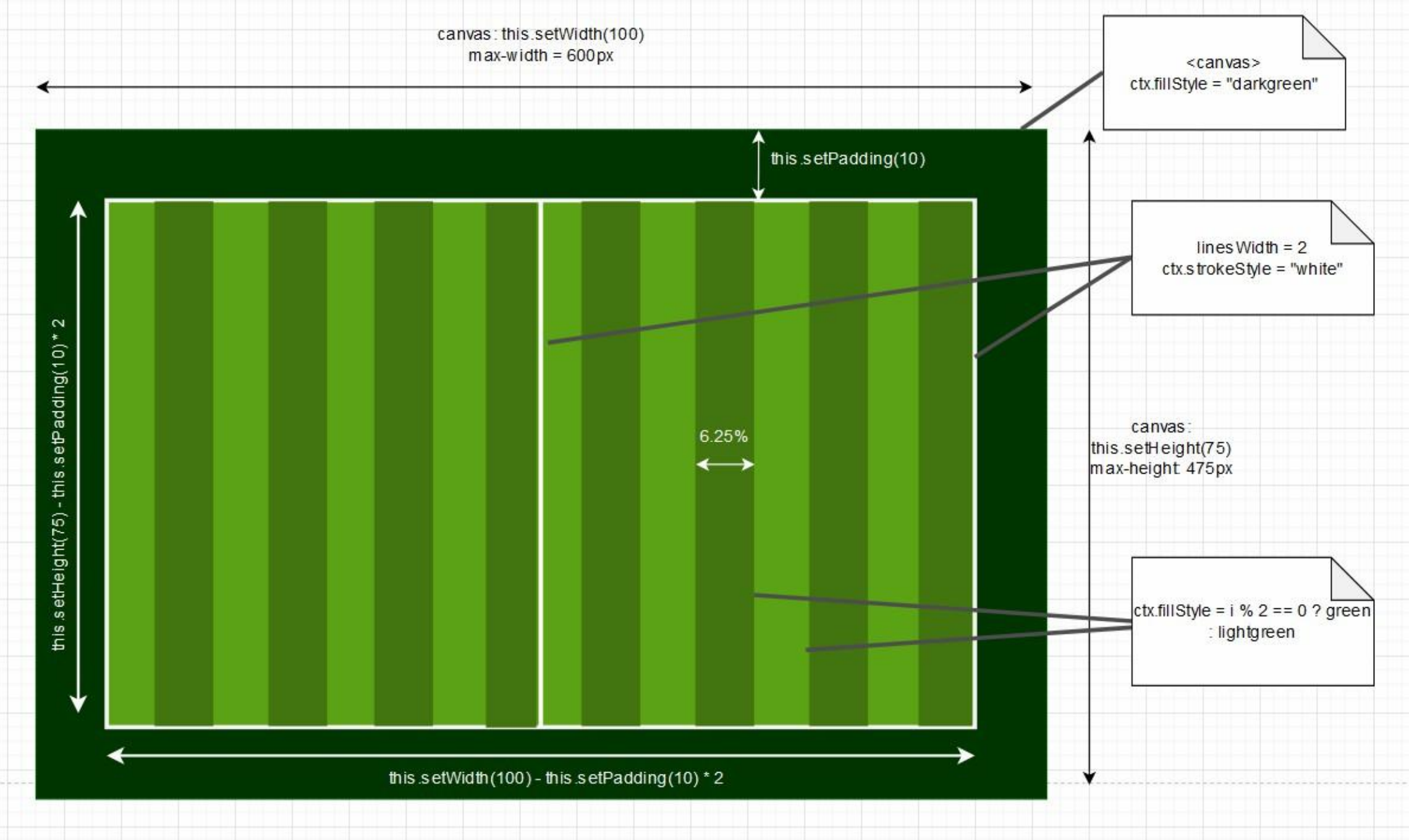
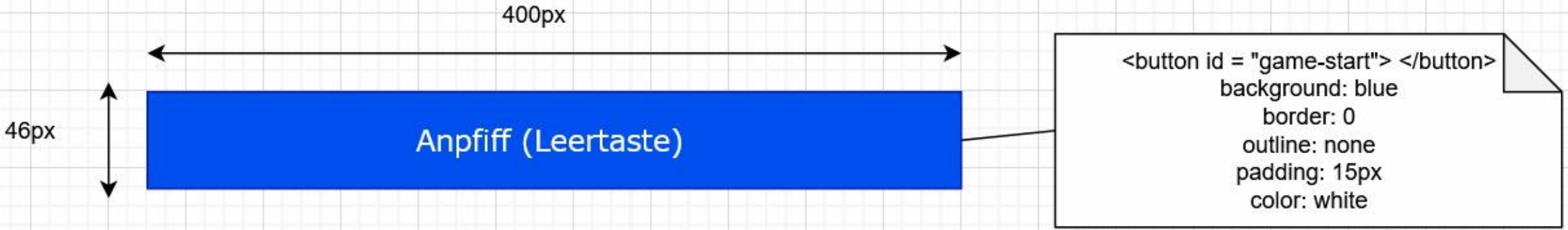
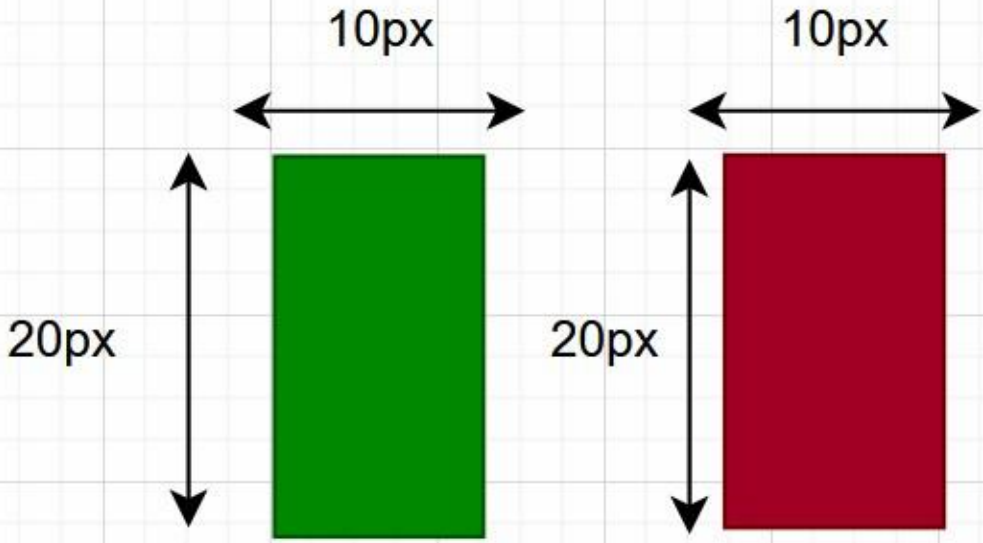
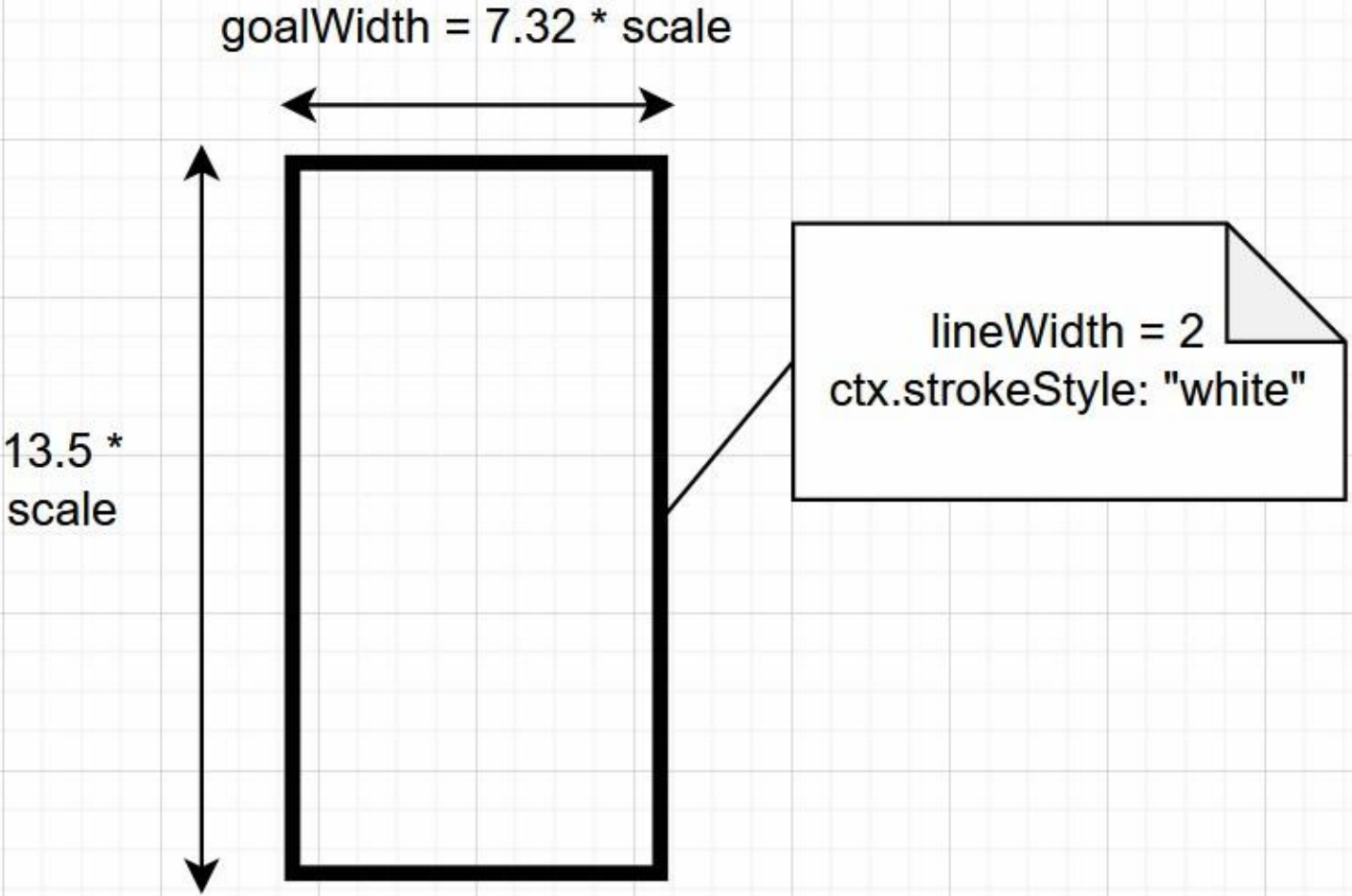
UI-Scribble

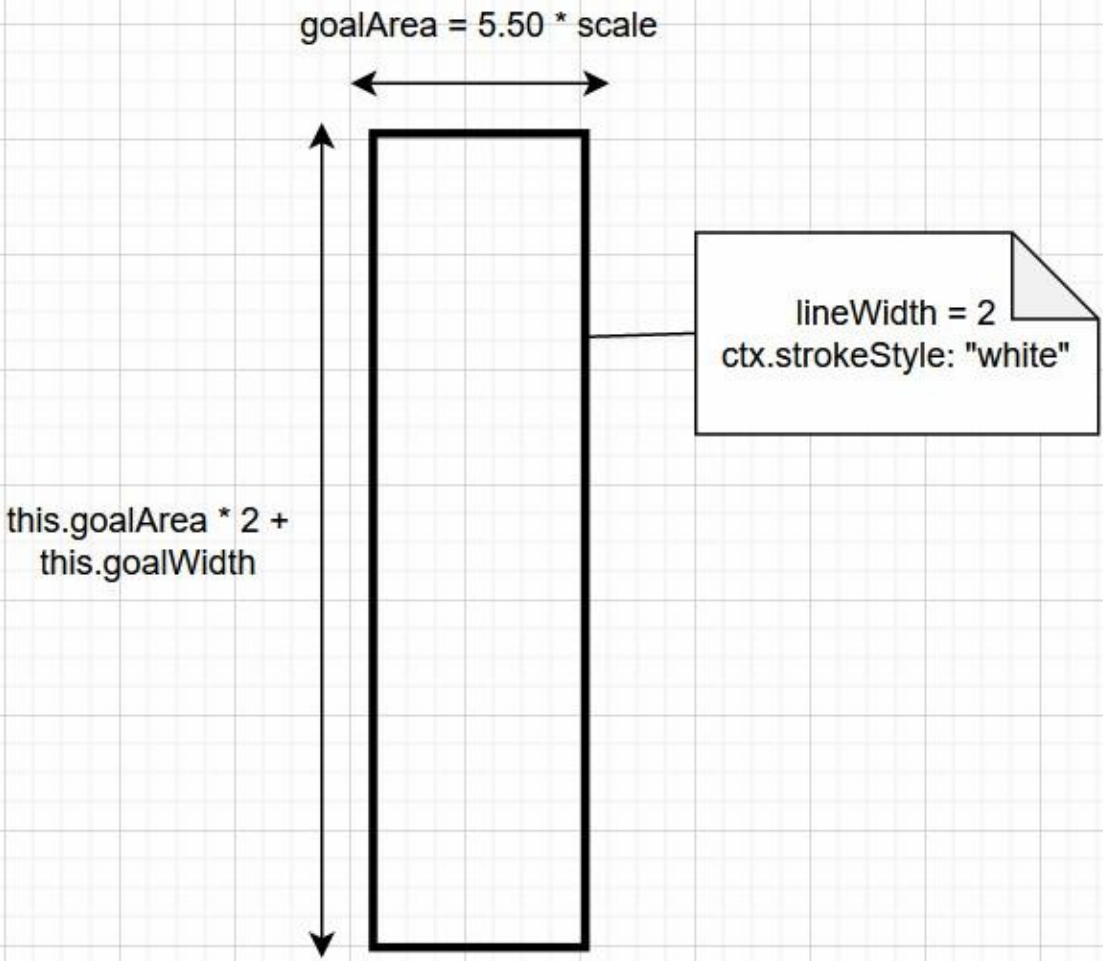

homeScore: number
awayScore: number
HTMLSpanElement
time: number = 0
lastTime: Date
setIntervall
lastTime = New Date()
HTMLSpanElement
class Ball extends Moveable
ctx.arc
ctx.fillStyle = "white"
ctx.strokeStyle = "black"
ctx.lineWidth = 1
class Arbitrator extends Moveable
ctx.arc
ctx.fillStyle = this.color
ctx.strokeStyle = "black"
ctx.lineWidth = "1"
class = score-pre
class = score-post
preColor.style
.background = v
postColor.style
.background = v
0 : 0
Zeit: 00:00
segments: number = 16
goalWidth: 7.32 * scale
goalArea: 5.50 * scale
penaltyArea = 11 * scale
lineWidth = 2
<canvas>
this.setWidth(100)
this.setHeight(75)
this.setPadding(10)
class Player extends Moveable
this.active = true
this.radius = 2
ctx.arc
ctx.fillStyle = this.color
ctx.strokeStyle = "black"
ctx.textAlign = "center"
ctx.fillStyle = "white"
preColor.style.background = v
postColor.style.background = v
action.radius: number = 30
class Linesman extends Moveable
ctx.arc
fillStyle = this.color
ctx.strokeStyle = "black"
cxc.lineWidth = 2
Anpfiff (Leertaste)
 <button id = "game-start"> </button>
<th>
Heimteam
Auswärtsteam
Farbe
<p>
<input> textfield to change trikot color of the team
v: string
m.setColor(v)
postColor.style.background = v
id = "team-body"
HTMLTableRowElement
HTMLSpanElement
!p.isActive() = players are not active
event: DragEvent
<input type checkbox id="toggle-player-radius">
Zeige Wahrnehmungsbereiche
Zeige Originalpositionen
id = "current-player"
Spielername
Player 7
Schusskraft
99
Präzision
1
Geschwindigkeit
1
UI:Helper.create Span
slider min (1) to max (99)
<input type checkbox id="toggle-player-origin">
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
99

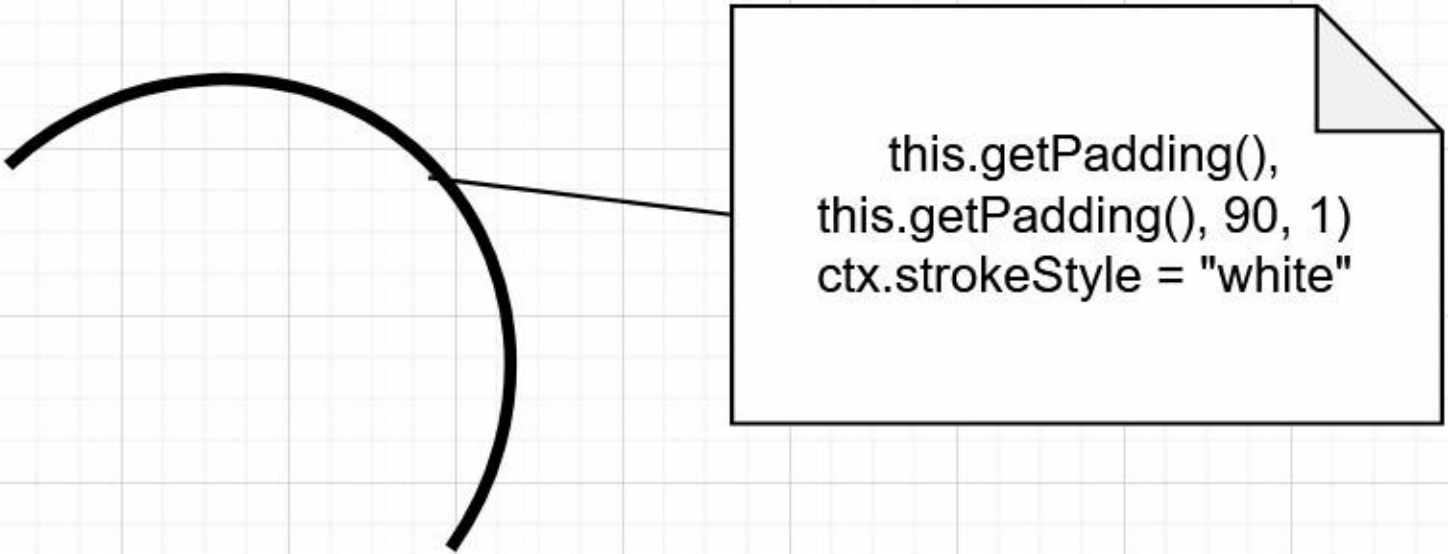

canvas: this.setWidth(100)
max-width = 600px
<canvas>
ctx.fillStyle = "darkgreen"
this.setPadding(10)
linesWidth = 2
ctx.strokeStyle = "white"
this.setHeight(75) - this.setPadding(10) * 2
6.25%
canvas:
this.setHeight(75)
max-height: 475px
ctx.fillStyle = i % 2 == 0 ? green
: lightgreen
this.setWidth(100) - this.setPadding(10) * 2

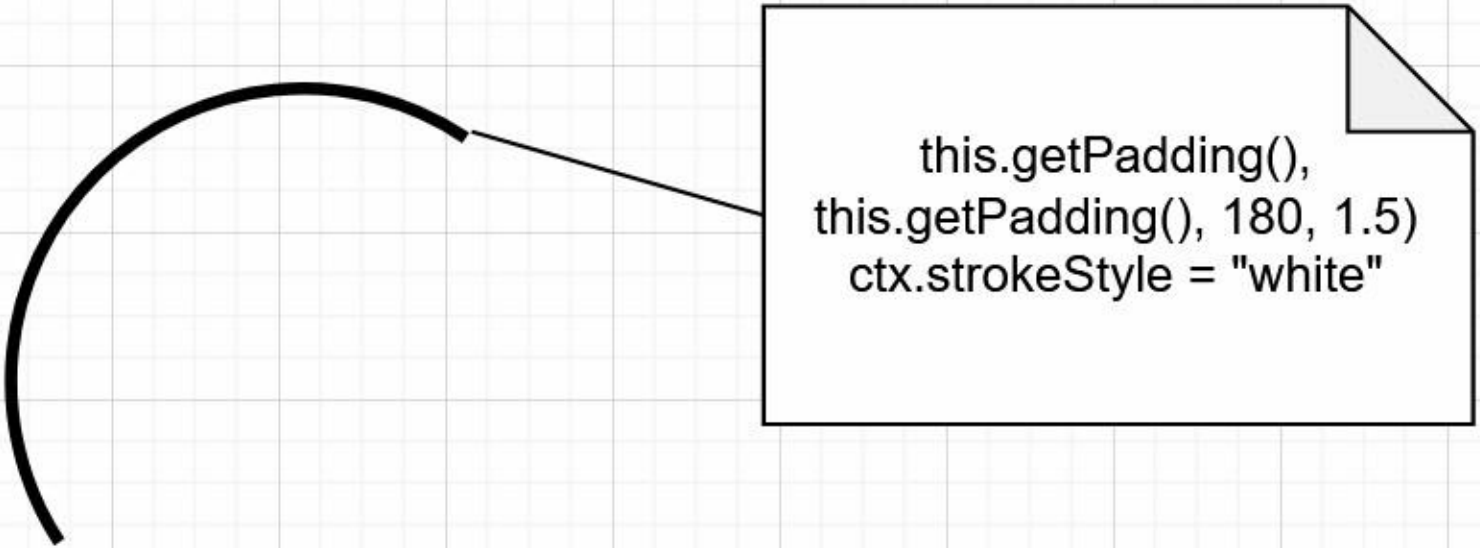

400px
46px
Anpfiff (Leertaste)
<button id = "game-start"> </button>
background: blue
border: 0
outline: none
padding: 15px
color: white

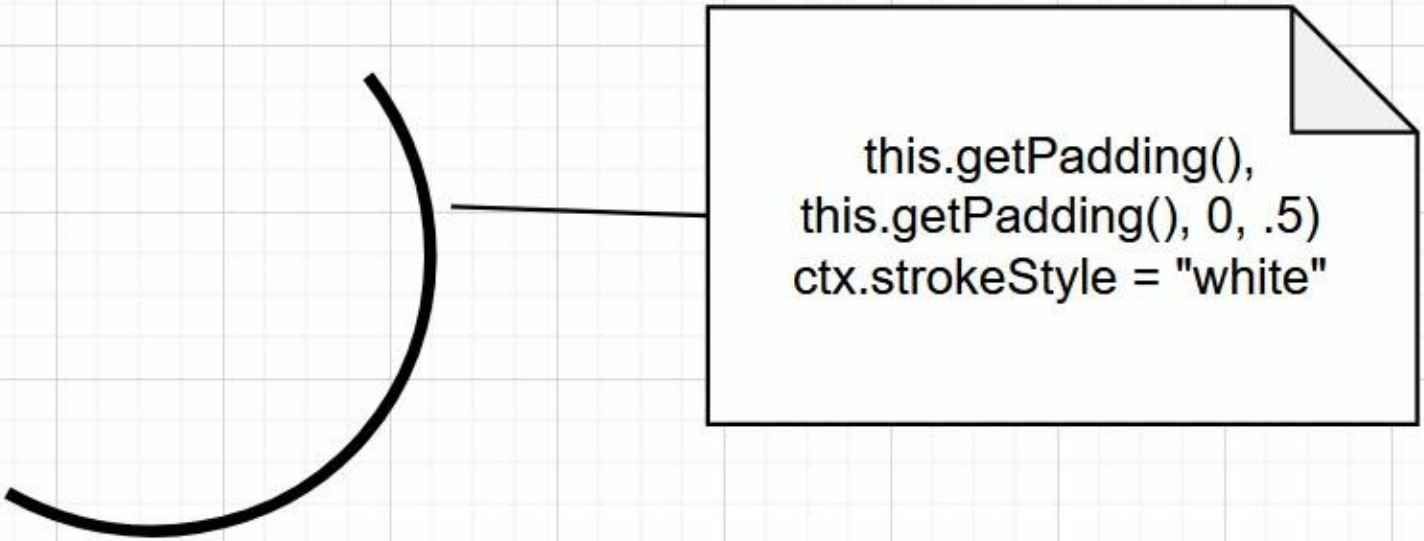

10px
10px
20px
20px

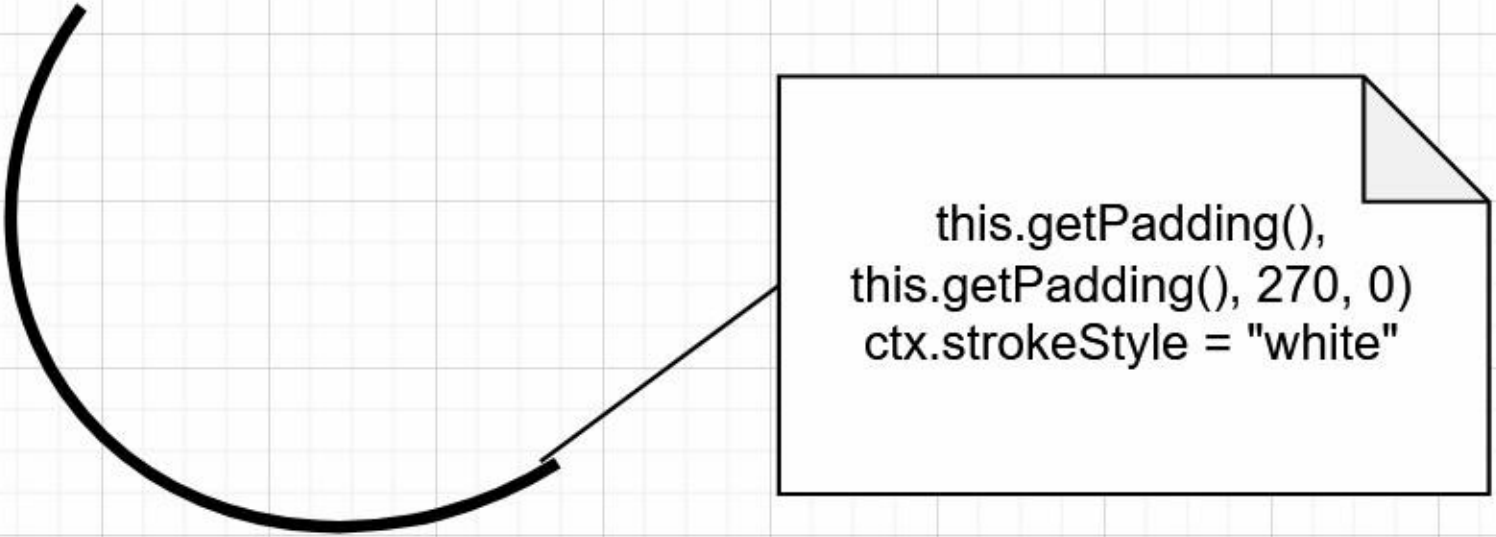

goalWidth = 7.32 * scale
13.5 * scale
lineWidth = 2
ctx.strokeStyle: "white"

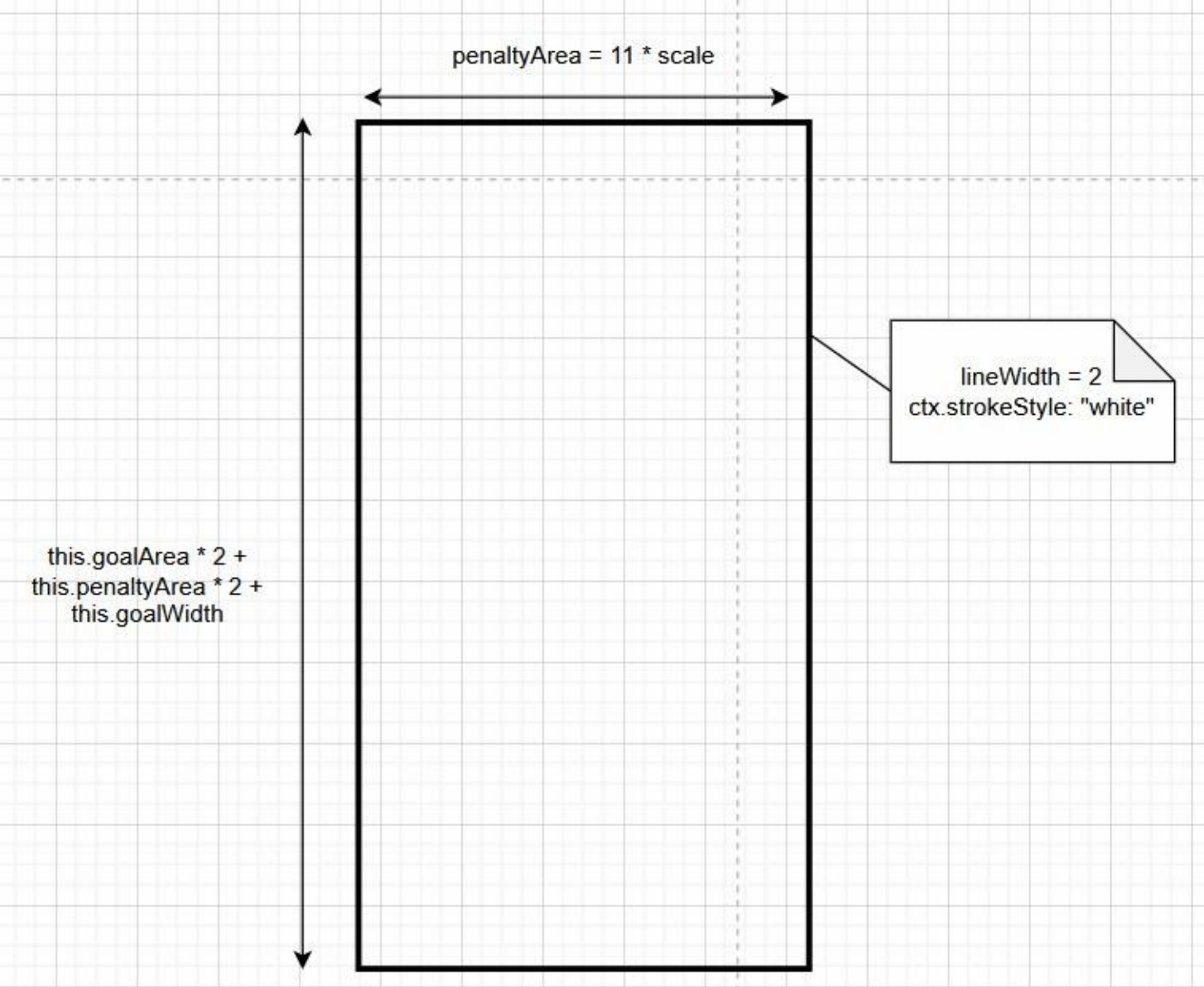

goalArea = 5.50 * scale
lineWidth = 2
ctx.strokeStyle: "white"
this.goalArea * 2 +
this.goalWidth

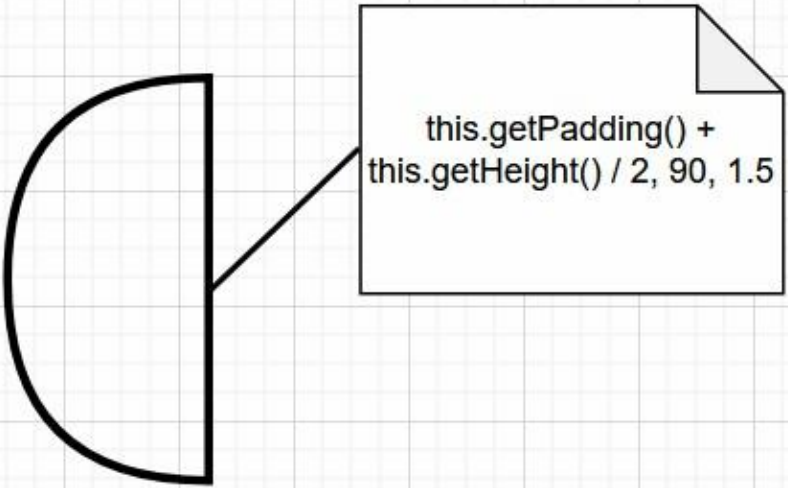

this.getPadding(),
this.getPadding(), 90, 1)
ctx.strokeStyle = "white"

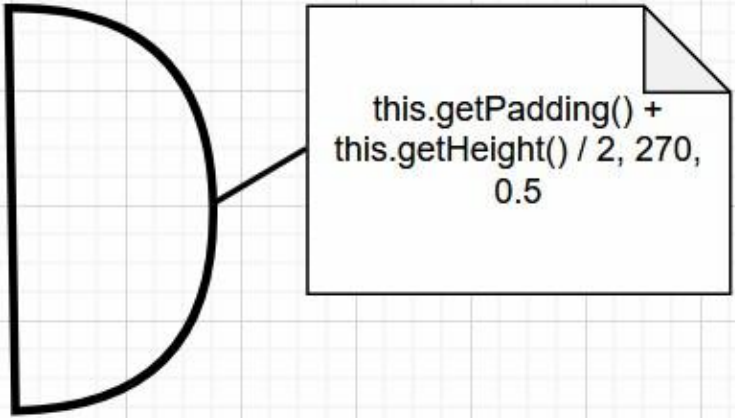

this.getPadding(),
this.getPadding(), 180, 1.5)
ctx.strokeStyle = "white"

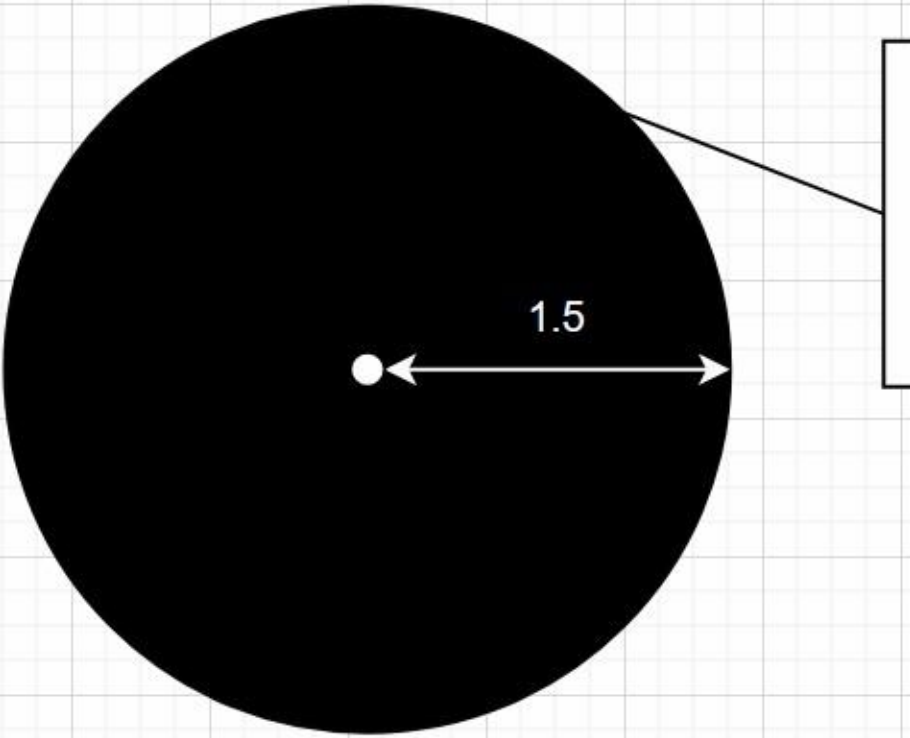

this.getPadding(),
this.getPadding(), 0, .5)
ctx.strokeStyle = "white"

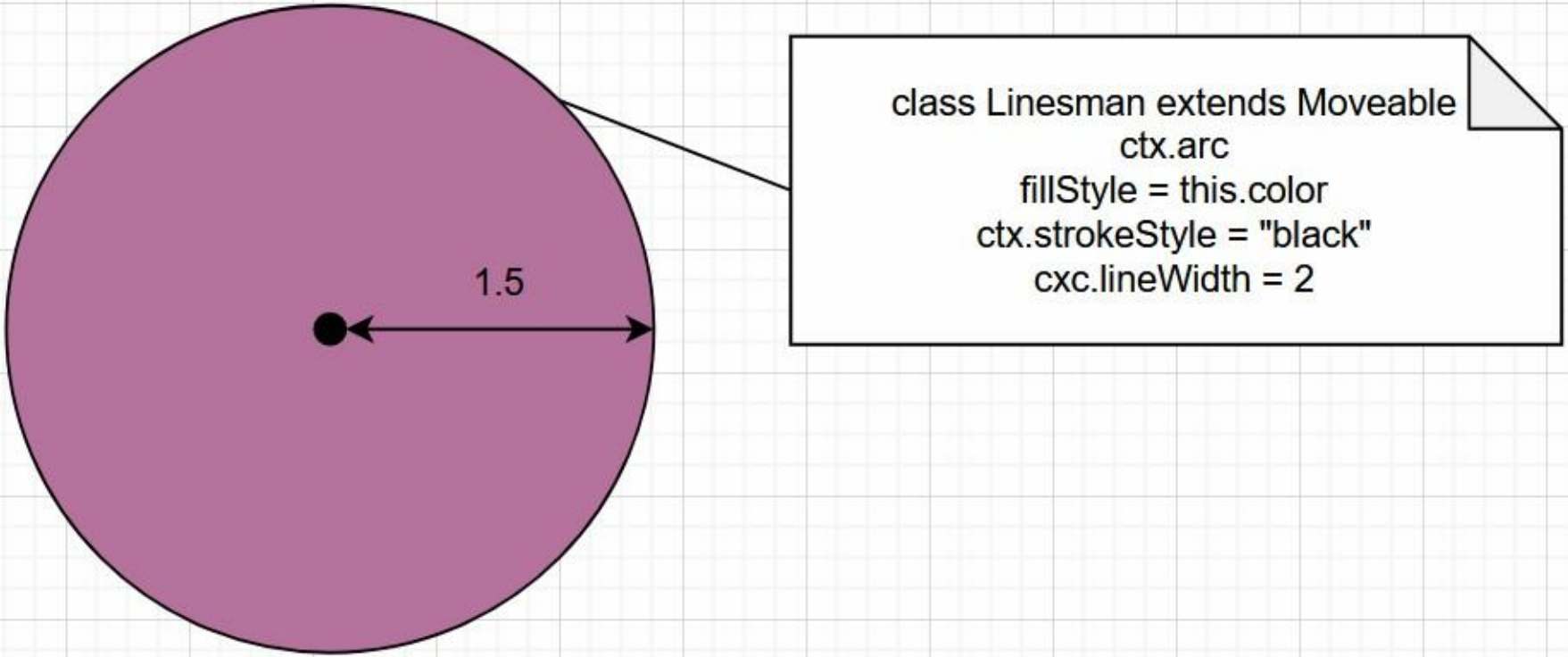

this.getPadding(),
this.getPadding(), 270, 0)
ctx.strokeStyle = "white"

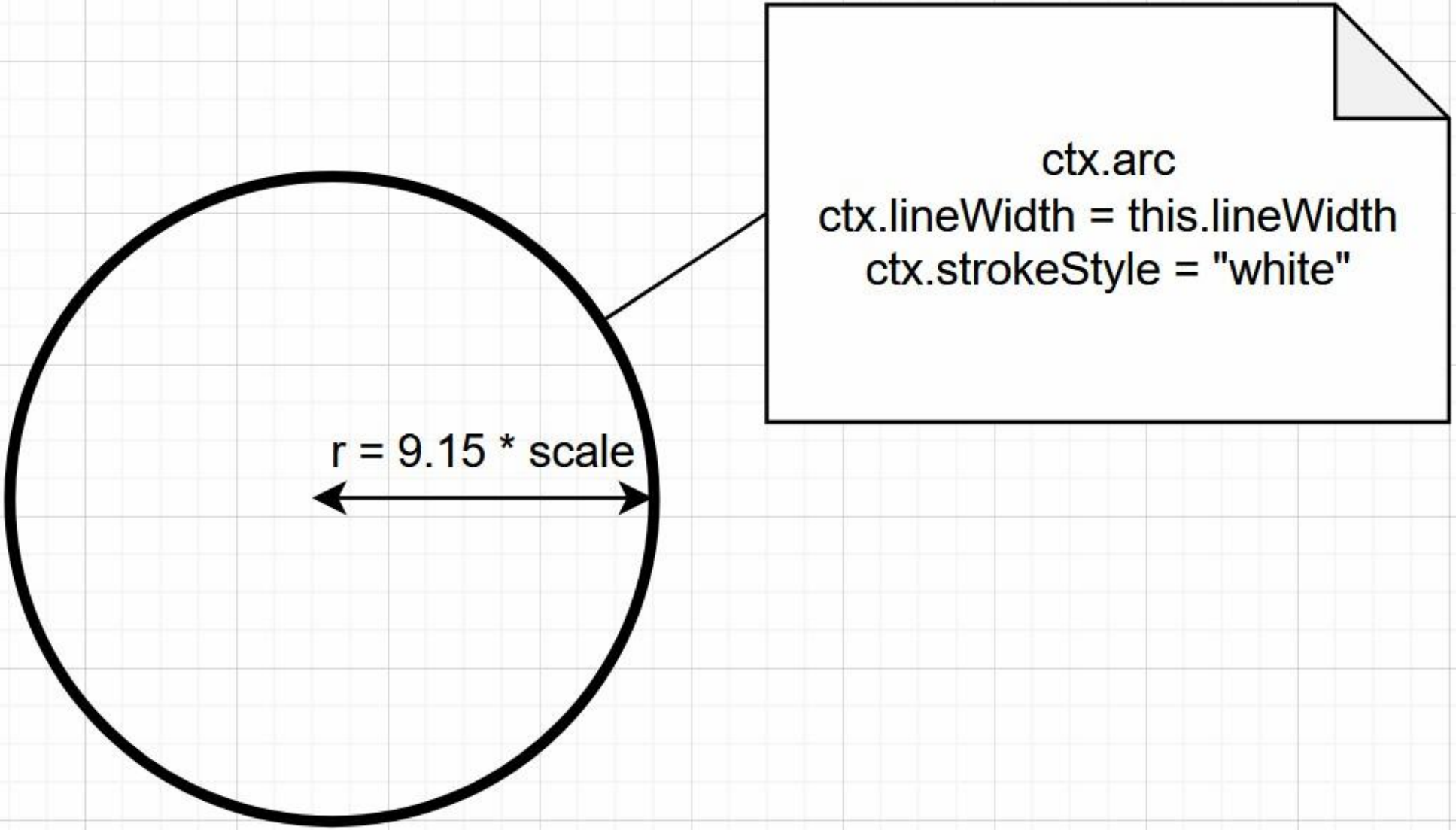

penaltyArea = 11 * scale
lineWidth = 2
ctx.strokeStyle: "white"
this.goalArea * 2 +
this.penaltyArea * 2 +
this.goalWidth

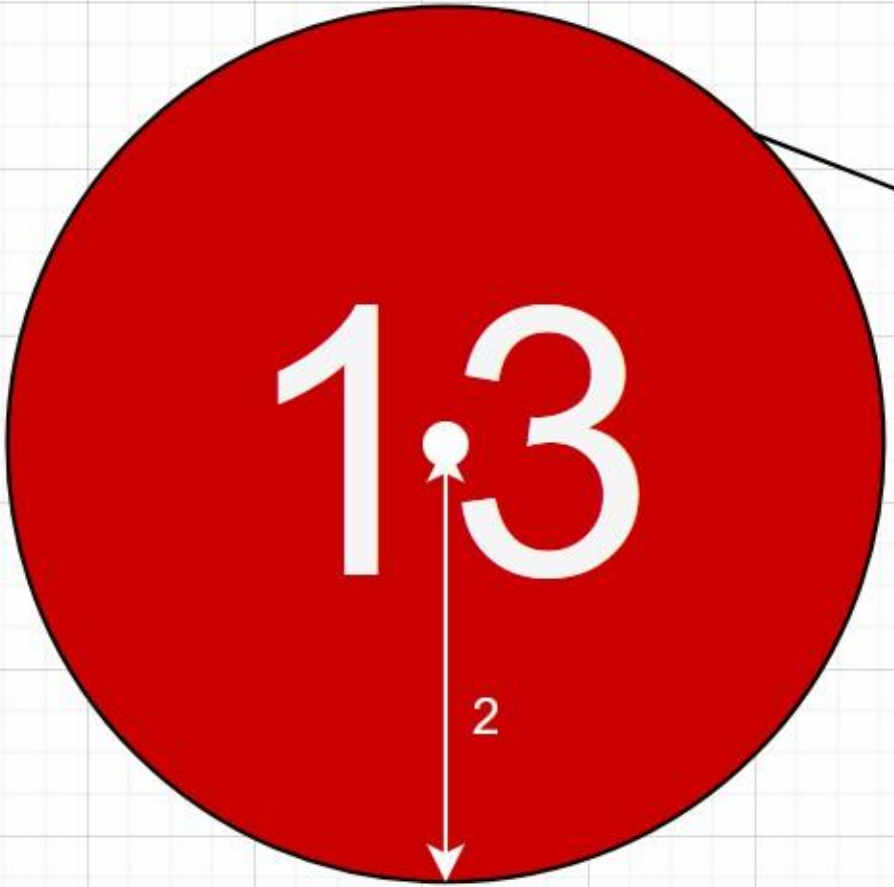

this.getPadding() +
this.getHeight() / 2, 90, 1.5


this.getPadding() +
this.getHeight() / 2, 270,
0.5


1.5

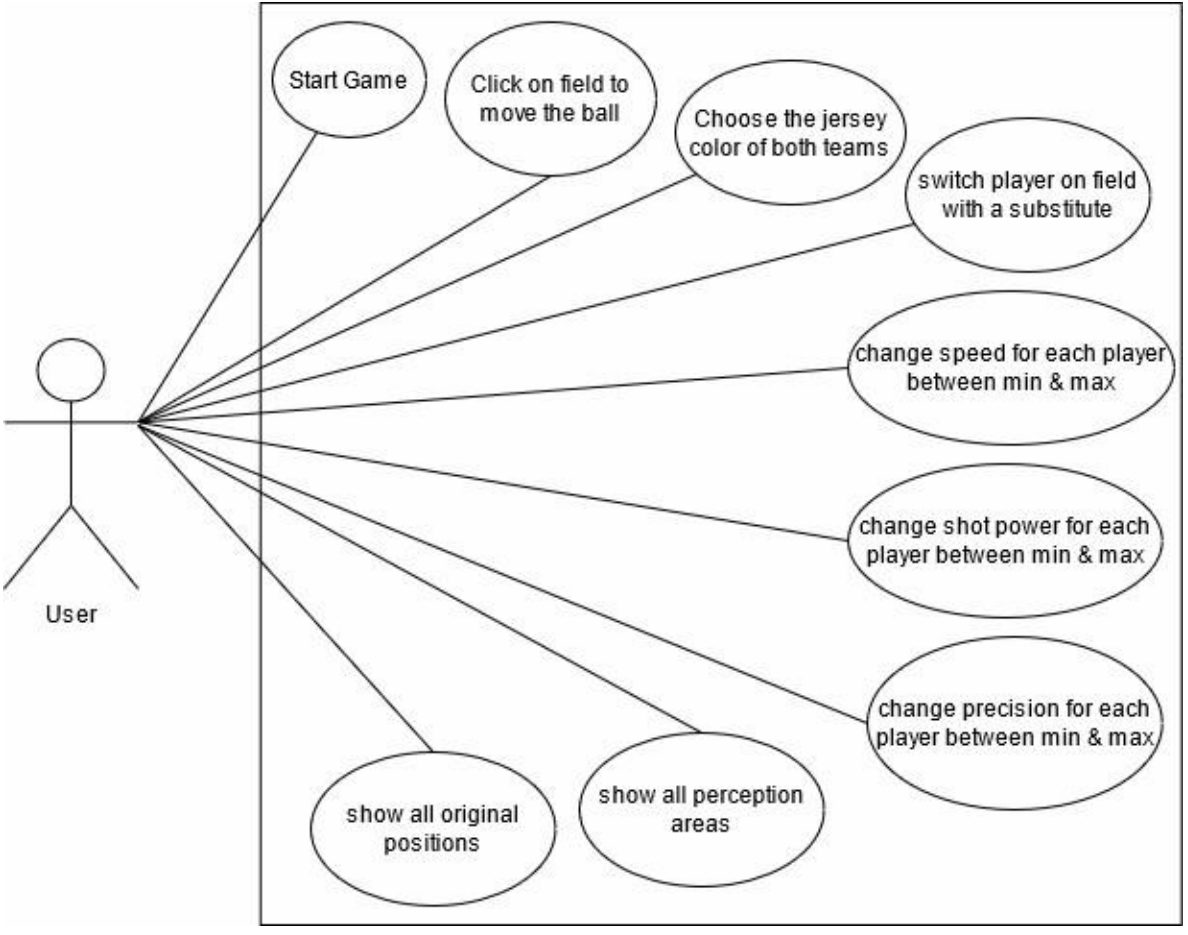

1.5
class Linesman extends Moveable
ctx.arc
fillStyle = this.color
ctx.strokeStyle = "black"
cxc.lineWidth = 2


ctx.arc
ctx.lineWidth = this.lineWidth
ctx.strokeStyle = "white"
r = 9.15 * scale

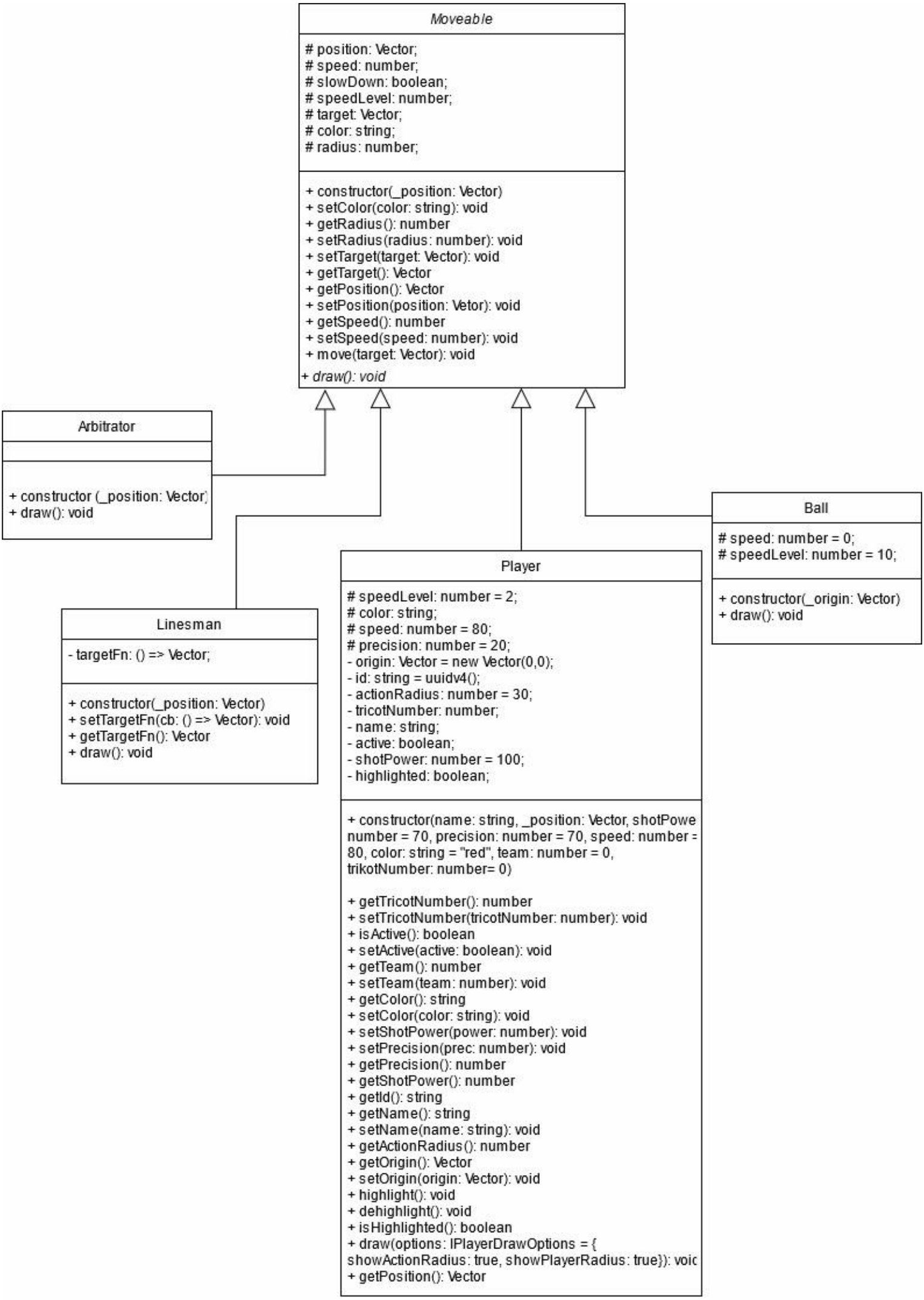



```
class Player extends Moveable
ctx.arc
ctx.fillStyle = this.color
ctx.strokeStyle = "black"
ctx.textAlign = "center"
ctx.fillStyle = "white"
```

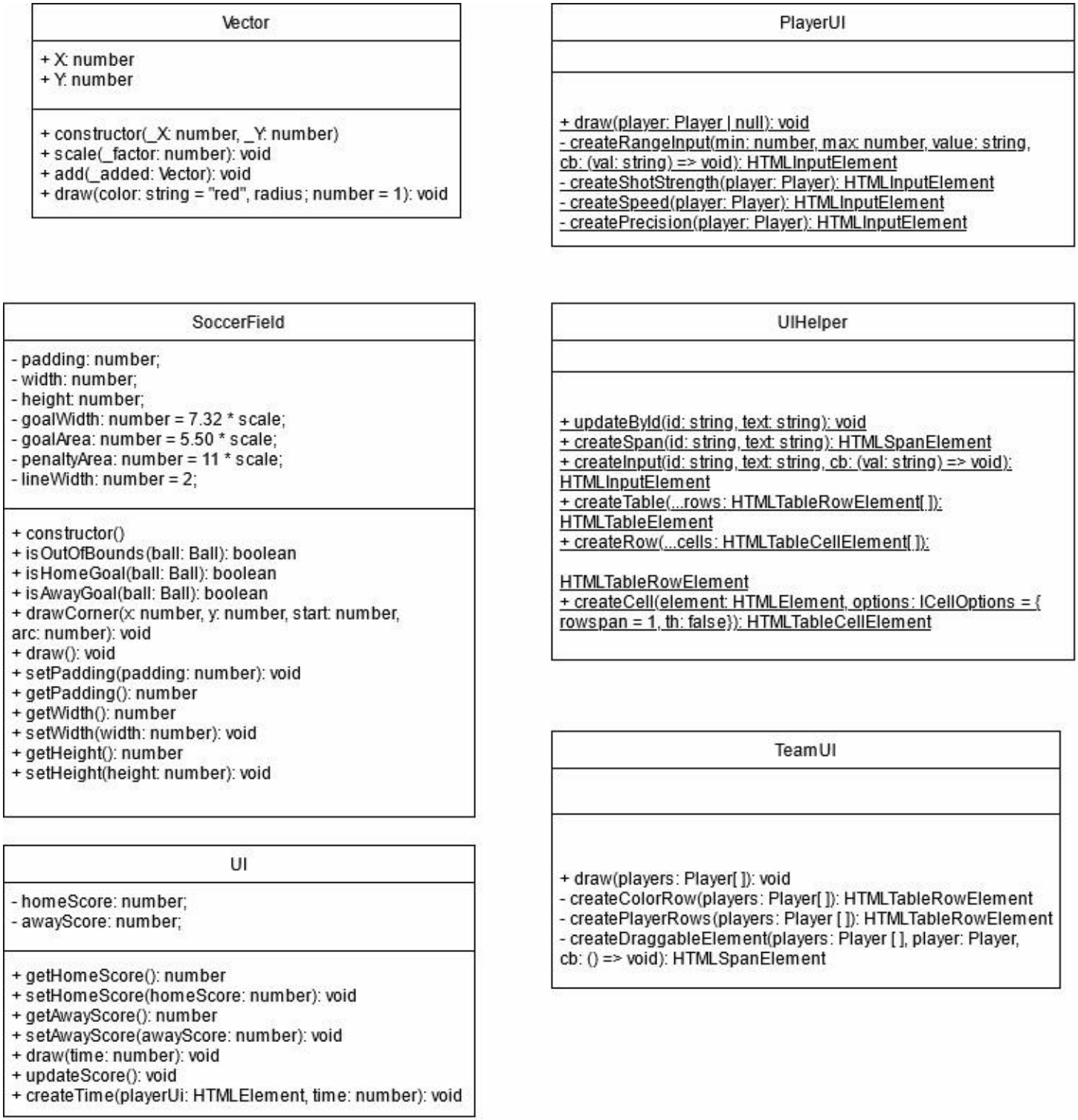
Use-Case Diagram



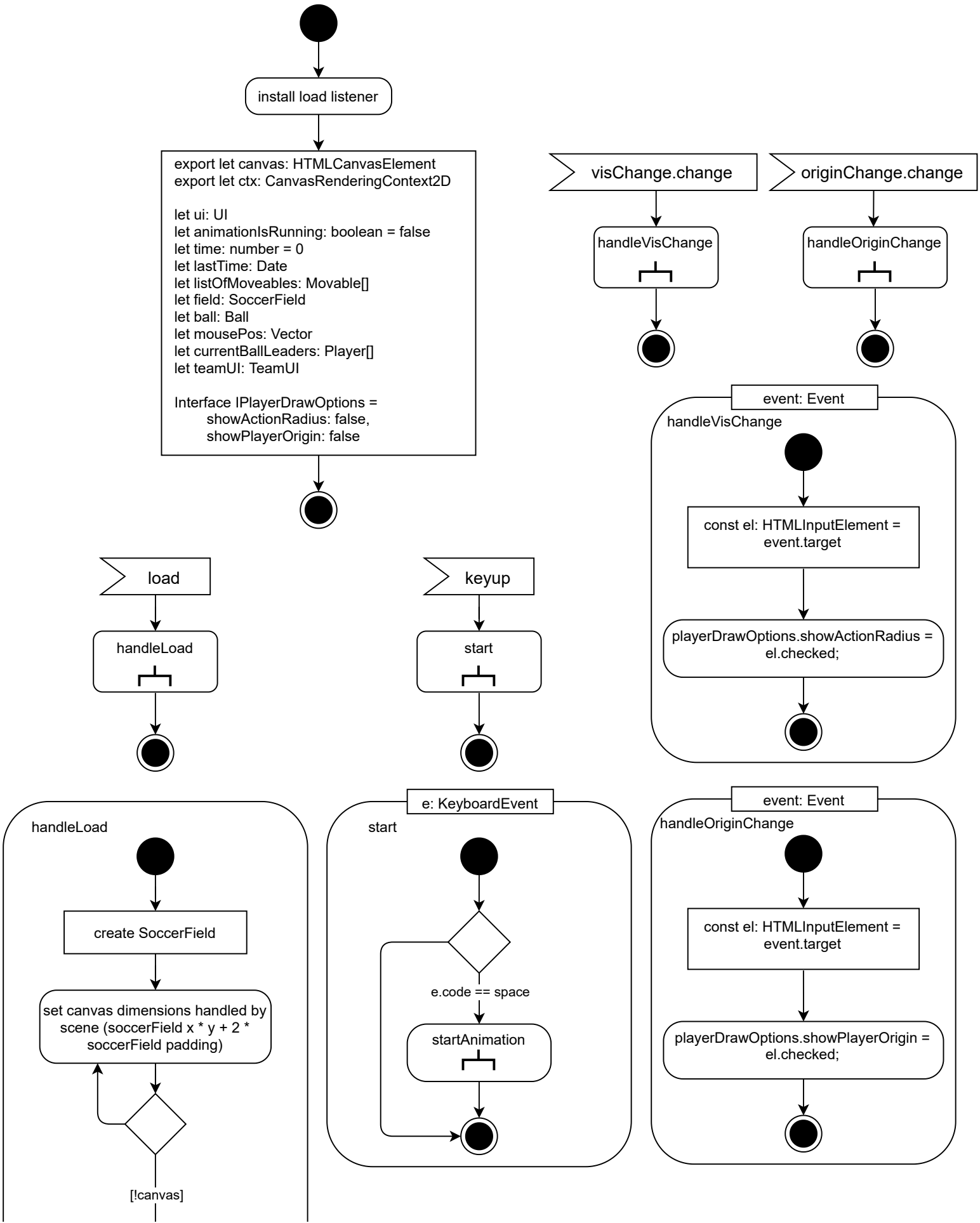
Class Diagram: Moveable.ts

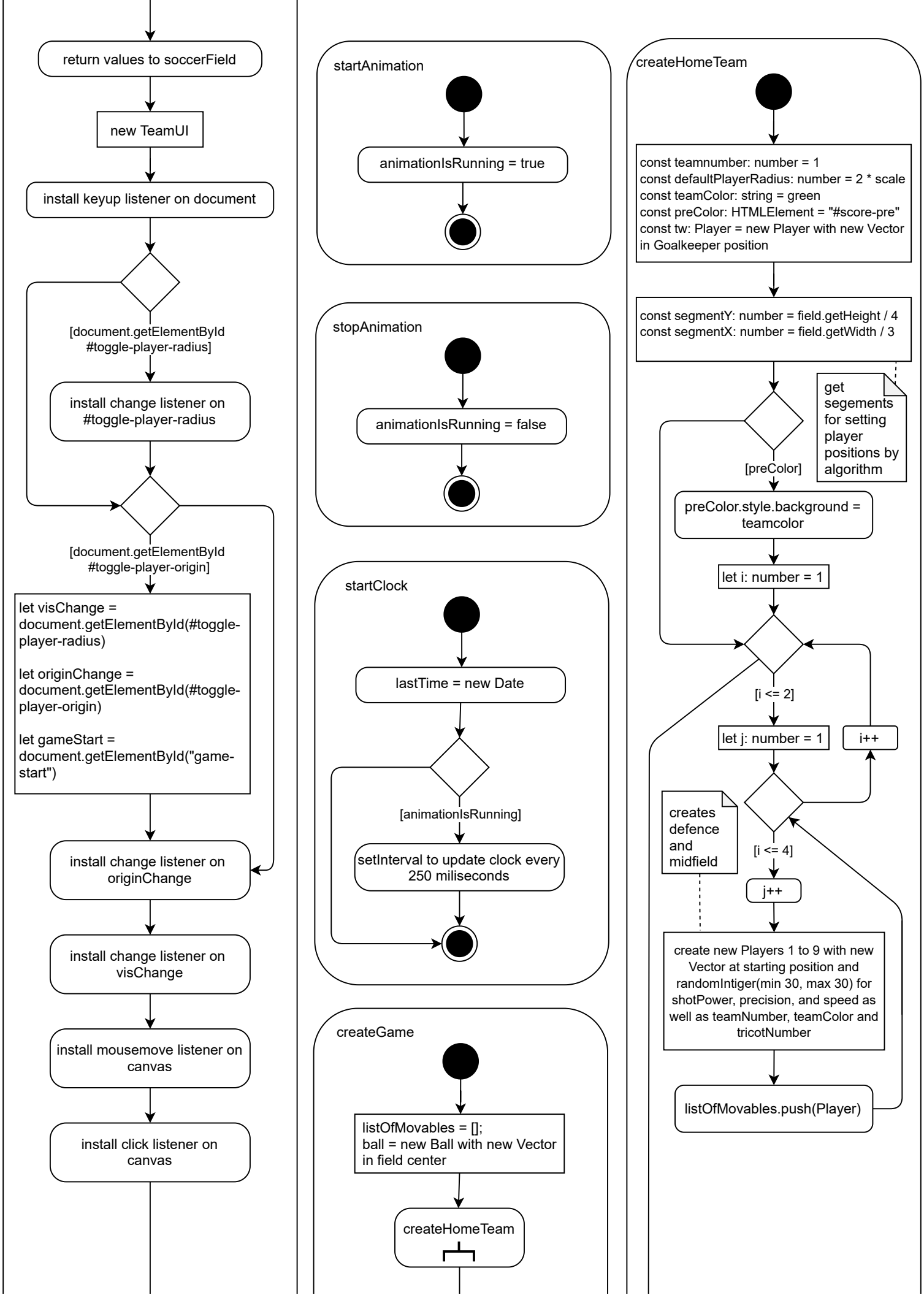


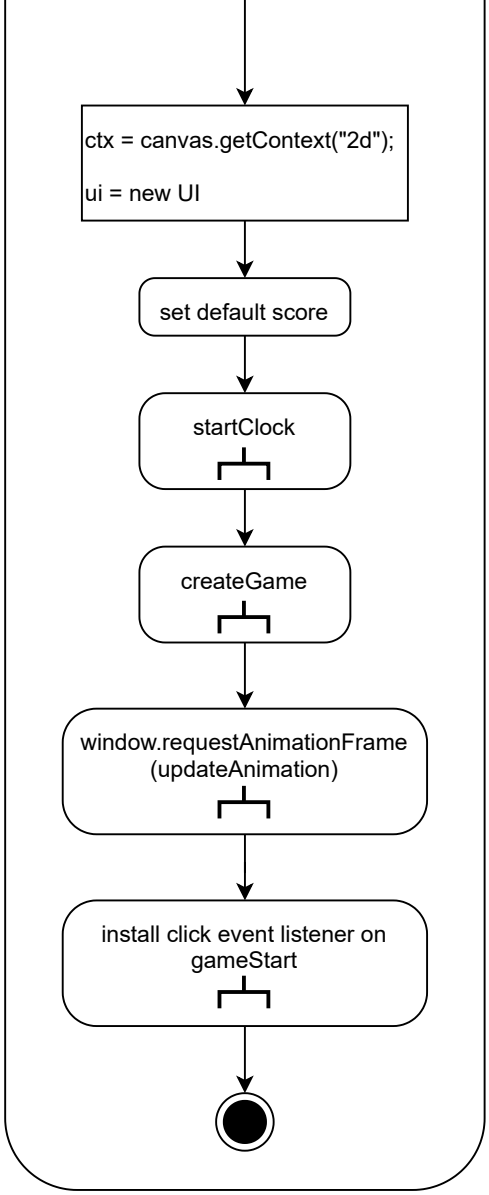
Class Diagram: Rest

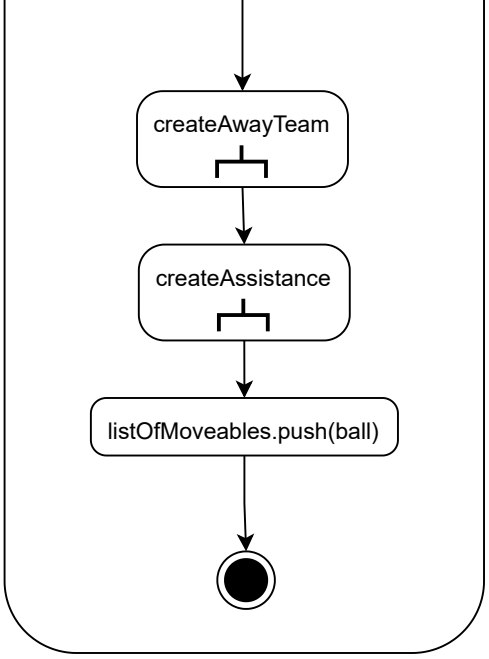


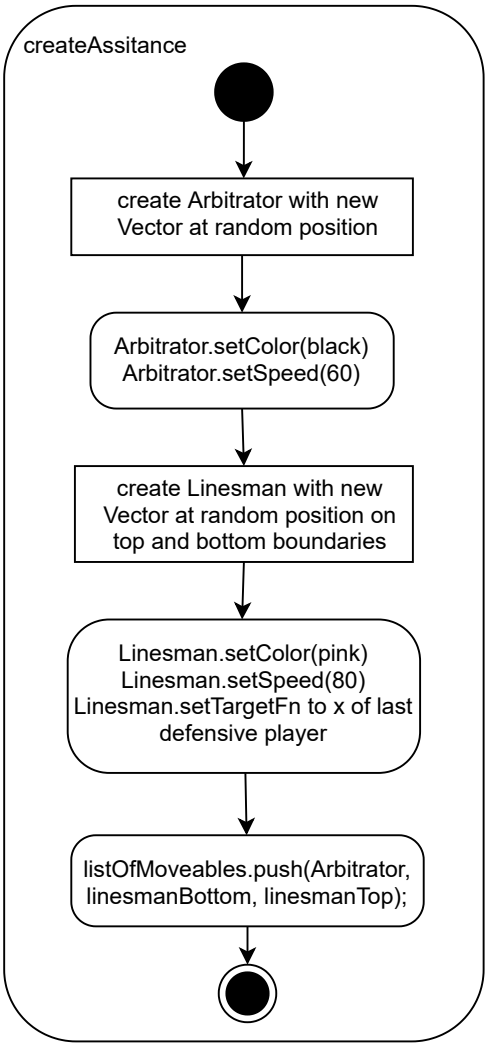
main.ts

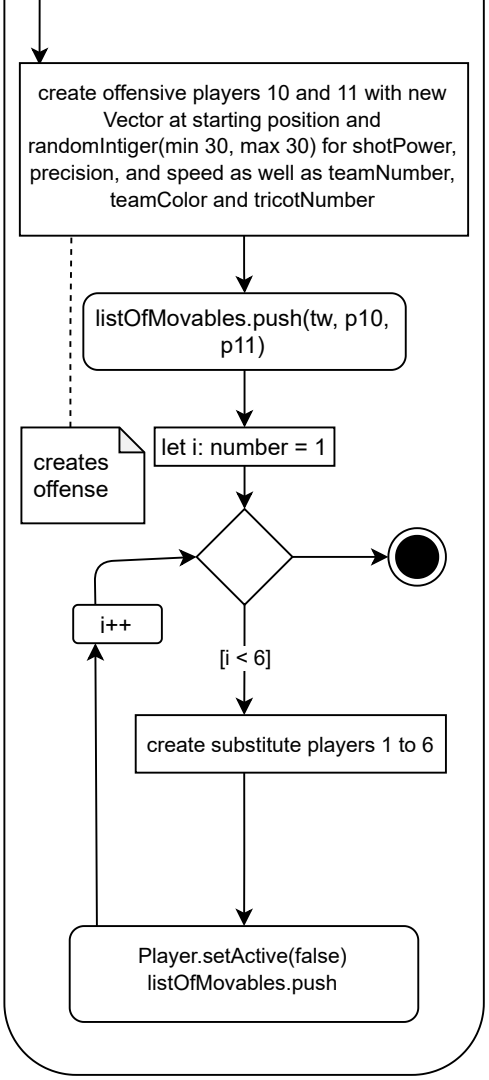

install load listener
export let canvas: HTMLCanvasElement
export let ctx: CanvasRenderingContext2D
let ui: UI
let animationIsRunning: boolean = false
let time: number = 0
let lastTime: Date
let listOfMoveables: Movable[]
let field: SoccerField
let ball: Ball
let mousePos: Vector
let currentBallLeaders: Player[]
let teamUI: TeamUI
Interface IPlayerDrawOptions =
showActionRadius: false,
showPlayerOrigin: false
visChange.change
handleVisChange
originChange.change
handleOriginChange
event: Event
handleVisChange
const el: HTMLInputElement = event.target
playerDrawOptions.showActionRadius = el.checked;
load
handleLoad
keyup
start
handleLoad
create SoccerField
set canvas dimensions handled by scene (soccerField x * y + 2 * soccerField padding)
[!canvas]
e: KeyboardEvent
start
e.code == space
startAnimation
event: Event
handleOriginChange
const el: HTMLInputElement = event.target
playerDrawOptions.showPlayerOrigin = el.checked;

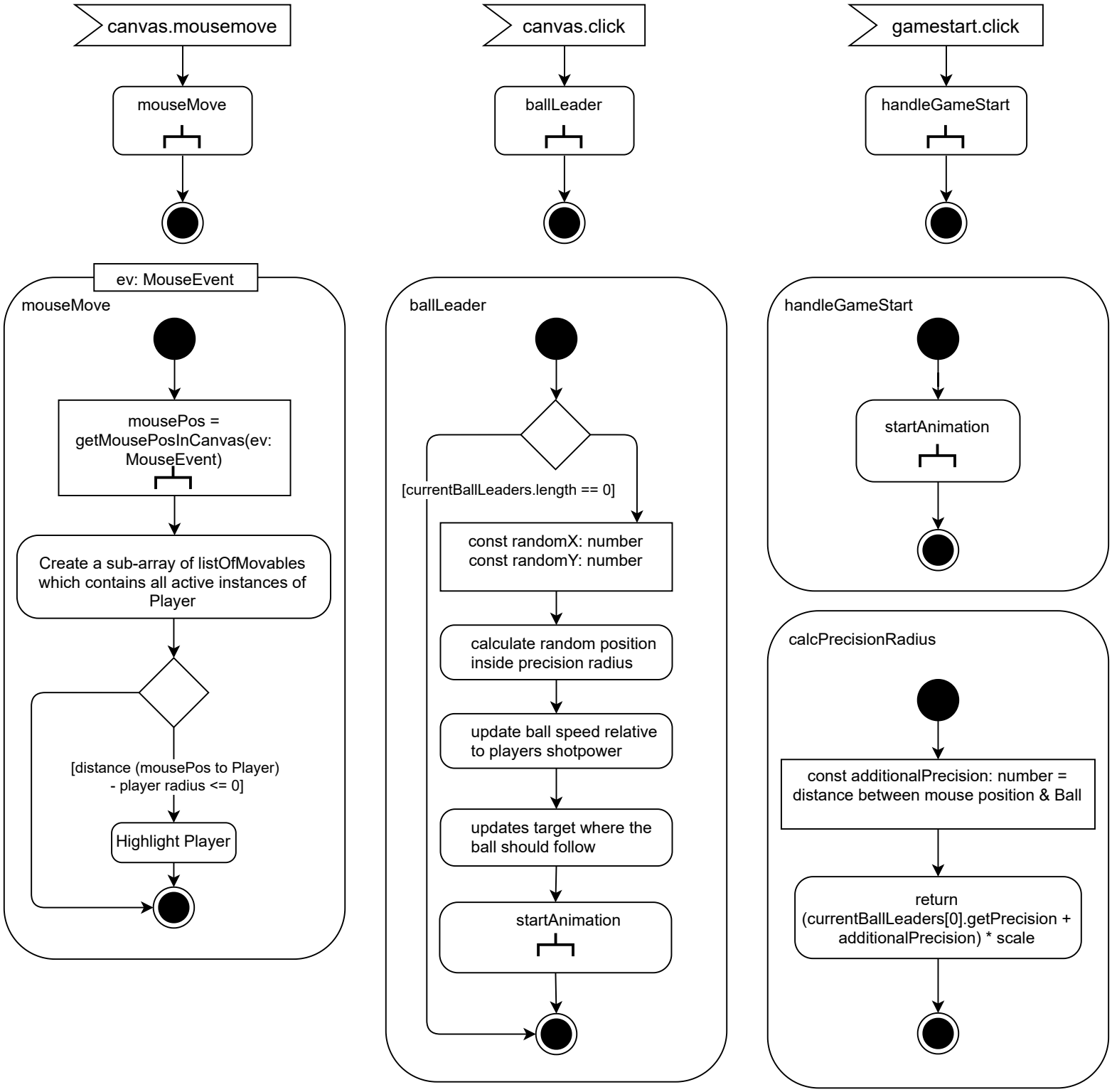

return values to soccerField
new TeamUI
install keyup listener on document
[document.getElementById #toggle-player-radius]
install change listener on #toggle-player-radius
[document.getElementById #toggle-player-origin]
let visChange = document.getElementById(#toggle-player-radius)
let originChange = document.getElementById(#toggle-player-origin)
let gameStart = document.getElementById("game-start")
install change listener on originChange
install change listener on visChange
install mousemove listener on canvas
install click listener on canvas
startAnimation
animationIsRunning = true
stopAnimation
animationIsRunning = false
startClock
lastTime = new Date
[animationIsRunning]
setInterval to update clock every 250 miliseconds
createGame
listOfMovables = [];
ball = new Ball with new Vector in field center
createHomeTeam
createHomeTeam
const teamnumber: number = 1
const defaultPlayerRadius: number = 2 * scale
const teamColor: string = green
const preColor: HTMLElement = "#score-pre"
const tw: Player = new Player with new Vector in Goalkeeper position
const segmentY: number = field.getHeight / 4
const segmentX: number = field.getWidth / 3
get segements for setting player positions by algorithm
[preColor]
preColor.style.background = teamcolor
let i: number = 1
[i <= 2]
let j: number = 1
i++
creates defence and midfield
[i <= 4]
j++
create new Players 1 to 9 with new Vector at starting position and randomIntiger(min 30, max 30) for shotPower, precision, and speed as well as teamNumber, teamColor and tricotNumber
listOfMovables.push(Player)

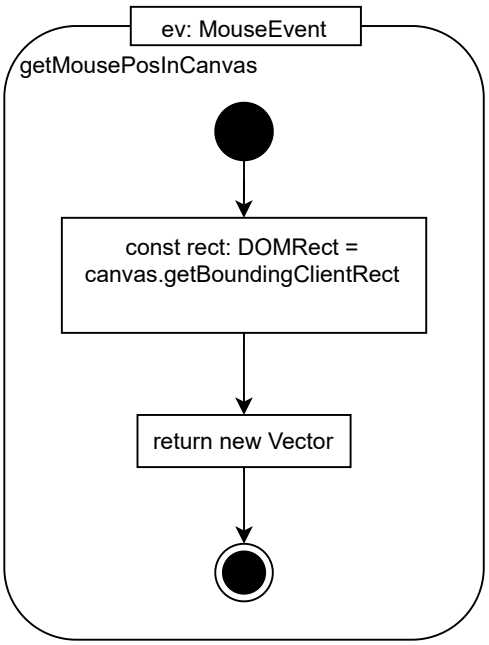

ctx = canvas.getContext("2d");
ui = new UI
set default score
startClock
createGame
window.requestAnimationFrame (updateAnimation)
install click event listener on gameStart

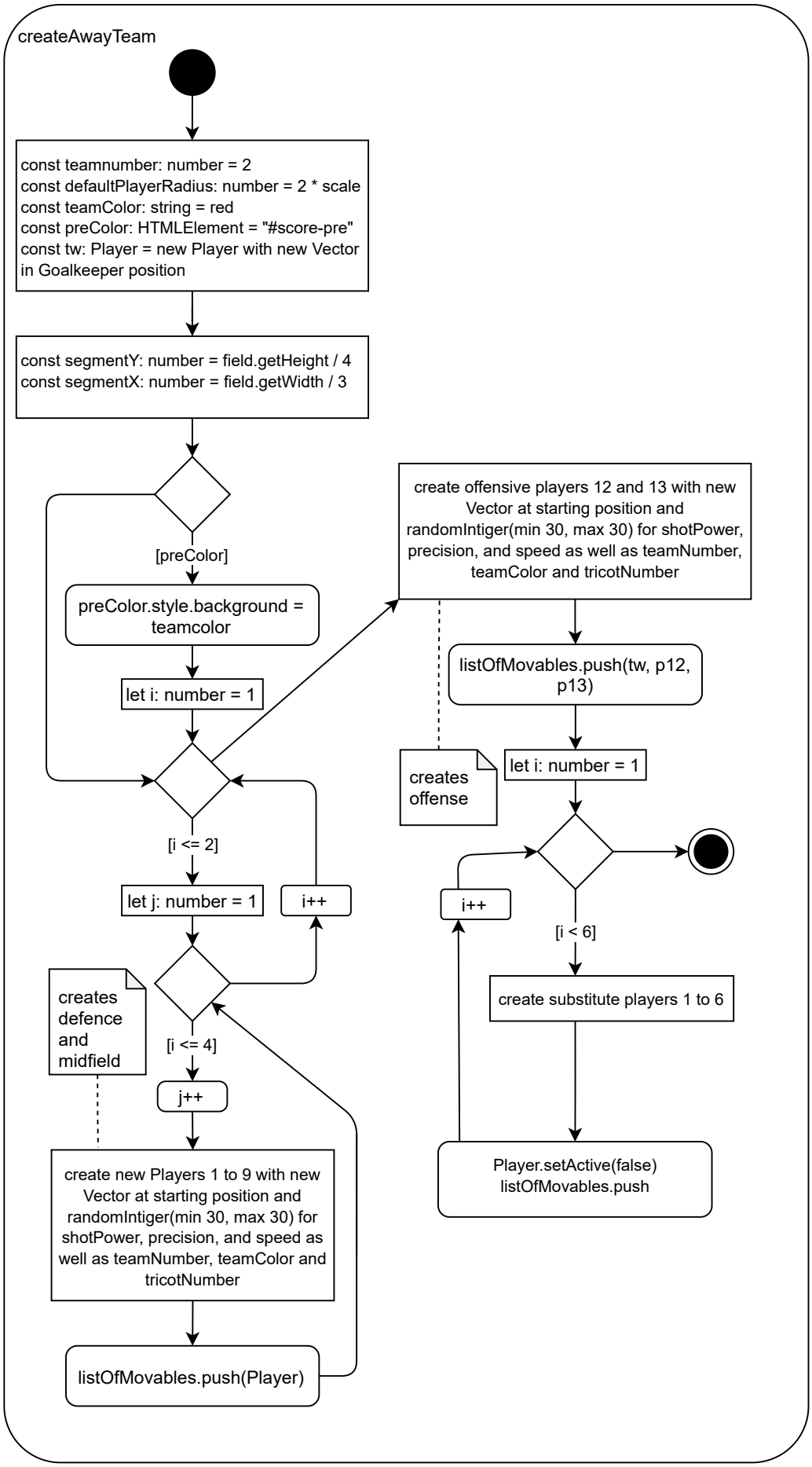

createAwayTeam
createAssistance
listOfMoveables.push(ball)

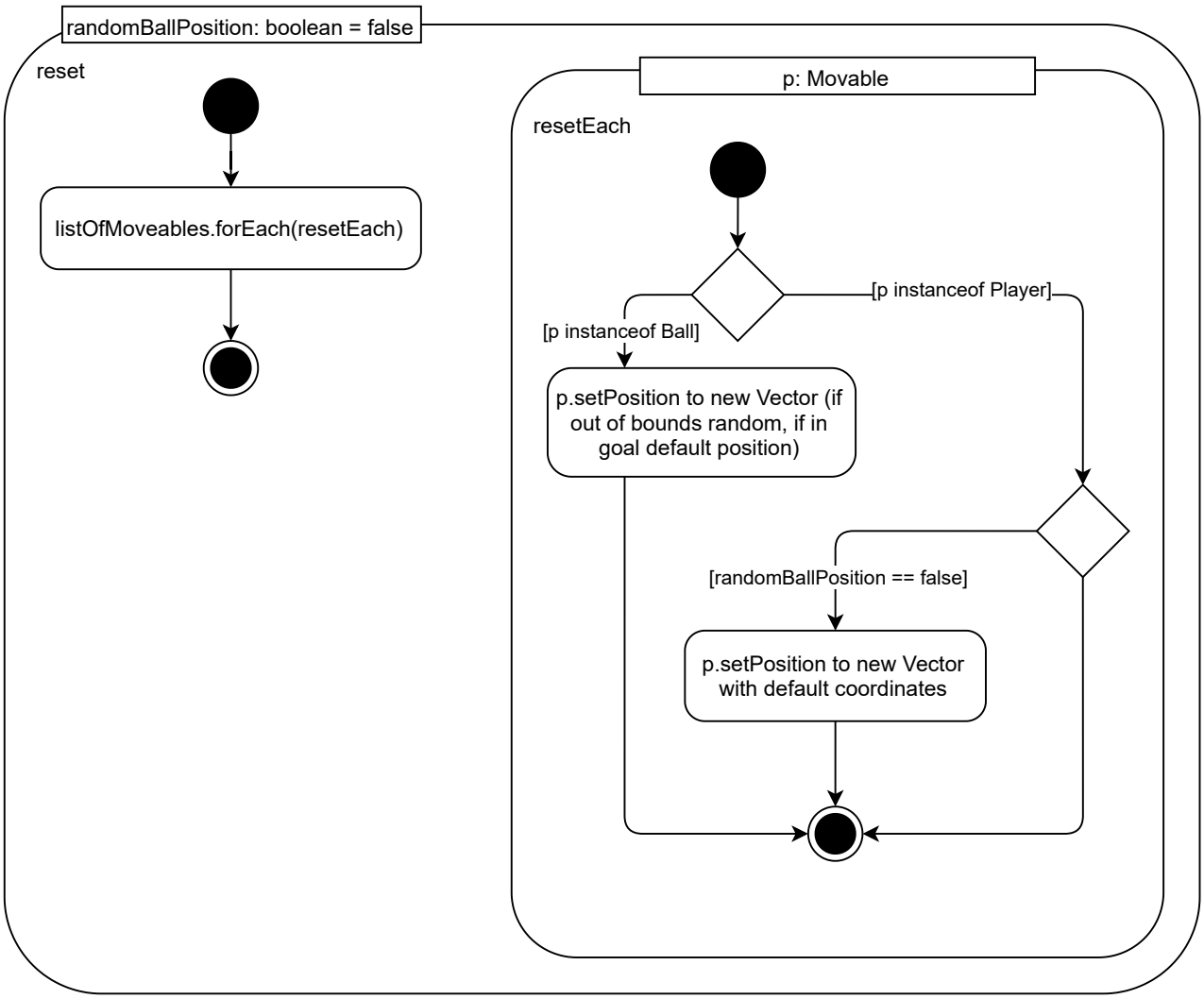

createAssitance
create Arbitrator with new Vector at random position
Arbitrator.setColor(black)
Arbitrator.setSpeed(60)
create Linesman with new Vector at random position on top and bottom boundaries
Linesman.setColor(pink)
Linesman.setSpeed(80)
Linesman.setTargetFn to x of last defensive player
listOfMoveables.push(Arbitrator, linesmanBottom, linesmanTop);

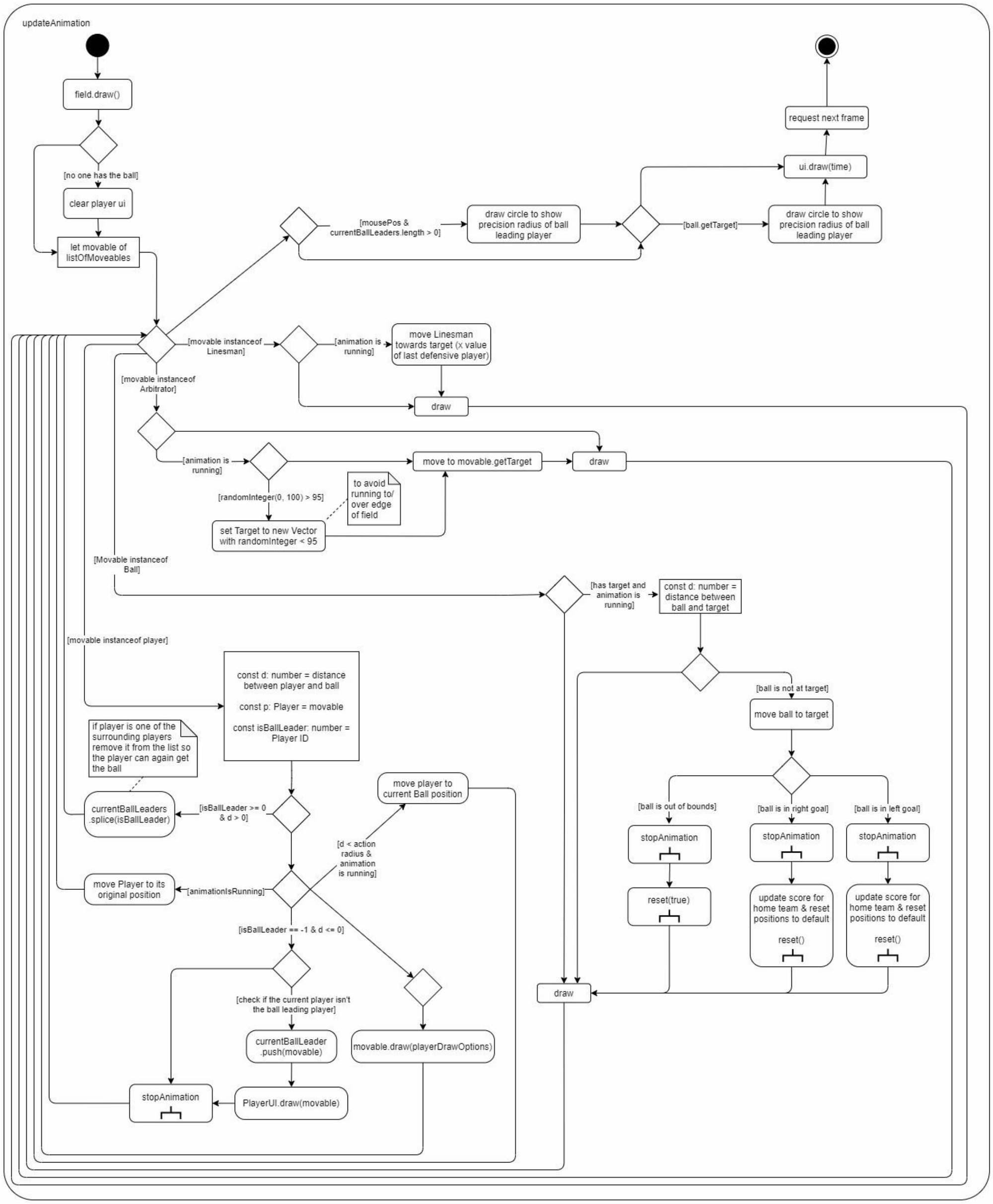

create offensive players 10 and 11 with new Vector at starting position and randomIntiger(min 30, max 30) for shotPower, precision, and speed as well as teamNumber, teamColor and tricotNumber
listOfMovables.push(tw, p10, p11)
creates offense
let i: number = 1
i++
[i < 6]
create substitute players 1 to 6
Player.setActive(false)
listOfMovables.push


canvas.mousemove
mouseMove
canvas.click
ballLeader
gamestart.click
handleGameStart
ev: MouseEvent
mouseMove
mousePos = getMousePosInCanvas(ev: MouseEvent)
Create a sub-array of listOfMovables which contains all active instances of Player
[distance (mousePos to Player) - player radius <= 0]
Highlight Player
ballLeader
[currentBallLeaders.length == 0]
const randomX: number
const randomY: number
calculate random position inside precision radius
update ball speed relative to players shotpower
updates target where the ball should follow
startAnimation
handleGameStart
startAnimation
calcPrecisionRadius
const additionalPrecision: number = distance between mouse position & Ball
return (currentBallLeaders[0].getPrecision + additionalPrecision) * scale

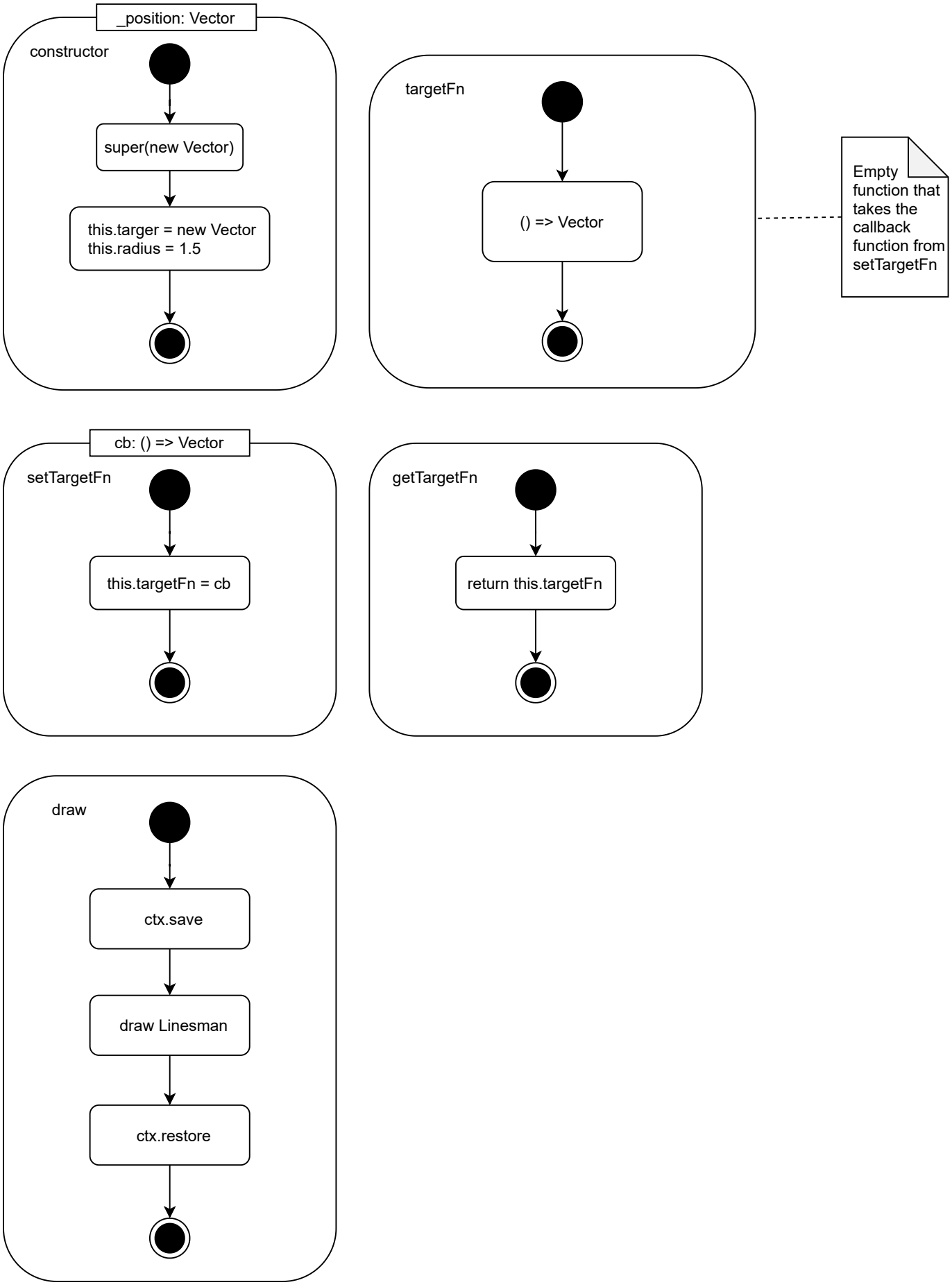

ev: MouseEvent
getMousePosInCanvas
const rect: DOMRect = canvas.getBoundingClientRect
return new Vector


createAwayTeam
const teamnumber: number = 2
const defaultPlayerRadius: number = 2 * scale
const teamColor: string = red
const preColor: HTMLElement = "#score-pre"
const tw: Player = new Player with new Vector in Goalkeeper position
const segmentY: number = field.getHeight / 4
const segmentX: number = field.getWidth / 3
[preColor]
preColor.style.background = teamcolor
let i: number = 1
[i <= 2]
let j: number = 1
i++
creates defence and midfield
[i <= 4]
j++
create new Players 1 to 9 with new Vector at starting position and randomIntiger(min 30, max 30) for shotPower, precision, and speed as well as teamNumber, teamColor and tricotNumber
listOfMovables.push(Player)
create offensive players 12 and 13 with new Vector at starting position and randomIntiger(min 30, max 30) for shotPower, precision, and speed as well as teamNumber, teamColor and tricotNumber
listOfMovables.push(tw, p12, p13)
creates offense
let i: number = 1
i++
[i < 6]
create substitute players 1 to 6
Player.setActive(false)
listOfMovables.push

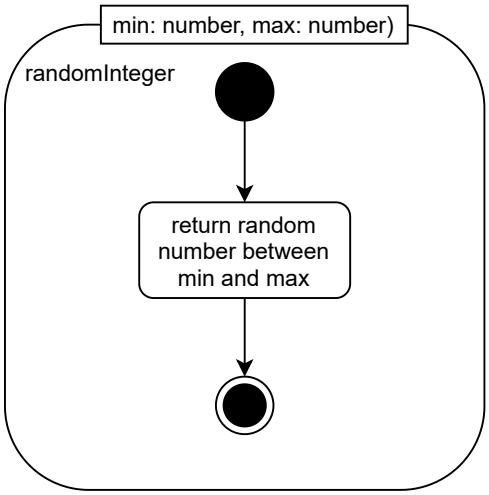

randomBallPosition: boolean = false
reset
listOfMoveables.forEach(resetEach)
p: Movable
resetEach
[p instanceof Ball]
[p instanceof Player]
p.setPosition to new Vector (if out of bounds random, if in goal default position)
[randomBallPosition == false]
p.setPosition to new Vector with default coordinates

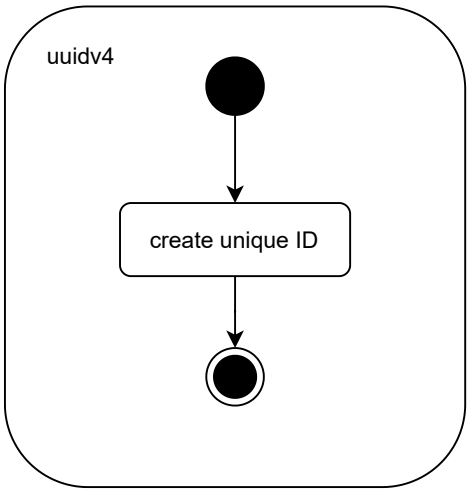

updateAnimation
field.draw()
[no one has the ball]
clear player ui
let movable of listOfMoveables
[mousePos & currentBallLeaders.length > 0]
draw circle to show precision radius of ball leading player
[ball.getTarget]
draw circle to show precision radius of ball leading player
ui.draw(time)
request next frame
[movable instanceof Linesman]
[animation is running]
move Linesman towards target (x value of last defensive player)
draw
[movable instanceof Arbitrator]
[animation is running]
move to movable.getTarget
draw
[randomInteger(0, 100) > 95]
to avoid running to/ over edge of field
set Target to new Vector with randomInteger < 95
[Movable instanceof Ball]
[has target and animation is running]
const d: number = distance between ball and target
[ball is not at target]
move ball to target
[ball is out of bounds]
[ball is in right goal]
[ball is in left goal]
stopAnimation
stopAnimation
stopAnimation
reset(true)
update score for home team & reset positions to default
reset()
update score for home team & reset positions to default
reset()
draw
[movable instanceof player]
const d: number = distance between player and ball
const p: Player = movable
const isBallLeader: number = Player ID
if player is one of the surrounding players remove it from the list so the player can again get the ball
currentBallLeaders .splice(isBallLeader)
[isBallLeader >= 0 & d > 0]
move player to current Ball position
[d < action radius & animation is running]
move Player to its original position
[animationIsRunning]
[isBallLeader == -1 & d <= 0]
[check if the current player isn't the ball leading player]
currentBallLeader .push(movable)
movable.draw(playerDrawOptions)
stopAnimation
PlayerUI.draw(movable)

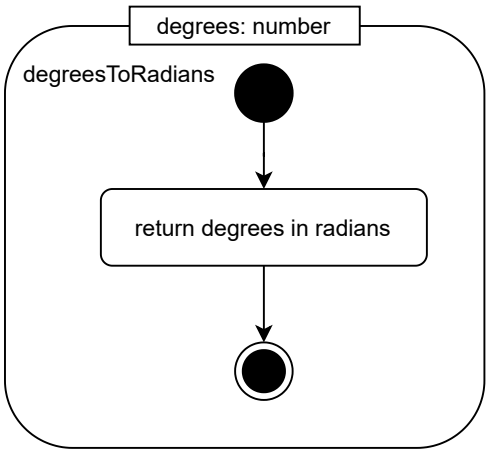
linesman.ts

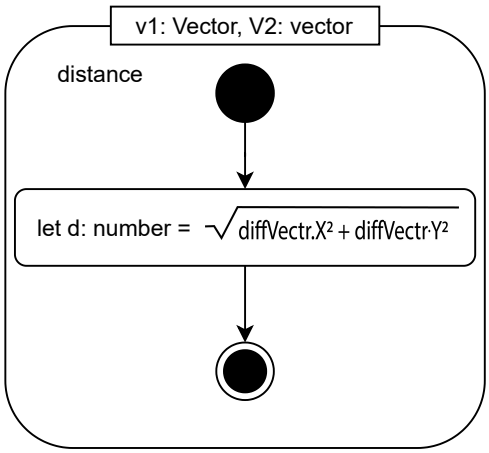

_position: Vector
constructor
super(new Vector)
this.targer = new Vector
this.radius = 1.5
targetFn
() => Vector
Empty function that takes the callback function from setTargetFn
cb: () => Vector
setTargetFn
this.targetFn = cb
getTargetFn
return this.targetFn
draw
ctx.save
draw Linesman
ctx.restore

global.ts

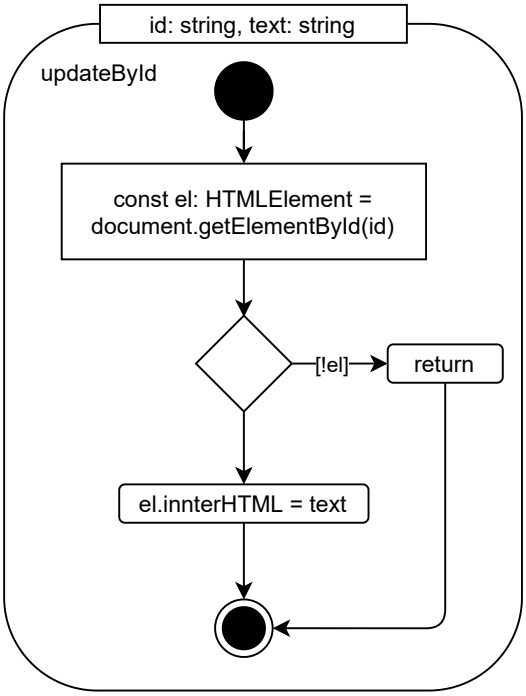

min: number, max: number)
randomInteger
return random number between min and max

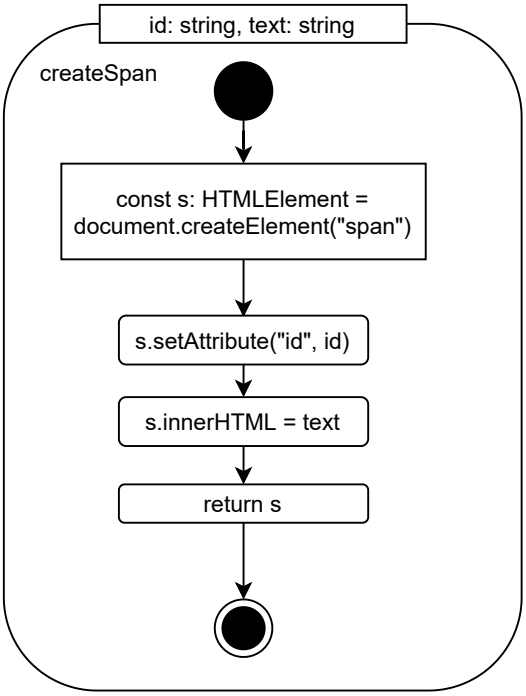

uuidv4
create unique ID

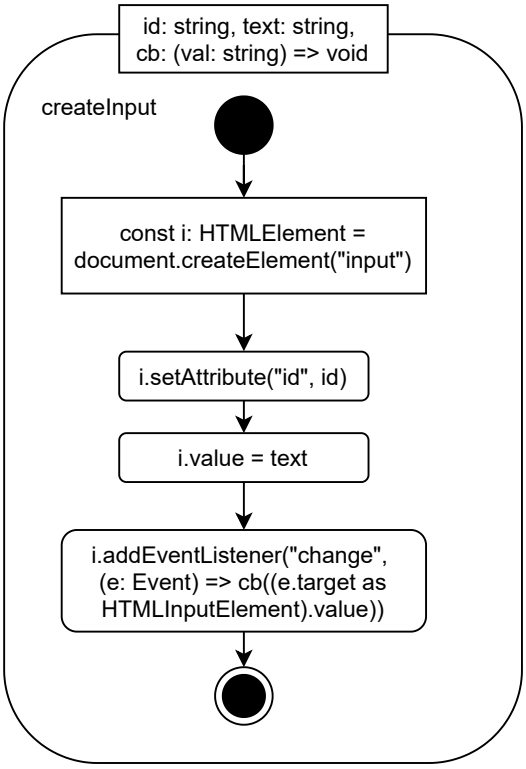

degrees: number
degreesToRadians
return degrees in radians

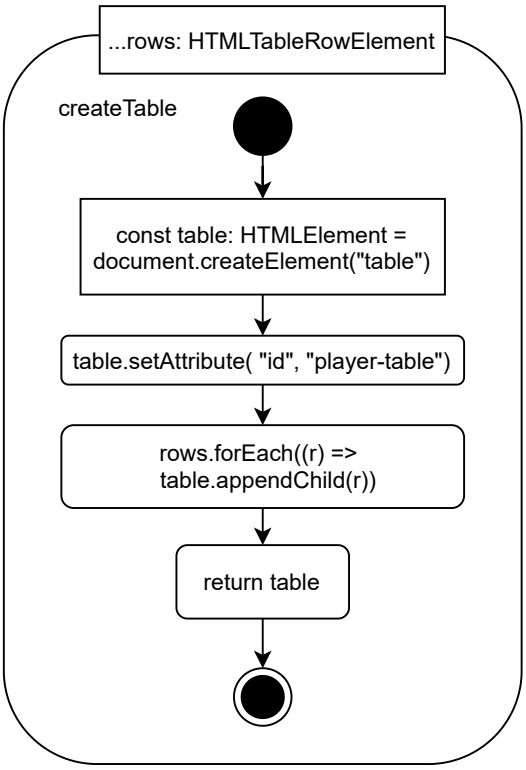

v1: Vector, V2: vector
distance
let d: number = √diffVectr.X² + diffVectr·Y²

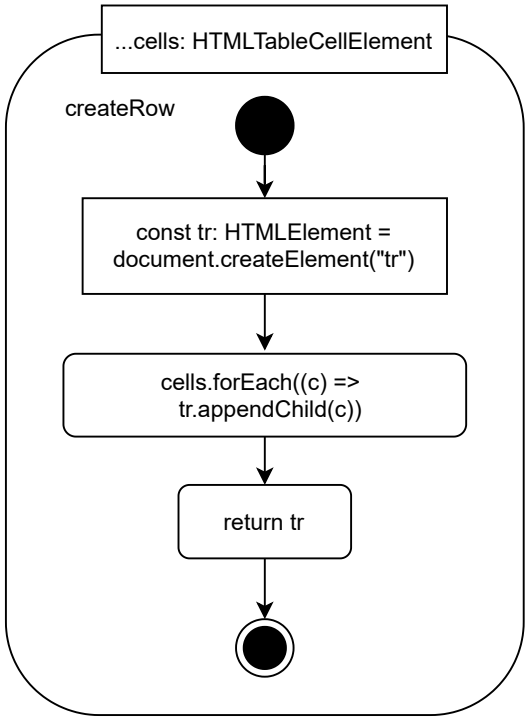
UIHelper.ts

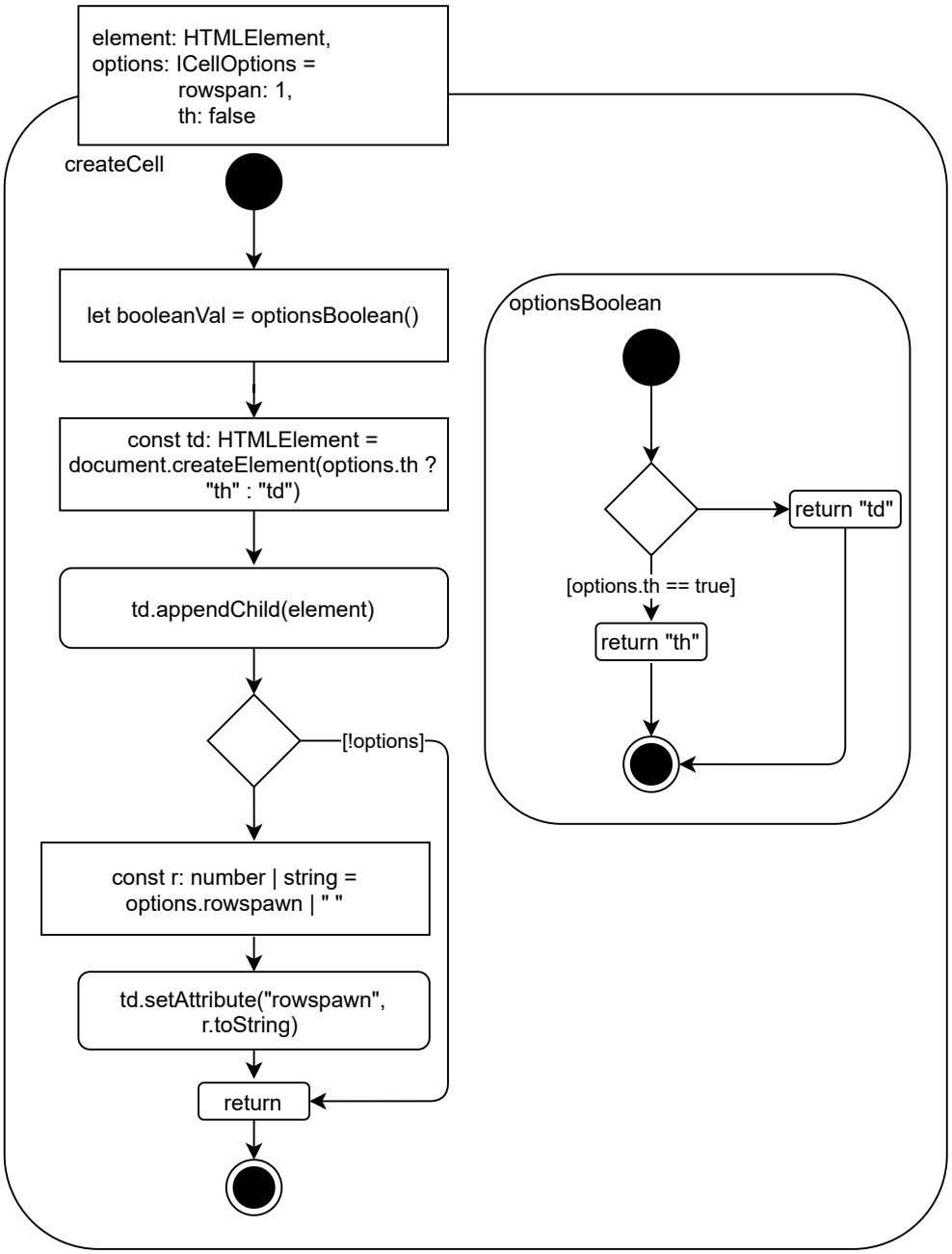

id: string, text: string
updateById
const el: HTMLElement = document.getElementById(id)
[!el]
return
el.innterHTML = text


id: string, text: string
createSpan
const s: HTMLElement = document.createElement("span")
s.setAttribute("id", id)
s.innerHTML = text
return s

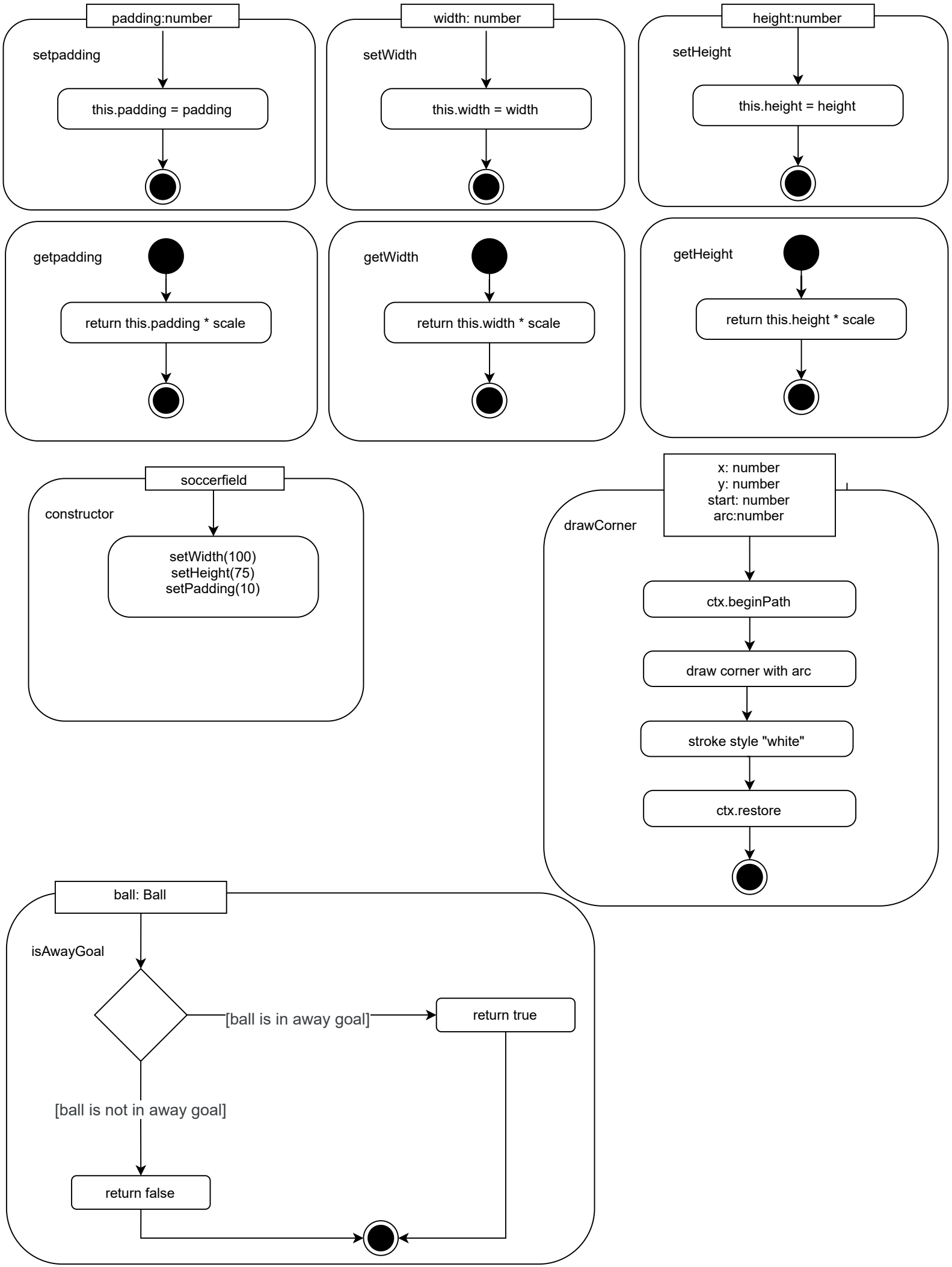

id: string, text: string, cb: (val: string) => void
createInput
const i: HTMLElement = document.createElement("input")
i.setAttribute("id", id)
i.value = text
i.addEventListener("change", (e: Event) => cb((e.target as HTMLInputElement).value))

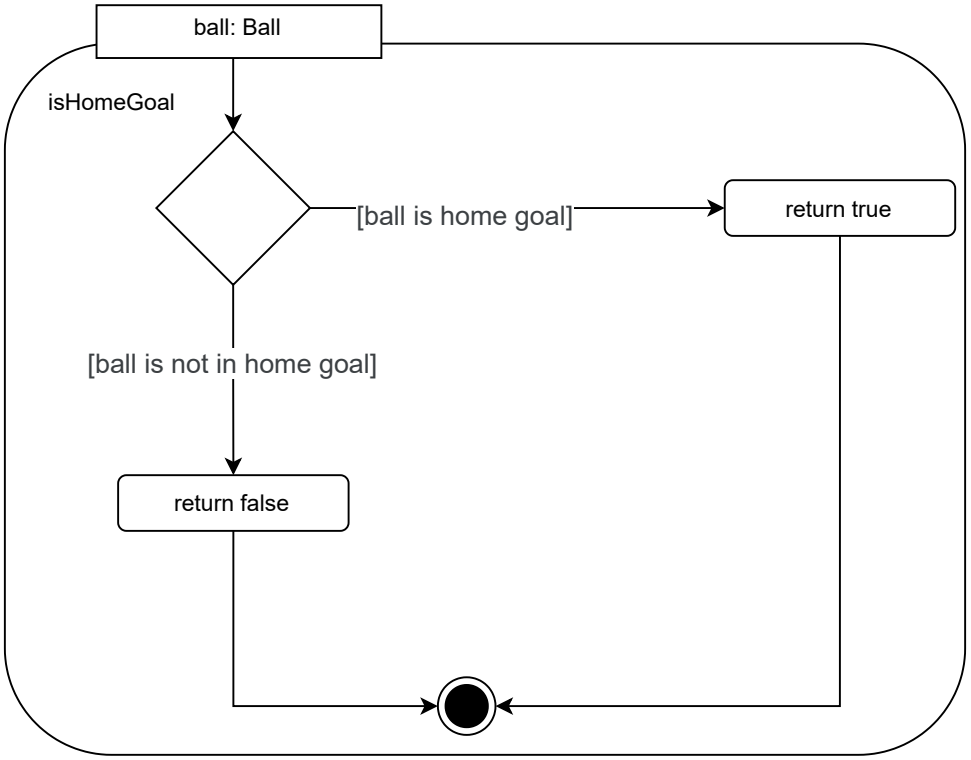

...rows: HTMLTableRowElement
createTable
const table: HTMLElement = document.createElement("table")
table.setAttribute("id", "player-table")
rows.forEach((r) => table.appendChild(r))
return table

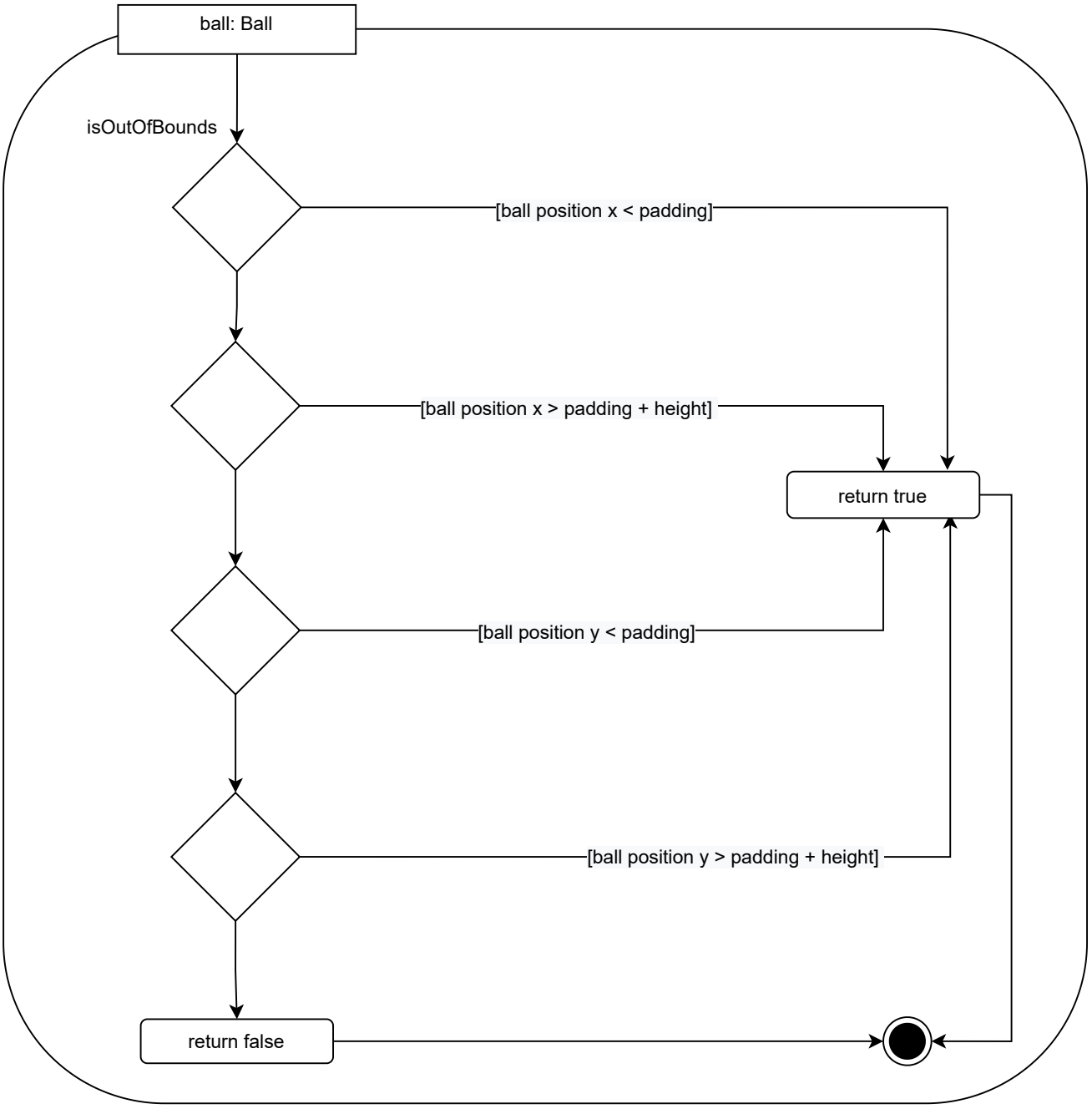

...cells: HTMLTableCellElement
createRow
const tr: HTMLElement =
document.createElement("tr")
cells.forEach((c) =>
tr.appendChild(c))
return tr

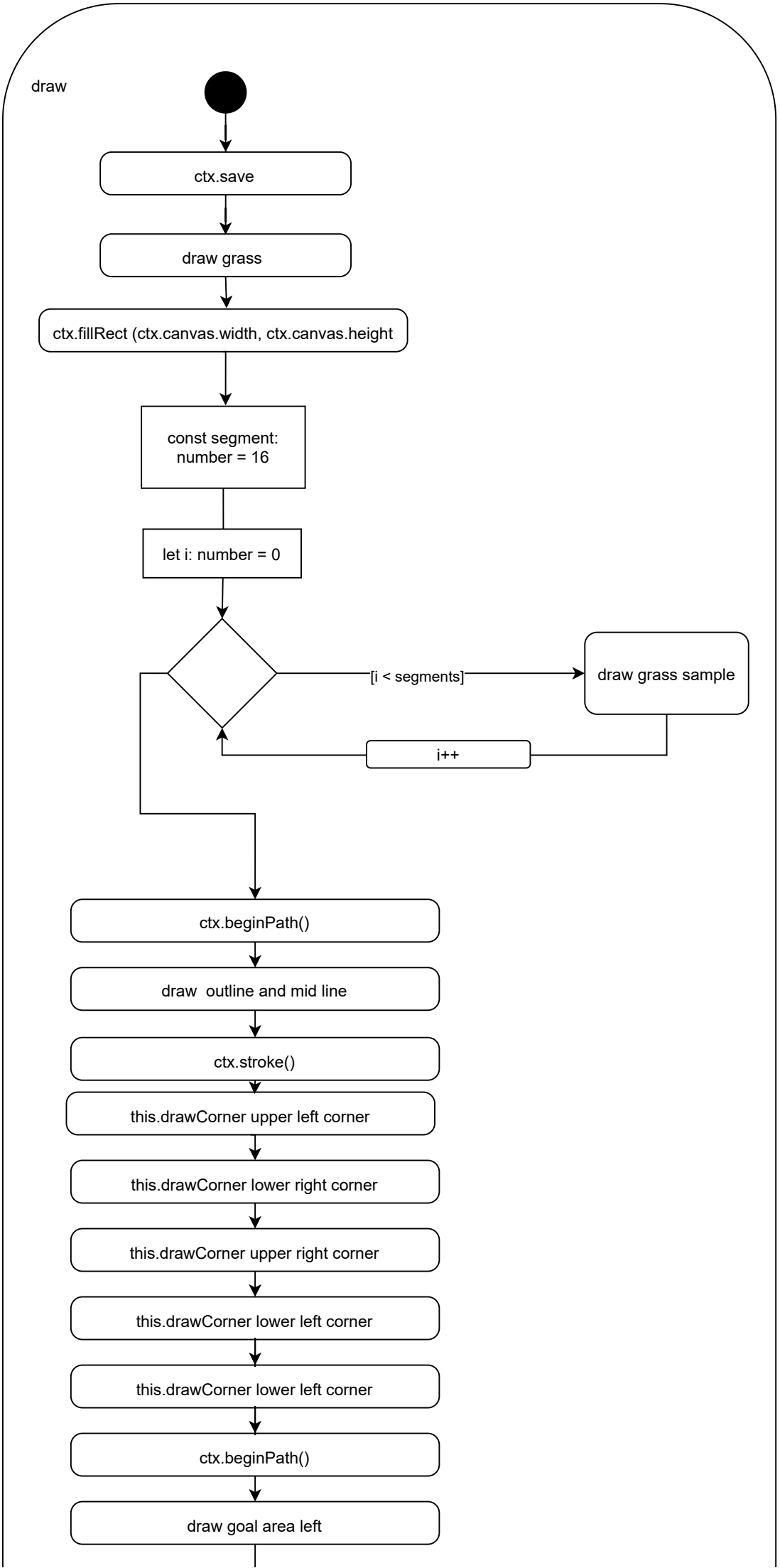

element: HTMLElement,
options: ICellOptions =
rowspan: 1,
th: false
createCell
let booleanVal = optionsBoolean()
const td: HTMLElement =
document.createElement(options.th ?
"th" : "td")
td.appendChild(element)
[!options]
const r: number | string =
options.rowspawn | " "
td.setAttribute("rowspawn",
r.toString)
return
optionsBoolean
return "td"
[options.th == true]
return "th"

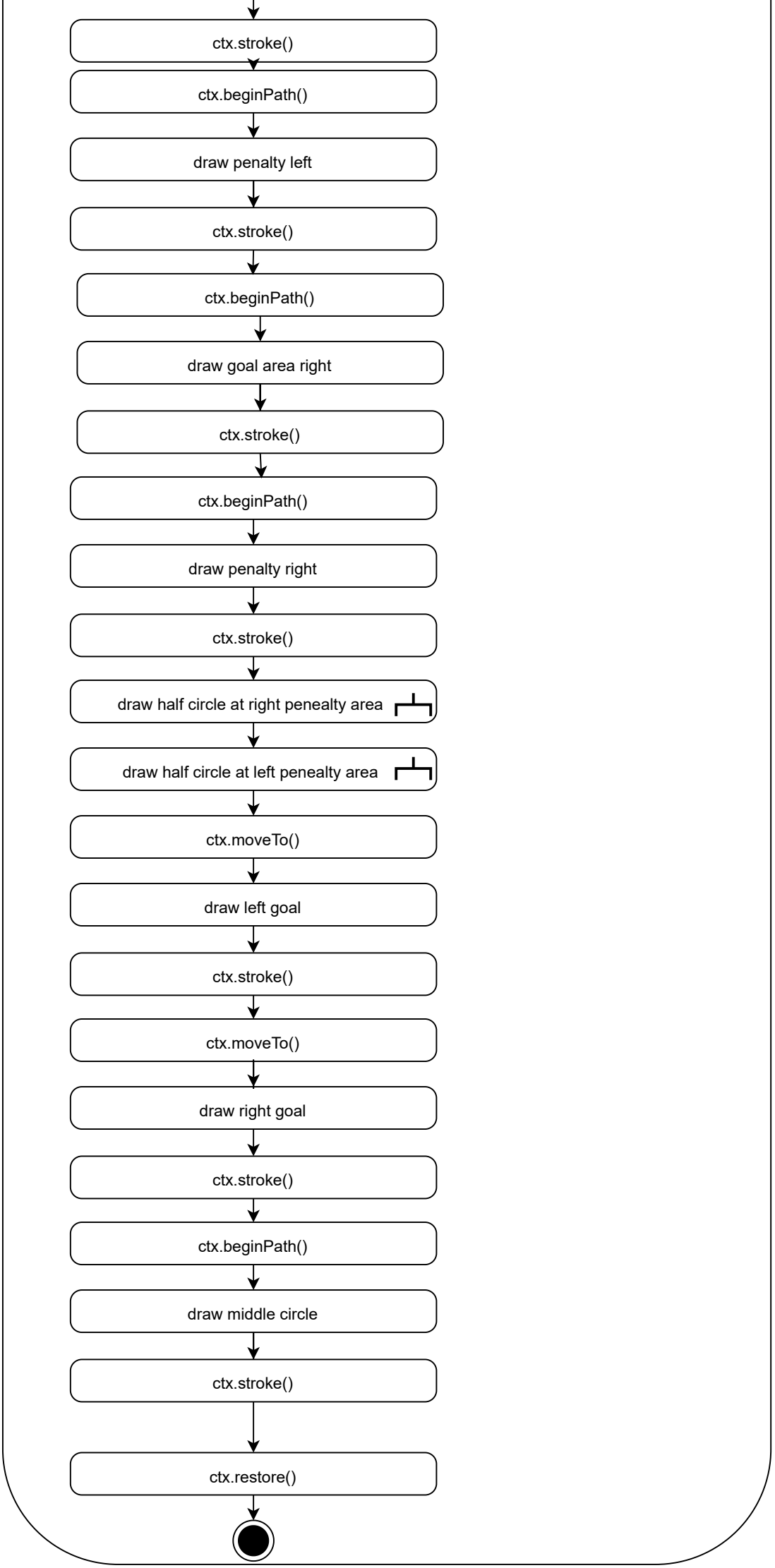
soccer-field.ts Activity Diagram



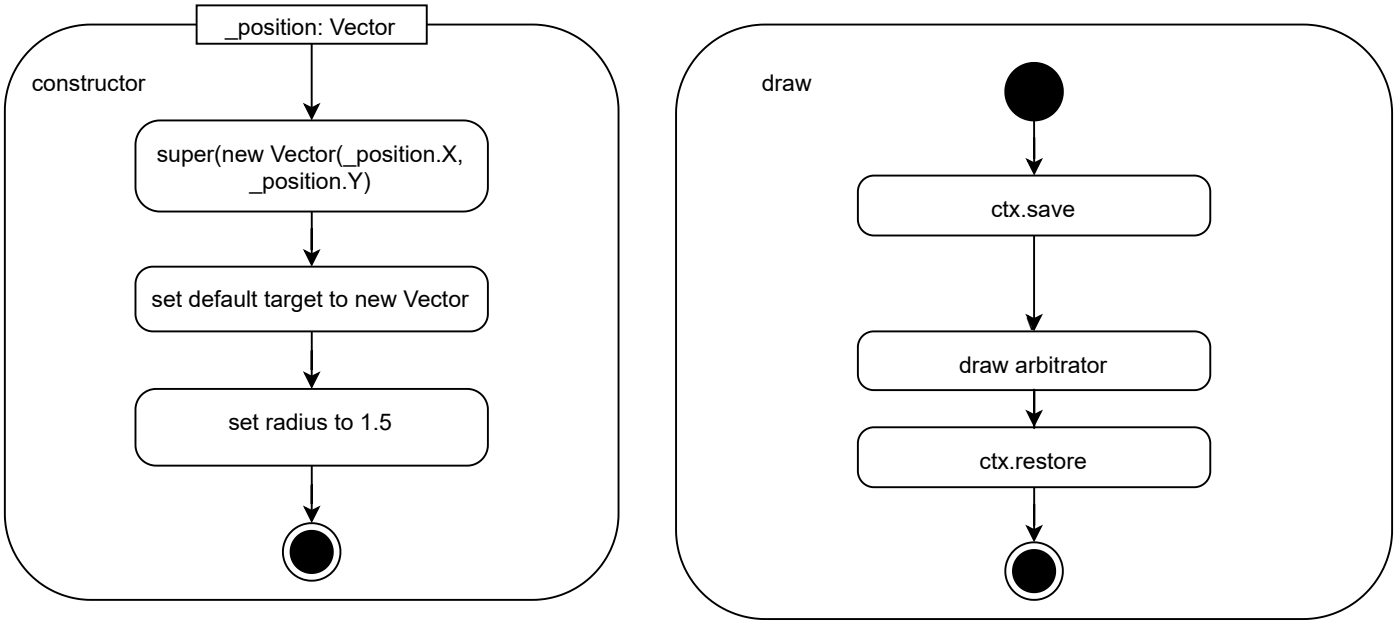

ball: Ball
isHomeGoal
[ball is home goal]
return true
[ball is not in home goal]
return false


ball: Ball
isOutOfBounds
[ball position x < padding]
[ball position x > padding + height]
return true
[ball position y < padding]
[ball position y > padding + height]
return false

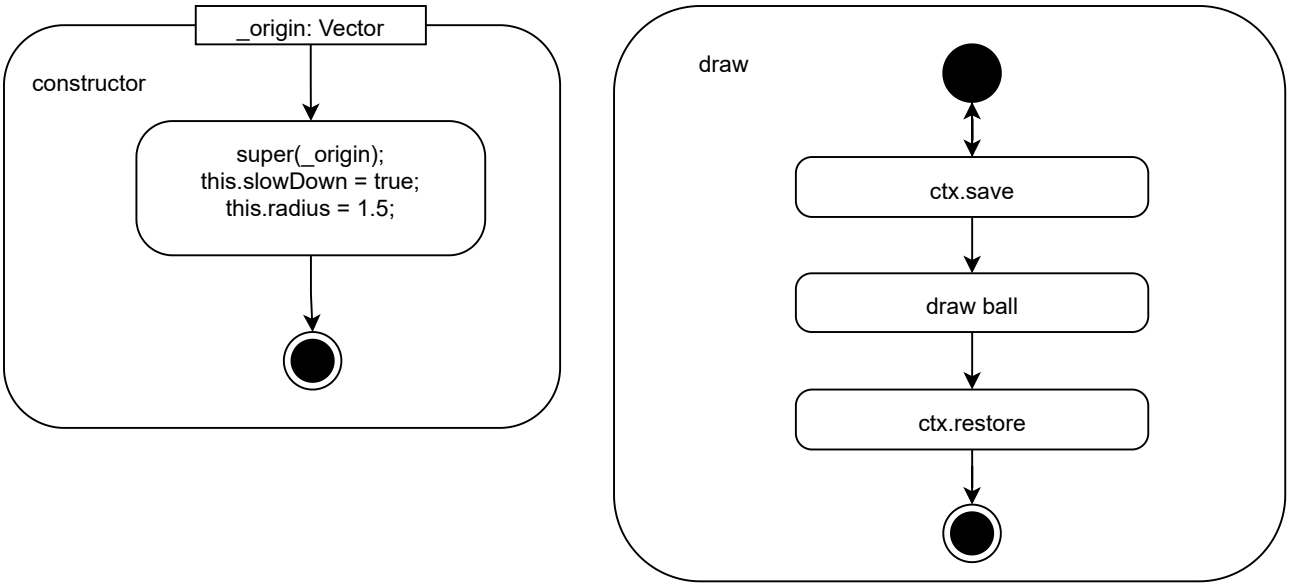

draw
ctx.save
draw grass
ctx.fillRect (ctx.canvas.width, ctx.canvas.height
const segment:
number = 16
let i: number = 0
[i < segments]
draw grass sample
i++
ctx.beginPath()
draw outline and mid line
ctx.stroke()
this.drawCorner upper left corner
this.drawCorner lower right corner
this.drawCorner upper right corner
this.drawCorner lower left corner
this.drawCorner lower left corner
ctx.beginPath()
draw goal area left


ctx.stroke()
ctx.beginPath()
draw penalty left
ctx.stroke()
ctx.beginPath()
draw goal area right
ctx.stroke()
ctx.beginPath()
draw penalty right
ctx.stroke()
draw half circle at right penealty area
draw half circle at left penealty area
ctx.moveTo()
draw left goal
ctx.stroke()
ctx.moveTo()
draw right goal
ctx.stroke()
ctx.beginPath()
draw middle circle
ctx.stroke()
ctx.restore()

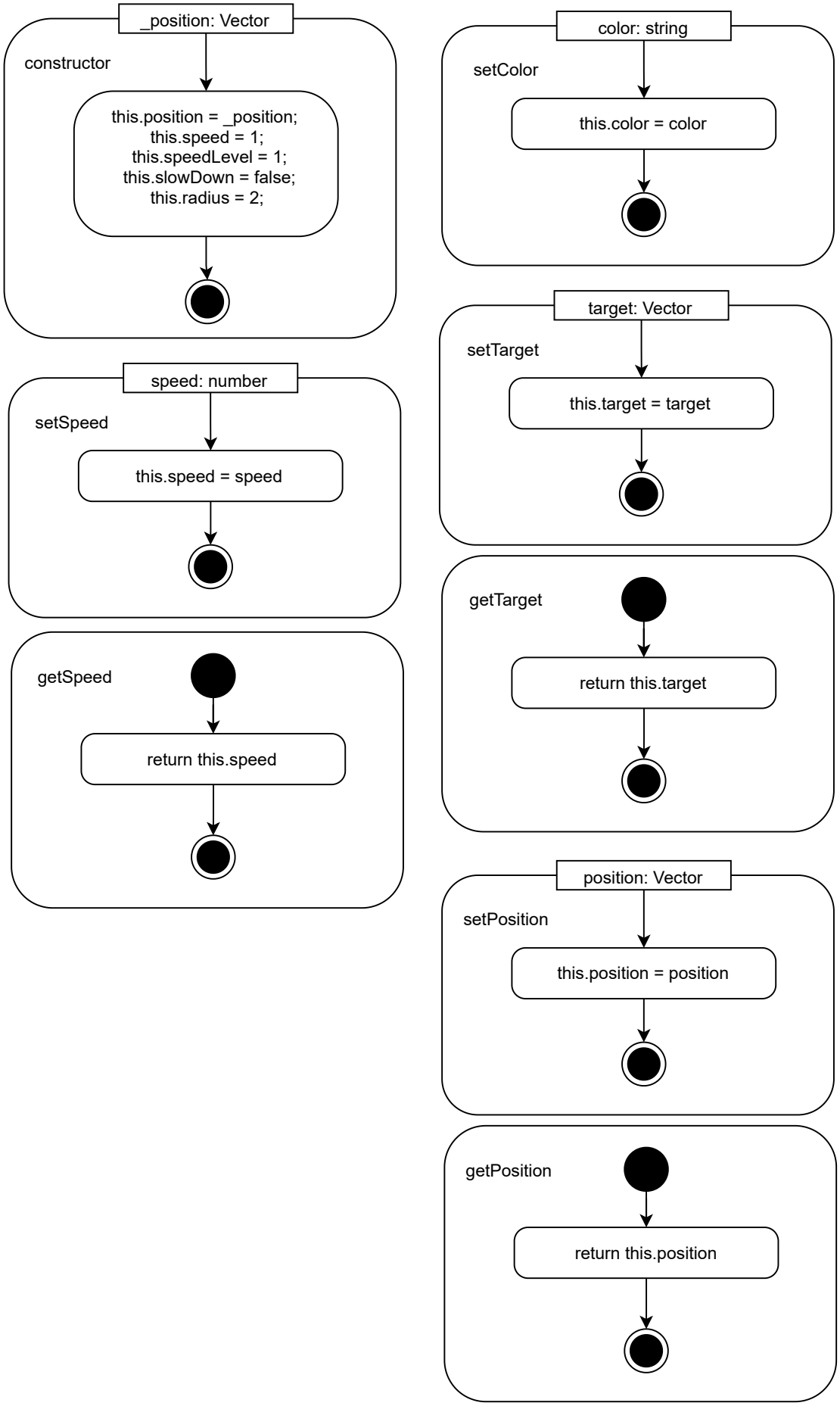
arbitrator.ts Activity Diagram

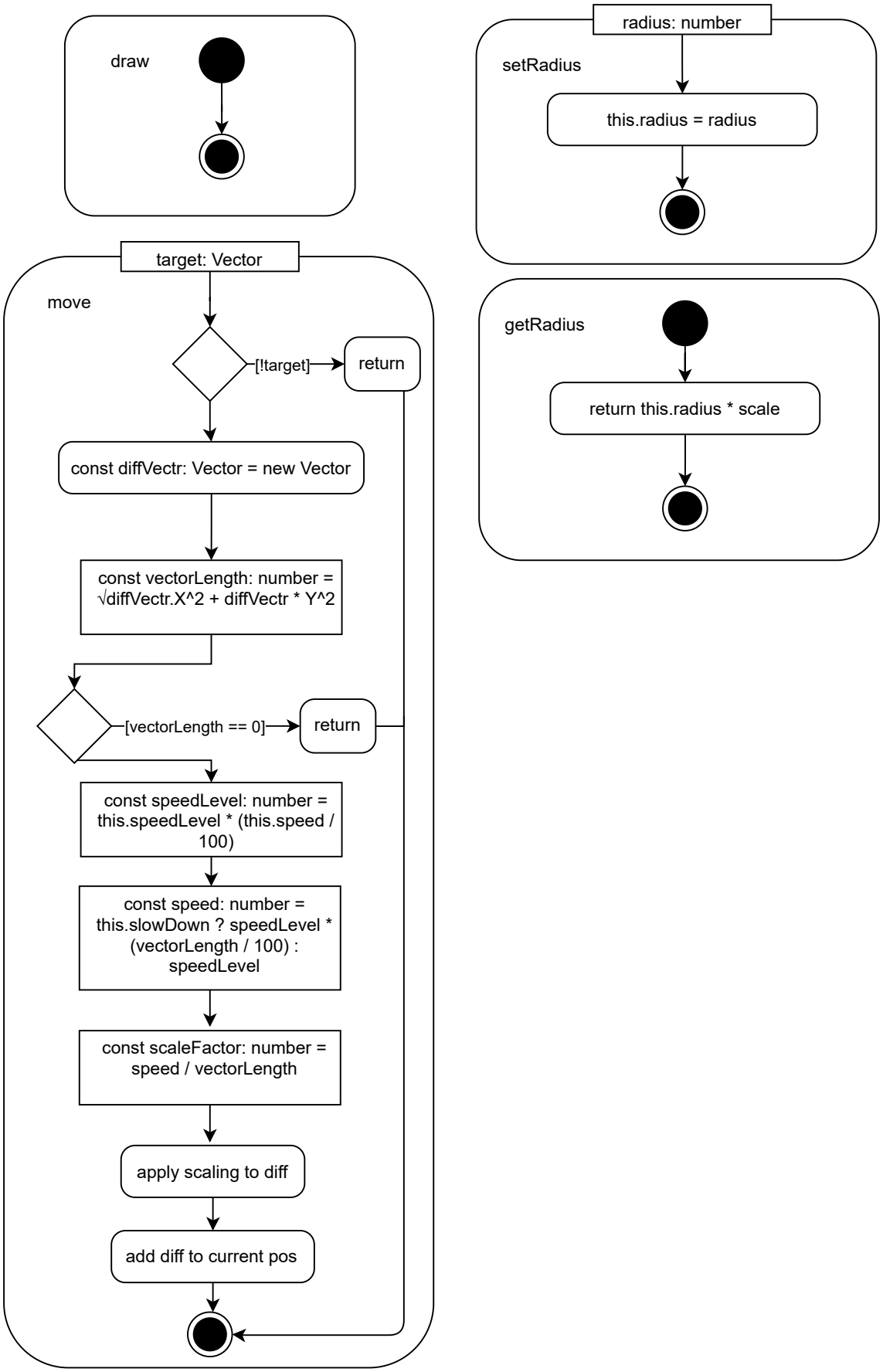


ball.ts Activity Diagram

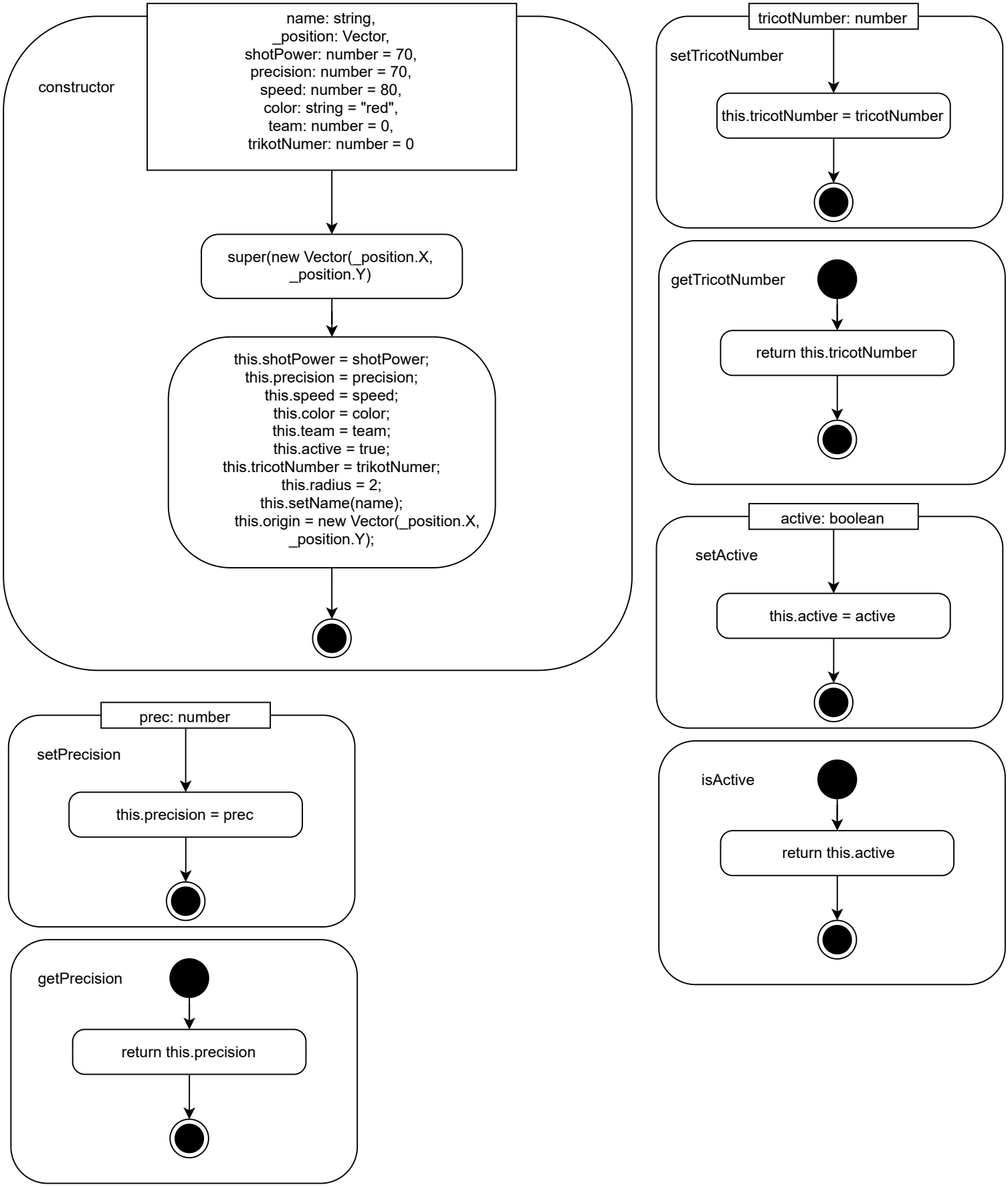


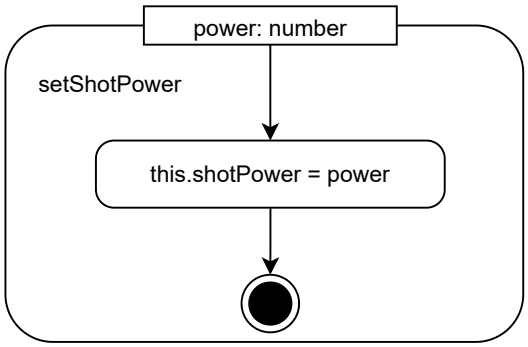
moveable.ts Activity Diagram

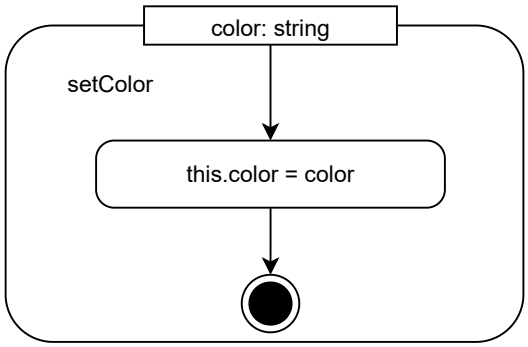


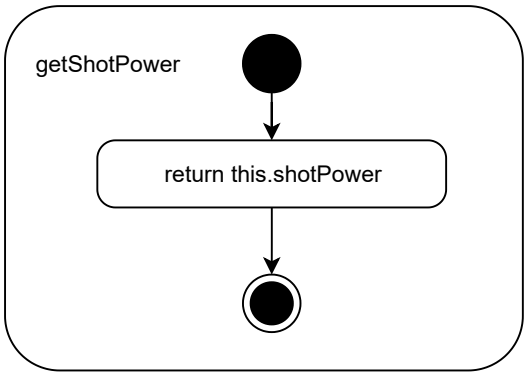

draw
radius: number
setRadius
this.radius = radius
target: Vector
move
[!target]
return
const diffVectr: Vector = new Vector
const vectorLength: number = √diffVectr.X^2 + diffVectr * Y^2
[vectorLength == 0]
return
const speedLevel: number = this.speedLevel * (this.speed / 100)
const speed: number = this.slowDown ? speedLevel * (vectorLength / 100) : speedLevel
const scaleFactor: number = speed / vectorLength
apply scaling to diff
add diff to current pos
getRadius
return this.radius * scale

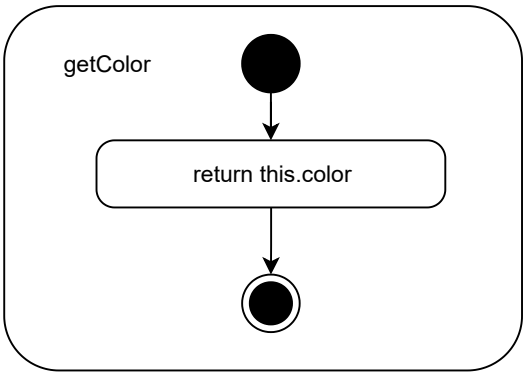
player.ts Activity Diagram

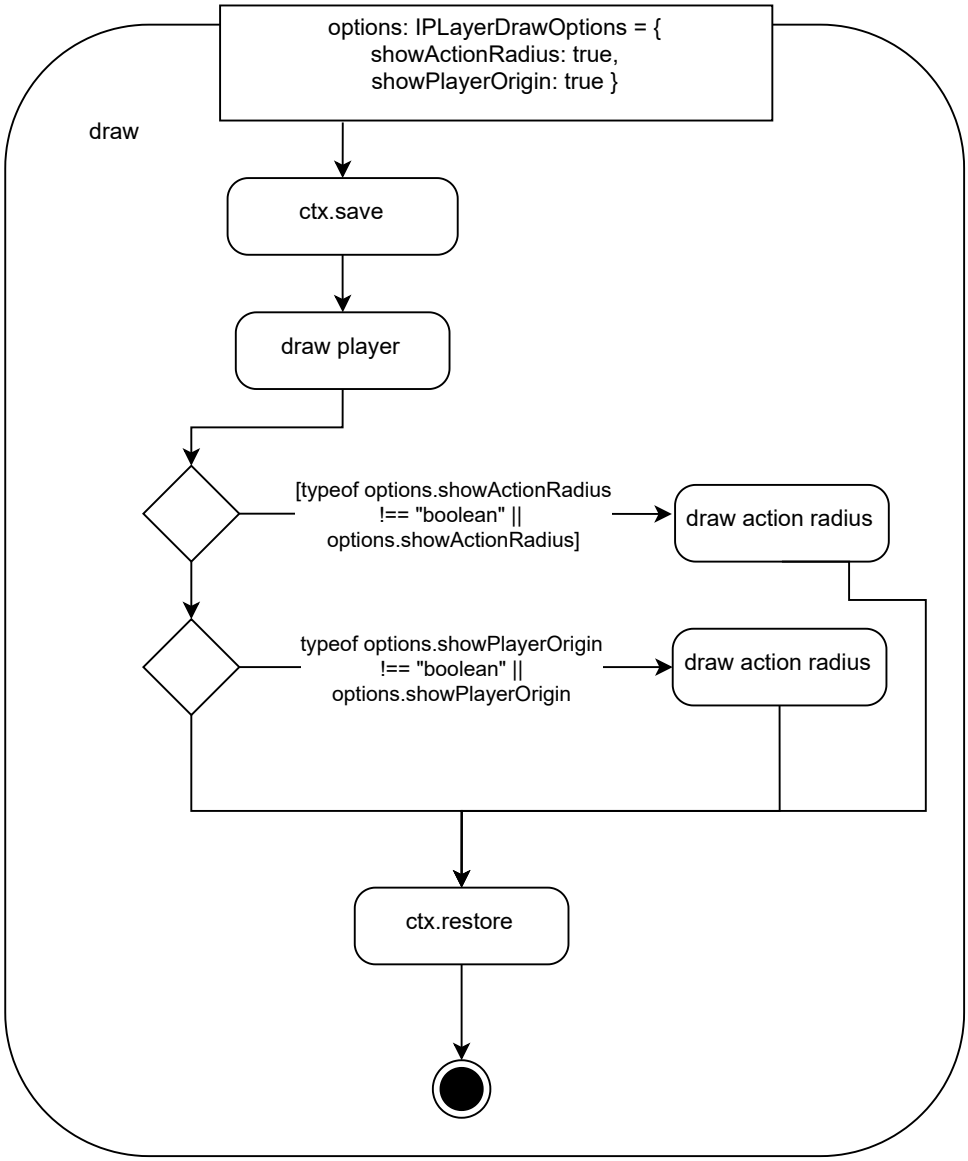


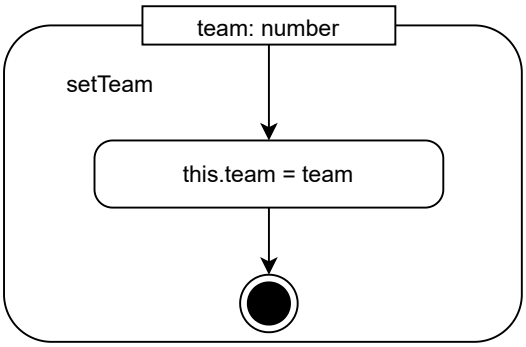

power: number
setShotPower
this.shotPower = power

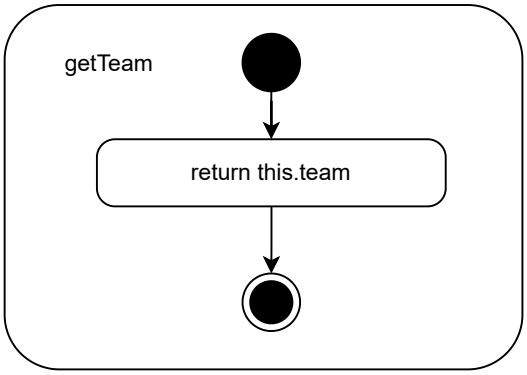

color: string
setColor
this.color = color

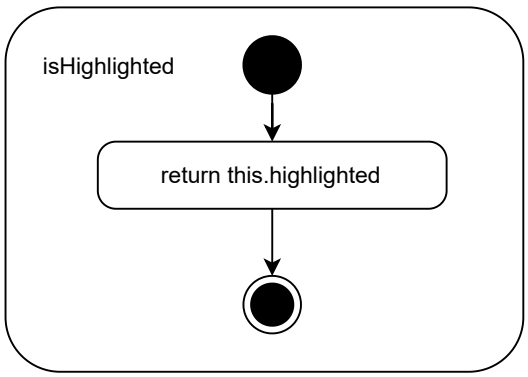

getShotPower
return this.shotPower

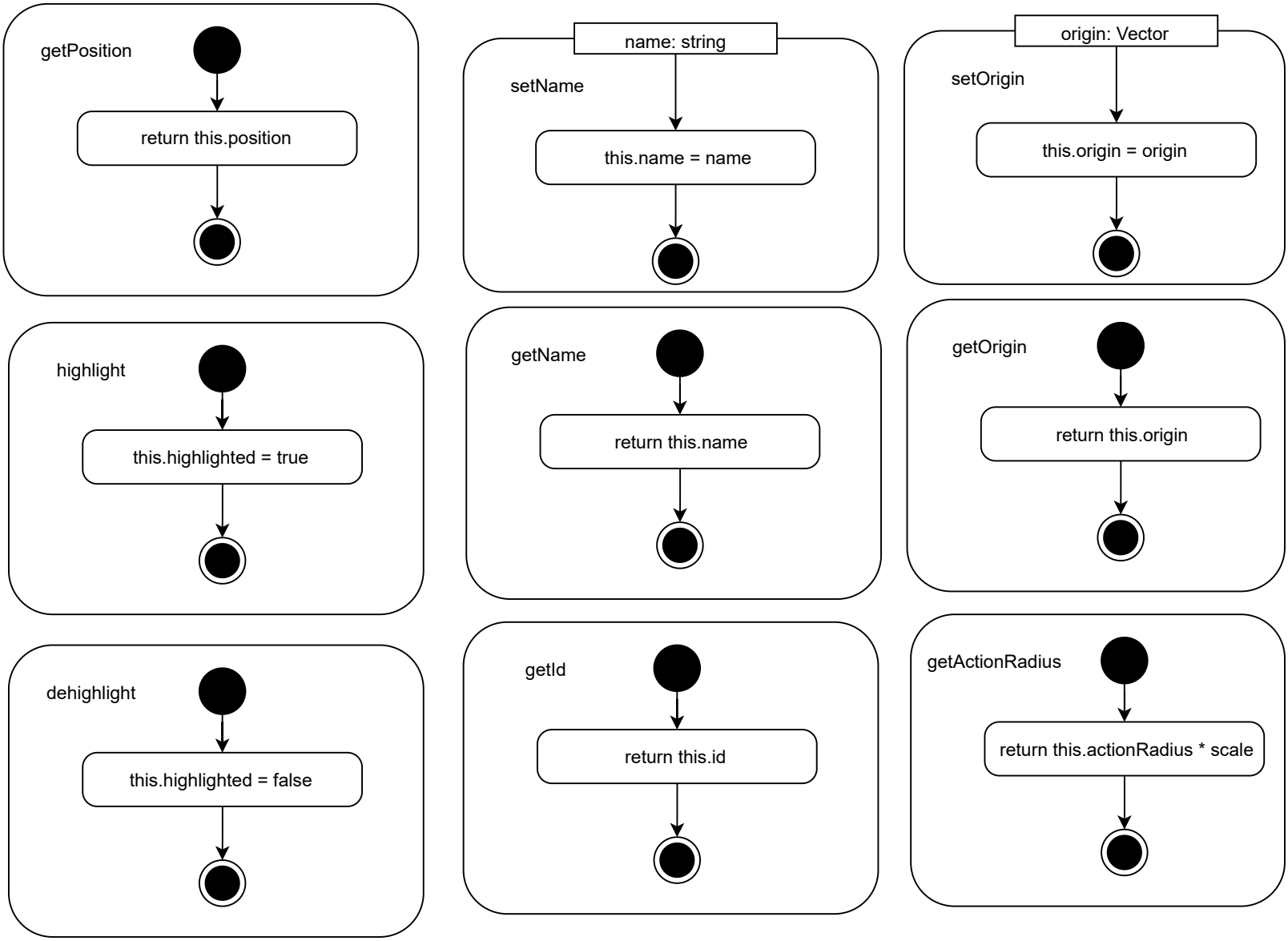

getColor
return this.color


options: IPLayerDrawOptions = {
showActionRadius: true,
showPlayerOrigin: true }
draw
ctx.save
draw player
[typeof options.showActionRadius
!== "boolean" ||
options.showActionRadius]
draw action radius
typeof options.showPlayerOrigin
!== "boolean" ||
options.showPlayerOrigin
draw action radius
ctx.restore

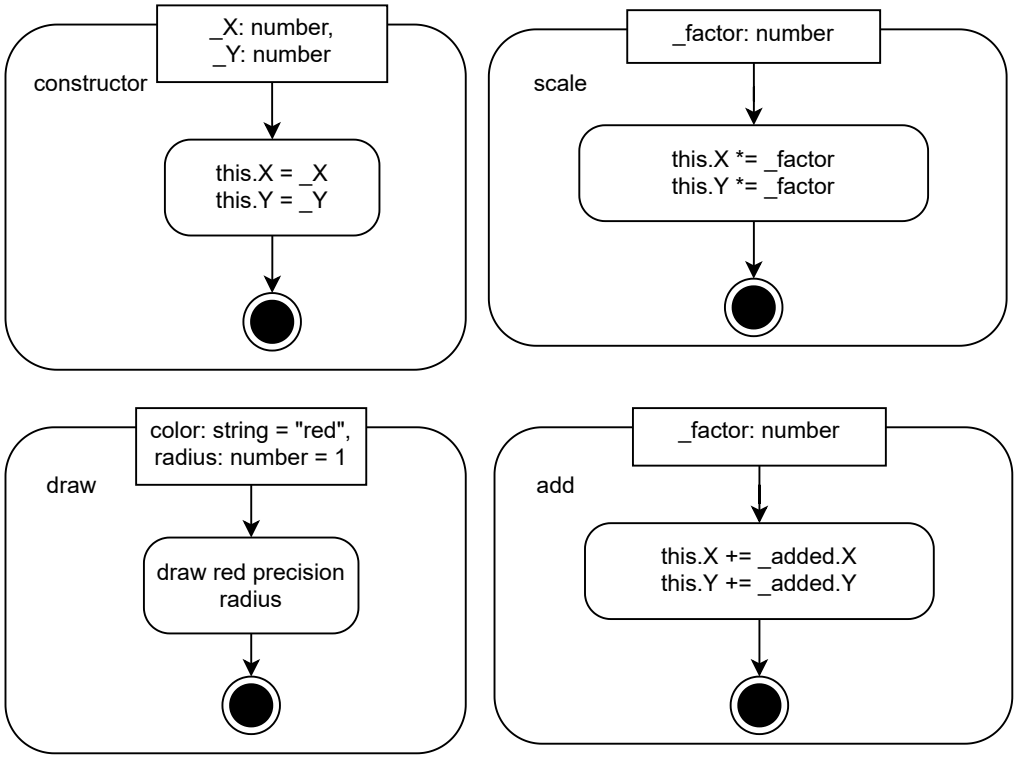

team: number
setTeam
this.team = team


getTeam
return this.team

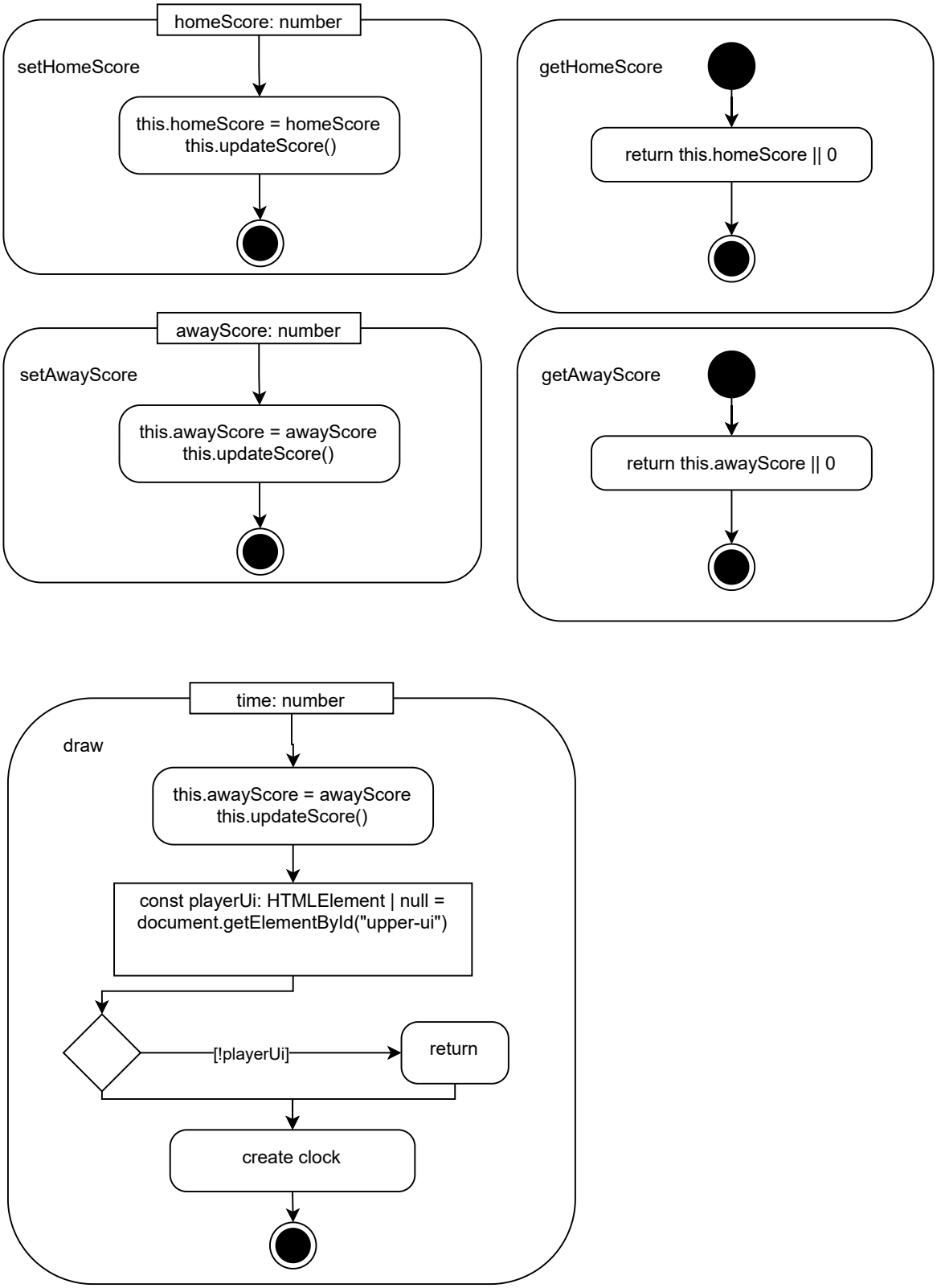

isHighlighted
return this.highlighted

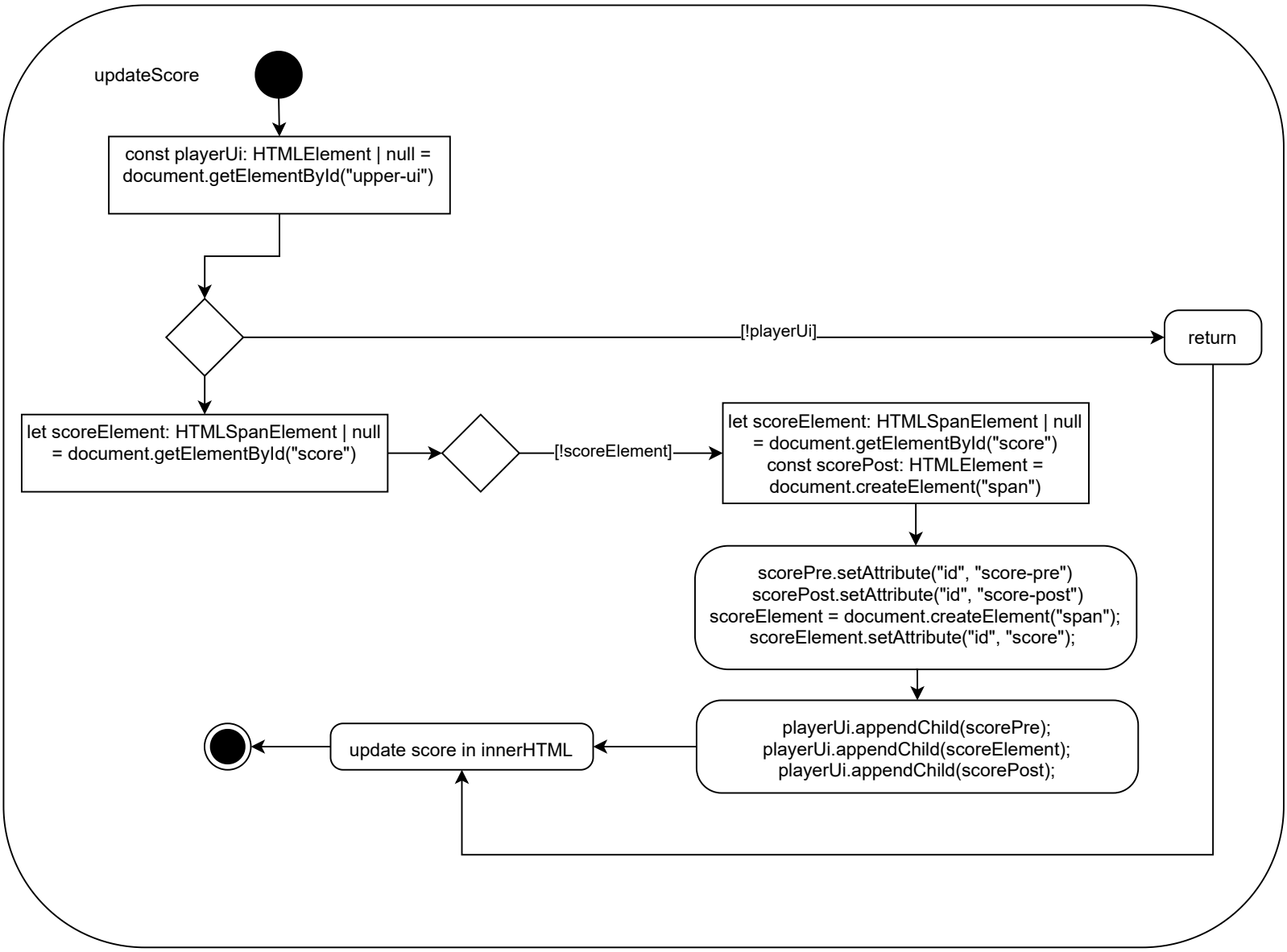


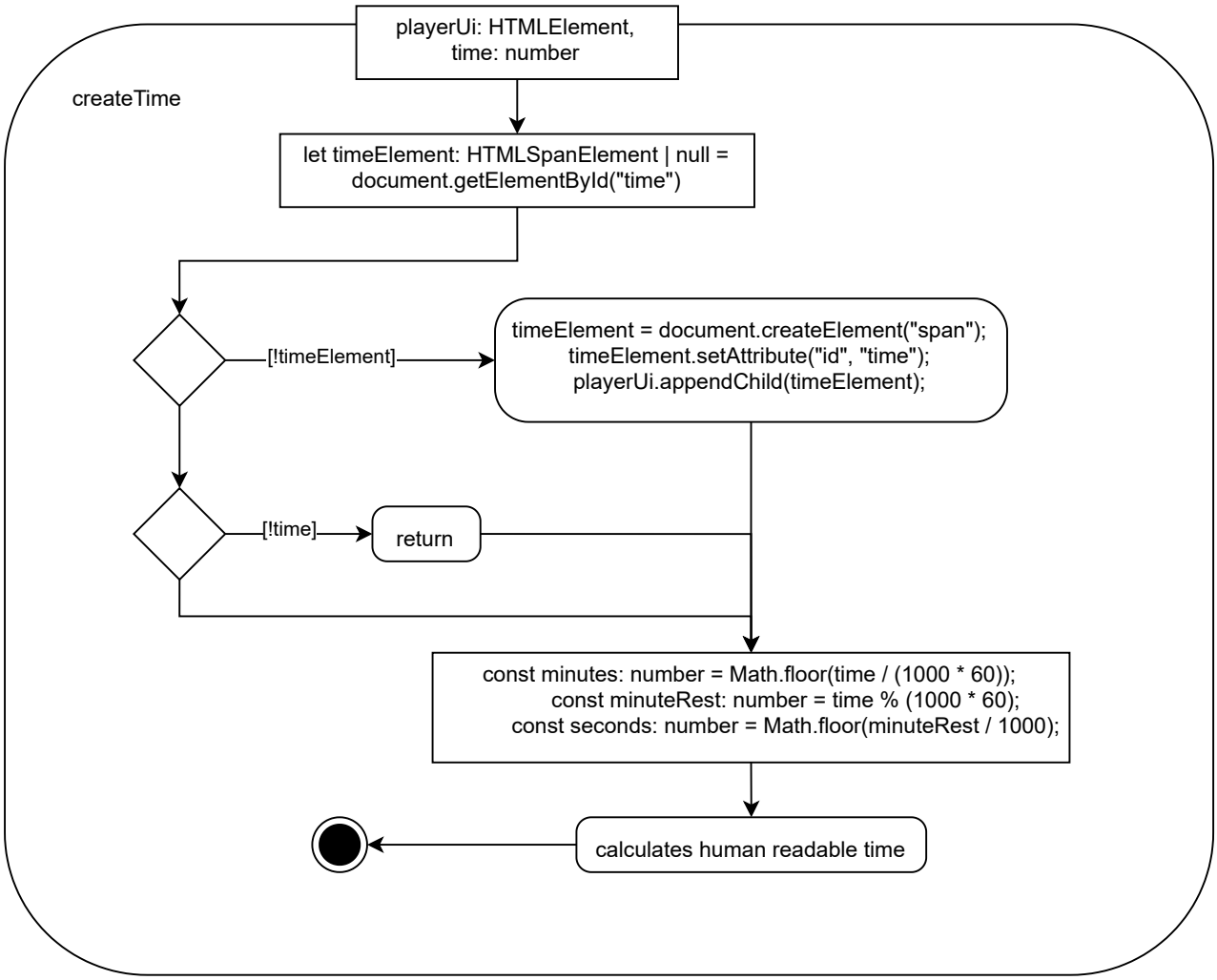
vector.ts Activity Diagram



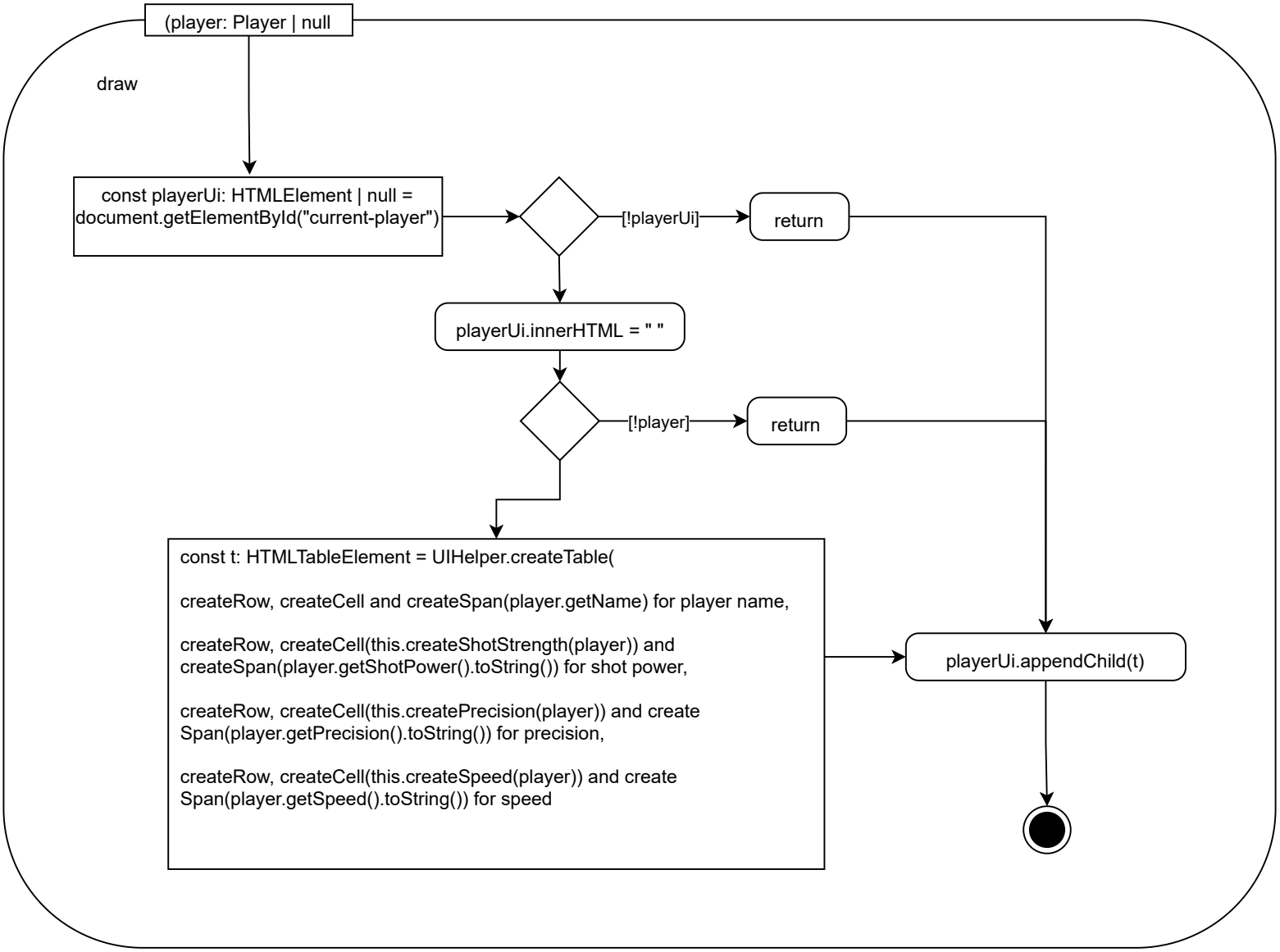
ui.ts Activity Diagram

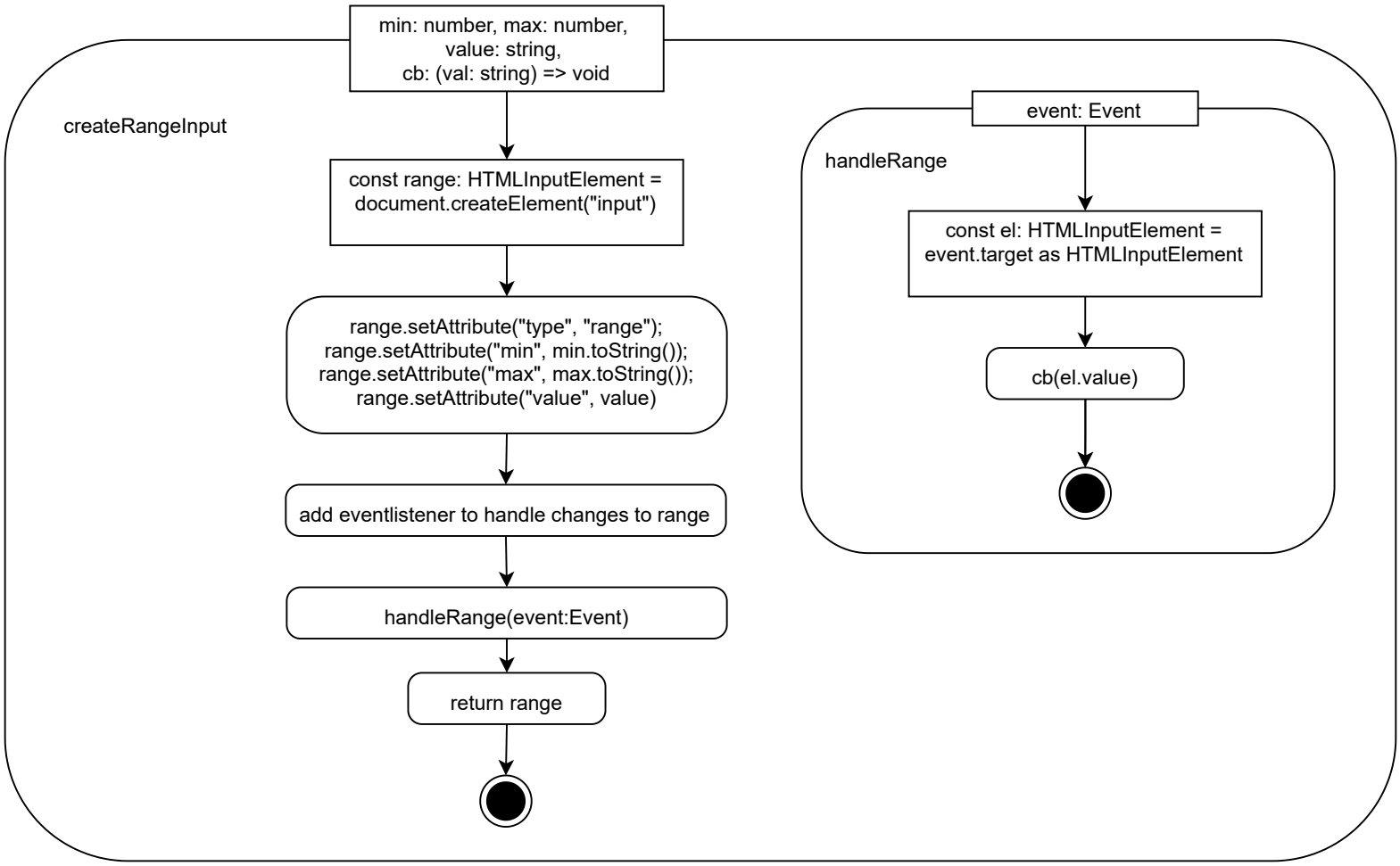


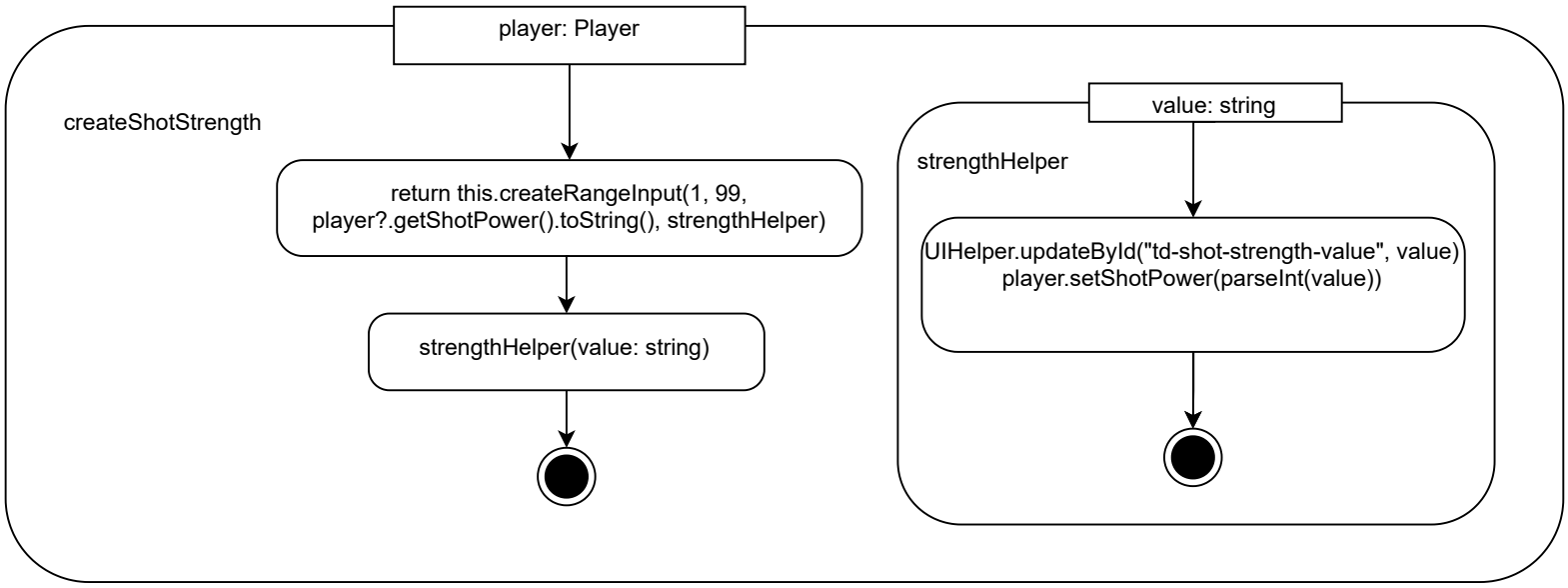

updateScore
const playerUi: HTMLElement | null = document.getElementById("upper-ui")
[!playerUi]
return
let scoreElement: HTMLSpanElement | null = document.getElementById("score")
[!scoreElement]
let scoreElement: HTMLSpanElement | null = document.getElementById("score") const scorePost: HTMLElement = document.createElement("span")
scorePre.setAttribute("id", "score-pre") scorePost.setAttribute("id", "score-post") scoreElement = document.createElement("span"); scoreElement.setAttribute("id", "score");
playerUi.appendChild(scorePre); playerUi.appendChild(scoreElement); playerUi.appendChild(scorePost);
update score in innerHTML

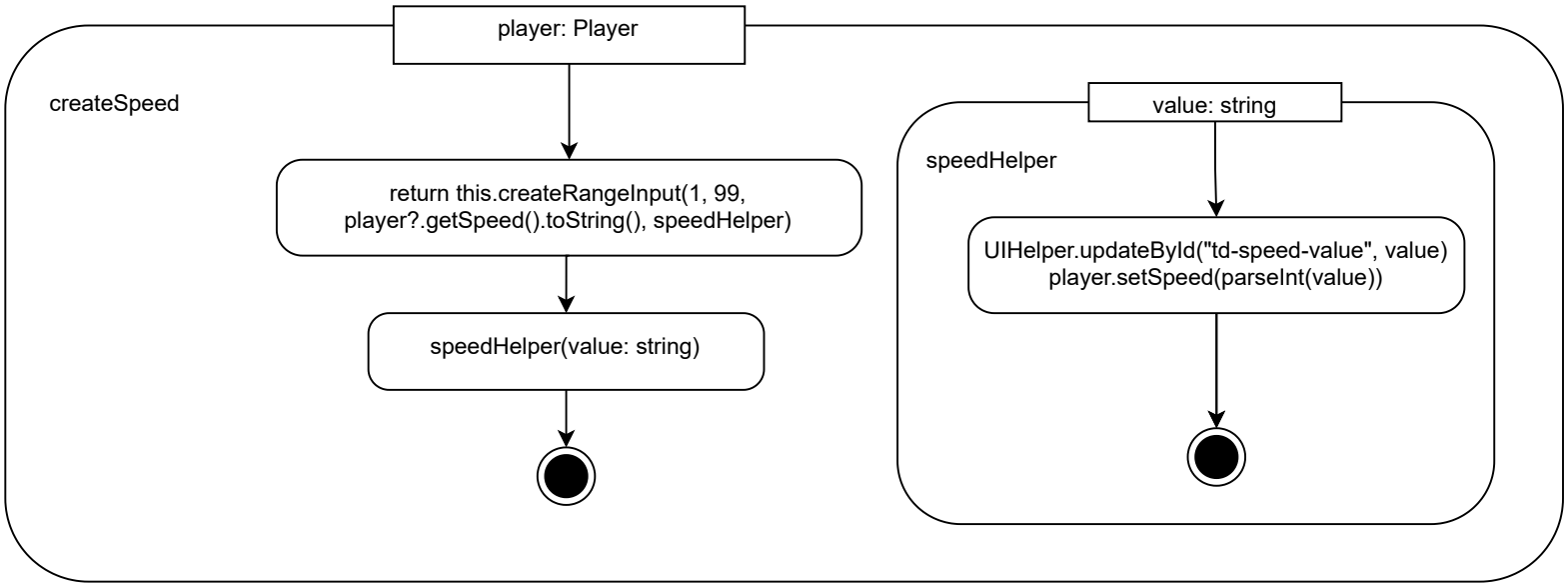

playerUi: HTMLElement,
time: number
createTime
let timeElement: HTMLSpanElement | null =
document.getElementById("time")
[!timeElement]
timeElement = document.createElement("span");
timeElement.setAttribute("id", "time");
playerUi.appendChild(timeElement);
[!time]
return
const minutes: number = Math.floor(time / (1000 * 60));
const minuteRest: number = time % (1000 * 60);
const seconds: number = Math.floor(minuteRest / 1000);
calculates human readable time

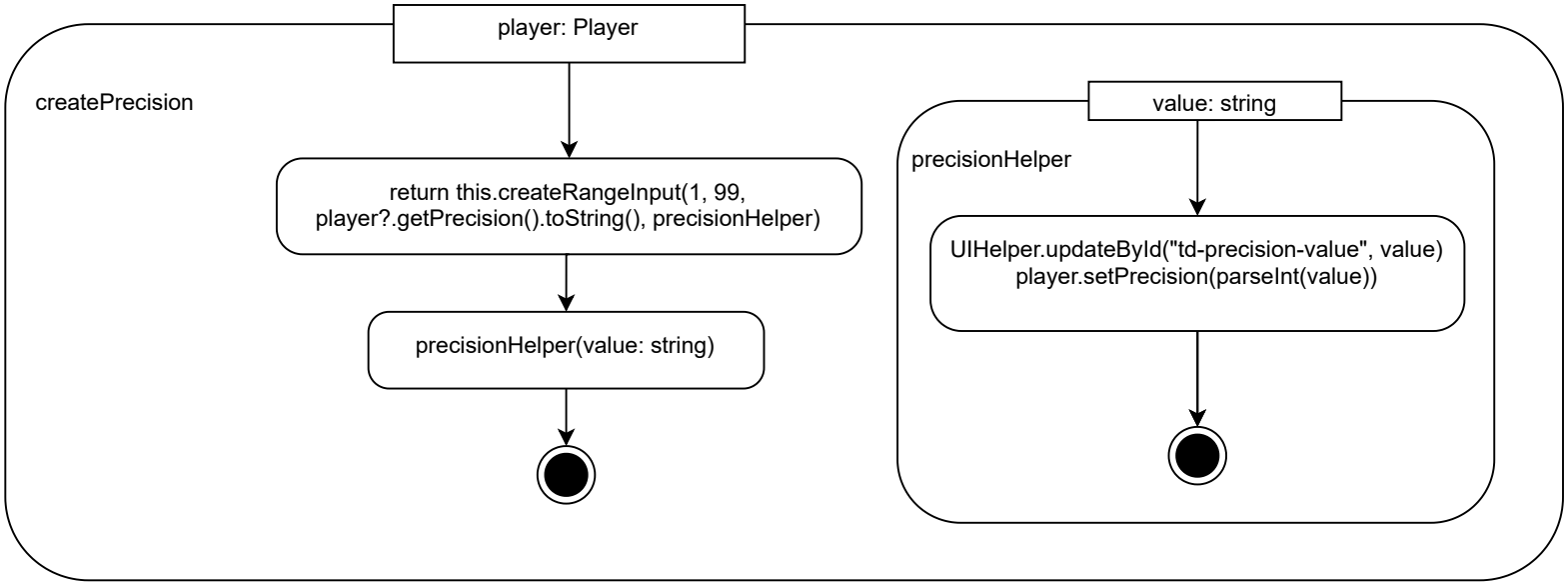
player-ui.ts Activity Diagram



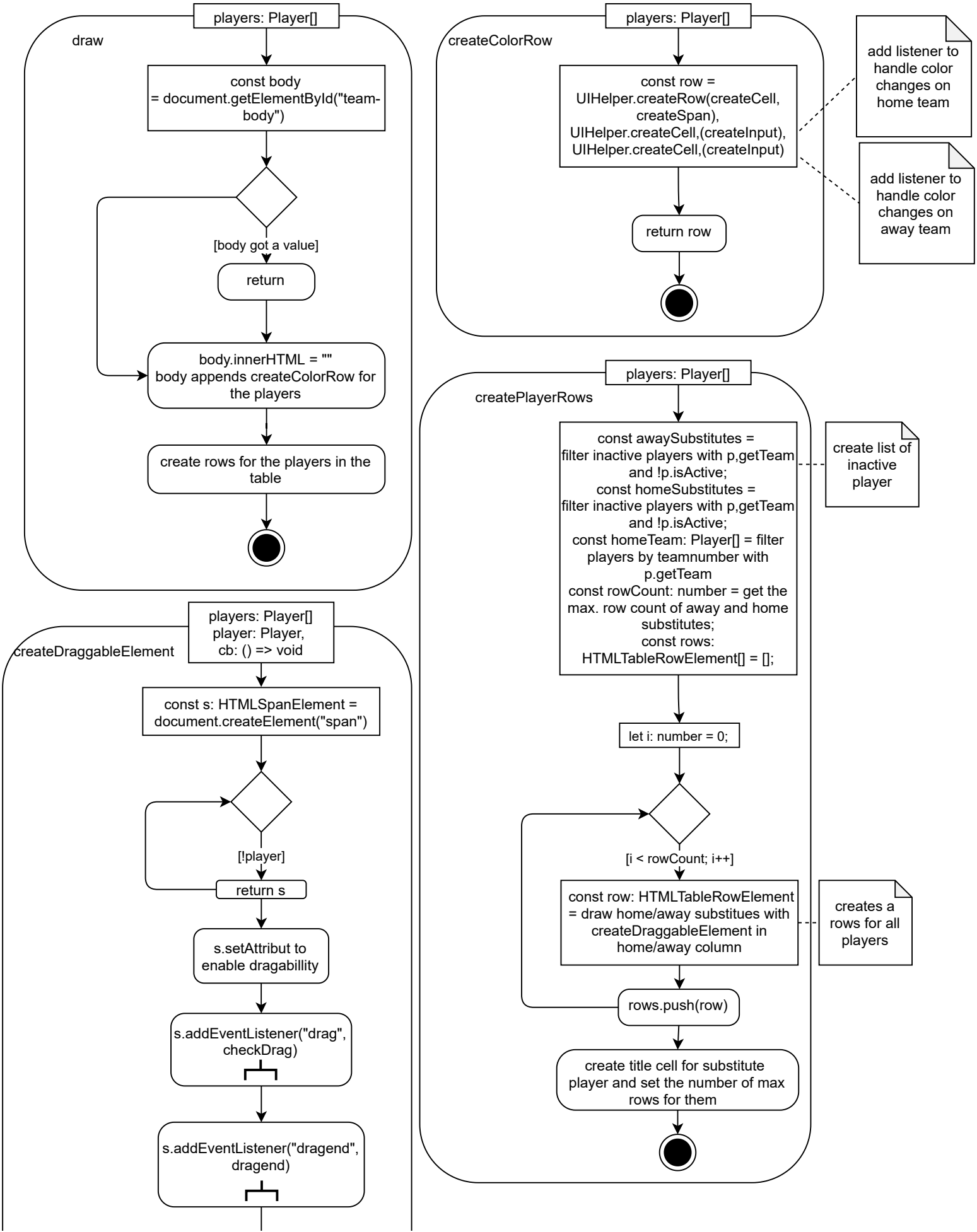

min: number, max: number,
value: string,
cb: (val: string) => void
createRangeInput
const range: HTMLInputElement =
document.createElement("input")
range.setAttribute("type", "range");
range.setAttribute("min", min.toString());
range.setAttribute("max", max.toString());
range.setAttribute("value", value)
add eventlistener to handle changes to range
handleRange(event:Event)
return range
event: Event
handleRange
const el: HTMLInputElement =
event.target as HTMLInputElement
cb(el.value)

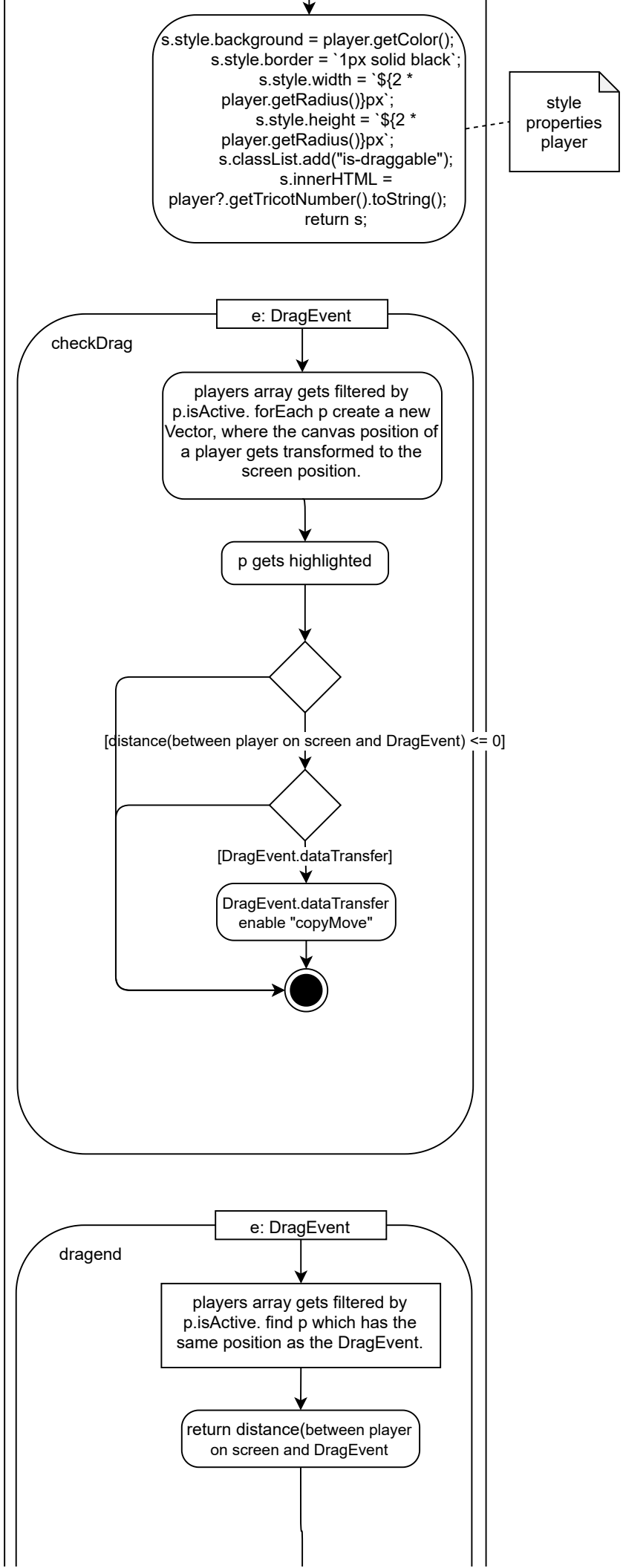

player: Player
createShotStrength
return this.createRangeInput(1, 99,
player?.getShotPower().toString(), strengthHelper)
strengthHelper(value: string)
value: string
strengthHelper
UIHelper.updateById("td-shot-strength-value", value)
player.setShotPower(parseInt(value))

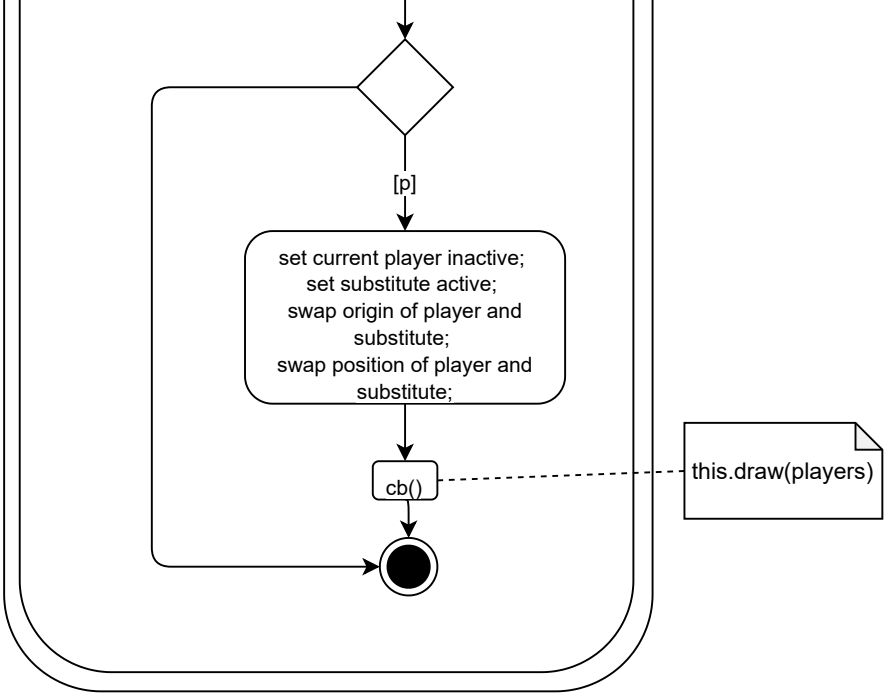

player: Player
createSpeed
return this.createRangeInput(1, 99,
player?.getSpeed().toString(), speedHelper)
speedHelper(value: string)
value: string
speedHelper
UIHelper.updateById("td-speed-value", value)
player.setSpeed(parseInt(value))


player: Player
createPrecision
return this.createRangeInput(1, 99,
player?.getPrecision().toString(), precisionHelper)
precisionHelper(value: string)
value: string
precisionHelper
UIHelper.updateById("td-precision-value", value)
player.setPrecision(parseInt(value))

team-ui.ts


draw
players: Player[]
const body = document.getElementById("team-body")
[body got a value]
return
body.innerHTML = ""
body appends createColorRow for the players
create rows for the players in the table
createColorRow
players: Player[]
const row = UIHelper.createRow(createCell, createSpan), UIHelper.createCell,(createInput), UIHelper.createCell,(createInput)
return row
add listener to handle color changes on home team
add listener to handle color changes on away team
createPlayerRows
players: Player[]
const awaySubstitutes = filter inactive players with p,getTeam and !p.isActive;
const homeSubstitutes = filter inactive players with p,getTeam and !p.isActive;
const homeTeam: Player[] = filter players by teamnumber with p.getTeam
const rowCount: number = get the max. row count of away and home substitutes;
const rows: HTMLTableRowElement[] = [];
create list of inactive player
let i: number = 0;
[i < rowCount; i++]
const row: HTMLTableRowElement = draw home/away substitues with createDraggableElement in home/away column
creates a rows for all players
rows.push(row)
create title cell for substitute player and set the number of max rows for them
createDraggableElement
players: Player[]
player: Player,
cb: () => void
const s: HTMLSpanElement = document.createElement("span")
[!player]
return s
s.setAttribut to enable dragabillity
s.addEventListener("drag", checkDrag)
s.addEventListener("dragend", dragend)


s.style.background = player.getColor();
s.style.border = `1px solid black`;
s.style.width = `${2 * player.getRadius()}px`;
s.style.height = `${2 * player.getRadius()}px`;
s.classList.add("is-draggable");
s.innerHTML = player?.getTricotNumber().toString();
return s;
style properties player
e: DragEvent
checkDrag
players array gets filtered by p.isActive. forEach p create a new Vector, where the canvas position of a player gets transformed to the screen position.
p gets highlighted
[distance(between player on screen and DragEvent) <= 0]
[DragEvent.dataTransfer]
DragEvent.dataTransfer enable "copyMove"
e: DragEvent
dragend
players array gets filtered by p.isActive. find p which has the same position as the DragEvent.
return distance(between player on screen and DragEvent


[p]
set current player inactive;
set substitute active;
swap origin of player and substitute;
swap position of player and substitute;
cb()
this.draw(players)