Anleitung zur Interaktion mit der Anwendung

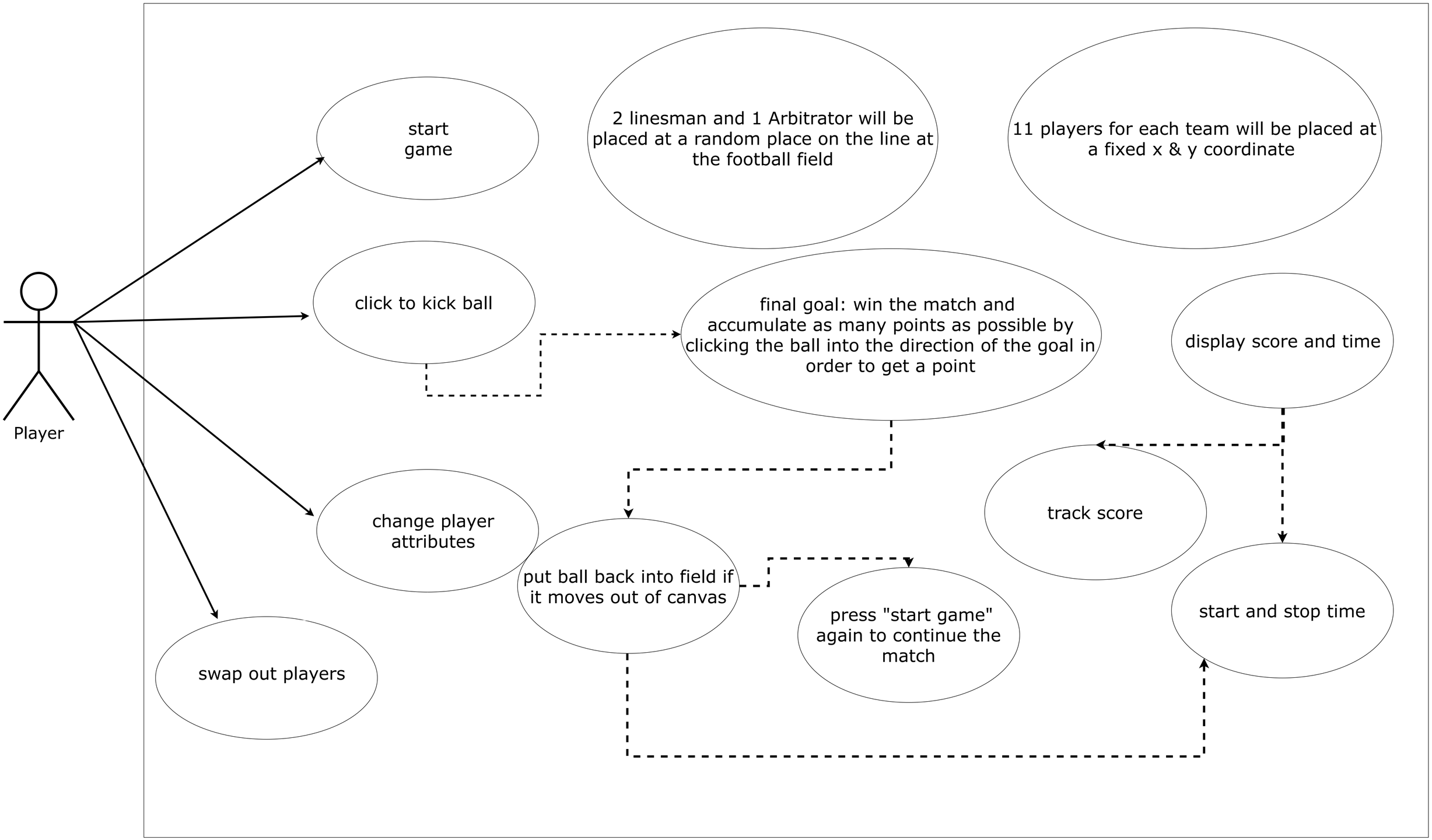
1. Doppelklick auf die ZIP-Datei um diese zu entpacken und die notwendigen Dateien einzusehen.

2. Doppelklick auf die index.html Datei. Die Datei bzw. Anwendung öffnet sich dann auf deinem präferierten Browser.

3. Nun kannst du das Spiel mit Leertaste oder mit Klick auf "Anpfiff" starten.

4. Versuche nun ein Tor zu schießen. Wenn dir das geglückt ist, drücke wieder auf die Leertaste oder auf "Anpfiff" um wieder zu starten.

Use-Case-Diagram



Time only runs when ball is rolling/being shot and stops when ball reaches one player

Keyboard Event with tab to start or resume game after ball left soccerfield or a goal has been shot

UI Scribble

Position randomized each time page refreshes

Anpfiff

Time

0:0

canvas 600px x 475px

Difference between active and inactive players so attributes only apply to active players

Linesman moves on x - vector if time is running

Control over position of ball with mouse click

Sprint Geschwindigkeit

Präzision

Schusskraft

Swap players by Drag & Drop

7 1 4 9 8 11

Auswechseln der Spieler

7 2 12 10 8 4

Team 01

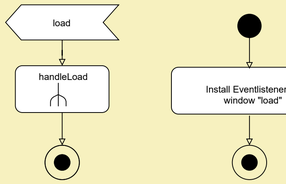
Team 02

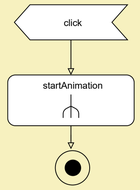
Farbe

Change tricot-color for each team by typing color names (red, blue..) or Hex codes

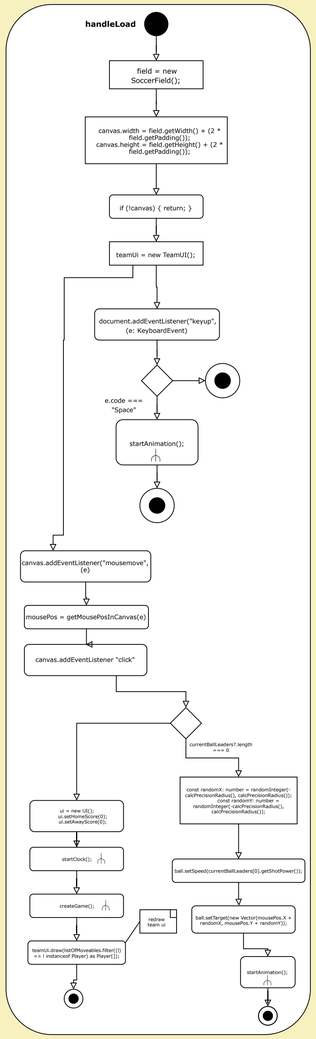
Precision Radius displayed

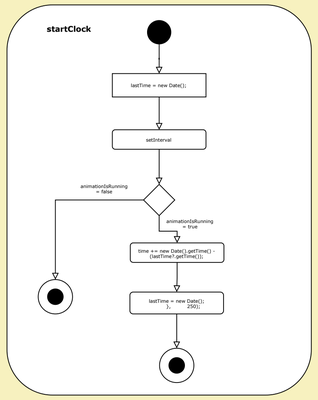
MAIN

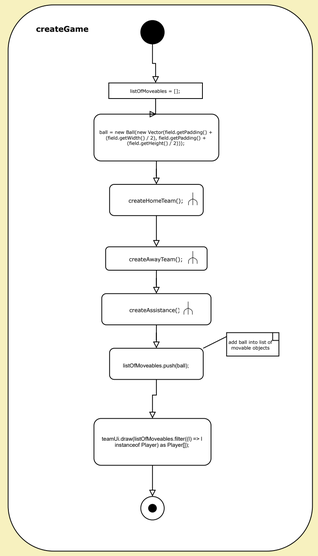


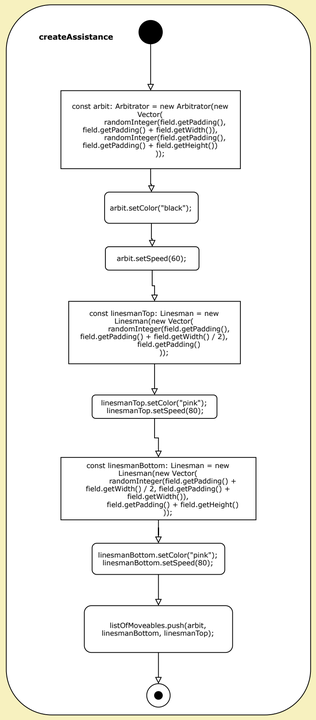


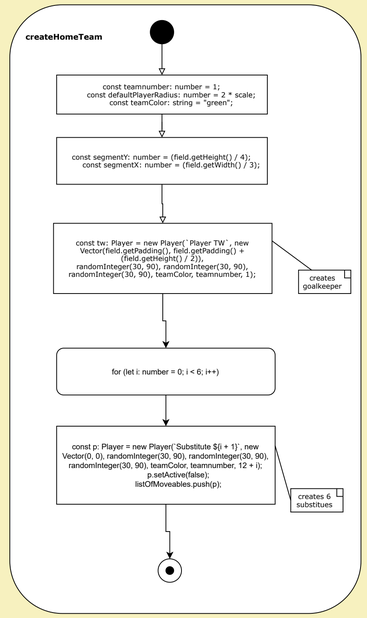

```
let canvas: HTMLCanvasElement = document.querySelector ("canvas")!;
let ctx: CanvasRenderingContext;
ctx = canvas.getContext ("2d")!;
let listOfMovables: Movable [] = [];
let currentBallLeaders: Player[] = [];
let ui: UI;
let animationIsRunning: boolean = false;
let time: number = 0;
let lastTime: Date;
let listOfMoveables: Movable[] = [];
let field: SoccerField;
let ball: Ball;
let mousePos: Vector;
```

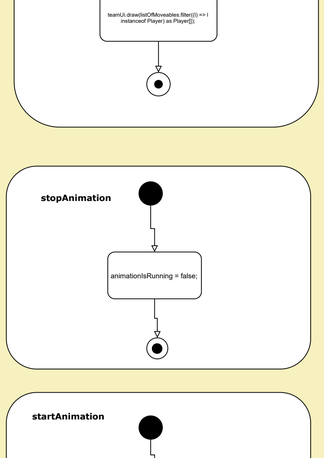


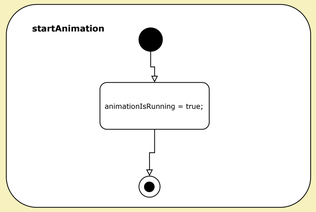


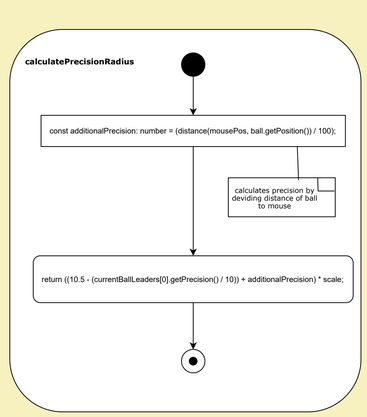






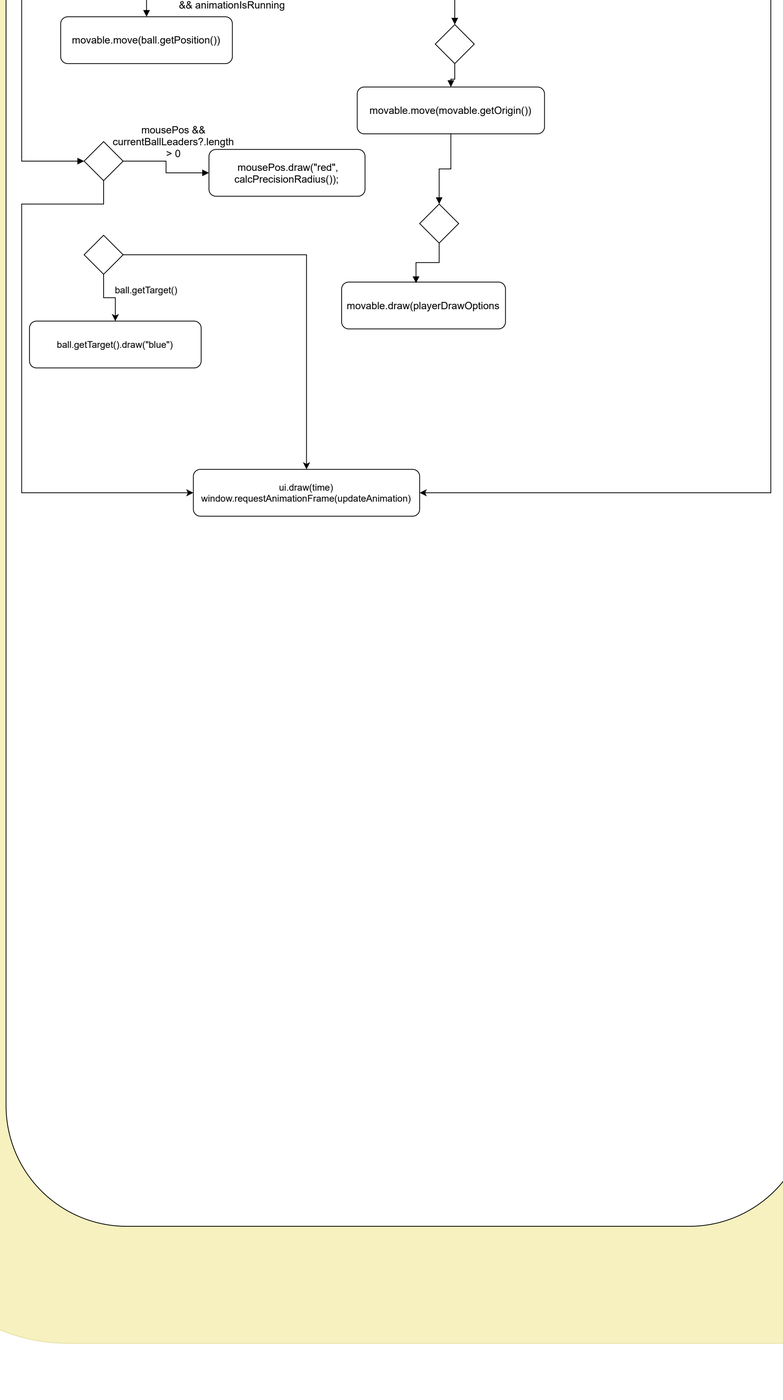






MAIN

updateAnimation
field.draw();
currentBallLeaders.length === 0
{ PlayerUI.draw(null); }
for (let movable of listOfMoveables)
movable instanceof Arbitrator
movable instanceof Linesman
animationIsRunning
animationIsRunning
movable.move(movable.getTargetFn()); }
movable.draw();
randomInteger(0, 100) > 95
randomInteger(0, 100) > 95
movable.setTarget(new Vector(
randomInteger(field.getPadding(),
field.getPadding() + field.getWidth()),
randomInteger(field.getPadding()),
field.getPadding() + field.getHeight())
));
movable instanceof Ball
ball.getTarget() && animationIsRunning
const d: number = distance(movable.getPosition(), ball.getTarget())
d > 0
movable.move(ball.getTarget())
field.isAwayGoal(movable))
movable instanceof Player
const d: number = distance(movable.getPosition(), ball.getPosition()) - movable.getRadius() - ball.getRadius();
const p: Player = movable;
const isBallLeader: number = currentBallLeaders.findIndex((l) => l.getId() === p.getId());
isBallLeader >= 0 && d > 0
currentBallLeaders.splice(isBallLeader);
ui.setHomeScore(ui.getHomeScore() + 1);
stopAnimation();
reset();
d <= 0 && isBallLeader === -1
field.isHomeGoal(movable
ui.setAwayScore(ui.getAwayScore() + 1)
stopAnimation()
reset()
(field.isOutOfBounds(movable))
stopAnimation()
reset(true)
currentBallLeaders.push(movable);
PlayerUI.draw(movable);
stopAnimation();
d <= movable.getActionRadius()


&& animationIsRunning
movable.move(ball.getPosition())
movable.move(movable.getOrigin())
mousePos &&
currentBallLeaders?.length
> 0
mousePos.draw("red",
calcPrecisionRadius());
ball.getTarget()
ball.getTarget().draw("blue")
movable.draw(playerDrawOptions
ui.draw(time)
window.requestAnimationFrame(updateAnimation)

Linesman

this.target = new Vector(_position.X, _position.Y);
_position: Vector
new Vector(_position.X, _position.Y))

Linesman draw()

public draw()

ctx.save();

ctx.beginPath();

ctx.arc(this.position.X, this.position.Y, this.getRadius() , 0, 2 * Math.PI, false);

ctx.fillStyle = this.color;
ctx.fill();

ctx.lineWidth = 1;
ctx.strokeStyle = "#003300";
ctx.stroke();

ctx.restore();

Arbitrator

_position: Vector
new Vector(_position.X, _position.Y)

Arbitrator draw()

ctx.save();

ctx.beginPath();

ctx.arc(this.position.X, this.position.Y, this.getRadius(), 0, 2 * Math.PI, false);

ctx.fillStyle = this.color;
ctx.fill();

ctx.lineWidth = 1;
ctx.strokeStyle = "#003300";

ctx.stroke();
ctx.restore();

Table

createSpan

id: string,
text: string

const s: HTMLSpanElement =
document.createElement("span");

s.setAttribute("id", id);
s.innerHTML = text;

return s;

createInput

id: string, text: string, cb: (val: string

const s: HTMLInputElement =
document.createElement("input");

s.setAttribute("id", id);
s.value = text;

s.addEventListener("change", (e: Event) =>
cb((e.target as HTMLInputElement)?.value));

return s;

createTable

(...rows: HTMLTableRowElement[]): HTMLTableElement

const table: HTMLTableElement =
document.createElement("table");

table.setAttribute("id", "player-table");

rows.forEach((r) =>
{table.appendChild(r);
});

return table;

createRow

...cells: HTMLTableCellElement[]): HTMLTableRowElement

const tr: HTMLTableRowElement =
document.createElement("tr");

cells.forEach((c) => {
tr.appendChild(c);
});

return tr;

createCell

element: HTMLElement, options: ICellOptions = { rowspan: 1, th: false }): HTMLTableCellElement

const td: HTMLTableCellElement =
document.createElement(options?.th ? "th" : "td");

td.appendChild(element);
td.setAttribute("rowspan", r?.toString());

return td;

MOVEABLE

_position: Vector

this.position = _position;
this.speed = 1;
this.speedLevel = 1;
this.slowDown = false;
this.radius = 2;

draw()

position: Vector;
speed: number;
slowDown: boolean;
speedLevel: number;
target: Vector;
color: string;
radius: number;

public setColor(color: string): void {
this.color = color;
}

public getRadius(): number {
return this.radius * scale;
}

public setRadius(radius: number): void {
this.radius = radius;
}

public setTarget(target: Vector): void {
this.target = target;
}

public getTarget(): Vector {
return this.target;
}

public getPosition(): Vector {
return this.position;
}

public setPosition(position: Vector): void {
this.position = position;
}

public getSpeed(): number {
return this.speed;
}

public setSpeed(speed: number): void {
this.speed = speed;
}

target: Vector

if (!target) { return; }

const diffVectr: Vector = new Vector(
target.X - this.position.X,
target.Y - this.position.Y);

const vectorLength: number =
Math.sqrt(Math.pow(diffVectr.X, 2) + Math.pow(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;

diffVectr.scale(scaleFactor);

this.position.add(diffVectr);

player

speedLevel: number = 2;
color: string;
speed: number = 80;
precision: number = 20;
origin: Vector = new Vector(0, 0);
id: string = uuidv4();
actionRadius: number = 30;
tricotNumber: number;
team: number;
name: string;
active: boolean;
shotPower: number = 100;
highlighted: boolean;

constructor

name: string, _position: Vector, shotPower: number = 70, precision: number = 70, speed: number = 80, color: string = "red", team: number = 0, trikotNumer: number = 0

super(
new Vector(_position.X, _position.Y));

this.shotPower = shotPower;
this.precision = precision;
this.speed = speed;
this.color = color;
this.team = team;
this.active = true;
this.tricotNumber = trikotNumer;
this.radius = 2;

this.setName(name);
this.origin = new Vector(_position.X, _position.Y);

getTricotNumber

: number

return this.tricotNumber;

setTeam

team: number

this.team = team;

getColor

return this.color;

setPrecision

prec: number

this.precision = prec;

getId

: string

return this.id;

getActionRadius

return this.actionRadius * scale;

highlight

this.highlighted = true;

setTricotNumber

tricotNumber: number

this.tricotNumber = tricotNumber;

getTeam

: number

return this.team;

setColor

color: string

this.color = color;

getPrecision

: number

return this.precision;

getName

: string

return this.name;

getOrigin

: Vector

return this.origin;

dehighlight

this.highlighted = false;

isActive

: boolean

return this.active;

setActive

active: boolean

this.active = active;

setShotPower

power: number

this.shotPower = power;

getShotPower

: number

return this.shotPower;

setName

name: string

this.name = name;

setOrigin

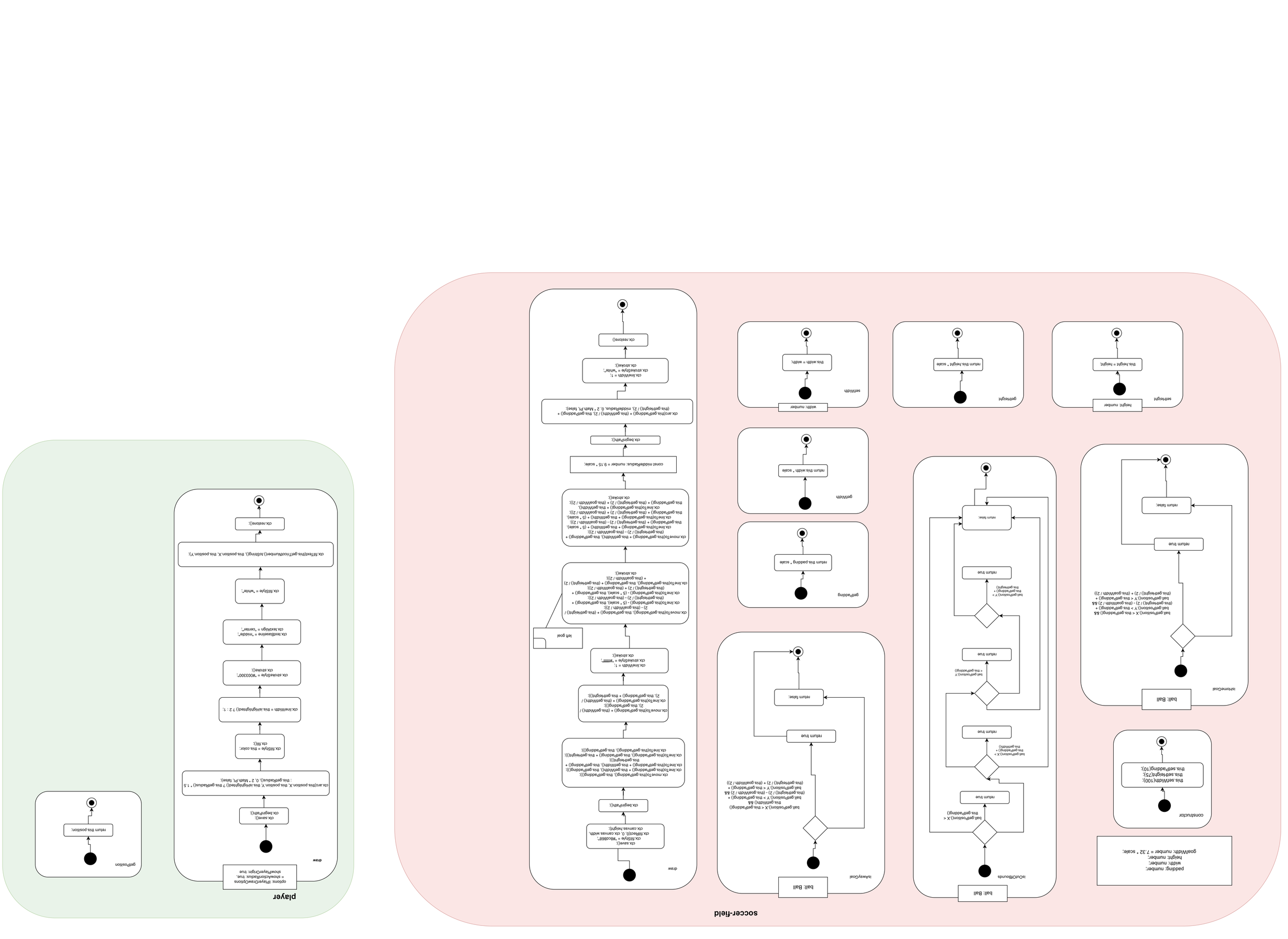
origin: Vector

this.origin = origin;

isHighlighted

: boolean

return this.highlighted;


soccer-field
padding: number;
width: number;
height: number;
goalWidth: number = 7.32 * scale;
constructor
this.setWidth(100);
this.setHeight(75);
this.setPadding(10);
ball: Ball
isOutOfBounds
ball.getPosition().X < this.getPadding()
return true
ball.getPosition().X > this.width - this.getPadding()
return true
ball.getPosition().Y < this.getPadding()
return true
ball.getPosition().Y > this.height - this.getPadding()
return true
return false;
ball: Ball
isHomeGoal
ball.getPosition().X < this.getPadding() &&
ball.getPosition().Y > this.getPadding() &&
ball.getPosition().Y < this.getPadding() + (this.getHeight() / 2) - (this.goalWidth / 2)
return true
return false;
ball: Ball
isAwayGoal
ball.getPosition().X < this.getPadding()
this.getWidth() &&
ball.getPosition().Y > this.getPadding() &&
(this.getHeight() / 2) - (this.goalWidth / 2) &&
ball.getPosition().Y < this.getPadding() +
(this.getHeight() / 2) + (this.goalWidth / 2))
return true
return false;
getWidth
return this.width * scale
getPadding
return this.padding * scale
draw
ctx.save();
ctx.fillStyle = "#dcdddd";
ctx.fillRect(0, 0, ctx.canvas.width,
ctx.canvas.height);
ctx.beginPath();
ctx.moveTo(this.getPadding(), this.getPadding());
ctx.lineTo(this.getPadding() + this.getWidth(), this.getPadding());
ctx.lineTo(this.getPadding() + this.getWidth(), this.getPadding() +
this.getHeight());
ctx.lineTo(this.getPadding(), this.getPadding() + this.getHeight());
ctx.lineTo(this.getPadding(), this.getPadding());
ctx.moveTo(this.getPadding() + (this.getWidth() /
2), this.getPadding());
ctx.lineTo(this.getPadding() + (this.getWidth() /
2), this.getPadding() + this.getHeight());
ctx.lineWidth = 1;
ctx.strokeStyle = "#ffffff";
ctx.stroke();
left goal
ctx.moveTo(this.getPadding(), this.getPadding() + (this.getHeight() /
2) - (this.goalWidth / 2));
ctx.lineTo(this.getPadding() - (5 * scale), this.getPadding() +
(this.getHeight() / 2) - (this.goalWidth / 2));
ctx.lineTo(this.getPadding() - (5 * scale), this.getPadding() +
(this.getHeight() / 2) + (this.goalWidth / 2));
ctx.lineTo(this.getPadding(), this.getPadding() + (this.getHeight() / 2)
+ (this.goalWidth / 2));
ctx.stroke();
ctx.restore()
ctx.moveTo(this.getPadding() + this.getWidth(), this.getPadding() +
(this.getHeight() / 2) - (this.goalWidth / 2));
ctx.lineTo(this.getPadding() + this.getWidth() - (5 * scale),
this.getPadding() + (this.getHeight() / 2) - (this.goalWidth / 2));
ctx.lineTo(this.getPadding() + this.getWidth() - (5 * scale),
this.getPadding() + (this.getHeight() / 2) + (this.goalWidth / 2));
ctx.lineTo(this.getPadding() + this.getWidth(),
this.getPadding() + (this.getHeight() / 2) + (this.goalWidth / 2));
ctx.stroke();
const middleRadius: number = 9.15 * scale;
ctx.beginPath();
player
options: IPlayerDrawOptions
= {showAuraRadius: true,
showPlayerOrigin: true
draw
ctx.save();
ctx.beginPath();
ctx.arc(this.position.X, this.position.Y, this.isHighlighted() ? 7 :5
: this.getRadius(), 0, 2 * Math.PI, false);
ctx.fillStyle = this.color;
ctx.fill();
ctx.lineWidth = this.isHighlighted() ? 2 : 1;
ctx.strokeStyle = "#003300";
ctx.stroke();
ctx.textBaseline = "middle";
ctx.textAlign = "center";
ctx.fillStyle = "white";
ctx.fillText(this.getTrackNumber().toString(), this.position.X, this.position.Y);
ctx.restore()
getPosition
return this.position;

Klassendiagramm

<< interface >>

ICellOptions

rowspan?: number;
th?: boolean;

<< interface >>

IPlayerDrawOptions

showActionRadius?: boolean;
showPlayerOrigin?: boolean;

CanvasRenderingContext

Moveable

#position: Vector;
#speed: number;
#slowDown: boolean;
#speedLevel: number;
#target: Vector;
#color: string;
#radius: number;
#position: Vector;
#speed: number;
#slowDown: boolean;
#speedLevel: number;
#target: Vector;
#color: string;
#radius: number;

constructor(_position: Vector)
+ move(target: Vector): void
+setColor(color: string): void
+getRadius(): number
+setRadius(radius: number): void
+setTarget(target: Vector): void
+getTarget(): Vector
+getPosition(): Vector
+setPosition(position: Vector): void
+setSpeed(speed: number): void
+getSpeed(): number

Vector

+ X: number;
+ Y: number;

constructor(_X: number, _Y: number
+ scale(_factor: number): void
+ add(_added: Vector): void
+ draw(color: string = "red", radius: number = 1): void

TeamUI

+ draw(players: Player[]): void
- createColorRow(players: Player[]): HTMLTableRowElement
- createPlayerRows(players: Player[]): HTMLTableRowElement[]
- createDraggableElement(players: Player[], player: Player, cb: () => void): HTMLSpanElement

UiHelper

+ createSpan(id: string, text: string): HTMLSpanElement
+ createInput(id: string, text: string, cb: (val: string) => void): HTMLInputElement
+ createTable(...rows: HTMLTableRowElement[]): HTMLTableElement
+ createRow(...cells: HTMLTableCellElement[]): HTMLTableRowElement
createCell(element: HTMLElement, options: ICellOptions = { rowspan: 1, th: false
}): HTMLTableCellElement

UI

- homeScore: number;
- awayScore: number;

+ createTime(playerUi: HTMLElement, time: number): void
+ updateScore(): void
+ draw(time: number): void
+ setAwayScore(awayScore: number): void
+ getHomeScore(homeScore: number): void

PlayerUI

+ draw(player: Player | null): void
- updateById(id: string, text: string): void
- createRangeInput(min: number, max: number, value: string, cb: (val: string) => void): HTMLInputElement
- createShotStrength(player: Player): HTMLInputElement
- createSpeed(player: Player): HTMLInputElement
- createPrecision(player: Player): HTMLInputElement

Arbitrator

constructor(_position: Vector)
+ draw(): void

Ball

speed: number = 0;
speedLevel: number = 10;

constructor(_origin: Vector)
+ draw(): void

Linesman

- new Vector(_position.X, _position.Y)

- targetFn: () => Vector
+ constructor(_position: Vector)
+ draw(): void
+ getTargetFn(): Vector
+ setTargetFn(cb: () => Vector): void

Soccerfield

- padding: number;
- width: number;
- height: number;
- goalWidth: number = 7.32 * scale;

constructor()
+ isOutOfBounds(ball: Ball): boolean
+ isHomeGoal(ball: Ball): boolean
+ isAwayGoal(ball: Ball): boolean
+ draw(): void
+ setPadding(padding: number): void
+ getPadding(): number
+ getWidth(): number
+ setWidth(width: number): void
+ getHeight(): number
+ setHeight(height: number): void

Player

speedLevel: number = 2;
color: string;
speed: number = 80;
#precision: number = 20;
- origin: Vector = new Vector(0, 0);
- id: string = uuidv4();
- actionRadius: number = 30;
- tricotNumber: number;
- team: number;
- name: string;
- active: boolean;
- shotPower: number = 100;
- highlighted: boolean;

constructor(name: string, _position: Vector, shotPower: number = 70, precision: number = 70, speed: number = 80, color: string = "red", team: number = 0, trikotNumer: number = 0)
+ getTricotNumber(): number
+ setTricotNumber(tricotNumber: number): void
+ isActive(): boolean
+ public draw(options: IPlayerDrawOptions = { showActionRadius: true, showPlayerOrigin: true}): void
+ isHighlighted(): boolean
+ dehighlight(): void
+ highlight(): void
+ setOrigin(origin: Vector): void
+ getOrigin(): Vector
+ getActionRadius(): number
+ setName(name: string): void
+ getName(): string
+ getId(): string
+ getTeam(): number
+ setTeam(team: number): void
+ getColor(): string
+ setColor(color: string): void
+ setShotPower(power: number): void
+ setPrecision(prec: number): void
+ getPrecision(): number
+ getShotPower(): number
+ setTricotNumber(tricotNumber: number): void

ball

_origin: Vector

constructor

super(_origin);
this.slowDown = true;
this.radius = 1.5;

draw

ctx.save();
ctx.beginPath();

ctx.arc(this.position.X, this.position.Y, this.getRadius(), 0, 2 * Math.PI, false);

ctx.fillStyle = "white";
ctx.fill();

ctx.lineWidth = 1;
ctx.strokeStyle = "black";
ctx.stroke();

ctx.restore();

speed: number = 0;
speedLevel: number = 10;

global

min: number, max: number

randomInteger

return Math.floor(Math.random() * (max - min + 1)) + min;

const scale: number = 5;

uuidv4

Universally unique identifier

return "xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx".replace(/[xy]/g, (c) =>

const r: number = Math.random() * 16 | 0, v: number = c == "x" ? r : (r & 0x3 | 0x8);

return v.toString(16)

v1: Vector, v2: Vector

distance

let d: number = Math.sqrt(Math.pow(v2.X - v1.X, 2) + Math.pow(v2.Y - v1.Y, 2));

return d;

ui

homeScore: number

setHomeScore

this.homeScore = homeScore;

this.updateScore();

getHomeScore

return this.homeScore || 0;

awayScore: number

setAwayScore

this.awayScore = awayScore;

this.updateScore();

getAwayScore

return this.awayScore || 0;

private homeScore: number;
private awayScore: number;

time: number

draw

const playerUi: HTMLElement | null = document.getElementById("upper-ui")

!playerUi

{ return; }

!playerUi

{ return; }

this.createTime(playerUi, time);

updateScore

const playerUi: HTMLElement | null = document.getElementById("upper-ui");

!playerUi

{ return; }

let scoreElement: HTMLSpanElement | null = document.getElementById("score");

!scoreElement

scoreElement = document.createElement("span");
scoreElement.setAttribute("id", "score");
playerUi.appendChild(scoreElement);

scoreElement.innerHTML = `Spielstand: ${this.getHomeScore()} : ${this.getAwayScore()}`;

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);

timeElement.innerHTML = `Zeit: ${minutes.toString().length < 2 ? `0${minutes}` : minutes}:${seconds.toString().length < 2 ? `0${seconds}` : seconds}`;

team-ui

players: Player[ ]

draw

const body: HTMLElement | null = document.getElementById("team-body")

if (!body) { return; }

body.innerHTML = ""

body.appendChild(this.createColorRow(players))

this.createPlayerRows(players).forEach((r)

body.appendChild(r)

players: Player[ ]: HTMLTableRowElement

createColorRow
creates row to adjust colors of teams

const row: HTMLTableRowElement = UIHelper.createRow

UIHelper.createCell(UIHelper.createSpan("I", "Farbe"), { th: true })

UIHelper.createCell(UIHelper.createInput("color-home", "", (v: string)

players.forEach((m)

m instanceof Player && m.getTeam() === 1

m.setColor(v)

this.draw(players)

from here: add listener to handle color changes on away team

UIHelper.createCell(UIHelper.createInput("color-away", "", (v: string)

players.forEach((m)

m instanceof Player && m.getTeam() === 2

m.setColor(v)

this.draw(players)

return row;

players: Player[], player: Player, cb: () => void): HTMLSpanElement

createDraggableElement

const s: HTMLSpanElement = document.createElement("span");

!player

return s

to enable draggability

s.setAttribute("draggable", "true");

s.addEventListener("drag", (e: DragEvent)

players.filter((p) => p.isActive()).forEach((p) =>

add listener for drag events to highlight hovered players

const v: Vector = new Vector(canvas.getBoundingClientRect().x + p.getPosition().X, canvas.getBoundingClientRect().y + p.getPosition().Y);

p.dehighlight();

false

distance(v, new Vector(e.clientX, e.clientY)) - player.getRadius() * 2 <= 0

false

e.dataTransfer

e.dataTransfer.effectAllowed = "copyMove";

p.highlight();

s.addEventListener("dragend", (e: DragEvent) =>

add listener for dragend to swap players

const p: Player = players.filter((p) => p.isActive()).find((p) => { const v: Vector = new Vector(canvas.getBoundingClientRect().x + p.getPosition().X, canvas.getBoundingClientRect().y + p.getPosition().Y);

return distance(v, new Vector(e.clientX, e.clientY)) - player.getRadius() * 2 <= 0;

(p)

p.setActive(false);
player.setActive(true);
player.setOrigin(new Vector(p.getOrigin().X, p.getOrigin().Y));
player.setPosition(new Vector(p.getPosition().X, p.getPosition().Y));
cb();

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;

players: Player[]: HTMLTableRowElement[]

createPlayerRows

const awaySubstitutes: Player[] = players.filter((p) => p.getTeam() === 2 && !p.isActive());
const homeSubstitutes: Player[] = players.filter((p) => p.getTeam() === 1 && !p.isActive());

const rowCount: number = Math.max(awaySubstitutes.length, homeSubstitutes.length);
const rows: HTMLTableRowElement[] = [];

let i: number = 0

i > rowCount = false

i < rowCount; i++

const row: HTMLTableRowElement = UIHelper.createRow

UIHelper.createCell(this.createDraggableElement(players.filter((p) => p.getTeam() == 1), homeSubstitutes[i], () => this.draw(players)))

UIHelper.createCell(this.createDraggableElement(players.filter((p) => p.getTeam() == 2), awaySubstitutes[i], () => this.draw(players)))

rows.push(row);

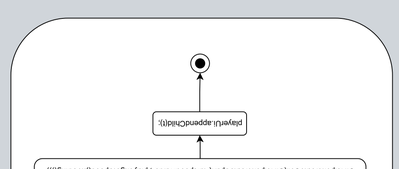
rows[0].insertBefore(UIHelper.createCell(UIHelper.createSpan("I", "Spieler")

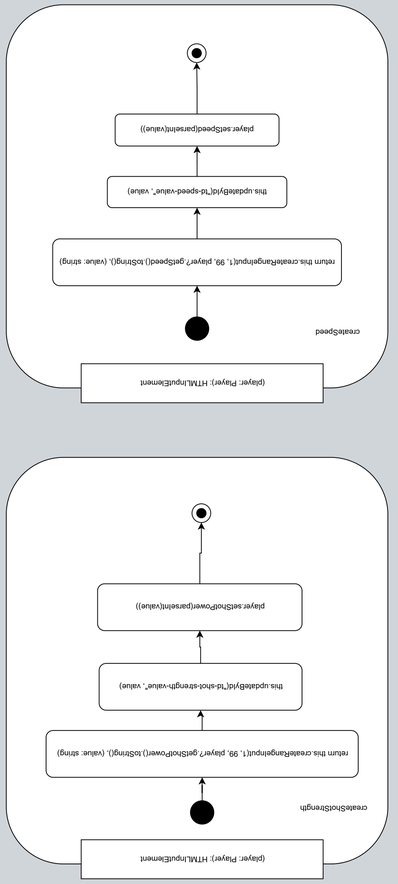
rowspan: rowCount, th: true

rows[0].children[0]

return rows;

PlayerUI





vector