Een korte inleiding tot Soccer-Fun

Peter Achten
Radboud Universiteit Nijmegen, Nederland
P.Achten@cs.ru.nl

March 10, 2017

Samenvatting

Deze notitie dient als kleine handleiding voor het programmeren van voetbalbreinen in Soccer-Fun. Er wordt aangenomen dat de lezer programmeerervaring heeft met de functionele programmeertaal Clean.

1 Het brein

In het voetbalraamwerk draait alles om het brein van voetballers. Johan Cruijff heeft in een interview met *De Tijd* (7 mei 1982) de volgende uitspraak gedaan:

> "Als ik een bal aan de voet heb die ik wil afspelen, dan moet ik rekening houden met mijn bewaker, de wind, het gras, de snelheid waarmee de spelers lopen. Wij berekenen de kracht waarmee je moet schoppen en de richting waarin in ééntiende seconde. De computer doet daar twee minuten over!"

Een andere manier om tegen deze uitspraak aan te kijken, is dat Johan Cruijff in feite zegt dat een voetballer op ieder moment een beslissing neemt, en deze beslissing laat afhangen van een groot aantal parameters. Dit is hetzelfde wat een functie doet: gegeven een aantal parameters, levert deze een resultaat op die eventueel afhankelijk is van die parameters. Dat is de aanpak waarop het voetbalraamwerk gebaseerd is. Het "brein" van een voetballer modelleren we middels een functie die een groot aantal argumenten heeft. Voor het gemak 'bundelen' we deze argumenten tot één enkel argument (een *struct* of *record*) met de naam `BrainInput`. De beslissing van het brein bundelen we ook, met naam `BrainOutput memory`. Daarnaast beschikt een voetballer over een "geheugen" (`memory`). Deze dient eveneens als argument van de breinfunctie en kan door de breinfunctie aangepast worden. De gehele breinfunctie heeft het type `FootballerAI memory` en is als volgt gedefinieerd:

```
::  FootballerAI memory :== (BrainInput, memory) -> (BrainOutput, memory)
::  BrainInput          =   { referee   :: [RefereeAction]                    1.1
                            , football  :: FootballState                      1.2
                            , others    :: [Footballer]                       1.3
```

```
                          , me        :: Footballer                    1.4
                          }
:: BrainOutput        :== FootballerAction
```

In dit hoofdstuk zullen we ieder van de componenten van de `BrainInput` de revu laten passeren, en leggen uit waar ze voor bedoeld zijn, en wat je er mee kan doen. De rol van het geheugen wordt in sectie 1.5 uitgelegd. Heel eenvoudige voetbalbreinen kunnen volstaan zonder geheugen. In hoofdstuk 2 bespreken we de mogelijke acties die een voetballer kan ondernemen. Dat zijn dus waarden van het type `BrainOutput` (dit is een synoniem-type van het type `FootballerAction`).

1.1 RefereeAction

Tijdens een wedstrijd horen spelers naar de beslissingen van de scheidsrechter te luisteren. Deze krijgt hij binnen als een lijst van `RefereeAction` waarden. Dit is een nogal uitgebreid algebraïsch data type. We laten niet de hele definitie zien (je vindt ze in `Footballer.dcl`), maar benoemen de mogelijke acties:

Vijf `RefereeAction` waarden hebben betrekking op het detecteren van overtredingen: `Hands`, `OwnBallIllegally`, `DangerousPlay`, `Offside` en `TackleDetected`. Een speler *id* die een overtreding begaat ontvangt een reprimande *r* als (`ReprimandPlayer` *id r*). De reprimande *r* kan een waarschuwing zijn (`Warning`), een gele kaart (`YellowCard`) of zelfs een rode kaart (`RedCard`). Een speler die twee gele kaarten ontvangt, ontvangt daarmee ook een rode kaart. Spelers die een rode kaart ontvangen worden normaliter van het veld verwijderd (dit is overigens afhankelijk van de scheidsrechter).

Zes `RefereeAction` waarden hebben betrekking op de duur van een wedstrijd / training: (`Pause`/`Continue`)`Game`, `EndHalf`, `GameOver`, (`AddTime` *t*), waarbij *t* het aantal minuten extra speeltijd is, en tenslotte (`GameCancelled` *mh*), waarbij *mh* eventueel het team aanwijst dat de wedstrijd wint. Een reden voor de laatste beslissing is bijvoorbeeld dat te veel spelers van een team van het veld zijn verwijderd en daardoor te weinig spelers over heeft.

Tijdens een wedstrijd moeten spelers *zelf* in de gaten houden in welke helft ze spelen. Dit kan gedetecteerd en onthouden worden met:

```
brein ( {referee}, geheugen=:{tweede_helft} )
    = (..., { geheugen & tweede_helft = tweede_helft || einde_1e_helft })
where
    einde_1e_helft = any isEndHalf referee
    ...
```

De initiële waarde in het geheugen van het veld `tweede_helft` is `False`.

Zodra de scheidsrechter beslist dat een team een goal gescoord heeft, onderneemt hij de (`Goal` *h*) actie waarbij *h* een waarde is van type `:: Home = West | East`. Deze waarde geeft de speelhelft aan van het team dat de goal heeft gescoord. Nadat een goal gescoord is voor team *h*, wordt een (`CenterKick` (`other` *h*)) toegekend. De functie `other` is gedefinieerd voor twee-waardige domeinen en levert 'de andere' waarde dan zijn argument op.

Het spel kan hervat worden door een team *h* op positie *p* met een (`DirectFreeKick` *h p*) en (`ThrowIn` *h p*) of (`GoalKick` *h*) en (`Penalty` *h*) (de laatste wordt overigens enkel geregistreerd en moet nog geïmplementeerd worden). De laatste mogelijke spelhervatting is via een (`Corner` *h e*) waar *e* een `:: Edge = North | South` waarde is. Bij een spelhervatting

zal de scheidsrechter in het algemeen spelers s_i verplaatsen naar nieuwe positie p_i om er voor te zorgen dat deze voldoende afstand bewaren tot de bal. Deze gebeurtenis is een (`DisplacePlayers` `[`(s_0, p_0),...(s_n, p_n) `]`) actie. Niet iedere overtreding leidt tot het stoppen van het spel als de benadeelde partij daar extra door benadeeld wordt. In dat geval wordt voordeel gegeven, (`Advantage` h) aan team h.

Voor trainingssessies beschikt de scheidsrechter tenslotte nog over de mogelijkheid om een tekst t aan de gebruiker te tonen met (`TellMessage` t).

1.2 FootballState

Tijdens het voetballen is er natuurlijk een bal in het spel. De voetbal kan zich vrij op of boven het voetbalveld bevinden (`Free` *bal*) of is 'in het bezit' van een voetballer met identificatie *id* (`GainedBy` *id*). Dit levert daarom de volgende representatie op van het type `FootballState`:

```
::  FootballState = Free Football | GainedBy FootballerID
```

De bal zelf wordt gerepresenteerd door een record type met de naam `Football`:

```
:: Football = { ballPos :: Position3D, ballSpeed :: Speed3D }
```

Op ieder moment bevindt de bal zich op een positie (het `ballPos` veld) en heeft het een snelheid (het `ballSpeed` veld). Laten we eerst de positie bekijken. Deze is opgesplitst in een positie op het veld, ook wel *grondpositie* genaamd, (`pxy`) die zelf een x-coordinaat en een y-coordinaat heeft (de velden `px` en `py` van het record type `Position`) en een hoogte boven het maaiveld (`pz`).

```
:: Position3D = { pxy :: Position, pz :: Metre }
:: Position   = { px  :: Metre,    py :: Metre }

m :: Real -> Metre
```

Het voordeel van deze opsplitsing is dat je snel aan de veldpositie van de bal kunt komen: als `bal` een `Football` is, dan is `bal.pxy` de grondpositie van de bal. Dat is vaak handig om snel de richting uit te rekenen waar de bal is ten opzichte van jezelf, en wat de afstand is tussen jou als speler en de grondpositie van de bal.

Afstanden worden in *meters* uitgedrukt. Je kunt een *decimaal getal* van type `Real` omzetten naar een afstand van type `Metre` met behulp van de functie `m`. Dus (`m 10.0`) is de afstand $10.0m$.

Het coördinaatstelsel van het voetbalraamwerk lijkt op het stelsel dat je tijdens de wiskunde-les hebt geleerd: de oorsprong (met coördinaten $(0, 0)$) ligt precies in het *midden* van het voetbalveld, de x-coördinaat loopt op van links naar rechts, en de y-coördinaat loopt op van onder naar boven. De verschillen zijn dat de x-coördinaat benoemd wordt met de naam `px` en de y-coördinaat met de naam `py`, en dat in plaats van decimale getallen waarden van type `Metre` gebruikt worden.

De afmetingen van een voetbalveld worden gegeven door een record met type naam `FootballField`:

```
::  FootballField = { fwidth :: FieldWidth, flength :: FieldLength }
::  FieldWidth  :== Metre
::  FieldLength :== Metre
```

De breedte van een voetbalveld (`fwidth`) mag variëren tussen (`m 64.0`) en (`m 75.0`), en de lengte (`flength`) tussen (`m 100.0`) en (`m 110.0`). Dit geeft dus de volgende hoekpunten van het voetbalveld (laat `veld` een waarde van type `FootballField` zijn):

- Het punt `{ px = scale -0.5 veld.flength, py = scale 0.5 veld.fwidth }` ligt linksboven op het voetbalveld (`Home` waarde `West` en `Edge` waarde `North`);
- het punt `{ px = scale -0.5 veld.flength, py = scale -0.5 veld.fwidth }` ligt linksonder op het voetbalveld (`Home` waarde `West` en `Edge` waarde `South`);
- het punt `{ px = scale 0.5 veld.flength, py = scale 0.5 veld.fwidth }` ligt rechtsboven op het voetbalveld (`Home` waarde `East` en `Edge` waarde `North`);
- het punt `{ px = scale 0.5 veld.flength, py = scale -0.5 veld.fwidth }` ligt rechtsonder op het voetbalveld (`Home` waarde `East` en `Edge` waarde `South`).

Deze definities laten zien dat je niet zomaar meters met elkaar kunt vermenigvuldigen (dat zou immers oppervlakte-waarden opleveren). In plaats daarvan is er een functie gemaakt, `scale`, die als eerste argument het getal krijgt waarmee je wilt vermenigvuldigen, en als tweede argument de entiteit die vermenigvuldigd moet worden (andere opties zijn het vermenigvuldigen van seconden (`Seconds`), minuten (`Minutes`), snelheid (`Velocity`), hoeken (`Angle`) en vectoren (`RVector` en `RVector3D`)). De berekening (`scale -0.5 veld.flength`) levert dus de meest westelijke x-coördinaat op die mogelijk is op het voetbalveld met de afmetingen `veld`.

Net als de positie is ook de bewegingssnelheid van de voetbal opgesplitst in een grondbewegingssnelheid (`vxy`) en een hoogte-snelheid (`vz`).

```
:: Speed3D = { vxy       :: Speed, vz       :: Velocity }
:: Speed   = { direction :: Angle, velocity :: Velocity }

rad       :: Real -> Angle
degree    :: Int  -> Angle
ms        :: Real -> Velocity
```

De grondbewegingssnelheid heeft een richting, `direction`. De richting van type `Angle` kun je opschrijven in *radialen* of *graden*. Voor *radialen* gelden de volgende waarden: (`rad 0.0`) wijst recht naar het oosten, (`rad (0.5*pi)`) wijst recht naar het noorden, (`rad pi`) wijst recht naar het westen, en (`rad (1.5*pi)`) is recht naar het zuiden gericht. Voor *graden* gelden de volgende waarden: (`degree 0`) wijst recht naar het oosten, (`degree 90`) wijst recht naar het noorden, (`degree 180`) wijst recht naar het westen, en (`degree 270`) is recht naar het zuiden gericht. Merk op dat je bij radialen met decimale getallen werkt, en bij graden met gehele getallen.

Naast de richting is er een snelheid (van type `Velocity`), zowel voor de grondsnelheid als de z-as snelheid. Deze `Velocity` wordt in meters/seconde uitgedrukt. De snelheid $10.0\frac{m}{s}$ wordt genoteerd met (`ms 10.0`). Merk op dat de `velocity` van een grondsnelheid altijd ≥ 0 is, maar dat de z-as snelheid positief kan zijn (bal beweegt omhoog), of negatief (bal beweegt naar beneden), of (`ms 0.0`) (bal beweegt noch omhoog, noch omlaag).

De meeste voetbalbreinen willen weten waar de bal is. Er is een handige functie, `getBall`, die dat voor je uitzoekt:

```
getBall :: BrainInput -> Football
```

Meestal roep je deze functie als volgt aan:

```
brein (input,geheugen)
    = ...
where
    bal = getBall input
    ...
```

1.3 Alle andere spelers

Het derde veld van het `BrainInput` record, `others`, somt alle spelers behalve jezelf op. Deze informatie heb je nodig om te bepalen waar je teamgenoten zich bevinden, of ze eventueel in balbezit zijn, of dat ze aanspeelbaar zijn (mocht je zelf in balbezit zijn of de bal ligt in je directe omgeving) – en idem dito voor de tegenstanders. Er zijn in module `FootballerFunctions` twee functies gemaakt, `team` en `opponents`, die voor jou uitzoeken wie je teamgenoten zijn en wie je tegenstanders zijn. Deze kun je bijvoorbeeld als volgt gebruiken in je breinfunctie:

```
brein (input=:{others, me},geheugen)
    = ...
where
    teamgenoten   = me team      others
    tegenstanders = me opponents others
```

In deze berekening is het resultaat `teamgenoten` alle spelers die in jouw team zitten (behalve jezelf). Het resultaat `tegenstanders` zijn alle tegenstanders. Alle spelers zijn van type `Footballer`. Deze wordt in de volgende sectie uitvoerig beschreven.

1.4 Jezelf: `Footballer`

Het laatste veld van het `BrainInput` record, `me`, representeert *jezelf*. Een vrij belangrijk argument dus. Een voetballer zelf is een record van type `Footballer`:

```
:: Footballer = E. memory:
                { playerID :: FootballerID                          1.4.1
                , name     :: String                                1.4.2
                , length   :: Length                                1.4.3
                , pos      :: Position                              1.4.4
                , speed    :: Speed                                 1.4.5
                , nose     :: Angle                                 1.4.6
                , skills   :: MajorSkills                           1.4.7
                , effect   :: Maybe FootballerEffect                1.4.8
                , stamina  :: Stamina                               1.4.9
                , health   :: Health                                1.4.10
                , brain    :: Brain (FootballerAI memory) memory    1.4.11
                }
```

Ook deze onderdelen zullen we, net als de argumenten van de breinfunctie, de revu laten passeren.

1.4.1 playerID

Het `playerID` veld identificeert de spelers. Het is een record type:

```
::  FootballerID = { clubName :: ClubName
                   , playerNr :: PlayersNumber
                   }
```

De enige restricties zijn dat de keeper `playerNr` 1 heeft, en dat alle spelers uit hetzelfde team *verschillende* `playerNr` waarden hebben, en uiteraard voor dezelfde `clubName` spelen. Het SoccerFun raamwerk gebruikt deze identificatie om spelers van elkaar te onderscheiden. Het is dus niet toegestaan dat twee teams met dezelfde `clubName` tegen elkaar spelen. De reeds aanwezige team-definities houden hier rekening mee door achter de naam de initiële speelzijde toe te voegen (`"_W"` of `"_E"`).

1.4.2 name

Dit is de naam van de speler. Dit kan elke willekeurige `String` zijn. Deze naam wordt op het scherm bij iedere speler getoond. Het is handig om alle spelers een andere (korte) naam te geven, maar dat hoeft niet.

1.4.3 length

Dit is de lengte van de speler, uitgedrukt in meters. De lengte van een speler verandert uiteraard niet tijdens een spel. De beginlengte die je opgeeft wordt door het raamwerk gecorrigeerd tot een waarde tussen de constanten `min_length` en `max_length`. Kortere personen hebben een relatief voordeel bij het bemachtigen van de bal, worden minder snel vermoeid bij sprints, kunnen beter dribbelen, lopen minder 'schade' op bij valpartijen, en zijn wendbaarder. Langere personen hebben een relatief voordeel bij hun reikwijdte bij het onderscheppen, schoppen en koppen van de bal, worden minder snel vermoeid bij looppas, en kunnen de bal harder wegtrappen.

1.4.4 pos

Dit is de huidige positie (`Position`) van de voetballer. Zoals eerder uitgelegd in sectie 1.2 is `Position` een grondcoördinaat, ofwel een paar van een x-coördinaat (`px`) en een y-coördinaat (`py`). Spelers bevinden zich dus niet in de lucht, maar maken altijd grondcontact.

1.4.5 speed

Dit is de bewegingssnelheid van de voetballer. Omdat voetballers altijd grondcontact maken, is hun bewegingssnelheid vanzelfsprekend de grondbewegingssnelheid (zie 1.2). Een speler met bewegingssnelheid `{direction = rad a, velocity = ms v}` beweegt richting (`rad a`) en heeft snelheid (`ms v`).

1.4.6 nose

Dit is de kijkrichting van de speler. De kijkrichting beïnvloedt de effectiviteit van veel acties: vooruit lopen gaat sneller dan achteruit lopen, vooruit de bal spelen gaat beter dan achteruit

of zijwaarts spelen, enzovoorts. Een speler met `nose = rad 0.0` kijkt recht naar het oosten, met `nose = rad (0.5*pi)` kijkt recht naar het noorden, enzovoorts. Een speler met `nose = rad pi` en `speed = {direction = rad (1.5*pi), velocity = ms 5.0}` kijkt recht naar het westen (vanwege diens `nose`) terwijl hij recht naar het zuiden loopt (vanwege diens `direction`) met een snelheid van $5.0\frac{m}{s}$ (vanwege `velocity`).

1.4.7 `skills`

Elke voetballer heeft de keuze uit *drie* hoofdvaardigheden. Deze leveren hem een bonus op bij uit te voeren acties. Er zijn tien vaardigheden (`Skills`) waaruit een voetballer kan kiezen:

```
:: MajorSkills :== (Skill,Skill,Skill)
:: Skill = Running  | Dribbling | Rotating   | Gaining
         | Kicking  | Heading   | Feinting   | Jumping
         | Catching | Tackling
```

Een voetballer met hoofdvaardigheid `Running` kan sneller over het veld bewegen als hij niet in balbezit is; dit is een handige eigenschap voor een verdediger. Voor een aanvaller is juist de hoofdvaardigheid `Dribbling` van meer belang: dit stelt hem in staat om beter over het veld te bewegen als hij in balbezit is. `Rotating` maakt een speler wendbaarder. `Gaining` zorgt ervoor dat een speler beter de bal kan afpakken van een andere speler. De `Kicking` vaardigheid zorgt ervoor dat je beter en met minder afwijking de bal wegschopt. Met de `Heading` vaardigheid kun je beter de bal koppen, dit is een handige combinatie voor lange spelers. `Feinting` zorgt ervoor dat je schijnbewegingen een groter bereik hebben, dit is handig als aanvaller om verdedigers te omzeilen die in de weg staan. Hoewel we gezegd hebben dat spelers altijd grondcontact maken kan het voorkomen dat ze voor een actie zoals het bemachtigen van een bal, of koppen, tijdelijk de lucht in moeten springen. Spelers met de `Jumping` vaardigheid hebben een groter bereik dan direct uit hun lengte volgt. De enige vaardigheid die van extra belang is voor de doelman is `Catching`. Hoewel iedere speler in principe de bal kan proberen te vangen, is dit natuurlijk alleen voor de doelman binnen het strafschopgebied toegestaan. Normaal gesproken zal de scheidsrechter dit als een overtreding zien. De laatste hoofdvaardigheid is `Tackling`. Hiermee kan een speler proberen een tegenstander zodanig te hinderen dat hij op de grond belandt, en dus tijdelijk is uitgeschakeld aan deelname aan het spel. Merk op dat dit kan leiden tot een afname van gezondheid van zowel de uitvoerende als getroffen speler, en dat de scheidsrechter kan beoordelen dat er sprake is van gevaarlijk spel.

Probeer in je team de juiste variatie aan te brengen van vaardigheden die passen bij de rol van de speler. Kies voor snelle, wendbare dribbelaars met een goed schot voor aanvallers, en goede lopers die de bal goed kunnen afpakken voor verdedigers. Een goede keeper is een langere voetballer die goed kan springen, en goed in balbezit kan komen (vooral met vangen).

1.4.8 `effect`

Dit is de informatie die teruggekoppeld wordt aan de voetballer om het slagen van de vorige actie door te geven. Je zult deze informatie niet direct nodig hebben. We beschrijven hem uitvoeriger als we de mogelijke acties van de voetballer beschrijven in hoofdstuk 2.

1.4.9 `stamina`

Iedere voetballer beschikt tijdens het spel over een mate van uithoudingsvermogen (`Stamina`). Dit is een waarde tussen 0.0 en 1.0. Je uithoudingsvermogen beïnvloedt al je acties: hoe vermoeider je bent, des te minder kans je hebt om een actie tot een succesvol einde te brengen. Als je je voetballer voortdurend op topsnelheid laat ronddraven, dan zul je merken dat hij meer afzwaaiers produceert, minder vaak de bal van een tegenstander zal weten te ontfutselen, enz. Vermoeidheid kun je verminderen door de speler af en toe uit te laten rusten. Dat hoeft niet per sé stilstand te betekenen, een rustige looppas kan ook voldoende zijn om van de vermoeidheid te herstellen.

1.4.10 `health`

Iedere voetballer beschikt tijdens het spel ook over een mate van gezondheid (`Health`). Net als `Stamina` is dit een waarde tussen 0.0 en 1.0, en beïnvloedt deze je prestaties. Je gezondheid kan door acties van tegenstanders verminderd worden omdat je bijvoorbeeld getackled wordt, of omdat iemand met hoge snelheid tegen je aanbotst. Dat kun je zelf natuurlijk ook doen. Het is dus verstandig om niet al te vaak tegen voetballers op te lopen, ook al zul je daar in eerste instantie waarschijnlijk geen rekening mee houden. Je gezondheid herstelt niet tijdens een wedstrijd. Het kan dus zijn dat je als speler min of meer uitgeschakeld wordt door je eigen gedrag of dat van je tegenstanders (of zelfs medespelers, hetgeen een beetje sneu is).

1.4.11 `brain`

Het laatste onderdeel van een `Footballer` is wellicht zijn meest belangrijke onderdeel, namelijk zijn brein. Het klinkt misschien vreemd dat je zou kunnen beschikken over het brein van een voetballer in de breinfunctie van een voetballer, maar gelukkig ben je niet in staat tot gedachten lezen. Het brein programmeer je en geef je aan een voetballer, maar daarna kun je 'er niet meer bij'. Het brein wordt beschermd als een *zwarte doos*, en dit gebeurt door het vreemde sleutelwoord `E. memory` dat aan het begin van een `Footballer` record staat. Deze zorgt ervoor dat je niet aan het brein van een voetballer mag komen, en je kunt dit argument dan ook gevoeglijk negeren in je eigen breinfunctie.

Een brein bestaat uit twee onderdelen, n.l. een geheugen en een breinfunctie die van het geheugen gebruik kan maken. De breinfunctie is nu juist de functie die we nu aan het beschrijven zijn. Zijn type is uitgelegd in het begin van dit hoofdstuk op blz. 1. De datastructuur die het brein beschrijft bestaat dus uit twee onderdelen:

```
:: Footballer      = E. memory:
                     { ...
                     , brain :: Brain (FootballerAI memory) memory
                     }
::  Brain ai m     = { ai    :: ai, memory :: m }
```

Het geheugen wordt hieronder uitgelegd.

1.5 Je geheugen

Naast het `BrainInput` record, heeft je voetballer ook de beschikking over een *geheugen* (`memory`) dat als argument meegegeven wordt aan je breinfunctie, en die naast de voetballeractie

opgeleverd moet worden door de breinfunctie. Dat betekent dat je de waarde van het geheugen kunt laten afhangen van de vorderingen van je speler. Het geheugen is een datastructuur die je zelf mag ontwerpen.

Het meest eenvoudige geheugen onthoudt helemaal niets. Voor een dergelijk geheugen kun je het 'flauwe' type `Void` gebruiken waar je niets mee kunt:

```
:: Void = Void
```

Dit type is voorgedefiniëerd en kun je meteen gebruiken. Een breinfunctie die vervolgens het geheugen 'met rust laat' kan dat eenvoudig doen door het argument als resultaat op te leveren:

```
brein (input, geheugen)
    = (actie, geheugen)
where
    actie =  berekening die actie oplevert
```

Een ietwat interessanter gebruik van het geheugen is om de favoriete lokatie van de speler in het veld aan te geven. Hiermee kun je er voor zorgen dat je spelers zich verdelen over het veld. Het is het handigst als je het geheugen wilt gebruiken, om hier meteen een record van te maken, omdat deze makkelijker uit te breiden zijn in toekomstige versies. Neem nu aan dat je besluit de favoriete veldpositie van een speler in zijn geheugen op te slaan. Dit kun je als volgt opschrijven:

```
:: Geheugen = { favoriete_positie :: Position }
```

Je kunt nu een breinfunctie maken die er voor zorgt dat de voetballer altijd naar zijn favoriete plekje op het veld rent. De beslissing die het brein moet nemen is: als hij binnen een acceptabele afstand van de favoriete positie is, blijf dan stilstaan; als hij niet binnen een acceptabele afstand van de favoriete positie is, ren er dan naar toe. De hoofdstructuur van je breinfunctie zal er dus als volgt uitzien:

```
brein :: (BrainInput,Geheugen) -> (BrainOutput,Geheugen)
brein ({ me }, geheugen=:{ favoriete_positie } )
| afstand_tot_favoriete_plek < acceptabele_afstand
    = (sta_stil,geheugen)
| otherwise
    = (ren_naar_positie,geheugen)
where
    afstand_tot_favoriete_plek = dist me favoriete_positie
    acceptabele_afstand        = m ...
    sta_stil                   = ...
    ren_naar_positie           = ...
```

Je zult merken dat je erg vaak de hoek zult moeten uitrekenen tussen verschillende objecten: bijv. de hoek tussen jezelf en de bal, de hoek tussen jou en een medespeler waarnaar je de bal wilt afspelen, de hoek tussen jou en de doelpalen om een schot op doel te wagen, enz. Voor deze veel voorkomende berekening kun je de functie `bearing` gebruiken:

```
bearing :: Angle base target -> Angle | toPosition base & toPosition target
```

Stel dat je de hoek wilt bepalen die er is tussen een object `base` naar een object `target`, dan kun je dit doen middels de aanroep (`bearing zero base target`). Het eerste argument van deze

functie-aanroep geeft aan ten opzichte van welke hoek je de gewenste hoek wilt berekenen. Voor verplaatsingen in absolute richting is deze hoek altijd nul (recht naar oosten). Voor nul kun je ofwel `zero` ofwel `rad 0.0` ofwel `degree 0` gebruiken.

Een compleet voorbeeld van deze functie vind je hieronder:

```
brein :: (BrainInput, Geheugen) -> (BrainOutput, Geheugen)
brein ({ me }, geheugen=:{ favoriete_positie })
| afstand_tot_favoriete_plek < acceptabele_afstand
    = (sta_stil,geheugen)
| otherwise
    = (ren_naar_positie,geheugen)
where
    afstand_tot_favoriete_plek
                        = dist me favoriete_positie
    acceptabele_afstand = m 2.0
    sta_stil            = Move zero zero
    ren_naar_positie    = Move { direction = bearing zero me favoriete_positie, velocity = ms 5.0 }
                               (bearing me.nose me favoriete_positie)
```

2 De acties van een voetballer

In het vorige hoofdstuk zijn alle argumenten besproken die de breinfunctie van een voetballer krijgt. Met behulp van deze argumenten moet de breinfunctie twee nieuwe waarden uitrekenen:

1. een *actie* om uit te voeren: dit is een waarde van type `FootballerAction`. Deze wordt in dit hoofdstuk uitvoerig besproken.

2. een nieuwe waarde van het *geheugen*. Dit is al uitvoerig aan bod gekomen in 1.5.

Voordat we de mogelijke `FootballerAction`s uitleggen, moeten we eerst twee belangrijke uitgangspunten van het voetbalraamwerk ter sprake brengen:

Het brein genereert een intentie: Een actie die bedacht wordt door het brein van een voetballer moet op de een of andere manier overgebracht worden in de 'realiteit'. Een brein zou wel kunnen bedenken dat de voetballer *nu* stil moet staan (door de waarde `(Move zero zero)` te bedenken), maar als de voetballer op dit moment aan het sprinten is, dan kan hij onmogelijk direct daarna stilstaan. De beslissingen van het voetbalbrein zijn dus *intenties*, en het voetbalraamwerk zal deze op een realistische manier omzetten naar daadwerkelijke acties.

Acties kunnen falen of afwijkingen vertonen: In het echte voetbal moet je rekening houden met het feit dat je acties niet altijd precies zo uitgevoerd worden zoals je dat zou willen. Je trapt bijvoorbeeld tegen de bal om een doelpunt te scoren, maar om de een of andere reden zwaait de bal af en vliegt meters naast het doel. Het raamwerk zorgt ervoor dat iedere actie een zekere afwijking heeft. Deze wordt groter naarmate je meer vermoeid (zie 1.4.9) of minder gezond (zie 1.4.10) bent. Je hoofdvaardigheden (zie 1.4.7) maken de desbetreffende afwijking juist kleiner.

> Voor het testen van je voetballers kan het handig zijn om het raamwerk geen afwijkingen te laten genereren. Dit kun je uitschakelen en inschakelen middels het commando Game:Mode:Predictable. Voor een potje 'echt' voetbal kun je beter het commando Game:Mode:Realistic kiezen.

In de module `Footballer` vind je een aantal functies die je kunt gebruiken om te bepalen of en wanneer het zinvol is om een bepaalde actie uit te voeren. Dit zijn de functies `maxGainReach` tot en met `maxFeintStep`. De functie `maxGainReach` levert de afstand op tot hoever je er nog in kan slagen om in balbezit te komen, en de functie `maxFeintStep` levert op de maximale afstand op die met een schijnbeweging gehaald kan worden.

Vanwege beide bovenstaande redenen wordt het voetbalbrein geïnformeerd over het succes van zijn bedachte acties in de vorm van een effect (`Maybe FootballerEffect`) in de beschrijving van de voetballer (het veld `effect`, zie 1.4.9).

Het voetbalbrein kan de volgende mogelijke acties bedenken voor een voetballer, weergegeven als het algebraïsch datatype `FootballerAction`:

```
::  FootballerAction
    =   Move     Speed Angle                                    2.1
    |   Feint    FeintDirection                                 2.2
    |   KickBall Speed3D                                        2.3
    |   HeadBall Speed3D                                        2.4
    |   GainBall                                                2.5
    |   CatchBall                                               2.6
    |   Tackle   FootballerID Velocity                          2.7
```

We bespreken elk van de mogelijke acties die een brein kan bedenken:

2.1 Move

Een van de twee manieren voor een voetballer om te bewegen is middels de actie `(Move` s a `)` (de andere manier is m.b.v. `(Feint` d `)`, zie 2.2), waarbij s een waarde van type `Speed` is, en a een `Angle`.

Waar je rekening mee moet houden is dat een voetballer een *kijkrichting* heeft. Deze is vastgelegd in het veld `nose` van de voetballer (zie 1.4.6). Lopen m.b.v. `Move` s a verandert *eerst* de kijkrichting van de voetballer met hoek a. Dit is dus *relatief*. De kijkrichting beïnvloedt de effectiviteit van de snelheid s: deze is 100% in de kijkrichting `nose` en het laagst in tegenovergestelde richting (de mate hangt af van zijn `Running` vaardigheid, zie 1.4.7). Een speler die in balbezit is, rent ook langzamer omdat hij de bal aan de voet moet houden (deze factor wordt positief beïnvloed door zijn `Dribbling` vaardigheid, zie 1.4.7).

De richting s`.direction` is *absoluut* t.o.v. de huidige richting (`nose`) van de voetballer.

Het succes van deze actie wordt teruggekoppeld als `(Moved` s' a' `)`, waarbij s' en a' de echte snelheden en rotaties zijn die je gerealiseerd hebt.

2.2 Feint

Een voetballer kan besluiten een *schijnbeweging* uit voeren middels de `(Feint` d `)` actie, waarbij d van type `FeintDirection` is. Dit houdt in dat de voetballer een schijnbeweging *naar links* kan maken (d heeft waarde `FeintLeft`) of een schijnbeweging *naar rechts* (d heeft

waarde `FeintRight`). Deze actie slaagt altijd, maar wordt beïnvloed door de vaardigheid van de voetballer en zijn snelheid. Merk op dat deze actie nooit zijn richting verandert, maar wel zijn positie.

Het succes van deze actie wordt teruggekoppeld met het `(Feinted` d `)` event, waarbij d exact dezelfde waarde heeft als aangegeven in de actie zelf.

2.3 KickBall

Je kunt besluiten de bal weg te trappen om deze bijvoorbeeld af te spelen naar een medespeler of een doelpoging te wagen middels de `(KickBall` v `)` actie, waarbij v een `Speed3D` waarde is. Dat betekent dus dat je de bal ook een hoogtesnelheid mee kunt geven. Dit is meestal verstandig omdat de luchtweerstand kleiner is dan de weerstand van de grasmat. De bal zal daardoor minder hard afgeremd worden en dus een groter bereik krijgen. Bovendien is hij lastiger te onderscheppen door tegenstanders (maar ook door je teamgenoten).

Je hoeft niet in balbezit te zijn om deze actie uit te voeren. Als de bal vrij is, en binnen je bereik ligt, dan kun je ook tegen de bal schoppen. Ook deze actie kan falen of een afwijking hebben. Het succes wordt teruggekoppeld als `(KickedBall` mv' `)` waarbij mv' een 'misschien' waarde is van type `(Maybe Speed3D)`. Als je er *niet* in geslaagd bent de bal te spelen, dan is mv' `Nothing`. Ben je er *wel* in geslaagd, dan is mv' `(Just` v' `)`, waarbij v' de uiteindelijke snelheid is waarmee de bal is gespeeld.

2.4 HeadBall

Dit is eigenlijk hetzelfde als het schoppen tegen de bal, behalve dat je het met je hoofd doet. Het voordeel van koppen is dat je de richting en snelheid van de bal kunt aanpassen terwijl deze nog in de lucht is. Je krijgt dus een sneller spelverloop.

Het succes van deze actie wordt teruggekoppeld op analoge wijze, met het effect `(HeadedBall` mv' `)`.

2.5 GainBall

Met deze actie zal de voetballer een poging doen in balbezit te komen. Dit is een voorbeeld van een actie die kan falen. Het kan zijn dat een andere speler reeds in balbezit is, en de bal blijft houden. Een andere reden kan zijn dat je speler net te vroeg besluit om de bal proberen te verkrijgen, maar dat de bal buiten bereik ligt.

Het succes van deze actie wordt teruggekoppeld als `(GainedBall` s `)` event, waarbij s de waarde `Success` (het is gelukt) of `Fail` (het is niet gelukt) kan hebben. Uiteraard kun je er ook achterkomen door de `FootballState` (1.2) parameter van je breinfunctie de volgende keer te inspecteren (en dit is ook wat meestal gebeurt).

2.6 CatchBall

De `CatchBall` actie is alleen legaal voor de doelman binnen zijn eigen strafschopgebied. De actie werkt net als de `GainBall`, en kan dus falen. Het succes van deze actie wordt gerapporteerd met het `(CaughtBall` s `)` event, waarbij s het succes aangeeft, op dezelfde wijze als bij `GainBall` in 2.5.

Veldspelers die een `CatchBall` uitvoeren kunnen bestraft worden door de scheidsrechter, evenals doelmannen die buiten hun strafschopgebied staan.

2.7 Tackle

Tot de minder sportieve onderdelen van voetbalacties horen *tackles*. Een voetballer kan besluiten een andere speler met `playerID` s te *tacklen* met een zekere snelheid v door de actie `(Tackle` s v `)`. Een succesvolle *tackle* zorgt ervoor dat het lijdend voorwerp valt en een tijd op de grond blijft liggen. Bovendien kan zijn gezondheid hierdoor afnemen. Uiteraard kan deze actie bestraft worden door de scheidsrechter, dus je loopt het risico dat je speler het veld uit gestuurd wordt.

Het succes van deze actie wordt teruggekoppeld met het event `(Tackled` s v *succes* `)`, waarbij s en v dezelfde waarden hebben, en *succes* het succes aangeeft (zoals beschreven bij `GainBall` in 2.5). De speler die ten val is gekomen komt dit te weten via het effect (`OnTheGround` n). Als dit het geval is, blijft de speler n ($n > 0$) *frames* op de grond liggen, en worden al hun beslissingen genegeerd.

3 Afronding

Je kunt een team in Soccer-Fun toevoegen door een nieuwe implementatie-module te maken. Stel dat je je team `MijnTeam` wilt noemen. Creëer een implementatie module:

```
implementation module MijnTeam

import StdEnv, StdIO, Footballer, FootballerFunctions
```

In deze module maak je een functie die als eerste argument een waarde van type `Home` verwacht en als tweede argument een waarde van type `FootballField`. Het resultaat-type van deze functie is `Team`:

```
MijnTeam :: Home FootballField -> Team
MijnTeam thuis veld
```

Het argument `thuis` heb je nodig om te weten of je een opstelling moet maken voor de `West` of de `East` zijde. Bovendien moet je de afmetingen van het veld weten, waarvoor je het tweede argument gebruikt. Een handige manier om een elftal te definiëren is als volgt:

```
MijnTeam :: Home FootballField -> Team
MijnTeam thuis veld
| thuis==West     = westTeam
| otherwise       = oostTeam
where
    club          = "Mijn Club"
    oostTeam      = mirror veld westTeam
    naam          = "MijnTeam_" +++ if (thuis == West) "W" "E"
    westTeam      = [  speler thuis veld { px = scale (-0.5*dx) veld.flength
                                         , py = scale (0.5*dy) veld.fwidth
                                         } { clubName = club, playerNr = nr }
                    \\ (dx,dy) <- west_posities
```

```
                 & nr      <- [1 ..]
                ]
west_posities = [(0.0, 0.50),(0.20,0.30),(0.20,0.70),(0.23,0.50)
                ,(0.50,0.05),(0.50,0.95),(0.60,0.50),(0.70,0.15),(0.70,0.85)
                ,(0.90,0.45),(0.90,0.55)
                ]
```

Je kunt volstaan met het bedenken van een opstelling voor een team van bijvoorbeeld de `West` zijde, zoals hier gedaan is. Om niet met omslachtige `Position` waardes te werken, worden percentages gebruikt van de helft van de veld-lengte (voor de x-coördinaten) en de helft van de veld-breedte (voor de y-coördinaten). De overloaded functie `mirror` wordt gebruikt om de opstelling en de beginrichting van alle spelers te spiegelen ten opzichte van de y-as. Omdat teams verschillende namen moeten hebben, is het verstandig om west teams een andere naam te geven dan east teams. Om een voetballer te maken hebben we nog een aparte functie geïntroduceerd, `speler`, die naast dezelfde twee argumenten nog een veldpositie en een rugnummer krijgt. Deze functie levert uiteindelijk de `Footballer` op:

```
speler :: Home FootballField Position FootballerID -> Footballer                1
speler thuis veld positie playerID=:{ playerNr }                                 2
    = { playerID    = playerID                                                   3
      , name        = "MijnNaam." <+++ playerNr                                  4
      , length      = min_length                                                 5
      , pos         = positie                                                    6
      , speed       = zero                                                       7
      , nose        = zero                                                       8
      , skills      = (Running, Kicking, Rotating)                               9
      , effect      = Nothing                                                    10
      , stamina     = max_stamina                                                11
      , health      = max_health                                                 12
      , brain       = { memory = mijngeheugen, ai = mijnbrein }                  13
      }                                                                          14
```

Het rugnummer gebruiken we om een unieke naam per speler te maken (regel 4) (hoewel dat strict gesproken niet echt nodig was, maar je ziet wel handig op het scherm welke speler wat doet). We kiezen hier voor kleine spelers (regel 5). De positie wordt aan elke speler meegegeven (regel 6). Voor spelers die op `West` beginnen ligt het voor de hand om naar het oosten te kijken, ofwel hun `speed.direction` en `nose` zijn beide `zero` (regel 7 en 8). Voor de hoofdvaardigheden hebben we gekozen voor goed rennen, trappen en draaien (regel 9). In het begin is er nog niets gebeurd (regel 10) en is iedereen nog topfit en gezond (regel 11 en 12). Tenslotte heeft de speler een brein met als functie `mijnbrein` en een `mijngeheugen` waarde (regel 13).

3.1 Integratie in *Soccer-Fun*

Om je al programmeermoeite in actie te zien, moet je tenslotte nog twee acties uitvoeren.

3.1.1 Je team exporteren

Ten eerste moet je je teamfunctie aan de buitenwereld bekend maken door de functie `MijnTeam` te *exporteren*:

```
definition module MijnTeam

import Footballer

MijnTeam :: Home FootballField -> Team
```

Vanaf nu kan iedereen het team `MijnTeam` gebruiken. Daarvoor moet je eerst de module `MijnTeam` *importeren*:

3.1.2 Je team importeren

In *Soccer-Fun* worden alle teams verzameld in de module `Team.icl`. De allereerste functie in deze module heet `allAvailableTeams` en deze doet niets anders dan alle functies opnoemen van hetzelfde type als `MijnTeam`. Voordat dat kan, moeten deze functies *geïmporteerd* worden. In `Team.icl` ziet dat er als volgt uit:

```
implementation module Team                                          1
                                                                    2
import StdEnvExt                                                    3
import Footballer                                                   4
import TeamMiniEffie                                                5
import Team_Opponent_Slalom_Assignment                              6
import Team_Opponent_Passing_Assignment                             7
import Team_Opponent_DeepPass_Assignment                            8
import Team_Opponent_Keeper_Assignment                              9
import Team_Student_Slalom_Assignment                               10
import Team_Student_Passing_Assignment                              11
import Team_Student_DeepPass_Assignment                             12
import Team_Student_Keeper_Assignment                               13
import MijnTeam                                                     14
                                                                    15
allAvailableTeams :: [Home FootballField -> Team]                   16
allAvailableTeams = [ Team_MiniEffies, Harmless                     17
                    , Team_Student_Slalom                           18
                    , Team_Student_Passing                          19
                    , Team_Student_DeepPass                         20
                    , Team_Student_Keeper                           21
                    , Team_Opponent_Slalom                          22
                    , Team_Opponent_Passing                         23
                    , Team_Opponent_DeepPass                        24
                    , Team_Opponent_Keeper                          25
                    , MijnTeam                                      26
                    ]                                               27
```

In regel 14 wordt `MijnTeam` geïmporteerd, en in regel 26 wordt het in alle beschikbare teams toegevoegd. Vanaf dit moment (na opnieuw compileren), is `MijnTeam` officiëel toegetreden tot de *Soccer-Fun* competitie! Je kunt het team selecteren middels het commando Game:Match.

`TeamMiniEffie` bestaat uit spelers die allemaal achter de bal aan rennen en deze in de goal van de tegenstander proberen te schoppen. De teams waarvan de namen starten met `Team_Opponent_` en `Team_Student_` hebben betrekking op de trainings-oefeningen. Marc School-

derman heeft een team gecreëerd waartegen je goed kunt oefenen, met de naam `Team_Harmless`. Je hebt geen toegang tot de bron-code van dit team. Als je er in slaagt dit team te verslaan dan heb je een goede kans om kampioen te worden.