Een korte inleiding tot Soccer-Fun

Peter Achten

Radboud Universiteit Nijmegen, Nederland
versie 18 mei 2009
P.Achten@cs.ru.nl

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 memory`. De beslissing van het brein bundelen we ook, met naam `BrainOutput memory`:

```
:: BrainInput   memory =  { referee   :: [RefereeAction]          1.
                          , football  :: FootballState            2.
                          , others    :: [Footballer]             3.
                          , me        :: Footballer               4.
                          , memory    :: memory                   5.
                          }
:: BrainOutput  memory :== (FootballerAction,memory)
:: FootballerAI memory :== (BrainInput memory) -> BrainOutput memory
```

In dit hoofdstuk zullen we ieder van deze argumenten de revu laten passeren, en leggen uit waar ze voor bedoeld zijn, en wat je er mee kan doen. In het volgende hoofdstuk, 2, bespreken we de mogelijke acties die een voetballer kan ondernemen. Dat zijn dus waarden 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:

Zeven `RefereeAction` waarden hebben betrekking op het detecteren van overtredingen: `Hands`, `OwnBallIllegally`, `DangerousPlay`, `Offside`, and (`Tackle`/`Schwalbe`/`Theater`)`Detected`. Een speler *id* die een overtreding begaat ontvangt een reprimande *r* als (`ReprimandPlayer` *id r*). Er zijn vijf acties die betrekking hebben op de duur van een wedstrijd / training: (`Pause`/`Continue`)`Game`, `EndHalf`, `GameOver`, en (`AddTime` *t*), waarbij *t* het aantal minuten extra speeltijd is. Tijdens een wedstrijd moeten spelers *zelf* in de gaten houden in welke helft ze spelen. Dit kan eenvoudig gedetecteerd worden met:

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

Zodra de scheidsrechter beslist dat een team een goal gescoord heeft, onderneemt hij de (`Goal` *t*) actie waarbij *t* een waarde is van type `:: ATeam = Team1 | Team2`. `Team1` is het team dat de wedstrijd op de westelijke zijde begonnen is (`West`), en `Team2` is het andere team. Nadat een goal gescoord is voor team *t*, wordt een (`CenterKick` (`other` *t*)) toegekend. De functie `other` is gedefiniëerd voor twee-waardige domeinen en levert 'de andere' waarde dan zijn argument op.

Het spel kan hervat worden door een team *t* op positie *p* met een (`DirectFreeKick` *t p*) en (`ThrowIn` *t p*) of (`GoalKick` *t*) en (`Penalty` *t*) (de laatste wordt overigens enkel geregistreerd en moet nog geïmplementeerd worden). De laatste spelhervatting is via een (`Corner` *t 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` $[(s_0, p_0), \ldots (s_n, p_n)]$) 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` *t*) aan dat team *t*.

Voor trainingssessies beschikt de scheidsrechter tenslotte nog over de mogelijkheid met de gebruiker te spreken. Een tekst *t* wordt met (`TellMessage` *t*) getoond.

1.2 FootballState

Tijdens het voetballen is er natuurlijk een bal in het spel. Deze 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`). Alle dimensies worden in meters uitgedrukt.

```
:: Position3D = { pxy :: Position, pz :: ZPos }
:: Position   = { px  :: XPos,     py :: YPos }
:: XPos     :== Metre
:: YPos     :== Metre
:: ZPos     :== 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 positie van de bal als deze op het veld zou liggen. Dat is vaak handig om snel de richting uit te rekenen waarin de bal ligt, en wat de afstand is tussen jou als speler en de bal.

Het coördinaatstelsel van het voetbalraamwerk is een klein beetje anders dan wat je normaal tijdens de wiskunde-les hebt geleerd: daar ligt de oorsprong (met coördinaten $(0, 0)$) van een grafiek meestal linksonder, terwijl de x-coördinaat oploopt van links naar rechts, en de y-coördinaat oploopt van onder naar boven. In het voetbalraamwerk ligt de oorsprong juist *linksboven*, en loopt de y-coördinaat op van boven naar beneden. 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 mag variëren tussen 64m en 75m, en de lengte tussen 100m en 110m). Dit geeft dus de volgende hoekpunten van het voetbalveld (laat `veld` een waarde van type `FootballField` zijn):

- Het punt { `px = 0.0`, `py = 0.0` } (dat je ook mag noteren als `zero`) ligt linksboven op het voetbalveld;
- het punt { `px = 0.0`, `py = veld.fwidth` } ligt linksonder op het voetbalveld;
- het punt { `px = veld.flength`, `py = 0.0` } ligt rechtsboven op het voetbalveld;
- het punt { `px = veld.flength`, `py = veld.fwidth` } ligt rechtsonder op het voetbalveld.

Ook de snelheid is opgesplitst in een grondsnelheid (`vxy`) en een z-as snelheid (`vz`). De grondsnelheid heeft een richting (`direction`) die gemeten wordt in radialen. Hierbij gelden de volgende waarden: 0.0 radialen is naar het oosten, $\frac{1}{2}\pi$ radialen is naar het zuiden, π radialen is naar het westen, en $\frac{3}{2}\pi$ radialen is naar het noorden gericht. 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. Merk op dat de `velocity` van een grondsnelheid altijd ≥ 0 is, en dat de z-as snelheid positief kan zijn (bal beweegt omhoog), of negatief (bal beweegt naar beneden), of 0.0 (bal beweegt noch omhoog, noch omlaag).

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

Nu we de `Football` uitvoerig besproken hebben, kunnen we het hebben over het feit dat het tweede veld, `football`, van het `BrainInput` record niet type `Football` heeft, maar type `FootballState`. De bal bevindt zich ofwel vrij op het veld (en kan in principe door iedere speler in bezit genomen worden) ofwel de bal wordt bezeten door een speler. Als de bal vrij is, dan geeft het

tweede argument van de breinfunctie de bal als (`Free bal`) waarde weer. Als de bal in bezit van een speler is, dan is deze waarde (`GainedBy speler`), waarbij `speler` de speler identificeert die in balbezit is. Omdat je bijna altijd wilt weten waar de bal is, is er een handige functie die *altijd* de bal oplevert, mits deze toegepast wordt op de `FootballState` en *alle* spelers op het veld. Deze functie heet `getFootball` en hij heeft het volgende type:

```
getFootball :: FootballState [Footballer] -> Football
```

Meestal roep je deze functie als volgt aan (regel 6 en 7):

```
brein :: (BrainInput memory) -> BrainOutput memory                    1.
brein { football, others, me }                                        2.
    = ...                                                             3.
where                                                                 4.
    alle_spelers = [me : others]                                      5.
    de_bal       = getFootball football alle_spelers                  6.
```

Een veel voorkomende fout is overigens dat vergeten wordt jezelf bij de lijst van spelers toe te voegen (regel 5). Als jij dan in balbezit bent, zal het programma een *runtime error* genereren.

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. Je kunt gemakkelijk onderscheid maken tussen teamgenoten en tegenstanders:

```
brein :: (BrainInput memory) -> BrainOutput memory
brein { others, me }
    = ...
where
    (teamgenoten,tegenstanders) = spanfilter (sameClub me) others
```

1.4 Jezelf: `Footballer`

Het één-na-laatste veld van het `BrainInput` record, `me`, representeert *jezelf*. Een vrij belangrijk argument dus als je iets over jezelf te weten wilt komen. Een voetballer zelf is een waarde van type `Footballer` en dit is een vrij uitgebreide record structuur:

```
:: Footballer = E. memory:
                { playerID :: FootballerID                            1.
                , name     :: String                                  2.
                , length   :: Length                                  3.
                , pos      :: Position                                4.
                , speed    :: Speed                                   5.
                , nose     :: Angle                                   6.
                , skills   :: MajorSkills                             7.
                , effect   :: Maybe FootballerEffect                  8.
                , stamina  :: Stamina                                 9.
```

```
, health   :: Health                                                    10.
, brain    :: Brain (FootballerAI memory) memory                        11.
}
```

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

1. **playerID:** dit is de identificatie die door het raamwerk gebruikt wordt om spelers te identificeren. Het is een record:

   ```
   :: 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.
2. **name:** dit is de naam van de speler. Dit kan elke willekeurige `String` zijn. Deze naam wordt in het voetbalraamwerk op het scherm bij iedere voetbalspeler getoond. Het is dus wel handig om alle spelers een andere naam te geven, maar dat hoeft niet.
3. **length:** dit is de lengte van de speler, uitgedrukt in meters (als een `Real` waarde). 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` (1.60m) en `max_length` (2.10m). Kortere personen hebben een relatief voordeel bij het bemachtigen van de bal, worden minder snel vermoeid bij sprints, kunnen beter dribbelen, lopen minder 'schade' 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.
   Het is dus verstandig een mengeling van verschillende lengtes in je team aan te brengen. Dat kan dus eenvoudig door de lengte af te wisselen.
4. **pos:** dit is de huidige positie (`Position`) van de voetballer. Zoals eerder uitgelegd in 1.2 is `Position` een grondcoördinaat, ofwel een paar van een x-coördinaat en een y-coördinaat. Spelers bevinden zich dus niet in de lucht, maar maken altijd grondcontact.
   De afstand tussen twee posities (of dat nu `Position` of `Position3D` waarden zijn, kun je uitrekenen met de functie `dist` die twee van dit soort argumenten wil hebben, en de afstand uitgedrukt in meters oplevert.
5. **speed:** dit is de snelheid van de voetballer. Omdat voetballers altijd grondcontact maken, is hun snelheid vanzelfsprekend de grondsnelheid (zie eveneens 1.2). Deze snelheid is altijd *absoluut*: je kunt dus ongeacht je kijkrichting (volgende onderdeel) iedere kant oplopen.
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 schieten, enz. De richtingswaarde is *absoluut*. Hierbij is 0.0 richting oosten, `0.5*pi` richting zuiden, `pi` richting westen, en `1.5*pi` richting noorden.
7. **skills:** elke voetballer heeft de keuze uit *drie* hoofdvaardigheden. Deze leveren hem een bonus op bij uit te voeren acties. Er zijn twaalf vaardigheden (`Skills`) waaruit een voetballer kan kiezen:

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

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. Tenslotte zijn er drie vaardigheden die met 'realistisch' voetbal te maken hebben: `Tackling`, `Schwalbing` en `PlayingTheater`. De eerste laat je beter een tackle uitvoeren waarmee je een tegenstander kunt beschadigen. De tweede laat je beter net doen alsof je net getackled bent door een tegenstander, en de laatste is bedoeld om beter in staat te zijn de scheidsrechter om de tuin te leiden en net te doen of je gewond bent door een actie van een tegenstander. In deze laatste drie gevallen kan een scheidsrechter er in trappen en een tegenstander bestraffen, of je spel doorzien en jou van een gele of rode kaart voorzien.

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 met goede pak, sprong en vangvaardigheid.

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.
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.
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).

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                11.
                     }
:: Brain ai memory = { m :: memory, ai :: ai }
```

Het geheugen wordt hieronder uitgelegd.

1.5 Je geheugen

Het laatste veld van het `BrainInput` record, `memory`, is je *geheugen*. Dit is een datastructuur die je zelf mag ontwerpen, en deze kan eenvoudig zijn, maar ook erg ingewikkeld, hetgeen natuurlijk afhankelijk is van de mate van geavanceerdheid van je voetbalbrein. Het geheugen is, naast de voetbalactie die de breinfunctie moet opleveren, het enige andere resultaat van de breinfunctie. Dat betekent dat je de waarde van het geheugen kunt laten afhangen van de vorderingen van je speler.

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 :: (BrainInput Void) -> BrainOutput Void                                      1.
brein input=:{ memory }                                                             2.
    = (actie, memory)                                                               3.
where                                                                               4.
    actie =  berekening die actie oplevert                                          5.
```

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, memory }
| afstand_tot_favoriete_plek < acceptabele_afstand
    = (sta_stil,memory)
| otherwise
    = (ren_naar_positie,memory)
where
    ...
```

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. Omdat dit een veelvoorkomende berekening is, zijn er twee functies die je hiervoor goed kunt gebruiken:

```
angleWithObject       ::  Position        Position -> Angle
angleWithObjectForRun :: (Position,Angle) Position -> Angle
```

De meest algemene functie is `angeWithObject` die twee `Position` waardes krijgt en de hoek daartussen uitrekent. De uitkomst van deze functie kun je gebruiken voor het afgeven van de bal, of koppen, en dergelijke. Deze functie is echter minder geschikt voor het lopen van een speler: een speler heeft immers een kijkrichting (`nose`) waar je rekening mee zult moeten houden. Stel nu dat je huidige positie *pos* is, en je huidige kijkrichting r is, en je wilt positie *doel* bereiken. Dat bereken je middels `angleWithObjectForRun` (*pos*,r) *doel*. Een voorbeeld van deze functie vind je hier onder, waarin we alle details van de breinfunctie kunnen invullen:

```
brein :: (BrainInput Geheugen) -> BrainOutput Geheugen
brein { me, memory }
| afstand_tot_favoriete_plek < acceptabele_afstand
                      = (sta_stil,        memory)
| otherwise            = (ren_naar_positie,memory)
where
    mijn_plekje       = memory.favoriete_positie
    afstand_tot_favoriete_plek
                      = dist me.pos mijn_plekje
    acceptabele_afstand = 5.0
    sta_stil          = Move zero zero
    ren_naar_positie  = Move { direction = me.nose+richting, velocity = loopsnelheid } richting
    richting          = angleWithObjectForRun (me.pos, me.nose) mijn_plekje
    loopsnelheid      = 5.0
```

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 uiteindelijk 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 bent (zie 1.4.8), minder gezond (zie 1.4.9). Je hoofdvaardigheden (zie 1.4.5) 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 moet je deze wel ingeschakeld hebben.

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                          1.
    |   Feint     FeintDirection                       2.
    |   KickBall  Speed3D                              3.
    |   HeadBall  Speed3D                              4.
    |   GainBall                                       5.
    |   CatchBall                                      6.
    |   Tackle    FootballerID Velocity                7.
    |   Schwalbe                                       8.
    |   PlayTheater                                    9.
```

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

1. **Move** s a**:** 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 volgende punt), waarbij s een waarde van type `Speed` is, en a een `Angle`.
   Waar je rekening mee moet houden is dat een voetballer een *richting* heeft. Deze is vastgelegd in het veld `nose` van de voetballer. Lopen m.b.v. `Move` s a verandert *eerst* de kijkrichting van de voetballer met a radialen. 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 1.4.5). 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 1.4.5).
   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. **Feint** d**:** 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.
3. **KickBall** v**:** 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.
4. **HeadBall** v**:** 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').
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).

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 punt 5.
   Veldspelers die een `CatchBall` uitvoeren kunnen bestraft worden door de scheidsrechter, evenals doelmannen die buiten hun strafschopgebied staan.
7. **Tackle** s v**:** 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 *fbID* en v dezelfde waarden hebben, en *succes* het succes aangeeft (zoals beschreven bij `GainBall` in punt 5).
8. **Schwalbe:** Een speler kan net doen alsof hij gevallen is vanwege een onoirbare actie van een tegenstander door de `Schwalbe` actie. Het gevolg is dat hij een aantal ronden op de grond zal liggen, en geen verdere acties kan uitvoeren. Ook hier loopt de speler het risico dat hij bestraft wordt door de scheidsrechter, maar het kan ook zo zijn dat een tegenstander bestraft wordt voor gevaarlijk spel.
   De speler wordt geïnformeerd over deze actie middels het `Schwalbed` event.
9. **PlayTheater:** De laatste actie die helaas bij realistisch voetbal hoort is dat een voetballer net doet of hij gewond is geraakt door een actie van een tegenstander d.m.v. de actie `PlayTheater`. Deze actie zal tijdelijk de gezondheid van de speler verlagen, zodat de scheidsrechter eventueel kan beslissen dat voor deze plotseling afname van gezondheid een tegenspeler ter verantwoording geroepen moet worden. De scheidsrechter kan echter ook de speler die de `PlayTheater` actie inzet bestraffen.
   De speler wordt geïnformeerd over deze actie middels het `PlayedTheater` event.

Met deze beschrijving van alle mogelijke `FootballerActions` hebben we eveneens bijna alle `FootballerEffects` besproken. Er is er nog één over:

OnTheGround n**:** Spelers kunnen *getackled* worden, en daardoor ten val komen. Als dit het geval is, blijven ze n ($n > 0$) ronden op de grond liggen, en worden al hun beslissingen genegeerd.

3 Afronding

Je kunt een team in *Soccer-Fun* integreren door een functie te maken die twee argumenten krijgt, n.l. `Home` en `FootballField`, en die een team oplevert: `Team`. Laten we deze functie `MijnTeam` noemen. Wat we dus moeten invullen is:

```
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                                          1.
MijnTeam thuis veld                                                             2.
| thuis==West     = westTeam                                                    3.
| otherwise       = oostTeam                                                    4.
where                                                                           5.
    club          = "Mijn Club"                                                 6.
    oostTeam      = mirror veld westTeam                                        7.
    naam          = "MijnTeam_" ++ if (thuis == West) "W" "E"                   8.
    middenlijn_x  = veld.flength/2.0                                            9.
    breedte_veld  = veld.fwidth                                                 10.
    westTeam      = [  speler thuis veld {px=dx*middenlijn_x,py=dy*breedte_veld} 11.
                                         {clubName=club,playerNr=nr}            12.
                    \\ (dx,dy) <- west_posities_spelers                         13.
                     & nr      <- [1..]                                         14.
                    ]                                                           15.
    west_posities = [(0.0, 0.50),(0.20,0.30),(0.20,0.70),(0.23,0.50)            16.
                    ,(0.50,0.05),(0.50,0.95),(0.60,0.50),(0.70,0.15),(0.70,0.85) 17.
                    ,(0.90,0.45),(0.90,0.55)                                    18.
                    ]                                                           19.
```

Je kunt volstaan met het bedenken van een opstelling voor een team van bijvoorbeeld de `West` zijde, zoals hier gedaan is. De overloaded functie `mirror` wordt gebruikt om de opstelling en de beginrichting van alle spelers te spiegelen. 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.
      }
```

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` is $0.0 \cdot \pi$ (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.

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*:

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.

import StdEnvExt                                                    2.
import Footballer                                                   3.
import TeamMiniEffie                                                4.
import Team_Opponent_Slalom_Assignment                              5.
import Team_Opponent_Passing_Assignment                             6.
import Team_Opponent_DeepPass_Assignment                            7.
import Team_Opponent_Keeper_Assignment                              8.
import Team_Student_Slalom_Assignment                               9.
import Team_Student_Passing_Assignment                              10.
import Team_Student_DeepPass_Assignment                             11.
import Team_Student_Keeper_Assignment                               12.
import MijnTeam                                                     13.

allAvailableTeams :: [Home FootballField -> Team]                   14.
allAvailableTeams = [ Team_MiniEffies                               15.
                    , Team_Student_Slalom                           16.
                    , Team_Student_Passing                          17.
                    , Team_Student_DeepPass                         18.
                    , Team_Student_Keeper                           19.
                    , Team_Opponent_Slalom                          20.
                    , Team_Opponent_Passing                         21.
```

```
    , Team_Opponent_DeepPass                                    22.
    , Team_Opponent_Keeper                                      23.
    , MijnTeam                                                  24.
    ]                                                           23.
```

In regel 13 wordt `MijnTeam` geïmporteerd, en in regel 24 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 als het programma loopt als het westelijke team door het menu Team1:MijnTeam_W te kiezen, en als oostelijke team door het menu Team2:MijnTeam_E te kiezen.

Je kunt nu `MijnTeam` een partij voetbal laten spelen en ideeën krijgen voor betere voetballers.