Die WWUScorchers

Realisierung einer RoboCup Soccer Server Mannschaft

Projektseminar: „Neuroinformatik und Agenten - Lernende Systeme“

Robin Baumgarten, Daniel Beckmann, Philip Harborth,
Philipp Kühl, Marcel Scheibmayer, Thomas Waldeyer

26. September 2006

Gliederung

- **Einleitung**
- Aufbau der Mannschaft
- Nutzung künstlicher Intelligenz
- Bewertung und Zusammenfassung

Überblick

- RoboCup
  - **Zielsetzung:** humanoide Roboterfußballmannschaft im Jahr 2050, die gegen aktuellen Weltmeister gewinnt
  - Verschiedene Ligen, unter anderem die **Simulationsliga**
  - **Multiagentensimulation**!
- Spieler
  - Erfüllen die Voraussetzungen eines **Agenten** (autonom, proaktiv, reaktiv, kommunikativ, lernfähig, [...])
- Neuroinformatik
  - Einsatz und Nutzung von **künstlicher Intelligenz** beim RoboCup: Entscheidungsfindung in den softwarerealisierten Spielern

Projektübersicht

- Projektvorgaben:
  - **Entwicklung einer konkurrenzfähige Mannschaft** für den RoboCup in Java
  - Nutzung von **künstlicher Intelligenz**
- Herausforderungen:
  - Projektplanung, -koordination und -durchführung
    - Aufgabendefinition, -verteilung und –durchführung
    - Ablaufplanung, -kontrolle
  - Stetiger Austausch zwischen den Seminarteilnehmern
  - **Evolutionäres, inkrementelles Modell** (stufenweise, Einbeziehung gewonnener Erfahrungen in die Entwicklung)

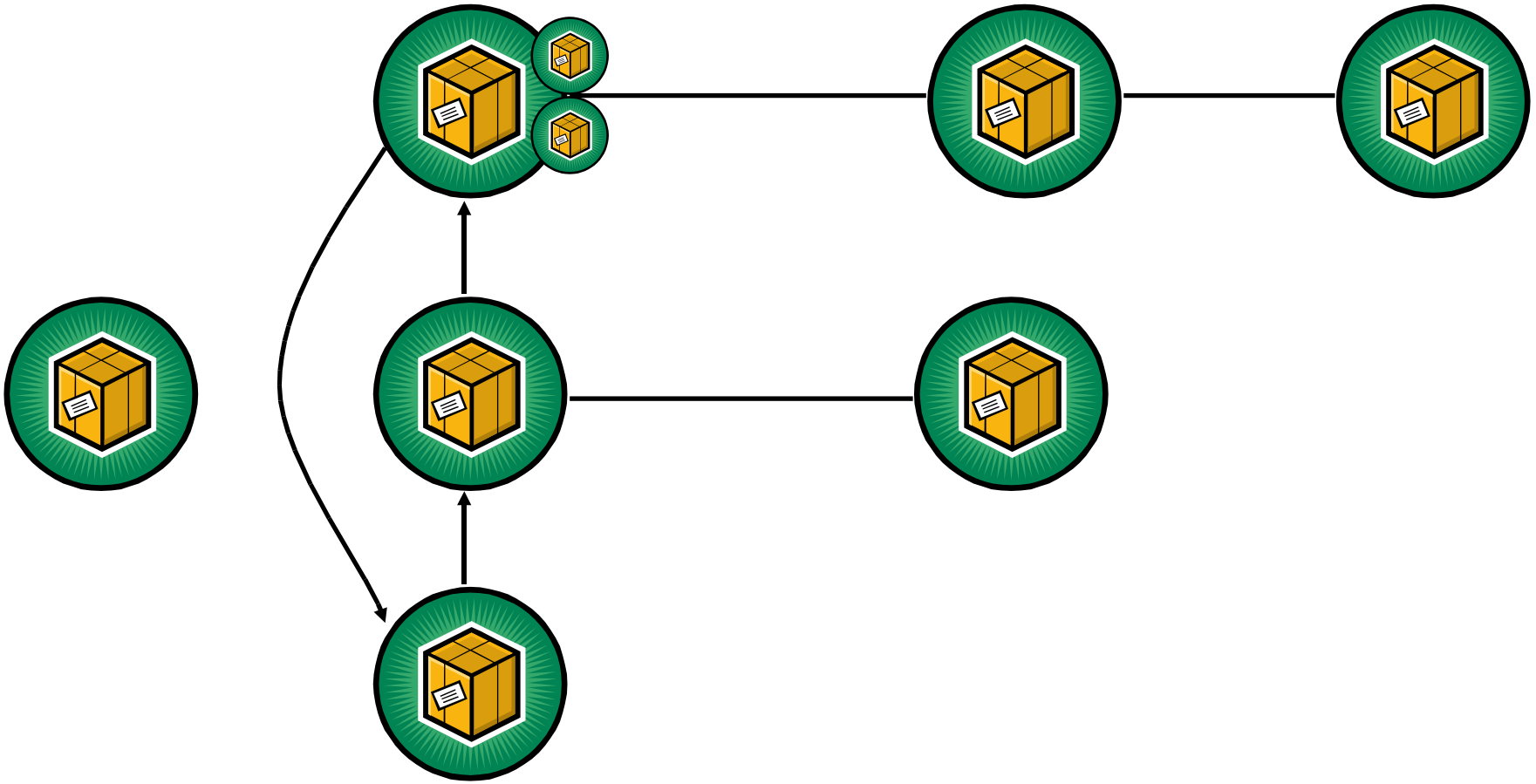
Projektablauf

- Vorüberlegung, Projektumriss, Planung
- **Aufbau** der Mannschaft
  - Basisfunktionen, essentielle Klassen
  - Evolutionäre und inkrementelle Entwicklung
  - **Spiel ohne KI**
- **Integration der KI** in die Mannschaft
  - Einarbeitung in, Nutzung und Einsatz von künstlicher Intelligenz
  - **Spiel mit KI**
- **Bewertung** der Mannschaft
  - Kritische Betrachtung der Mannschaft
  - Vergleich „mit KI vs. ohne KI“

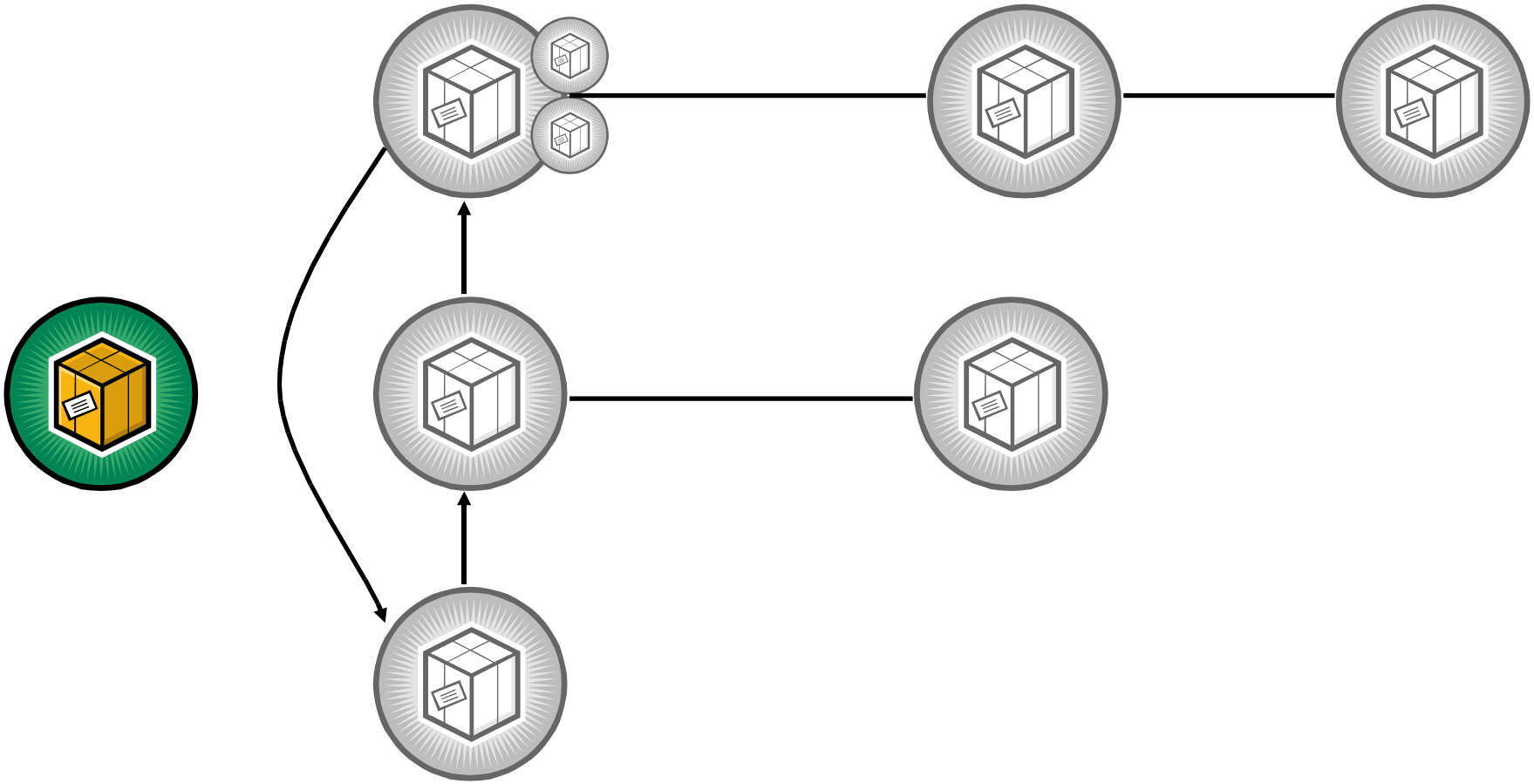
Gliederung

- Einleitung
- **Aufbau der Mannschaft**
- Nutzung künstlicher Intelligenz
  - Aufbau unserer Java-Pakete
    - basics
    - communication
    - information
    - objects
    - logic
- Bewertung und Zusammenfassung

Überblick über die Java -Pakete



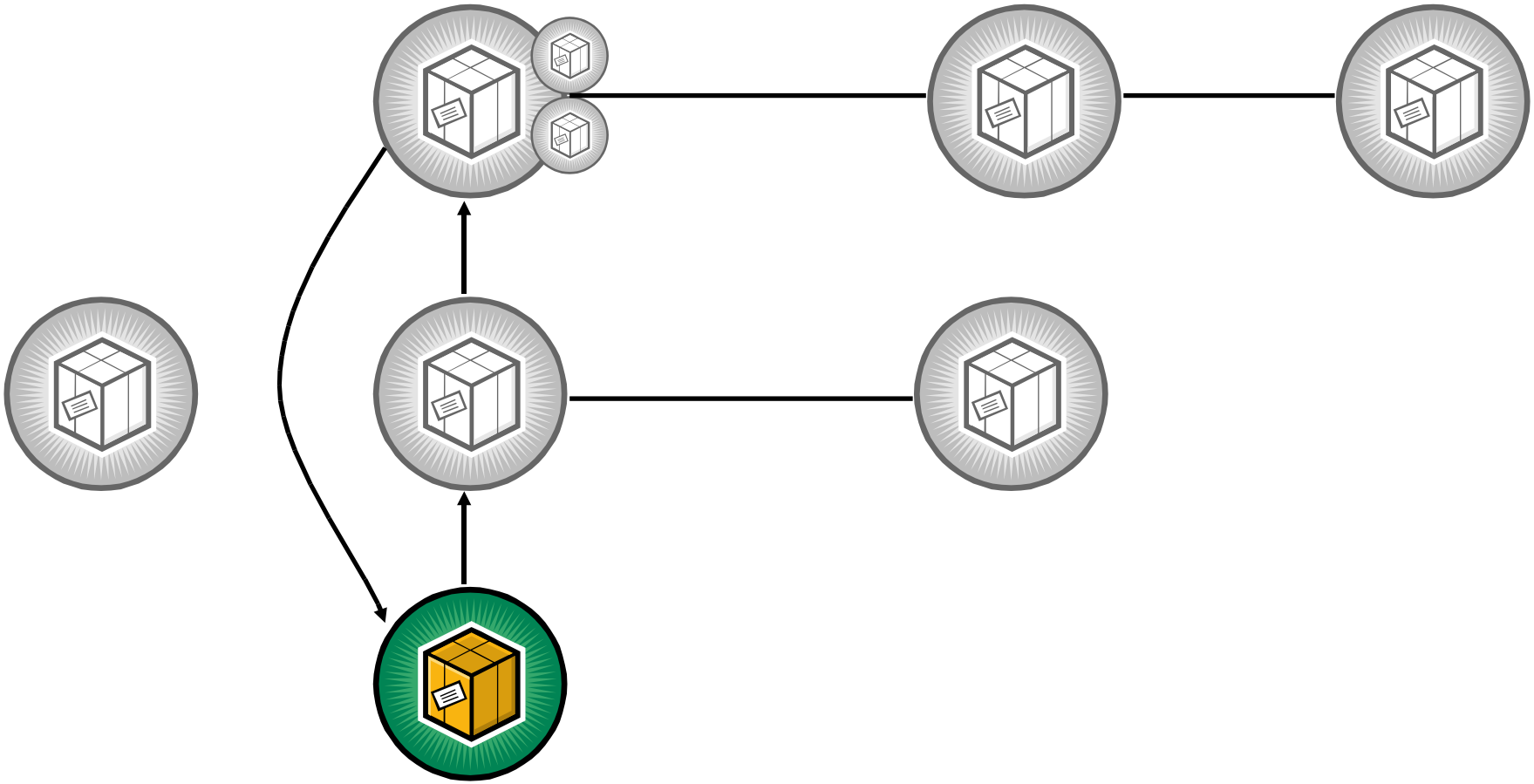
basics – Start des Teams



basics – Start des Teams

- StarteTeam
  - erzeugt und initialisiert 11 Spieler
  - meldet Mannschaft am Soccer Server an
  - danach autonomes Handeln jedes einzelnen Spielers
- PlayerControl
  - initiiert alle benötigten Objekte des Spielers
- Tools
  - enthält verschiedene Methoden, die global genutzt werden
  - z.B. Berechnung der Entfernung zwischen zwei Punkten

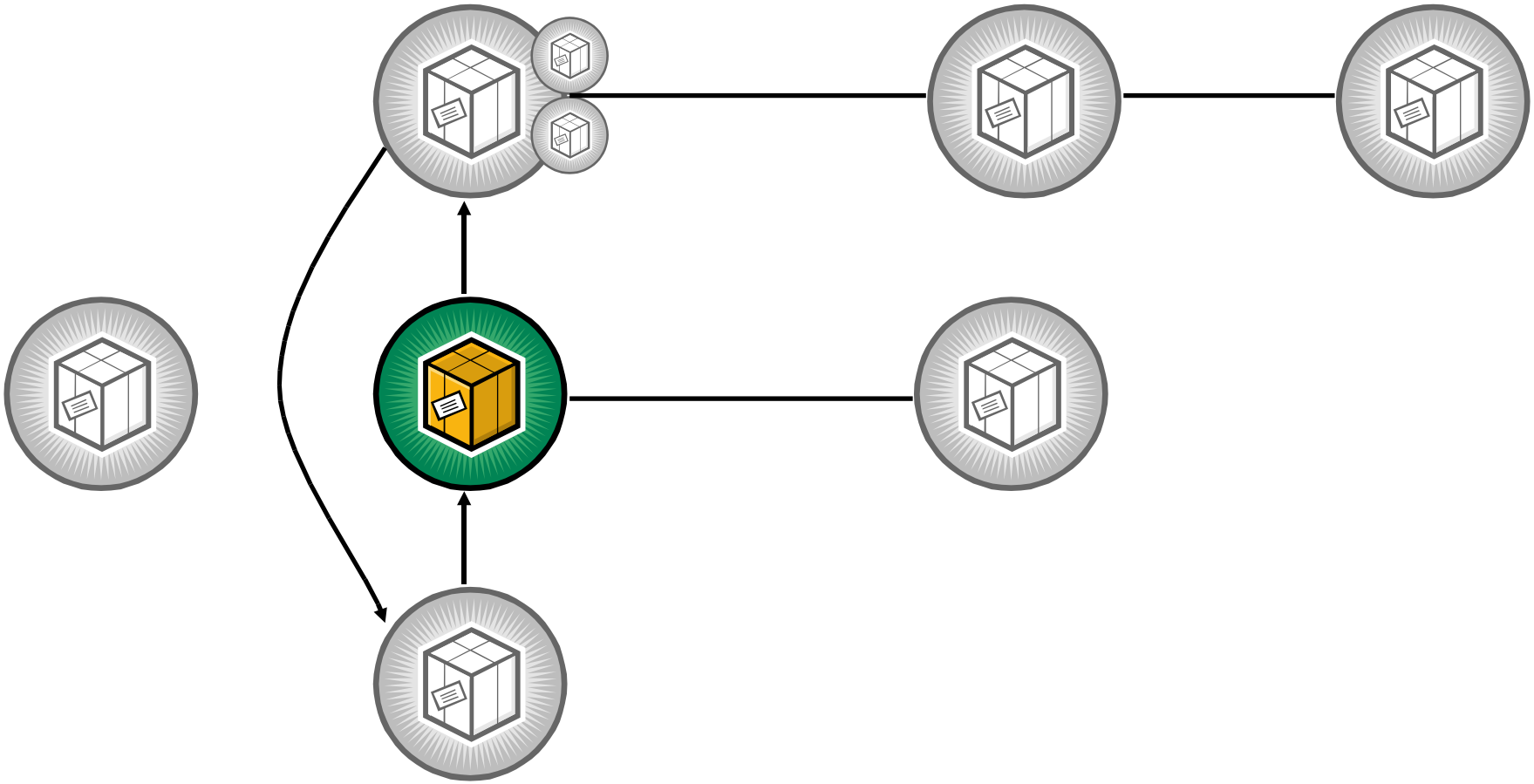
communication – Aufbau der Kommunikation mit dem Server



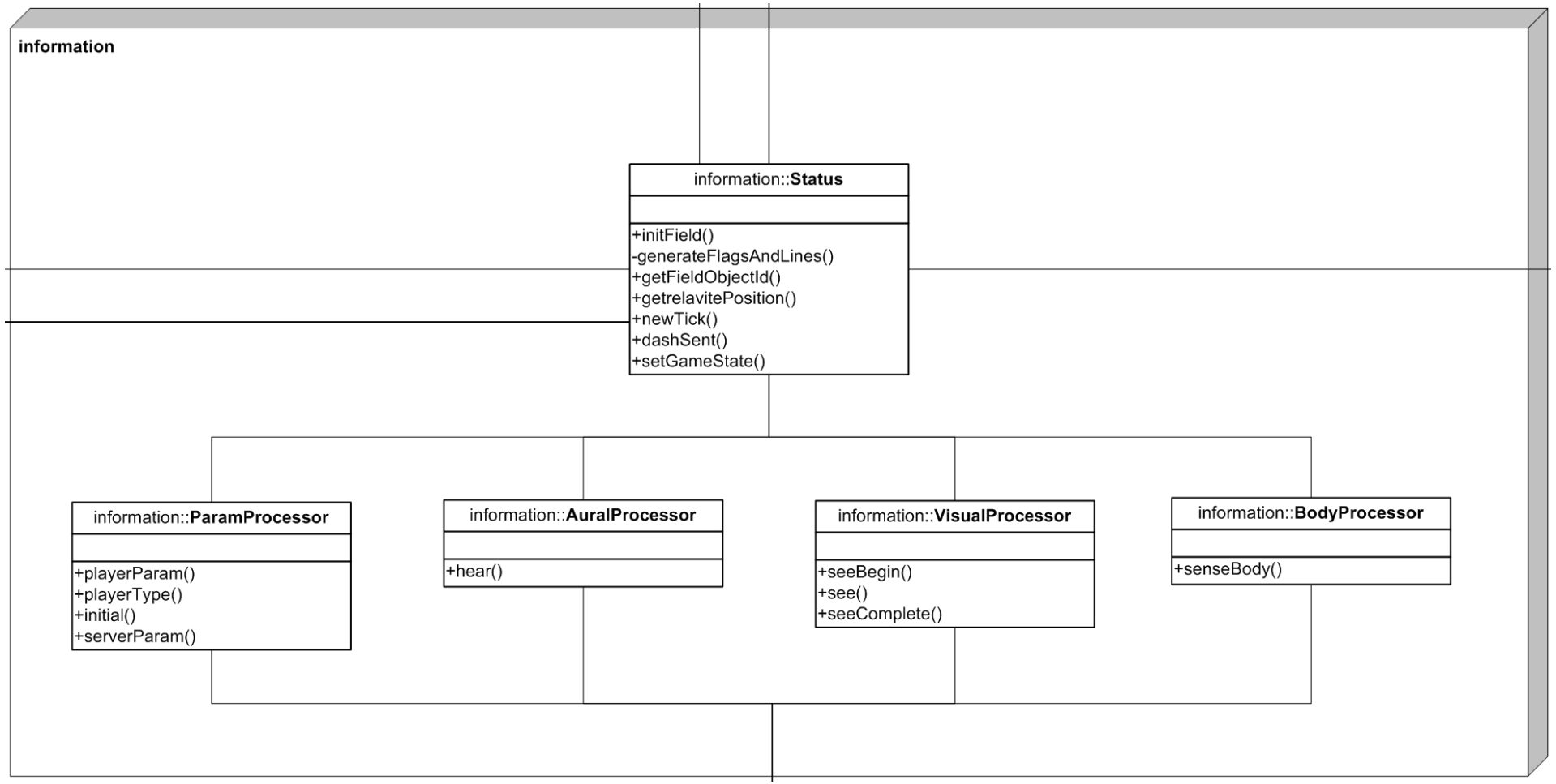
communication – Aufbau der Kommunikation mit dem Server

- Communicator
  - hält die Kommunikation zwischen Client (Spieler) und Server aufrecht
  - sendet und empfängt Daten-Strings
- Sender
  - nimmt Sendedaten entgegen und leitet sie zum Commmunicator
- Receiver
  - „horcht", ob Communicator Daten-Strings empfängt
  - leitet diese sofort an den MessageParser weiter
- MessageParser
  - syntaktische Verarbeitung der Daten-Strings
  - Informationen als String -> Informationen als Zahlenwerte (float …)

information – Verarbeitung der Informationen



information



information - AuralProcessor

- Nachrichten vom Schiedsrichter
  - Z.B. KICK_OFF_L, GOAL_R, OFFSIDE_R, HALF_TIME, TIME_UP

- Nachrichten von anderen Spielern

information - BodyProcessor

- Erhält jeden Tick Daten
- Spielerrelevante Eigenschaften, z.B.
  - Stamina
  - Kopfdrehung
  - Geschwindigkeit
  - Sehqualität
  - Blickwinkel

information - ParamProcessor

- Grundlegende Variablen werden übermittelt, z.B.
  - Breite des Tors
  - Eigene Rückennummer
  - Seite des Spielers

information - VisualProcessor

- Sammelt alle ‚gesehenen' Daten des Spielers:
  - Flaggen
  - Linien
  - Andere Spieler
  - Ball
- Jedes Objekt wird durch **Winkel** und **Entfernung** festgelegt
- Daten sind verrauscht

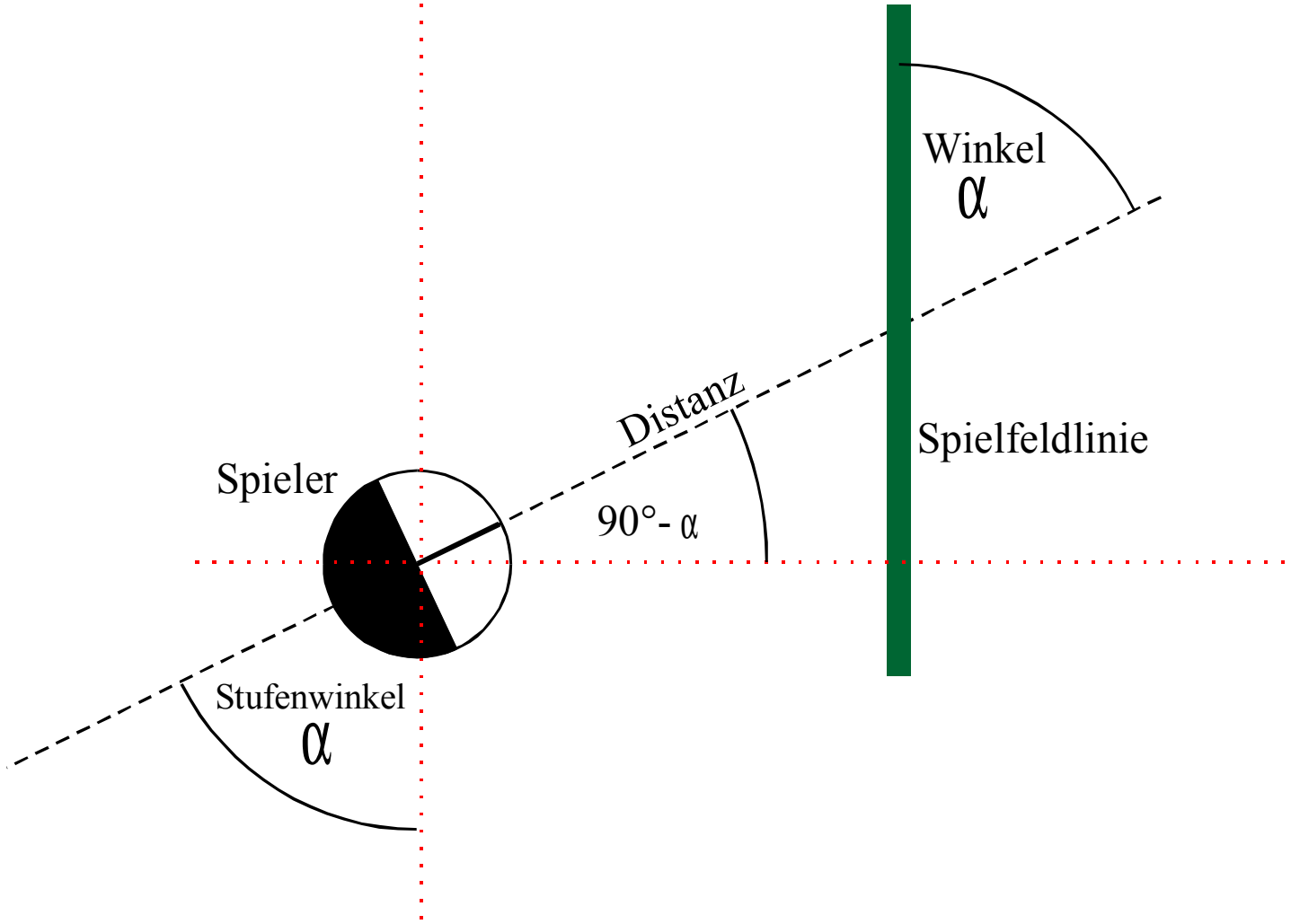
information - VisualProcessor

- Lokalisierung: Umwandeln der relativen Positionsangaben in absolute
  - Bestimmung der Drehung
  - Bestimmung der x-, y-Koordinaten

information - VisualProcessor

- Bestimmung der Drehung:
  - Voraussetzung: Zu jedem Zeitpunkt des Spiels ist eine Linie sichtbar
  - Übergabe einer Linie:
    - Entfernung zum Schnittpunkt mit der Blickrichtungshalbgerade
    - Winkel zwischen Blickrichtungshalbgerade und Linie

information - VisualProcessor

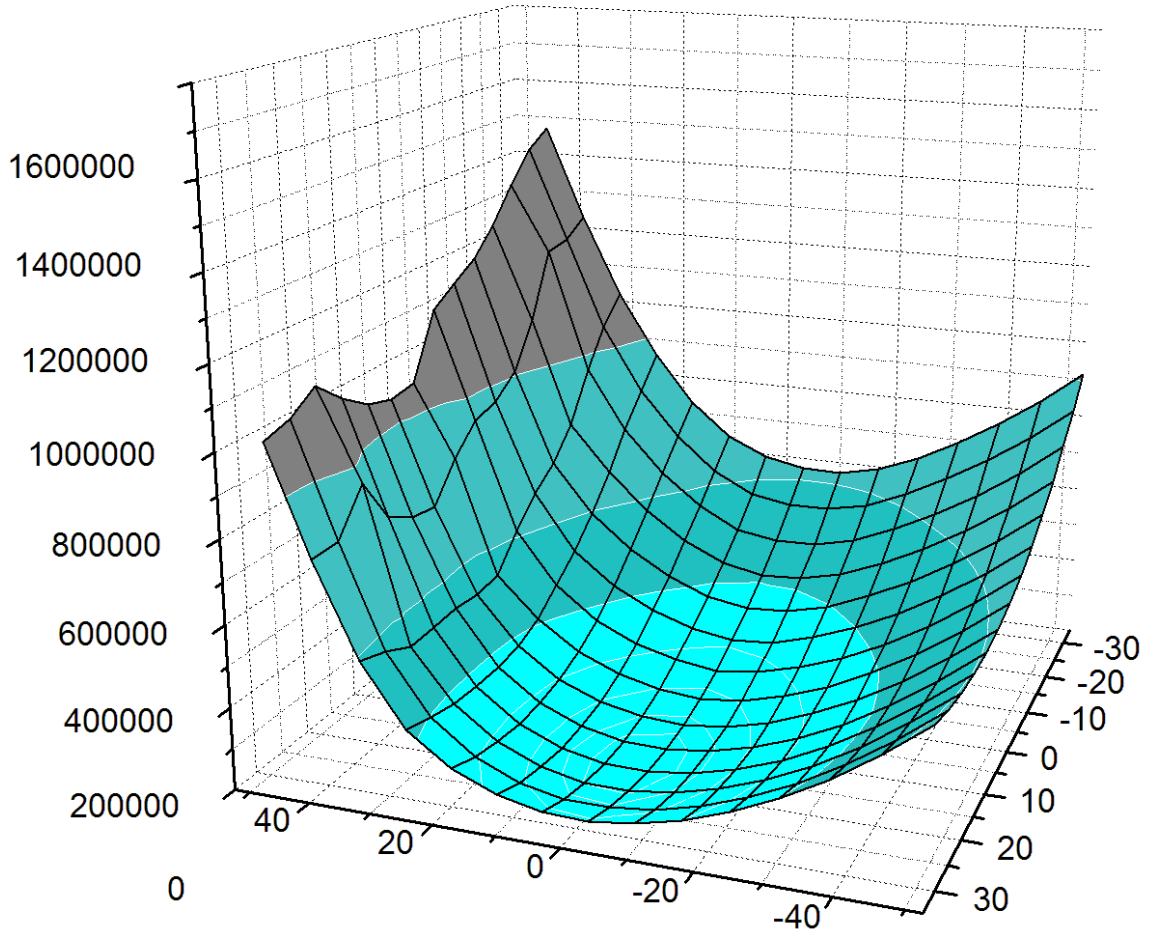


information - VisualProcessor

- Bestimmung der Koordinaten:
  - Um den Fehler durch Verrauschen zu minimieren, bestimme eigene Position anhand *aller* gesehener Flaggen
  - Dazu:
    - Schlage Position vor
    - Bestimme Entfernung und Winkel, die die Flaggen dort hätten
    - Differenz zu den gemessenen Daten ergibt Fehlerwert

information - VisualProcessor

Fehlergebirge bei festen Eingabedaten:



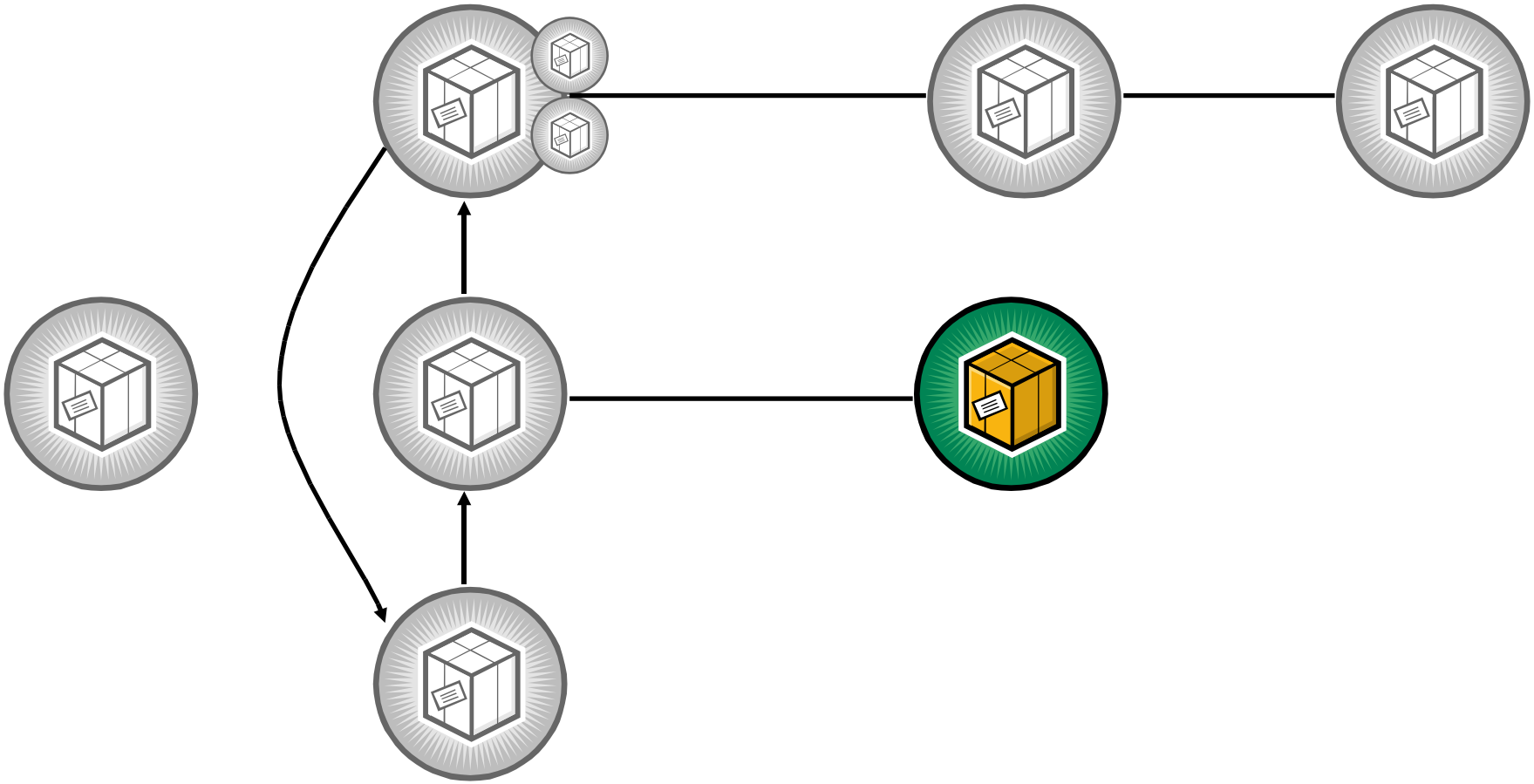
information - VisualProcessor

- Auftretendes ‚Fehlergebirge' ist ‚glatt':
  - Es gibt keine lokalen Minima außer dem globalen Minimum
- Bergsteigeralgorithmus konvergiert gegen dieses globale Minimum
- Absolute Positionen anderer Objekte können nun direkt bestimmt werden

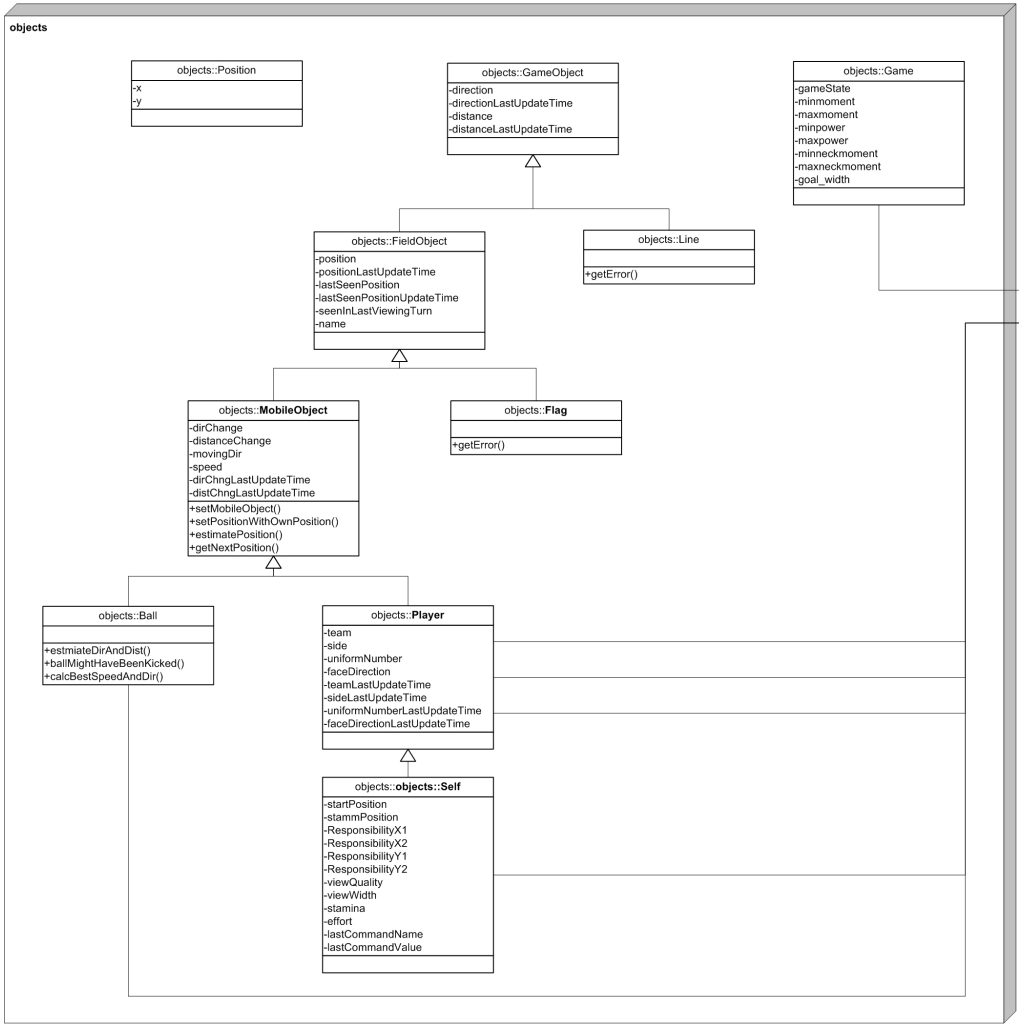
information - Status

- Zentrale Schnittstelle für alle Informationen, die der Spieler besitzt
- Empfängt Daten von den Processoren, und speichert sie in adäquaten Strukturen

objects – Bereitstellung der Daten



objects


objects
objects::Position
-x
-y
objects::GameObject
-direction
-directionLastUpdateTime
-distance
-distanceLastUpdateTime
objects::Game
-gameState
-minmoment
-maxmoment
-minpower
-maxpower
-minneckmoment
-maxneckmoment
-goal_width
objects::FieldObject
-position
-positionLastUpdateTime
-lastSeenPosition
-lastSeenPositionUpdateTime
-seenInLastViewingTurn
-name
objects::Line
+getError()
objects::MobileObject
-dirChange
-distanceChange
-movingDir
-speed
-dirChngLastUpdateTime
-distChngLastUpdateTime
+setMobileObject()
+setPositionWithOwnPosition()
+estimatePosition()
+getNextPosition()
objects::Flag
+getError()
objects::Ball
+estmiateDirAndDist()
+ballMightHaveBeenKicked()
+calcBestSpeedAndDir()
objects::Player
-team
-side
-uniformNumber
-faceDirection
-teamLastUpdateTime
-sideLastUpdateTime
-uniformNumberLastUpdateTime
-faceDirectionLastUpdateTime
objects::objects::Self
-startPosition
-stammPosition
-ResponsibilityX1
-ResponsibilityX2
-ResponsibilityY1
-ResponsibilityY2
-viewQuality
-viewWidth
-stamina
-effort
-lastCommandName
-lastCommandValue

objects

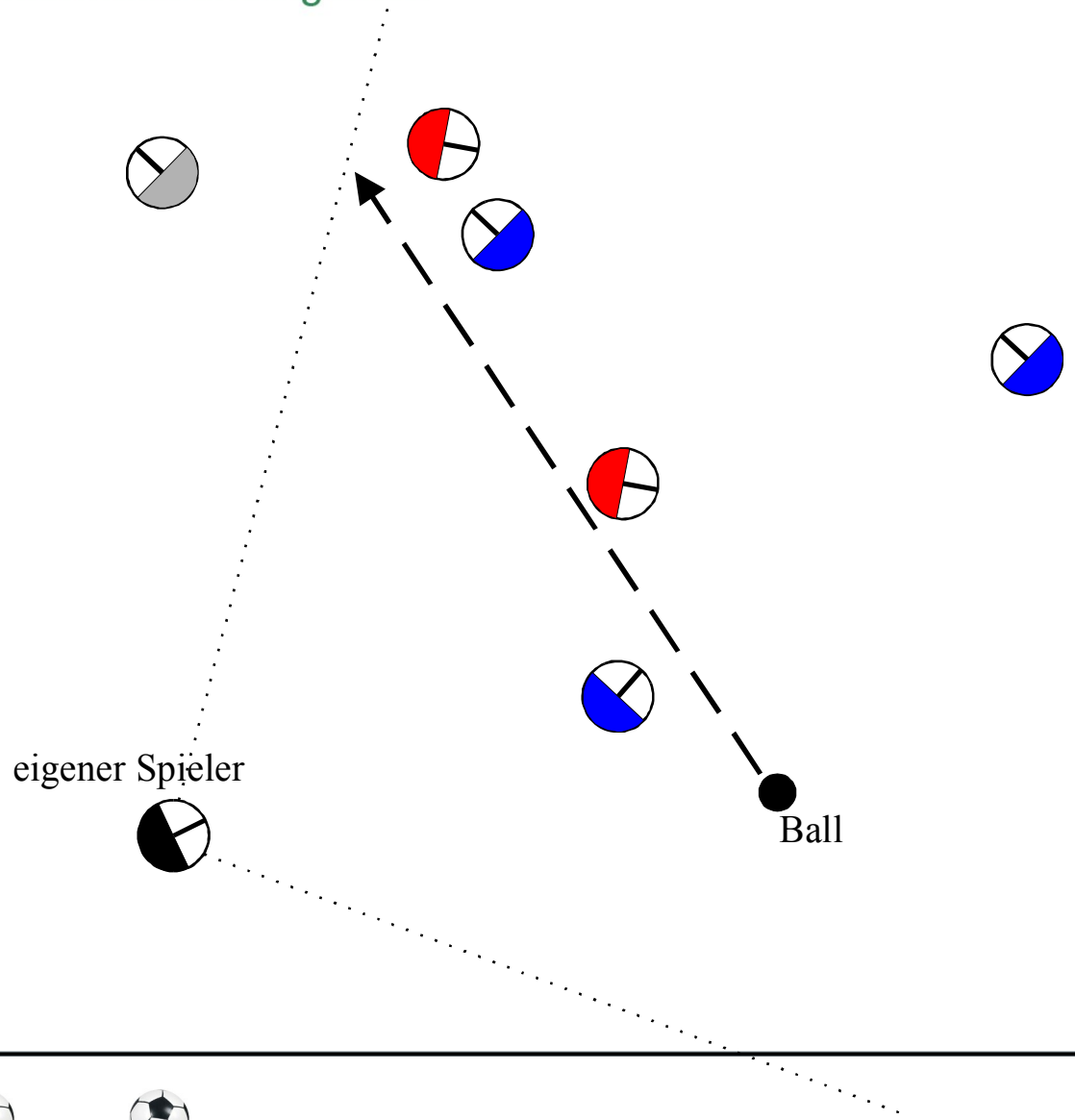
- Enthält die Klassen:
  - GameObject
  - FieldObject
  - Line
  - Flag
  - MobileObject
  - Ball
  - Player
  - Self
  - Game

- Objekte dienen der geordneten Speicherung der durch die Prozessoren gewonnenen Daten
- Einige Objekte enthalten weiterverarbeitende Methoden:
  - Ballbesitz
  - Bestimmung der Bewegungsrichtung und Geschwindigkeit

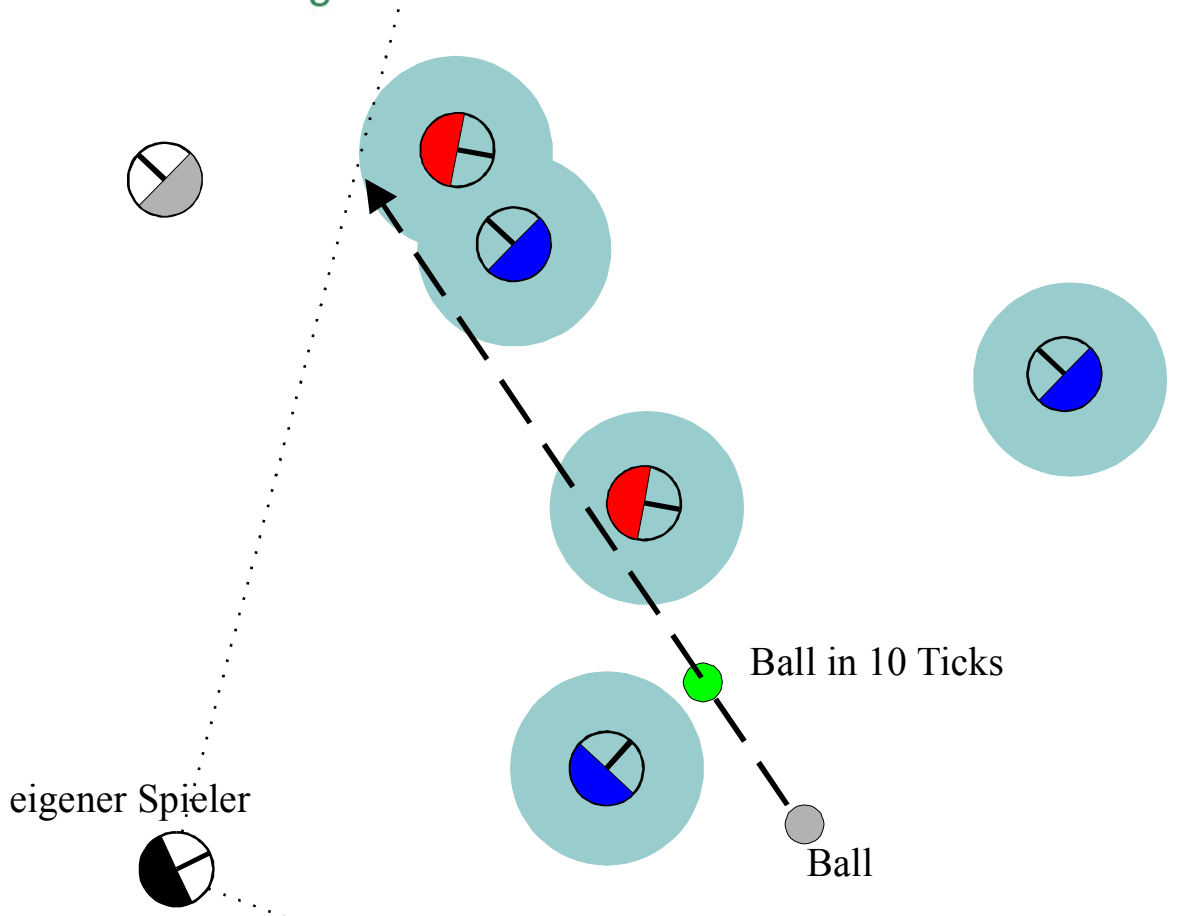
objects – Der Ball

- Ballbesitz:
  - In zukünftigen Positionen des Balles wird nahster Spieler bestimmt
  - Abhängig von Entfernung werden Teams Punkte gutgeschrieben
  - Punkteverhältnis ergibt Ballbesitzverhältnis
  - Unsicherheitsfaktor

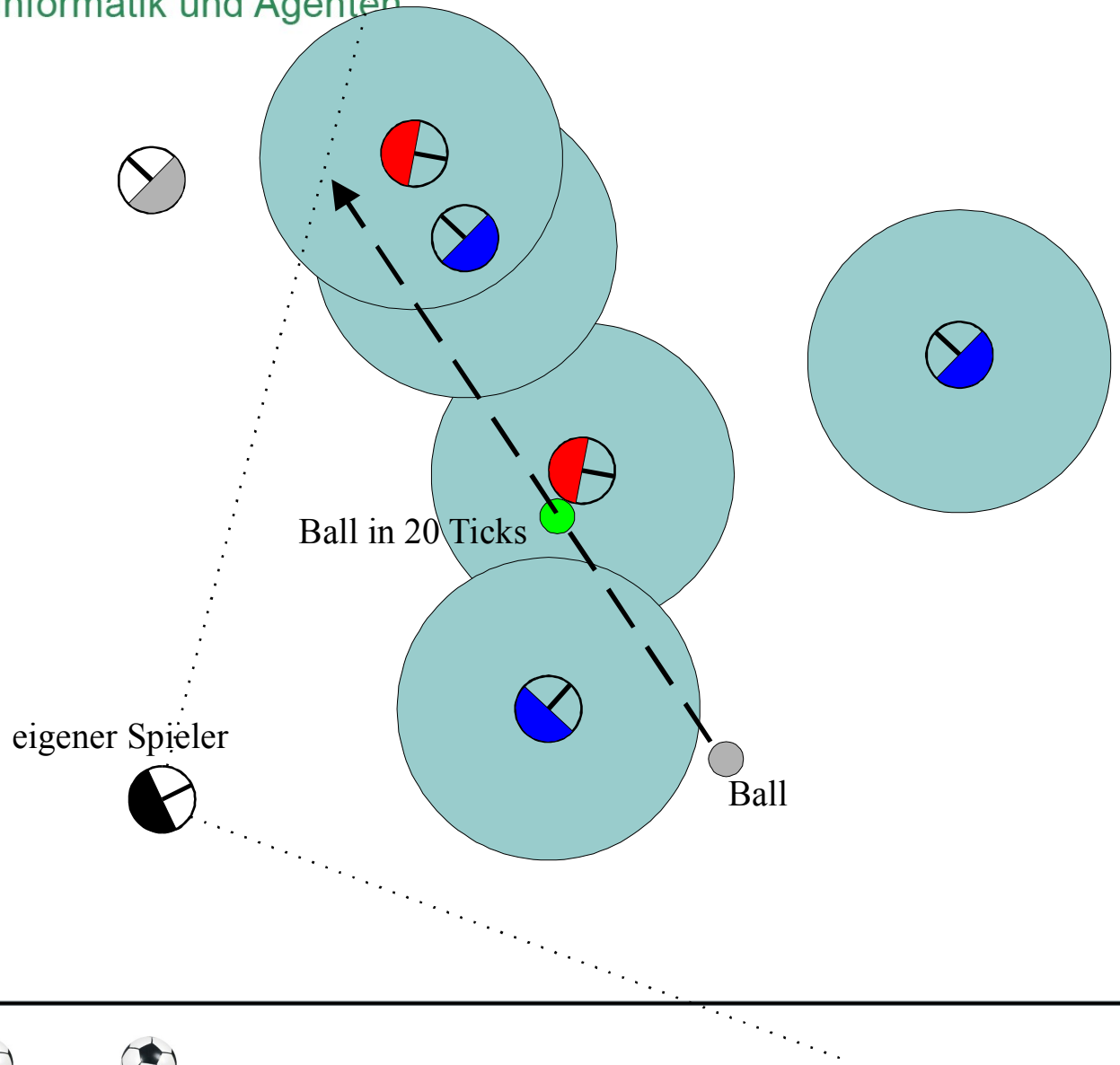
objects – Der Ball



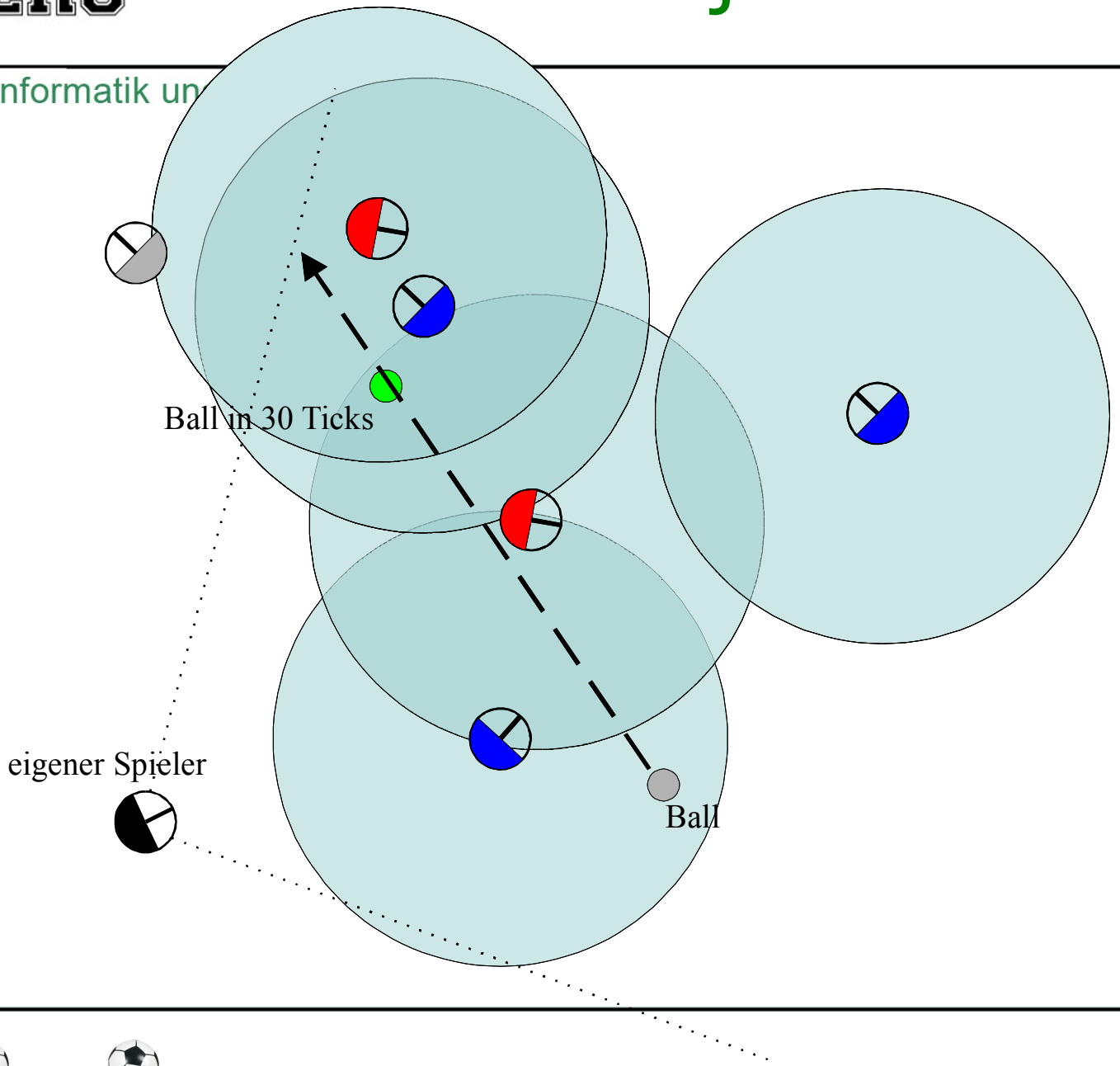
objects – Der Ball



objects – Der Ball



objects – Der Ball



objects – Der Ball

- Ballgeschwindigkeit und –bewegungsrichtung
  - Da erfasste Daten verrauscht sind, ist Ballposition ungenau
    - Abweichungen von einigen Metern sind möglich
  - Bestimmung der Geschwindigkeit und Richtung über zwei aufeinander folgende Positionen schlecht
  - Daher: Alternative mit Überprüfung, ob Ball gekickt

objects – Der Ball

- Ballgeschwindigkeit und –bewegungsrichtung
  - Dazu Berücksichtigung von mehreren Faktoren:
    - Steht ein **anderer Spieler** nah am Ball?
    - Steht der **eigene Spieler** nah am Ball und hat ein KICK-Befehl gesendet?
    - Hat der Ball den **Pfosten** berührt?
    - Tritt zusätzlich noch eine große Geschwindigkeits- oder Richtungsänderung auf, wird Ball als gekickt erkannt.

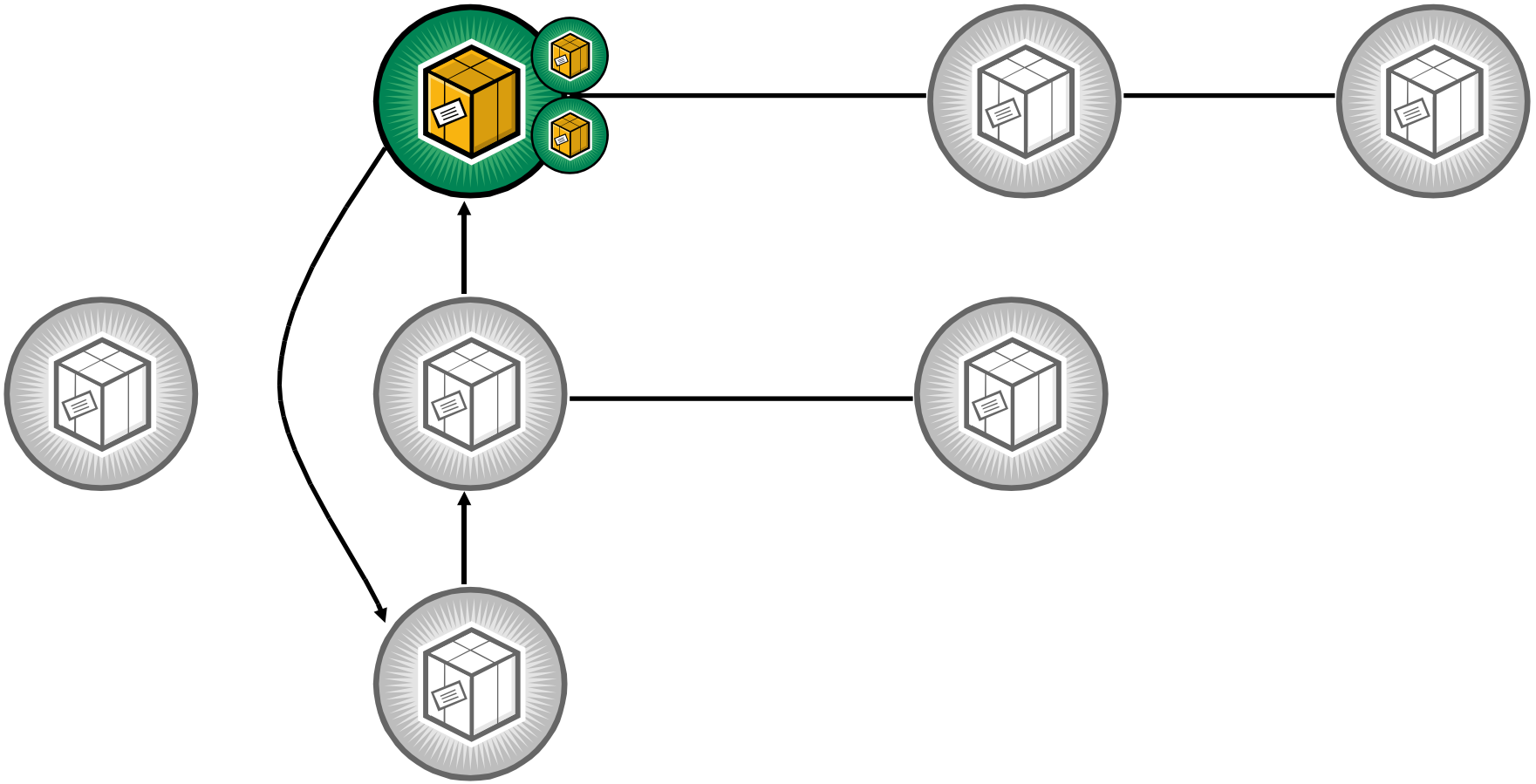
objects – Der Ball

- Ballgeschwindigkeit und –bewegungsrichtung
  - Falls Ball als *gekickt* erkannt wurde:
    - Speichere Position des Balles als **Kickposition**
    - Soll zu einem späteren Zeitpunkt Bewegungsrichtung bestimmt werden, so durch Winkel der Ballposition mit Kickposition
    - Ballgeschwindigkeit: Mittele alle errechneten Geschwindigkeiten ab Kickzeitpunkt (Beschleunigung berücksichtigt)

objects – andere Spieler

- Speicherung der anderen Spieler
  - Fast nie alle Spieler im Blickfeld, oft keine Rückennummern erkennbar
  - Daher: Zwei Speicherstrukturen
    - *Temporäre* Liste von allen gesehenen Spielern ohne erkannte Rückennummer
    - *Feste* Liste mit allen Spielern
      - Aktualisiert Spieler, falls Rückennummer erkannt wird
      - Falls nur ungefähre Position benötigt wird, kann Position auch einige Zeit nach Verlassen des Blickfeldes geschätzt werden

Überblick über die Java -Pakete

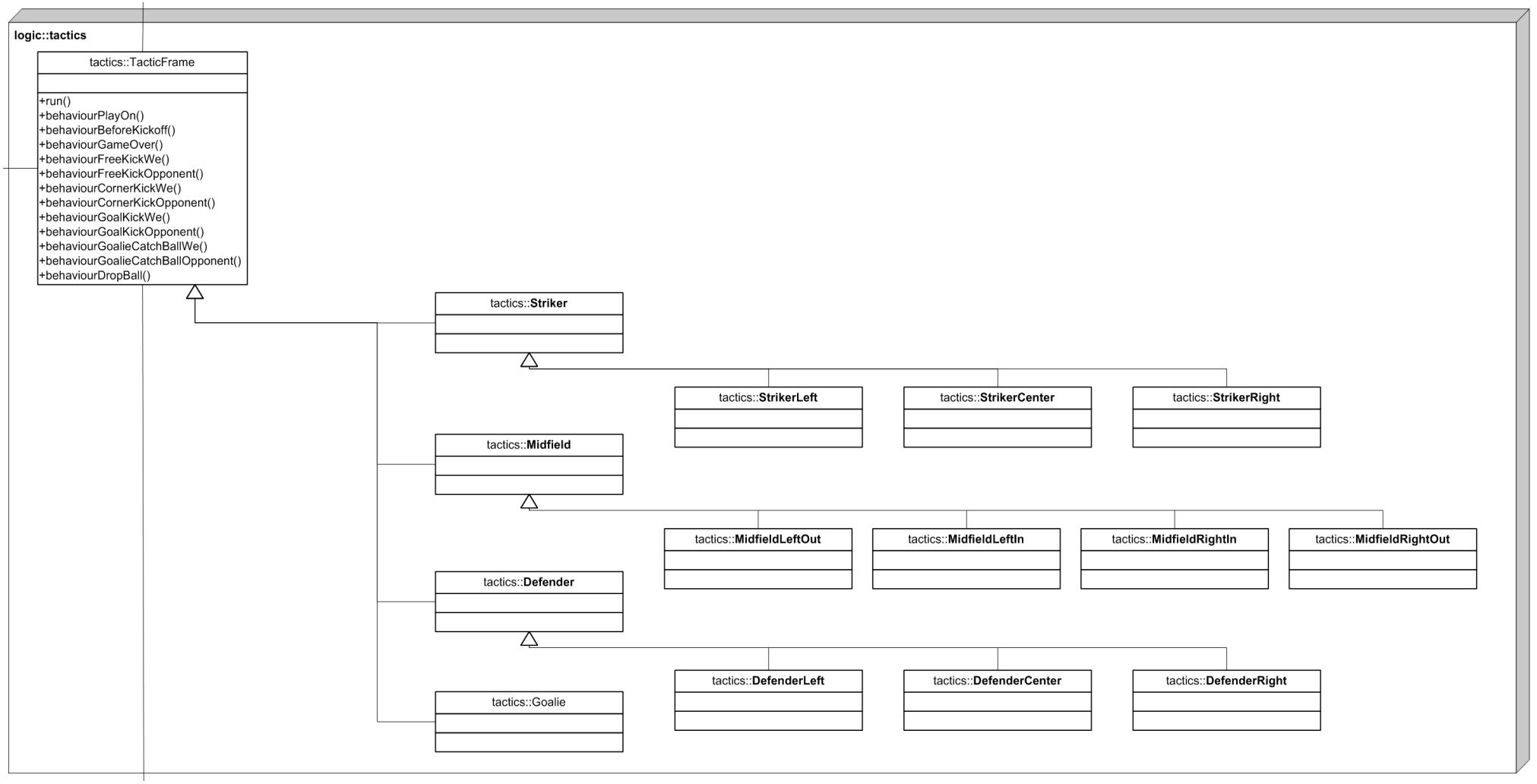


- Das Paket *logic* enthält folgende Unterpakete und Klassen:
  - logic.Strategy
  - logic.tactics.*
  - logic.Operation
  - logic.moves.*
  - logic.BasicFunctions

- Eigenständiger Thread auf höchster Entscheidungsebene:
  - Entscheidet abhängig vom Spielverlauf über die **Taktik der Spieler**
  - „**globale**" Entscheidungen
  - Erfordert „feste" Informationen

- Eigenständiger Thread:
  - Wird von der Informationsverarbeitung angestoßen
  - Entscheidet jede Runde über den konkreten **Spielzug eines Spielers**
  - Verschiedene Positionen besitzen unterschiedliche Taktiken
  - **Grundverhalten** sind bei allen Spielern gleich.
  - Realisierung durch **Vererbung** in Java.

Klassendiagramm logic.tactics.*

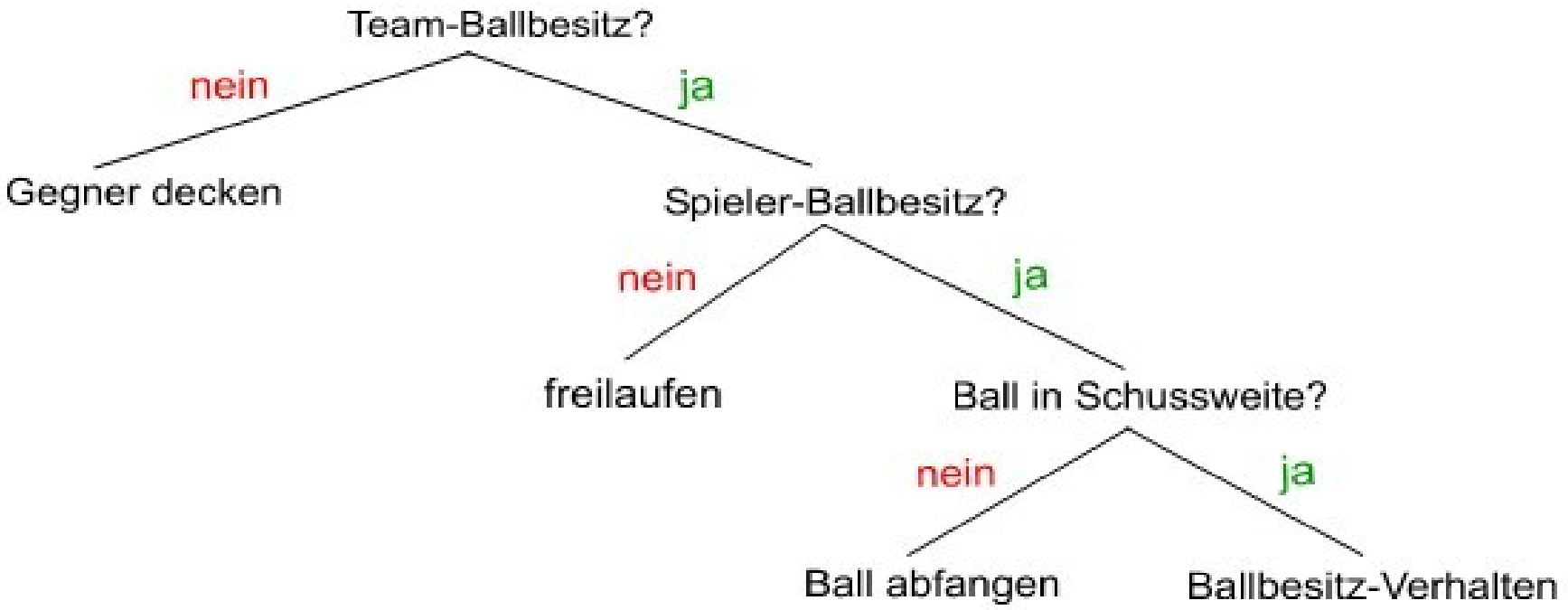


Entscheidungsfindung (1)

- Entscheidung über das Verhalten des Spielers abhängig von 3 Fällen:
  - Spieler im Ballbesitz
  - Team im Ballbesitz
  - Gegner im Ballbesitz

- *Feste Spielzüge* bei Team/Gegner im Ballbesitz
- Interessantester Fall: Eigener Ballbesitz

Entscheidungsfindung (2)



Danger-System (1)

- Drei möglichen Aktionen bei eigenem Ballbesitz:
  - Dribbeln
  - Passen
  - Torschuss

- Entscheidungsfindung durch **Danger-System**

Danger-System (2)

- Bestimmung des Danger-Wertes:
  1. Betrachtung **zukünftiger Ballpositionen**
  2. Bestimmung eines **Grundwertes**
  3. Multiplikation mit einem Positions-Faktor („**positionRating**“)
  4. **Manuelle Anpassung** des Wertes für spezielle Situationen

Danger-System (3)

- Danger-Wert beschreibt *Güte* einer Aktion.
- Bei eigenem Ballbesitz werden folgende Danger-Werte berechnet:
  - *Dribbling-Danger*
  - *Pass-Danger*
  - *GoalKick-Danger*
- Die Position mit dem jeweils geringsten Danger-Wert wird ausgewählt und übergeben.

Danger-System (4)

- Vergleich der Danger-Werte
  - → Entscheidungsfindung

- Vorteile dieses Systems:
  - Flexibel
  - Leicht anpassbar
  - Lösungen für alle Situationen

logic.Operation

- *Bindeglied* zwischen logic.tactics und logic.moves:
  - „Kapselt" Spielzüge
  - Ermöglicht **rundenübergreifende** Spielzüge
  - Kontrolliert **Nackensteuerung**

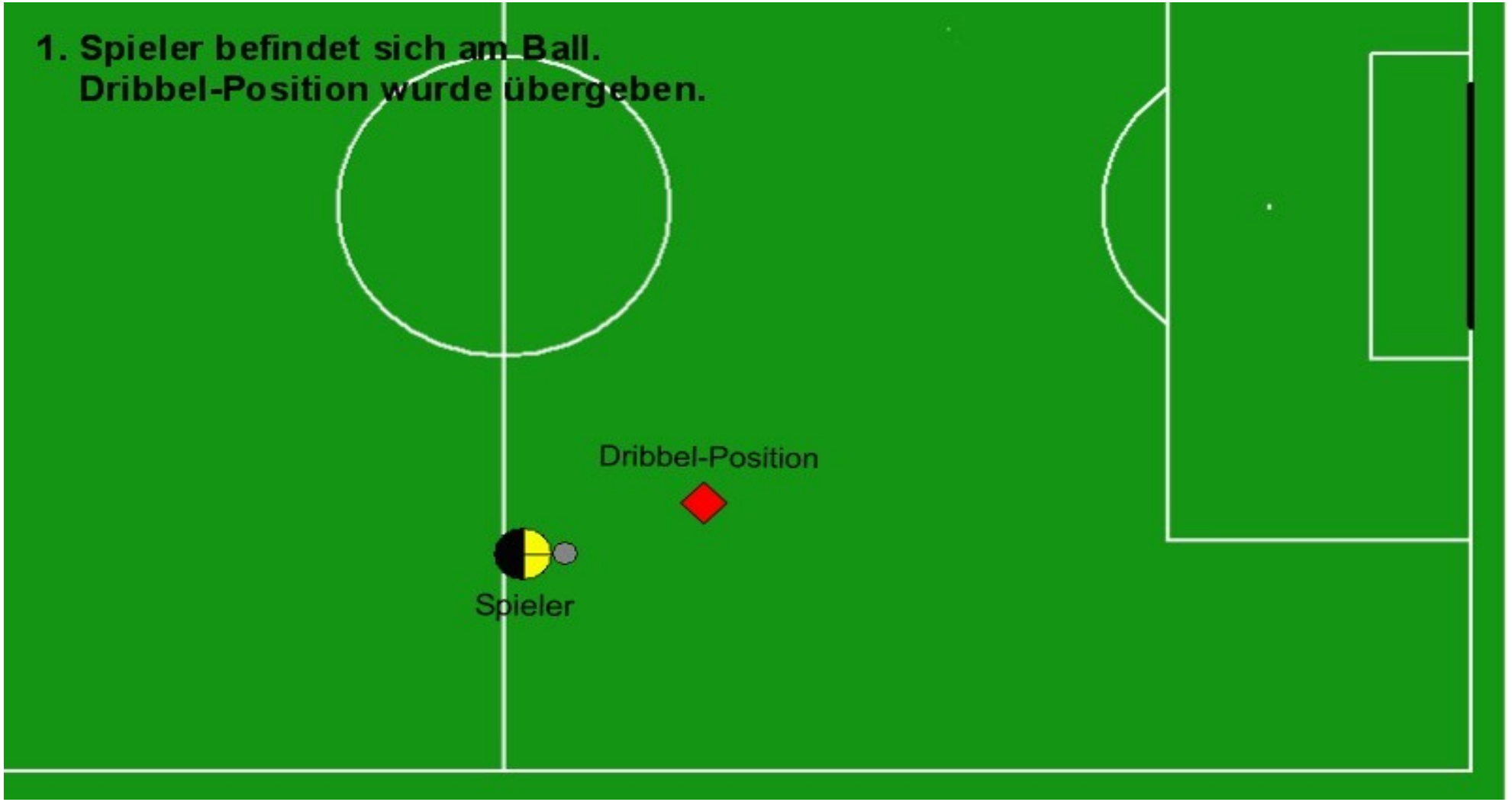
- Klassen setzen komplexere Bewegungsabläufe um:
  - Informationen von Status/Taktik
  - Berechnung **zusätzlicher Informationen**
  - Aufruf von **Basisfunktionen**

- Zu komplex für Basisfunktionen

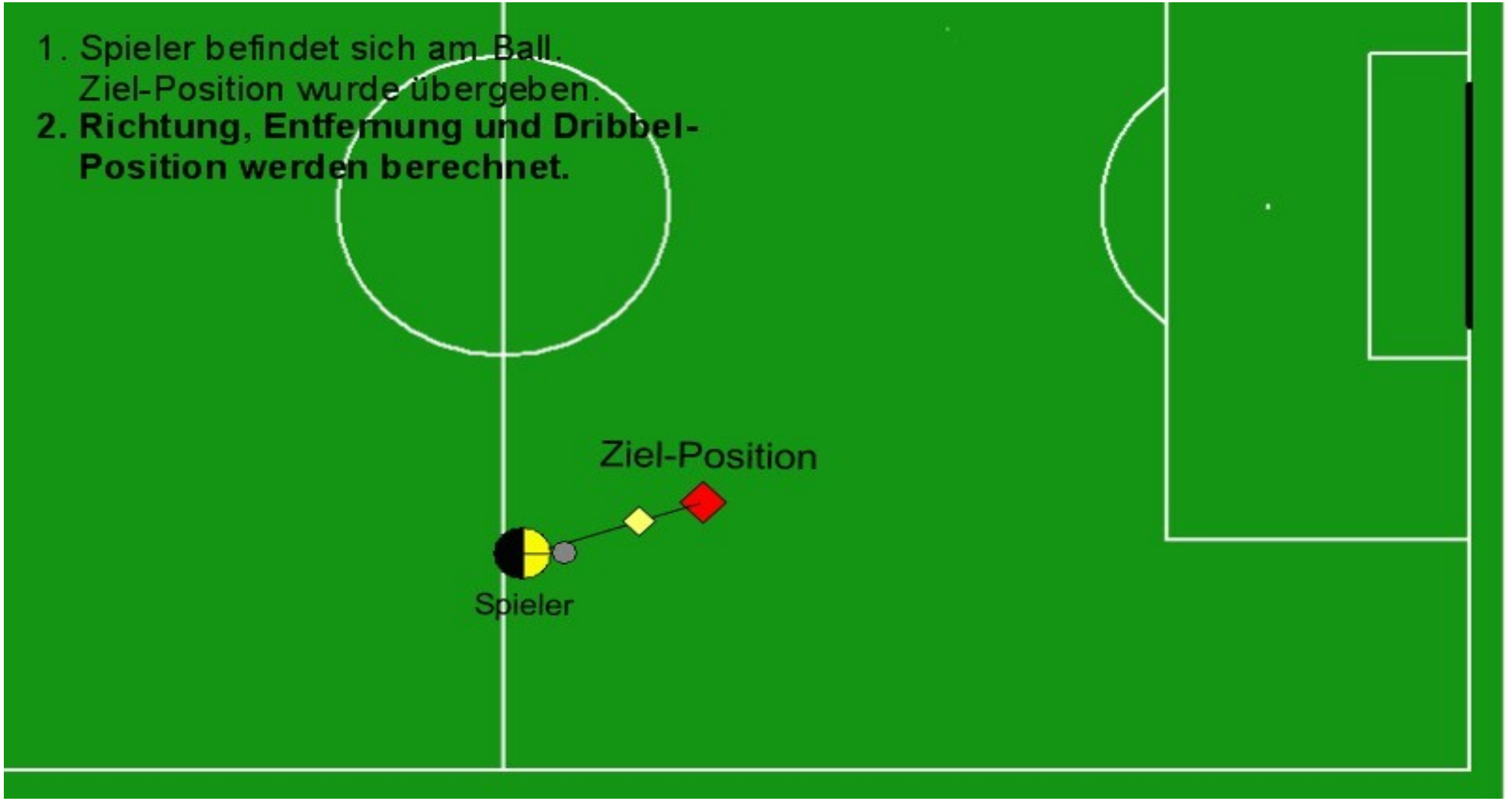
logic.Moves.*

- Folgende Spielzüge wurden implementiert:
  - Goalkick
  - Dribbling
  - Pass
  - GoTo
  - ReceivePass
  - InterceptBall
  - CoverOpponent
  - GoalieMoves
  - SearchBall
  - Shouts

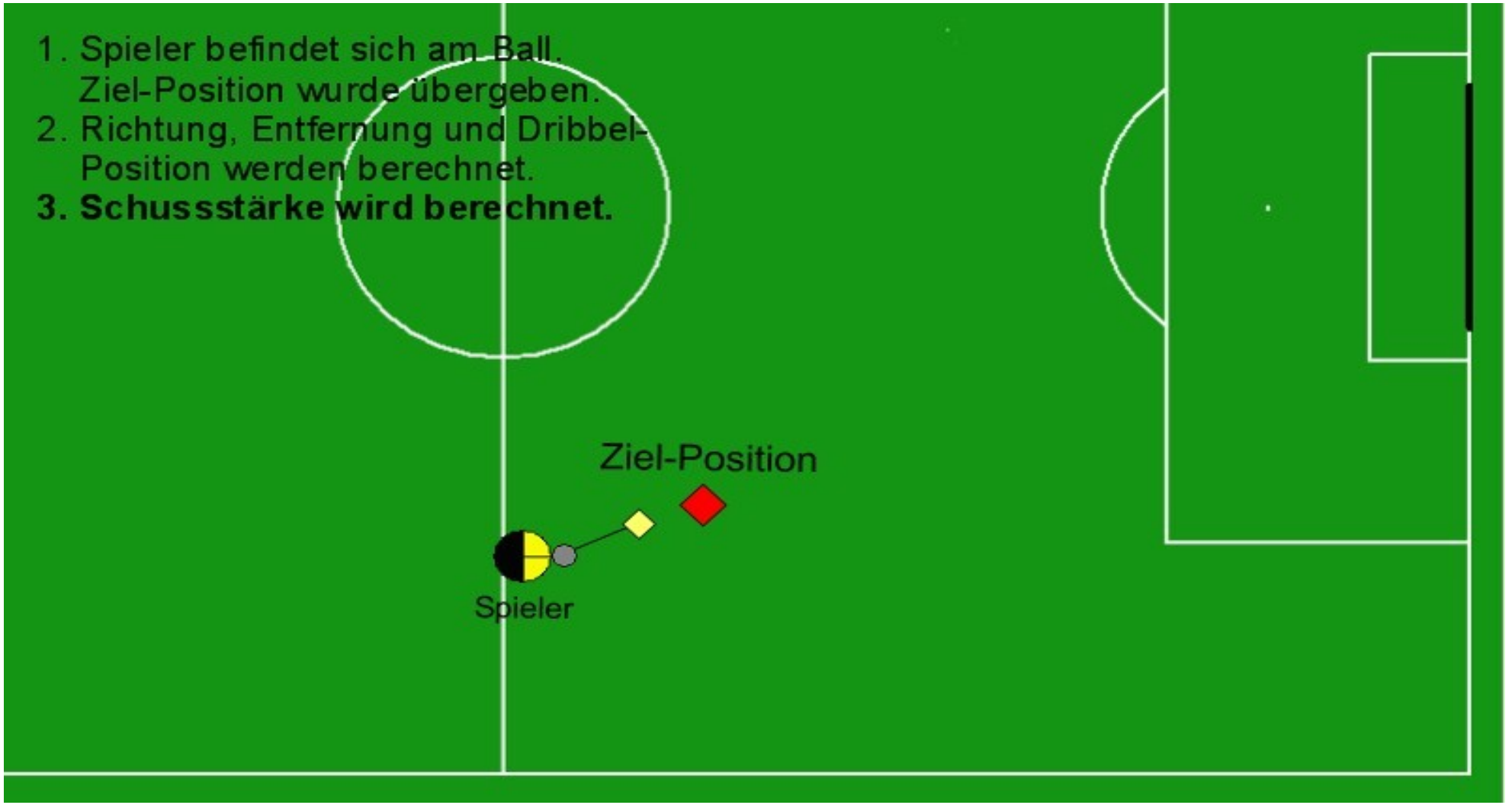
Beispiel: Dribbeln (1)



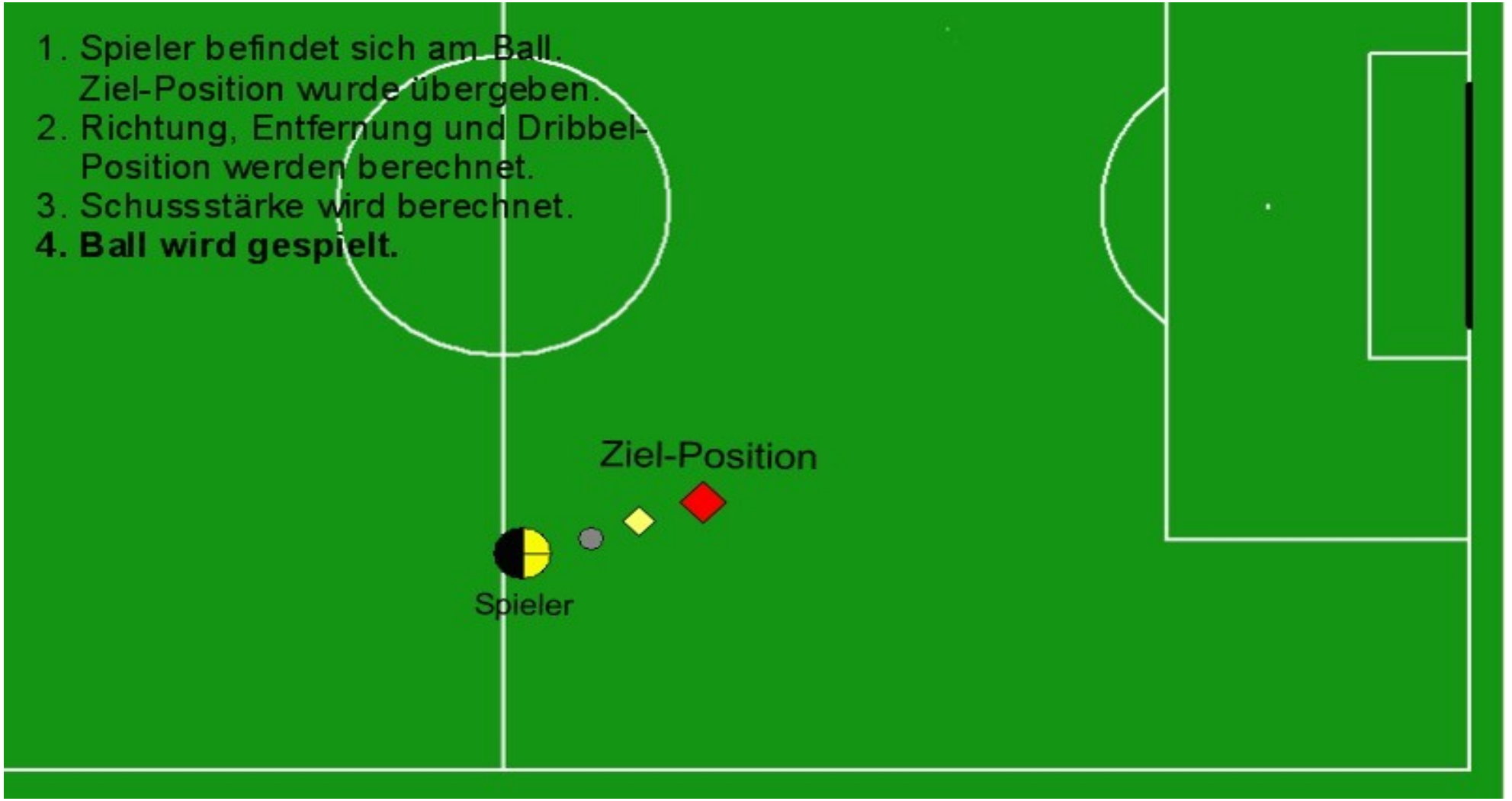
Beispiel: Dribbeln (2)



Beispiel: Dribbeln (3)



Beispiel: Dribbeln (4)



Beispiel: Dribbeln (5)

logic.BasicFunctions

- Kommunikation mit Server durch Grundbefehle
- Weitere Aufgaben:
  - Überprüft die **Gültigkeit** der übergebenen Argumente
  - Berechnet tatsächlichen Drehmoment bei **turn**
  - Korrigiert ggf. Werte im Status

Gliederung

- Einleitung
- Aufbau der Mannschaft
- **Nutzung künstlicher Intelligenz**
- Bewertung und Zusammenfassung
  - errorBackpropagation

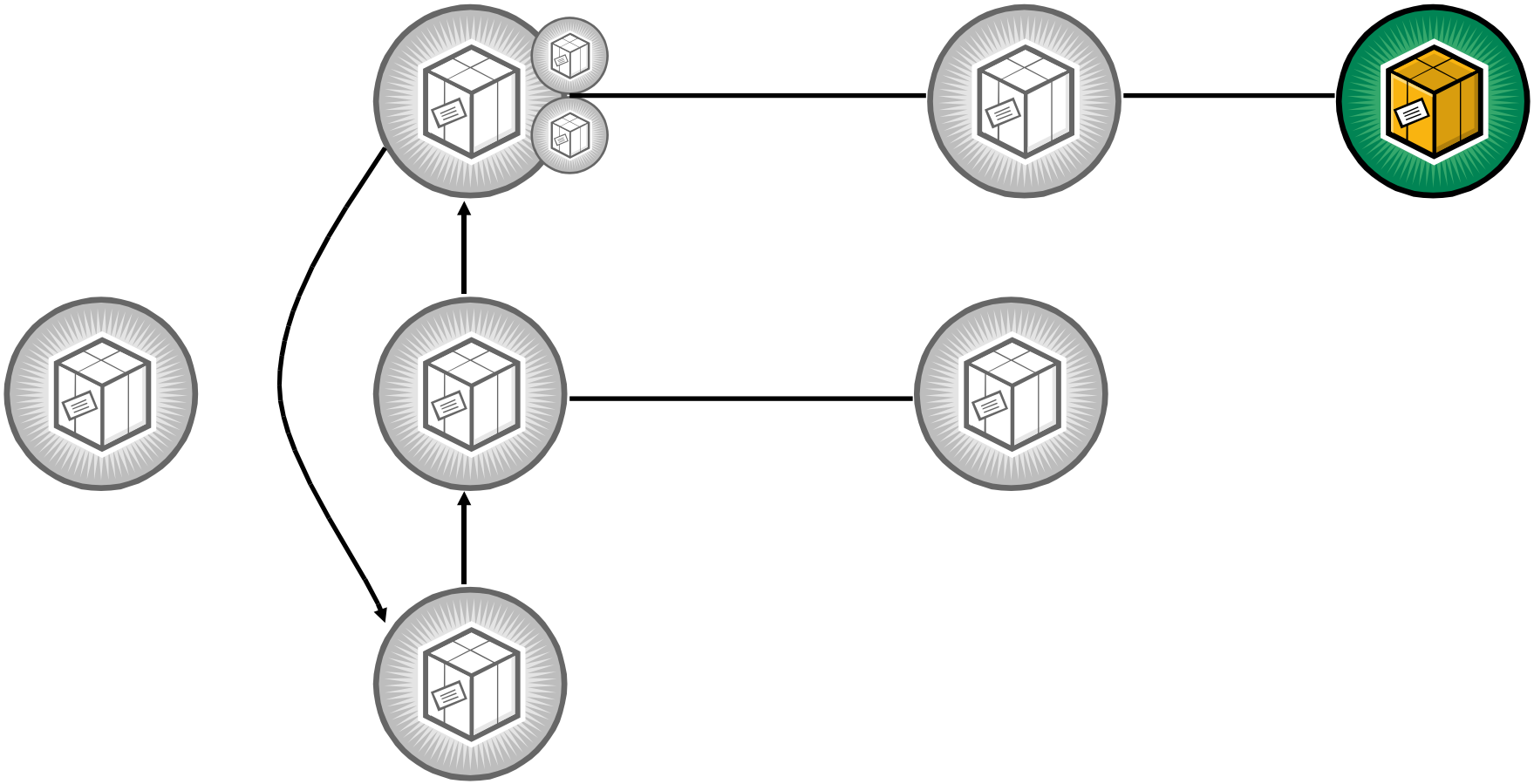
Nutzung künstlicher Intelligenz

- Verhalten der Spieler soll durch den **Einsatz von KI** verbessert werden
  - z.B. optimales Ballabfangen
- Vorraussetzungen:
  - Ein- und Ausgabe – **Parameter** festlegen
  - Daten sammeln
  - KI **Methode** auswählen

- Tools: MFOSS, Neurodimension, Joone → erfolglos !

➔ Eigene Implementierung eines KNN mit Hilfe anderer Ansätze

Überblick über die Java -Pakete

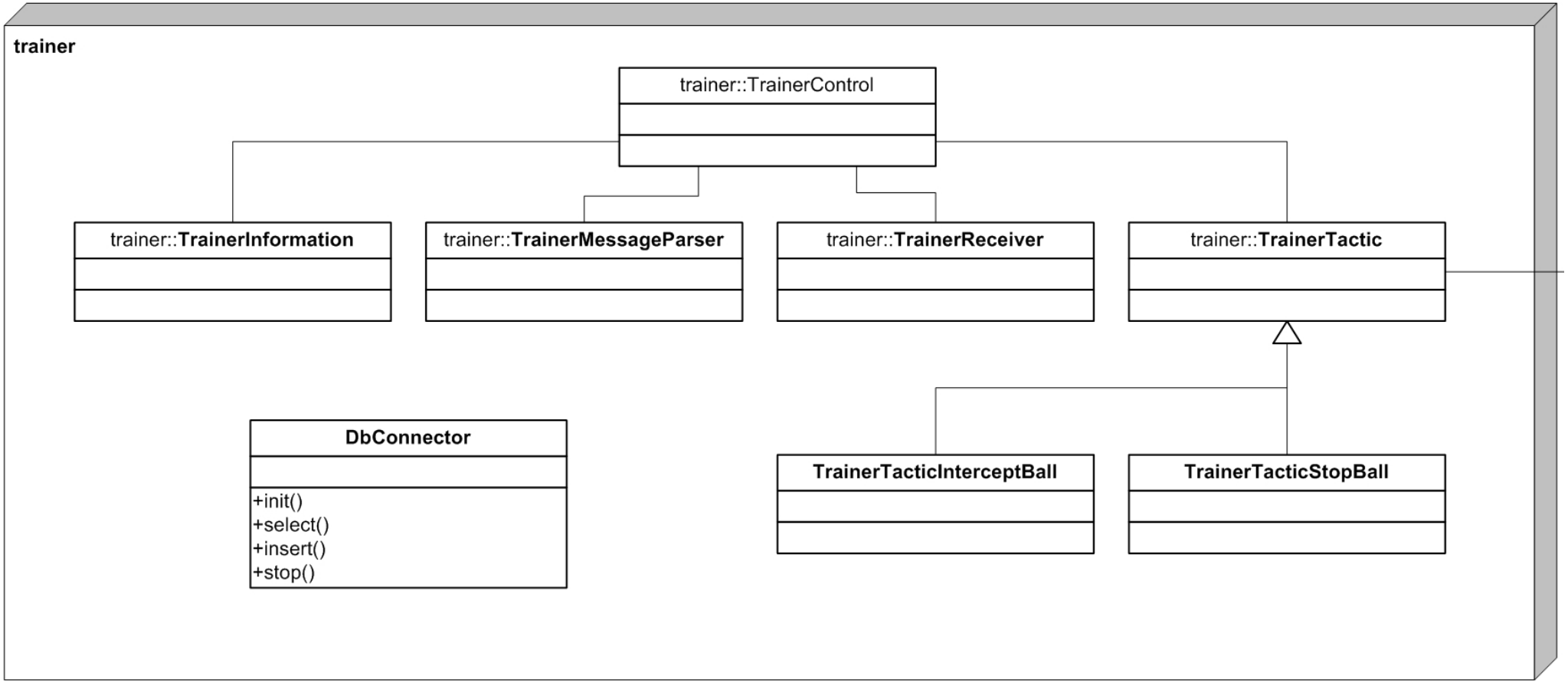


Der Trainer

- Schaffung von **Trainingssituationen** mit Hilfe des Trainers
  - Online-Coach vs. Offline-Coach
  - Anmeldung beim Server wie jeder Spieler
  - Kommunikation über eigenen Port
  - Trainer erhält **unverzerrte** Daten
  - Alle Objekt **sichtbar** und **beeinflussbar**:
    - Spielerpositionen verändern, Spielerbewegung vorgeben
    - Ballposition verändern, Richtung und Geschwindigkeit vorgeben

➔ Einfaches Erschaffung von Trainingssituationen

Der Trainer


trainer
trainer::TrainerControl
trainer::TrainerInformation
trainer::TrainerMessageParser
trainer::TrainerReceiver
trainer::TrainerTactic
DbConnector
+init()
+select()
+insert()
+stop()
TrainerTacticInterceptBall
TrainerTacticStopBall

Der Trainer

- Klassen zur Trainer-Benutzung
  - TrainerControl
    - Initialisierung
  - TrainerTactic
    - Methoden zur **Steuerung** des Ablaufs der Trainingssituation
  - TrainerInformation
    - Informationsbereitstellung der durch den Trainer gewonnen **Daten**
  - Außerdem eigener MessageParser und MessageReceiver, um Trennung zu Spielerkommunikation zu erreichen

Der Trainer

- Verwendung des Trainers
  - Beispiel: Ball an bestimmte Position legen und Richtung und Geschwindigkeit festlegen
  - Methode MoveBall schickt Nachricht an den Server der die Aktion durchführt
  - Ständige Informationen über aktuelle Ball- und Spielerpositionen können für weitere Berechnungen innerhalb der Trainingssituation verwendet werden (KI-Berechnungen)
  - Trainer hat eigenes **InformationsStatus-Objekt**
  - StatusObjekt des zu trainierenden Spielers wird übergeben
  - **Hilfsmethoden**: EyeMode, Recover, ChangePlayMode

Trainingsdaten sammeln

- Trennung von Sammeln der Daten und der späteren Berechnung
- „Zwischenspeicher" MySQL **Datenbank** auf zentralem Server
- Klasse **DbConnector** für den Zugriff aus die Datenbank
- Methoden zum Ein- und Ausgeben von Daten

Ansätze anderer Entwicklerteams

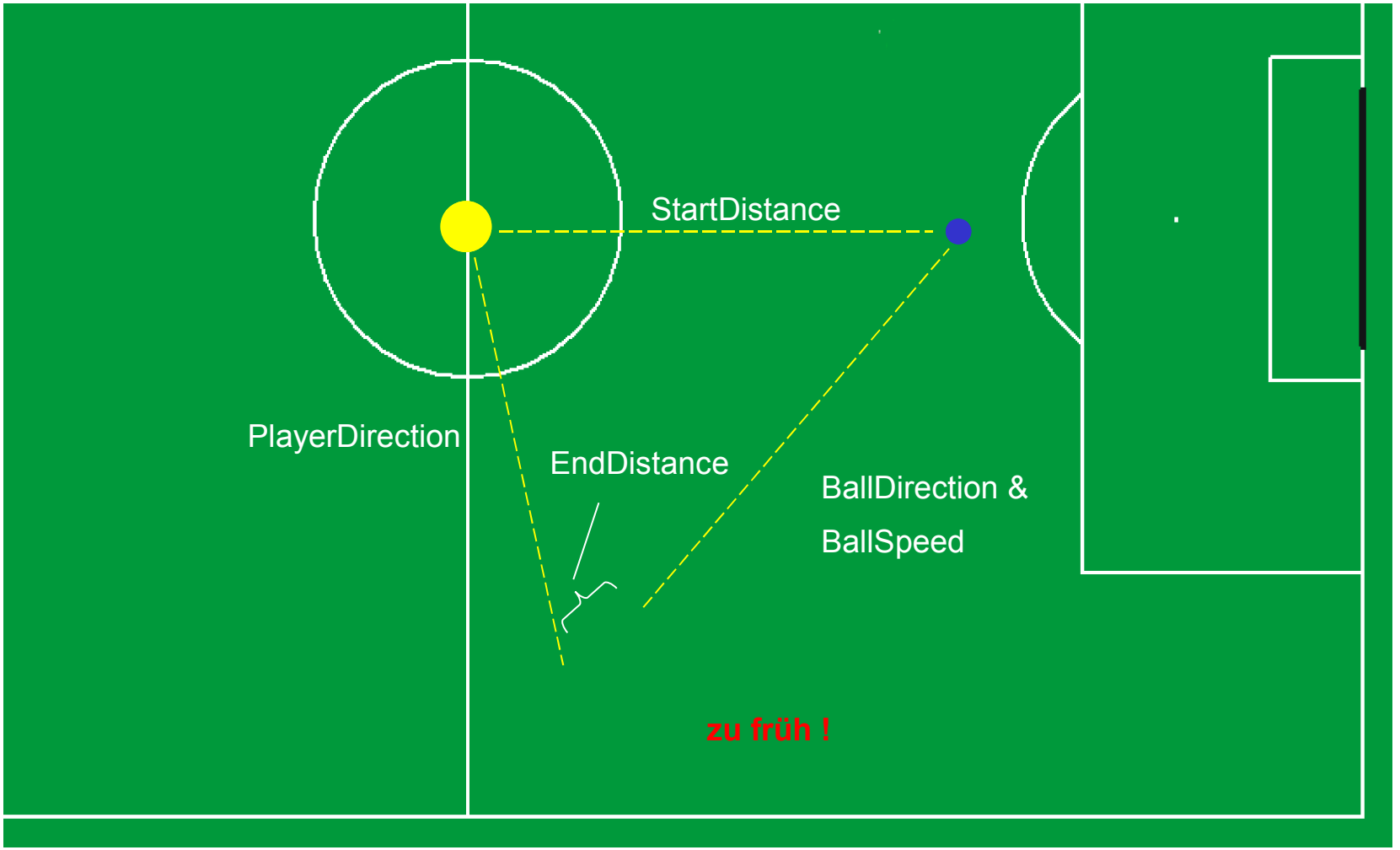
- Trainingssituation „**Ballabfangen**“ bei anderen Teams
  - Nur 34% positive Situationen, in denen der Ball richtig abgefangen wurde

  → Zu wenig Trainingsdaten gesammelt (ca. 4000)

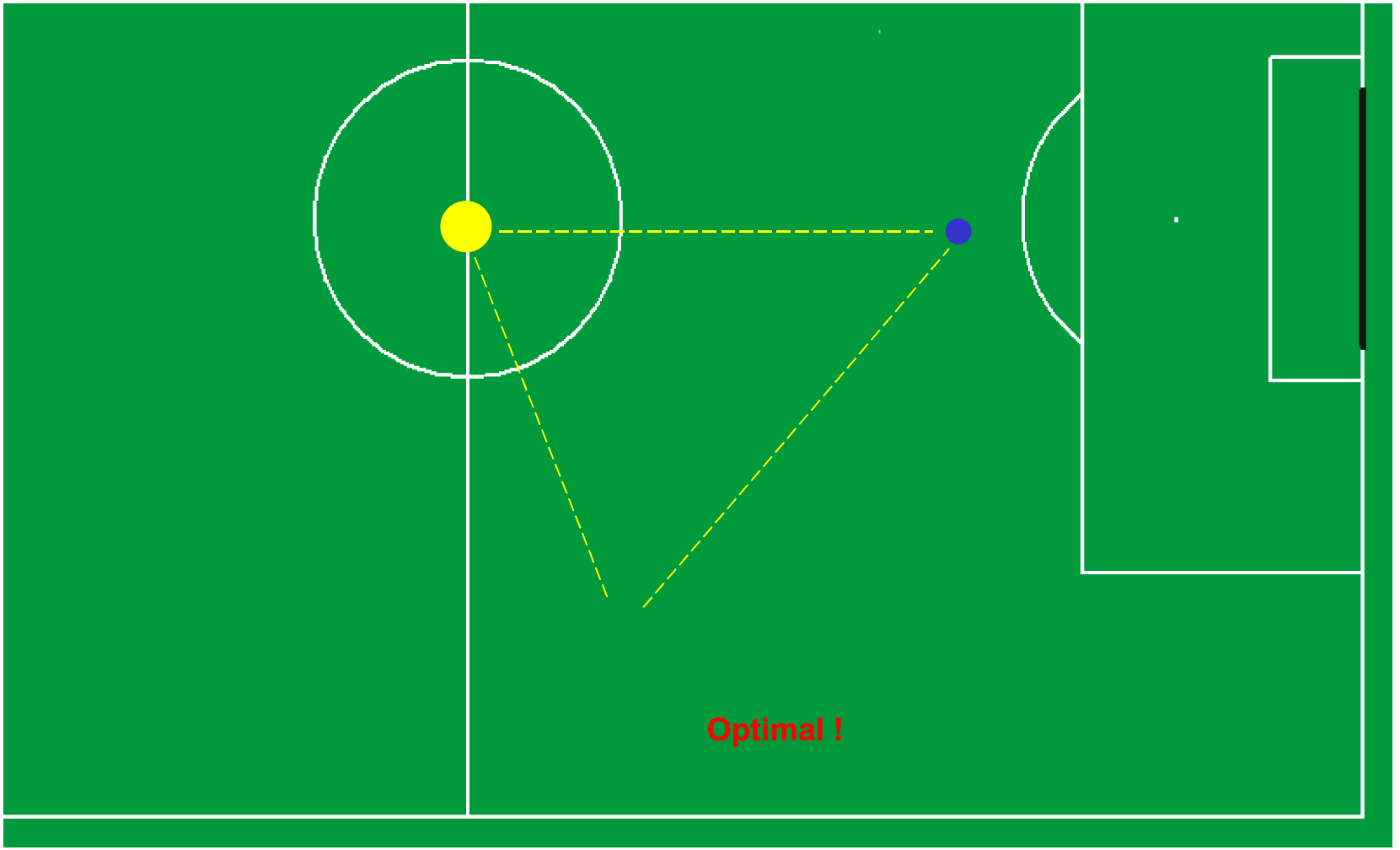
  → In unserem Training: ca. 120.000 Datensätze gespeichert

  ➔ Größere Wahrscheinlichkeit eines guten Ballabfangens

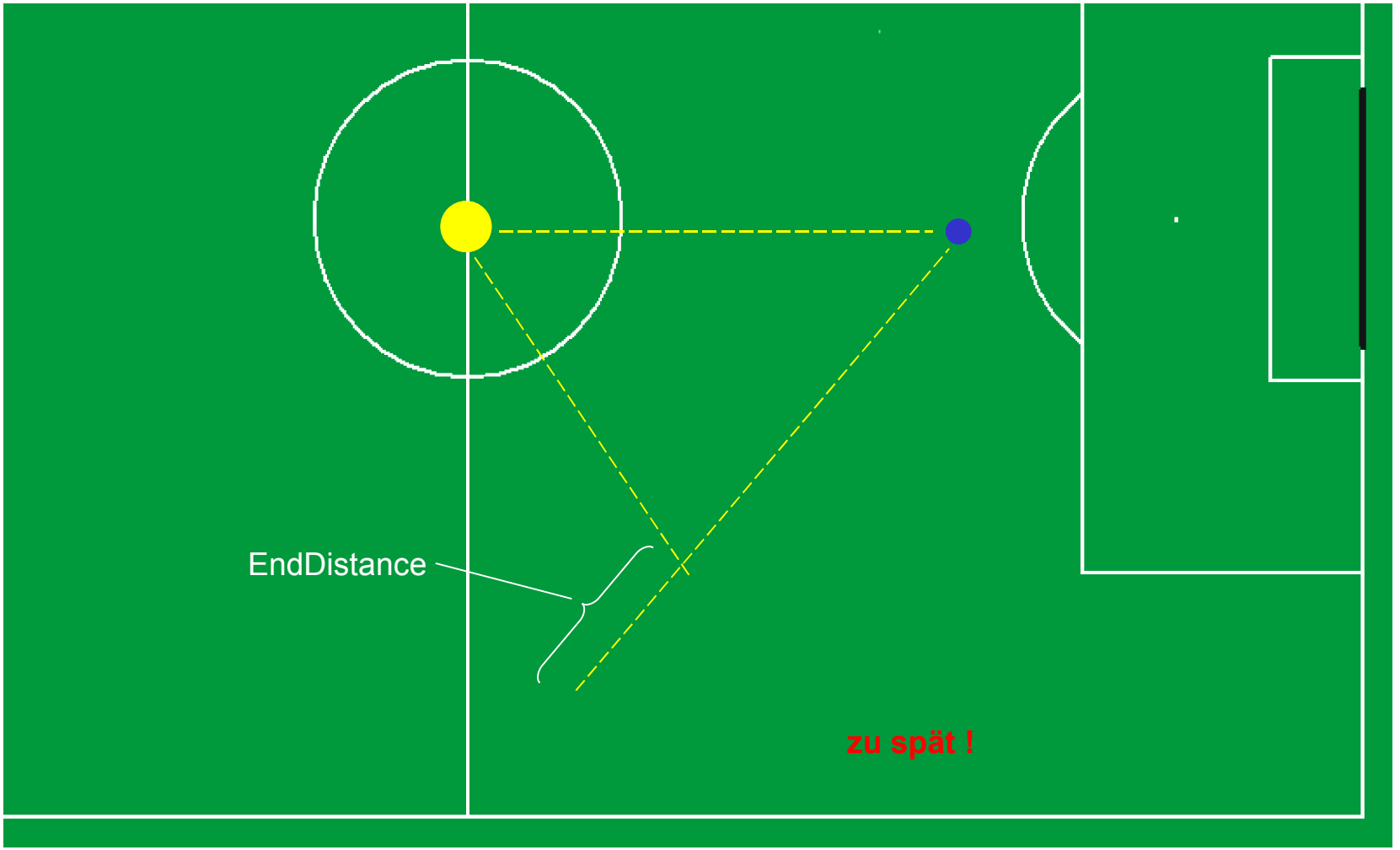
Trainingssituation - Ballabfangen



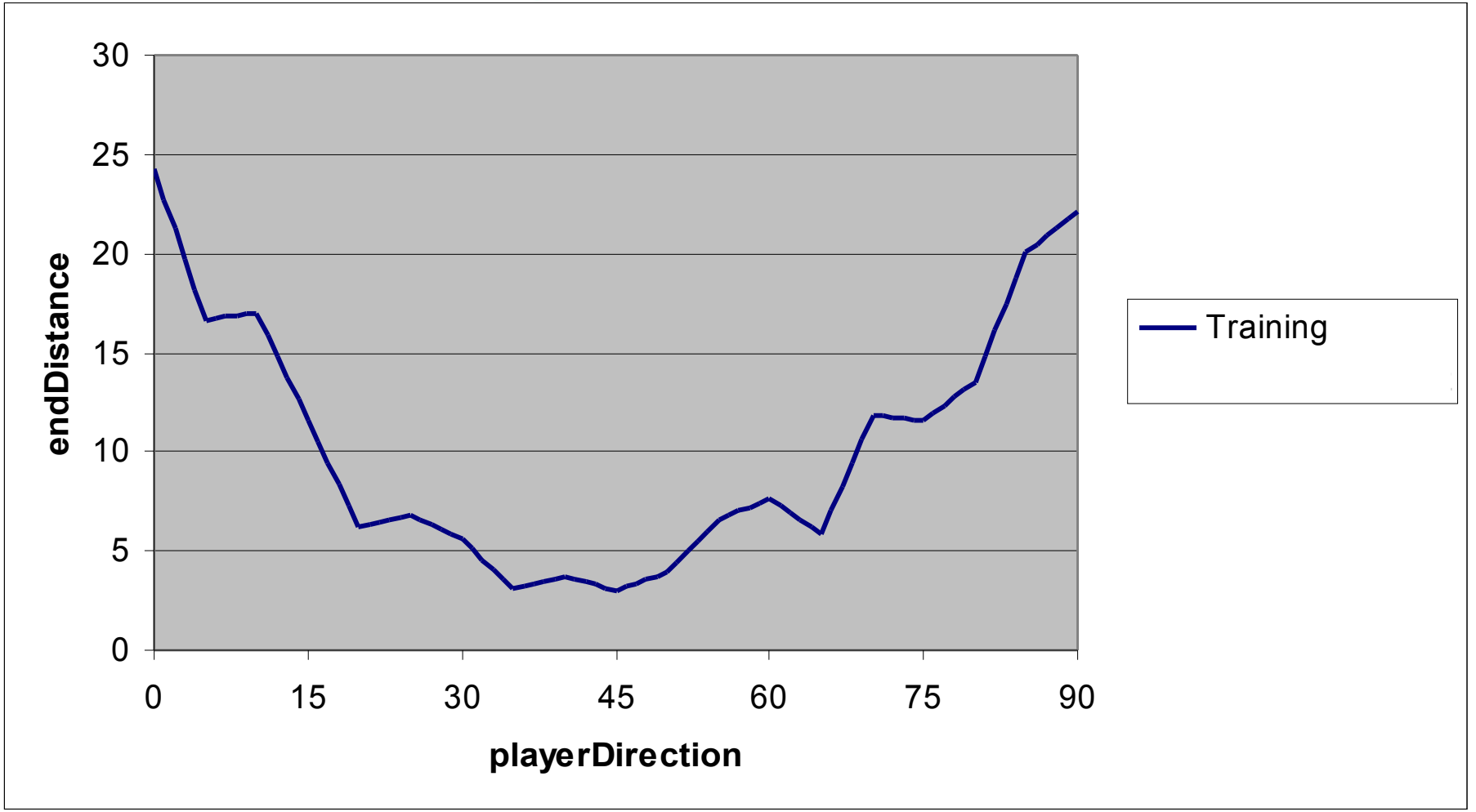
Trainingssituation - Ballabfangen



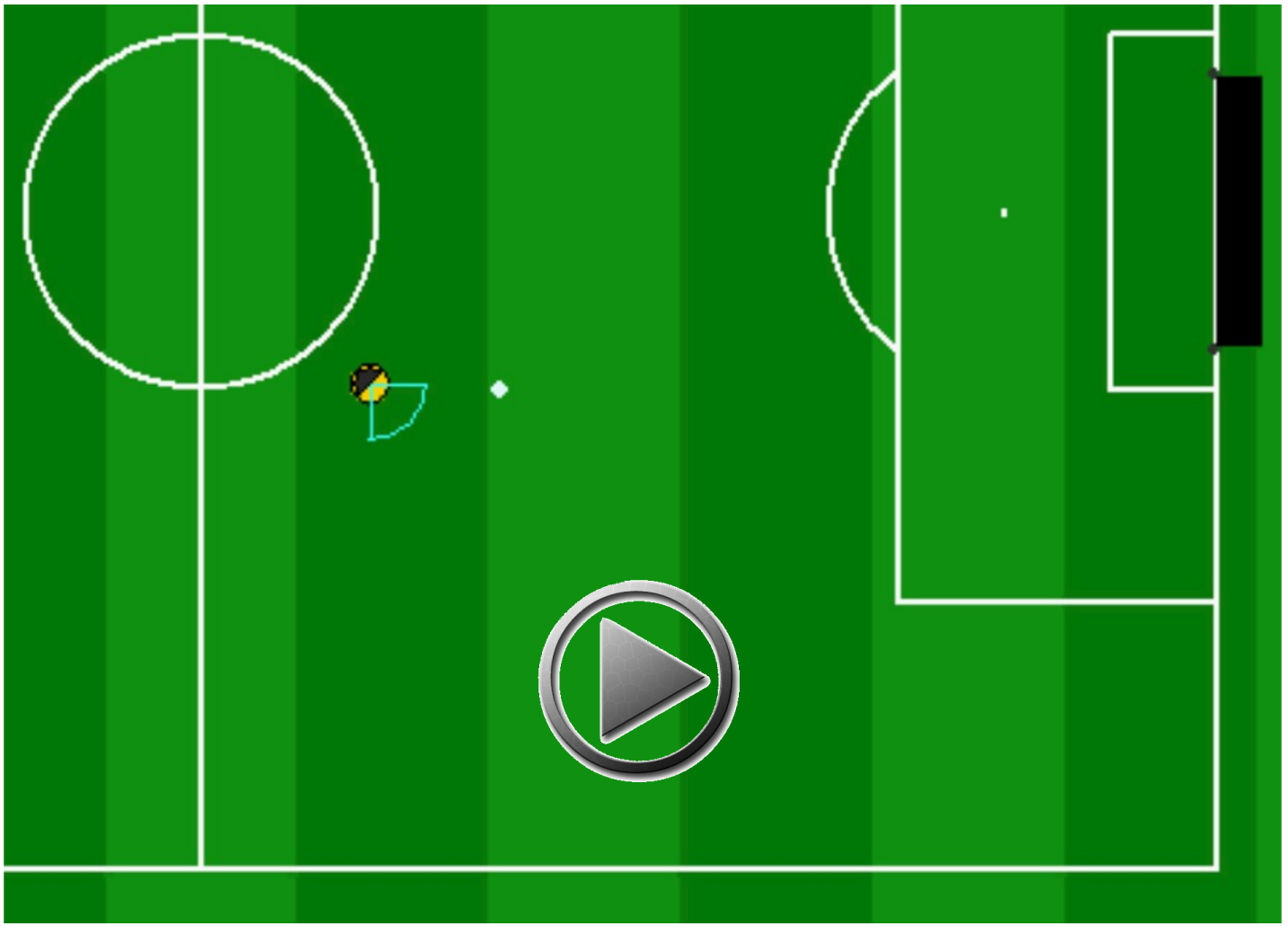
Trainingssituation - Ballabfangen



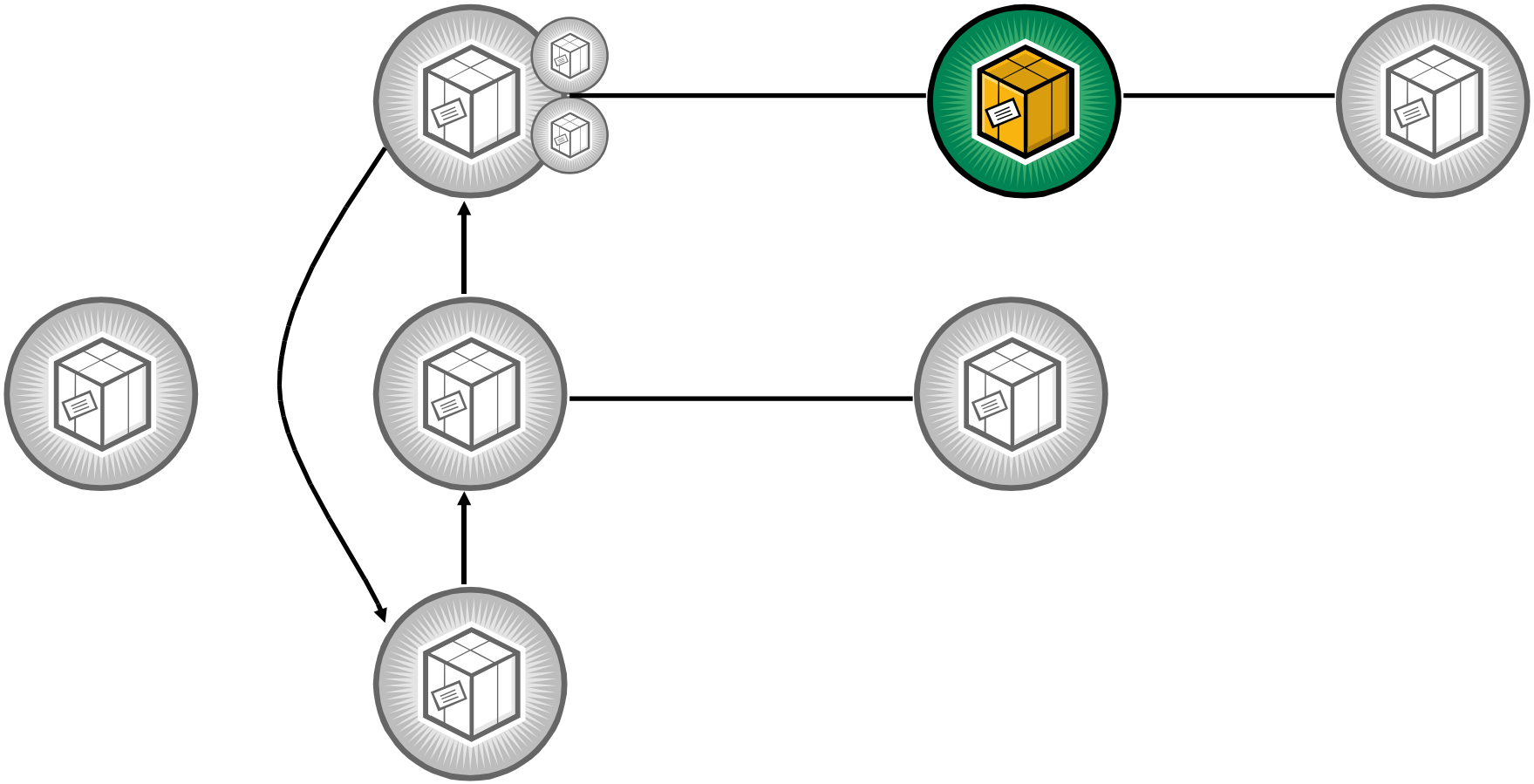
Trainingsdaten



Trainingssituation



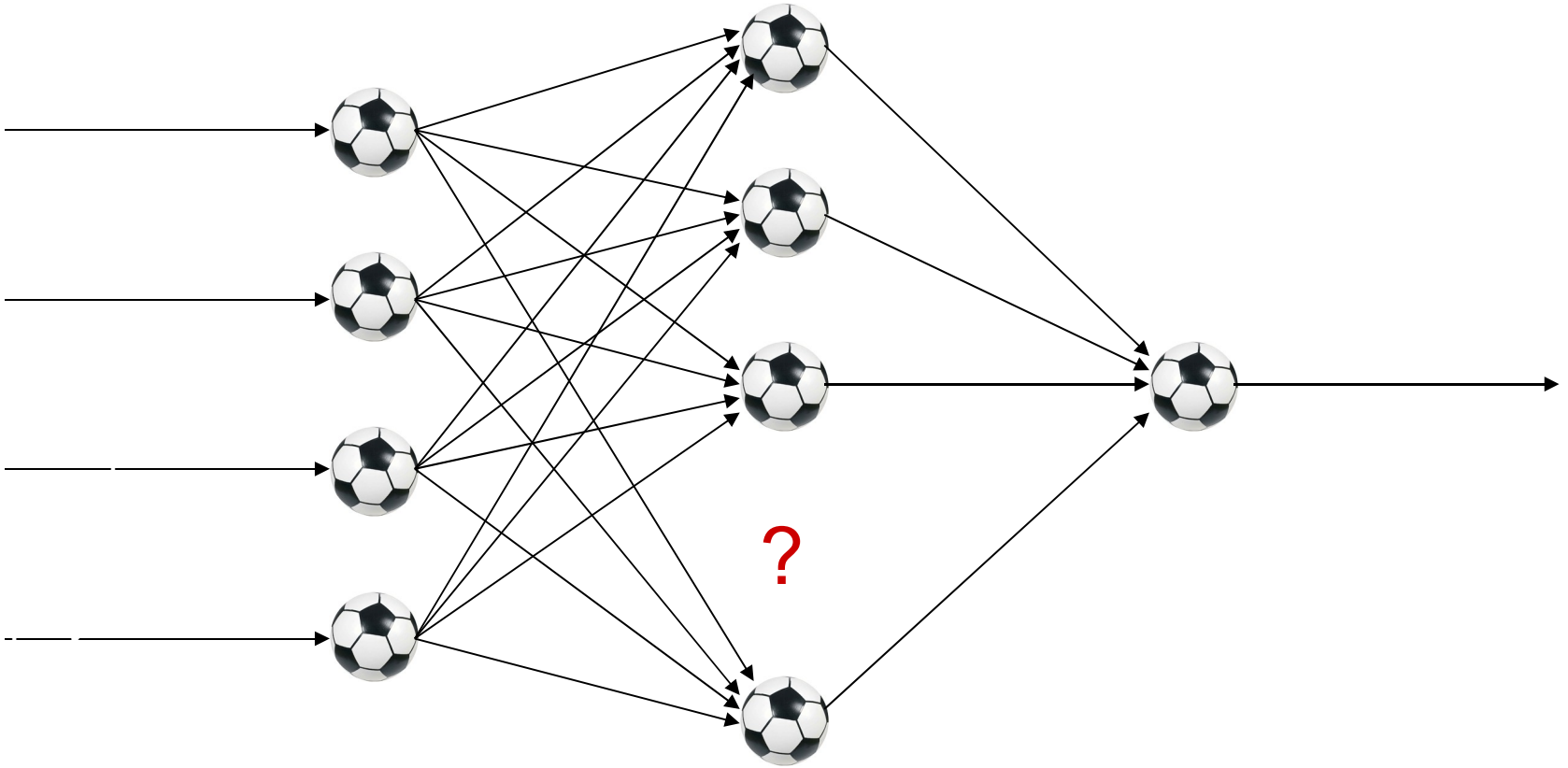
Überblick über die Java -Pakete



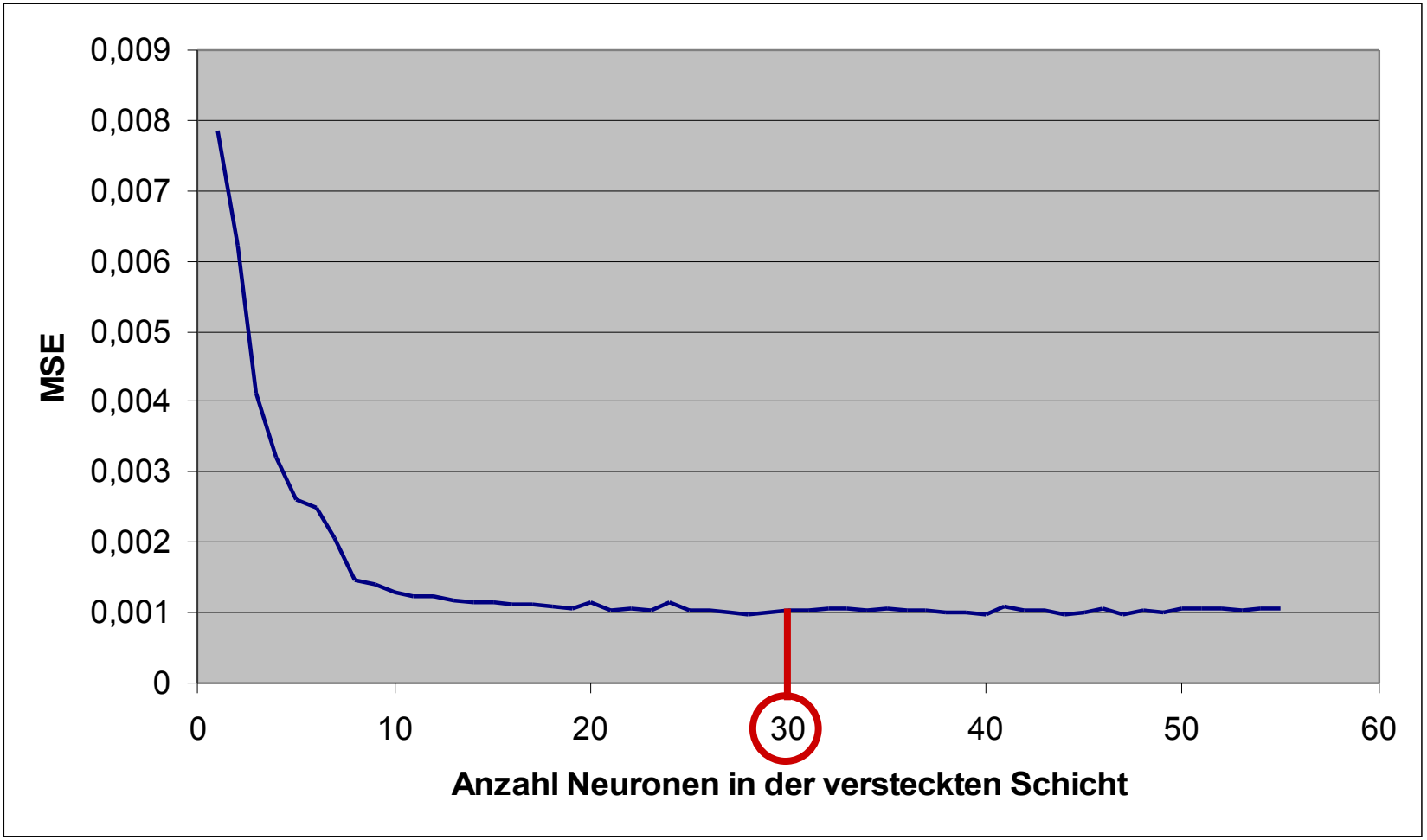
errorBackpropagationNet

- Aufbau des Error-Backpropagation-Netzes:
  - 1 Eingabeschicht
  - 1 versteckte Schicht
  - 1 Ausgabeschicht
  - mit jeweils beliebig vielen Neuronen
- Zur Verfügung stehende Methoden:
  - Initialisieren
  - Trainieren
  - Testen
  - Anwenden

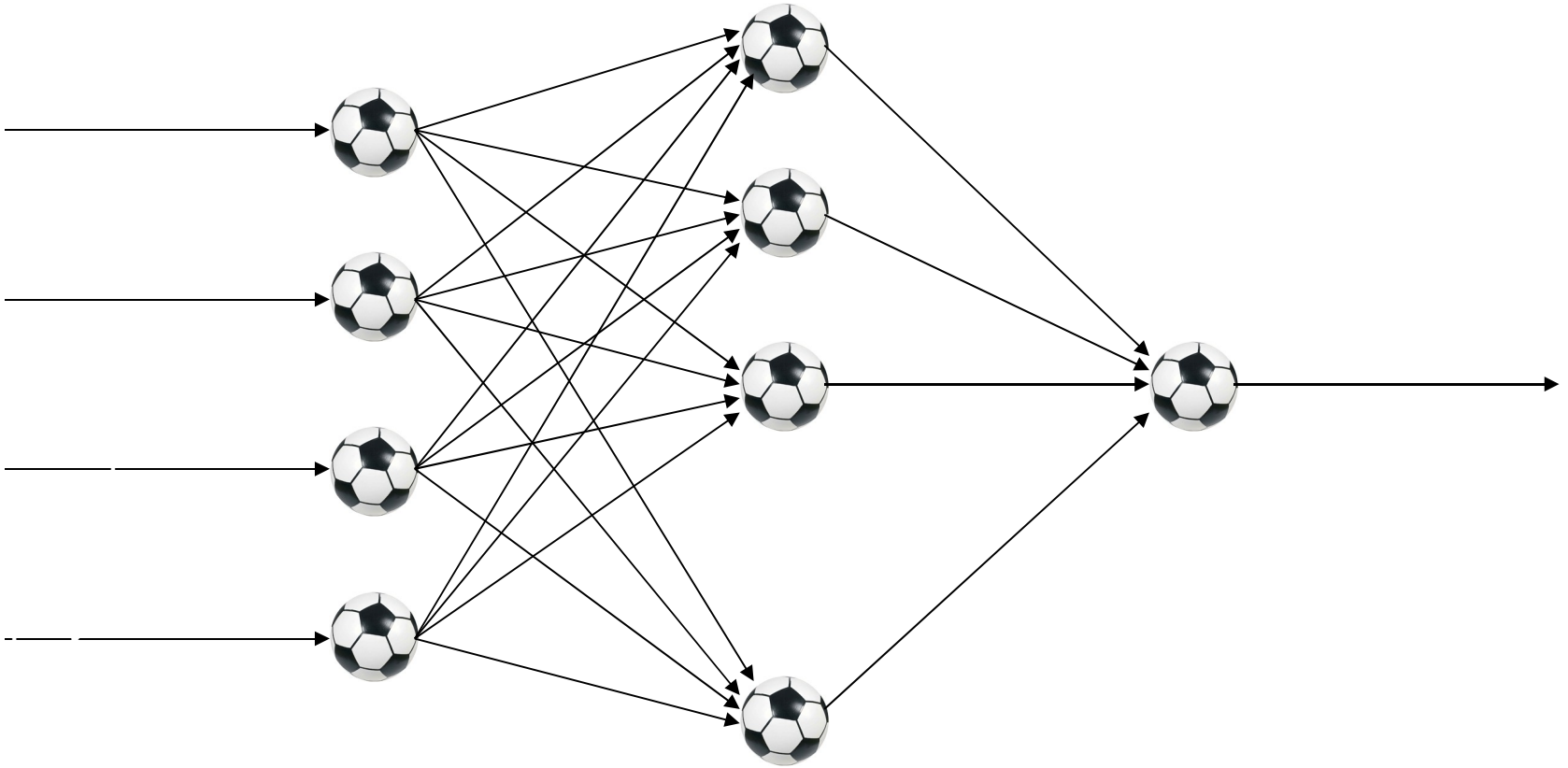
1. Error-Backpropagation-Netz



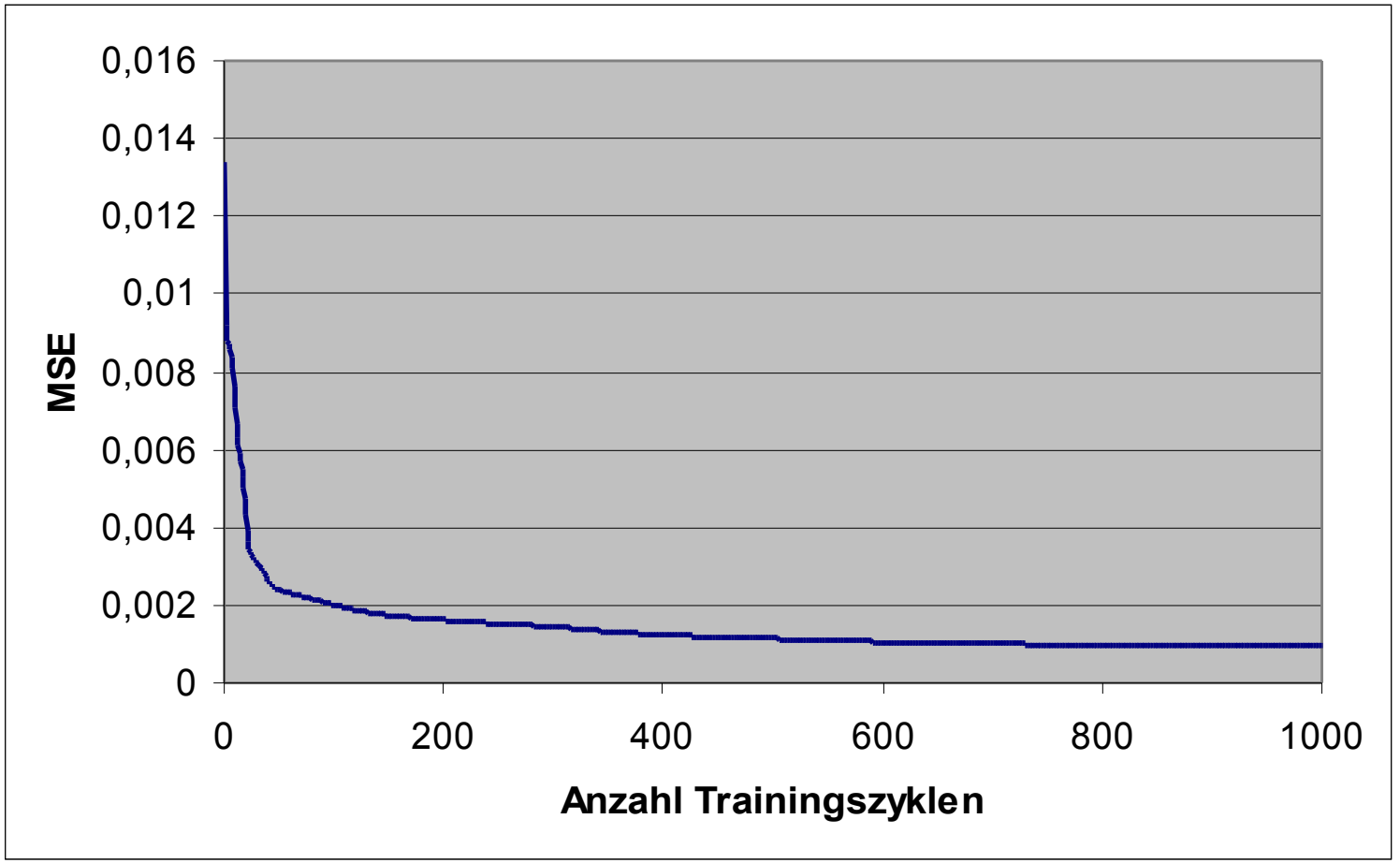
Anzahl Neuronen



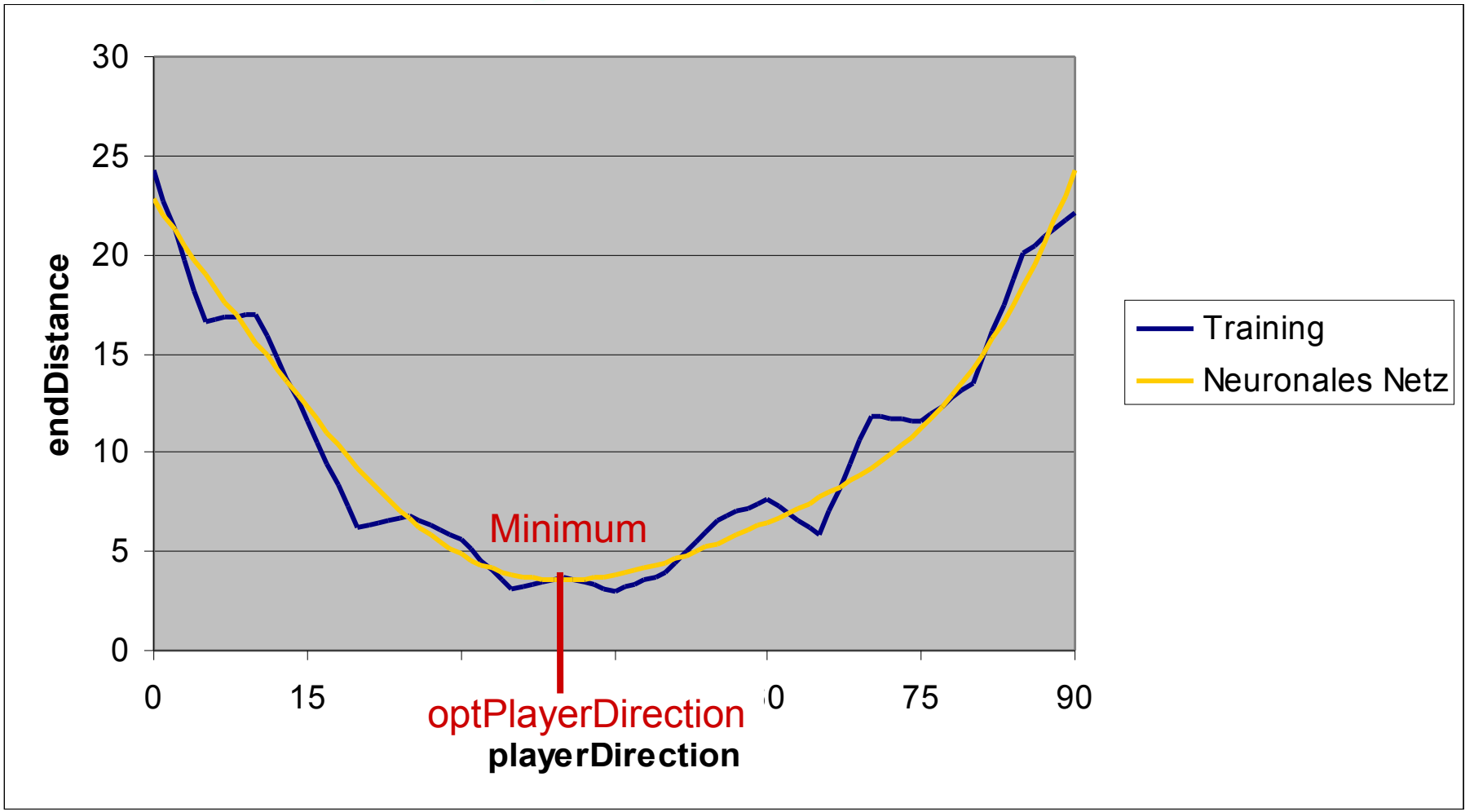
1. Error-Backpropagation-Netz



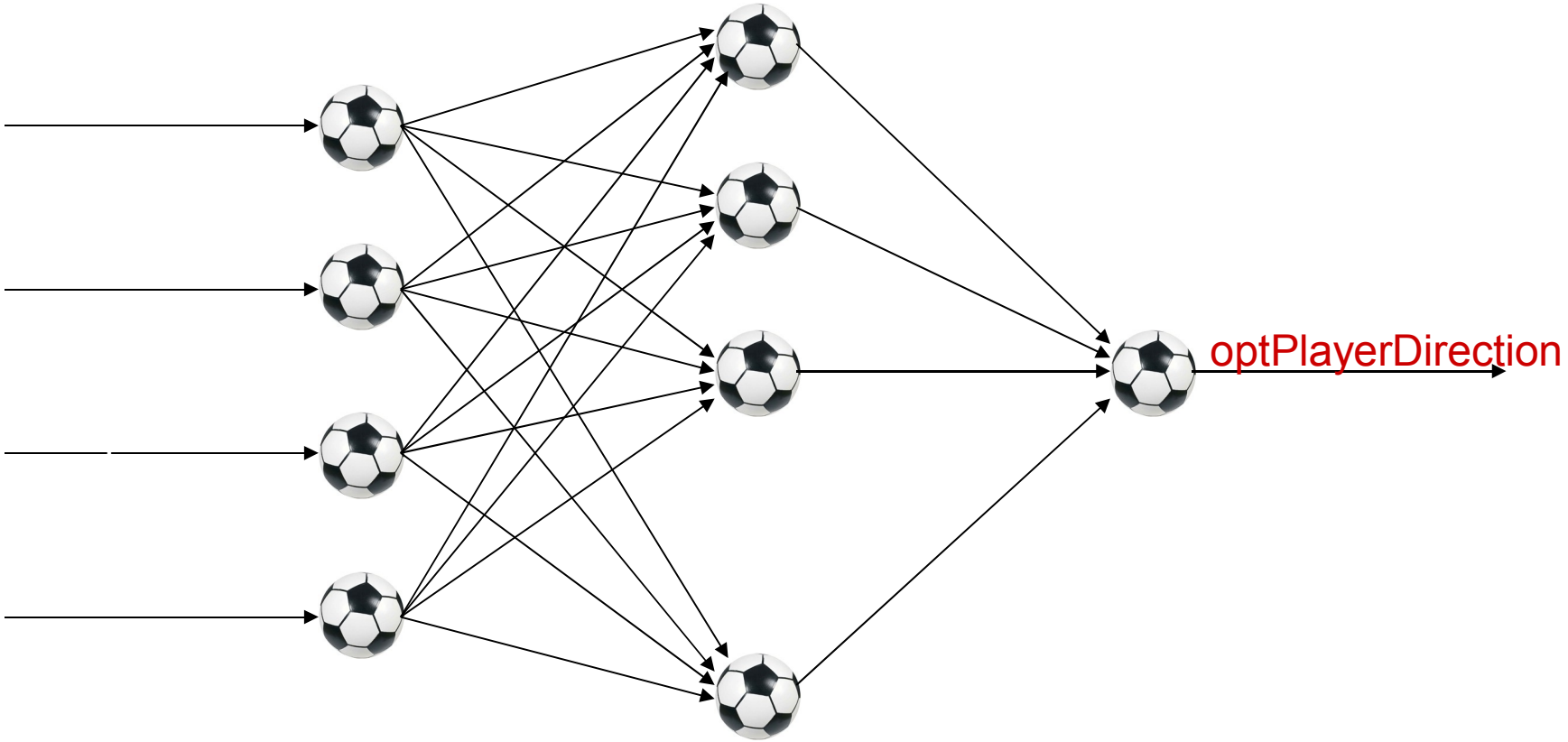
Anzahl Trainingszyklen



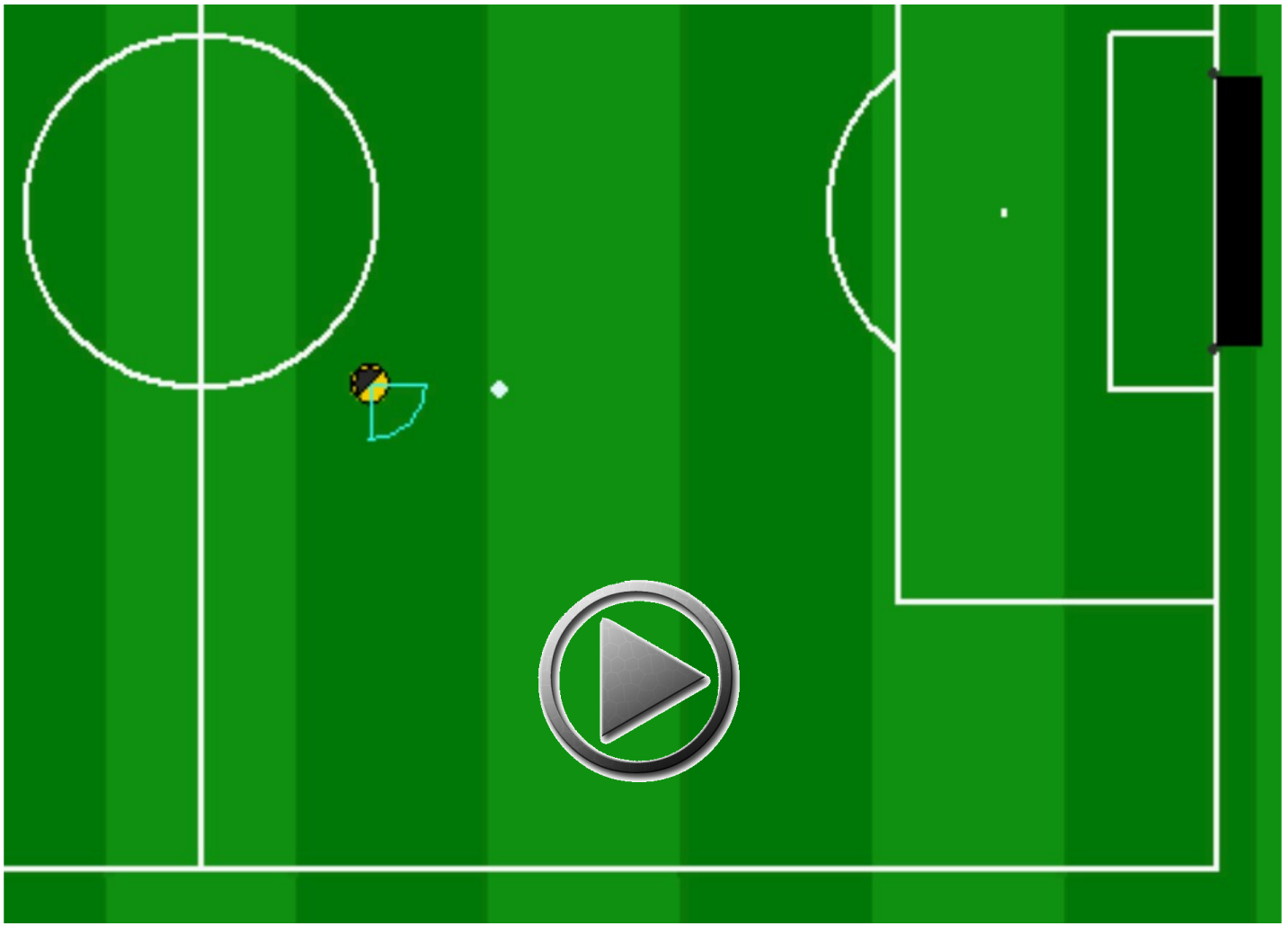
Ergebnis


30
25
20
15
10
5
0
endDistance
0
15
0
75
90
Minimum
optPlayerDirection
playerDirection
Training
Neuronales Netz

2. Error-Backpropagation-Netz



Trainierter Spieler



Gliederung

- Einleitung
- Aufbau der Mannschaft
- Nutzung künstlicher Intelligenz
- **Bewertung und Zusammenfassung**

Zusammenfassung

- Erfüllte Ziele:
  - spielfähige und konkurrenzfähige Mannschaft
  - Einbau von künstlicher Intelligenz
- Aufgetretene Herausforderungen:
  - Realisierung (nur) durch evolutionären Entwicklungsprozess
  - Koordination des Teams über den gesamten Zeitraum
- Fazit:

„Die WWUScorchers stellen eine Mannschaft mit umfangreichen Fähigkeiten, integrierter KI und der Option der einfachen Erweiterbarkeit dar."


WWU
SCORCHERS

...und jetzt wird gespielt!