

Prüfer:	Prof. Paul Levi
Betreuer:	Moritz Schulé
Beginn am:	03.04.00
Beendet am:	02.10.00

Studienarbeit-Nr. 1783

Steuerung und Aktionsvisualisierung des CoPS-  
Stuttgart Teams

Thomas Karle

# 1 Einleitung

## 1.1 Motivation

Das Multi-Agentensystem einer Fußballroboter-Mannschaft ist kompliziert und die Bewegung der einzelnen Spieler nicht mehr immer direkt nachvollziehbar. Immer wieder kommt es vor, daß einzelne Spieler scheinbar sinnlos über den Platz laufen oder sich nur suchend im Kreis drehen, obwohl der Ball sich direkt vor ihnen befindet.

Im normalen Fußball hat sich inzwischen die Videoanalyse zur Verbesserung der taktischen (und auch technischen) Maßnahmen durchgesetzt. Auch beim Roboterfußball kann die Videoanalyse viel bringen, allerdings ist es sinnvoller, die vorhandene Kommunikation und die Selbstlokalisierung der Roboter auszunutzen und in die Analyse mit einzubringen.

Zur Realisierung kann ein Rechner, der außerhalb des Spielfeldes stationiert ist, als Trainer die Daten empfangen, speichern, darstellen und auch Anweisungen an die Spieler zurücksenden. Auswechslungen müssen zwar weiterhin von Hand durchgeführt werden, aber es sollte auch möglich sein, alle Spieler von einem Rechner gleichzeitig zu starten, zu stoppen und zu unterbrechen, wenn es die Regeln verlangen. Besonders Regelverstöße, die zu Minutenstrafen oder Ausweisungen aus dem Spielfeld führen, könnte man bequem vom Bildschirm aus steuern.

Zusätzlich ist es auch hilfreich, wenn der Spielstand und die vergangene Zeit (oder verbleibende Spielzeit) zur besseren Übersicht zentral dargestellt wird.

## 1.2 Aufgabenstellung

### 1.2.1 Die Spielfelddarstellung

Das Spielfeld soll in einem eigenen Fenster dargestellt werden, das möglichst verschiedene Größen annehmen kann. Die Farben entsprechen den Farben des Originalfeldes, es gibt also ein grünes Hauptfeld, ein gelbes und ein blaues Tor, weiße Berandungen, Ecken, Mittellinie, Strafräume und eventuell ein weißer Anstoßkreis. Die Spielfeldgröße ist proportional zum Originalfeld. Das Spielfeld muß in allen vier Richtungen darstellbar sein, damit bei der Anwendung keine Mißverständnisse auftreten können. Zusätzlich dazu sollte auch ein Pfeil andeuten, in welche Richtung die eigene Mannschaft spielt.

Die Spieler werden durch einen nicht in den Spielfeldfarben ausgefüllten Kreis dargestellt und die Blickrichtung durch einen schwarzen Balken angezeigt. Weiterhin sollten zu jedem Spieler Zusatzinformationen (mindestens die Spielernummer, evt. aktueller Zustand) in der Nähe des Kreises dargestellt werden, um Verwechslungen zu vermeiden. Hat ein Spieler den Ball im Besitz, so kann dies vielleicht durch einen weiteren farbigen konzentrischen Kreis angedeutet werden.

Die von den Spielern ermittelten Ballpositionen werden durch einen ausgefüllten Kreis in der jeweiligen Spielerfarbe an den entsprechenden Position dargestellt. Möglich ist eine Mittelung der Balldaten zu einem gemeinsamen orangen Ball, wobei sehr abwei-

chende Bälle immer noch einzeln gezeichnet werden sollen, die in der Mittelung einbezogenen Bälle aber nicht.

Optional kann eine Gegnermarkierung eingeführt werden, die alle unbekanntem Objekte mit einem schwarzen Kreis markiert. Eventuell kann hier eine Bewegungsrichtung dadurch dargestellt werden, daß alte Positionen noch mit einem Punkt markiert werden, so daß sich ein Schwanz am schwarzen Objekt ergibt.

### **1.2.2 Das Kontrollpanel**

Das Kontrollpanel soll Steuerungsknöpfe enthalten, welche die gesamte Mannschaft steuern kann. Hierzu gehören die Möglichkeiten, die Spieler starten zu lassen, stoppen zu lassen, zu unterbrechen, ihre Startpositionen einnehmen zu lassen usw. Die Steuerungsbefehle dürfen nur reagieren, wenn sie sinnvoll sind.

Weiterhin muß auch jeder einzelne Roboter individuell steuerbar sein. Dazu ist es notwendig, daß er von der Mannschaft abgekoppelt werden kann. Außerdem sollen alle Mannschaftsbefehle von ihm auch einzeln erledigt werden können. Jeder Spieler soll auch weitere Informationen (eventuell in einem zusätzlichen Fenster) darstellen können, wie seinen Status oder sein Kamerabild. Sinnvoll sind eigene Programme oder Programmausführungen bei Strafstoßen oder dem sogenannten Contest-Wettbewerb.

Zur Identifizierung ist jeder Spieler mit Namen und Nummer auf dem Controlpanel angezeigt. Zur Vermeidungen von falschen Standortinformationen kann bei jedem Spieler zusätzlich noch die Balance der Selbstlokalisierung von Ultraschall und Lasererkennung verschoben werden.

Nützliche Zusatzinformationen liefern eine Zeitanzeige zur Darstellung von vergangener oder verbleibender Spielzeit, sowie eine Spielstandsanzeige, die beide Steuerungsknöpfe für die richtige Einstellung oder Wertänderung benötigen.

## **1.3 Durchführung**

Da es sich bei diesem Programm um eine Realzeitanwendung handelt, ist es besonders wichtig, daß es keine zu umfangreichen Berechnungen beinhaltet. Aus diesem Grund werden die verwendeten Algorithmen einfach gehalten, auch wenn die Genauigkeit damit herabgesetzt wird.

Um bei der Darstellung auf vorgefertigte Elemente und vor allem auf eine große Farbtiefe zurückgreifen zu können, wurde die Programmiersprache Java gewählt, die es ermöglicht Lightweight-Oberflächenelemente zu benutzen und vor allem problemlos bei der Spielfelddarstellung auf eine Farbtiefe von 24 Bit zuzugreifen, die genügend Freiheiten für die Farbwahl der verschiedenen Spieler erlaubt. Java ermöglicht es auch, die Anwendungen plattformunabhängig zu gestalten, wobei ein sicheres Funktionieren nur bei Red Hat Linux, dem benutzten Betriebssystem gewährleistet ist.

Für die Einbindung der Lightwightelemente wurde mit Java-Swing programmiert, wobei keine weiteren Oberflächenstrukturen entwickelt wurden, sondern nur die jeweilige Windowsoberfläche von den entsprechenden Windowmanagern übernommen wurde.

Ein wichtiger Bestandteil für das Programm ist die Datenübertragung von den einzelnen Spielern zu dem Trainer. Sie wurde mittels Datenkanäle über CORBA realisiert, wobei sie allerdings keine Bestandteil dieser Studienarbeit sind. Durch ihre Notwendigkeit werden sie allerdings mehrmals in dieser Arbeit erwähnt, besonders in Hinblick auf noch zu erweiternde Bereiche des Programms.

## 2 Aktionsvisualisierung des CoPS Team

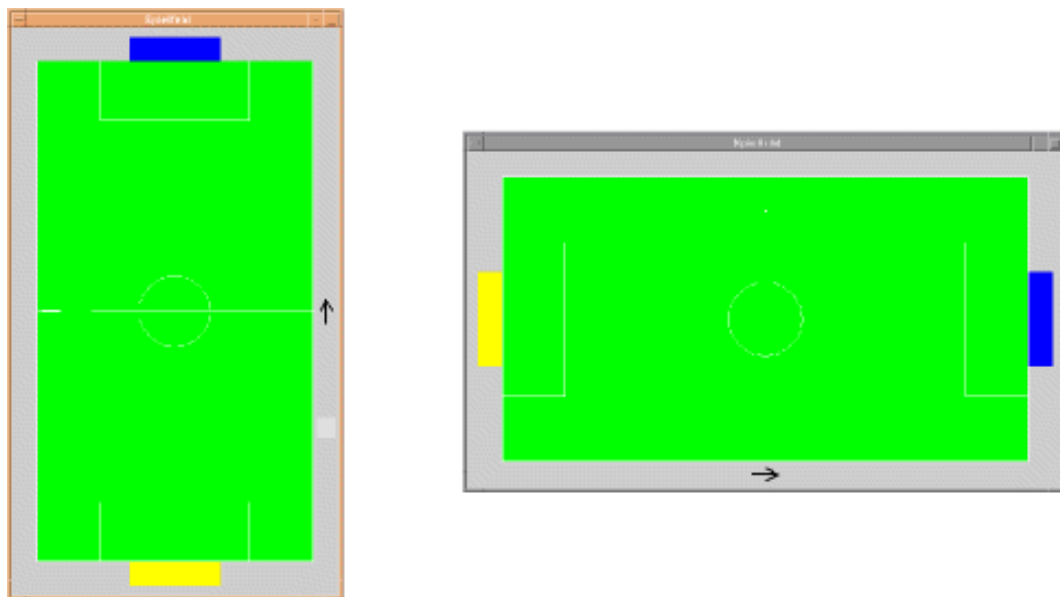
Das erste Programm ist zur Visualisierung des Spieles. Es ist als eigenständiges Programm geschrieben um eine bessere Modularisierung herzustellen. Außerdem ist es auch von seiner Funktion prinzipiell unabhängig vom anderen Aufgabenteil. Es werden neben der Spielfelddarstellung die Spieler und ihre vermuteten Ballpositionen simuliert, was eine bessere Übersicht während dem Spiel und durch Speicherung der Positionen in einer Log-Datei spätere Analysen ermöglicht.

### 2.1 Visuelle Bestandteile

Hauptaufgabe der Applikation ist die Darstellung der einzelnen Objekte auf dem Bildschirm in einem eigenen Fenster. Die Objekte werden hier unterschiedlich gehandhabt, je nachdem ob sie fest oder beweglich sind. Während das Spielfeld im Hintergrund immer fest bleibt und dementsprechend implementiert ist, werden Spieler und Bälle als Grafiken in Labels untergebracht, um eine leicht Handhabung und schnellere Darstellung zu gewährleisten.

#### 2.1.1 Das Spielfeld

In Abbildung 1 auf Seite 5 ist erst mal ein typisches Beispiel in zwei verschiedenen Ausrichtungen abgebildet.



**ABBILDUNG 1. Screenshots von Spielfeldern vertikal und horizontal**

Die wichtigste Eigenschaft, die in der Darstellung eingehalten werden soll, ist die Proportionalität der Feldmaße. Hierfür wird die Spielfeldlänge auf 100 Pixel festgelegt. Alle anderen Maße wie Spielfeldbreite, Torbreite und Tortiefe werden proportional dazu umgerechnet und in Pixeln dargestellt.

Auf einem Bildschirm mit 17 bis 20 Zoll Durchmesser ist eine Darstellung mit 100 Pixeln sehr klein. Deswegen ist es bei Programmaufruf möglich, eine Skalierung anzugeben, welche mit der Spielfeldlänge multipliziert wird, so daß eine Länge von 200 bis 800 oder mehr Pixeln entstehen kann.

Die Farben des Spielfelds werden aus den Hauptfarben von Java genommen. Grün für den "Rasen", blau und gelb für die Tore, sowie weiß für die Spielfeldbegrenzungen. Alle diese Hintergrundflächen (außer dem Mittelkreis) werden durch einfache Linien oder Rechtecke gebildet, was zu einer schnellen Umsetzung führt.

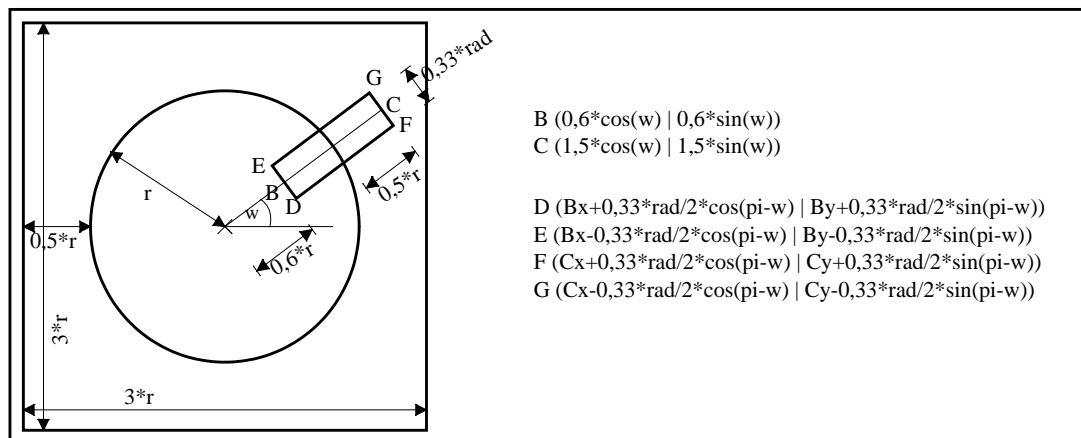
Der Pfeil zum Anzeigen der Spielrichtung der eigenen Mannschaft ist bei horizontalen Spielfeldern unterhalb, bei vertikalen Spielfeldern rechts angebracht und wird durch einen geschlossenen Polygonzug repräsentiert.

Um das Fenster nicht zu eng um das Feld zu legen, wurde noch Ränder an beiden Seiten und hinter den Toren angebracht. Die Größe der Ränder sind in der Konfiguration einstellbar, genauso wie Länge, Breite und Linienbreite des Pfeiles. Die Skalierung ändert die Größe des Spielfeldes, somit müssen auch der Pfeil, die Strafraumlinien, die Ecken und der Mittelkreis an diese Größenänderungen angepasst werden können.

Eine weitere Schwierigkeit ergibt sich aus den verschiedenen möglichen Richtungen des Spielfeldes. Unterschieden wird bei der Darstellung des Feldes meistens nur in Horizontalität oder Vertikalität, da die Elemente meist symmetrisch angeordnet sind. Die Transformation besteht hier aus der Vertauschung von x- und y-Werten. Nur bei den Torfarben hängt die Darstellung von allen vier Richtungen und zusätzlich von der Farbe des eigenen Tores ab. Zur Vereinfachung wird die Geometrie und die Farbe des Tores getrennt behandelt. Die Geometrie wird wie oben umgerechnet und das Rechteck in Abhängigkeit von Richtung und der eigenen Torfarbe eingefärbt.

### 2.1.2 Die Spieler

Die Darstellung des Spielers besteht aus zwei Elementen. Einmal ein farbiger Kreis an der entsprechenden Position mit einem schwarzen Balken als Orientierung. Dieses Element wird als Icon, einer kleinen Grafik implementiert. Das zweite Element ist ein Textlabel, der Zusatzinformationen darstellt, wie Spielernummer, Spielerzustand usw.



**ABBILDUNG 2. Berechnungen für ein Spielericon**

Auch der Spieler muß die richtige proportionale Größe im Spielfeld haben. Deswegen wird die Robotergröße in einen Spielerradius umgerechnet, der die Grundlage für die Berechnungen im Spielericon sind. Die Größe des Icons ist so dimensioniert, daß neben einem Kreis mit doppeltem Radius als Durchmesser auch noch der schwarze Richtungsbalken, der mit halben Radius übersteht, aufgenommen werden kann.

Die Endpunkte des Balken werden mit Winkelberechnungen über Sinus und Cosinus durchgeführt, so daß der Balken von 60% bis 150% vom Radius verläuft. Die Balkenbreite wird auf ein Drittel vom Radius gestellt. Dies sichert, daß er deutlich zu sehen ist, aber das Bild nicht dominiert.

Durch eine neue Winkeleingabe werden diese Endpunkte neu berechnet und bei einer Darstellung des Icons der Balken als gefüllter Polygonzug gezeichnet.

Die Zusatzangaben im Textfeld wie Rolle oder Zustand werden als Abkürzungen (in der Konfiguration einstellbar) geschrieben und mit einem Zeichen getrennt, um die Ausgabe nicht zu unübersichtlich zu machen.

### **2.1.3 Der Spielball**

Die Darstellung eines Balles entspricht die eines Spielers in vereinfachter Form. Er besteht aus einem dem jeweiligen Spieler entsprechender Farbe ausgefüllten Kreis mit dem aus der Ballgröße berechneten Radius.

Die einzige Erweiterung besteht hier in der Möglichkeit, daß der Ball eventuell gar nicht dargestellt wird, obwohl er vorhanden ist. Bei einer Mittelung von mehreren Ballpositionen wird anstelle der jeweiligen Bälle an der gemittelten Position ein Ball in einer besonderen Farbe (vorzugsweise orange) gezeichnet. Sind die vermuteten Ballpositionen zu weit auseinander, so werden nur wenige (oder keine) mit in die Berechnung mit einbezogen und die restlichen Bälle einzeln dargestellt.

Bei den beiden Elementen Spieler und Ball benötigt man neben der Icondarstellung für den Label einen Aufpunkt als Positionsangabe. Die Spieler senden ihre Originalposition in der Einheit Meter und dem Schnittpunkt der hinteren Torkante des eigenen Tores mit der unteren Spielfeldaußenkante als Nullpunkt. Diese wird zuerst in Abhän-

gigkeit der Skalierung in Pixel übertragen und anschließend wie später gezeigt transformiert.

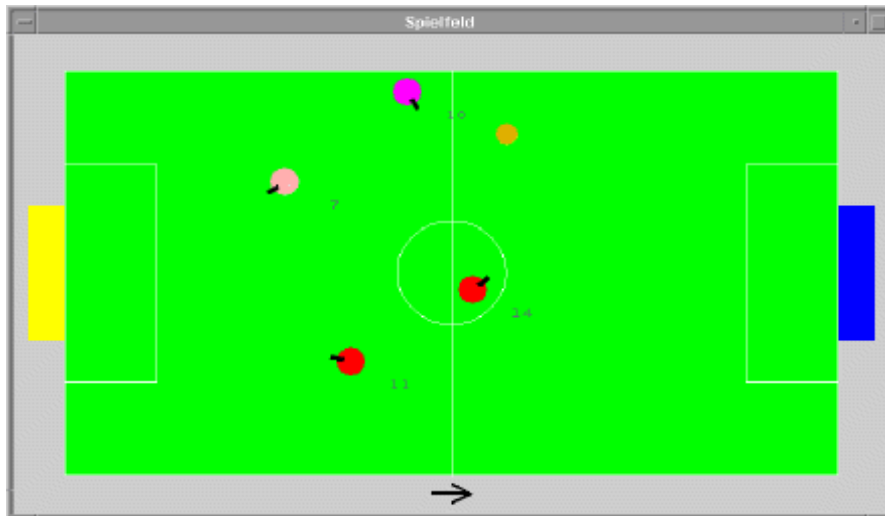


ABBILDUNG 3. Screenshot Spielfeld mit Spieler und fusioniertem Ball

## 2.2 Datenstrukturen

### 2.2.1 Spieler

Um einen Spieler umfassend zu repräsentieren, benötigt man neben seinen Namen, Spielernummer und Position noch einige weitere wichtige Daten. Es kommt hier auf Grund der Übersichtlichkeit oder zur schnelleren und logischeren Abarbeitung zu einigen Redundanzen, bei denen man überprüfen muß, ob sie bei Programmänderungen konsistent bleiben. Insbesondere die Positionen werden immer doppelt gespeichert. Einmal die Originalposition im Feld und weiterhin die Position auf dem Bildschirm..

TABELLE 1. Datenstruktur Spieler

Datum	Name	Typ
Name des Spielers	spielerName	String
Nummer des Spielers	spielerNummer	int
Originalposition in Meter	OrigPosX,origPosY	double,double
Position in Pixel	posX,posY	int,int
Winkel in Meter/Pixel	origWinkel,Winkel	double,int
Ballposition in Pixeln	ballX,ballY	int,int
Original Ballposition	ballPos	BallDaten
Spieler-/Ballradius in Pixeln	radius,ballRad	int,int
Grafikicons Spieler/Ball	icon,ballIcon	SpielerIcon,BallIcon
Farbe des Spielers	farbe	Color
Sichtbarkeit des Spielers	sichtbar	boolean
Anzahl unbewegte Zeit	aktuell	int
Positionierungslabels S/B	label,ballLabel	JLabel (Swing)

**TABELLE 1. Datenstruktur Spieler**

Datum	Name	Typ
Ballsichtbarkeit	ballSichtbar,ballLabelsichtbar	boolean
Abstand Spieler zu Ball	dBallSpieler	double

Dazu kommen noch Zeichenketten, Ganzzahlen und eine Booleanvariable für die Speicherung der Rollen und Aktivitäten für den aktuellen Zustand des Spielers. Die meisten Daten sind privat für die Objektklasse, nur Icons, Labels sind öffentlich, damit andere Programmeinheiten effizient darauf zugreifen können.

Die “Anzahl unbewegte Zeit” ist zur Feststellung, ob ein Spieler eventuell nicht mehr sendet und wird bei jeder neuen Positioneingabe auf Null gesetzt. In jedem neuen Zeitschritt wird sie um eins erhöht. Bei Überschreiten eines Maximalwertes kann eine Warnung erfolgen und der Spieler aus der Darstellung genommen werden, da angenommen werden muß, daß der entsprechende Rechner abgestürzt ist. Diese Funktion ist aber nicht implementiert.

### 2.2.2 Bälle

Die Datenstruktur der Bälle ist einfacher und aufgeteilt in *BallIcon* und *BallDaten*. In *BallIcon* befinden sich der Radius und die Farbe, in *BallDaten* die Positions- und Darstellungsparameter.

**TABELLE 2. Balldaten**

Datum	Name	Typ
Position	x,y	double,double
Entfernung zum Spieler	r	double
Darstellung	zeichnen	boolean

Das Hauptanwendungsgebiet dieser Daten liegt in der Berechnung einer Ballfusion, so daß hier alle Positions- und Entfernungsangaben in Metern sind.

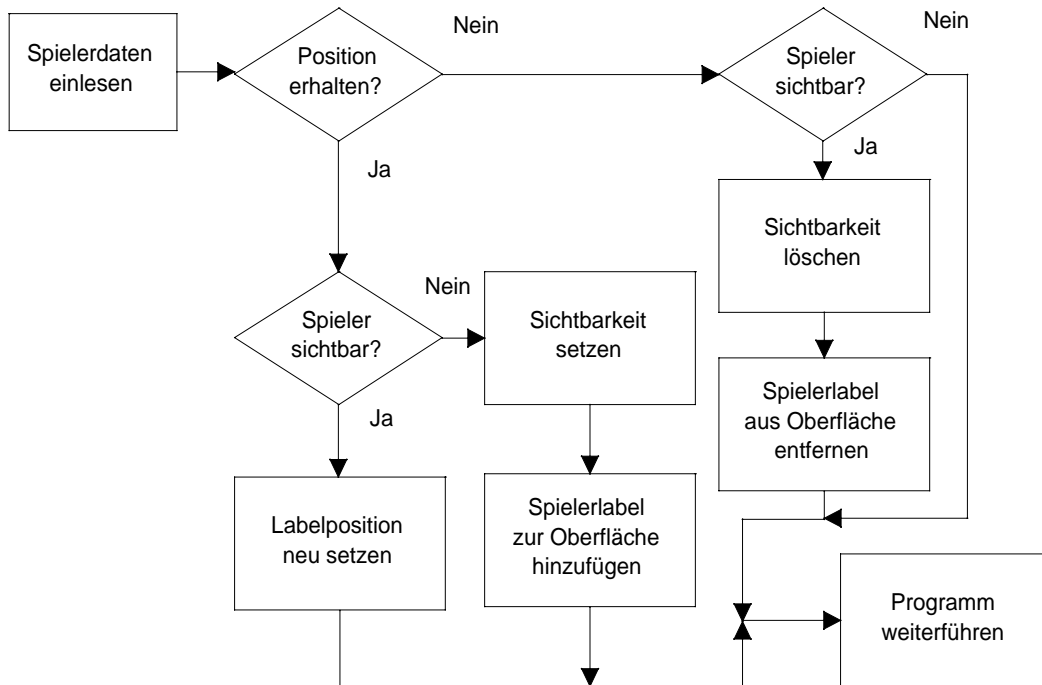
## 2.3 Algorithmische Bestandteile

In diesem Programm werden nicht viele Berechnungen durchgeführt. Die wichtigsten Algorithmen, die im Programm auftauchen werden in den nächsten Abschnitten aufgezeigt.

### 2.3.1 Timer

Das wichtigste Konstrukt zur Simulation der Spieler und Bälle ist der Timer. Am Anfang des Programms wird ein Timer initialisiert, der alle 50 Millisekunden einen *Actionevent* anstößt der in der Prozedur *actionPerformed(ActionEvent e)* abgefragt werden kann. Dies entspricht einer Wiederholrate von 20 mal in der Sekunde (etwas weniger als ein Fernsehbild). In unserem Fall ist dies ausreichend. Innerhalb dieser Prozedur werden die neuen Positionen von Spielern und Bällen mit Hilfe der Klasse *SpielerInfo*, welche die Kommunikation durch CORBA übernimmt, abgefragt.

Anschließend werden die Spielerlabels gelöscht, verschoben oder neu erstellt, damit die Spieler bei der nächsten Aktualisierung am richtigen Ort erscheinen. Die Abbildung 4 auf Seite 10 veranschaulicht die verschiedenen Möglichkeiten.



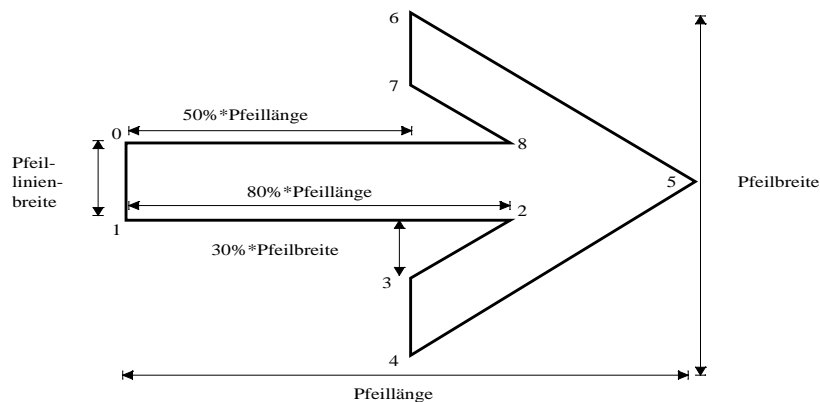
**ABBILDUNG 4. Spielerdarstellung innerhalb von Timer**

Danach wird eventuell eine Mittelung der Ballpositionen durchgeführt. Die Darstellung dieses fusionierten Balles mit Hilfe eines Labels wird dabei gleichzeitig durchgeführt. Anschließend werden für alle immer noch darzustellende Bälle die Labels erstellt, gelöscht oder verschoben.

### 2.3.2 Darstellung der einzelnen Elemente

Das Spielfeld wird prinzipiell als Hintergrundbild der Hauptapplikation gezeichnet. Die Grafikkommandos hierfür sind in der Funktion *paintComponent(Graphics g)* der Klasse *SpielfeldPanel* aufgeführt, die zuständig für die gesamte Steuerung und Darstellung ist. Jedesmal wenn sich eine Änderung im Fenster ergeben hat, wird der überdeckte Teil neu gezeichnet. Wie schon in Abschnitt 2.1.1 aufgezeigt, lässt sich das Spielfeld leicht mit Rechtecken, Linien und einem Kreis darstellen. Die zugehörigen Grafikkommandos sind *fillRect*, *drawRect*, *drawLine* und *drawArc*. Für eine Farbänderung wird der Befehl *setColor* mit den Argumenten *Color.white*, *Color.green*, *Color.blue* und *Color.yellow* ausgeführt. Der Richtungspfeil wird mit der Farbe *Color.black* und

dem Befehl *fillPolygon*, dem die Polygonpunkte mit Anzahl als Parameter übergeben wird, gezeichnet. Eine grafische Illustration befindet sich in Abbildung 5 auf Seite 11.



**ABBILDUNG 5. Richtungspfeil aus Polygon**

Die Darstellung von Ball und Spieler gründet sich auf Icons, die in einem Label aufgenommen werden. Neben dem Icon befindet sich beim Spieler noch im rechten unteren Teil des Labels ein Text, der Zusatzinformationen enthält, wie Spielernummer und augenblicklicher Zustand des Spielers. Bei einer Änderung wird einfach mit dem Befehl *setText* der entsprechende Inhalt geändert.

Bei einer Positionsänderung des Spielers oder des Balles muß das Label mit dem Befehl *setBounds* neu positioniert werden. Die Umrechnung für diese Position wird im nächsten Abschnitt erklärt.

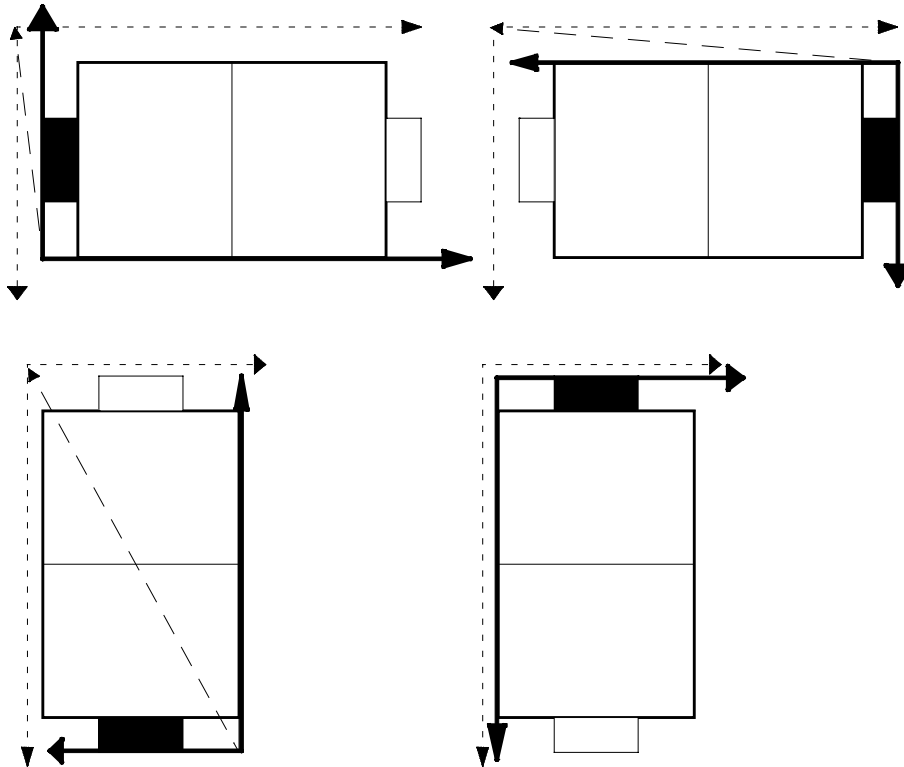
Beim Spieler entsteht noch die Möglichkeit, daß sich der Winkel der Orientierung zusätzlich ändert. Diese Änderung wird in der Klasse *SpielerIcon* selber durchgeführt. Bei einer neuen Winkeleingabe wird mit dem im Abschnitt 2.1.2 vorgestellten Berechnungen die jeweiligen Eckpunkte ermittelt und somit die darstellende Funktion *paintIcon* zu einer anderen Ausführung gezwungen. Bei beiden Iconarten wird mit *fillArc* ein gefüllter Kreis erzeugt und beim Spielericon zusätzlich ein gefüllter Polygonzug über die vier Eckpunkte des Orientierungsbalkens gezeichnet.

### 2.3.3 Umrechnen von Realfeld in Computerfeld

Die Übertragung der Spieler- und Ballpositionen erfolgt in einem Koordinatensystem, was in der Größe dem realen Spielfeldabmessungen mit der Einheit Meter entspricht und dessen Nullpunkt an der Kreuzung von der eigenen rechten Außenlinie mit der hinteren Torkante des eigenen Tores liegt. Die Positive X-Achse verläuft in Spielrichtung, die positive Y-Achse die eigene Torlinie entlang. Die Orientierung wird in einem Winkel zur positiven X-Achse angegeben.

Das auf dem Bildschirm dargestellte Feld (im weiteren Verlauf Spielfeld genannt) hat je nach Einstellung eine von vier verschiedenen möglichen Lagen. Hierzu kommt noch, daß zwischen dem Spielfeld und dem Fenster ein Rand ist und auf dem Bildschirm die Y-Achse nach unten zeigt. Aus diesen Umständen benötigen wir eine Transformation von den Daten die auf dem Realfeld basieren zu unserer Spielfeldbasis. Es handelt sich im Prinzip um eine affine Abbildung mit Verschiebung. Diese könnte man in einer dreidimensionalen Matrix problemlos zusammenfassen, jedoch ist es in diesem

Fall einfacher und schneller, eine zweidimensionale Matrix mit einem Verschiebevektor aufzubauen. In Abbildung 6 auf Seite 12 wird die Transformation für die vier Möglichkeiten grafisch aufgezeigt und ist mit den entsprechenden Matrizen und Verschiebevektoren versehen. Auch der Orientierungswinkel muß für die unterschiedlichen Einstellungen angepaßt werden.



**ABBILDUNG 6. 4 mögliche Spielrichtungen**

Der Einfachheit halber wird hier getrennt in eine Skalierung, die wir ja bereits aus Kapitel 2 kennen und der Transformation, die anschließend ausgeführt wird und welche die Größe der Objekte beibehält. Diese Berechnungen werden in den Funktionen *getSpielerPosition*, *getBallPosition* und *ballFusion* getätigt, während die Transformationsmatrix und die Skalierung bereits bei der Initialisierung in Abhängigkeit der Einstellung berechnet werden kann.

**ABBILDUNG 7. Transformationsgleichungen**