Es handelt sich hier nur um einen Vorschlag für die Struktur des Dokuments. Das Ding erhebt keinen Anspruch auf Vollständigkeit, sondern ist das Ergebnis von maximal 15 Minuten „Fuck, wir müssen den Shit fertig machen, ich fang mal an lol“. Bevor wir dann wirklich anfangen aufzuteilen, wer was schreibt (abseits der offensichtlichen Stellen), sollten wir die Struktur des Dokuments **zusammen** durchgehen und finalisieren. Btw, ich hab keinen Plan von Latex, sollte ich also gegen irgendwelche Konventionen verstoßen haben würde es mich nicht wundern... Hoffe aber es passt einigermaßen. Manu

Robotik Dokumentation - Team Robofreunde n.V.

Lohr, Schramm, Stumpf, Weber, Wurth

19. Juli 2015

Inhaltsverzeichnis

1 Einleitung **3**

2 Anforderungen **4**

3 Zeitplan **4**

4 RoboCup - Installation **4**

5 Framework **4**
5.1 Entscheidung für Magma 4
5.2 Überblick Magma 4

6 KI-Konzept **4**
6.1 Anforderungen 5
6.2 Komponenten 5
6.2.1 Elementare Bewegungen 5
6.2.2 Rollen 6
6.2.3 Strategien 6
6.2.4 Strategie-Entscheider 7
6.3 Gesamtübersicht 9

7 Elementare Bewegungen **9**
7.1 Anforderungen 9
7.2 Laufen 9
7.3 Dribbeln 9
7.4 Schießen 9
7.4.1 Behaviors 10
7.4.2 Evaluierung, Metrik Bots 10

8 Rollen **10**
8.1 Standardrollen: Torwart und Stürmer 10
8.1.1 Konzept 10
8.1.2 Standardrolle: Torwart 13
8.1.3 Standardrolle: Stürmer 16
8.2 Standardrolle: Verteidiger 16
8.3 Standardrolle: Mittelfeld 16
8.4 Spezialrollen: Standardsituationen 16

9 Grafisches Debugging **16**

10 Strategien **17**

11 Strategie-Entscheider **17**

12 Zusammenfassung und Ausblick **17**

13 Verzeichnisse **17**

1 Einleitung

Verfasser: Stumpf

Die Lehrveranstaltung Robotik im SS-2015 hatte die Programmierung des NAO – eines humanoiden Roboters von Aldebaran – als übergreifendes Thema. Diesem sollten für Fussball notwendige Bewegungsabläufe und Taktiken beigebracht werden. Aufgrund des großen Andrangs könnten sich nicht alle Teilnehmer mit dem einzigen zur Verfügung stehenden NAO Modell beschäftigen. Die Gruppe wurde also in ein Hardware und 2 Simulationsteams aufgeteilt.

Die vorgegebene Simulationumgebung war die auf Simspark basierende RoboCup 3D-Soccer Simulation League.
Das vorgegebene Ziel unserer beiden Simulationsteams war es, je eine eigene Mannschaft aufs Feld zu bringen und diese am Ende gegeneinander antreten zu lassen.

RoboCup ist eine Fussball-Liga humanoider Roboter. Die Teams mehrerer Länder treten gegeneinander an um ein bestes Team zu ermitteln.

In der RoboCup Soccer Simulation League werden die Spiele simuliert. Die Roboter werden durch virtuelle Instanzen ersetzt, die über definierte Schnittstellen auf den Server verbunden und gesteuert werden können.

Es gibt eine 2D und eine 3D Version. Für uns war insbesondere die 3D Version interessant, weil NAO hier das aktuelle Robotermodell ist. Anforderungen und Regeln der

3D Version sind vergleichbar mit dem echten RoboCup.

Da die Schnittstellen zum Server auf sehr niedriger Ebene definiert sind - Bewegung von Gelenken - ist es für den Zeitraum der Lehrveranstaltung zu aufwändig, einen virtuellen Roboter komplett selbstständig zu entwickeln. Es gibt allerdings einige Frameworks, auf denen die eigene Arbeit aufgebaut werden kann.

2 Anforderungen

-> **TODO: Freddy???** *Was waren die allgemeinen Anforderungen?*

3 Zeitplan

-> **TODO: Stumpfi** *Was wurde wann gemacht?*

4 RoboCup - Installation

5 Framework

-> **TODO: Stumpfi** *Ein paar einleitende Sätze.*

5.1 Entscheidung für Magma

-> **TODO: Stumpfi** *Warum Magma?*

5.2 Überblick Magma

-> **TODO: Stumpfi** *Alles, was wir so über das Framework rausgefunden haben und erwähnenswert ist.*

6 KI-Konzept

Für die KI wurde ein Konzept entwickelt, dass sich einfach in die bereits existierende Architektur des verwendeten Frameworks integrieren ließ. Ziel war es, eine möglichst flexible und gleichzeitig einfache Architektur zu entwerfen. Das Ergebnis ist eine Schichten-Architektur, deren Teilkomponenten einfach verändert oder ausgetauscht werden können. Nebeneffekte auf andere Module sind durch die Architektur weitgehend ausgeschlossen. So war es beispielsweise Problemlos möglich, verschiedene Herangehensweisen für die Implementierung von Rollen in das Gesamtsystem zu integrieren. Verschiedene Teammitglieder konnten also ihre selbst entwickelten Rollen schnell und einfach in das Gesamtsystem integrieren, ohne andere Komponenten anpassen zu müssen.

6.1 Anforderungen

Das gewählte Framework und die Tatsache, dass eine Steuerung für eigenständige Roboter entwickelt werden musste, stellt einige Anforderungen an die zu entwickelnde künstliche Intelligenz.

Jeder Roboter ist als unabhängiges Individuum zu verstehen. Das hat die direkte Folge, dass Entscheidungen nicht von einer übergeordneten Instanz getroffen werden können; Jeder Roboter muss aufgrund der ihm zur Verfügung stehenden Informationen seine eigenen Entscheidungen treffen. Das Fehlen einer zentralen Komponente erschwert es erheblich, ein aufeinander abgestimmtes Team-Spiel zu entwickeln. Dennoch gibt es für dieses Problem gute Ansätze. Die Anforderung, die sich zusammenfassend daraus ableiten lässt, ist die Notwendigkeit einer dezentralen Lösung für die KI.

Eine weitere Anforderung ergibt sich aus der Wahl des Frameworks. Das Framework arbeitet weitgehend mit Polling, also einem kontinuierlichen Abfragen der nächsten Aktion. Es ist daher naheliegend, auch für die hier zu entwickelnde KI einen Ansatz mit Polling zu zu verfolgen.

Neben den mehr oder weniger durch die äußeren Umstände vorgegebenen Anforderungen haben wir noch weitere zwei Anforderungen definiert. Roboter sollen ihre Rolle dynamisch wechseln, außerdem soll es zusätzlich die Möglichkeit geben, die aktuell verfolgte Spiel-Strategie nach Bedarf zu ändern.

6.2 Komponenten

Das KI-Grundkonzept besteht aus insgesamt 4 Schichten. Jede Schicht kennt ausschließlich die darunterliegende Schicht. Die unterste Ebene definiert Grundlegende Bewegungen eines Roboters. Direkt darüber befinden sich Rollen. Rollen sind zum Beispiel „Stürmer“oder „Torwart“, aber auch Spezialrollen wie „Anstoß“oder „Freistoß“. Über den Rollen befindet sich die Strategie-Ebene. Auf höchster Ebene befindet sich der Strategie-Entscheider.

6.2.1 Elementare Bewegungen

Elementare Bewegungen sind die Grundlage für alle höheren Schichten. Sie sind die unmittelbare Voraussetzung die darüber liegenden Rollen, weil sie von diesen direkt benutzt werden. Elementare Bewegungen sind zum Beispiel: „Laufen“, „Aufstehen“, „Schießen“, oder „Dribbeln“. Abgesehen vom „Dribbeln“waren alle Bewegungen bereits mehr oder weniger funktionsfähig im Framework vorhanden. Dennoch musste an einigen Stellen optimiert und angepasst werden, siehe Kapitel **??**.

6.2.2 Rollen

Rollen definieren das Grundlegende Verhalten eines Roboters. Ein Verteidiger verfolgt eine andere Spielweise als ein Torwart. Alle unsere Rollen sind Statemachines. Eine wichtige Frage ist, wo und wie die KI für Standard-Situationen wie zum Beispiel den „Anstoß“in der Architektur verankert werden soll. Eine naheliegende Lösung ist die Definition von Spezialrollen, die genau diese Standard-Situationen behandeln. Die Alternative wäre eine Erweiterung aller Standard-Rollen um die Algorithmik der Standard-Situationen. Einen wirklichen Mehrwert hat man dadurch allerdings nicht, denn der Algorithmus um einen Anstoß zu machen benötigt keine Routinen eines Stürmers. Umgekehrt gilt das Selbe. Die Entscheidung fiel also auf die Definition von Spezialrollen, die genau diese Standard-Situationen umsetzen. Ein Spieler, der normalerweise zum Beispiel Stürmer ist, wechselt bei Bedarf einfach seine Rolle in „Anstoß“ und im Anschluss wieder zurück.

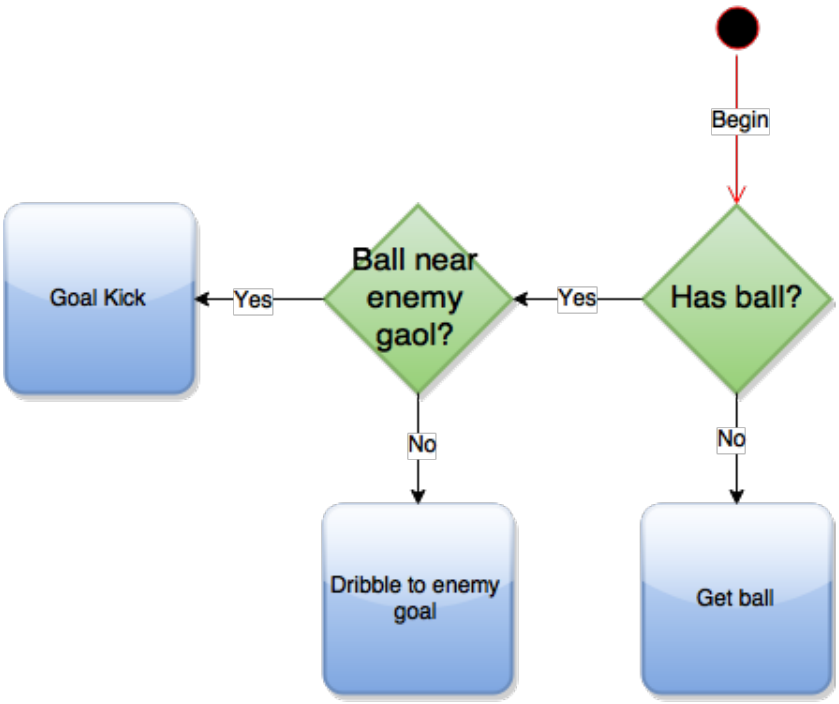

Abbildung 1: Beispiel Rolle: Einfacher Torwart

Abbildung 1 zeigt Beispielhaft den Entscheidungsbaum eines einfachen Torwarts.

6.2.3 Strategien

Rollen sollen nicht selbst entscheiden, ob und in welche andere Rolle gewechselt wird. Das widerspräche der Schichtenarchitektur. Deshalb wurde die darüber liegende Strategieschicht eingeführt. Die Aufgabe dieser Schicht ist es, Rollen dynamisch zuzuordnen. Daher ist die Bezeichnung „Strategie“auch treffend; Eine Strategie steuert unter Anderem die Gesamtaufstellung des Teams. Damit lässt sich die allgemeine Spielweise des

gesamten Teams steuern, mehr Verteidiger bedeutet eine defensive, mehr Angreifer eine aggressive Strategie. Eine Strategie benötigt einen Pool an Rollen, die sie „verteilen“darf. Dieser Pool definiert die grundlegende Spielweise eines Teams. Aufgabe jeder Strategie ist es dann, Rollen dynamisch nach Bedarf zu verteilen. Darunter fällt auch die Vergabe der Spezialrollen.

6.2.4 Strategie-Entscheider

In der Strategieebene können verschiedene Strategien für verschiedene Gesamtaufstellungen des Teams definiert werden. Im Laufe eines Spiels kann es sinnvoll sein, die Aufstellung an die Gegebenheiten und Anforderungen an das Spiel anzupassen. Dafür Wurde auf höchster Ebene ein Entscheidungsmechanismus etabliert. Die Aufgabe dieser Schicht ist es, je nach Bedarf eine passende Strategie auszuwählen. Dafür gibt es verschiedene Ansätze, zum Beispiel könnte eine passende Strategie je nach aktuellem Spielstand ausgesucht werden. Beispiel:

- Mannschaft liegt vorne: Defensive Strategie
- Unentschieden: Aggresive Strategie
- Mannschaft liegt hinten: Aggressive Strategie
- Mannschaft liegt hinten und Zeit läuft bald aus: Risiko Strategie

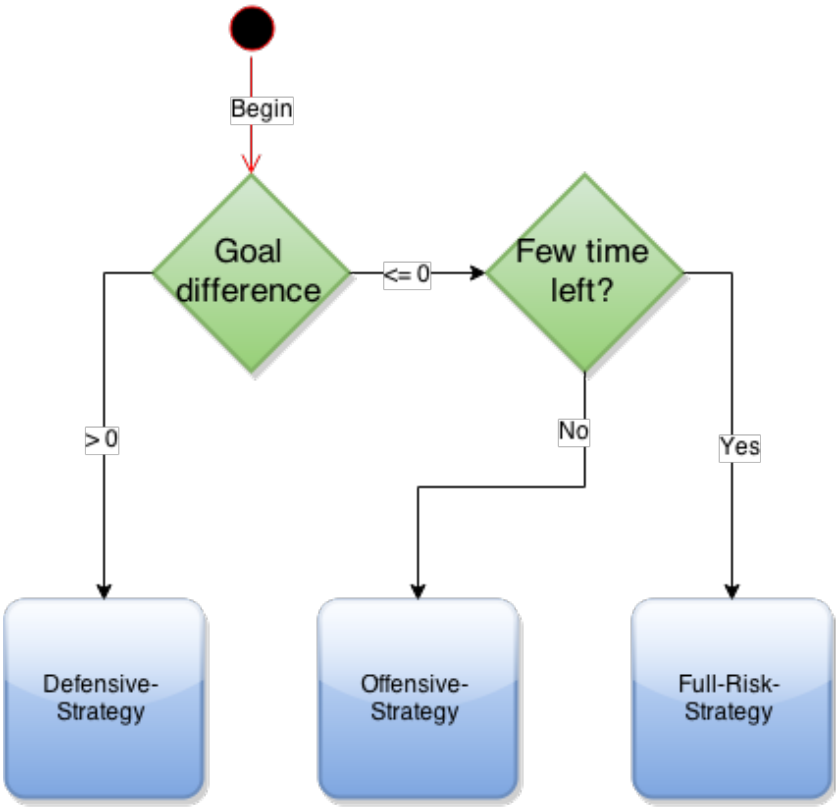

Abbildung 2: Einfacher Strategie Entscheider

Abbildung 2 zeigt den Enscheidungsbaum eines sehr einfachen Entscheidungsmeachanismus für Strategien.

6.3 Gesamtübersicht

Abbildung 3: Gesamtkonzept KI

Abbildung 3 zeigt das Gesamtkonzept mit allen 4 Ebenen. Ganz unten befinden sich die elementaren Bewegungen. Rollen werden auf der Schicht darüber definiert. Sie greifen als einzige Schicht auf elementare Bewegungen direkt zu. Über den Rollen befindet sich die Strategie-Ebene. Jede Strategie hält einen Pool an Rollen vor und Steuert damit die konkrete Aufstellung auf dem Spielfeld. Auf höchster Ebene befindet sich der Entscheidungsmechanismus, der Situationsabhängig eine passende Strategie aktiviert.

7 Elementare Bewegungen

7.1 Anforderungen

-> TODO: Stumpfi

7.2 Laufen

-> TODO: Felix

7.3 Dribbeln

-> TODO: Felix

7.4 Schießen

-> TODO: Stumpfi

7.4.1 Behaviors

-> TODO: Stumpfi

7.4.2 Evaluierung, Metrik Bots

-> TODO: Stumpfi

8 Rollen

Rollen legen fest, welches übergeordnete Ziel ein Roboter zu einem bestimmten Zeitpunkt verfolgt aber vor Allem, wie dieses Ziel erreicht werden soll. Ziele wie Aufstehen nach einem Sturz, oder die eigene Position vor dem Ball korrigieren, sind untergeordnete Ziele die vor allem in den elementaren Bewegungen verfolgt werden. Ein Stürmer, dessen finales Ziel es ist, ein Tor zu schießen, muss natürlich einige Schritte Vorarbeit zum Erreichen dieses Ziels leisten. Zum Beispiel muss er zuerst den Ball erobern. Die Algorithmik, die zum Erreichen eines solchen Ziels erforderlich ist, findet ihren Platz in den Rollen. Wie in Kapitel 6.2.2 bereits erwähnt, werden Rollen in zwei Kategorien unterteilt, nämlich Standard-Rollen und Spezial-Rollen.
Unter Standardrollen fallen alle normalen Spielertypen wie Stürmer, Verteidiger oder Torwart. In diesem Projekt wurden insgesamt 4 Standardrollen implementiert, deren genauere Beschreibung in diesem Kapitel Platz findet. Die Implementierung der Standardrollen wurde von verschiedenen Team-Mitgliedern durchgeführt. Stürmer und Torwart wurden von einer Person, Verteidiger und Mittelfeldspieler auch von jeweils einer Person entwickelt. Die Herangehensweisen unterscheiden sich deshalb zwischen den Rollen stark.

Spezialrollen wiederum kümmern sich um spezielle Situationen wie zum Beispiel den Anstoß oder den Kick-In.

8.1 Standardrollen: Torwart und Stürmer

Die Rolle Torwart und die Rolle Stürmer wurden von einer Person entwickelt. Sie teilen sich deshalb ein gemeinsames Grundkonzept.

8.1.1 Konzept

Wie alle anderen Rollen auch, sind die Rollen für Torwart und Stürmer State-Machines. Jede State-Machine beinhaltet mehrere States. Jeder State hat eine ganz bestimmte Aufgabe, zum Beispiel: Zu einer bestimmten Position laufen. Im Kern befindet sich innerhalb eines States nur die Parametrisierung einer konkreten elementaren Bewegung. Diese Bewegung kann dann mittels Poll von außen abgefragt werden.

Der erste Ansatz zur Modellierung der State-Machine war klassisch; Jeder State hatte eine bestimmte Menge an Folgestates und konnte demzufolge ausschließlich in diese

States übergehen. Nach und nach offenbarte sich aber die Problematik dieser Designentscheidung. Bei laufenden Tests viel auf, dass bestimmte Transitionen von einem in einen anderen State nicht bedacht wurden. Folglich musste die State-Machine überarbeitet werden. Das führte recht schnell zu einer anderen Herangehensweise; Jeder State hat jeden anderen State der State-Machine als potenziellen Nachfolger. Ein enormer Flexibilitätszuwachs zur ersten Variante. Jetzt konnte der Entscheidungsmechanismus zur Findung des nächsten States zentral etabliert werden.

Dazu wurde für beide Rollen jeweils eine Basisklasse angelegt. Diese Klasse kümmert sich um das Vorhalten aller States der StateMachine. Außerdem findet in ihr die Methode Platz, die einen Folgestate auswählt. Diese Methode ist das Herzstück der Rolle, sie definiert was wann gemacht werden soll.

```
1  public abstract class BaseAttackerCenter extends BaseState {
2      private DribbleState dribbleState;
3      private GoalKickState goalKickState;
4      private GoToBallState goToBallState;
5      private GoToPositionState goToPositionState;
6      private PassState passState;
7
8      public void bootstrap(DribbleState dribbleState,
9                            GoalKickState goalKickState,
10                           GoToBallState goToBallState,
11                           GoToPositionState goToPositionState,
12                           PassState passState)
13     {
14         this.dribbleState = dribbleState;
15         this.goalKickState = goalKickState;
16         this.goToBallState = goToBallState;
17         this.goToPositionState = goToPositionState;
18         this.passState = passState;
19     }
20
21     public BaseState decideNextState() {
22         ...
23     }
24 }
```

Listing 1: Beispiel Basisklasse Stürmer

1 Zeigt den Aufbau einer Basisklasse für States am Beispiel des Stürmers. Sie bekommt in der Methode bootstrap() von Außen alle States und hält sie dann vor. Außerdem befindet sich ab Zeile 21 der die Methode decideNextState().
Jeder Konkrete State leitet von diesem Basis-State ab und erbst somit alle States der

Statemachine sowie die Methode decideNextState().

```
1  public class GoToBallState extends BaseAttackerCenter {
2      IBehavior moveToBall;
3
4      @Override
5      public void init(Player player) {
6          super.init(player);
7          this.moveToBall = null;
8      }
9
10     @Override
11     public BotState update() {
12         BaseState nextState = decideNextState();
13
14         if (nextState == this) {
15             RunToPosition moveToBall = (RunToPosition)
                   getPlayer().getBehavior(IBehaviorConstants.
                   RUN_TO_POSITION);
16             moveToBall.setPosition(calcProperRunToBallPose(),
                   0, 0, 75, false);
17             this.moveToBall = moveToBall;
18             return null;
19         }
20         return nextState;
21     }
22
23     @Override
24     public IBehavior getBehavior() {
25         return this.moveToBall;
26     }
27 }
```

Listing 2: Beispiel eines konkreten States: GoToBallState

Listing 2 zeigt Beispielhaft den State, der für das Laufen zum Ball verantwortlich ist. Er hält eine sogenannte IBehavior vor, die am Ende für das Ausführen der Bewegung benötigt wird. Innerhalb des States findet also nicht die Ausführung der Bewegung statt, sondern nur deren Parametrisierung. In der update()-Methode passiert in Zeile 12 zuerst die Abfrage des für die aktuelle Situation besten States. Dazu wird die geerbte Methode decideNextState() aufgerufen. Nur wenn der zurückgegebene State er selbst ist, soll er seinen Code ausführen und die IBehavior bearbeiten (Zeilen 14 - 18). Wurde ein anderer State zurückgegeben, wird kein eigener Code ausgeführt und der neue State zurückgegeben (Zeile 20). Damit die IBehavior von außen abgerufen werden kann, wurde

hierfür ein Getter erzeugt (Zeilen 23 - 26).

8.1.2 Standardrolle: Torwart

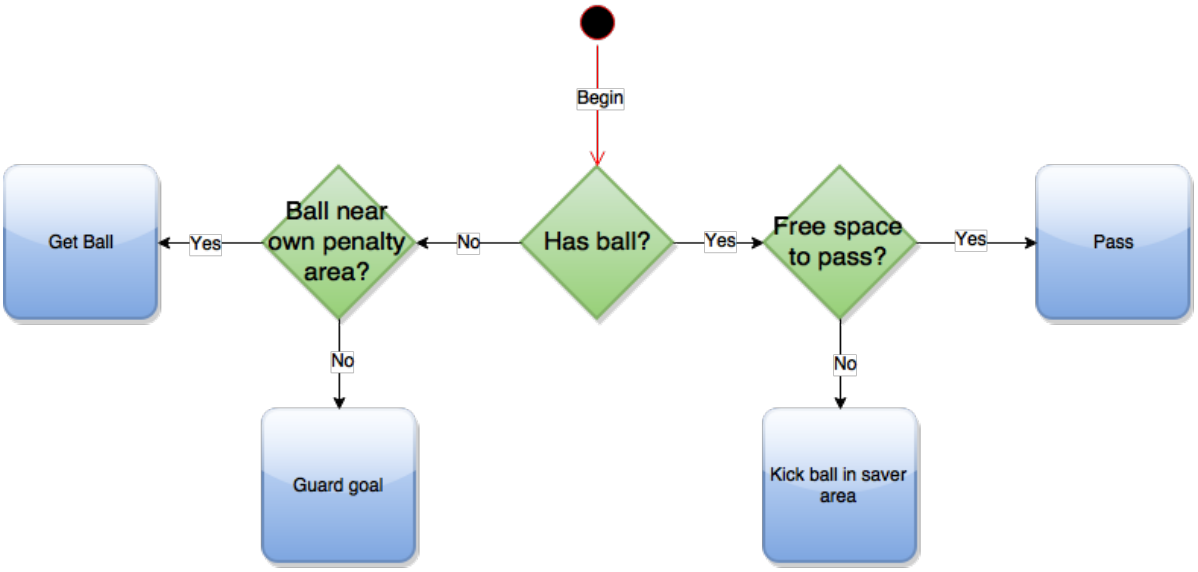

Abbildung 4: Torwart Entscheidungsbaum

Abbildung 4 zeigt den Entscheidungsbaum des Torwarts. Die Entscheidungsbäume von Torwart und Stürmer finden ihre direkte Umsetzung in der decideNextState() des Basis-States. Es war also sehr sinnvoll vor der Implementierung derartige Entscheidungsbäume zu entwerfen.

Die Statemachine des Torwarts besteht aus insgesamt 4 States. Die erste Überprüfung ist, ob der Spieler den Ball besitzt. Besitzt er den Ball, will er ihn schnell wieder loswerden, schließlich ist er ein Torwart. Dazu Sucht er im eigenen Team passende Passpartner. Ein passender Passpartner steht in einer passenden Entfernung und wird nicht von einem gegnerischen Spieler geblockt. Ist ein passender Spieler gefunden wird der Pass ausgeführt. Findet sich kein Spieler, soll der Ball in Richtung der Mittellinie geschossen werden, also möglichst weit weg von eigenen Tor.

Besitzt der Torwart den Ball nicht, kommt es darauf an wie weit der Ball weg ist. Ist der Ball ein gutes Stück entfernt, soll in den State GuardGoalState gewechselt werden. Was dieser State genau macht wird später beschrieben. Kommt der Ball dem eigenen Tor zu Nahe, soll der Torwart reagieren und auf den Ball zulaufen um ihn zu erobern.

Der GuardGoalState

Hat der Ball eine relativ große Entfernung zum Tor, soll sich der Torwart natürlich nicht vom eigenen Tor wegbewegen. Vielmehr sollte er eine geeignete Position vor dem

Tor einnehmen. Ziel ist es, eine möglichst große Fläche des Tors zu „verdecken“, um Distanzschüsse zu verhindern oder zumindest zu erschweren. Dafür ist der GuardGoalState zuständig.

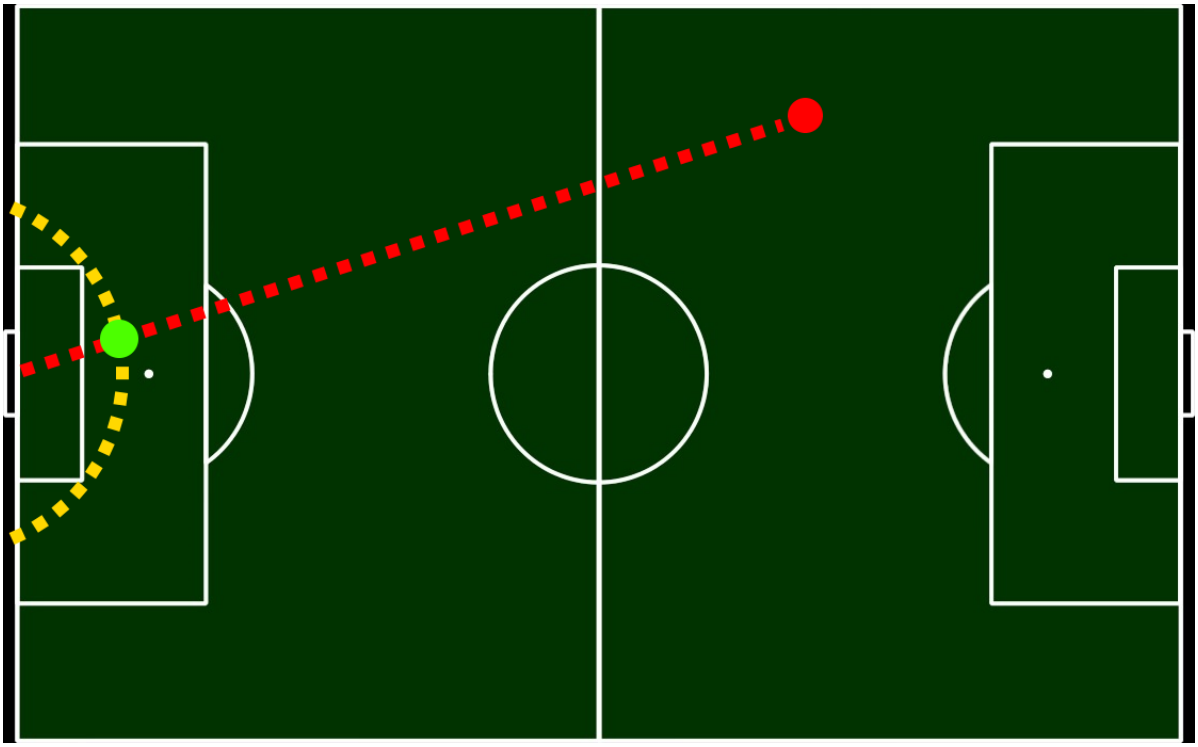

Abbildung 5: Veranschaulichung des GuardGoalStates

Abbildung 5 veranschaulicht die Berechnung der optimalen Position vor dem Tor anhand einer Grafik. Der gelbe Halbkreis definiert alle möglichen Positionen vor dem Tor, die der Torwart im GuardGoalState einnehmen kann. Die genaue Position hängt von der aktuellen Position des Balls ab. Dazu wird eine Gerade vom Ball in das Eigene Tor gelegt, der Schnittpunkt mit dem gelben Halbkreis entspricht der optimalen Position.

```
1 public Vector3D getGoodGoalieDefendPosition() {
2     Vector3D ballPosition = thoughtModel.getWorldModel().
          getBall().getPosition();
3     Vector3D goalPosition = thoughtModel.getWorldModel().
          getOwnGoalPosition();
4     Vector3D direction = ballPosition.subtract(goalPosition);
5     direction = direction.normalize().scalarMultiply(2.0f);
6     return goalPosition.add(direction);
7 }
```

Listing 3: Umsetzung des GuardGoalStates

Listing 3 zeigt die Berechnung der korrekten Position im Code. Notwendig sind jeweils die Position des Ball und des eigenen Tors (Zeilen 2 und 3). Dann wird ein Richtungsvektor berechnet, der vom Tor Richtung Ball zeigt (Zeile 4). Dieser wird zuerst normiert und dann auf die Länge 2 skaliert. Der Abstand 2 vom Tor hat sich als guter Wert herausgestellt (Zeile 5). Die gesuchte Position errechnet sich durch die Addition des Richtungsvektors auf die Torkoordinate (Zeile 6).

Energiesparmaßnahme
Der Torwart wird die meiste Zeit innerhalb des GuardGoalState bleiben. Bei jedem Poll wird eine neue Position anhand der Ball und Torposition berechnet, zu der er dann hinläuft. Bei Ball und Torkoordinate handelt es sich aber um circa Angaben, schließlich wird das Worldmodel anhand der Sensoren des Roboters kontinuierlich korrigiert. Dabei kommt es zu minimalen Schwankungen, die zu einer neu berechneten Laufposition führen. Somit kommt der Roboter quasi nie zum Stillstand, auch wenn er seiner berechneten Laufposition extrem nahe ist. Damit der Roboter stehen bleibt, wenn er hinreichend nahe an seiner Zielposition steht, wurde eine Energiesparmaßnahme etabliert.

```
1 if (nextState == this && (targetDistance > 0.2 || !
    angleCloseEnough(angleToBall))) {
2     ...
3 }
```

Listing 4: Energiesparmaßnahme Torwart

Listing 4 zeigt die Bedingungen für eine erneute Bewegnung. Nur, wenn die Zieldistanz einen hinreichend kleinen Wert überschreitet, oder der Winkel des Roboters zum Ball nicht gut genug ist, soll zur neuen Position gelaufen werden. Ursprünglich wurde nur die Distanz zum Ziel herangezogen, mit der Folge, dass der Roboter teilweise quer zum Ball stehen blieb. Damit verdeckt er aber weniger Fläche vom Tor, also wurde noch ein optimaler Winkel zum Ball miteinbezogen.

8.1.3 Standardrolle: Stürmer

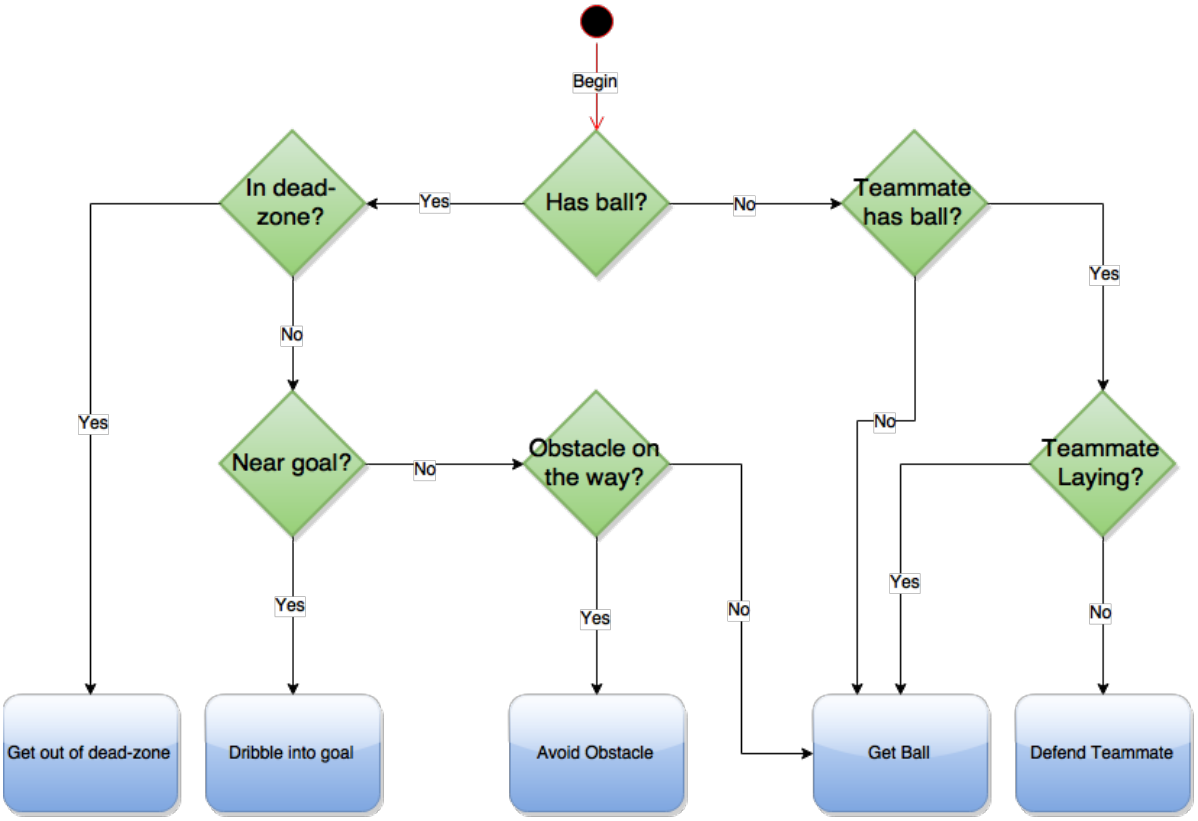

Abbildung 6: Stürmer Entscheidungsbaum

8.2 Standardrolle: Verteidiger

8.3 Standardrolle: Mittelfeld

8.4 Spezialrollen: Standardsituationen

-> **TODO: Stumpfi**

9 Grafisches Debugging

-> **TODO: Felix**

10 Strategien

11 Strategie-Entscheider

12 Zusammenfassung und Ausblick

Ergebnisse und Vermächtnis...

13 Verzeichnisse

Abbildungsverzeichnis

1 Beispiel Rolle: Einfacher Torwart 6
2 Einfacher Strategie Entscheider 8
3 Gesamtkonzept KI 9
4 Torwart Entscheidungsbaum 13
5 Veranschaulichung des GuardGoalStates 14
6 Stürmer Entscheidungsbaum 16

Listings

1 Beispiel Basisklasse Stürmer 11
2 Beispiel eines konkreten States: GoToBallState 12
3 Umsetzung des GuardGoalStates 14
4 Energiesparmaßnahme Torwart 15