

Konzept Robocup Client

Version 1.1
9. Mai 2002

Teammitglieder:

*Danny Franzreb
Jürgen Gall
Michael Geisser
Jan Kästle
Dennis Landmann
Christian Pantke
Felix von Reischach*

Inhaltsverzeichnis

0. Modifikationen	3
1. Geplante Architektur:	3
Communication:	3
PlayerInfo:	3
Player:	3
Visualization:	4
2. Gruppeneinteilung:	4
3. Zeitplan:	4
Entwurfsphase	4
Implementierungsphase	4
Test und Dokumentationsphase	4

0. Modifikationen

Im Verlauf des Praktikums werden möglicherweise Veränderungen an diesem Dokument erforderlich werden. Die Modifikationen werden in diesem Abschnitt zusammengefasst.

- *Version 1.0 Initialversion*
- *Version 1.1*
 - Inhaltsverzeichnis hinzugefügt
 - Abschnitt Gruppeneinteilung hinzugefügt
 - Abschnitt Zeitplan hinzugefügt

1. Geplante Architektur:

- eine Klasse "Communication", die die zwei Aufgaben hat, dem Spieler einerseits die Möglichkeit zu geben die Serverparameter zu empfangen, sowie per UDP dem Server zu antworten.
- eine Klasse "PlayerInfo", die als Schnittstelle für Communication, Player, Visualization dient. Sie greift des weiteren auf mehrere Nebenklassen für spezielle Datentypen (Ball, Score, OtherPlayers, Coordinate, Field, Referee) zurück.
- eine Klasse "Player" mit den wesentlichen Eigenschaften eines Spielers. Sie und ihre Subklassen fungieren als "Gehirn" des jeweiligen Spielers.
- eine Klasse "Visualization"

Communication:

Empfang:

- Generierung eines Sockets
- Binden des Sockets an den, dem Spieler entsprechenden, Port
- Empfang UDP Packages des Servers
- Auslesen der darin enthaltenen Informationen
- Weitergabe der Informationen an PlayerInfo

Senden:

- Aufbereitung der zu sendenden Informationen
- Synchronisation mit dem Server
- Senden der spieterspezifischen Kommandos

PlayerInfo:

- Datenspeicher für alle vom Server übermittelten Spielerparameter, Startposition, Spielrichtung
- unveränderlicher Datenspeicher für alle Flags, Lines, Goals
- Datenspeicher für alle empfangenen "Say"-Aufrufe (Referee, Player, Devil)
- Datenspeicher für Ball, Spielstand, Koordinaten der anderen Spieler, eventuell Formation
- Umrechnung der relativen Daten in absolute Daten (Coordinate)
- externer Zugriff auf Daten mittels set- und get- Funktionen
- Löschen alter Daten

Player:

- von Player erben Torwart und Feldspieler. Von Feldspieler erbt Abwehrspieler und Angreifer (eventuell noch Mittelfeldspieler).
- Geeignete Klassen/Methoden zur Steuerung des Spielerverhaltens auf dem Spielfeld. Diese Klassen/Methoden geben die Befehle weiter an "Communication".
- Daten über Ball, andere Spieler etc. erhält sie von der Klasse "PlayerInfo".
- Entscheidungen werden aufgrund von verschiedenen Spielzuständen getroffen.

Visualization:

- Visualisierung des Spielfeldes mit den Daten aus "PlayerInfo"
- Stand-Alone Anwendung (JFrame)
- Kann zusätzlich zum SoccerClient aufgerufen werden
- Wechsel zwischen einzelnen Spielern über Buttons
- Regelmäßige Aktualisierung

2. Gruppeneinteilung:

Die Gruppenaufteilung erfolgt bezüglich der Hauptklassen des Programms. Die Gruppenmitglieder sind sowohl für die Implementierung der Klasse und der zugehörigen Methoden, als auch für die mit der Klasse zusammenhängenden Planungs- und Dokumentationsschritte (z.B. UML Diagramme, Benutzerhandbuch), verantwortlich.

- **Communication:** *Danny Franzreb, Dennis Landmann*
- **PlayerInfo und Visualization:** *Jürgen Gall, Felix von Reischach*
- **Player:** *Michael Geisser, Jan Kästle, Christian Pantke*

3. Zeitplan:

Entwurfsphase

- Bis **14.5.** Erstellen einer Schnittstellenbeschreibung für **Communication**
- Bis **16.5.** Erstellen einer Schnittstellenbeschreibung für **PlayerInfo**
- Bis **18.5.** Erstellen der Schnittstellenbeschreibung für **Player** und **Visualization**
- Bis **20.5.** Zusammenfassen der einzelnen Schnittstellenbeschreibungen zu einem ersten *UML Diagramm*, erstellen von *Aktivitäts- und Zustandsdiagrammen*

Implementierungsphase

- Bis **3.6.** Implementierung von Basisfunktionen (bei **Communication** und **PlayerInfo**) bzw. einer Basisstruktur (bei **Player** und **Visualization**) des Programms
- Bis **24.6.** Implementierung erweiterter Funktionalität (z.B. Weltmodell, Strategie, Visualisierung,...)

Test und Dokumentationsphase

- Bis **8.7.**
 - Testen der Programmstruktur auf Robustheit und auf logische Fehler in der Strategie- und Daten-Komponente. Testen der Korrektheit des Visualisierungsmodells. Ausbesserung dieser Fehler und Verfeinerung der Strategie → fertige Implementierung
 - überarbeiten der Klassen-, Aktivitäts- und Zustandsdiagramme
 - erstellen eines Benutzerhandbuchs