

Lab-2006

Mobile Hat trick

Thibaut Britz, Andri Toggeburger, Roger Seidel,
Michael Lorenzi

Lab-2006

Summer Term 2006

Tutor: Michael Kuhn

Supervisor: Prof. Roger Wattenhofer

07.07.2006

Inhaltsverzeichnis

1	Aufgabenstellung	4
2	Spiel	5
2.1	Spielidee	5
2.2	Spielablauf	5
2.3	Requirements	6
3	GUI	8
3.1	GUI Programmierung mit MIDP 2.0	8
3.2	Entscheide, die zum Gebrauch der Low Level API führten	9
3.3	Probleme mit der Low Level API	9
3.4	Entwickelte GUI Elemente	10
3.5	Zentraler Ereignis Dispatcher	11
4	Simulation	13
4.1	Teamgenerierung	13
4.1.1	Spieler-Attribute	14
4.2	Spielsimulation	15
4.2.1	Chancenberechnung	15
4.2.2	Chancensimulation	15
4.2.3	Verletzungen	16
4.3	Verbesserungen	16
4.3.1	Nach einem Spiel	16
4.3.2	Nach einer Saison	18
4.4	Ranking	19
4.5	Ausblick	20
4.5.1	Geld	20
4.5.2	Transfermarkt	20
5	Persistente Speicherung	21
5.1	RMS vs. File Connection API	21
5.2	Architektur	21
5.3	Future Work	23

6	Kommunikation	24
6.1	Mobiltelefon zu Mobiltelefon	24
6.1.1	Architektur	24
6.1.2	Future Work	25
6.2	Mobiltelefon zu Server	26
6.2.1	Future Work	26
7	Security	27
7.1	Angreifer-Modell	27
7.2	Implementation	27
7.2.1	PKI	27
7.3	Protokolle	28
7.3.1	Nomenklatur	28
7.3.2	handshake()	28
7.3.3	playgame()	28
7.3.4	playnext()	29
7.4	Registrierung	29
7.5	Synchronisation	31
7.6	Attacken	31
8	Server	34
8.1	Zusammenfassung	34
8.2	Modell der schummelnden Spieler	35
8.3	Registration	36
8.4	Synchronisation	36
8.5	Verifikation	36
8.6	Future work	40
8.7	Banlist	41
8.8	Server Rangliste	41
8.8.1	Future work	41
8.9	Datenbank Tabellen	41
8.10	Test Client	41
9	Installation	43
9.1	Builden des Spiels	43
9.1.1	Obfuscation	43
9.1.2	Permissions	43
9.1.3	Signatur	44
9.2	Installation auf den Handys	44
9.3	Builden des Servers	44
9.4	Installation des Servers	44

10	Erfahrungsberichte	46
10.1	Andri Toggenburger	46
10.2	Thibaut Britz	49
10.2.1	Aufgabe	49
10.2.2	Probleme und Erfahrungen	50
10.2.3	Fazit	51
10.3	Michael Lorenzi	52
10.3.1	Hintergrund und Motivation	52
10.3.2	Aufgaben	52
10.3.3	Probleme und Erfahrungen	53
10.3.4	Fazit	54
10.4	Roger Seidel	55
10.4.1	Fazit	56
11	Appendix	57
11.1	2-Armeen-Problem	57

Kapitel 1

Aufgabenstellung

Unsere Aufgabe als neue Mitarbeiter von DoPE (Doing Private Entertainment AG) war es, ein Multiuser-Spiel für mobile Geräte zu entwerfen. Es wurde vorgeschlagen, ein Fußball-Management-Spiel nach dem Vorbild von Hattrick [3] zu entwickeln. Wir sollten das Konzept von Hattrick auf eine mobile Umgebung erweitern und so deren Erfolg wiederholen.

Jeder Spieler soll ein Team auf seinem Mobiltelefon gespeichert haben, und gegen einen anderen Spieler spielen können, der sich im Bluetooth-Empfangsbereich befindet. Transfers können gemacht werden und Punkte für eine Rangliste (analog ELO oder ATP) werden je nach Ausgang des Spiels vergeben. Bei der Entwicklung des Konzepts soll auf folgende Punkte geachtet werden:

- Wegen der hohen Tarife und der intransparenten Kosten soll das Spiel ohne mobilen Internetzugang spielbar sein.
- Egoistische Spieler, welche auf den eigenen Vorteil aus sind, sollen ehrlichen Spielern den Spielspass nicht verderben können.
- Sadistisch veranlagte Spieler dürfen keine Möglichkeit haben, ehrlichen Spielern Schaden zuzufügen.

Mit einem cleveren Spielkonzept, das in einer ersten Phase des Projekts erarbeitet werden soll, sollen diese Probleme vermieden werden. In der zweiten Phase soll das Konzept dann für reale Geräte mit Java implementiert werden.

Kapitel 2

Spiel

Mobile Hattrick ist ein Peer-to-peer Fussballmanagerspiel für Java-fähige Mobiltelefone, angelehnt an das bekannte Onlinespiel Hattrick [3].

2.1 Spielidee

Über Bluetooth können zwei Spieler mit ihren Teams gegeneinander spielen. Dabei hat jeder Spieler verschiedene Einflussmöglichkeiten auf das Resultat:

- Formationen wählen (3-5-2, 4-4-2, etc.)
- Spieler aufstellen
- Auswechslungen in der Halbzeit vornehmen

Nach dem Spiel werden Punkte für eine globale Rangliste verteilt. Weiteres zur Rangliste ist in Kapitel 3.4 beschrieben. Zudem erhält der Sieger des Spiels eine Anzahl Stärkepunkte, die er auf seine Spieler verteilen kann um diese zu verbessern (Kapitel 3.3).

Nach einigen Partien kann ein Spieler mit dem Mobile-Hattrick-Server in Kontakt treten (siehe Abbildung 2.1). Dieser bescheinigt ihm dann die Korrektheit seiner Spiele und Verbesserungen und weist ihm Punkte im globalen Ranking aller Spieler zu. Ohne Server wäre es nicht möglich, Betrügereien aufzudecken und die Spiele auch im Fehlerfall für beide Parteien korrekt zu werten.

2.2 Spielablauf

Über Bluetooth können zwei Spieler eine Partie gegeneinander spielen. Der Ablauf eines Spiels ist in Abbildung 2.2 dargestellt.

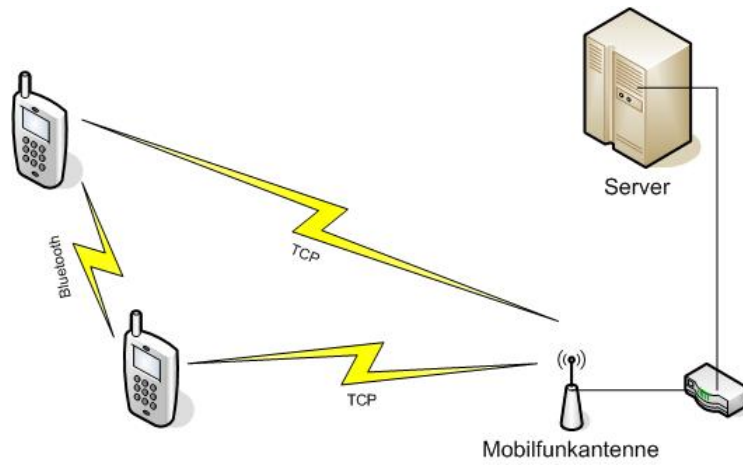


Abbildung 2.1: Spieler und Server

2.3 Requirements

Um Mobile Hat trick zu spielen, muss das Mobiltelefon folgende Features aufweisen:

- MIDP 2.0¹
- CLDC 1.1²
- JSR - 82 (Bluetooth API)
- JSR - 75 (File Connection API)
- Socket Connections

¹J2ME Profil: Mobile Information Device Profile

²J2ME Konfiguration: Connected Limited Device Class

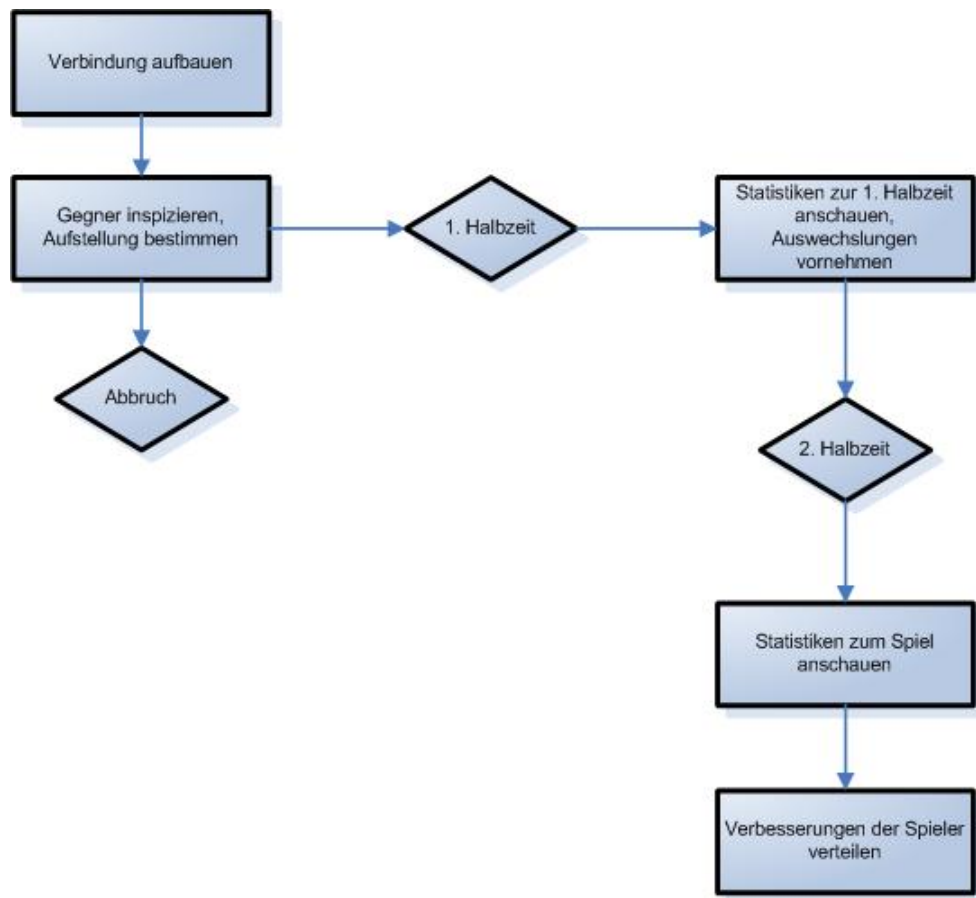


Abbildung 2.2: Ablauf eines Spiels

Kapitel 3

GUI

3.1 GUI Programmierung mit MIDP 2.0

MIDP 2.0 stellt eine GUI Bibliothek ähnlich wie AWT zur Verfügung, nur dass alles stark vereinfacht und an die Bedürfnisse von mobilen Geräten angepasst ist.

Anstatt mit Fenstern arbeitet man mit sogenannten Screens. Ein Screen stellt ähnliche Funktionalität zur Verfügung wie ein Fenster im ursprünglichen Sinn. Jedoch wird immer nur ein Screen aufs mal auf dem Display angezeigt. Man kann andere Screens in den Vordergrund bringen, worauf aber der zuvor angezeigte Screen verschwindet. Man kann einem Screen Menu Einträge hinzufügen, die angezeigt werden, wenn man einen der Softkeys auf dem Gerät drückt. So kann man einfach ein GUI Element mit Funktionalität ergänzen.

Es werden zwei APIs zur Verfügung gestellt durch MIDP 2.0. Das eine API ist die High Level API. Hier hat man viele vorgefertigte GUI Elemente wie Listen, Eingabefelder etc., die man anpassen und auf einem Screen einfügen kann. Leider kann man diese Elemente nur sehr begrenzt den eigenen Bedürfnissen anpassen. Dafür ist höchste Kompatibilität mit den Geräten gewährleistet. Die Eventbehandlung beim High Level API läuft so, dass man das GUI Element einem `CommandListener` hinzufügt. Wird nun ein Event ausgelöst (normalerweise Auswahl eines Menu Eintrages in einem Softkey Menu), so wird die Methode `CommandAction()` des `CommandListeners` ausgeführt. Wenn man andere Tasten als die Softkeys drückt, so können diese Events nicht durch die High Level API behandelt werden.

Die Low Level API bietet unter anderem die Funktionalität einer `Canvas` Klasse. Dies ist eine Zeichnungs Oberfläche, auf die man mit `drawLine()` etc. eigene Primitiven zeichnen kann. Dabei überschreibt man die `paint()` Methode. Hier ist es auch möglich, auf Low Level Events zu reagieren. So kann man Events der Tastatur behandeln und mit Key Codes feststellen,

welcher Button gedrückt wurde. Spezielle Probleme (Siehe 3.3) stellen sich, da die Tastatur Codes und die Bildschirm Auflösungen von Gerät zu Gerät verschieden sind. Kompatibilität mit allen Geräten ist also nicht gewährleistet. Trotz dieser Nachteile haben wir uns für die Low Level API aus verschiedenen Gründen entschieden (Siehe 3.2).

3.2 Entscheide, die zum Gebrauch der Low Level API führten

- Im High Level API hat man sehr begrenzte Möglichkeiten, seine GUIs anzupassen und zu individualisieren. Gerade bei einem Spiel ist es aber wichtig, dass man eine schöne Benutzeroberfläche hat. Deshalb war schon früh klar, dass die Low Level API verwendet werden musste, auch wenn es zu Problemen mit den Geräten kommen sollte.
- Bei der High Level API hat man keine Möglichkeit, auf Fullscreen Modus umzuschalten. Das Spiel muss aber im Fullscreen Modul laufen, da man sonst etwa 1/3 der Bildschirmhöhe für Titelleiste etc. der Applikation verschwendet. Dies ist zu vermeiden, da die Bildschirme sonst schon sehr klein sind.
- Gewisse Fenster im Spiel müssen sowieso mit `Canvas` programmiert werden, da keine vorgefertigten GUI Elemente zur Lösung des Problems bestehen.
- In gewissen Fenstern muss mit Hilfe der Low Level API auf Low Level Events reagiert werden.

3.3 Probleme mit der Low Level API

- Einige Geräte haben einen Bug, so dass sie falsche Angaben über die Höhe und Breite des Bildschirms zurückgeben. Dieses Problem wurde so gelöst, dass in der Klasse `Devices` für diese Geräte die Angaben über die Auflösung von Hand eingetragen wurden.
- Einige Geräte haben falsche Tastaturcodes für den Select Button. Es gibt eine Methode in `Canvas`, die den Tastatur Code des Select Buttons eines Gerätes zurückgeben sollte. Diese Funktion ist aber auf den einen Geräten fehlerhaft implementiert. Deshalb ist bei diesen Geräten der richtige Tastaturcode für den Select Button in der Klasse `Devices` von Hand eingetragen.
- Bei der Eingabe von Daten mittels der Tastatur musste auf ein High Level `TextField` zurückgegriffen werden, obwohl es grafisch nicht gut aussieht. Kompatibilität mit sehr vielen Geräten ist hier ausserst wichtig. Wenn

man es mittels `Canvas` programmieren würde, könnte es sein, dass man für jedes Gerät Codes für alle Tastatur Keys von Hand in `Devices` eingeben müsste.

3.4 Entwickelte GUI Elemente

Zum Einsatz in Mobile Hattrick wurden folgende GUI Elemente auf Basis der Klasse `Canvas` implementiert:

- `HattrickList`: Diese Klasse implementiert eine Liste, die auf die Bedürfnisse von Mobile Hattrick abgestimmt ist. Die Benutzer Oberfläche wurde zum Thema Fussball passend gestaltet. Man kann in der Liste einzelne Zeilen mit dem Select Button auswählen. Es wird ein High Level Event generiert, so dass man die Logik wie bei einem High Level Element programmieren kann. Die Liste unterstützt auch Scrolling, falls nicht alles auf einen Bildschirm passt.
- `AufstellungDetailMenu`: Hier kann man Spieler auf verschiedene Positionen auf dem Spielfeld setzen. Je nach gewählter Aufstellung erscheinen verschieden viele Kreise auf den unterschiedlichen Positionen auf dem Spielfeld. Mit den Pfeiltasten kann man von Position zu Position springen. Drückt man nun die Select Taste, erscheint ein Menu, wo man einen Spieler für diese Position auswählen kann.

Das Klassendiagramm dieser Klassen ist in Abbildung 3.2 zu sehen.

Die `HattrickList` ist sehr vielseitig einsetzbar. Die meisten anderen Screens wie Menus oder Auflistung der Stärken eines Spielers etc. sind von der `HattrickList` abgeleitet. Die `handleEvent(Command com)` Methoden werden hierbei überschrieben, um eigene Funktionslogik zu programmieren. Abbildung 3.1 zeigt ein typisches Menu, das mit Hilfe einer `HattrickList` implementiert wurde.

Alle Screens in Mobile Hattrick implementieren das Interface `IHATTRICK-SCREEN`. Dieses Interface definiert folgende Methoden:

- `handleEvent(Command comm)`: Wird vom `EventDispatcher` aufgerufen, enthält Event Handling Logik.
- `init()`: einmalige Initialisierungen werden hier programmiert. Vom Konstruktor aufgerufen.
- `refresh()`: Dieser Code wird ausgeführt, wenn ein GUI Element neu in den Vordergrund kommt. Hier kann man allfällige Daten aktualisieren, bevor das GUI Element (erneut) angezeigt wird. Wird von `display()` aufgerufen.



Abbildung 3.1: Ein typisches Menu, das mit Hilfe einer `HattrickList` implementiert wurde

- `display()`: Zeigt dieses GUI Element auf dem Bildschirm an und bringt das aufrufende GUI Element in den Hintergrund.

Diese programmierten Klassen unterstützen double buffering, damit der Bildaufbau nicht sichtbar ist. Double-Buffering ist nicht direkt in MIDP 2.0 implementiert und muss von Hand programmiert werden. Dazu zeichnet man das Fenster mit all seinen geometrischen Primitiven und Schriftzügen zuerst in ein Bitmap im Memory. Wenn fertig gezeichnet ist, wird das Bitmap auf den Screen gezeichnet.

3.5 Zentraler Ereignis Dispatcher

Damit man in einem GUI Element nur Events erhält, die wirklich an dieses GUI Element gerichtet sind, wurde ein `EventDispatcher` programmiert (Siehe Abbildung 3.3). Jedes GUI Element registriert sich beim Dispatcher, falls es Events erhalten möchte. Der Dispatcher erhält alle Events (via Callback Funktion `commandAction(Command comm)`) und leitet diese an den richtigen Screen weiter. Dabei kann man auch seine eigenen Ereignisse definieren und versenden. Dies geschieht zum Beispiel in den Callback Funktionen der Bluetooth Socket Klassen. Wenn zum Beispiel ein Discovery fertig ist wird im Callback, der vom Betriebssystem aufgerufen wird, ein Event an das GUI Element geschickt. Darauf kann sich das GUI Element mit den neuen Daten neu zeichnen oder, falls ein Fehler aufgetreten ist, eine Fehlermeldung anzeigen.

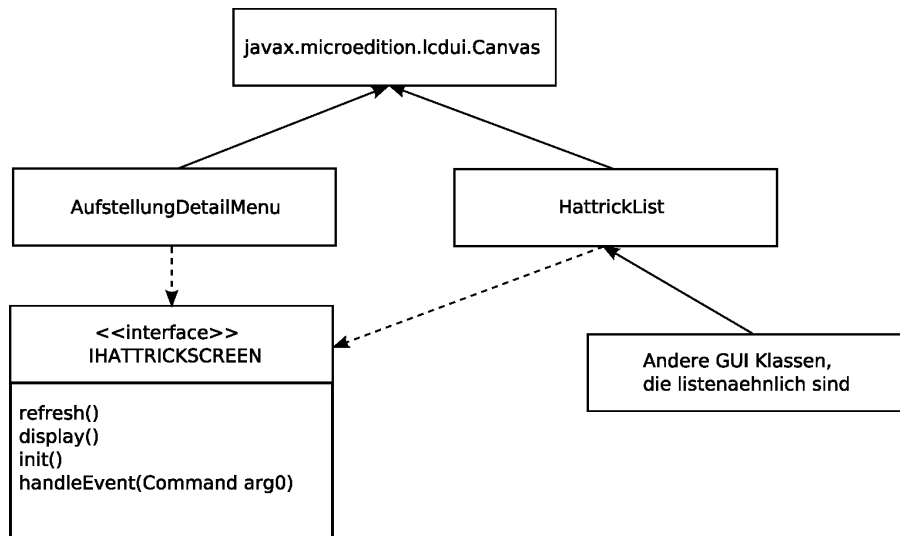


Abbildung 3.2: Das Klassendiagramm der GUI Klassen

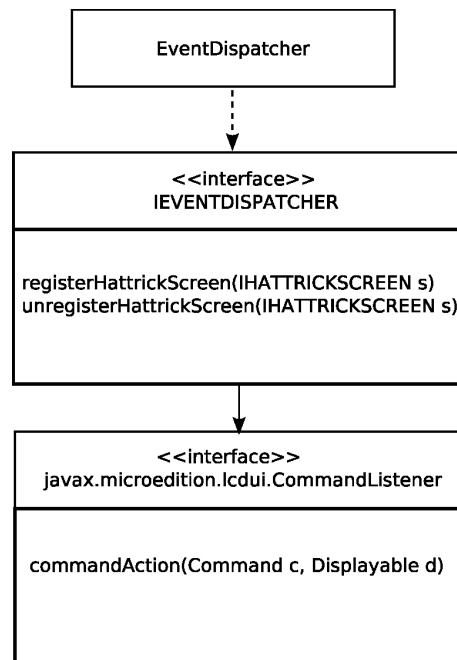


Abbildung 3.3: Das Klassendiagramm des EventDispatchers

Kapitel 4

Simulation

4.1 Teamgenerierung

Jeder User kann zu Beginn bei der Erstellung des Teams selber wählen, mit welcher Aufstellung er am liebsten spielt. Natürlich kann im Verlauf des Spiels die Aufstellung beliebig geändert werden, es kann aber gut sein, dass man dann vielleicht zu wenig Spieler einer Position zur Verfügung hat. Zur Auswahl stehen die folgenden Aufstellungen:

- **3-4-3**
- **3-5-2**
- **4-3-3**
- **4-4-2**
- **4-5-1**
- **5-3-2**
- **5-4-1**

Ein Team besteht aus 18 Spielern, wobei die Anzahl Spieler einer Position von der anfänglich gewählten Aufstellung abhängt. Es werden 2 Torhüter plus je 2 zusätzliche Spieler jeder Position generiert.

Beispiel: Aufstellung = 4-4-2, Team besteht aus 2 Torhütern, 4+2 Verteidigern, 4+2 Mittelfeldspielern und 2+2 Angreifern.

4.1.1 Spieler-Attribute

Jeder Spieler besitzt folgende Attribute:

- **Alter** (zw. 18-37): Je jünger ein Spieler ist, desto grösser sind seine Fortschritte nach einem Spiel.
- **Karriereende** (zw. 34-37): Erreicht ein Spieler dieses Alter so beendet er seine Karriere und ein Spieler aus dem Nachwuchs rückt nach.
- **Position** (Torwart, Verteidigung, Mittelfeld, Angriff): Die bevorzugte Position des Spielers

Folgende Attribute haben einen Wert zwischen 1 und 20, wobei 1 sehr schlecht und 20 sehr gut ist. Nach jedem Spiel kann sich ein Spieler verbessern bzw. verschlechtern. Auch nach einer Saison von 20 Spielen werden die Attribute der Spieler, abhängig der Resultate, nochmals geupdated.

Anmerkung: Im Programm wird mit 100 - 2000 gerechnet um zwei Dezimalstellen zu simulieren, da Java ME keine Floating-Point Operationen zulässt.

- **Torwart** (zu Beginn = 4): Drückt die Stärke eines Torwarts aus.
- **Verteidigung** (4): Bestimmt wie gut ein Spieler verteidigen kann und ob ein Stürmer gegen ihn zum Abschluss kommt oder nicht.
- **Aufbau** (4): Je nach Stärke der Mittelfeldspieler kommt ein Team zu mehr oder weniger Chancen.
- **Torschuss** (4): Je stärker ein Stürmer ist, desto grösser wird die Wahrscheinlichkeit, dass er seine Chance wahrnimmt und ein Tor erzielt.
- **Kondition** (3): Die Kondition spielt in der zweiten Halbzeit eine Rolle. Ist ein Team konditionell stark, so wird es die zweite Halbzeit dominieren.
- **Erfahrung** (5): Je erfahrener ein Spieler ist umso mehr ist er in der Lage während des Spiels sein volles Potenzial abzurufen. Die Erfahrung nimmt pro gespielter Partie zu.

Jeder User kann sein individuelles Team zusammenstellen, indem er bei der Erstellung des Teams jedem Spieler zusätzlich zu den Anfangswerten noch 3 Stärkepunkte zuweisen kann.

4.2 Spielsimulation

Die Spielsimulation ist in zwei Phasen unterteilt. In der Ersten wird die Anzahl Chancen pro Halbzeit berechnet und in der Zweiten wird dann jede Chance einzeln simuliert. Hierfür gibt es die Klasse `Simulation`

Die Klasse `SimulationWrapper` dient dazu, die ganze Simulation zu koordinieren. Sie simuliert die erste bzw. zweite Halbzeit, kümmert sich um die Updates nach einem Spiel bzw. Saison, verwaltet und updated alle Statistiken und berechnet auch den neuen ELO-Punktwert der Teams.

4.2.1 Chancenberechnung

Bei der Berechnung der Chancenanzahl werden die Durchschnittswerte des Mittelfeldes beider Teams miteinander verglichen. Dabei kommen die Attribute `Aufbau`, `Erfahrung` und in der 2. Halbzeit `Kondition` zum tragen. Ist das eine Team dem Anderen im Mittelfeld stark überlegen, so hat es auch mit grosser Wahrscheinlichkeit ein Chancenplus. Zwei Zufallszahlen fliessen hier ebenfalls ein, die Eine bestimmt darüber ob es ein eher chancenreiches oder chancenarmes Spiel wird und die Andere variiert die Anzahl der Chancen noch wenig.

4.2.2 Chancensimulation

Zuerst wird zufällig ein Stürmer bestimmt, der zur Chance kommt. Mit kleiner Wahrscheinlichkeit passiert es auch mal, dass ein Mittelfeldspieler bzw. ein Verteidiger zum Abschluss kommt. Ebenfalls zufällig werden ein bis zwei Verteidiger ausgewählt, wobei dann der Bessere gegen den Stürmer verteidigt. Dieser Zweikampf wird folgendermassen simuliert und entscheidet ob ein Stürmer zum Torschuss kommt oder nicht:

- Besserer Wert B und schlechterer Wert S , z.B. $B > S$ (Stürmer > Verteidiger) dann $B = \text{Stürmer}$, $S = \text{Verteidiger}$, sonst umgekehrt.
- $B_e = \text{Erfahrungswert von B}$, $S_e = \text{Erfahrungswert von S}$
- $B = B * \frac{B_e}{S_e}$, $S = S * \frac{S_e}{B_e}$
- Range der Randomnumber ist $[0, (B + S)]$
- S gewinnt wenn Seed zwischen $[0, S]$, sonst gewinnt B
- Je grösser der Unterschied zwischen den Spielern, desto grösser ist die Chance, dass B gewinnt.
- Gewinnt der Verteidiger, so ist die Chance vertan, gewinnt der Stürmer so kommt er zum Abschluss.
- **Abschluss:** Nochmals die gleiche Simulation, einfach zwischen Stürmer und Torwart. Gewinnt der Stürmer erneut, so hat er ein Tor erzielt.

4.2.3 Verletzungen

Ein Spieler kann sich während des Spiels verletzen. Simuliert wird das folgendermassen: Bei jeder Chance werden zwei Zufallszahlen generiert. Liegen diese nicht mehr als 1 auseinander, so verletzt sich zufällig ein Spieler einer Mannschaft. Dabei wird der Spieler automatisch durch einen Ersatzspieler ersetzt. Sollte dabei kein Spieler mehr auf der Bank sein, der die selbe Position spielt wie der Verletzte, so kommt irgend ein Spieler für ihn ins Spiel. Die Verletzungsdauer eines Spielers kann zwischen 1 und 11 Wochen betragen, wobei eine kleine Verletzung mit nur einer Woche Pause wahrscheinlicher ist.

4.3 Verbesserungen

Die Verbesserungen der Spieler geschehen zu zwei Zeitpunkten. Nach einem Spiel und nach einer Saison von 20 Spielen.

4.3.1 Nach einem Spiel

Die Klasse `AfterGame` führt nach jedem Spiel die Updates durch. Es werden die ganzen Statistiken des vorangegangenen Spiels übergeben und daraus berechnet, wie stark sich ein Spieler verbessert. Ein wichtiger Wert dabei ist der Vergleich der beiden Stärken der Mannschaften, wobei nur die aufgestellten Spielerwerte miteinflussen.

Erfahrung

- Die Erhöhung der Erfahrung ist in etwa proportional mit dem Älterwerden der Spieler, falls ein Spieler jedes Spiel gespielt hat.
- Ist das Team gleichstark oder besser als der Gegner so erhöht sich die Erfahrung pro Spieler um 0.02.
- Ist das Team um maximal 3 Stärkepunkte schlechter als der Gegner so erhöht sich die Erfahrung pro Spieler um 0.05.
- Ist das Team um mehr als 3 Stärkepunkte schlechter als der Gegner so erhöht sich die Erfahrung pro Spieler um 0.1.
- Bei einer Niederlage gibt es noch zusätzlich einen Abzug von 0.02 Punkten pro Spieler. Verliert man also gegen ein gleichstarkes bzw. schlechteres Team so nimmt die Erfahrung nicht zu.

Torhüter

- Ein Torhüter verbessert sich bis zu seinem 32. Lebensjahr um max. 4 Punkte pro Jahr.
- Grundverbesserung X_0 ist
 - = 0, wenn er mehr Tore bekommen hat als gehaltene Schüsse.
 - = 0.1, wenn er zwischen $\frac{1}{2}$ und $\frac{2}{3}$ aller Schüsse parriert hat.
 - = 0.15, wenn er mehr als $\frac{2}{3}$ aller Schüsse parriert hat.

Verteidiger

- Ein Verteidiger verbessert sich bis zu seinem 30. Lebensjahr um max. 4 Punkte pro Jahr.
- Grundverbesserung X_0 ist
 - = 0, wenn der Verteidiger mehr Zweikämpfe verloren als gewonnen hat.
 - = 0.1, wenn der Verteidiger zwischen $\frac{1}{2}$ und $\frac{2}{3}$ aller Zweikämpfe gewonnen hat.
 - = 0.15, wenn der Verteidiger mehr als $\frac{2}{3}$ aller Zweikämpfe gewonnen hat.

Mittelfeld

- Beim Mittelfeld wird zwischen besseren und schlechterem Team unterschieden. Sind beide Mannschaften gleichstark, so verbessert sich nur das Team mit der grösseren Anzahl an Chancen. Ist ein Team um mehr als 3 Stärkerpunkte besser, so verbessert sich kein Team.
- **Besseres Team:** (0 - 3 Stärkerpunkte besser)
 - falls Chancenverhältniss zwischen 1 und 1.5, dann Grundverbesserung $X_0 = 0.1$.
 - falls Chancenverhältniss grösser als 1.5, dann Grundverbesserung $X_0 = 0.15$.
 - sonst $X_0 = 0$.
- **Schlechteres Team:**
 - falls Chancenverhältniss über 1, dann Grundverbesserung $X_0 = 0.2$.
 - falls Chancenverhältniss zwischen 0.75 und 1, dann Grundverbesserung $X_0 = 0.15$.
 - falls Chancenverhältniss zwischen 0.5 und 0.75, dann Grundverbesserung $X_0 = 0.1$.
 - sonst $X_0 = 0$.

Sturm

- Ein Stürmer verbessert sich nur wenn er ein Tor erzielt hat.
- bei 1 - 2 Toren, Grundverbesserung $X_0 = 0.1$.
- bei mehr als 2 Toren, Grundverbesserung $X_0 = 0.15$

Alterseinfluss:

- Bei den vier Attributen Torwart, Verteidigung, Mittelfeld und Sturm fließt der Faktor Alter folgendermassen noch ein:
- Alter zwischen:
 - 18 und 21, $X = X_0 \cdot \frac{4}{3}$.
 - 21 und 26, $X = X_0 \cdot \frac{6}{4}$.
 - 26 und 30, $X = X_0$. (Beim Torwart ist hier die obere Grenze gleich 32)
 - älter als 30, $X = 0$. (Auch hier, Torwart bis 32 Jahren)

Kondition

- Die Kondition verbessert sich bei allen Spielern die gespielt haben um 0.15, plus dem oben schon genannten Alterseinfluss.

4.3.2 Nach einer Saison

Nach einer Saison von 20 Spielen verbessern sich die Spieler erneut, zu alte Spieler beenden ihre Karriere und neue Spieler rücken nach. Das Ganze wird in der Klasse `AfterSeason` ausgeführt. Dabei wird jeder Spieler ein Jahr älter und basierend auf den Ergebnissen der vorangegangenen Saison wird jeder Spieler nochmals verbessert. Anders als nach einem Spiel sind die Verbesserungen nach einer Saison zufälliger, damit sich nicht jeder Spieler gleichstark entwickelt.

- Verbesserung X ist
 - = $[0.2, 0.3]$, bei 1 - 3 gewonnenen Spielen. (Schwächebonus)
 - = $[0.1, 0.2]$, bei 5 - 10 gewonnenen Spielen.
 - = $[0.2, 0.4]$, bei 10 - 15 gewonnenen Spielen.
 - = $[0.4, 0.6]$, bei 15 - 20 gewonnenen Spielen.
- Zusätzlich verbessert sich der Torhüter und die Verteidiger bei keiner Niederlage um $[0.2, 0.4]$.
- **Altersabzug:** zwischen 32 - 34 Jahren = $[0.1, 0.3]$, ab 34 Jahren $[0.2, 0.5]$.

4.4 Ranking

Das Ranking der Teams wurde mit Hilfe des ELO-Punktesystems erstellt, dass vor allem beim Schach üblich ist. Hierfür steht die Klasse `EloPunktesystem` zur Verfügung. Gerechnet wird folgendermassen, wobei jedes Team als Startwert 1800 ELO-Punkten hat:

Parameter

- $t1$ = ELO-Punktzahl vom zu berechnendem Team
- $t2$ = ELO-Punktzahl des Gegners
- $S_a = \begin{cases} 1, & \text{bei einem Sieg} \\ 0.5, & \text{bei einem Remis} \\ 0, & \text{bei einer Niederlage} \end{cases}$
- $k = \begin{cases} 15, \\ 10, & \text{falls } t1 > 2400 \text{ ist} \\ 25, & \text{falls weniger als 30 gewertete Partien} \end{cases}$

Berechnung

- Erwartungswert $E_a = \frac{1}{1+10^{\frac{t2-t1}{400}}}$
- Neuer ELO-Wert $R_a = t1 + k(S_a - E_a)$

Anmerkung Da man nicht mit 0.5 rechnen kann, wurde die Rechnung angepasst. D.h. S_a ist 2, 1 oder 0 und die Berechnung für R_a wird zuerst *2 gerechnet und am Schluss das Ergebniss durch 2 geteilt.

4.5 Ausblick

4.5.1 Geld

Mit der Hinzunahme von Geld eröffnen sich natürlich viele Möglichkeiten. Die erste Frage wäre da, wie kommt man zu Geld? Ein einfacher Weg wäre, wenn man für jedes gespielte Spiel Prämien bekommen würde, abhängig von der Stärke der Teams. Ein wenig aufwändiger wäre die Möglichkeit eines Stadions das Geld generieren würde. Zum Einen durch Zuschauereinnahmen und zum Anderen mit der Vermietung von Werbeflächen und VIP-Logen an Sponsoren. Die Möglichkeit von verschiedenen Wettbewerben wie Champions-League oder ähnlichem schliesse ich aus, da durch die Peer-to-Peer Nutzung des Spiels ein koordinierter Saisonablauf ohne zentrale Stelle fast unmöglich ist.

Nun hätte man Geld, nur was kann man damit machen? Auch hier gibt es eine breite Palette an Möglichkeiten. Das geht vom Ausbau der Plätze im Stadion, was die Einnahmen erhöht bis zu der Möglichkeit, verschiedene Trainer, Ärzte, Masseur etc. einstellen zu können. Dabei würde sich das Team viel individueller entwickeln. Jedoch müsste dann bei der Verbesserung gewisse Korrekturen vorgenommen werden, damit sich ein Team je nach eingestelltem Personal unterschiedlich entwickelt. Auch ein Training nach jedem Spiel wäre denkbar, wobei man ziemlich genau darauf achten müsste, wie sich ein Team entwickeln kann, damit User, die viel spielen, nicht zu stark bevorteilt werden.

4.5.2 Transfermarkt

Ein einfacher Ansatz wäre, jedem Spieler einen gewissen Wert zuzuschreiben, abhängig von seinen Attributen. Somit würde das Programm selber denn Marktwert eines Spielers festlegen und die Möglichkeit ein Jahrhunderttalent für wenig Geld zu bekommen wäre nicht vorhanden. Andererseits besteht so die Möglichkeit, dass zwei Parteien Spieler tauschen können, ohne dass eine Drittpartei darauf Einfluss nehmen könnte.

Der aufwändigere Ansatz wäre ein zentraler Transfermarkt, der eigentlich so funktioniert wie ein normaler Transfermarkt. Das Problem dabei ist, dass damit das Spiel ziemlich stark an eine zentrale Stelle gekoppelt ist und das häufige synchronisieren mit dem Server unumgänglich ist. Dabei verliert man einen grossen Teil der P2P-Eigenschaft, die dieses Spiel mit sich bringt.

Eine weitere Möglichkeit, die jedoch genauer analysiert werden müsste, wäre die Propagation einer Transferliste unter den einzelnen Usern. Man würde dabei ohne zentrale Stelle auskommen, jedoch hat man so nur eine Teilansicht auf das ganze Transfergeschehen. Des weiteren müsste eine Lösung gefunden werden, für das Problem bei dem ein User ein Spieler von einem anderen User kaufen will, den er weder kennt noch jemals mit ihm gespielt hat. Austausch per SMS wäre ein Ansatz, wobei man aber wieder an die gleichen Probleme aneckt wie die Synchronisierung zweier User vor einem Spiel. Wer entscheidet ob jetzt die Spieler getauscht werden dürfen und wie verhindert man, dass ein Spieler zweimal vorkommt?

Kapitel 5

Persistente Speicherung

5.1 RMS vs. File Connection API

Zur persistenten Speicherung von Daten aus einem Midlet stehen 2 Möglichkeiten zur Verfügung: RMS (Record Management System) oder der File Connection API. RMS ist standardmässig in MIDP 2.0 implementiert, jedoch variiert der Platz der ihm zugeteilt wird stark und ist teilweise sehr klein. Mit der File Connection API hat man Zugriff auf einen Teil des Filesystems des Mobiltelefons und hat deswegen mehr Speicherplatz zur Verfügung. Der entscheidende Nachteil ist jedoch, dass ihre Implementierung (JSR-75) in MIDP 2.0 optional ist. Erst wenige Mobiltelefone unterstützen die API. Aufgrund der Tatsache dass Mobile Hatrick relativ (für mobile Geräte) grosse Datenmengen speichern muss haben wir uns doch für die File Connection API entschieden. Die Verbreitung von JSR-75-implementierenden Mobiltelefonen wird durch die nächste Generation der Geräte mit Sicherheit wachsen.

5.2 Architektur

Daten von Mobile Hatrick, die persistent gespeichert werden müssen sind:

Team Die Spieler, ihre Attribute sowie die letzte Aufstellung

Spiele Gespielte Spiele, abgebrochene Spiele, letzte gameid

Zustand Nickname, letzte Synchronisation mit dem Server, Keys für Signaturen, Zertifikate, Banlist

Diese Daten werden werden auf folgende 6 Files verteilt

- register.db
- teams.db

- sync.db
- gameid.db
- games.db
- tempgame.db

Es ist unbedingt notwendig, dass das Mobiltelefon die File Connection API (JSR-75) implementiert. Dabei ist die File-System-Root von Gerät zu Gerät unterschiedlich. Uns bekannte Roots sind

Sun Wireless Toolkit "file:///root1/"

Nokia 6680 System.getProperty("fileconn.dir.private")

Sony Ericsson w800i, w810i "file:///c:/other/"

mobileht.storage

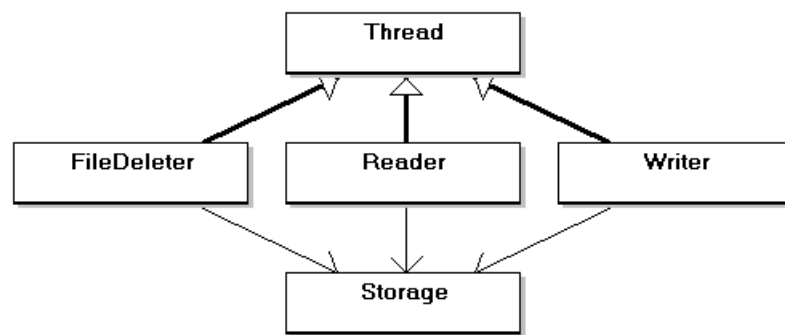


Abbildung 5.1: Klassendiagramm mobileht.storage

Beim Aufstarten des Spiels wird eine Instanz der Klasse Storage kreiert, die danach von überall her mit `toplevel.getStorage()` zur Verfügung steht. Die 6 oben erwähnten Dateien werden erstellt falls sie noch nicht existieren. Und die Streams für den Zugriff auf sie werden erstellt. Danach stellt die Klasse Storage verschieden Methoden zur Verfügung:

- `readFile()`, `writeFile()`
- `append()`, `append_front()`
- `fileSize()`

Bei diesen Methoden wird jeweils das File sowie gegebenenfalls ein Byte-Array übergeben. Das Lesen, Schreiben und Löschen muss in separaten Threads ausgeführt werden, um den `CommandListener` der Applikation nicht zu blockieren. Der Hauptthread wartet jedoch mit `join()` darauf, bis die Aktion beendet ist. Danach können die gelesenen Bytes mit `getData()` abgerufen werden.

5.3 Future Work

- Ausbau auf mehrere Geräte, dynamisches Erkennen des Geräts (im Moment muss die Applikation für ein spezielles Gerät kompiliert werden)
- Öffnen der Files und der Streams erst bei Gebrauch der Files. Hat Thread-Komplikationen verursacht und wird deshalb jetzt im Konstruktor des MIDlets gemacht.
- Alternative Möglichkeit die Daten mit RMS zu speichern, falls JSR-75 nicht implementiert wird.

Kapitel 6

Kommunikation

6.1 Mobiltelefon zu Mobiltelefon

6.1.1 Architektur

Die Kommunikation zwischen zwei Mobiltelefonen wird über eine Bluetooth-Verbindung abgewickelt. Es gibt grundsätzlich 2 Möglichkeiten, Mitspieler die in der Nähe sind zu finden und ein Spiel zu beginnen.

Host/Client Wenn zwei Spieler sich kennen und wissen dass sie gegeneinander spielen wollen geht einer von ihnen in den „Host-Status“, und einer in den „Discovery-Status“. Diese Variante des Device Discovery ist sehr robust und zuverlässig.

Hintergrundsuche Wenn man sich an bevölkerten Orten (Zug, Vorlesung) befindet und nicht weiss ob sonst noch jemand Mobile Hattrick spielen will startet man die „Hintergrundsuche“. Dabei befindet sich das Spiel gleichzeitig im Host- und Discovery-Status, was aus Bluetooth-eigenen Gründen weniger robust ist. Das Auffinden von Mitspielern kann mehrere Minuten dauern. Wird ein Mitspieler gefunden (dieser muss sich entweder im Host-Status befinden oder ebenfalls eine Hintergrundsuche gestartet haben), vibriert das Mobiltelefon für eine Sekunde.

Host-Status Das Spiel stellt den Bluetooth-Service von Mobile Hattrick zur Verfügung und wartet auf Anfragen.

Discovery-Status Es wird nach einer zufälligen Anzahl Sekunden zwischen 30 und 50 nach Geräten gesucht, die den Mobile Hattrick-Service anbieten. Die gefundenen Geräte werden angezeigt.

Im Discovery-Status oder während der Hintergrundsuche werden die Geräte auf dem Mobiltelefon angezeigt und es ist möglich sich mit dem anderen Gerät zu verbinden.

6.2 Mobiltelefon zu Server

Für die Kommunikation mit dem Server zur Registration oder Synchronisation wird eine TCP-Verbindung verwendet. Diese wird entweder über GPRS aufgebaut, oder es wird über den PC gemacht (z. B. via Sony Ericsson Device Explorer). Letzteres würde keine Kosten verursachen. Dazu können die Methoden

- `register(nickname)`: User registrieren
- `sync(games, playerid)`: Synchronisation
- `syncblocking(games, playerid)`: Blockierende Synchronisation

der Klasse `mobileht.servercommunication.TCPSocket` aufgerufen. Die genauere Beschreibung der Funktionen befinden sich in Kapitel 8.

6.2.1 Future Work

- Kommunikation über SMS/MMS (einfacheres Kostenmodell, bessere Unterstützung durch Mobiltelefone)

Kapitel 7

Security

7.1 Angreifer-Modell

Das Modell eines Angreifers bei der Entwicklung von Mobile Hat trick ist:

- Einzelner Angreifer (keine Koalitionen)
- Kryptographie kann nicht gebrochen werden

Koalitionen von Gegnern ist bei Mobile Hat trick nur sehr schwer beizukommen. Zusammenarbeitende Spieler sind nicht von ganz normalen Spielern zu unterscheiden. Zu verhindern, dass jemand mit zwei Mobiltelefonen zwei Teams bei Mobile Hat trick besitzt, ist nicht möglich. In der Punkteverteilung wird allerdings versucht, Koalitionen so stark wie möglich zu schwächen, allerdings ohne „echten“ Gruppen von Spielern zu schaden. Mehrmaliges Spielen gegen den gleichen Gegner soll weniger Nutzen (Verbesserungen, Ranglistenpunkte) bringen als das Spielen gegen viele verschiedene Spieler.

7.2 Implementation

Zur Implementation der Signaturen und Verschlüsselung werden Schlüsselpaare (secret key, public key) verwendet. Der Server und jeder einzelne Spieler hat ein solches Paar.

7.2.1 PKI

Bei der Registrierung erhält der Spieler

- Eigenes Schlüsselpaar (je 1024 bit)
- Public Key des Servers

- Zertifikat für das eigene Schlüsselpaar

Zur Generierung der Schlüssel und der Verifikation von Signaturen werden die Klassen von `bouncycastle`¹ verwendet.

7.3 Protokolle

7.3.1 Nomenklatur

Konkatenierung `msg1|msg2`

Signatur $\{msg\}^{sk}$

Verschlüsselung $\{msg\}_{pk}$

sk Secret Key

pk Public Key

server_pk Public Key des Servers

7.3.2 `handshake()`

Im Handshake werden Informationen zum jeweiligen Spieler im Klartext übertragen. Zusätzlich ein Zertifikat für den eigenen Public Key. Anhand dieser Daten kann der Spieler feststellen ob der gegnerische Spieler gebannt² ist, wieviele verifizierte Spiele er bestritten hat, und was sein Rang ist. (siehe 7.1)

7.3.3 `playgame()`

In dieser Methode findet der Austausch der Commitments zu den einzelnen Seed-Startwerten statt, auf Basis deren die Simulation später ausgeführt wird. Die Seed-Startwerte der beiden Spieler werden mit bitweisem XOR verknüpft. Das Resultat wird als Startwert eines Pseudozufallsgenerators gesetzt, der die Zufallszahlen für die Simulation generiert. So kann kein einzelner Spieler allein den Ausgang des Spiels beeinflussen. Es wird jeweils für beide Halbzeiten ein solcher Startwert generiert. Die sensiblen Daten (Seed-Startwerte, Startaufstellung) werden ebenfalls verschlüsselt mit dem Public Key des Servers ausgetauscht, so dass der Server im Falle eines Spielabbruchs das Spiel, mit Ausnahme der Auswechslungen die in der Halbzeit getätigt werden könnten, zu Ende berechnen kann, ohne auf die Daten des zweiten Spielers angewiesen zu sein. (siehe Abbildung 7.2)

Sollte ein Fehler auftreten (z.B. bei der Verifikation der Signaturen), gilt das Spiel als nicht gespielt. Wird dieser Punkt doch passiert, gilt das Spiel als angefangen, und wird notfalls vom Server zu Ende berechnet. Dabei stellt sich

¹Eine frei verfügbare Kryptographie-Bibliothek für J2SE und J2ME [12]

²Der Server führt eine Liste mit gebannten Spielern die bei jeder Synchronisation auf das Mobiltelefon geladen wird

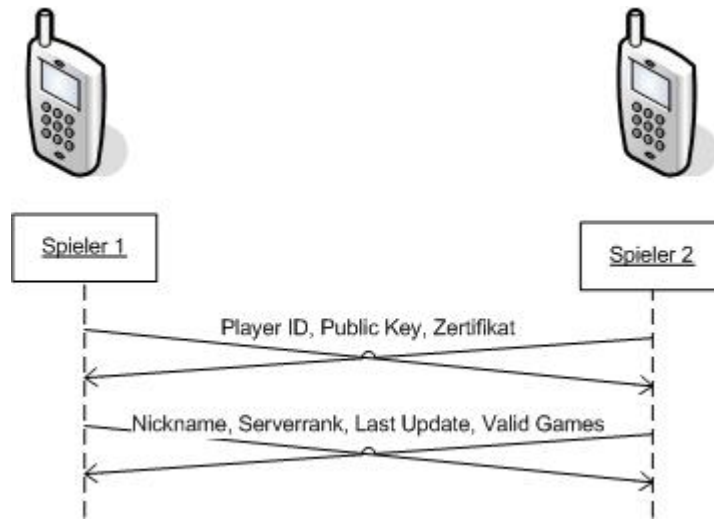


Abbildung 7.1: handshake: Austausch von Spielerinformationen

im letzten Informationsaustausch das bekannte 2-Armeen-Problem³, es müssten unendlich viele Bestätigungen hin- und hergeschickt werden, damit sie sich über den Zustand des anderen sicher sein könnten.

7.3.4 playnext()

`playnext()` führt die nächste Halbzeit aus. Die Teams und deren Commitments werden übertragen und dann die Seed-Startwerte für die jeweilige Halbzeit aufgedeckt. Die Halbzeit kann danach von beiden Spielern berechnet werden und sie erhalten das gleiche Resultat (siehe Abbildung 7.3). Sollte ein Fehler auftreten, wird eine Zwangssynchronisation⁴ mit dem Server ausgeführt, von dem man dann das neue Team erhält.

7.4 Registrierung

Bevor man das erste Spiel bestreiten kann, muss man sich beim Mobile Hatrick Server registrieren. Man erhält unter anderem eine Player ID und die nötigen Keys. (siehe Abbildung 7.4) Danach wird automatisch eine Synchronisation (siehe Kapitel 7.5) ausgeführt, um die nötigen Zertifikate zu erhalten.

³siehe Appendix 11.1

⁴Beide Spieler müssen sich mit dem Server synchronisieren um weitere Spiele spielen zu können

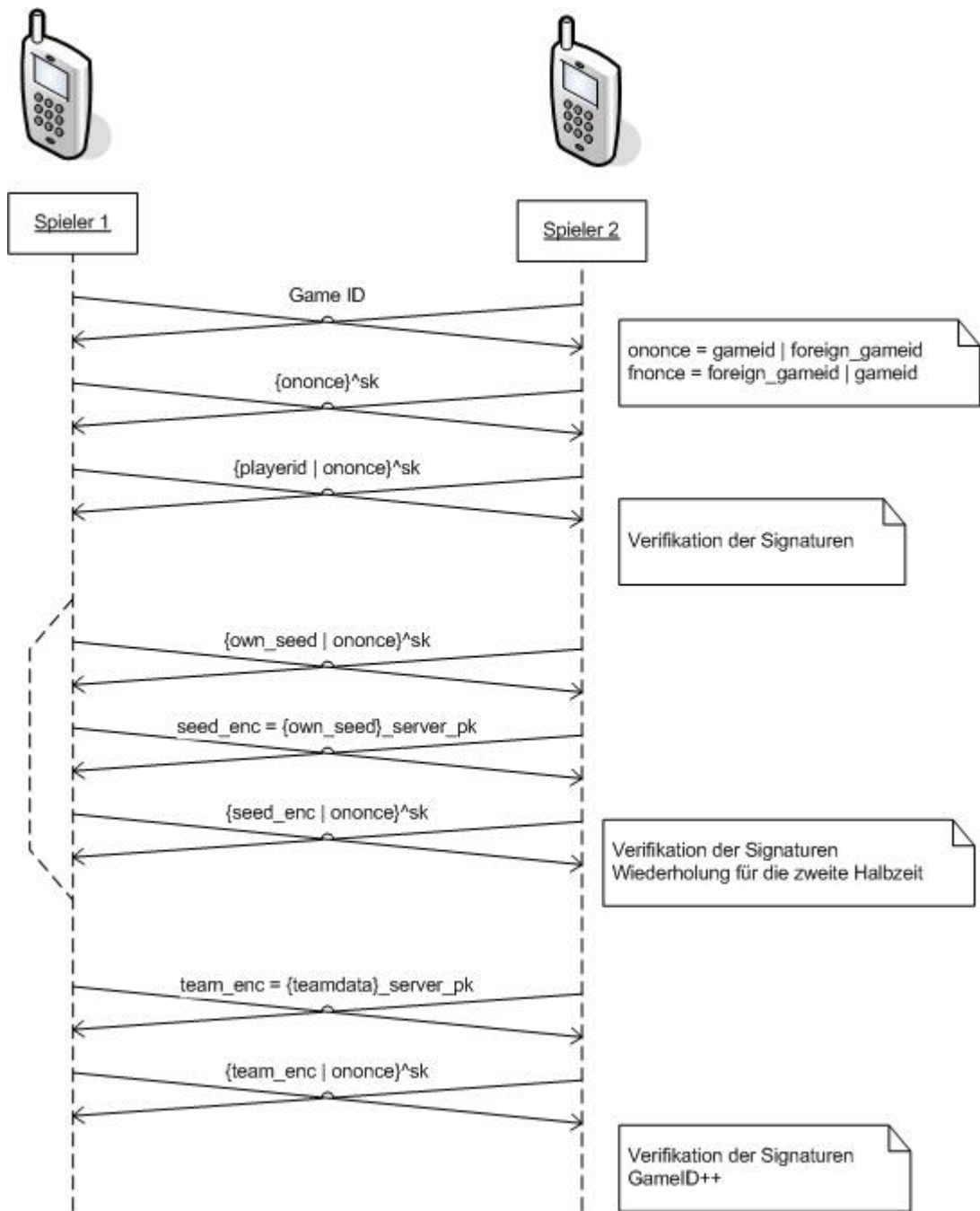


Abbildung 7.2: playgame: Austausch der Commitments

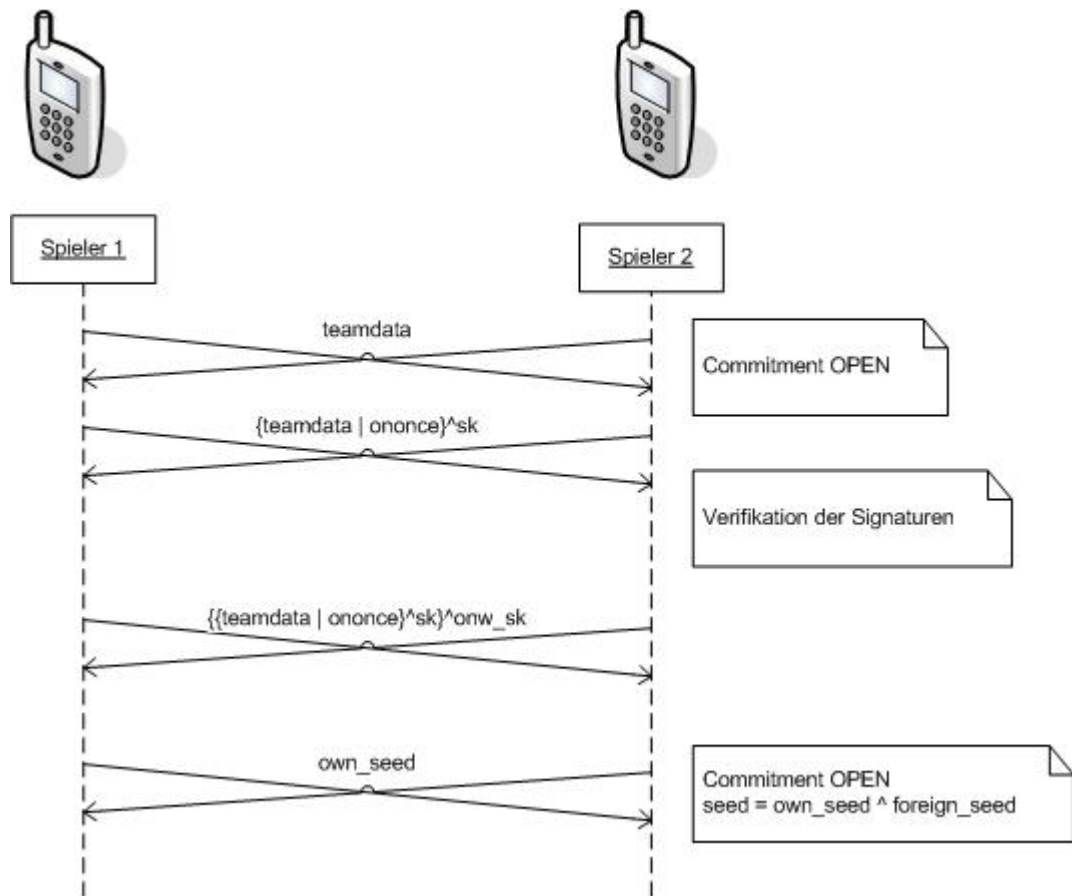


Abbildung 7.3: playnext: Öffnen der Commitments

7.5 Synchronisation

Von Zeit zu Zeit sollte man sich mit dem Mobile Hat trick Server synchronisieren. Dieser verifiziert ob die Spiele korrekt abgelaufen sind und aktualisiert die globale Rangliste (siehe Abbildung 7.5). Ausserdem erhält man neue Zertifikate zu dem globalen Rang, den verifizierten Spielen, und dem letzten Update. Die genaue Funktionsweise des Servers und des Verifiers wird in Kapitel 8.5 genauer beschrieben.

7.6 Attacken

Zwei Attacken sind mit dem oben beschriebenen Protokoll weiterhin möglich:

Absichtlicher Verbindungsabbruch Ein Client könnte absichtlich die Ver-

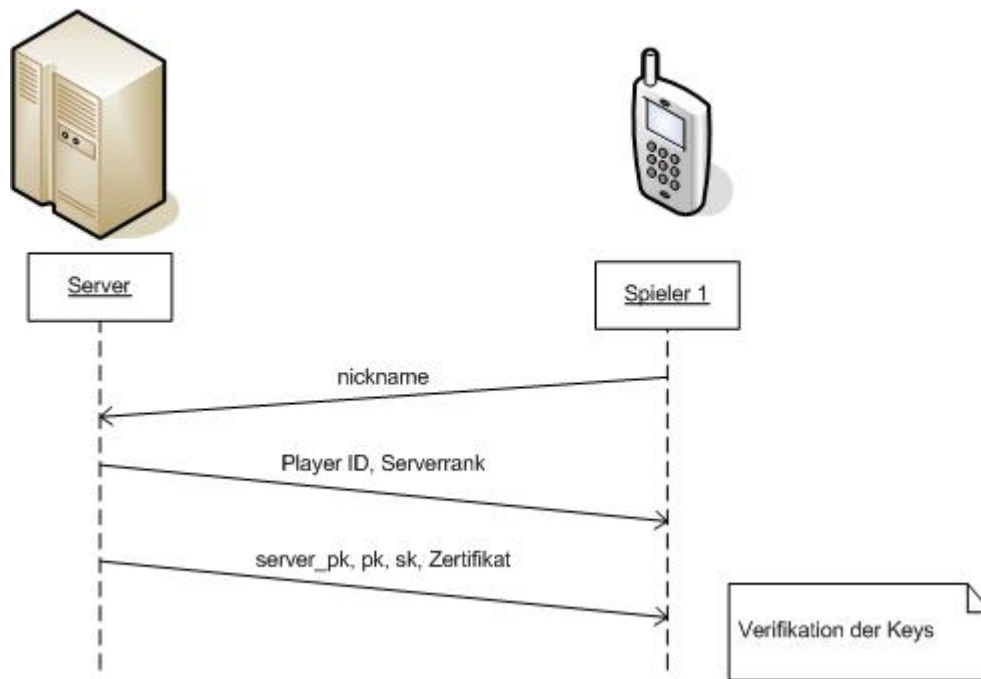


Abbildung 7.4: Registrierung beim Server

bindung in der Halbzeit abbrechen, um zu verhindern dass der Gegner Auswechslungen vornhemen kann (er kann dann aber auch selbst nicht auswechseln). Doch hat er dadurch keinen klaren Vorteil, da er das Endresultat dann selbst nicht errechnen kann.

Koalition Durch eine Koalition mit einem anderen Spieler könnte ein Client immer nur die gewonnen Spiele hochladen, in denen sich sein Team verbessert. Wegen des 2-Armeen-Problems⁵ würde es beim zweiten Spieler nicht gewertet werden (da man es immer in der ersten Phase abbrechen würde) und sich dessen Team nicht verschlechtern. Eine Lösung wäre, auf Simulationsebene mehrmalige Spiele gegen Gruppen von Spielern nicht so hoch zu bewerten. Wir gehen aber davon aus dass es keine Koalitionen gibt.

⁵siehe Appendix 11.1

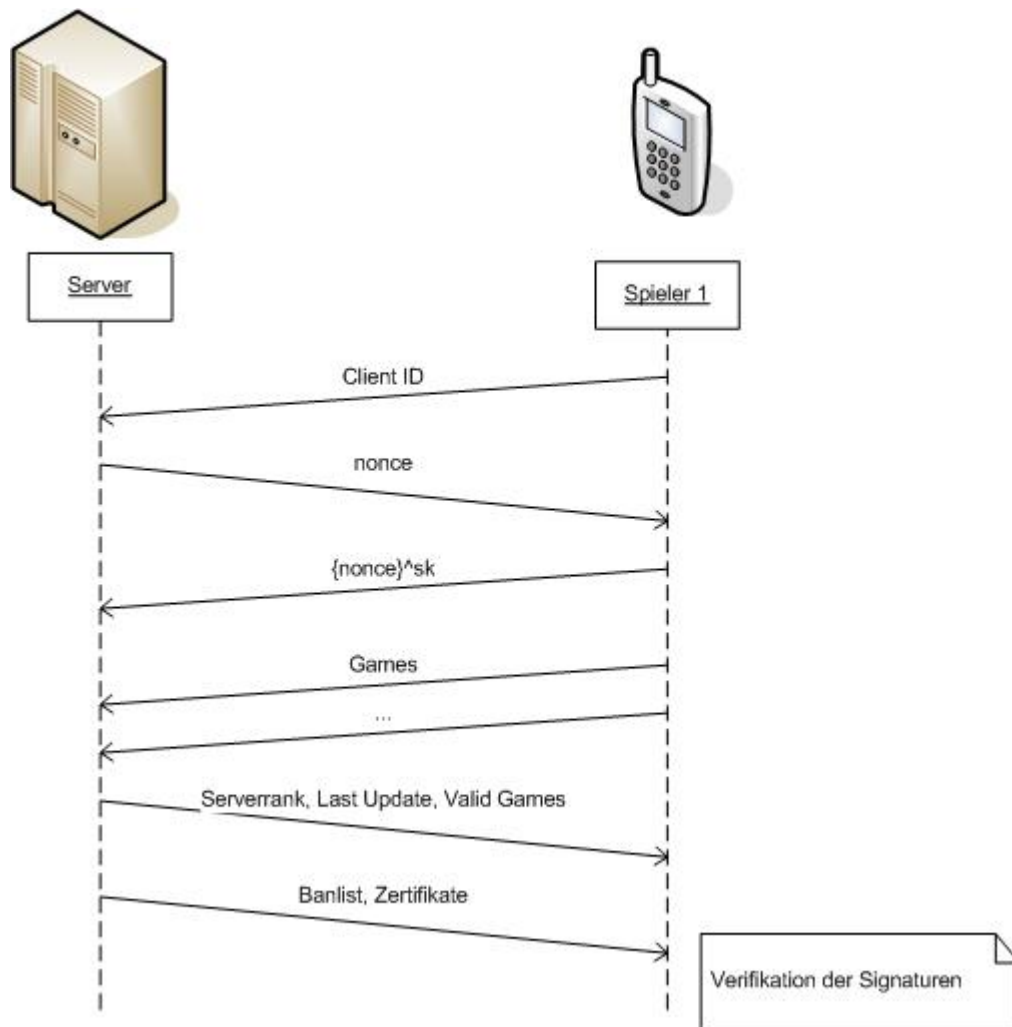


Abbildung 7.5: Synchronisation mit dem Server

Kapitel 8

Server

8.1 Zusammenfassung

Jeder Spieler muss sich am Anfang beim Server registrieren und erhält ein Key Paar sowie eine Spieler ID. In Mobile Hattrick spielt man Spiele, ohne dass man permanent mit einem zentralen Server in Verbindung steht (P2P). Dies bringt einige Probleme mit sich. Zum einen ist es möglich im Spiel zu cheaten, da der andere Spieler nur sehr begrenzte Möglichkeiten hat, einen Cheater selbst zu erkennen. Es können falsche Teams geschickt werden, verlorene Spiele als nie dagewesen betrachtet werden und so weiter. Zum anderen hat man nur eine lokale Rangliste, das heißt man kann sich nicht mit Spielern vergleichen, gegen die man nie gespielt hat. Die Idee war deshalb, einen Server zu erstellen, mit dem sich jeder Spieler periodisch synchronisieren kann. Der Spieler lädt dabei alle seine ausgeführten Spiele mit allen benötigten Daten hoch. Die wichtigsten Daten eines Games sind:

- Die beiden Teams, die ein Spieler vom Gegner bekommen hat
- Die beiden Seeds, die ein Spieler vom Gegner bekommen hat
- Die beiden eigenen Teams, die im Spiel verwendet wurden
- Die beiden eigenen Seeds, die in einem Spiel verwendet wurden
- Alle Signaturen, die man vom Gegner erhalten hat (bescheinigen, dass die Daten wirklich vom Gegner sind und nichts verändert wurde)
- Mit dem Server Key verschlüsseltes erstes Team des Gegners und zwei verschlüsselte Seeds

Der Server führt nun die Sequenz dieser Spiele selbst aus und schaut, ob die Resultat-Teams jeden Spiels mit den Anfangs-Teams des nächsten Spiels übereinstimmen. Zudem werden alle Signaturen überprüft. Da jedes Spiel von 2

Spielern hochgeladen wird, werden auch Quervergleiche der Daten ausgeführt. So kann man erkennen, wenn ein Spieler andere Daten zum Server hochlädt, als er dem Gegner geschickt hat. Wenn ein Spieler vom Server als Cheater erkannt wird, so wird er zu einer Ban-Liste hinzugefügt. Diese Liste wird bei jedem Synchronisieren an die Spieler zurückgegeben und kann vom Spieler dazu verwendet werden, einen Cheater zu erkennen bevor ein Spiel mit ihm gestartet wird. Die Spiele werden auf dem Server auch simuliert, wenn einer der Spieler als Cheater erkannt wird, da die Version der Teams auf dem Server sonst nicht mehr der Version auf den Handys entspricht. Da aber bei einem Spiel, bei dem ein Spieler vom Server als Cheater erkannt wurde, keine Punkte verteilt werden und sich die Auswirkungen auf das Team in Grenzen halten, kann dies toleriert werden.

Wenn ein Spiel vorzeitig abgebrochen wird, so müssen beide Spieler eine Zwangssynchronisation ausführen. Erst nach dieser Zwangssynchronisation können sie weitere Spiele ausführen. Dabei wird das nicht vollständig ausgeführte Spiel zum Server hochgeladen und der Server führt das Spiel mit den verschlüsselten (mit Server PK) Daten aus. Danach wird das Resultat-Team direkt an den Spieler zurückgegeben.

Eine andere Aufgabe des Servers ist es, eine globale Rangliste zu erstellen. Dabei werden für jedes Spiel, bei dem niemand als Cheater erkannt wurde, Punkte an die Spieler verteilt. Aufgrund dieser Punkte und der Tordifferenz kann dann vom Server der Rang jeden Spielers berechnet werden (Siehe 8.8).

Der Server archiviert alle hochgeladenen Spielinformationen der Spieler. So kann zu jeder Zeit vom Server der Spielablauf auf jedem Geraet von Anfang an nachvollzogen werden. Dies koennte verwendet werden, um im Nachhinein offline nach speziellen Mustern in den abgelaufenen Spielen zu suchen, um zum Beispiel potentiell cheatende Koalitionen zu erkennen. Es waere auch moeglich anhand dieser Daten die Simulation des Fussballspiels ausgewogener zu implementieren, wenn man zum Beispiel sieht dass es im Durchschnitt zu viele Tore gibt.

8.2 Modell der schummelnden Spieler

Ein Spiel wird vom Server korrekt ausgewertet und die Cheater erkannt, falls keine Koalitionen von 2 oder mehreren Spielern auftreten. Es ist im Fall von Koalitionen prinzipiell für eine 3. Instanz nicht mehr möglich, im Nachhinein einen Cheater zu erkennen. Zudem sind beliebige Verbindungsabbrüche beim Spielen sowie auch beim Synchronisieren mit dem Server erlaubt. Von jedem Spieler wird erwartet, dass er sich mindestens einmal im Monat mit dem Server synchronisiert, ausser wenn er seit dem letzten Synchronisieren keine neuen Spiele angefangen hat (sonst kann er gebannt werden). Ein Spieler kann beliebige falsche Daten zum Server hochladen. Er wird dann, falls prinzipiell möglich, als Cheater erkannt und gebannt.

8.3 Registration

Siehe 7.4

8.4 Synchronisation

Siehe 7.5

8.5 Verifikation

Alle Spiele, die beim Synchronisieren hochgeladen werden, werden in einer Datenbank abgelegt. Dabei hat man für jedes Spiel je 2 separate Datensätze. Die einen Daten sind vom einen Spieler hochgeladen worden und die anderen Daten vom anderen Spieler. Beide Datensätze beschreiben das Spiel vollständig und enthalten im Allgemeinen die gleichen Daten, ausser wenn jemand gecheatet hat. Die Verifikation der Spiele geschieht nicht synchron mit dem Hochladen der Spiele, da auf das Hochladen desselben Spiels vom Gegner gewartet werden muss. Periodisch (im Moment alle 20s) wird vom Server eine Verifikationsrunde dieser in einer Datenbank abgelegten Spiele ausgeführt.

Dabei wird ein Spiel vom Server simuliert. Das erhaltene Team wird am Schluss in der Datenbank abgelegt, damit man beim nächsten Verifizieren das vom Server berechnete Team mit dem entsprechenden vom Handy berechneten Team vergleichen kann. Jedes Spiel wird einzeln simuliert, das heisst einmal für den einen Player und ein zweites mal für den anderen Player. Die Änderungen an der Datenbank werden erst nach dem fehlerfreien Durchgang des Verifiers committet, damit man immer konsistente Daten hat.

Sind nicht alle Daten da (d.h. das Spiel wurde auf den Handys abgebrochen), wird mit den von den Handys mit dem server public key verschlüsselten Teams und Seeds simuliert. Diese Daten sind immer da, da sie vor dem eigentlichen Start des Spiels ausgetauscht werden.

Es werden alle Daten möglichst vielseitig überprüft (Signaturen, Vergleiche mit Datenbank, Quervergleiche mit anderem Game). Schlägt eine Überprüfung fehl, so wird der schuldige Spieler gebannt. Dabei ist immer derjenige Spieler schuld, der nachweislich Daten gefälscht hat. Das Game wird aber trotzdem weiter gespielt, damit die Teamversionen in der Datenbank konsistent mit den Teamversionen auf dem Handy bleiben. Es werden aber keine Punkte gutgeschrieben, falls jemand schummelt.

Da gemäss des '2-Armeen-Problems' (vgl. Appendix A) ein Spieler nicht entscheiden kann, ob der andere ein Spiel angefangen oder nicht, muss man

sehr vorsichtig damit umgehen, Spieler zu bannen, die ein Spiel nicht hochladen (Timeout nach 1 Monat). Es kann sein, dass das Spiel aus ihrer Sicht gar nie angefangen hat. Es ist aber bis zu einem gewissen Grad möglich mit den Daten, die der Gegner hochlädt, zu beweisen, dass der Spieler das Spiel angefangen hat (Bestätigungen der Signaturen).

Das Flussdiagramm (8.1) veranschaulicht vereinfacht die Schritte, die bei einem Durchgang des Verifiers ausgeführt werden. Die Bezeichnung `fteam` steht dabei für das Team des Gegners.

Erklärung der Schritte:

1. Prüfen, ob noch ein Game in der Datenbank ist, das an der Reihe ist.
2. Prüfen, ob der minimale Datensatz, der vor eigentlichem Beginn des Spiels ausgetauscht wird, hier ist. Verifizierung der Signaturen. Überprüfen des 1. Teams, das der Spieler hochlädt, mit Hilfe des in der Datenbank gespeicherten Teams. Falls Daten fehlen oder Signaturen falsch sind, so wird der Spieler gebannt.
3. Es wird geprüft, ob alle Spiele des anderen Spielers bis eins vor dieses Spiel schon vom Verifier bearbeitet worden sind. Falls dies der Fall ist, so ist das Team des anderen Spielers in der Datenbank genug weit simuliert und kann verwendet werden. Falls man schon einen Monat auf den anderen Player wartet, so wird er gebannt und es wird weiter gemacht.
4. Es wird geprüft, ob das andere Game vom anderen Spieler schon hochgeladen ist. Dies wird benötigt, um diverse Qürvergleiche zu machen. Ist es nach einem Monat immer noch nicht hier, so wird weiter gemacht und der andere Spieler (falls nachgewiesen werden kann, dass er das Spiel angefangen hat) gebannt.
5. Es wird mit Hilfe des durch den Gegner eingereichten Games der Punkt bestimmt, der der 1. Spieler mindestens erreicht hat im Ablauf des Spiels. So kann der 1. Player später gebannt werden, falls er nachweislich Daten nicht hochgeladen hat.
6. Es wird geprüft, ob das Spiel den Zustand nach Austausch und Bestätigung des 1. Teams erreicht hat. Falls Daten falsch sind, so wird der fehlerhafte Spieler gebannt und der Speicherpunkt als nicht erreicht betrachtet.
7. Es wird geprüft, ob das Spiel den Speicherpunkt nach Erhalt des 1. Seeds erreicht hat. Falls Daten falsch sind, so wird der fehlerhafte Spieler gebannt und der Speicherpunkt als nicht erreicht betrachtet.
8. Es wird geprüft, ob das Spiel den Speicherpunkt nach Erhalt des 2. Teams erreicht hat. Falls Daten falsch sind, so wird der fehlerhafte Spieler gebannt und der Speicherpunkt als nicht erreicht betrachtet.

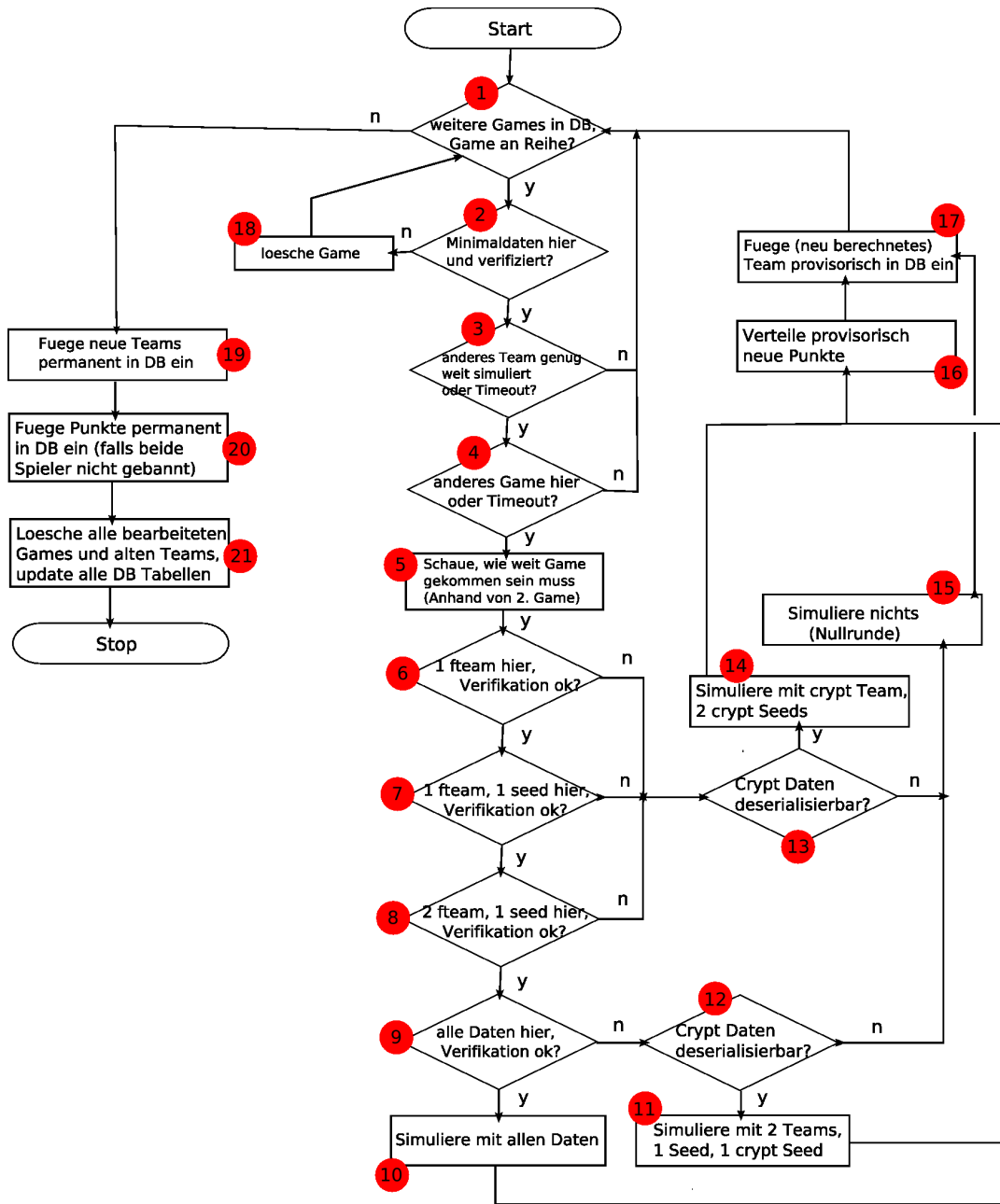


Abbildung 8.1: Das Flussdiagramm des Verifiers

9. Es wird geprüft, ob das Spiel den Speicherpunkt nach Erhalt des 2. Seeds erreicht hat. Falls Daten falsch sind, so wird der fehlerhafte Spieler gebannt

und der Speicherpunkt als nicht erreicht betrachtet. Falls die Daten okay sind, so wurde das Spiel komplett von diesem Spieler ausgeführt.

10. Es wird mit vollständigen Daten simuliert. Falls das durch den Gegner eingereichte Game hier ist, werden Qürvergleiche der Teams ausgeführt und evtl. Spieler gebannt. Das 2. Team der 2. Halbzeit wird mit dem erhaltenen Team nach der 1. Halbzeit verglichen.
11. Es wird mit den zwischen den Handys ausgetauschten Teams und dem 1. ausgetauschten Seed simuliert. Der 2. Seed wird von den verschlüsselten Seeds genommen. Falls das durch den Gegner eingereichte Game hier ist, werden Qürvergleiche der Teams ausgeführt und evtl. Spieler gebannt. Das 2. Team der 2. Halbzeit wird mit dem erhaltenen Team nach der 1. Halbzeit verglichen.
12. Es wird geprüft, ob die verschlüsselten Daten überhaupt deserialisierbar sind. Falls nicht, so wird der 2. Spieler gebannt, da er sie verschlüsselt hat.
13. Es wird geprüft, ob die durch den Server entschlüsselten Daten überhaupt deserialisierbar sind. Falls nicht, so wird der 2. Spieler gebannt, da er sie verschlüsselt hat.
14. Es wird nur mit den durch den Server entschlüsselten Daten simuliert. Das ganze Spiel wird mit pro Seite je einem Team simuliert, das heisst in der Pause können keine Spieler ausgewechselt werden. Falls das durch den Gegner eingereichte Game hier ist, werden Qürvergleiche der Teams ausgeführt und evtl. Spieler gebannt.
15. Es wird nichts simuliert, d.h das Team bleibt unverändert.
16. Es werden provisorisch Punkte berechnet und provisorisch in die Datenbank eingefügt. Falls später keiner der beteiligten Spieler gebannt ist, werden die Punkte definitiv auf das Punktekonto gutgeschrieben.
17. Das Team, das man nach der 2. Halbzeit erhält (berechnet), wird provisorisch (unsichtbar für den Verifier) in die Datenbank eingefügt. Dies ist nötig, damit die Teams nicht ändern während die Verifikation noch in Gang ist. Wenn man das 2. zugehörige Game simulieren will, braucht man noch die alte Version des Teams.
18. Lösche ein Game aus der Datenbank. Games, die nicht die minimalen Daten vorweisen können (mit allen Signaturen korrekt), werden als nicht hochgeladen betrachtet.
19. Nach dem Durchlauf durch alle Games werden die provisorisch gespeicherten Teams sichtbar gemacht und permanent gespeichert.
20. Falls beide Spieler des Games nicht gebannt sind, so werden die Punkte permanent zur Datenbank hinzugefügt. Dies ist nötig, da man erst am Schluss eines Durchgangs des Verifiers alle gebannten Spieler kennt.

21. Es werden alle nicht mehr benötigten Daten (wie alle abgearbeiteten Games, alte Teams) aus der Datenbank gelöscht. Es werden auch alle Tabellen geupdated, die aktualisiert werden müssen.

8.6 Future work

- Bitfehler bei der Synchronisation werden nicht berücksichtigt. Man könnte einen Fehlerkorrektur-Algorithmus einbauen, damit man mit höherer Wahrscheinlichkeit die richtigen Daten auf den Server hochlädt. Im schlimmsten Fall kann ein Spieler gebannt werden, wenn seine Daten Bitfehler enthalten (Signaturen können evtl. nicht mehr zu den Daten passen). Da der Server die Daten asynchron verarbeitet, ist es nicht ohne weiteres möglich, die Daten einfach erneut hochzuladen nach einem Fehler bei der Verifikation.
- Es ist zur Zeit möglich, mehrere Teams pro SIM-Karte (auf verschiedenen Mobiltelefonen) zu erstellen. Dies ist nicht wünschenswert, da dann eine als Cheater erkannte Person ohne Probleme ein neues Team erstellen und registrieren kann. Eine relativ einfache Gegenmassnahme wäre es, wenn man vor der Registrierung eines Teams einen Registrierungscode per SMS anfordern müsste. Diesen Code müsste man dann beim Registrieren als Parameter mitgeben. Pro Handy Nummer würde der Server nur einmal einen Code zurück schicken. Man könnte also pro SIM Karte nur genau ein Team registrieren.
- Im Moment werden Spiele nicht rückgängig gemacht, wenn einer der Spieler gecheatet hat. Man könnte implementieren, dass alle Spiele, die von einem Spieler abhängig sind, der als Cheater erkannt wurde, vom Server rückgängig gemacht werden. Beim Synchronisieren würde man dann sein Team erhalten, wie es der Server ohne Cheater berechnet hat. Im Worst Case müsste man jedoch fast alle Spiele rückgängig machen. Besonders mühsam wäre dies für Spieler, die lange nicht mehr synchronisiert haben und am Anfang gegen einen Cheater gespielt haben. Alle ihre Spiele würden rückgängig gemacht und die Mannschaft in den Ursprungszustand zurückgesetzt.
- Eine Begrenzung auf n Spiele bis zur nächsten Synchronisation könnte eingebaut werden, damit die Daten im allgemeinen innert kurzer Frist auf dem Server sind.
- Mit den archivierten Daten aller Spiele könnten verschiedene Analysen (Koalitionen erkennen...) gemacht werden. Diese könnten jeweils Offline auf einer anderen Maschine ausgeführt werden oder zum Beispiel in der Nacht, wenn der Server nicht belastet ist.

8.7 Banlist

Der Server unterhält eine Liste, in der alle Spieler, die als Cheater erkannt worden sind, aufgeführt sind. Diese Liste wird beim Synchronisieren jeweils an die Spieler verteilt, damit sie später herausfinden, ob ein Mitspieler ein Cheater ist oder nicht.

8.8 Server Rangliste

Nach jedem Spiel erhalten die beteiligten Spieler Punkte auf ihr Punktekonto gutgeschrieben. Das Punktesystem ist dabei ELO, wie schon auf der Handy Seite für die lokale Rangliste verwendet wird (siehe 4.4). Der neue ELO-Punkttestand wird nur permanent gespeichert, wenn keiner der beiden Spieler als Schummler ertappt wird. Sonst wird der Punkttestand nicht geupdated. Beim Synchronisieren bekommt ein Client seine aktuelle Position in der globalen Rangliste vom Server. Haben zwei Spieler den gleichen Punkttestand, so hat derjenige mit dem besseren Torverhältnis den besseren Rang.

8.8.1 Future work

- Im Moment werden immer gleich viele Punkte verteilt, egal wie viel mal man gegen einen bestimmten Spieler gespielt hat. Es wäre wünschenswert, wenn man weniger Punkte bekommt, wenn man sehr viele Male hintereinander gegen den selben Spieler spielt. So könnte man die Rangliste vor cheatenden Koalitionen schützen, die immer wieder gegen sich spielen und immer den einen Spieler gewinnen lassen.

8.9 Datenbank Tabellen

Folgende Tabellen werden vom Server in der init Methode erstellt:

- musers: **enthält** Daten zu den Registrierten Benutzern.
- mteams: **enthält** das aktuelle Team eines Users (so wie es beim letzten Synchronisieren vom Server berechnet wurde).
- mhgames: **enthält** alle Games, die hochgeladen wurden und vom Verifier noch abgearbeitet werden müssen.
- mhpoints: **enthält** Punkte und Torverhältnis für jeden User. Daraus kann der Rang berechnet werden.

8.10 Test Client

Die Serverseite von Mobile Hattrick ist sehr schwer zu debuggen, wenn man nur den Handy Simulator braucht. Man muss im Simulator viele Daten im GUI

eingeben, bis man ein Spiel gespielt hat, was sehr zeitaufwändig ist. Zudem funktionierte die Simulation auf den Handys leider bis kurz vor Schluss des Projekts wegen verschiedener Bugs noch nicht.

Deshalb wurde ein auf TCP basierender Test Client implementiert. Mehrere Test Clients können in einer Testumgebung gegen einander spielen, wobei die Daten über TCP ausgetauscht werden. Dabei implementieren die Test Clients die gleichen Protokolle wie das richtige Spiel (Registrieren, playgame,...). Sie halten auch ihren State aufrecht über alle Spiele und verhalten sich (vom Server aus gesehen) genau gleich wie ein richtiger Client. Es wird jedoch nichts auf ein permanentes Speichermedium gespeichert, so dass der State eines Clients verloren geht nach Abschluss der Tests. Auch werden nach der 1. Halbzeit aus nahe liegenden Gründen keine manuellen Auswechslungen vollzogen.

Ein simuliertes Spiel läuft dabei so ab, dass ein Client in den Hosting-Zustand gesetzt wird, worauf er auf verbindende Peers wartet (in einem neuen Thread). Wenn sich ein anderer Client verbindet, dann werden die selben Protokolle wie auf den Handys abgearbeitet (in einem neuen Thread, der blockiert, bis das Spiel fertig ist). An einigen Stellen kann man bewusst Verbindungsabbrüche auslösen, worauf der Client das Socket schliesst. Die Test Clients reagieren dann, wie die Handys, mit Zwangssynchronisation. Die Zwangssynchronisation wird automatisch gestartet, wenn ein Verbindungsabbruch erfolgte. Nachdem die Clients ein paar Spiele gespielt haben, kann man einen Client mit dem Server synchronisieren. Darauf läuft auf dem Server die Verifikation der upgeloadeten Spiele und die Verteilung der neuen ELO-Punkte. Bevor man synchronisiert, kann man im Testszenario beliebige Daten eines gespielten Spiels abändern, um die Banning-Funktionalität des Servers zu testen.

Die wichtigsten Funktionen, die der Test Client (`serverht.main.TestClient`) anbietet, sind folgende:

- `startHosting()`: setzt den Clienten in den Hosting-Zustand, worauf er auf Port `6000+playerid` auf Verbindungen wartet.
- `connectToHost(int playerid)`: Verbindet sich zu einem Host auf Port `6000+playerid`. Startet automatisch das Spiel.
- `sync()`: Der Client synchronisiert sich mit dem Server und lädt alle Spiele hoch.
- `register(String nickname)`: Der Client registriert sich beim Server.
- `performGame()`: Hier ist die Spiel-Logik implementiert, die bei einem Spiel von Host und Peer ausgeführt wird.

Eine kleine Testumgebung mit 3 Clienten, die mehrere Szenarien abarbeitet und die Funktionsweise des Test Clients veranschaulicht, ist in der Main-Methode des Test Clients implementiert. Die Main Methode braucht dabei als Input-Parameter den Port des Servers und die Adresse des Servers.

Kapitel 9

Installation

9.1 Builden des Spiels

Um das Midlet auf einem Mobiltelefon installieren zu können, muss es zuerst in ein jad- und ein jar-File gepackt werden. Dazu kann das Eclipse Plugin EclipseME [5] verwendet werden. Aufgrund der gerätespezifischen Eigenschaften muss in `mobileht.Devices` das Feld `device` auf das jeweilige Gerät gestellt werden, für das das Spiel kompiliert werden soll.

9.1.1 Obfuscation

Um das jar-File zu verkleinern kann ein sogenanntes „Obfuscated Package“ erstellt werden. Es braucht dann weniger Speicherplatz, der auf mobilen Geräten nicht im Überfluss vorhanden ist, und die Performance wird verbessert. Zusätzlich wird die Sicherheit erhöht, da das dekompileieren der Klassen dadurch erschwert wird. Alle Daten (Klassennamen, Variablennamen, Funktionsnamen) werden umbenannt und alle Kommentare gelöscht. Ein Programm das diese Aufgabe übernimmt ist Proguard [11].

9.1.2 Permissions

Die Verwendung von Bluetooth, der Zugriff auf Files und die TCP-Verbindung im Mobile Hatrick-Midlet benötigt verschiedene Permissions, die im jad-File deklariert sein müssen:

- `javax.microedition.io.Connector.socket`
- `javax.microedition.io.Connector.bluetooth.client`
- `javax.microedition.io.Connector.bluetooth.server`
- `javax.microedition.io.Connector.file.read`
- `javax.microedition.io.Connector.file.write`

9.1.3 Signatur

Mobile Hatrick verwendet verschieden Ressourcen des Mobiltelefons:

- Bluetooth Adapter
- TCP Connection
- File Access

Diese Fähigkeiten ermächtigen ein Midlet auch, Schaden anzurichten. Deshalb muss bei jedem Zugriff auf eine dieser Ressourcen eine Bestätigung gegeben werden. Da Mobile Hatrick diese Funktionen ziemlich exzessiv verwendet, ist diese Lösung nicht praktikabel. Verschiedene Mobiltelefone erlauben sogenannten „untrusted“Midlets nicht einmal, auf Files zuzugreifen.

Deshalb muss das Mobile Hatrick Midlet signiert werden. Es bekommt dann den Status „trusted“, und die Bestätigungsmeldungen können ausgeschaltet werden.

9.2 Installation auf den Handys

Nach dem builden des Spiels erhält man das jad- und ein jar-File des Midlets (siehe [2] für weitere Informationen zu J2ME und Midlets). Das Midlet kann dann entweder aus dem Internet heruntergeladen oder lokal installiert werden. Für letzteres kann Bluetooth, USB oder Infrarot verwendet werden. Die Hersteller der Mobiltelefone stellen dazu Programme bereit, die Daten zwischen Computer und Mobiltelefon transferieren können. Da uns Nokia und Sony Mobiltelefone zur Verfügung standen, haben wir Programme zu diesen Marken gesucht und benutzt:

Nokia Nokia PC Suite [9]

Sony Ericsson Sony Ericsson SDK [10]

9.3 Builden des Servers

Im Verzeichnis `MobileHTServer` befindet sich das Ant Build File `build.xml`. Das Target `jar` baut die Projekte `MobileHTServer` und `Mobile Hatrick` und erstellt das File `server.jar`.

9.4 Installation des Servers

1. Installation von MySQL, downloaden des Mysql-Connectors fuer Java.
2. Builden des Servers (`server.jar`).

3. Kopieren des .jar des Servers an den gewuenschten Ort.
4. Kopieren des .jar des MySql Connectors an den selben Ort. Das File muss den Namen `mysql-connector.jar` haben, damit es ohne Aenderungen des Manifests in der .jar Datei des Servers funktioniert.
5. Starten des Servers: `java -jar server.jar <port> <Datenbank Verbindung> [<init>]`. `<port>` ist der Port, auf den die Handys verbinden. `<Datenbank Verbindung>` ist ein String wie `jdbc:mysql://dgc.ethz.ch/db/lab06?user=username&password=pass`. Mehr dazu ist in der Dokumentation des MySql Connectors zu finden. Wenn die Option `-i` in `<init>` angegeben ist, so werden schon existierende Tabellen beim Start des Servers geloescht und alle Tabellen neu erstellt.

Kapitel 10

Erfahrungsberichte

10.1 Andri Toggenburger

Die Idee, ein Spiel auf Mobiltelefonen zu programmieren gefiel mir von Anfang an sehr gut, da ich noch nie etwas auf Mobiltelefonen programmiert hatte. Das Projekt, ein Multiplayerspiel mit Kommunikation über Bluetooth von Grund an neu zu entwickeln, hörte sich sehr spannend an, da es bis jetzt nicht viele Multiplayerspiele auf Handys gibt. Auch ist es interessant, ein GUI auf sehr beschränktem Platz zu implementieren. Natürlich reizten mich auch die Probleme, die in einer Umgebung ohne zentralen Server während dem Spiel auftreten können und deren Lösung.

Am Anfang stand noch nicht fest, was für ein Spiel wir implementieren wollten. Zuerst tendierten wir Richtung Poker, aber die Probleme, die beim Poker auftreten können, sind schon gelöst. Weit interessanter ist es, ein Spiel zu implementieren, bei dem der State nach Ende einer Spielrunde erhalten bleibt und als Input für die nächste Spielrunde gebraucht wird, ohne dass der State auf einem zentralen Server abgespeichert werden muss. Hier treten sofort viele neue Probleme auf, die es zu lösen gilt. So muss man vermeiden, dass ein Spieler seinen State (=Team) zu seinen Gunsten abändert, um in der nächsten Runde einen Vorteil zu haben. Auch gilt es, eine Rangliste zu erstellen, ohne dass man von Anfang an weiss, wer gegeneinander spielt. Weitere Probleme stellen sich, da ein Handy auch während eines Spiels abstürzen kann oder die Verbindung getrennt wird. Ohne eine 3. Instanz, die den Spielablauf überwacht hat man zudem mit dem **2-Armeen Problem** zu kämpfen (vgl. Appendix A), so dass ein Spieler nicht sagen kann, welchen State der andere Spieler genau hat.

Nach etwa einer Woche stand fest, dass wir Mobile Hattrick programmieren. Thibaut hatte sich im Voraus schon viel zu Mobile Hattrick überlegt, was zu einem schnellen Start des Projektes führte. Andererseits gab es auch einige Differenzen zu lösen, da nicht alle mit allem einverstanden waren. So mussten

wir z. B. entscheiden, ob ein Team neue Stärkepunkte bekommen kann oder ob eine fixe Anzahl von Stärkepunkten am Anfang vom Server verteilt werden und dann nur zwischen den Spielern verschoben werden (d. h. keine neuen Punkte erzeugt werden).

Doch nach kurzer Zeit stand ungefähr fest, wie das Spielprinzip ausschaute. Jeder hatte sich schon Gedanken über das eine oder andere Problem gemacht bzw. hatte ein Lieblingsgebiet und so war die Aufteilung in verschiedene Gebiete, die dann von je einer Person bearbeitet wurden, ziemlich einfach. Ich entschied mich für das GUI von Mobile Hattrick.

Von nun an programmierte jeder alleine an seinem Teilgebiet. Wir trafen uns als ganzes Team normalerweise einmal in der Woche, um grundsätzliche Probleme zu diskutieren. Da ich als Programmierer des GUI Berührungspunkte mit den anderen Teilgebieten hatte (vom GUI ruft man die Funktionalität der anderen Teilgebiete auf) traf ich mich bei Problemen mit den jeweiligen Gruppenmitgliedern oder diskutierte Probleme via e-mail. Dies hat eigentlich ziemlich gut funktioniert. Zum Teil musste ich natürlich GUI Elemente später wieder abändern, da gewisse Funktionalität nicht oder auf andere Art und Weise als am Anfang gedacht implementiert wurde und so zu einem anderen GUI führte. Zuerst wollte ich das GUI mit den vorgefertigten GUI Elementen von MIDP 2.0 implementieren. Da Fullscreen mit diesen aber nicht möglich ist und alles sehr „grau in grau“ausschaut, entschied ich mich schnell, alle Klassen von Grund auf mit geometrischen Primitiven auf einer Zeichnungsfläche zu zeichnen. Dies war auch ohne grössere Probleme möglich. Nur die Kompatibilität mit einigen Geräten, die Bugs in bestimmten Funktionen haben, musste speziell behandelt werden.

Eine Schwierigkeit bei der Programmierung der GUI Elemente war zudem, dass man sie nicht immer leicht debuggen konnte. Ich musste zum Teil warten, bis die Spiel-Funktionalität hinter dem GUI fehlerfrei funktionierte. Zum Teil entdeckte ich so auch Bugs im Code der anderen Gruppenmitglieder und machte diese darauf aufmerksam.

Nach etwas mehr als einem Monat waren die meisten GUI Elemente mit ihrer Logik fertig. Nun widmete ich mich der Server Seite von Mobile Hattrick. Thibaut hatte schon ein Grundgerüst für den Server implementiert, das ich dann erweiterte. Meine Aufgabe war es insbesondere, den Verifier zu implementieren. Der Verifier führt auf der Serverseite alle Spiele aus und bannt Spieler, die er als Cheater erkennt. Weiter verteilt er Punkte je nach Ausgang eines Spiels. Das Erste stellte sich als schwereres Problem heraus, als ich am Anfang dachte. Während der Entwicklung traf ich mich immer wieder mit Thibaut, der den Gameflow auf den Handys sowie das Protokoll zur Synchronisierung mit dem Server programmierte. Viele Male wurden neue Probleme und mögliche Angriffsszenarien erkannt und das Protokoll geändert, um diese zu lösen. Darauf musste ich jeweils auch wieder viel umstellen oder

neu programmieren im Verifier. Etwa 3 Mal mussten wir grössere Änderungen am Protokoll vornehmen, was auch praktisch zu einer jeweiligen Neuprogrammierung des Verifiers führte. Zuerst wollten wir zulassen, dass man nach einem Verbindungsabbruch bei einem Spiel einfach ein neues Spiel starten kann. Dies stellte sich aber als extrem kompliziert und fehleranfällig heraus, so dass wir Zwangssynchronisierung nach einem Spielabbruch einführen mussten.

Nach einer längeren Diskussion mit Thibaut und Michael K. entstand ein neues Protokoll, bei dem die Handys zuerst einen minimalen Datensatz austauschen, bevor das Spiel als angefangen gilt. Dabei werden auch Daten verschlüsselt mit dem Server Public Key ausgetauscht, die dann vom Server für die Berechnung eines abgebrochenen Spiels entschlüsselt werden können. Zudem muss man nun nach einem Verbindungsabbruch eine Zwangssynchronisation ausführen. Man erhält dabei das Team zurück, wie es der Server mit den verschlüsselten Daten (die immer da sind) zu Ende berechnet. Unglücklicherweise hat die Simulation eines Spiels bis kurz vor Schluss des Projekts nicht vollständig funktioniert und so war es sehr schwer, den Verifier zu testen.

In der letzten Woche erstellte ich deshalb einen Test Clienten, der alle Protokolle wie Synchronisation etc. implementiert. So kann man die Server Seite des Projekts völlig ohne Handys automatisch testen. Man kann verschiedene Test Clients über TCP gegen einander spielen lassen und erhält dann die gleichen Resultate wie wenn man mit den Handys spielen würde. Mit diesen Test Clients erstellte ich noch eine kleine Testumgebung, wo verschiedene (Fehler) szenarien abgearbeitet werden. So kann man überprüfen, ob der Server richtig funktioniert.

Alles in allem war dieses Projekt eine sehr interessante Erfahrung für mich. Zum einen galt es knifflige Probleme zu lösen, vor allem beim Verifier. Zum anderen war es interessant, in einem zusammengewürfelten Team zu arbeiten. Auf viele Ideen oder Lösungen eines Problems wären wir wahrscheinlich alleine erst viel später oder gar nicht gekommen. Der Austausch im Team stellte sich als extrem wertvoll heraus.

10.2 Thibaut Britz

10.2.1 Aufgabe

Da kurzerhand der Algorithmus Track für den ich mich anmelden wollte, gestrichen wurde, musste ich mich nach einem Lab umsehen. Ich entschied mich dann mit Michael für das Mobile Hattrick Lab, da mir die anderen Projekte nicht besonders zusprachen, und mir das Mobile Hattrick Lab ganz interessant erschien. Später stiessen dann noch Roger und Andri dem Projekt hinzu, die ich vorher noch nicht kannte.

Die ersten zwei Wochen nutzen wir um das Spielkonzept zu erstellen. Nach anfänglichen Schwierigkeiten, haben wir uns dann auf das Mobile Hattrick Spiel geeinigt, und ein Spielkonzept erstellt. Zusätzlich wurde von mir ein für alle online editierbares Dokument erstellt um die Kommunikation zu erleichtern und Konzepte aufzuschreiben. Die spätere Aufgabenteilung ergab sich dann nach Interessen der jeweiligen Personen, deren Kenntnisse und Motivation.

Mein erster Teil der Arbeit bestand darin sich um die Kommunikation, die übertragenen Daten und die Abspeicherung dieser Daten, zu kümmern. Ein Problem war, dass mein Teil sehr von den Teilen der anderen Gruppenmitglieder abhing, und deswegen lange nicht getestet werden wurde (z.B. funktionierte die Simulation und die Team Generierung leider erst Ende der zweitletzten Woche), und ich immer wieder auf andere warten musste.

Zum einen musste ich mit Andri und Michael zusammenarbeiten, um meinen Teil im GUI unterzubringen. Zusätzlich kümmerte ich mich mit Michael um die Abspeicherung der Daten. Er erstellte auch das Bluetooth Socket, auf dem dann Später meine Client-Client Kommunikation lief. Bei Roger's Teil kümmerte ich mich um die Erstellung und Abspeicherung der Spielklasse, die alle Informationen enthält, die Roger für die Simulation des Spiels benötigt.

Da mein Teil von so vielen Veränderungen beeinflussbar gewesen ist, kam es auch machmal vor, dass etwas nicht mehr ging da eine andere Person etwas anderes geändert hat. In der Zeit wo ich warten musste, erstellte ich dann den Server wo Andri dann später den Verifier implementierte. Im Gespräch mit Andri, was leider erst in der letzten Phase des Projekts erfolgte, zeigten sich dann auch viele Schwächen im Protokoll, an die keiner von uns am Anfang dachte. So einigten wir uns dann in der zweitletzten Woche auf eine neue Version die die möglichen Angriffspunkte eines Cheaters aufs Mindestmass reduzierte, indem wir eine Zwangssynchronisation nach Spielabbuch einführten. Die letzte Woche vor der Präsentation nutzen Andri und ich dann um die restlichen Bugs im MIDlet zu fixen.

10.2.2 Probleme und Erfahrungen

Zeitaufwendig und manchmal etwas frustrierend, waren die diversen Bugs, auf die ich im Rahmen meiner Programmierstätigkeit traf:

- Thread Problem: Sobald man versuchte in irgend einer Weise auf einen Thread zuzugreifen (z.B. um zu schauen ob dieser fertig ist), blockierten die Connector Aufrufe in diesem Thread und es gab einen Deadlock. Der Code lief aber einwandfrei falls man den Code als eigenes MIDlet implementierte. Schlussendlich haben wir die Connector Objekte dann direkt beim Start den MIDlet erstellt, was dann funktionierte.
- Diverse Handy Bugs: Die Nokia Handys machen eine TCP Connection auf, senden jedoch nichts über die Verbindung. Bei den Sony Handys muss man alle 256 Bytes eine Pause einlegen (z.B. selbst ein ACK Byte schicken), sonst , so scheint es, überschreibt man den lokalen Buffer. Das Gleiche gilt auch für die Bluetooth Connection. Leider sind diese Fehler nirgends dokumentiert, und man bekommt auch wenig Hilfe in den jeweiligen Support Foren des Herstellers.
- Bouncy Castle API: Zur Verschlüsselung benutzen wir das Bouncy Castle API [12]. Auch hier gab es mehrere Probleme. Zum einem muss der Schlüssel über 1024 Bits gross sein, sonst scheitert die Signaturerstellung. Zum anderen dürfen die Blöcke, die man verschlüsseln will, nicht mit einem Null Byte anfangen. Ist dies der Fall, schlägt entweder die Entschlüsselung später ganz fehl, oder man bekommt ein unvollständiges Array zurück (ein paar Anfangsbytes fehlen einfach). Ausserdem funktionierte die Verschlüsselung erst auf den Handys nachdem man die Jar Datei mit Proguard obfuskerte (Sonst gab es immer einen `java.lang.Error` Fehler). Im Simulator funktionierte es aber auch ohne Obfuskator.

Zu keinem der Probleme liessen sich im Internet Informationen finden. Zusätzlich erschwerend war, dass das Hochladen und Starten des MIDlets immer mehrere Minuten dauert. Ausserdem konnte man die fehlerhaften Stellen, ohne viel Aufwand zu betreiben, nur schwer automatisieren um einen Bug zu finden, und man musste sich mühselig durch die Menus klicken. Manchmal musste man auch das Handy aus unbekanntem Gründen neu starten, da sonst das MIDlet nicht mehr richtig funktioniert, was wieder viel Zeit kostete.

Zum Glück stellt Sony Ericsson den Entwicklern ein Tool bereit über das man die Konsolen Nachrichten auf dem PC ausgeben kann. (Auch wenn im Falle eines Fehlers der Stacktrace leider fehlt, und nur die Art des Fehlers angezeigt wird). Bei den Nokia Handys gibt es nur die Möglichkeit Nachrichten am Display auszugeben oder in eine Datei zu schreiben, die man dann später auf den Computer überkopieren kann.

Persönlich hat mir das Projekt viel gebracht: Ausser Uebungen in den diversen Vorlesungen hatte ich nie etwas in Java programmiert. Das Projekt war eine grossartige Gelegenheit etwas Programmiererfahrung in Java,

Teamprogrammierung, und Handyentwicklung zu bekommen. Auch musste ich Kompromissbereitschaft zeigen. Je mehr Leute an einem Projekt arbeiten, desto mehr verschiedene Meinungen gibt es, und so schwieriger ist es sich auf einen Kompromiss zu einigen, oder zu versuchen die Anderen von seiner Meinung zu überzeugen ohne mit ihnen in Konflikt zu geraten. Am Anfang dachte ich auch das Projekt sei nicht sehr schwierig. Doch traten grade auf Handyseite (Bugs) und in der Kommunikation (Protkoll) Probleme auf, mit denen ich nicht gerechnet hatte.

Es gibt aber auch Dinge die ich bei einem zukünftigen Projekt anders machen würde. Im Nachhinein glaube ich dass unsere Kommunikation etwas besser hätte sein könnte. Mehr Treffen, Kommunikation und eine klarere Definition der Ziele und Aufteilung der Arbeit hätten sicher nicht geschadet. (So hätte man vielleicht verhindern können, dass die Simulation erst sehr spät lief). Auch hätten wir mehr Zeit für die Recherche einbinden können und die Konzepte etwas besser überdenken sollen. Man hätte meiner Meinung nach mit j2me polish eine ganze Menge Arbeit sparen können, wenn man im Vorfeld darauf gestossen wäre. Oder wenn wir die Protokolle am Anfang etwas detaillierter ausgearbeitet hätten, so dass wir sie nachher nicht hätten umändern müssen. Auch hat am Ende keiner von uns richtig Gebrauch vom Writely Dokument für Konzepte und Ideen mehr gemacht.

10.2.3 Fazit

Trotz all der Probleme war es eine gute Erfahrung das Lab gemacht zu haben, die mir gezeigt hat, dass etwas, was am Anfang leicht aussieht, später doch schwieriger sein kann, als im Vorfeld gedacht. Sehr gut fand ich die Unterstützung der DCG Gruppe auf Hardware Seite und Betreuungsseite, wo sich der Assitent auch nicht scheute, mir bei der Handy Bug Suche zu helfen.

10.3 Michael Lorenzi

10.3.1 Hintergrund und Motivation

Um den Major im Bereich Distributed Systems abzuschliessen habe ich im Frühjahr 2006 ein Lab gesucht. Das ausgeschriebene Projekt „Mobile Hattrick“ hat mich aufgrund meiner Vorkenntnisse besonders angesprochen. Im letzten Semester hatte ich in der Vorlesung „Mobile System-Architekturen 1“ erste Erfahrungen mit der Programmierung mobiler Geräte gemacht. In den Vorlesungen „Kryptographische Protokolle“ und „Information Security“ lernte ich einige Grundlagen im Bereich Kryptographie. Und auch weil ich schon seit längerem passionierter Hattrick-Spieler [3] bin, ist mir die Idee, Hattrick für mobile Geräte zu entwickeln, sofort sympathisch gewesen.

10.3.2 Aufgaben

Aufgrund meiner Vorkenntnisse mit der Programmierung mobiler Geräte habe ich mich dazu entschieden, den low-level-Teil des Projekts zu übernehmen. Das heisst:

- Kommunikation über Bluetooth
- Persistente Speicherung von Spieldaten

Zum gemeinsamen Arbeiten am Code habe ich ein CVS-Repository mit einem Muster-Midlet erstellt. Um unsere Ideen schriftlich festzuhalten, hat Thibaut zusätzlich ein Dokument im Writely [4] erstellt, das alle in Echtzeit editieren konnten.

Das Aufsetzen der Entwicklungsumgebung für Midlets kannte ich schon von der erwähnten Vorlesung, deshalb konnte ich den anderen eine Anleitung dazu geben. Wir verwendeten das Eclipse-Plugin EclipseME [5] und das Wireless Toolkit 2.3 von Sun [6].

Um erste Versuche mit Bluetooth zu machen, musste ich zuerst ein einfaches GUI erstellen, mit den `lcdui`-Elementen von MIDP 2.0. Die Internetseite `benhui` [7] hat mir bei den ersten Schritten mit Bluetooth sehr geholfen. Als ich mich zunehmend mit den verschiedenen Geräten und ihren Eigenheiten beschäftigen musste landete ich oft auf den Community-Seiten von Nokia [8] und Sony Ericsson [10]. Viele Probleme mit den einzelnen Geräten sind schon in diesen Foren besprochen und gelöst worden. Dennoch verwendete das Debugging auf den Geräten enorm viel Zeit und Nerven.

Zum späteren Zeitpunkt des Projekts, als meine Bereiche einigermaßen stabil liefen, konnte ich mich auch weiteren Aufgaben widmen:

- Menus für den Spielablauf
- Einbindung der Simulation in das GUI
- Allgemeines Debugging
- Aufsetzen des Latex-Frameworks für die Dokumentation

10.3.3 Probleme und Erfahrungen

Installation auf den Mobiltelefonen: Beim Deployment und der Installation eines Midlets auf ein Mobiltelefon sind schon viele Schwierigkeiten aufgetreten. Die PC-Mobiltelefon Verbindungsprogramme der Hersteller Nokia und vor allem Sony Ericsson scheinen noch nicht perfekt ausgereift zu sein. Ein anderes Problem war das jad-File, das alle Requirements und Permissions des Midlets beinhalten muss. Dazu müssen einige Felder richtig gesetzt sein. Das hat uns einige Kopfschmerzen bereitet bei der ersten Installation, und bei jedem verschiedenen Gerät von neuem.

Debugging: Die Möglichkeiten zum Debugging sind bei Mobiltelefonen etwas beschränkt. Es gibt keine Konsole, wie man es sich bei Desktop-Applikationen gewohnt ist. Die Software von Sony Ericsson schreibt praktischerweise den Output auf eine Konsole auf dem PC, wenn man während der Ausführung des Programms mit dem PC verbunden ist. Das war mit ein Grund (zusammen mit Problemen bei der TCP-Verbindung), dass wir uns im Laufe des Projekts auf die Sony Ericsson Mobiltelefone beschränkt haben. Diverse Handy-Bugs erschwerten die Arbeit zusätzlich.

Thread-Synchronisation: Zu Beginn des Labs, als ich noch wenig Erfahrungen mit den Problemen von Mobiltelefonen hatte, konzentrierte ich mich darauf, das Midlet im Emulator von Sun korrekt zum Laufen zu bringen. Doch schon beim Bluetooth Discovery musste ich die Erfahrung machen, dass der Emulator und die physikalischen Geräte zwei verschiedene Kapitel sind. Während beim Simulator die Device- und Service-Discovery in Bruchteilen von Sekunden abläuft, kann die gleiche Funktion auf den Geräten mehrere Sekunden dauern. Das führte zu vielen Problemen und willkürlichem Verhalten, das zu Beginn nicht nachvollziehbar war. Das Problem rührte daher, dass im Bluetooth-Layer verschiedene Threads verwendet werden, die gleichzeitig auf den Bluetooth-Adapter zugreifen. Die Lösung des Problems war es, die Service-Discoveries auf den einzelnen Geräten hintereinander ablaufen zu lassen, wenn mehrere Geräte als Kandidaten gefunden wurden.

File-System-Roots: Der File-Access muss auch in separaten Threads erfolgen, was zu ähnlichen Problemen wie bei Bluetooth führte. Zusätzlich stellte sich die Frage, auf welche Teile des Speichers der Mobiltelefone der Zugriff erlaubt ist. Nach dem Suchen auf den Herstellerseiten stellte sich heraus, dass die File-System-Roots von Hersteller zu Hersteller unterschiedlich

sind (siehe Kapitel 5). Das würde für eine Verwendung von RMS sprechen, das vollständig in MIDP 2.0 integriert ist. Davon haben wir aber aus den in Kapitel 5.1 aufgeführten Gründen abgesehen.

Teamarbeit: Zu Beginn des Projekts haben wir es etwas verschlafen, einen strikten Zeitplan aufzustellen. Zusätzlich haben wir zuwenig genau festgehalten wie die Applikation genau aussehen soll. Die Fragen kamen erst mit dem Ausprogrammieren, was natürlich mühsamer ist als es vorweg zu besprechen. Die Kommunikation innerhalb des Teams funktionierte aber mit dem Fortlauf des Projekts besser, und einmal pro Woche fand ein Meeting statt, was auch dringend notwendig war. Ich denke eine seriösere/wissenschaftlichere Herangehensweise und eine klarere Hierarchie (Bestimmung eines Chefs oder Koordinators) wäre von Vorteil gewesen.

10.3.4 Fazit

Es war das erste Mal, dass ich ein Projekt mit mehr als 2 Mitarbeitern durchgeführt habe. Dies eine lehrreiche Erfahrung für mich, die Wichtigkeit der Planung und Dokumentation wird einem sehr bewusst. Im technischen Bereich ist mir aufgefallen wie verschieden die Mobiltelefone der einzelnen Hersteller zu Programmieren sind, trotz Java-Modell. Besonders wenn man auf Hardware-Elemente zugreift wird es sehr gerätespezifisch, das hätte ich nicht erwartet, ist man sich doch von J2SE, der Desktop-Version von Java, anderes gewohnt. Die Programmierung von PC's unterscheidet sich doch stärker von der von Mobiltelefonen als ich zuerst angenommen hatte. Meine Hochachtung vor Programmierern mobiler Geräte ist auf jeden Fall gestiegen. Wahrscheinlich wird die Unterstützung von J2ME auf den Geräten und die Emulatoren und Zusatzprogramme für den PC in Zukunft besser werden, so dass sich die Lücke zwischen der Programmierung von Desktop-Computern und Mobiltelefonen weiter schliesst.

10.4 Roger Seidel

Als Andri und ich uns für ein Lab umschaute, erweckte die Idee von einem Fussballmanager auf Mobile Devices schnell unser Interesse. Nach einer kleiner Erläuterung von Michael Kuhn über die Details des Labs war für uns klar, dass wir daran teilhaben möchten. Zum Anfang stand noch die Idee im Raum, ein Kartenspiel auf dieser Basis zu machen. Da es aber schon Arbeiten über Poker auf Peer-to-Peer Basis gab, wurde die Idee schnell wieder vergessen.

Nach der ersten Teamsitzung wurde mir schnell klar, dass Michael L. und Thibaut grosses Interesse daran hatten, denn Bluetooth-Teil zu übernehmen. Da die zwei ein grösseres Wissen in diesem Bereich mitbrachten, hatte ich auch nichts dagegen einzuwenden. Als grosser Fussballfan und erfahrener Fussballmanagerspieler wollte ich unbedingt die Simulation des Spiels programmieren und da Andri Interesse am GUI hatte, fiel uns die Arbeitsteilung ziemlich leicht.

Ich informierte mich zuerst mal genauer über die Fussballsimulation Hattrick.org[3], an der ja die Idee für dieses Lab anlehnte. Zum Glück ist auf der Homepage ziemlich ausführlich beschrieben, wie die Simulation in etwa funktioniert. Also machte ich mir einmal Gedanken über die Datenstruktur die ich verwenden wollte. Die grössten Probleme hatte ich dabei bei den verschiedenen Aufstellungstypen. Ich wollte hier eine Lösung, bei der eine dynamische Änderung der Aufstellung im Spiel ziemlich einfach machbar ist. Ich entschied mich für eine abstrakte Klasse `Aufstellung`, in der alle Spieler gespeichert sind und abgeleitete Klassen für jeden Aufstellungstyp, der wie ein Raster über die Datenstruktur der Spieler gelegt wird. So war es ziemlich einfach, die nötigen Daten wie Spieler einzelner Positionen o.ä. auszulesen und weiter zu verwenden. Nachdem ich nun alle nötigen Daten wie Teams, Spieler und Aufstellung hatte, began ich mit der eigentlichen Simulation. Das Prinzip dabei ist ziemlich einfach. In einem ersten Schritt werden die beiden Mittelfelder der Teams miteinander verglichen und dabei errechnet, wie viele Chancen jedes Team pro Halbzeit hat. In einem zweiten Schritt ging es darum, die Chancen einzeln zu simulieren. Damit jede Position Einfluss auf den Ausgang einer Partie haben kann, wird die Chance folgendermassen berechnet. Zuerst werden Stürmer, Verteidiger und Torwart zufällig bestimmt, die an dieser Chance teilhaben. Danach gibt es ein erstes Duell zwischen Verteidiger und Stürmer und falls der Stürmer gewinnt, das zweite Duell zwischen Torwart und Stürmer. Gewinnt der Stürmer erneut, so hat er ein Tor erzielt. Ich hatte dabei anfänglich Probleme, dass es viel zu viele Chancen gab und das Ergebniss eher einem Eishockeyspiel als einem Fussballspiel glich. Nach dem Ausprobieren verschiedener Parameter bekam ich auch dieses Problem schnell in den Griff.

Nachdem ein Spiel schliesslich ohne Probleme simuliert werden konnte, befasste ich mich mit den Verbesserungen einzelner Spieler nach einem Spiel und nach einer Saison. Ich hatte ziemlich klare Vorstellungen davon, wie stark sich ein Spieler nach einem Spiel, bzw. innerhalb einer Saison entwickeln sollte. So hatte ich das ganze auch schnell implementiert und es traten beim Testen auch keine Probleme auf.

Abschliessend baute ich noch kleine Dinge ein wie Verletzungen von Spie-

lern, die zufällig während einem Spiel auftreten, eine Klasse die alle Statistiken speichert und ausgibt, die ELO-Punkterangliste, die aussagt wie gut sich ein User mit seinem Team schlägt und die automatische Generierung von einem Team zu Beginn der Simulation.

10.4.1 Fazit

Die Arbeit an einem grösseren Projekt hat ziemlich Spass gemacht, da man während dem Studium nicht viel Gelegenheit bekommt an etwas Grösserem zu arbeiten. Auch die Teamarbeit hat nach meiner Meinung gut geklappt, obwohl ich auch sagen muss, dass ich nicht viel mit dem Teil der Anderen zu tun hatte, da meine Arbeit ziemlich eigenständig war. Das war zwar Angenehm, da mir keiner gross drein redete, aber vielleicht auch ein wenig mühsam. Zum Einen deshalb, weil sich die Anderen nicht gross für meine Implementation interessierten, was verständlich ist, angesichts der Probleme mit denen sie sich auseinandersetzen mussten. Zum Anderen, weil ich keinen grossen Einblick bekam in die Programmierung von Mobile Devices und nicht zu jedem Zeitpunkt wusste, ob meine Vorstellung von der Simulation sich mit derjenigen der Anderen deckte. Gerne hätte ich natürlich mehr mit den eigentlichen Problemen einer Bluetoothverbindung zwischen PDA Devices zu tun gehabt um davon zu lernen, doch ich merkte schnell dass ich keinen grossen Beitrag dazu leisten konnte. Trotzdem bin ich zufrieden mit diesem Lab, da die Atmosphäre unter uns angenehm war und wir einen coolen Fussballmanager für ein Mobilephone aus dem Boden gestampft haben.

Kapitel 11

Appendix

11.1 2-Armeen-Problem

Man denke sich 2 Armeen A und B die eine Armee X angreifen wollen. Greifen sie die Armee X einzeln an sind sie zu schwach, aber wenn sie beide gleichzeitig angreifen sind sie zahlenmässig überlegen. Die Armeen A und B befinden sich aber auf 2 Anhöhen und können nur über Brieftauben miteinander kommunizieren. So müssen sie sich auf einen gemeinsamen Angriffszeitpunkt einigen. Die Brieftaube muss aber immer über die Armee X fliegen und kann von dieser abgeschossen werden. A sendet B eine Nachricht, und B sendet eine Bestätigung zurück. Dabei ergeben sich zwei Probleme

- A sendet eine Nachricht mit dem Angriffszeitpunkt, und es kommt keine Bestätigung. Jetzt weiss A nicht ob die ursprüngliche Nachricht oder die Bestätigung abgefangen wurde.
- B erhält die Nachricht und schickt eine Bestätigung. Er kann sich jetzt nicht sicher sein ob seine Bestätigung angekommen ist.

Das 2 Armeen-Problem ist *unlösbar*. Es müssten unendlich viele Bestätigungen hin- und hergeschickt werden.

Literaturverzeichnis

- [1] Distributed Computing Group
<http://www.dcg.ethz.ch>
- [2] Java for mobile phones
<http://java.sun.com/javame/>
- [3] Hattrick
<http://www.hattrick.org>
- [4] Writely, kollaborativer Texteditor
<http://www.writely.com>
- [5] EclipseME, J2ME-Plugin für Eclipse
<http://www.eclipseme.org>
- [6] Sun Wireless Toolkit
<http://java.sun.com/products/sjwtoolkit/>
- [7] Benhui
<http://www.benhui.net>
- [8] Nokia Forum
<http://www.forum.nokia.com/>
- [9] Nokia PC Suite
http://www.nokia.ch/german/support/software/pc_suite/index.html
- [10] Sony Ericsson Developer World
<http://developer.sonyericsson.com/>
- [11] Proguard, Obfuscating Tool
<http://proguard.sourceforge.net/>
- [12] Bouncycastle Crypto API
<http://www.bouncycastle.org/>