Collaborative Chess

Semesterarbeit Herbst-Winter 2009/2010

Nicolas Perrenoud

Betreuer: Michael Kuhn, Christoph Lenzen

Prof. Dr. Roger Wattenhofer

Distributed Computing Group, ETH Zürich

Inhalt

1 Einleitung ... 3
2 Related Work ... 5
3 Technischer Teil ... 8
3.1 Aufgabenstellung ... 8
3.2 Lösungsansätze ... 8
3.2.1 Kommunikation ... 8
3.2.2 Abstimmungen ... 9
3.2.3 Variantenbaum ... 11
3.2.4 Pfeile und Benutzeroberfläche ... 13
3.2.5 Formung von Teams ... 14
3.2.6 Spielmodus ... 14
3.3 Architektur des Systems ... 16
3.3.1 Technischer Hintergrund ... 16
3.3.2 Facebook ... 18
4 Verbesserungsmöglichkeiten ... 19
4.1.1 Ranking ... 19
4.1.2 Zusatzzeit ... 19
4.1.3 Interface ... 19
4.1.4 Schachcomputer ... 20
5 Zusammenfassung ... 20
6 Anhang ... 21
6.1 Verwendete Bibliotheken ... 21
6.1.1 PHP ... 21
6.1.2 JavaScript ... 21
6.2 Verzeichnisstruktur ... 21
6.3 Datenbankschema ... 22
7 Referenzen ... 23

1 Einleitung

Es war ein spannender Kampf Mensch gegen Maschine, als Deep Blue[1], der Schachcomputer von IBM, im Jahr 1997 den amtierenden Schachweltmeiser Garri Kasparow schlug. Es war der erste Wettkampfsieg eines Schachcomputers gegen einen amtierenden Schachweltmeister und der Kampf wird von einigen als das spektakulärste Schach-Ereignis der Geschichte bezeichnet. Viele Schachspieler und Technik-Enthusiasten waren via Internet live dabei, als Deep Blue das Turnier 3½ zu 2½ gewann. In der zweiten Partie gab es ein interessantes Ende: Kasparow gab das Spiel auf. Die Zuschauer im Internet, welche laufend mit dem aktuellen Spielstand beliefert wurden, diskutierten, ob Kasparow in dieser Situation anders hätte reagieren sollen. Sie begannen via Chat- und E-Mail die verschiedenen Möglichkeiten zu diskutieren, simulierten diese auf ihren eigenen Schachcomputern und fanden zusammen eine Lösung, welche Kasparow ein Unentschieden ermöglicht hätte (McGrew, 2007). Er hatte diese Variante nicht in Betracht gezogen, weil er es nicht für möglich hielt, dass der Computer eine solche taktische Möglichkeit übersehen könnte. Nach dieser vermeidbaren Niederlage war Kasparow psychologisch angeschlagen.

Zwei Jahre nach seiner Niederlage gegen Deep Blue bestritt Garry Kasparow eine Partie Schach gegen „die Welt“ (über 50‘000 Spieler). Er gewann nach 62 Zügen über vier Monate das Spiel, sagte jedoch, dass es für ihn das grösste und komplexeste Spiel der Geschichte gewesen sei und er mehr Zeit als bei jedem anderen Spiel für die Analyse verbraucht habe (Kasparov, 2000). Man kann darin sehen, dass die Gruppenleistung sehr wohl Vorteile bringen kann, ob sie aber jemals ausreichen wird um einen Grossmeister zu schlagen, wird die Zeit zeigen.

Heutzutage ist es dank der raschen Ausbreitung des Internets immer einfacher, durch computergestützte Zusammenarbeit schneller zu einem Ergebnis zu kommen. Team-Kommunikation war noch nie so einfach, und durch das Zusammentragen von Wissen entstehen riesige Datenbanken. Das Forschungsgebiet, welches sich mit dieser Thematik befasst, heisst Computer Supported Cooperative Work (CSCW), also computergestützte Zusammenarbeit. Es gibt immer mehr Anwendungssoftware und Web-Applikationen wie z.B. Groupware, Google Apps oder Microsoft Groove, welche die Konzepte aus der Forschung umzusetzen versuchen.

Wir wollen in dieser Arbeit Entscheidungsfindungsprozesse von Gruppen besser verstehen lernen und das Gruppenverhalten während einer Zusammenarbeit analysieren können. Unser Ansatz ist, dies mithilfe eines Team-Schachspiels zu ermöglichen. Das Schachspiel ist ein ideales Beispiel, um Lösungsverfahren von Problemen zu untersuchen und wurde schon oft für solche Aufgaben verwendet. Die Komplexität der Probleme kann sehr gross sein, aber

[1] http://www.research.ibm.com/deepblue/

durch die Beschränkungen durch das Brett und die Regeln (man kann zu jeder Zeit nur einen Zug machen, nur mit einer Figur und nur zu einem Ziel-Feld, Spieler wechseln sich ab usw.) kann man die Lösungen genau messen; zudem ist Schach eines der bekanntesten Brettspiele und man findet daher schnell Test-Spieler.

Das Ziel dieser Semesterarbeit ist es, eine Webapplikation, genannt „Collaborative Chess“ zu erstellen, bei der mehrere Spieler gemeinsam in Teams gegeneinander eine Partie Schach spielen. Der Fokus liegt auf kleinen Teams von 2-10 Spielern, aber auch grössere Gruppen sollen gegeneinander antreten können, ohne dass man dazu die Software stark modifizieren muss (die Modifikationen sollten sich auf die Benutzeroberfläche und eine allfällige Performance-Optimierung beschränken). Kleine Gruppen bieten den Vorteil, dass man effektiv testen und auswerten kann, daher sind sie die Hauptklientel. Im Vordergrund steht der Spielspass für Spieler aller Stärken, auch in heterogenen Teams. Die Benutzeroberfläche soll ansprechend und intuitiv gestaltet werden. Das Ziehen einer Figur soll nicht nur durch Chatten oder einfaches Abstimmen beschlossen werden, sondern in einer Art und Weise dass die Zusammenarbeit wirklich einen Vorteil bringen kann.

Der Rest dieses Dokuments ist wie folgt strukturiert: Zuerst werden einige existierende Arbeiten und Projekte untersucht. Danach folgt ein technischer Teil, welcher die Lösungsansätze und die Architektur der Applikation aufzeigt. Am Schluss folgen Verbesserungsmöglichkeiten und ein Fazit über die gesamte Arbeit. Im Anhang befinden sich technische Details zur Applikation wie Datenbankschemas, verwendete Bibliotheken und eine Liste der Verzeichnisse mit ihrem Verwendungszweck.

2 Related Work

Durch die Beschleunigung der Arbeits- und Kommunikationsprozesse braucht es neue Modelle der Zusammenarbeit, welche mit geografisch verteilten und zeitlich nicht immer verfügbaren Informationen umgehen können. Berechtigungen und Befehlsketten sind nicht mehr hierarchisch aufgebaut, sondern werden je nach Aufgabe immer wieder neu zusammengestellt. Die „Computer Supported Cooperative Work“ (CSCW) versucht diese Modelle zu entwickeln. Nach (Ellis, 1991) lautet die Definition für CSCW-Systeme:

,,Computer-basierte Systeme, die eine Gruppe von Menschen bei ihrer gemeinsamen Aufgabe unterstützen, und die eine Schnittstelle zu einer gemeinsamen Umgebung bereitstellen.``

Ein solches CSCW-System bietet folgende Vorteile[2]:

- flexible und dynamische an der Problemstellung orientierte Gruppenbildung
- flexible Anpassung der Gruppenhierarchie und deren Verwaltung an die jeweiligen Erfordernisse
- schnelle Informationsgewinnung, -verbreitung und -weiterverarbeitung
- Reduzierung der Kosten der Lösung einer Aufgabe
- Ausnutzung der verfügbaren Technologie

Mit unserer Arbeit wollen wir am Beispiel des Team-Schachspiels den Prozess der Entscheidungsfindung untersuchen und die Informationsverbreitung (in diesem Falle den Stand der Partie und die Abstimmungsresultate) verfügbar machen. Das Spiel soll für jedermann einfach verfügbar sein. Wir erreichen dies durch Programmierung einer Webbrowser-Applikation und Nutzung der damit aktuell verfügbaren Techniken zur einfachen Synchronisation von Daten.

In (Surowiecki, 2004) „The Wisdom of Crowds“ argumentiert der Autor, dass die Ansammlung von Informationen in Gruppen zu gemeinsamen Entscheidungen führen, welche oft besser sind als die Lösungsansätze einzelner Teilnehmer. Dies gründet darauf, dass jeder Mensch unterschiedliches Wissen über einen Sachverhalt besitzt, die Meinung eines Einzelnen nicht durch die Gruppe festgelegt ist und es Methoden gibt, aus den Meinungen eines Einzelnen eine Gruppenmeinung zu bilden.

Kollektive Intelligenz kann aber auch zu Fehlentscheidungen führen; dies geschieht, wenn sich einzelne Mitglieder bei ihrer Entscheidungsfindung zu fest auf die Ansichten anderer Gruppenmitglieder abstützen. In einer sog. Informations-Kaskade ignorieren Individuen ihre anfängliche private Entscheidung und stimmen wie der Rest der Gruppe ab (Hold, 1997). Es

[2] http://www.kbs.uni-hannover.de/theses/97/wolpers/node6.html

zeigt sich, dass sich gerade bei Abstimmungen ein Teil der Leute von der Anzahl der bereits abgegebenen Stimmen beeinflussen lassen. Der Wert ihrer Information ist somit sehr gering. Dieses Massenverhalten kann zudem sehr instabil sein (Bikhchandani, 1992). Wir müssen also einen Weg finden, wie wir eine solche Beeinflussung verhindern oder zu unserem Vorteil nutzen können. Unser Ansatz ist die Verwendung von (geheimen) Abstimmungen gekoppelt mit Variantenbäumen (siehe 3.2.3).

Wie in der Einleitung erwähnt wollen wir den Entscheidungsfindungsprozess in einem Computerspiel untersuchen. Es gibt diverse Projekte welche sich damit befassen, zum Beispiel ChessEdu (Miguel Angel Mora, Roberto Moriyón, 2001). ChessEdu ist eine Kollaborations-Software, welche es einer Gruppe (inklusive eines Tutors) ermöglicht, über ein Computernetzwerk gemeinsam oder asynchron an einer Partie Schach teilzunehmen, um das Spiel in einem geführten Umfeld zu lernen. ChessEdu benutzt Spielhistorie, Analyse und Anmerkungen zu Zügen (in Bild und Ton) um den Lernprozess zu verbessern. Teilnehmer und Tutoren können Züge vorschlagen und darüber diskutieren. Die Anwendung benötigt einen Client und einen Server. Unsere Arbeit legt den Schwerpunkt mehr auf Spielspass als auf Lernen sowie auf einfache Benutzung. Man braucht keinen Server oder zusätzliche Software, ein Browser reicht aus. Anders als bei ChessEdu können bei Collaborative Chess Folgen von Zügen vorgeschlagen werden. Das bedeutet, man kann, nachdem man eine eigene Figur gezogen hat, direkt auf dem Brett zeigen welche Züge der Gegner machen könnte, um dann wieder mit eigenen Figuren weitere Züge zu zeigen. Man kann das beliebig lang fortsetzen. Wir benutzen keine Anmerkungen oder ein Tutoren-System, trotzdem kann mit unserem Variantenbaum-System (siehe 3.2.3) ein gewisser Lern-Effekt erzielt werden,

Das Portal Chess.com[3] bietet verschieden Varianten von Online-Schach an, unter anderem Spieler gegen Spieler, Spieler gegen Computer sowie „Voted Chess“. Bei „Voted Chess“ können bis zu mehrere hundert Spieler gemeinsam eine Partie Langzeit-Schach spielen. Die Zeit für einen Zug beträgt normalerweise ein bis zwei Tage. Spieler können ausgehend von der aktuellen Situation einen Zug wählen, sie können diesen aber nicht zurücknehmen. Die Stellung kann via Kommentarfunktion diskutiert werden. Der Nachteil von „Voted Chess“ sind die viel zu langen Zeiten pro Zug; man verliert schnell die Lust am Spiel, da nichts passiert. Wenn man nachträglich einen besseren Zug entdeckt, kann man seine Wahl nicht mehr ändern. Wir favorisieren für unser Spiel kurze Zeiten von wenigen Minuten, so dass man eine Partie an einem Abend durchspielen kann. Wir erhoffen uns davon höheren Spielspass und weniger Fluktuation bei den Mitspielern.

Die Motivation, an einem Onlinespiel teilzunehmen kann von Person zu Person unterschiedlich sein. Man kann sie grob in drei Gruppen unterteilen (Yee, 2006): Erfolg (Fortschritt, besser sein als Andere, Herausforderungen), Gesellschaft (Zusammenarbeit, Beziehungen) und Eintauchen ins Spiel (Spezialwissen haben, Abschalten vom Alltag). Wir

[3] http://www.chess.com

versuchen, alle drei Punkte anzusprechen. Menschen erlangen Befriedigung wenn sie als Teil einer Gruppe Erfolg haben und anderen helfen können. Die Herausforderung, eine gegnerische Gruppe zu schlagen und diesen Erfolg auch öffentlich sichtbar zu machen, kann ebenfalls ein guter Motivator sein. Wir wollen dadurch Schach auch für Gelegenheitsspieler zugänglich machen, welche nicht an einer 1-zu-1 Online-Partie teilnehmen, jedoch durch die Gruppenmotivation in einem Team-Schachspiel mitmachen würden und so ihren Teil zum Erfolg beitragen können. Um einen Spieler an ein Spiel zu binden und ihn zu motivieren, das Spiel mehrmals zu spielen braucht er nach (Dongseong, 2004) ein optimales Spielerlebnis. Dies kann erreicht werden indem mal dem Spielprinzip entsprechende persönliche (Ziele, Rückmeldungen) und soziale (Werkzeuge und Orte um zu kommunizieren) Interaktionen anbietet. Diese Interaktionen werden oft durch Game-Design-Elemente erreicht. Bei Schach ist das Ziel (gegnerischen König schlagen) und die Rückmeldung (man verliert eine Figur) klar vorgegeben. Wir erreichen soziale Kommunikation indem man als Team die Strategie plant (ermöglicht durch die Spieloberfläche) und durch die Einbindung in ein Online-Portal (Facebook).

Da die Benutzeroberfläche möglichst intuitiv gestaltet werden soll, versuchen wir alle relevanten Bedienelemente auf dem Schachbrett unterzubringen, damit der Spieler seinen Fokus nicht ständig wechseln muss. Usability-Studien (Nielsen, 2009) zeigen, dass Menschen auf Webseiten zuerst die obere linke Ecke einer Webseite anschauen, deshalb versuchen wir, das Schachbrett und alle relevanten Bedienelemente in diesem Bereich zu positionieren (siehe 3.2.4).

3 Technischer Teil

In diesem Teil wird zuerst die Aufgabenstellung präzisiert, dann die Lösungsansätze ausgehend von der Aufgabenstellung präsentiert. Zum Schluss folgt ein kurzer Teil über die Architektur des Systems (detaillierte technische Informationen finden sich im Anhang).

3.1 Aufgabenstellung

Das Ziel ist es, die Webapplikation „Collaborative Chess“ zu erstellen, bei der mehrere Spieler gemeinsam in Teams gegeneinander eine Partie Schach spielen. Ein Zug soll nicht nur durch Chatten oder einfaches Abstimmen beschlossen werden, sondern in einer Art und Weise wie die Zusammenarbeit wirklich einen Vorteil bringen kann.

Es stellen sich dabei weitere Fragen:

- Wie findet man Mitspieler und wie können diese einem Team beitreten? Braucht es überhaupt Teams als solches, oder reicht eine Zweckgemeinschaft, die in jedem Spiel anders zusammengesetzt ist?
- Werden die Teams gelost oder kann man sein Team wählen?
- Können Spieler nachträglich dem Spiel beitreten?
- Welchen Spielmodus (zum Beispiel Blitzschach oder Langzeitschach) wählt man? Wie viel Zeit hat ein Team für einen Zug?
- Wie kommunizieren die Spieler untereinander? Braucht es eine Kommentarfunktion oder einen Chat? Sollen die Kommunikationsmöglichkeiten überhaupt vorhanden sein oder spielt man anonym?
- Wenn abgestimmt wird, was passiert bei einem Gleichstand der Stimmen?
- Was machen wir, wenn sich alle Spieler einig sind und noch viel Zeit übrigbleibt?

3.2 Lösungsansätze

3.2.1 Kommunikation

Wie entscheidet ein Team darüber, welches der beste Zug ist? Es gibt verschiedene Möglichkeiten, wie die Spieler miteinander kommunizieren können. Eine offensichtliche Möglichkeit wäre, wenn man einen Zug auf dem Brett vorschlagen und diesen dann kommentieren könnte. Da es aber recht viele Vorschläge geben kann, besteht die Gefahr dass man aufgrund der vielen verschiedenen parallel laufenden Diskussionen den Überblick verliert und viel klicken muss, um alles mitzubekommen. Ein Chat ist eine weitere einfache Möglichkeit und erlaubt die meisten Freiheiten. Der Vorteil an einem Chat ist, dass die Kommunikation persönlicher wird und das Teamgefühl gestärkt wird. Das Problem ist aber auch hier, dass man schnell den Überblick über die Diskussion verliert; dies ist vor allem

ungünstig für Spieler welche neu ins Spiel einsteigen, da sie zuerst die Chat-Historie durchlesen müssen. Ein weiterer Nachteil ist, dass sich Spiele im Nachhinein schlecht analysieren lassen, wenn die ganzen Entscheidungen nur in Gesprächen getroffen werden, vor allem wenn man Spieler aufgrund ihrer Vorschläge bewerten möchte.

Einige Spiele (vor allem im Action-Bereich) sowie diverse Instant-Messaging-Programme verwenden Voice-Chat. Der Vorteil an einem Voice-Chat-System ist, dass keine Zeit durch Tippen verlorengeht. Jedoch gibt es sehr schnell ein Durcheinander wenn mehr als 2-3 Leute teilnehmen, da es a priori keine Regelung gibt, wer wann sprechen darf. Die Einführung irgendwelcher Sprechregeln würde den zeitlichen Vorteil wieder zunichtemachen. Bei Massen-Onlinespielen wie World of Warcraft oder Guild Wars wird oft Voice-Chat verwendet, um Gruppen von bis zu 40 Spielern zu koordinieren. Dort leitet oft ein Gruppenleiter die Diskussion, und nur wenige Spieler ausser ihm dürfen sprechen. Diese Gruppen sind aber meistens auf längere Zeit zusammen, so dass sich ein Leiter entweder gewählt wird oder sich von selbst ergibt, da er die Gruppe gegründet hat. Auch ist die Strategie der Gruppe oft im Voraus klar, so dass sich längere Diskussionen erübrigen. Bei unserem Spiel arbeiten wir aber mit Ad-Hoc Gruppen welche keine klare Hierarchie haben, deshalb denken wir, dass ein Voice-Chat wenig Sinn macht.

Aufgrund dieser Überlegungen sollte unsere Spieloberfläche so gemacht sein, dass die Chat-Diskussion nicht mehr zwingend nötig ist. Der Chat kann aber als Zusatz-Feature eingeschaltet werden.

3.2.2 Abstimmungen

Der erste Ansatz war, ein Abstimmungssystem einzuführen, ähnlich dem System anderer „Voted Chess“ Anwendungen. Jeder Spieler hat die Möglichkeit, eine Figur auf dem Brett zu ziehen. Dazu kann er die Figur mit der Maus auf dem Brett verschieben. Wenn er die Figur loslässt, wird der Zugvorschlag geprüft und wenn er gültig ist, gespeichert. Gleichzeitig wird die Stimme des Spielers für diesen Zug gezählt. Die anderen Spieler werden über den Zug benachrichtigt. Dies funktioniert so, dass die Spieler neben dem Brett eine Liste aller vorgeschlagenen Züge haben, welche periodisch aktualisiert wird. Wenn sich ein Spieler

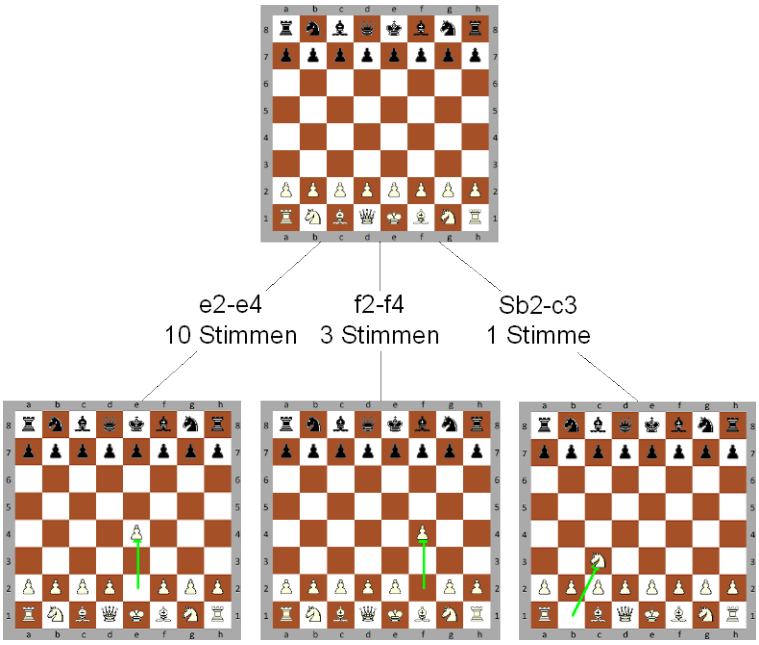

Abbildung 1: Einfaches Abstimmen über Züge

nachträglich anders entscheidet, kann er seinen Vorschlag zurückziehen und einen anderen Zug wählen. Die Rückzug-Möglichkeit ist wichtig, damit das Teamspiel einen Vorteil daraus ziehen kann. Bei bestehenden Umsetzungen von Gemeinschafts-Schach gab es diese meistens nicht, und man sah erst nach der Abstimmung, was die anderen Spieler gewählt hatten. Dies hat zwar den Vorteil, dass man nicht beeinflusst wird, aber der Vorteil des Team-Zusammenspiels fehlt, da man nicht mehr auf einen besseren Vorschlag, den man übersehen hat, umschwenken kann.

Ein Problem stellt sich bei Gleichstand der Stimmen: Wie wird entschieden wenn am Ende der Zeit, welche man für einen Zug hat, zwei oder mehr Varianten mit gleichviel Stimmen führen? Da wir mit kleinen Gruppen arbeiten wollen, wird sich dieses Problem oft stellen. Man könnte warten bis sich alle auf eine Variante geeinigt haben (benötigt aber irgendeine Kommunikationsmöglichkeit wie z.B. Chat) oder die Software entscheiden lassen. Da wir mit festen Zeiten spielen wollen, um einen gewissen Zeitdruck zu halten und das gegnerische Team nicht zu lange warten zu lassen, kommt die erstgenannte Möglichkeit nicht in Frage. Man könnte bei vielen Stimmen auch alle Varianten mit weniger Stimmen als die führenden Varianten eliminieren, und die betreffenden Spieler müssten sich dann für eine der Top-gesetzten Varianten entscheiden. Doch auch dann kann es noch Gleichstand geben. Es braucht daher einen Algorithmus, der sich nach Ablauf der Zeit für eine Variante entscheidet. Wir haben dafür mehrere Möglichkeiten in Betracht gezogen: Zufällige Wahl, Bewertung der Spieler und damit Gewichtung der Stimmen oder ein Teamleiter der am Schluss entscheiden kann. Die Spielergewichtung benötigt eine Bewertungsfunktion der Spieler, welche erst nach einigen Spielen eine sinnvolle Gewichtung liefern kann. Ein Teamleiter kann zufällig bei Spielbeginn gewählt werden, jede Runde ändern, oder der Spieler mit der besten Bewertung sein. Wir entschieden uns aus Gründen der Einfachheit vorläufig für die zufällige Wahl.

Abbildung 2: Spiel mit Stimmen für verschiede Züge

Abstimmungen über einen Zug nutzen aber nur beschränkt das Potential der Gruppe, dies ist vor allem bei kleinen Gruppen ein Problem. Ein Spieler mit einer objektiv besseren Lösung hat keine Möglichkeit die Gruppe von seinem Zug zu überzeugen oder die Gruppe zu warnen wenn viele für einen schlechten Zug stimmen. Man kann natürlich Kommunikationsmöglichkeiten wie Chat und Anmerkungen benutzen, jedoch lesen nicht alle Leute immer die Kommentare mit, und man vertraut oft mehr dem, was man subjektiv auf dem Brett sieht. Wir stellten uns also die Frage, wie Spieler auf einen Zugvorschlag eines

Mitspielers einen gefährlichen gegnerischen Zug (oder sogar eine Zugfolge) anzeigen können, um das Team davon zu überzeugen dass der Zug nicht gut ist (oder im positiven Sinne, um die Auswirkungen des eigenen Vorschlags zu zeigen, z.B. Matt in 3 Zügen).

3.2.3 Variantenbaum

Wir möchten also, dass Spieler ausgehend von einer Situation auf dem Schachbrett einen Zug vorschlagen und zu diesem Zug auch mögliche Züge des Gegners anzeigen können, sei dies im positiven oder negativen Sinn.

Da Schachspieler meist mehrere Züge in die Zukunft denken, erweiterten wir diese Idee, so dass man nun beliebig lange Folgen von Zügen vorschlagen kann. Jeder Zugvorschlag kann mehrere „Antworten" der gegnerischen Seite haben, da die verschiedenen Spieler eines Teams oft unterschiedliche Ideen über den nächsten Zug haben. Aus allen Vorschlägen und Antwortfolgen baut sich mit der Zeit ein Baum aus Varianten auf, welchen es graphisch darzustellen gilt.

Um sowohl gute als auch schlechte Züge aufzuzeigen, kann ein Spieler nun pro Zug mehrere Vorschläge machen und auf jeden Vorschlag eine Folge von Antworten zeigen. Somit hat man als Spieler die Möglichkeit, das Team auf Gefahren hinzuweisen. Da man nun auf dem Brett in die Zukunft denken kann, wäre es gut, wenn man die Zugfolgen nicht jede Runde neu vorschlagen müsste. Wir haben uns daher überlegt, ob es möglich wäre, dass das Team während des gesamten Spiels an einem einzigen grossen Variantenbaum arbeitet. Unser Ansatz sieht folgendermassen aus: Während das eigene Team am Zug ist, kann beliebig in die Tiefe gespielt werden, man kann also Züge mit eigenen Figuren vorschlagen und mögliche Antworten des Gegners aufzeigen. Weiter kann man auf diese Antworten wieder eigene Züge machen usw. Macht nun das gegnerische Team während seiner Zug-Zeit genau den Zug, den man sich gedacht hat, so sind in der folgenden Runde des eigenen Teams alle Antworten auf den gegnerischen Zug bereits als Vorschläge aufgelistet. Man kann also sagen, dass vom Variantenbaum alles abgeschnitten wird ausser dem gegnerischen Zug, falls dieser tatsächlich als Antwort vorgeschlagen wurde. Ist dies nicht der Fall, beginnt man die nächste Runde einfach ohne Vorschläge und muss von dort seine Strategie neu aufbauen. Mit diesem Ansatz arbeitet man also zusammen während des gesamten Spiels an den Vorschlägen, das Brett dient gleichzeitig als „Notizblock", um seine Strategie für die folgenden Runden festzulegen. Daher kann ein Spieler jederzeit mit den Figuren auf dem Brett ziehen und Strategien ausprobieren, auch wenn die Gegenseite am Zug ist, da ein Schachspieler dies in Gedanken auch machen würde.

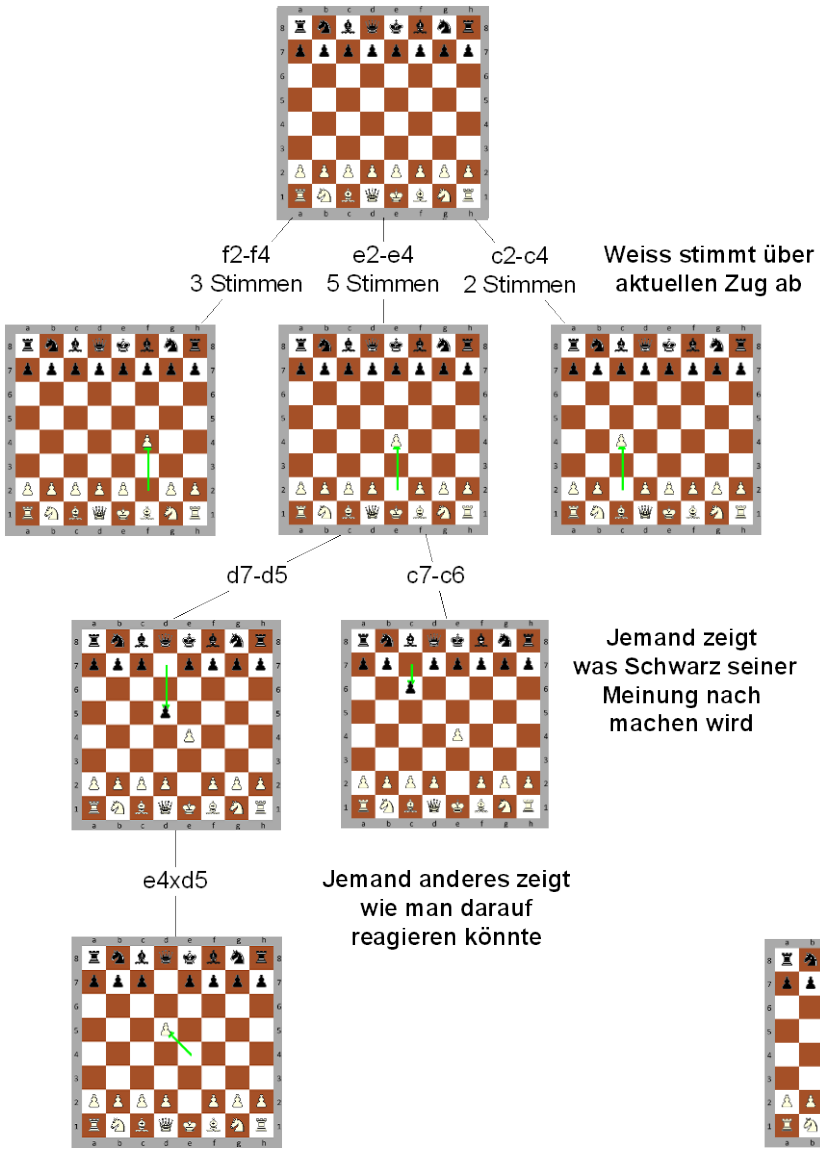

Abbildung 4: Variantenbaum für Weiss

Abbildung 3: Variantenbaum wird beschnitten

Da jeder Spieler mehrere Züge vorschlagen kann, haben wir das Ziehen der Figur und das Abstimmen über einen Zug getrennt. Spieler müssen nun manuell über die Züge abstimmen, wir haben dies aber insofern vereinfach als dass für den letztgemachten Zugvorschlag automatisch eine Stimme abgegeben wird, man kann aber jederzeit für einen anderen Vorschlag abstimmen.

Abbildung 5: Benutzeroberfläche für Variantenbaum

Die Benutzeroberfläche haben wir so gestaltet, dass man auf der einen Seite das Spielbrett sieht, auf der anderen Seite den ausgeklappten Baum mit allen vorgeschlagenen Zügen, dem Symbol der Figur, dem Namen des Zuges, also z.B. e2-e4, den Abstimmungsergebnissen und der Uhr. Dieser Baum wird automatisch aktualisiert sobald ein anderer Spieler einen Zug macht oder die Zeit abläuft.

Um Informationskaskaden (Hold, 1997) zu vermeiden, kann die Abstimmung optional „blind“ durchgeführt werden, d.h. man sieht erst nachdem man selbst einen Zug vorgeschlagen hat die Varianten und Resultate der anderen Spieler.

3.2.4 Pfeile und Benutzeroberfläche

Erste Versuche haben gezeigt dass es mühsam ist, zwischen Spielbrett und Baumansicht hin- und her zu schauen. Webseiten-Benutzer schauen meistens zuerst in die linke obere Ecke und fahren dann mit dem Blick die Form eines F‘s ab (Nielsen, 2009). Wir suchten eine Möglichkeit, die wichtigsten Informationen alle auf dem Spielbrett darzustellen, welches auf der linken Seite des Programmfensters platziert ist. Diese sind: unser Zugvorschlag sowie die Varianten und die möglichen gegnerischen Antworten dazu. Unser Ansatz benutzt dazu verschiedenfarbige Pfeile, welche über das Spielbrett gelegt werden. Alle Züge aus der aktuellen Situation werden mit einem gelben Pfeil dargestellt, der gerade gezogene Zug mit einem grünen Pfeil. Gegnerische Antworten werden durch rote Pfeile dargestellt. Zusätzlich wird der Zug, der in die aktuelle Stellung hineinführte, durch einen grauen Pfeil dargestellt; dieser wird auch verwendet, wenn man sich die Spiel-Historie anschaut.

Abbildung 6: Vorgeschlagene Züge (gelb), aktuell vorgeschlagener Zug (grün) und mögliche Antworten des Gegners (rot)

Wenn man eine Figur gezogen hat und gerne einen alternativen Vorschlag machen möchte, müsste man seine Figur zuerst zurücknehmen, um in die Ausgangsstellung zu gelangen. Dies ist möglich, indem man im Spielbaum auf die Ausgangsstellung, also das Wurzelelement, klickt. Wir haben zwei Hilfen eingebaut, um auch hier auf dem Spielbrett zu bleiben. Einerseits erscheint nach dem Zug ein „Zurück"-Knopf, um einen Zug zurückzuspringen. Andererseits kann man nach einem Zug einfach eine Figur derselben Farbe ziehen, dann wird die erste Figur automatisch zurückgesetzt (dies funktioniert aber logischerweise nicht, wenn man mit genau derselben Figur einen anderen Zug machen möchte, deshalb muss man in dieser Situation den „Zurück"-Knopf benutzen).

3.2.5 Formung von Teams

Ein weiteres Problem betraf die Formung der Teams. Sollten sich Teams ähnlich wie Clans in Netzwerk-Computerspielen organisieren können oder für jedes Spiel neu zusammengestellt werden? Braucht es eine Teambewertung? Da wir das Spiel als Facebook-Anwendung laufen lassen wollen, muss dieser Prozess vor allem sehr einfach sein, da sich Teilnehmer nicht lange damit aufhalten wollen. Da zudem die Fluktuation in einer solche Webanwendung recht gross sein kann, entschieden wir uns, einfache Ad-Hoc-Teams zu machen. Diese werden automatisch erstellt wenn jemand ein Spiel startet, und jeder Spieler, der dem Spiel beitritt, wird automatisch dem Team der jeweiligen Seite zugeordnet. Nach Ende des Spiels werden die Teams nicht mehr benötigt und somit aufgelöst.

3.2.6 Spielmodus

Wir mussten uns auch entscheiden, welchen Spielmodus man anbieten soll. Viele Online-Schach-Varianten sind eher auf Langzeitspiele ausgelegt (z.B. E-Mail Schach). Das geht gut, solange nur zwei Spieler eine Partie spielen. Bei einer Partie mit vielen Teilnehmern besteht aber die Gefahr, dass einige das Spiel vorzeitig verlassen, wenn es zu lang geht. Unsere Überlegung war daher, dass eine Partie nicht mehr als ungefähr zwei Stunden dauern sollte, damit der Spielspass nicht

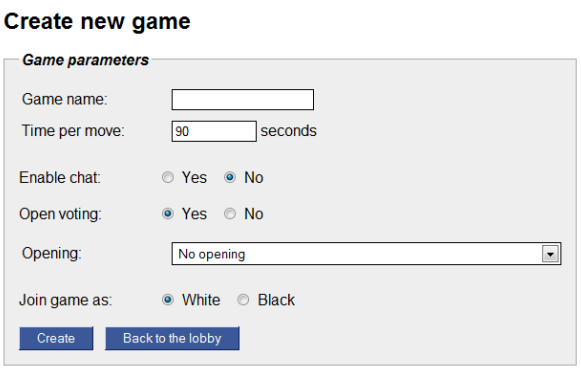

Abbildung 7: Optionen bei der Erstellung eines neuen Spiels

verloren geht und auch ein gewisser Zeitdruck vorhanden ist.

Wir entschieden uns für folgenden Modus: Jedes Team hat einige wenige Minuten Zeit um über einen Zug zu entscheiden. Wenn wir annehmen, dass eine Schachpartie ca. 40 Züge dauert und wir eine Spielzeit von 2 Stunden erreichen möchten, ergibt das 90 Sekunden Zeit pro „Halbzug“ (also pro Zug und Farbe). Wir wählten diesen Wert als Standard. Die Zeit kann beim Spielstart festgelegt werden, aber während dem Spiel nicht mehr geändert werden. Fällt ein Team innerhalb dieser Zeit keine Entscheidung, gilt das Spiel als verloren. Nach Ablauf der Zeit analysiert das Spiel alle eingegangenen Stimmen und wählt den Zug mit den meisten Stimmen. Wenn sich alle Spieler bereits vor Ablauf der Zeit auf einen Zug einigen konnten, können Sie den Zug direkt ausführen lassen, so kann man vor allem in der Anfangsphase Zeit einsparen.

3.3 Architektur des Systems

3.3.1 Technischer Hintergrund

Aufgrund früherer Erfahrungen mit der Programmiersprache PHP fiel die Entscheidung schnell, die Software in PHP zu schreiben. Zudem ist die Sprache relativ einfach zu erlernen, bietet in der Grundausstattung schon sehr viele Funktionen, es gibt massenweise Zusatz-Bibliotheken und sie hat eine recht gute Performance. PHP ist eine Skriptsprache, welche die HTML-Ausgabe eines Webservers aufgrund von HTTP-GET- und POST-Anfragen zusammenstellt und ausgibt. Zum Speichern der Daten wird eine MySQL Datenbank verwendet, die mit PHP sehr gut zusammen harmoniert. Das klassische Setup ist eine (L)AMP-Umgebung, also Apache Webserver, MySQL Datenbank und PHP auf Linux, welches wir auch für dieses Projekt verwenden, und bei Webhosting-Anbietern heutzutage zum Standard gehört.

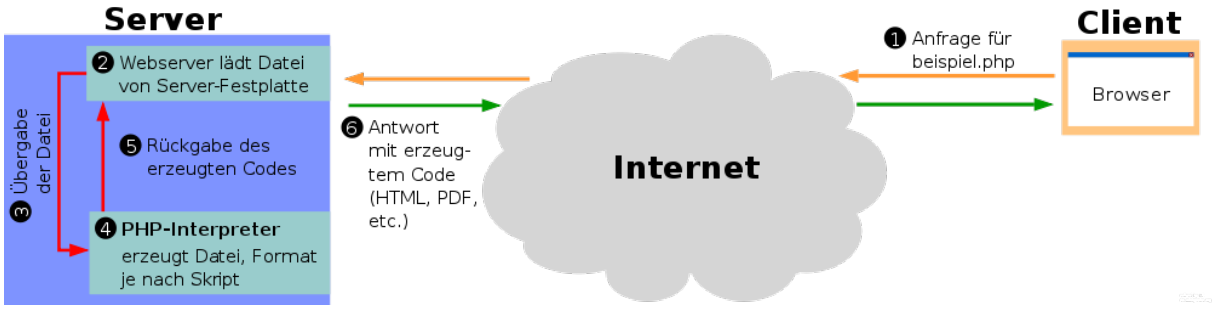

Abbildung 8: Client sendet Anfrage an Webserver, dieser generiert mit Hilfe von PHP die Ausgabe und sendet sie zurück

Die Applikation benutzt ein Model-View-Controller[4] Framework. Dies ist ein Architekturmuster, um Datenmodell, Programmsteuerung und Präsentation zu trennen und damit die Softwareentwicklung flexibel zu gestalten und nachträgliche Änderungen oder Erweiterungen zu erleichtern. Die Anfrage des Benutzers erreicht zuerst einen sogenannten Router, welcher anhand der Adresse feststellt, welcher Controller benötigt wird. Der Controller verarbeitet dann diese Anfrage. Er setzt die „Geschäftslogik“ um, indem er

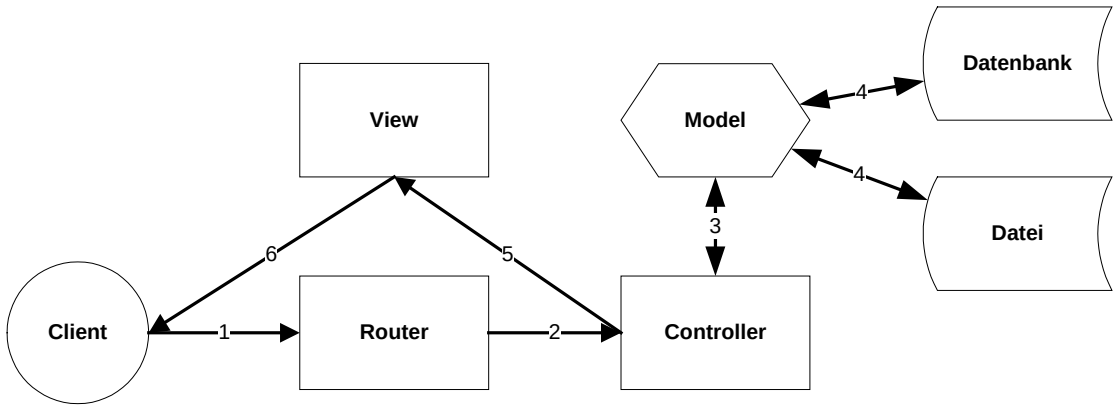

Abbildung 9: MVC Architektur

[4] http://de.wikipedia.org/wiki/Model_View_Controller

mithilfe eines Modells Daten lädt, diese verarbeitet und dann an die Ausgabe weiterleitet. Ein Modell ist eine Abstraktionsschicht zwischen dem Teil des Programms welcher die Daten verarbeitet und den realen Datenstrukturen (zum Beispiel eine Datenbanktabelle oder eine Datei). Es bietet einfache und sichere (durch Prüfung auf korrekte Eingaben und SQL-Injection Angriffe) Funktionen zur Datenmanipulation. Views sind HTML Dateien, in welche die Ausgabe des Controllers durch Platzhalter eingesetzt wird. Bei unserem Projekt wird damit alles ausser der eigentlichen Spieloberfläche damit realisiert, also Lobby, Spiel-Eröffnung, Liste der gespielten Spiele, Spielerprofil, Benutzerverwaltung etc.

Das eigentliche Spiel baut stark auf JavaScript und AJAX (asynchrones JavaScript und XML) auf, eine Technik, welche es ermöglicht, ohne komplettes Neuladen einer Webseite Daten zwischen Server und Browser auszutauschen. Somit können das Spielbrett, die Historie und der Spielbaum laufend aktualisiert werden. Technisch wird dies mithilfe des XMLHttpRequest-Objekts von JavaScript gelöst, welches vom Server asynchron Daten in Form von XML-Strukturen anfordern oder zum Server senden kann. JavaScript kann dann das Document-Object-Model, also die Struktur einer HTML-Seite, manipulieren und so z.B. Inhalte von Elementen mit neuen Daten, welche es vom Server erhalten hat, füllen. In unserer Anwendung wird vor allem mit Polling gearbeitet, dass bedeutet, dass alle paar Sekunden der Spielbaum durch eine Anfrage an den Server aktualisiert wird. Die Anzeige einer Spielsituation auf dem Brett, das Speichern eines Zuges, nachdem man eine Figur gezogen hat, die Abstimmungen werden auch mit AJAX-Anfragen realisiert.

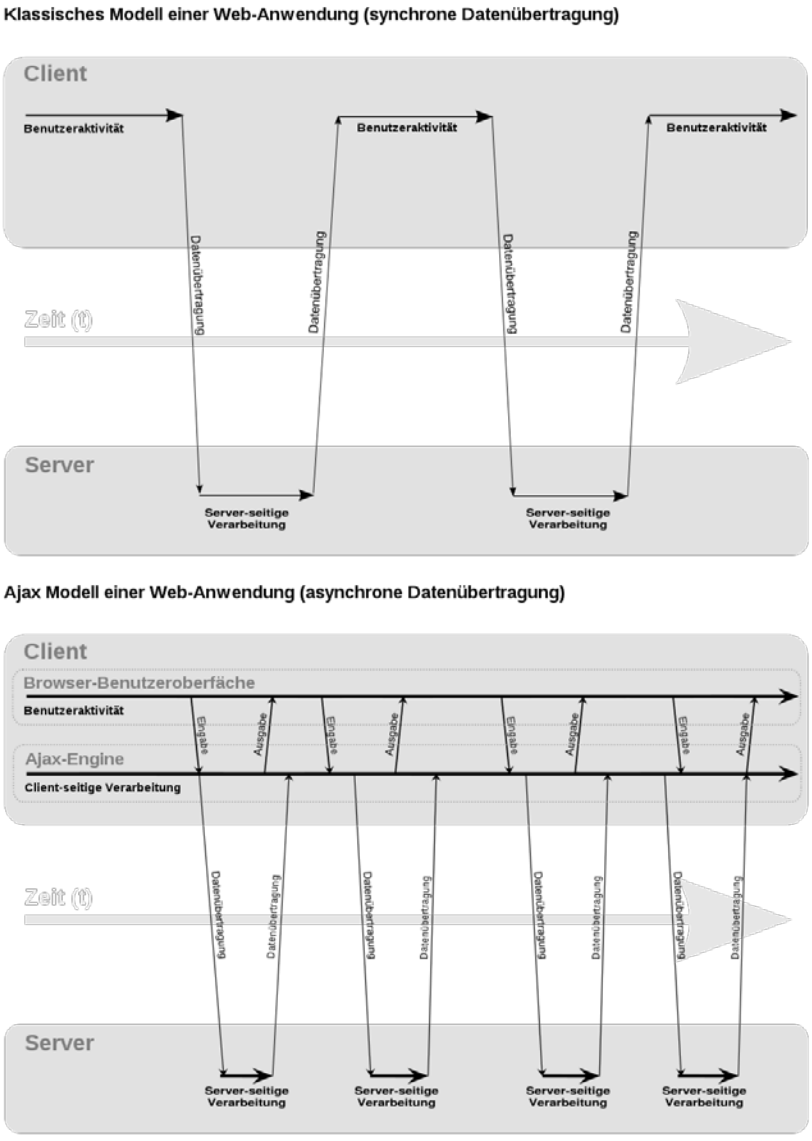

Abbildung 10: Vergleich der Prozessflüsse einer Web-Anwendung (klassisch und asynchron)

3.3.2 Facebook

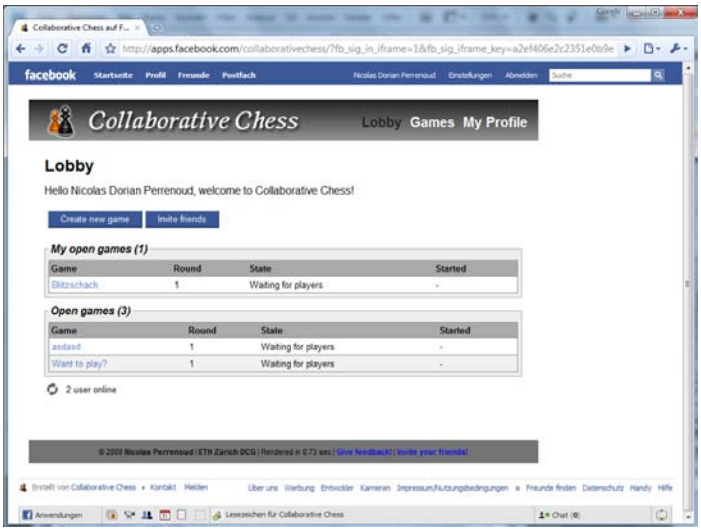

Abbildung 11: Startseite der FaceBook Applikation

Die Webapplikation soll als Facebook-Anwendung implementiert werden, da dies momentan ein idealer Weg ist um Mitspieler zu erreichen. Die Kommunikation mit Facebook beschränkt sich auf den Austausch von Benutzerinformation und Session/Login-Status, ansonsten ist das Spiel selbstständig. Facebook bietet eine API für alle möglichen Programmiersprachen an, um diesen Prozess einfach zu gestalten. Collaborative Chess hat intern eine eigene Benutzer- und Sitzungsverwaltung. Wenn ein Benutzer die Anwendung „Collaborative Chess“ zu seinem Profil hinzufügt, wird der Benutzerdatensatz der „Collaborative Chess“-Datenbank hinzugefügt. Facebook übergibt beim Starten der Anwendung die User-ID und ein Token an die Applikation, diese Daten werden dann mit der Datenbank abgeglichen und der Benutzer wird eingeloggt. Eine weitere Verbindung zu Facebook besteht in der „Einladen“ Funktion, die es ermöglicht, den Link zur Anwendung an eine Liste von Freunden zu senden damit sie „Collaborative Chess“ hinzufügen können. Die Anwendung ist bei Facebook unter der Adresse http://apps.facebook.com/collaborativechess erreichbar.

4 Verbesserungsmöglichkeiten

Das Projekt ist in der im technischen Teil beschriebenen Form funktionsfähig, es gibt aber noch viele Möglichkeiten, es zu erweitern.

4.1.1 Ranking

Gerade bei Online-Games ist vielen Leuten eine (öffentliche) Bewertung wichtig. Eine Rangliste stellt einen gewissen Ansporn dar, und lädt die Spieler dazu ein, das Spiel regelmässig zu spielen. Die einfachste Rangliste würde auf einem Gewonnen/Verloren-Verhältnis und Anzahl der gespielten Spiele basieren, dies muss aber nicht unbedingt der individuellen Stärke des Spielers entsprechen. Man kann durch Verfahren wie ELO[5] ein genaueres Ranking erhalten, das für schwächere Gegner weniger und für stärkere Gegner mehr Punkte vergibt. Eine individuellere Bewertung könnte man durch Analyse der Abstimmungen auf dem Variantenbaums gewinnen. Man kann dazu z.B. für alle Abstimmungen, welche dann auch wirklich gezogen wurden, Zusatzpunkte vergeben, falls damit die Situation verbessert wurde oder das Spiel gewonnen wurde. Diese Punktevergabe richtig zu balancieren wäre aber recht anspruchsvoll.

4.1.2 Zusatzzeit

Bei Spielbeginn hat man oft zu viel Zeit, im Mittelteil fehlt diese dann. Eine Idee wäre, dass man Zeit aufsparen kann, indem man vor Ablauf der Uhr mit seiner Figur zieht und diese Zeit später zusätzlich beziehen kann (dies müsste auch per Abstimmung geregelt werden). Eine andere Idee wäre, dass man ein fixes Zeitbudget n-mal zusätzlich beziehen kann.

4.1.3 Interface

Interfaceverbesserungen könnten z.B. die Pfeile betreffen. Die gelben Pfeile, welche die Zugvorschläge darstellen, könnten mit einem Tooltipp versehen werden, der die aktuelle Anzahl Stimmen anzeigt. Eine andere Idee wäre, unterschiedliche Farben je nach Anzahl Stimmen zu verwenden, dort muss aber aufgepasst werden, dass es kein Farb-Chaos mit den bereits existierenden Farben gibt. Eine weitere Verbesserung wäre die Verwendung von Sounds, wichtig wäre vor allem eine Art Alarm wenn nicht mehr viel Zeit übrig bleibt, oder wenn der Gegner seine Figur gezogen hat und das eigene Team wieder an der Reihe ist (dies könnte auch grafisch gelöst werden, z.B. durch eine blinkende Nachricht).

[5] http://en.wikipedia.org/wiki/Elo_rating_system

4.1.4 Schachcomputer

Mit der Implementierung eines (einfachen) Schachcomputers können z.B. folgende Sachen verbessert werden:

- Bootstrap-Prozess: Ein Mehrspieler-Projekt lebt davon, dass eine gewisse kritische Masse an Spieler immer da ist, um neue Spiele zu eröffnen. Durch einen Schachcomputer-Spieler, der z.B. nur die halbe Gewichtung bei Abstimmungen hat, könnte man immer offene Spiele anbieten, was das Projekt für neu hinzugekommene Spieler interessant macht und sie zum Bleiben einlädt. Sind dann erst mal ein paar Spiele mit echten Mitspielern am Laufen, wird die Motivation auch steigen, einem laufenden Spiel beizutreten.
- Bewertung: Bei nachträglicher Analyse des Variantenbaums könnte ein Schachcomputer Spieler anhand ihrer Abstimmungen bewerten und so helfen, ein individuelles Spielerranking zu erhalten

Der Schachcomputer lässt sich auf verschiedene Weise implementieren: Eine einfache Variante wäre, ihn in der bereits verwendeten Programmiersprache zu schreiben und die vorhandene Schach-Bibliothek zu nutzen. Die Bewertung wäre sehr einfach, da er nur die nächsten 2-3 Züge anschauen würde und anhand Figurenstärke entscheiden könnte. Eine andere Möglichkeit wäre die Benutzung eines Webservices, falls es so einen mal geben würde. Eine dritte Möglichkeit ist, einen Schachcomputer, welcher in einer anderen Programmiersprache vorhanden ist, zu erweitern, so dass er auf die Collaborative-Chess Datenbank zugreifen kann und dort direkt Spielstände auslesen und Aktionen durchführen kann.

5 Zusammenfassung

Wir haben mit dieser Arbeit ein Team-Schachspiel entwickelt, um Entscheidungsprozesse in Gruppen zu verstehen, Abstimmungsmethoden zu testen und Spielspass zu fördern. Wir hoffen dies durch Variantenbäume, an welchen das ganze Team arbeitet, und durch eine kompakte Benutzeroberfläche erreicht zu haben. Es gibt jedoch - wie bereits dargelegt - einige Verbesserungsmöglichkeiten, um das Spiel noch interessanter zu gestalten. Ebenfalls können in einer weiteren Arbeit die Lösungsansätze durch Testen mit grösseren Gruppen und der Deaktivierung bestimmter Features evaluiert werden. Die Arbeit an diesem Projekt war sehr lehrreich und zeigte auf, wie wichtig und vor allem auch zeitaufwändig die Gestaltung einer einfach zu bedienenden Benutzeroberfläche sein kann.

6 Anhang

6.1 Verwendete Bibliotheken

6.1.1 PHP

Smarty Template Engine	Ermöglicht die Trennung von HTML Layout und PHP-Code. Ausgaben der PHP Skripte werden an bestimmten Stellen im HTML Layout eingefügt, welche mit sog. Smarty-Tags markiert sind. http://www.smarty.net
XAJAX	AJAX Framework für asynchrone Kommunikation zwischen Browser und Server. http://xajaxproject.org
Facebook API	Kommunikation zwischen Facebook und Facebook-Applikationen. Ermöglicht Authentifizierung und den Austausch von Profildaten. http://wiki.developers.facebook.com
Quiveo	Einfaches Model-View-Controller Framework für PHP. http://code.google.com/p/quiveo

6.1.2 JavaScript

Scriptaculous	DOM-Handling, Drag'n'Drop und Animationen für JavaScript. Basiert auf Prototype. http://script.aculo.us
wz_jsgraphics	JavaScript Vektor-Grafik Library. Ermöglicht Linien, Formen und Textgrafiken

6.2 Verzeichnisstruktur

Verzeichnis	**Verwendungszweck**
cache	Temporäre Dateien, z.B. gerenderte HTML Layouts der Template Engine
classes	PHP Klassen, z.b. Schach-Library
config	Konfigurationsdateien, z.B. Datenbankkonfiguration
controller	Applikations-Logik, die hauptsächliche Arbeit geschieht hier (MVC)
cron	Zeitgesteuerte Skripte für UNIX Cron-Dienst, direkt aufzurufen
helper	Zusätzliche Funktionen, welche bei Bedarf nachgeladen werden können
hooks	Codedateien welche an bestimmten Stellen eingehängt werden
i18n	Übersetzungen
images	Bilder
js	Externe JavaScript-Dateien und Libraries
layouts	HTML Layouts für die Template-Engine
libs	Externe PHP Bibliotheken
logs	Logdateien, z.B. für Debugging
models	Datenmodelle der Datenbanktabellen (MVC)
plugins	Zusatzmodule für externen Bibliotheken
views	Ansichten (MVC), HTML-Dateien mit Platzhaltern

6.3 Datenbankschema

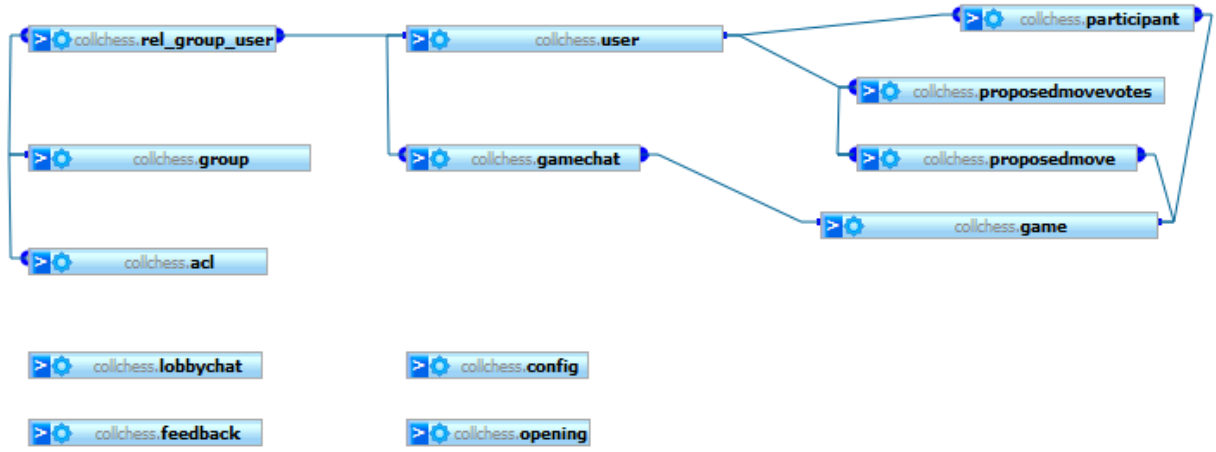

Abbildung 12: Datenbankschema von Collaborative Chess

Tabelle	Verwendungszweck
acl	Access Control Lists (Regelt Berechtigungen für Gruppen)
config	Grundlegende Einstellungen
feedback	Backup der Feedback-Fromular-Nachrichten, welche per E-Mail an die Entwickler gesendet werden
game	Spieldaten (Name, Zeit pro Runde, Resultat, Startzeit…)
gamechat	Chat-Nachrichten des Game-Chats. Nachrichten werden einem Spiel und einem User zugeordnet.
group	Benutzergruppen
lobbychat	Chat-Nachrichten des Lobby-Chats
participant	Verknpüft User mit Spielen und legt fest, wer auf welcher Seite spielt
proposedmove	Zugvorschläge/Variantenbaum. Werden einem Spiel und einem Eltern-Zugvorschlag zugeordnet.
proposedmovevotes	Stimmen der Spieler für einen Zugvorschlag
rel_group_user	Verknpüft Gruppen mit Benutzern (n:m)
user	Benutzer

7 Referenzen

Bikhchandani. (1992). *Theory of Fads, Fashion, Custom, and Cultural Change as Information Cascades.*

Dongseong, C. (2004). *Why People continue to play Online Games: In Search of Critical Design Factors to Increase Customer Loyality to Online Contents.*

Ellis, C. (1991). Groupware: Some issues and experiences. *Communications of the ACM, volume 1.34* , 38-58.

Hold, A. &. (1997). *Information Cascades in the Laboratory.*

Kasparov, G. (2000). *Kasparov against the world : the story of the greatest online challenge.* New York.

McGrew, T. (May, June 2007). Collaborative Intelligence - The Internet Chess Blub on Game 2 of Kasparov vs. Deep Blue. *IEEE Internet Society* , S. 38-42.

Miguel Angel Mora, Roberto Moriyón. (2001). Guided Collaborative Chess Tutoring through Game History Analysis. *CSCW.*

Nielsen, J. (2009). *Eyetracking Web Usability.* New Riders Press.

Surowiecki, J. (2004). *The Wisdom of Crowds: Why the Many Are Smarter Than the Few and How Collective Wisdom Shapes Business, Economies, Societies and Nations.* Doubleday; Anchor.

Yee, N. (2006). *Motivations for Play in Online Games.*