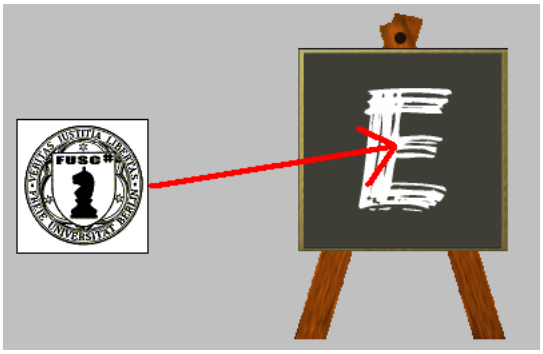
Schach spielen auf einer elektronischen Tafel

Marco Block, Gerald Friedland, Lars Knipping, Raúl Rojas
[block|fland|knipping|rojas]@inf.fu-berlin.de

Dezember 2004



Zusammenfassung

Dieser Artikel ist eine Einführung in die Verwendung von Chalklets. Chalklets sind eine Möglichkeit im E-Kreide-System [E-Kreide] eigene Java-Programme auszuführen. Eine wichtige Philosophie des E-Kreide-Systems ist, dass die Kreidetafel, wie sie von James Pillans 1801 [Pillans] erfunden wurde, auch im Multimediazeitalter ihren Platz behält. Daher dürfen keine typischen GUI-Elemente, wie Knöpfe oder Menüs auf der Tafel erscheinen.

Allerdings wurden bisher noch keine grafisch anspruchsvollen interaktiven Programme als Chalklet realisiert, sie waren lediglich strichzeichnungs- und schriftbasiert. Mit dem interaktiven Schachbrett, das für das Schachprogramm FUSc# [FUSch] entwickelt wurde, steht nun das erste Chalklet zur Verfügung, das auch komplexere, grafische Darstellungen und nicht nur gesten- bzw. schriftbasierte Eingabemöglichkeiten bietet.

Die Herausforderung war, eine konventionelle Schach-Oberfläche mit eigentlich grafischen Elementen, nur mit den von der E-Kreide zur Verfügung gestellten Mitteln - ausschliesslich Linienzüge - zu realsieren. In diesem Artikel werden die verwendeten Lösungsansätze in Bezug auf dieses Schachchalklet besprochen.

1 Einführung in Chalklets

Chalklets sind eine gute Möglichkeit eigene interaktive Java-Programme im E-Kreide-System [E-Kreide 2004] auszuführen und den Ausgabeverlauf dieser zu speichern. Die Kommunikation zwischen E-Kreide und Chalklet findet ausschliesslich durch Strokes statt. Das Chalklet selber kann ebenfalls Strokes in seinem Tafelbereich zeichnen. Im Gegensatz zu Bildern und Applets, muss der E-Kreide-Client das Chalklet nicht selber laden. Er erhält lediglich die vom Chalklet erzeugten Strokes und speichert diese als Board-Events [Knipping 2004].

Auf den ersten Blick erscheint es, als seien Programme mit einer grossen Anforderung an grafischen Elementen dabei kaum zu realisieren, denn die Philosophie der Chalklets besagt: "*Es sollten grafische Elemente in der gleichen Geschwindigkeit angezeigt werden, wie sie ein Mensch an der Tafel zeichnen könnte.*". Dadurch ist auch eine bessere Geschwindigkeit bei der Übertragung gewährleistet. Ein Stroke besteht aus Liniensegmenten. Eine obere Schranke für die Darstellung dieser Liniensegmente liegt bei 100 pro Sekunde [Knipping 2004]. Daher ist es in der Konzeptphase des Chalklets notwendig, für grafische Elemente die Anzahl der Liniensegmente zu minimieren. In Wirklichkeit lassen sich die Elemente daher sehr viel schneller anzeigen.

Auf der E-Kreide kann der Benutzer beim Einfügen eines Chalklets einen Bildbereich festlegen. Striche (Strokes), die der Benutzer in diesen Bereich der Tafel zeichnet, interpretiert das Chalklet als Eingabe und verarbeitet sie. Dabei ist darauf zu achten, dass der komplette Stroke innerhalb des Chalklets liegt, da er ansonsten nicht als Eingabe für das Chalklet interpretiert wird (siehe Abbildung 1).

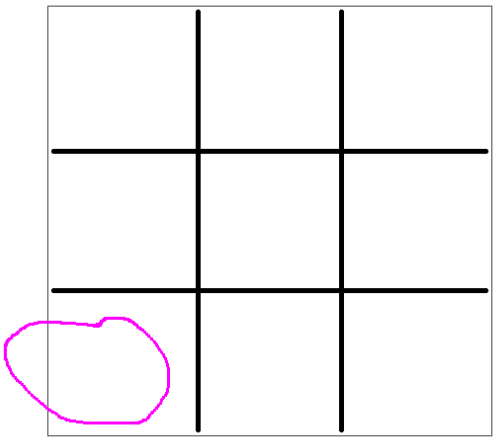


Abbildung 1: TicTacToe

Der Eingabe-Stroke liegt teilweise ausserhalb des Chalkletbereiches, daher wird er nicht als Eingabe interpretiert.

Im nachfolgenden Abschnitt wird die Implementierung und Einbettung eines Chalklets erläutert. In 2 werden einige Chalkletprojekte und im anschliessenden Kapitel das FUSc#-Chalklet vorgestellt.

1.1 Implementierung eines Chalklets

Die Klasse *ChalkletKit* ist die Schnittstelle zwischen E-Kreide und dem eigentlichen Java-Programm. Sie stellt den Rahmen des Chalklets dar. Bei der Einbettung in die E-Kreide werden neben dem Namen auch die minimale Grösse, eine Beschreibung und Informationen über eventuelle Eingabe-Parameter verwaltet. Um diese Methodik zu erfüllen ist folgendes minimal gehaltene ChalkletKit

notwendig.

```
 1 public class BeispielChalkletKit extends ChalkletKit {
 2  protected static final KitSetupInfo KIT_SETUP_INFO = new BeispielChalkletSetupInfo();
 3  private BeispielChalkletKit() {}
 4  public static Chalklet getChalklet(ChalkletContext anChalkletContext, String[] anArrayOfAString) {
 5     return new BeispielChalklet(anChalkletContext);
    }
 6  public static Dimension getMinimumChalkletSize(String[] anArrayOfAString) {
 7     return new Dimension(100, 100);
    }
 8  public static String getChalkletName(Locale aLocale) {
 9     return Txt.get("Beispiel Chalklet", aLocale);
    }
10  public static KitSetupInfo getKitSetupInfo() {
11     return KIT_SETUP_INFO;
    }
12  protected static class BeispielChalkletSetupInfo extends KitSetupInfo {
13     public String getDescription(Locale aLocale) {
14        return Txt.get("Ein einfaches Chalklet-Beispiel.", aLocale);
       }

15     public ParamInfo[] getParamInfo(Locale aLocale) {
16        return new ParamInfo[] {};
       }
    }

17  protected static class BeispielChalklet implements Chalklet  {
18     private boolean aliveFlag_ = true;

19     protected BeispielChalklet(ChalkletContext anChalkletContext) throws IllegalArgumentException {
20        if (anChalkletContext == null) throw new IllegalArgumentException("ChalkletContext is null");
       }

21     public synchronized void pushStroke(BoardStroke aBoardStroke) {
22        if (aliveFlag_) { ... }
       }

23     public synchronized void removeLastStroke() {}

24     public void endChalklet() {
25        aliveFlag_ = false;
       }
26     protected BoardStroke sendStroke(BoardStroke boardStroke) {
27        if (boardStroke != null) anChalkletContext_.sendStroke(boardStroke);
28        return boardStroke;
       }
    }
29  public String toString() {
30     return BeispielChalkletKit.getChalkletName(Locale.ENGLISH);
    }
   }
```

Zeile 6: Die minimale Grösse des Chalkletbereichs wird in Pixel festgelegt.

Zeile 12: Eine innere Klasse vom Typ *KitSetupInfo*. Die Methoden *getDescription()* und *getParamInfo()* werden von E-Kreide aufgerufen, um die Beschreibung des Chalklets und Informationen zu den Eingabe-Parametern anzuzeigen. Abbildung 2 zeigt die Ausgabe der Beschreibung und die Eingabe von Parametern in das Chalklet (Beispiel: SchachChalklet).

Zeile 17: Die innere Klasse vom Typ *Chalklet* beinhaltet die Kommunikation zwischen E-Kreide und Java-Programm.

Zeile 21: Alle Strokes, die innerhalb des Chalklet-Bereichs auf der E-Kreide erzeugt werden, gelangen durch Aufruf dieser Methode zum Chalklet. Nun kann eine Interpretation dieses Strokes stattfinden und möglicherweise dazu führen eigene Strokes zurückzusenden.

Zeile 23: Diese Methode sollte ebenfalls implementiert werden, um ein Undo zu ermöglichen.

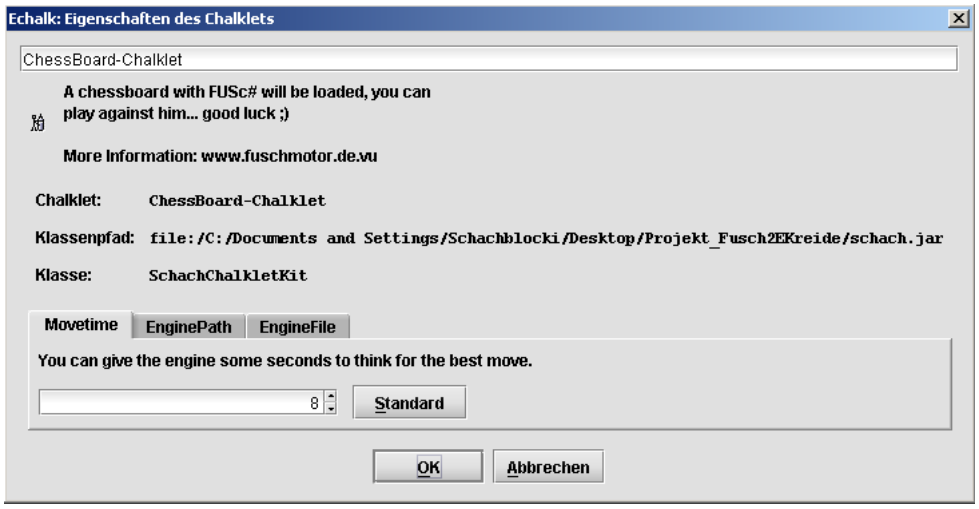
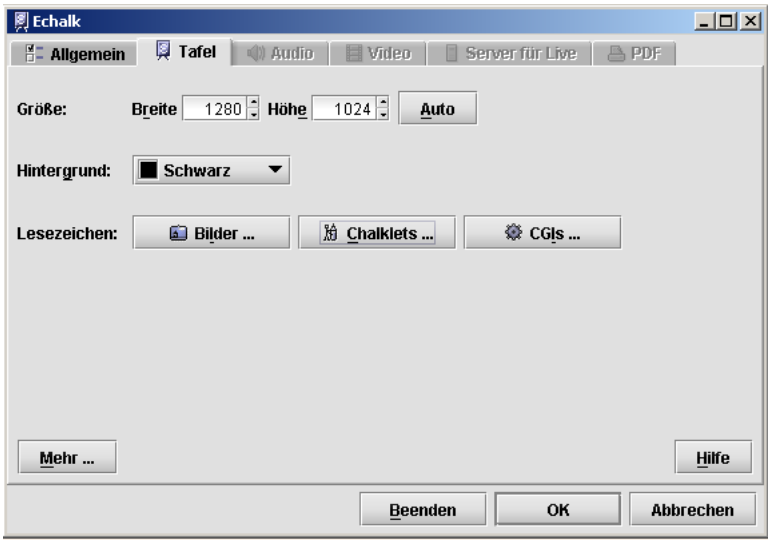


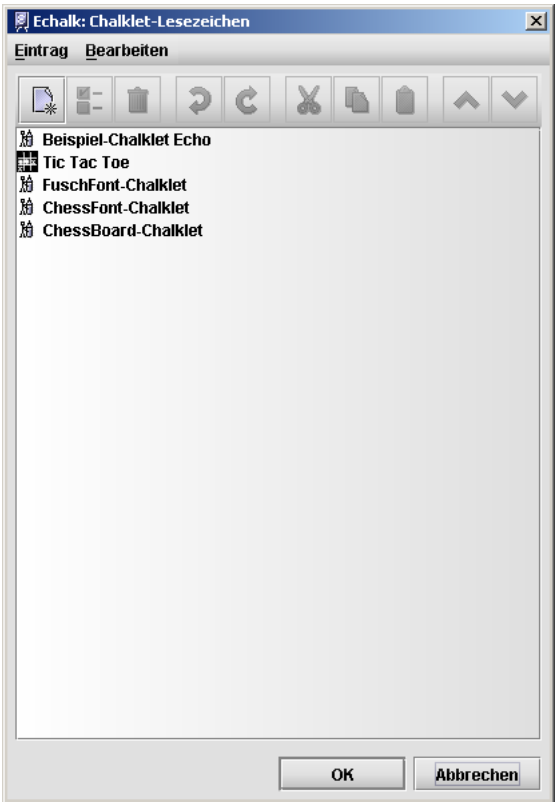
Abbildung 2: KitSetupInfo, Beispiel: SchachChalklet

1.2 Einbindung eines Chalklets

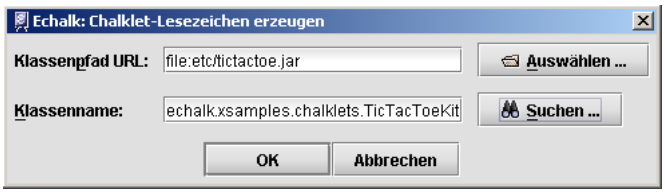
Das Chalklet und die entsprechend zusätzlich erzeugten Klassen müssen zu einem jar-File komprimiert werden. Dieses jar-File lässt sich nun in die E-Kreide wie folgt einbinden. Nach dem Start der E-Kreide muss ein Lesezeichen im Menü Tafel für das Chalklet angelegt werden.



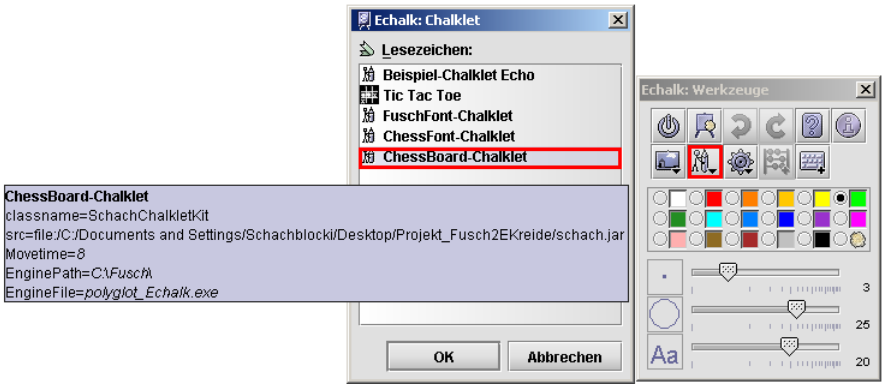
Dazu öffnet sich ein Chalklet-Manager-Dialog. Alle bisher geladenen Chalklets werden aufgelistet.



Nun ist es möglich ein neues Lesezeichen anzulegen. Das jar-File und zusätzlich die ChalkletKit-Klasse innerhalb des jar-Files (es können mehrere Chalklets in einem jar-File existieren) werden ausgewählt.



Da Chalklets Vorabinformationen erhalten können, bietet der Chalklet-Manager-Dialog eine Anpassung der Parameter. Im Verlauf einer E-Kreideaufnahme kann dann das Chalklet im Menü ausgewählt und eingebettet werden.



1.3 Beispielchalklet Echo

Ein einfaches Chalklet-Beispiel von Lars Knipping [Api][Echo]. Der Code ist in der API enthalten und kann sofort ausgeführt und getestet werden.

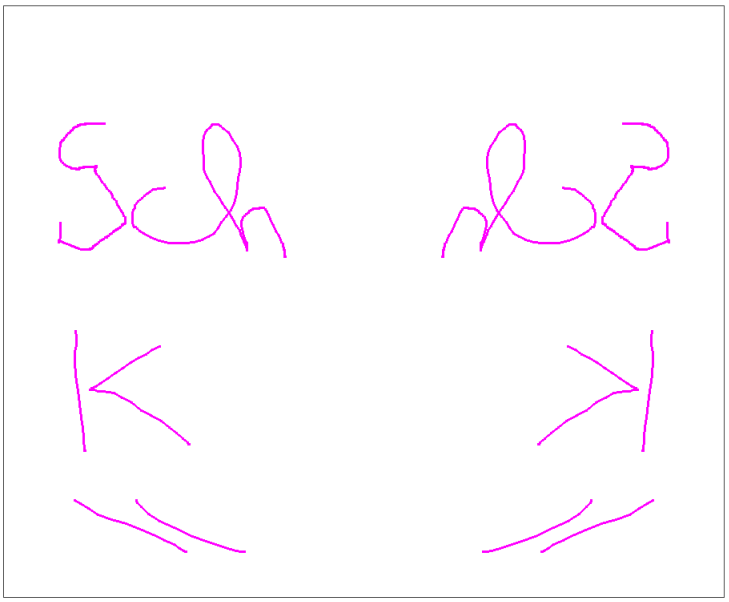


Abbildung 3: Echo-Chalklet

Das Chalklet spiegelt die eingebenen Strokes horizontal oder vertikal zur Mittelachse. Die Ausgabe erfolgt zeitlich verzögert. Als Eingabeparameter erwartet das Chalklet zum einen die Achse (ob horizontal oder vertikal gespiegelt werden soll) und zum anderen die Wartezeit in Millisekunden.

2 Verwandte Projekte

In diesem Kapitel werden einige Chalklet-Beispiele und das Schachprogramm FUSc# vorgestellt. Es soll dem Leser einen Eindruck vermitteln, wie Chalklets arbeiten und welche Möglichkeiten sich bieten eigene Programme für das E-Kreide-System zu entwickeln.

2.1 Tic-Tac-Toe

Ein von Marcus Liwicki geschriebene und von Lars Knipping später etwas überarbeitete Chalklet spielt mit dem Benutzer Tic-Tac-Toe [Api][TicTacToe]. Dabei wird zunächst per Zufall ermittelt, welcher Spieler am Zug ist. Wenn der Benutzer am Zug ist, erwartet das Chalklet eine Eingabe in einem der 9 Felder und setzt anschliessend sein Kreuz. Sollte der Benutzer ein ungültiges Feld markieren, so wird dieser Stroke zurückgenommen.

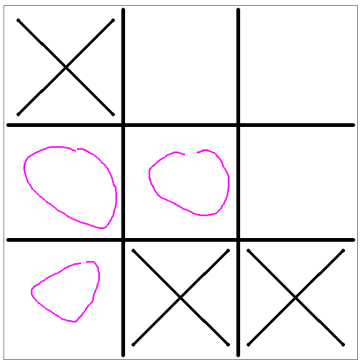


Abbildung 4: TicTacToe auf einer elektronischen Tafel

2.2 Erkennung logischer Schaltungen

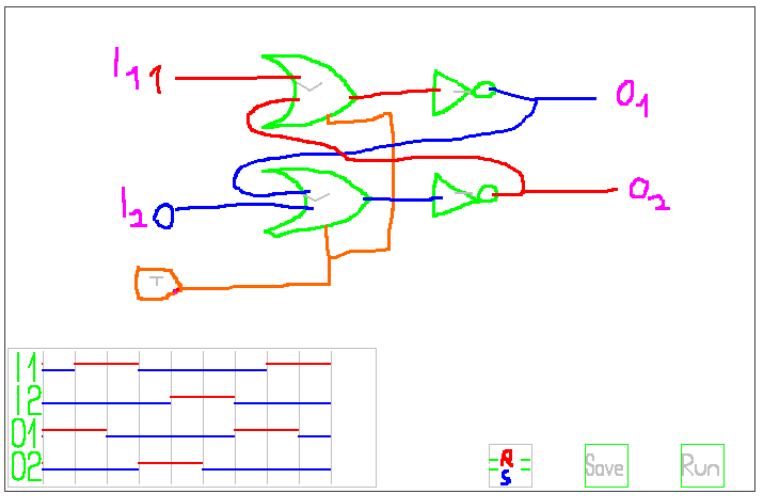


Abbildung 5: Logic-Chalklet

Beispiel einer Verkettung logischer Symbole. Die so entworfene Schaltung kann mit Eingabesignalen den entsprechend korrekten Output liefern.

Das von Markus Liwicki im Rahmen seiner Diplomarbeit [Liwicki 2004] erstellte Chalklet. Es können Symboleingaben, wie z.B. “UND”, “ODER” und “NICHT” erkannt und als logische Funktionssymbole interpretiert werden. Diese können verknüpft und zu Schaltungen zusammengefasst werden. Anschliessend kann die Funktionalität der Schaltung mittels einer Eingabe geprüft werden. Das Programm liefert eine korrekte Ausgabe der Schaltung.

2.3 PyChalk

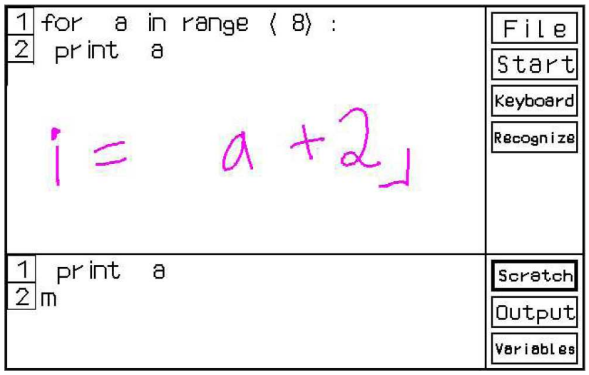


Abbildung 6: PyChalk-Chalklet

Ein aufwendiges Chalklet von Brendan O’Connor [PyChalk 2004] und Hendrik Steffien, das Pythoncode auf einer E-Kreide-Tafel interpretieren und ausführen kann. Marcus Liwicki erstellte den verwendeten Font. Da dieser Font aber Kreissegmente erzeugt und daher eine grosse Anzahl von Liniensegmente verwendet, ist er für die Darstellung in einem Chalklet eher ungeeignet, siehe Diskussion in 3.

2.4 Formelerkennung

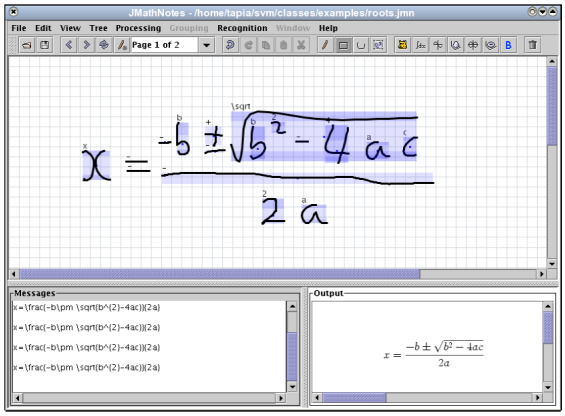


Abbildung 7: JMathNotes Editor

Der von Ernesto Tapia für seine Dissertation [Tapia 2004] entwickelte Editor JMathNotes zur Formelerkennung bietet seine Funktionalität auch in einem Chalklet an, damit können mathematische Ausdrücke quasi online interpretiert werden. Diese Formeln können anschliessend z.B. als Eingaben für Mathematica oder Maple verwendet werden.

2.5 Das Schachprogramm FUSc#

Die Entwicklung des Schachmotors hat im Oktober 2002 begonnen. Die momentane Stärke ist schwer abzuschätzen, liegt aber bei ca. 2000 Elo-Punkten. FUSc# ist ein studentisches Projekt und hat nicht den Anspruch im Jahr 2010 den Weltmeister zu schlagen, sondern eine Plattform zu bieten, experimentell Aspekte der KI, wie Suchstrategien [Plaat 1996] , Heuristiken aber auch Lernen [Block 2004] zu untersuchen.

Abbildung 8: FUSc# auf dem Online-Schach-Server

Derzeit spielen verschiedene FUSc#-Versionen auf einem Online-Schach-Server [OnlineServer]. Dabei handelt es sich teilweise um experimentelle Versionen, die Auskunft über die Qualität neuer Suchkonzepte oder Heuristiken liefern. Eine Eröffnungbibliothek kann dabei anhand gespielter Partien automatisch erweitert werden.

3 Das FUSc#-Chalklet

Ziel unseres Projektes war es, exemplarisch auf der Tafel eine Benutzeroberfläche zu entwickeln, um gegen ein Schachprogramm zu spielen. Dazu muss ein Schachbrett mit Figuren und ein Bereich zur Verfügung gestellt werden, in dem das Schachprogramm Informationen anzeigen kann, z.B. Suchtiefe und Bewertung. Die in 2.3 beschriebene Schriftart ist für eine sich schnell ändernde Darstellung nicht geeignet. Dazu wurde eine neue Schriftart, der FuschFont entwickelt. Die Anzahl der Stroke-Segmente wurde dabei minimal gehalten, was bei den meisten Buchstaben nicht sehr schwer ist. Lediglich Buchstaben mit Anteilen von Bögen mussten etwas angepasst werden. Anschliessend wurde versucht Schachfiguren mit dem gleichen Ziel zu entwerfen und schliesslich das Schachbrett. Als visuell günstig hat es sich erwiesen, die weissen Felder mit 5 diagonal verlaufenden Strokes zu versehen.

3.1 FuschFont

Um der Chalklet-Philosophie gerecht zu bleiben muss eine Schriftart exisitieren, die eine minimale Anzahl von Liniensegmenten aufweisst. Dann ist es möglich in angemessener Zeit eine Textausgabe vorzunehmen. Um die Verwendung dieser Schriftart zu zeigen, wurde sie in ein Chalklet gebettet [FuschFont]. Momentan umfasst der Zeichenvorrat Zahlen und Grossbuchstaben, wobei die Zahlen in digitaler Darstellung vorliegen.

Abbildung 9: Grossbuchstaben- und Zahlendarstellung

3.2 Schachfiguren

Das gleiche Problem wie bei der Schrift, existiert auch für die Spielfiguren. Ähnlich wie beim FuschFont wurde versucht, Schachfiguren mit einer minimalen Anzahl von Liniensegmenten zu zeichnen. Diese Schachfiguren sind gut skalierbar, so dass die Verwendung von mehreren Schachchalklets parallel möglich ist [ChessFont].

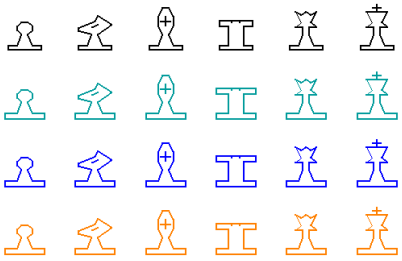


Abbildung 10: ChessFont-Chalklet

3.3 Kommunikation mit FUSc#

Der Schachmotor FUSc# muss nun noch mit dem Chalklet kommunizieren. Beim Laden des FUSc#-Chalklets [FuschChalklet] müssen zunächst folgende Parameter richtig eingestellt sein: **moveTime**, **enginePath**, **engineFile**. Damit der Schachmotor korrekt geladen werden kann. FUSc# ist ein in C# geschriebener Schachmotor und benötigt daher eine .net-Umgebung. Bei der Entwicklung dieses Chalklets musste zunächst eine Verbindung zwischen dem "Java"-Chalklet und dem "C#"-Schachprogramm realisiert werden, diese Schnittstelle ist rein konsolenbasiert und verwendet ein Standard-Protokoll[1] (siehe dazu [Winboard, UCI]). Gegen den Schachmotor FUSc# kann jederzeit im Internet auf unserem Online-Schach-Server[OnlineServer] gespielt werden.

3.3.1 Zugabgabe

Nach dem Einbetten des Chalklets spielt der Benutzer weiss und ist am Zug[2]. Ein Zug kann mit einem Stroke abgegeben werden (siehe Abbildung 12). Anfangs- und Endpunkt des Strokes werden entsprechend den Feldern zugeordnet. Einen Stroke von der E-Kreide zu nehmen ist momentan nicht möglich. Ähnlich wie auf der Kreidetafel kann ein Strich der andere ungewollt kreuzt nicht

[1] In der Schachprogrammierung gibt es momentan 2 Standardprotokolle, mit denen Schachmotoren mit einer grafischen Benutzeroberfläche kommunizieren können. UCI (universal chess interface) von Stefan Meyer-Kahlen und WB (WinBoard) von Tim Mann. Beide Protokolltypen lassen sich fast verlustfrei ineinander konvertieren (siehe Konverter Polyglot [Polyglot]).

[2] Die Planung sieht vor, durch Gesten auf dem Chalklet mehr Funktionalität anzubieten. Beispielsweise das *drehen des Brettes* oder einen *Zug zurücknehmen*.

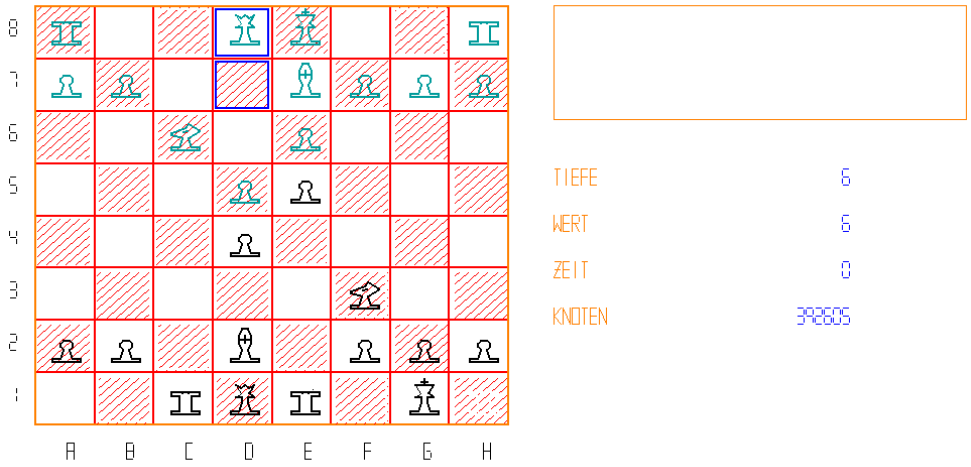


Abbildung 11: FUSc#-Chalklet

einfach entfernt werden. Das hat den Nachteil, dass man möglicherweise zu viele Komponenten updaten muss. Genau das trifft bei der zweiten Zugvariante zu. Das Updateverfahren verfährt wie folgt, der Stroke wird mit der Hintergrundfarbe überschrieben, hinterlässt dadurch aber Artefakte in Feldern und an Figuren, daher muss das Brett komplett neu gezeichnet werden, was auch sichtbar ist.

Die bessere Lösung ist das Umdenken auf Start- und Zielfeld. Ein Zug kann ebenso abgegeben werden, indem mit dem Stift zunächst in das Start- und anschliessend in das Zielfeld geklickt wird (siehe Abbildung 13). Die Felder werden markiert und der Zug wird ausgeführt. Das Chalklet übernimmt dann das Updaten der zwei beteiligten Felder. Dies ist sehr viel effizienter.

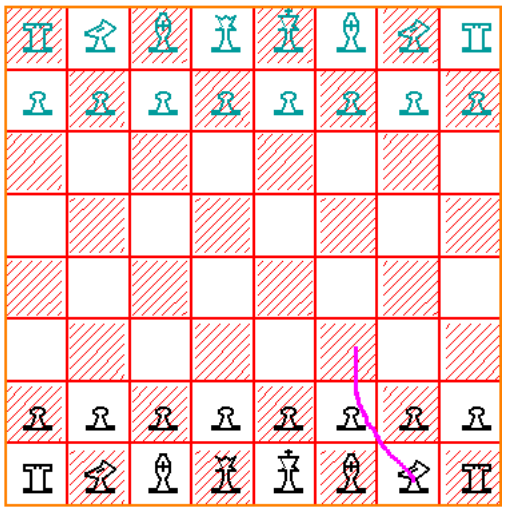


Abbildung 12: Zugabgabe mit einem Stroke

Wenn der Benutzer einen Zug abgegeben hat, beginnt der Schachmotor zu rechnen. Er benötigt für seine Suche **moveTime** Sekunden. Nachdem er einen Zug gespielt hat, wird dieser auf dem Brett ausgeführt und markiert.

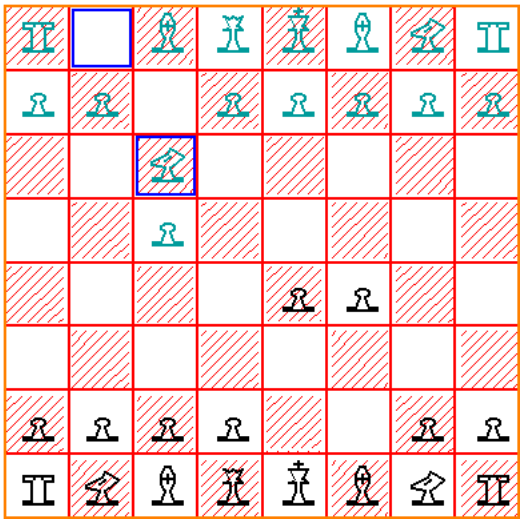


Abbildung 13: Zugabgabe mit zwei Strokes

3.3.2 Informationen zur Zugberechnung

Zu Beginn des Spiels ist es sehr wahrscheinlich, dass FUSc#, der mit einem kleinen Eröffnungsbuch ausgestattet ist, aus diesem Züge entnehmen kann. Dann gibt er lediglich den Zug ohne Bewertungsinformationen zurück. Wenn das Eröffnungsbuch nicht mehr ausreicht und FUSc# selbstständig nach dem besten Zug suchen muss, verwendet FUSc# einen iterativen Suchalgorithmus [Plaat 1996].

Er beginnt zunächst einen Suchbaum der Tiefe 2 durchzurechnen und verwendet die gesammelten Informationen, um die Tiefe 3 schneller berechnen zu können, usw. Nach jedem iterativen Schritt liefert er **Tiefe**, **Knotenanzahl**, **Bewertung** und verbrauchte **Zeit**. Diese Werte werden dann rechts neben dem Schachbrett angezeigt (siehe Abbildung 11).

4 Fazit und Planung

Chalklets sind eine interessante Möglichkeit, eigene interaktive Programme für die E-Kreide zu entwickeln. Auf die typische Herangehensweise bei der Ausgabe muss jedoch verzichtet werden, es wird daher die Sensibilisierung für die minimale Konstruktion grafischer Elemente gefördert. An typische fensterbasierte Elemente, wie Buttons, Menüs oder Frames ist nicht zu denken. Es müssen daher neue Mittel und Wege gefunden werden.

In Zukunft ist ein Ausbau des FUSc#-Chalklets geplant. Diverse Markierungsarten (Felder, Linien, Diagonalen, Pfeile, ...) sollen auf dem Brett möglich werden. Züge vor und zurück, sowie den Einsatz von FUSc# als Stellungsbewerter und Lieferant von Zugvorschlägen. Eine einfache Stellungseingabe ist ebenfalls geplant. Dieses Tool könnte den Einsatz der E-Kreide auf Schachvorträge (z.B. in Vereinen) erweitern.

Ein typisches Szenario: Ein Grossmeister wird in den Verein eingeladen und führt seine Partien vor oder macht Schachtraining anhand bestimmter Stellungen. Dieser Vortrag bleibt dann erhalten und kann jederzeit Wiederverwendung finden. Informationen darüber sind im Netz unter [Block] oder [E-Kreide] zu finden.

Literatur

[Knipping 2004] Knipping, L.: *"An Electronic Chalkboard for Classroom and Distance Teaching"*, Dissertation FU-Berlin, 2004

[E-Kreide 2004] Rojas, R.; Friedland, G.; Knipping, L.; Tapia E.: *"Teaching With an Intelligent Electronic Chalkboard"*, Proceedings of ACM Multimedia 2004, New York 2004

[Block 2004] Block, M: *"Verwendung von Temporale-Differenz-Methoden im Schachmotor FUSc#"*, Diplomarbeit, FU-Berlin, 2004

[Tapia 2004] Tapia, E.: *"Understanding Mathematics: A System for the Recognition of On-Line Handwritten Mathematical Expressions"*, Dissertation, FU-Berlin, 2004

[PyChalk 2004] O'Connor, B.: *"Pychalk: a chalklet for recognizing and running Python code on an electronic blackboard"*, Technical Report FU-Berlin, 2004

[Liwicki 2004] Liwicki, M.: *"Erkennung und Simulation von logischen Schaltungen für E-Chalk"*, Diplomarbeit, FU-Berlin, 2004

[Plaat 1996] Plaat, A.: *"RESEARCH RE:SEARCH & RE-SEARCH"*, Dissertation, Universität Rotterdam, 1996

[Polyglot] UCI to Winboard-Konverter von Fabien Letouzey:
http://wbec-ridderkerk.nl/html/download/fruit/polyglot_11.zip

[Winboard] Winboard-Protokoll von Tim Mann:
http://www.tim-mann.org/xboard.html

[UCI] UCI-Protokoll von Stefan Meyer-Kahlen:
http://shredder.fileburst.com/uci.zip

[OnlineServer] FUSc#-Online-Schachserver:
http://page.mi.fu-berlin.de/~fusch/

[Block] Webseite von Marco Block:
http://page.mi.fu-berlin.de/~block/

[Api] E-Chalk-API:
http://kazan.inf.fu-berlin.de/echalk/docs/api/index.html

[FUSch] Webseite des Forschungsprojektes FUSc#:
http://page.mi.fu-berlin.de/~fusch/

[E-Kreide] Webseite des E-Kreide-Projektes:
http://www.e-kreide.de/

[Pillans] Royal High School Edinburgh:
http://www.edinburgharchitecture.co.uk/royal_high_school.htm

[TicTacToe] Download vom TicTacToe-Chalklet:
http://kazan.inf.fu-berlin.de/echalk/docs/api/index.html

[Echo] Download vom Echo-Chalklet:
http://kazan.inf.fu-berlin.de/echalk/docs/api/index.html

[ChessFont] Download vom Schachfiguren-Chalklet:
http://page.mi.fu-berlin.de/~block/E-Kreide/ChessFontChalkletKit.java

[FuschFont] Download vom FuschFont-Chalklet:
http://page.mi.fu-berlin.de/~block/E-Kreide/FuschFontChalkletKit.java

[FuschChalklet] Download vom FUSc#-Chalklet:
http://page.mi.fu-berlin.de/~block/E-Kreide/schach.jar