Herberts Schachecke

Inhaltsverzeichnis

Lassen Sie die Besucher Ihrer Website fröhlich Schach spielen! Sie finden hier Hinweise

zur Installation,
zu einer „Brett”-Datei,
zur Browser-Server-Kommunikation,
zu den verwendeten URI,
zur Datenbank und den CGI-Skripten,
zur Datei `SR_schach.xslt`,
zur Datei `SR_schach.js`,
zum Brett in seiner Umgebung
mit Beispielen
zu den Dateiformaten in den Beispielen und
zum Gruppenschach.

Autor der Schachecke (der Funktion der Schachbretts) ist

Herbert Schiemann
Borkener Str. 167
46284 Dorsten
<h.schiemann@herbaer.de>
0049 2362 950343

Bitte wenden Sie sich an mich, wenn Sie Fragen, Wünsche oder Anregungen haben oder Fehler feststellen.

Herberts Schachecke: Installation

Voraussetzungen

- Ihre Website läuft unter Apache 2.4 [http://httpd.apache.org/docs/2.4/].
- Sie können in Ihren .htaccess-Dateien [http://httpd.apache.org/docs/2.4/howto/htaccess.html] mod_rewrite [http://httpd.apache.org/docs/2.4/rewrite/] nutzen.
- Sie können CGI-Skripte [http://httpd.apache.org/docs/2.4/howto/cgi.html] im Verzeichnis `/cgi-bin` (oder einem anderen Verzeichnis) nutzen.
- Perl 5 (5.28) mit DBI (1.627) ist verfügbar.
- Sie können mittels DBI eine MariaDB-Datenbank (10.3.27-MariaDB, Server-Version 5.5.29-standard) nutzen.

Ihr Webhoster bietet Ihnen wahrscheinlich diese Möglichkeiten. Die angegebenen Versionsnummern entsprechen meiner Umgebung. Ältere oder neuere Versionen dürften kein Problem sein.

Wenn Sie auf die Server-Konfiguration zugreifen können, können Sie natürlich .htaccess-Dateien vermeiden. Aber ich gehe von üblichen Einschränkungen (zur Sicherheit) aus.

Installation

Sie können die Schachecke in einem beliebigen Verzeichnis Ihres Servers einrichten. Ich nenne das Verzeichnis im Folgenden *`BASIS`*.

Ersetzen Sie in der Datei `htaccess` `BASIS` durch das Verzeichnis, in dem Sie die Schachecke einrichten. Laden Sie die Datei `htaccess` nach *`BASIS`*`/.htacess`.

Die Datei `SR_index.xhtml` ist die Begrüßung des Besuchers. Setzen Sie dort Ihre eigene Begrüßung ein. Der Verweis auf den Quelltext und die GPL sollten erhalten bleiben.

Die Datei `SB_more.xhtml` enthält mehr oder weniger wichtige und nützliche Hinweise für den Besucher. Passen Sie diese Datei an.

Laden Sie die Dateien `SB_*.xhtml`, `SD_*.xhtml`, `SR_*` und die gesamten Verzeichnisse `SR_symbols` und `SR_theme` nach *`BASIS`*`/`.

Setzen Sie in den Dateien `cgi-bin/*` die Daten für den Zugriff auf Ihre Datenbank ein und laden Sie diese Dateien in das Verzeichnis `/cgi-bin` Ihres Servers. Erlauben Sie, die Skripte auszuführen (**chmod +x**).

Rufen Sie in Ihrem Browser die Adresse `http://`*`IHRE_DOMAIN`*`/cgi-bin/sb_move_table` auf. Sie müssen natürlich *`IHRE_DOMAIN`* einsetzen.

Laden Sie die Dateien `GPL-3.txt` und das Archiv `schach.tar.bz2`, wie Sie es heruntergeladen haben, nach *`BASIS`*`/`.

Im Idealfall funktioniert jetzt die Schachecke. Rufen Sie zum Test in Ihrem Browser die Adresse `http://`*`IHRE_DOMAIN`*`/BASIS/` auf.

Schachbretter

Vielleicht sollen Ihre Besucher der Schachecke das Testbrett (`SB_test.xhtml`) nicht benutzen? Entfernen Sie aus der Datei `SR_bretter.xhtml` die Zeile `<li><a href = "SB_test.xhtml">Test</a></li>`. Es gibt

jetzt keinen Verweis auf das Testbrett mehr. Sie können dann auch die Dateien `SB_test.xhtml` und `SR_test.js` und das Skript `sb_set` entfernen.

Für Schachbretter sind Dateinamen `SB_*.xhtml` vorgesehen. Das Sternchen steht für einen oder mehrere Kleinbuchstaben. Wenn Sie ein Brett entfernen, dann müssen Sie auch den zugehörigen Eintrag in der Datei `SR_bretter.xhtml` entfernen. Wenn Sie ein Brett hinzufügen, dann sollten Sie das Brett in die Datei `SR_bretter.xhtml` eintragen, damit Ihre Besucher einen Verweis auf das Brett sehen.

In der Datei `.htaccess` ist das Blitz-Brett (`SB_blitz.xhtml`) als Vorgabe eingestellt. Stellen Sie die Vorgabe ein, wie Sie es wünschen. Wenn Sie das Blitz-Brett entfernen, müssen (sollten) Sie die Vorgabe ändern.

CGI-Verzeichnis

Wenn Sie ein anderes Verzeichnis als `/cgi-bin` für CGI-Skripte benutzen, dann müssen Sie in der Datei `.htaccess` `/cgi-bin` durch den Pfad des CGI-Verzeichnisses ersetzen.

Die Skript `sb_move_table` und `sb_cleanup` sind nicht für normale Besucher gedacht. Sie können sie in einem anderen geeigneten Verzeichnis installieren und sollten sie vielleicht auch in einem geschützten Verzeichnis installieren.

Perl-Installationspfad

Wenn Perl unter einem anderen Dateipfad als `/usr/bin/perl` installiert ist, müssen Sie den richtigen Dateipfad im Kopf der Skript-Dateien `cgi-bin/*` einsetzen.

Weitere Erklärungen für den Besucher

Die Datei `explain.xhtml` erklärt wortreich die Funktionsweise der Schachecke. Sie können diese Datei auf Ihren Server hochladen und auf sie verweisen, zum Beispiel in `SR_index.xhtml` oder `SD_more.xhtml`. Wenn Sie in einer Datei `SB_*.xhtml` oder `SD_*.xhtml` auf diese Erklärungsdatei verweisen, sollten Sie sie in `SD_explain.xhtml` umbenennen.

Das Verzeichnis `htdoc` enthält die Dokumentation im XHTML-Format. Sie können gern das komplette Verzeichnis `htdoc` auf Ihrer Website anbieten. Das Archiv `schach.tar.bz2` oder eine modifizierte Version sollten Sie bitte anbieten.

Herberts Schachecke: ein Brett

Brett-Datei

Eine Brett-Datei ist eine XHTML-Datei, deren Kopf einen Verweis auf `SR_schach.xslt` enthält. Der Dateiname entspricht dem Muster `SB_*.xhtml`. Das Sternchen steht für eine Folge von einem oder mehreren Kleinbuchstaben. Die Brett-Datei enthält Elemente, die durch die Transformation `SR_schach.xslt` besonders behandelt werden:

- Ein Element mit der ID `sb_menu.links`

  Ein Klick auf dieses Element zeigt unterhalb dieses Elements den Inhalt der Datei `SR_bretter.xhtml` an. Diese Datei enthält Verweise auf die verfügbaren Bretter. Wenn Sie ein Brett hinzufügen oder entfernen, sollten Sie `SR_bretter.xhtml` anpassen.

- Ein oder mehrere `div`-Elemente, deren Attribut `class` das Wort "`brett`" enthält.

  Ein solches Element repräsentiert ein Schachbrett. Eine normale Brett-Datei enthält genau ein solches Element. Ein zweites Brett zur Analyse kann sinnvoll sein. Die „Bretter” in einer Brett-Datei, die eine Verbindung zum Server haben, zeigen alle dasselbe Spiel.

Die Konfiguration eines Bretts

Zur Konfiguration eines Bretts dient das Attribut `data-use`. Der Wert ist eine durch Leerzeichen getrennt Folge von Optionen. Manche Optionen entsprechen Ein/Aus-Schaltern (logische Variable) Der Optionsname schaltet die Funktion ein, der Optionsname mit einem vorangestellten „`-`”-Zeichen schaltet die Funktion aus. Wenn der Optionsname sowohl mit als auch ohne das vorangestellte „`-`”-Zeichen in der Konfigurationseinstellung vorkommt, hat die „`-`”-Einstellung Vorrang.

`passive`

Das Brett zeigt die Spielsituation an, diese kann aber nicht geändert werden. Der Beobachter des Spiels („Kiebitz”) kann nicht ziehen, aufgeben, Remis anbieten oder die Spielsituation auf irgendeine Art ändern. Diese Option steuert, welche Anzeige-Element (Knöpfe) es gibt. Mit dieser Option gibt es rechts neben dem Brett nur die Knöpfe „Brett drehen” und „zur Analyse”. Ein Klick auf ein Feld oder einen Auswahl-Stein bleibt wirkungslos. Es wird keine Behandlungs-Funktion zugeordnet.

Mit dieser Einstellung hat die Variable `active` anfangs den Wert `0`. Die Einstellung `active` kann mit Vorrang einen anderen Wert der Variablen `active` festlegen, der geändert werden kann.

`local`

Das Brett hat keine Verbindung zum Server, nachdem die anfängliche Spielsituation (Stellung) geladen ist. Das Brett kann zur Analyse benutzt werden. Diese Option wird zu Anfang gesetzt, bleibt wirksam und kann nicht mehr geändert werden.

`active` / `-active`

Diese Einstellung entspricht der Variablen `active`. `-active` macht Klicks auf das Brett oder die Auswahlsteine wirkungslos und verhindert, dass Spielsituationen an den Server gesendet werden.

`val` / `-val`

In der Voreinstellung wird geprüft, ob ein eigener Zug den Regeln entspricht. Das entspricht der Option `val`.

Die Option `-val` schaltet diese Prüfung aus. Wenn Sie zusätzlich die Prüfung der empfangenen Stellung ausschalten (`-ana`), können Sie Räuberschach oder Schaf und Wolf spielen oder einfach die Aufmerksamkeit herausfordern.

`snow` / `-snow`

Nachdem der Server eine Spielsituation gemeldet hat, ist der Spieler (Besucher) normalerweise am Zug. Automatische Anfragen an den Server erfolgen dann normalerweise nur in großen Zeitabständen. Das ist die Voreinstellung und entspricht `-snow`.

Die Option `snow` bewirkt, dass die automatischen Anfragen gleich schnell erfolgen, unabhängig, ob der Spieler die Spielsituation selbst herbeigeführt (gezogen oder aufgegeben) hat oder ob der Server die Spielsituation geliefert hat und der Spieler nun am Zug ist. So kann ein erfahrener Spieler einem Anfänger zum Beispiel Zugfolgen zeigen.

`touch` / `-touch`

In der Voreinstellung wird geprüft, ob ein Stein auf einem „angeklickten" Feld ziehen kann. Wenn der Stein nicht ziehen kann, wird das Feld nicht ausgewählt. Das entspricht der Option `touch`.

Die Option `-touch` bewirkt, dass ein Stein der Farbe, die am Zuge ist, ausgewählt werden kann, auch wenn der Stein nicht ziehen kann.

`release` / `-release`

Was geschieht, wenn man ein besetztes Feld anklickt, während ein Feld ausgewählt ist, auf dem ein Stein derselben Farbe steht? In der Voreinstellung wird die Auswahl des ausgewählten Feldes aufgehoben, wenn die Option `strict` nicht wirkt oder der ausgewählte Stein nicht ziehen kann, und das angeklickte Feld wird neu ausgewählt. Das entspricht der Option `release`. Sonst muss man ein ausgewähltes Feld erneut anklicken, um die Auswahl aufzuheben. Das entspricht der Option `-release`.

`strict` / `-strict`

Die Option `strict` entspricht der Regel „gerührt - geführ". Ein Stein auf einem ausgewählten Feld muss gezogen werden, falls er ziehen kann. In der Voreinstellung kann man die Auswahl eines Feldes wieder aufheben und einen anderen Stein setzen. Das entspricht der Option `-strict`.

`king` / `-king`

Wenn in einer empfangenen Stellung eine Seite keinen König hat, beendet die Funktion `ana` das Spiel mit einem Fehler-Code. Das entspricht der Option `king`.

Gelegentlich möchte man Anfänger üben lassen, nur mit zwei Türmen den gegnerischen König mattzusetzen. Die Option `-king` macht es möglich. Die Funktion `ana` prüft nicht, ob die Könige vorhanden sind.

`ana` / `-ana`

In der Voreinstellung (entspricht `ana`) wird eine empfangene Stellung (Zug des Gegners) geprüft. ob der Spieler Matt oder Patt ist und ob wenigstens eine Seite überhaupt noch Mattsetzen kann (ausreichendes Material). Diese Überprüfung erfolgt in jedem Fall, wenn nur regelkomforme eigene Züge möglich sind (Option `val`). Der Spieler kann ziehen oder aufgeben. Was sollte er machen, wenn sein Gegner ihn pattgesetzt hat, das Patt aber nicht automatisch erkannt wird?

Die Einstellung `-ana` bedeutet, dass eine empfangene Stellung nicht geprüft wird, wenn eigene Züge nicht auf die Einhaltung der Spielregeln geprüft werden.

`autoremis` / `-autoremis`

In der Voreinstellung (entspricht `autoremis`) wird ein eigener Zug automatisch mit einem Remis-Angebot verbunden, wenn man nicht mehr mattsetzen kann. `-autoremis` schaltet das automatische Remisangebot aus.

`inter:`*`DURATION`*

Die automatischen Anfragen an den Server erfolgen, nachdem ein Spieler gezogen hat, zunächst in einem bestimmten Zeitabstand, später in einem Vielfachen dieses Zeitabstandes. Eine Überlastung des Servers soll

vermieden werden. Der anfängliche Zeitabstand zwischen zwei automatischen Anfragen ist auf eine Sekunde voreingestellt. Diese Option ändert den anfänglichen Zeitabstand auf *DURATION* Millisekunden. *DURATION* steht für eine ganze Dezimalzahl.

Wenn der Server in einem lokalen Netz läuft, in dem einige Spieler sind, dürfte ein Zeitabstand von 50 ms kein Problem sein. Vielleicht wollen manche zum Schutz vor einer Infektion lieber in einem lokalen Netz spielen als am Brett spielen.

`remis:`*`REPEATRR`*`:`*`MOVESRM`*

REPEAT und *MOVES* stehen für ganze Dezimalzahlen oder nichts, *RR* und *RM* für den Buchstaben „`r`" oder nichts.

Wenn dieselbe Spielsituation in einer Partie mehr als *REPEAT* mal vorkommt, kann die Partie remis enden. Wenn *RR* „`r`" ist, kann der Spieler Remis fordern, sonst endet die Partie automatisch remis. Wenn *REPEAT* `0` ist, kann dieselbe Spielsituation beliebig oft wiederholt werden, ohne dass die Partei automatisch remis endet oder der Spieler Remis wegen Stellungswiederholung fordern kann. Wenn *REPEAT* leer ist, bleibt die Einstellung für Remis nach Stellungswiederholung unverändert. Voreingestellt ist `2`.

MOVES ist die maximale Zahl aufeinanderfolgender Züge ohne eine dauerhafte Änderung der Stellung, bevor die Partie remis durch Überschreitung der Zugzahl endet. Eine dauerhafte Änderung ist der Verlust eines Rochaderechts, das Schlagen eines Steins oder ein Bauernzug. Voreingestellt ist 50. Nachdem zuletzt ein Stein geschlagen worden ist, können beide Seiten 50 „Ziehzüge" machen (kein Bauernzug, kein Schlagen). Wenn dann im 51. Zug ein Stein geschlagen oder ein Bauer gezogen wird, läuft die Partie weiter, bei einem anderen „Ziehzug" kann der Spieler Remis fordern, wenn *RM* „`r`" ist, sonst endet die Partie automatisch remis. Wenn *MOVES* `0`, ist ein Remis durch Überschreitung der Zugzahl ausgeschlossen. Wenn *MOVES* leer ist, bleibt die Einstellung für Remis durch Zugzahlüberschreitung unverändert.

`time:`*`START`*`+`*`FISCHER`*`-`*`BRONSTEIN`*`!`*`TAKT`*

Das Brett rechnet die Zeit auf von dem Moment, in dem eine neue Spielsituation angezeigt wird, bis zur Antwort (Zug) des Spielers. Wenn die verfügbare Zeit ausgeschöpft ist, meldet das Brett die Zeitüberschreitung an den Server und beendet damit eine Partie.

Die Platzhalter *START*, *FISCHER* und *BRONSTEIN* sind ganze Dezimalzahlen und stehen für Zeiten in Sekunden

START ist die anfangs verfügbare Zeit, *FISCHER* der Fischer-Bonus je Zug, *BRONSTEIN* der Bronstein-Bonus je Zug. Die Bedeutung der Zeiten ist in der Hilfe erklärt.

Die Prüfung auf Einhaltung der Bedenkzeit erfolgt, nachdem eine neue Stellung empfangen wird, wenn eine Spieler ein Schachfeld wählt (anklickt), und sonst *TAKT* Millisekunden nach einer Prüfung. Mit Glück kann ein Spieler die Zeit um bis zu *TAKT* Millisekunden überschreiten, ohne dass die Zeitüberschreitung erkannt wird.

Jeder der Platzhalter zusammen mit dem vorangehenden Zeichen `+`, `-` oder `!` ist optional, aber nicht die Reihenfolge der Platzhalter.

`initpos:`*`XX`*

XX ist eine Zeichenkette von Kleinbuchstaben und Ziffern. Sie gibt die Anfangsstellung an in dem Format, in dem auch Spielsituationen zwischen Browser und Server ausgetauscht werden.

Voreingestellt ist die normale Anfangsstellung.

`view:`*`V`*

Brett-Ansicht. Die möglichen Wert für *V* sind `w`, `b`, `wa`, `ba`, `a` oder die leere Zeichenkette.

Der Buchstabe `w` bedeutet, dass das Brett mit der weißen Seite unten angezeigt wird.

Der Buchstabe `b` bedeutet, dass das Brett mit der schwarzen Seite unten angezeigt wird.

Der Buchstabe `a` bedeutet, dass die Seite, die am Zug ist, unten angezeigt wird, nachdem eine neue Stellung empfangen worden ist. Wenn der Buchstabe `a` fehlt, wird das Brett nicht automatisch gedreht, nachdem eine neue Stellung empfangen worden ist.

`noinitld`

Wenn die Brett-Datei geladen ist, wird anfangs nicht automatisch die aktuelle Spielsituation vom Server geladen. Diese Einstellung wirkt nur einmal beim Laden der Brett-Datei.

Die Einstellungen `passive` und `local` können nicht im Verlauf eines „Besuchs" geändert werden. Es kann auch nicht geändert werden, ob mit oder ohne Zeitbeschränkung gespielt wird. Aber die einzelnen Zeiten können geändert werden.

Mehrere Bretter für ein Spiel

Es können mehrere Personen Bretter für dasselbe Spiel aufrufen. Die Adressen `http://`*`IHRE_DOMAIN/BASIS/`*`dorsten/SB_*.xhtml` zeigen zum Beispiel alle dasselbe Spiel „`dorsten`". Wenn eine dieser Personen zieht, dann können alle anderen mit einem Zug antworten. Wer am schnellsten zieht, „gewinnt" sozusagen. Das macht mit mehr als zwei Spielern nicht wirklich Spaß. Deswegen sollten alle anderen als die beiden Spieler das „Kiebitzbrett" `http://`*`IHRE_DOMAIN/BASIS`*`/dorsten/SB_kiebitz.xhtml` öffnen.

Die beiden Spieler brauchen aber nicht unter derselben Zeitbeschränkung zu spielen. So kann der stärkere Spieler unter der Adresse `http://`*`IHRE_DOMAIN/BASIS`*`/dorsten/SB_blitz.xhtml` unter einer strengen Zeitbeschränkung spielen, der schwächere Spieler kann unter der Adresse `http://`*`IHRE_DOMAIN/BASIS/`*`dorsten/SB_schnell.xhtml` mehr Bedenkzeit bekommen.

Das Brett in seiner Umgebung

Zwei Schachspieler begrüßen sich vor einer Partie, sie einigen sich, wer die weißen Steine führt, sie sehen, ob sie bereit sind, und wenn beide bereit sind, setzt der Schwarz-Spieler die Uhr des Gegners in Gang.

Eine entsprechende Kommunikation vor dem ersten Zug kennt das virtuelle „Schachbrett" nicht. Ich habe überlegt, einen Status „Spielbereit" zusätzlich einzuführen und den ersten Zug erst dann zuzulassen, wenn beide Spieler bereit sind. Ich habe mich entschieden, das „Brett" auf die Kernfunktionen zu beschränken, nämlich den Austausch und die Prüfung von Spielsituationen. Nun kann man argumentieren, dass „Zugbereitschaft" auch eine Art Spielsituation ist.

Das Brett ist in einem HTML-Dokument enthalten, das hier die Umgebung ist. Die Umgebung kann verlangen, dass sich beide Spieler anmelden, dass sie sich einigen, wer welche Steine führt, dass sie beide einen Knopf drücken, um anzuzeigen, dass sie bereit sind, kann sicherstellen, dass beide Spieler mit derselben Zeitbeschränkung oder wenigstens fairen Bedingungen spielen, und erst dann das Brett freigeben oder überhaupt erst anzeigen. Ich meine, dass es besser ist, alles, was über das reine Spiel hinausgeht, der Umgebung zu überlassen. Die Umgebung kann einfach sein, kann vielleicht auch ein „Online-Turnier" sein.

Wie auch immer die Umgebung ist, in fast allen Fällen muss das Skript der Umgebung mit dem Brett kommunizieren.

Herberts Schachecke: Browser-Server-Kommunikation

Die Spielsituationen

Unter einer normalen Spielsituation verstehe ich die Stellung zusammen mit dem Zugrecht, den Rochaderechten, dem En-passant-Schlagrecht und einem möglichen Remis-Angebot, das der Spieler, der am Zug ist, annehmen kann. End-Spielsituationen beenden eine Partie, zum Beispiel ein Matt, ein Patt, unzureichendes Material, Aufgabe oder Annahme eines Remis-Angebotes.

Eine Spielsituation wird durch eine Kette von Kleinbuchstaben und Ziffern von 1 bis 8 definiert. Technisch gesehen *ist* eine Spielsituation eine Zeichenkette von Kleinbuchstaben und Ziffern von 1 bis 8. Die Ziffern dienen zur Bezeichnung der Felder. Zwei aufeinander folgende Ziffern bezeichnen ein Feld. Die erste Ziffer 1 bis 8 bezeichnet die Linie a bis h, die zweite Ziffer ganz normal die Reihe.

Eine Spielsituation ist eine Aneinanderreihung einzelner Elemente. Es gibt drei Arten von Elementen:

zwei Buchstaben
ein Buchstabe und zwei Ziffern
zwei Ziffern

Normale Spielsituationen

In einer normalen Spielsituation werden die folgenden Elemente aus zwei Buchstaben benutzt:

`aw`

Die folgenden Elemente gelten für die weiße Partei.

`ab`

Die folgenden Elemente gelten für die schwarze Partei.

`ak`

Das Recht zur kurzen Rochade ist nicht dauerhaft verwirkt.

`aq`

Das Recht zur langen Rochade ist nicht dauerhaft verwirkt.

`am`

Die Partei ist am Zug.

`ai`

Die Partei ist am Zug, nachdem zuvor die Spielsituation unumkehrbar („irreversibel”) geändert worden ist. Unumkehrbare Änderungen sind Bauernzüge, Schlagen und Verlust eines Rochaderechts. Diese Kennzeichnung hilft, ein Remis gemäß der 50-Züge-Regel zu erkennen oder ein Remis durch dreimalige Wiederholung einer Spielsituation.

`rm`

Es besteht ein Remis-Angebot, das die Partei, die am Zug ist, annehmen kann.

Ein Buchstabe und zwei folgende Ziffern geben das Feld einer Figur oder das En-passant-Schlagfeld an. Die Buchstaben bedeuten:

`k`

König

`q`

Dame

`b`

Läufer

`n`

Springer

`r`

Turm

`e`

En-passant-Schlagfeld

Ein Element aus zwei Ziffern bezeichnet das Feld eines Bauern.

Die meisten Elemente gelten für eine Partei. Ihnen muss eines der Elemente `aw` oder `ab` vorangehen. Die Elemente, die unabhängig von einer zuvor genannten Partei sind, sind das En-passant-Schlagfeld `e99` und das Remis-Angebot `rm` sowie die Elemente `aw` und `ab` selbst.

Besondere Spielsituationen

Es gibt Spielzüge, die eine Schachpartie beenden, zum Beispiel weil sie den Gegenspieler mattsetzen oder pattsetzen oder weil nicht genug Material auf dem Brett bleibt oder weil eine Stellung einschließlich der damit verbundenen Zugrechte zum dritten Mal erreicht wird. Eine Partie kann aber auch erst nach dem letzten Spielzug enden, wenn der Gegenspieler ein Remisangebot annimmt oder aufgibt.

Für diese Schachecke endet eine Partie nach dem Muster der Remis-Annahme oder Aufgabe mit der Antwort des Gegenspielers auf den letzten normalen Spielzug. Im echten Schachspiel spielt ein Spieler zum Beispiel Dg2++, und die Partie ist beendet. Auf dem virtuellen Brett setzt ein Spieler Dg2, sein Gegenspieler antwortet: „Ich bin matt”, und die Partie ist beendet.

Die besonderen Spielsituationen, die eine Partie beenden, werden durch die folgenden Elemente bezeichnet:

`xa`

Ein Spieler nimmt ein Remis-Angebot an.

`xr`

Ein Spieler gibt auf.

`xm`

Ein Spieler ist mattgesetzt.

`xp`

Ein Spieler ist pattgesetzt.

`xu`

Remis wegen unzureichenden Materials

`xw`

Remis durch dreimalige Wiederholung einer Stellung

`xe`

Remis gemäß der 50-Züge-Regel

`xk`

Es fehlt wenigstens ein König.

`xt`

Ein Spieler verliert durch Zeitüberschreitung

`xg`

Ein Spieler überschreitet die Zeit, aber sein Geger kann ihn nicht mattsetzen.

`xx`

Ein Spieler bricht die Partie ab.

Ein Spieler kann durch Schaltknöpfe die Situationen `xa` oder `xr` herbeiführen. Die Situationen `xm`, `xp` und `xu` werden automatisch erkannt. Der letzte Zug wird entsprechend beantwortet. Auch die Situationen `xw` und `xe` können automatisch erkannt und beantwortet werden.

Die Situation `xx` wird in Ausnahmefällen durch die Umgebung vorgegeben.

In allen genannten besonderen Spielsituationen stellt der Browser automatische Anfragen an den Server ein. Das wird durch einen schwarzen Balken angezeigt.

Die Anfragen des Browsers

Die Daten der Anfragen des Browsers stecken im QUERY-String, der an die relative Adresse `SX_move` angehängt ist. Der QUERY-String der automatischen Anfragen ist `?t=ID`. Der Platzhalter `ID` steht für die letzte Spielsituations-Kennung, die der Server gesendet hat.

Wenn der Browser eine neue Spielsituation meldet, ist der QUERY-String `?t=ID&p=SITUATION`. Der Platzhalter `SITUATION` steht für die neue Spielsituation.

Die Antworten des Servers

Wenn die `ID`, die der Browser sendet, nicht mit der `ID` übereinstimmt, die der Server aktuell gespeichert hat, sendet der Server die neue `ID` mit der neuen Spielsituation. Der Antwort-Code ist 200.

Andernfalls sendet der Server die neue `ID`, wenn der Browser eine neue Spielsituation gemeldet hat. Der Antwort-Code ist 200.

Andernfalls sendet der Server nur den Antwort-Code 204.

Falls es einen Fehler beim Zugriff auf die Datenbank gibt, ist der Antwort-Code 503.

Wenn der Server erkennt, dass der QUERY-String falsch aufgebaut ist, ist der Anwort-Code 400.

Die Antwort ist einfacher Text. Die neue Kennung ist in der Antwort in der Form `<t>ID</t>` enthalten, die neue Spielsituation ist in der Antwort in der Form `<p>SITUATION</p>` enthalten.

Herberts Schachecke: URI und Dateipfade

Mehrere Spiele

Die Schachecke kann unter einer beliebigen Basis-URL (*BASIS*) installiert werden. Mehrere Spiele zur gleichen Zeit werden durch eine Spielkennung *SPIEL* in der URI unterschieden. Eine Spielkennung ist eine Folge von 1 bis 49 Kleinbuchstaben a bis z, Ziffern 0 bis 9 oder dem Unterstrich _.

Weitergeleitete und umgeleitete URI

Die URI, deren Pfad nicht einer Datei entspricht, sind folgendermaßen aufgebaut:

BASIS/*SPIEL*/SX_*SCRIPT*

wird weitergeleitet (rewrite) nach /cgi-bin/sb_*SCRIPT*?g=*SPIEL*. Der Query-String der Anfrage wird angehängt.

SCRIPT ist eine Folge von wenigstens einem Kleinbuchstaben. Die möglichen Werte sind in den nachfolgend aufgeführt.

BASIS/SX_*SCRIPT*

wird weitergeleitet (rewrite) nach /cgi-bin/sb_*SCRIPT*?g=DEFAULT. Der Query-String der Anfrage wird angehängt.

BASIS/*SPIEL*/SR_*

wird permanent umgeleitet (redirect, HTTP-Code 301) nach *BASIS*/SR_*.

BASIS/*SPIEL*/SD_*NAME*

wird weitergeleitet nach *BASIS*/SD_*NAME* *NAME* steht für eine Folge von einem oder mehreren Kleinbuchstaben. Diese Adressen haben Dokumente, auf die von „Brettern" verweisen wird und die umgekehrt auf Bretter verweisen. Bei diesen Verweisen bleibt das *SPIEL* erhalten.

BASIS/*SPIEL*/SB_*BRETT*

wird weitergeleitet nach *BASIS*/SB_*BRETT* *BRETT* steht für eine Folge von einem oder mehreren Kleinbuchstaben. Diese Adressen sind für verschieden konfigurierte „Bretter" vorgesehen, zum Beispiel für verschiedene Zeitbeschränkungen.

Alle URI im Einzelnen

BASIS/SB_index.xhtml

Die Startseite der Schachecke.

BASIS/SB_explain.xhtml

Eine wortreiche Erklärung der Funktionsweise

BASIS/GPL-3.txt

General Public License, Version 3

BASIS/schach.tar.bz2

Der Quelltext der Schachecke als kompimiertes tar-Archiv

BASIS/SD_help.xhtml

Hilfe zum Schachbrett für den Besucher der Schachecke

BASIS/SR_symbols/*.png

Diagramme der symbolischen Stellungen. Die Hilfe bindet diese Diagramme ein.

BASIS/SD_more.xhtml

Hinweise für den Besucher der Schachecke

BASIS/SB_analyze.xhtml , *BASIS*/SB_blitz.xhtml , *BASIS*/SB_dauer.xhtml , *BASIS*/SB_denk.xhtml , *BASIS*/SB_holz.xhtml , *BASIS*/SB_kiebitz.xhtml , *BASIS*/SB_klingel.xhtml , *BASIS*/SB_lern.xhtml , *BASIS*/SB_schnell.xhtml

Beispiele für verschieden konfigurierte „Schachbretter” Ein Besucher der Website sollte diese URI nicht direkt aufrufen, sondern zum Beispiel über *BASIS*/*SPIEL*/SB_schnell.xhtml mit Angabe eines „Spiels”.

BASIS/SB_test.xhtml

Ein besonderes Brett zum Testen der Brett-Funktionen. Es wird durch das Skript SR_test.js ergänzt. Diese URI sollte vorzugsweise unter Angabe eines speziellen Spiels (z.B. test) über *BASIS*/test/SB_test.xhtml aufgerufen werden.

BASIS/SR_test.js

Skript zum Test-Brett.

BASIS/SR_bretter.xhtml

Liste von Verweisen auf die verfügbaren Bretter. Diese Liste wird bei Bedarf zu einem Brett eingebunden.

BASIS/SR_schach.xslt , *BASIS*/SR_schach.js , *BASIS*/SR_schach.css

Stil-Dateien zu einem Brett, insbesondere für das eigentliche Brett und die Verweise auf die verfügbaren Bretter.

BASIS/SR_theme/*.svg

Bilder der Schachfiguren. Sie werden von Datei SR_schach.xslt eingebunden.

BASIS/*SPIEL*/SX_move?t=*ID* , /cgi-bin/sb_move?g=*SPIEL*&t=*ID*

Fragt an, ob im Spiel *SPIEL* die Stellung mit der Kennung *ID* noch aktuell ist.

BASIS/*SPIEL*/SX_move?t=*ID*&p=*POS* , /cgi-bin/sb_move?g=*SPIEL*&t=*ID*&p=*POS*

Teilt mit, dass im Spiel *SPIEL* die Spielsituation mit der Kennung *ID* lokal geändert ist. Die neue Spielsituation ist *POS*.

BASIS/*SPIEL*/SX_pos?p=*POS* , /cgi-bin/sb_pos?g=*SPIEL*p=*POS*

Die neue Spielsituation im Spiel *SPIEL* ist *POS* unabhängig von der bisherigen Spielsituation. Diese URI wird auf Anforderung der „Brett-Umgebung” aufgerufen, um eine bestimmte Spielsituation festzulegen. Die Antwort enthält die Kennung der Spielsituation.

BASIS/*SPIEL*/SX_set?p=*POS* , /cgi-bin/sb_set?g=*SPIEL*p=*POS*

Die neue Spielsituation im Spiel *SPIEL* ist *POS* unabhängig von der bisherigen Spielsituation. Diese URI wird im normalen Spiel nicht aufgerufen. Sie dient im Skript SR_test.js (SB_test.xhtml) zum Test. Sie wird auf Anforderung der „Brett-Umgebung” aufgerufen.

/cgi-bin/sb_move_table

Richtet zur Installation der Schachecke die Datenbank ein. Diese URI ist nicht für den normalen Besucher gedacht.

/cgi-bin/sb_cleanup

Löscht alte Spiele aus der Datenbank. Diese URI ist nicht für den normalen Besucher gedacht.

Herberts Schachecke: die Datenbank und die CGI-Skripte

Die Tabelle

Es wird eine Tabelle m mit den folgenden Feldern benutzt:

g VARCHAR(50)

Bezeichnet das Spiel und ist der Primärschlüssel.

Der Wert ist eine Folge von Kleinbuchstaben `a` bis `z`, Großbuchstaben `A` bis `Z`, Ziffern `0` bis `9` und Unterstrichen `_`.

Bisher sind Großbuchstaben nicht vorgesehen. Werte mit Großbuchstaben sind für besondere Zwecke reserviert, zum Beispiel Turniere.

t BIGINT UNSIGNED

Eine fortlaufende Zahl zur Unterscheidung der Spielsituationen.

Bei jeder Änderung wird die Zahl um 1 erhöht.

Der Browser sendet beim normalen Spiel eine neue Spielsituation zusammen mit dieser Zahl. Wenn die gespeicherte Zahl und die Zahl, die der Browser sendet, nicht übereinstimmen, dann hat der Browser eine Änderung der Spielsituation nicht erfasst. Die Spielsitaution, die der Browser sendet, wird verworfen.

p VARCHAR(120)

Die Spielsituation: eine Kette von Kleinbuchstaben und Ziffern.

d DATE

Hier wird bei jeder Änderung das Tagesdatum gespeichert. Anhand des Datums werden nicht mehr aktive Spiele erkannt und können gelöscht werden.

CGI-Skripte

`sb_move`

Dieses Skript behandelt das normale Spiel. Die Query-Parameter *g*, *t* und *p* entsprechen den Feldern der Tabelle. Der Parameter *g* ist erforderlich. Wenn *t* fehlt, wird der Wert `0` angenommen.

Die Antwort ist entweder nur der Status-Code 204 oder einfacher Text.

Wenn das Spiel *g* nicht gespeichert ist, wird ein neuer Datensatz mit *t* = 0 gespeichert. Falls der Browser keine Spielsitutation sendet, wird die leere Zeichenkette als Spielsituation gespeichert. Die Antwort ist „`<t>0</>`”.

Wenn der gesendete Wert und der gespeicherte Wert für t nicht übereinstimmen, ist die Antwort „`<t>`*T*`</t<<p>`*P*`</>`”. Die Platzhalter *T* und *P* stehen für die gespeicherten Werte in den Feldern t und g.

Wenn der Browser keine Spielsituation sendet, ist die Antwort der Status-Code 204.

Wenn der Browser eine Spielsitation sendet, wird die empfangene Spielsituation gespeichert und der Wert des Feldes t um 1 erhöht. Die Antwort ist „`<t>`*T*`</t<`”. *T* steht für den neuen Inhalt des Feldes t.

`sb_pos`

Dieses Skript speichert eine neue Spielsituation und antwortet mit der neuen Kennung der Spielsituation. Im Unterschied zum Skript `sb_move` sendet der Browser hier keine Kennung der bisherigen Spielsituation. Die Query-Parameter *g* und *p* entsprechen den Feldern der Tabelle und sind beide erforderlich. Die gesendete Spielsituation wird gespeichert und der Inhalt des Feldes t um 1 erhöht. (`envh`, Schlüssel `setup`)

Die Antwort ist:

Status 400, wenn einer der Parameter *g* oder *p* fehlt,
Status 503 bei einem Datenbank-Problem,
sonst Status 200 mit dem einfachen Text `<t>`*T*`</t>`. Der Platzhalter *T* steht für den Inhalt des Feldes t.

`sb_set`

Dieses Skript wird zum Test des Bretts von `SB_test.xhtml` (`SR_test.js`) aufgerufen. Im Beispiel `SB_mattk.xhtml` (`SR_mattk.js`) dient es dazu, das Spielgeschehen, das sonst auf den lokalen Rechner beschränkt bliebe, möglichen „Kiebitzen” mitzuteilen (`envh`, Schlüssel `push`) Im Unterschied zum Skript `sb_pos` antwortet der Server nicht mit der Kennung der neuen Stellung, sondern liefert nur den Status.

Die Query-String-Parameter *g* und *p* entsprechen den Feldern der Tabelle. Die gesendete Spielsitutation wird gespeichert und der Inhalt des Feldes t um 1 erhöht.

Die Antwort ist:

Status 400, wenn der Query-String-Parameter *g* fehlt oder wenn es keinen Datensatz mit diesem Schlüssel gibt,
Status 503, wenn es keine Verbindung zur Datenbank gibt oder wenn beim Speichern ein Fehler eintritt,
Status 204 sonst.

`sb_move_table`

Dieses Skript ist für den Betreuer der Schachecke gedacht. Es legt die Datenbank-Tabelle m an. Die Antwort ist ein kurzer Text.

`sb_cleanup`

Dieses Skript ist für den Betreuer der Schachecke gedacht. Es löscht Sätze in der Datenbank-Tabelle, die zehn Tage alt oder älter sind. Die Antwort ist ein kurzer Text.

Herberts Schachecke: SR_schach.xslt

Betroffene Elemente

Die Transformation SR_schach.xslt behandelt zwei HTML-Elemente besonders:

- das Element mit dem Attributwert id = "sb_menu.links"
- div-Elemente mit dem Attributwert class = "brett"

Alle anderen Elemente bleiben im wesentlichen unverändert. Im Kopf (head) und im Rumpf (body) werden am Ende zusätzliche Elemente eingefügt. Dazu wird im Kopf die benannte Vorlage head.content.schach aufgerufen und im Rumpf die benannte Vorlage body.content.schach.links.

Ein Brett-Element

Eine typische Brett-Datei enthält genau ein div-Element mit dem class-Wert brett („Brett-Element"). Es sind auch mehrere solche Elemente möglich. Das Element steht für die Darstellung eines Schachbretts. In ihm wird der entsprechende Inhalt eingesetzt. Vorhandener Inhalt wird ignoriert und ersetzt.

Die benannte Vorlage div.idvalue.schach liefert die ID des Brett-Elements.

ID-Werte der Elemente eines Bretts

id-Werte eingesetzter „Nachkommen"-Element beginnen mit dem id-Wert des Brett-Elements. In der folgenden Liste der id-Werte steht der Platzhalter *BID* für den id-Wert des Brett-Elements.

*BID*_sb

: Auswahlreihe für schwarze Steine

*BID*_bk

: Auswahlfeld schwarzer König

*BID*_bq

: Auswahlfeld schwarze Dame

*BID*_bb

: Auswahlfeld schwarzer Läufer

*BID*_bn

: Auswahlfeld schwarzer Springer

*BID*_br

: Auswahlfeld schwarzer Turm

*BID*_bp

: Auswahlfeld schwarzer Bauer

*BID*_*LR*

Ein Feld des Schachbretts. Die Buchstaben *L* und *R* stehen für Ziffern von 1 bis 8. *L* bezeichnet die Linie, links mit 1 beginnend. *R* bezeichnet die Reihe, unten mit 1 beginnend.

*BID*_sw

Auswahlreihe für weiße Steine

*BID*_wk

Auswahlfeld weißer König

*BID*_wq

Auswahlfeld weiße Dame

*BID*_wb

Auswahlfeld weißer Läufer

*BID*_wn

Auswahlfeld weißer Springer

*BID*_wr

Auswahlfeld weißer Turm

*BID*_wp

Auswahlfeld weißer Bauer

*BID*_btn1

Knopf „Brett drehen”

*BID*_btn2

Knopf „# Analyse”

*BID*_btn3

Knopf „Neu aufstellen”

*BID*_btn4

Knopf „Anfangsstellung”

*BID*_btn5

Knopf „Stellung merken”

*BID*_btn6

Knopf „Gemerkte Stellung”

*BID*_btn7

Knopf „# Analyse”

*BID*_btn8

Knopf „Zug zurücknehmen”

*BID*_btn9

Knopf „Aufgeben”

*BID*_btn10

Knopf „Remis anbieten / angeboten / annehmen”

*BID*_ampel

Farbbalken („Ampel”)

*BID*_time

Feld zur Anzeige der Restzeit

Class-Werte der Elemente

sb_brett

Kennzeichnet die Tabelle (`table`) der Schachbrettfelder und die Felder (`td`) selbst.

black

Bezeichnet ein schwarzes Schachfeld.

white

Bezeichnet ein weißes Schachfeld.

sb_button

Kennzeichnet das Feld (`div`) neben dem Schachbrett mit den Schaltknöpfen, dem Farbbalken („Ampel”) und der Restzeit-Anzeige.

sb_red

Roter Hintergrund für ein `p`-Element, wird für die „Ampel” benutzt.

sb_yellow

Gelber Hintergrund für ein `p`-Element, wird für die „Ampel” benutzt.

sb_green

Grüner Hintergrund für ein `p`-Element, wird für die „Ampel” benutzt.

sb_black

Schwarzer Hintergrund für ein `p`-Element, wird für die „Ampel” benutzt.

sb_black

Zeigt an, dass der Inhalt eines `p`-Elementes zentriert angezeigt werden soll, für die Anzeige der Restzeit benutzt.

`sb_hide`

Zeigt an, dass ein Element nicht angezeigt werden soll. Es wird für die Figurenreihen zur Auswahl (`table`) und den Abschnitt (`div`) mit den Verweisen auf die verfügbaren Bretter benutzt.

`sb_show`

Dieser Wert wird zum Ein- und Ausblenden des Abschnitts (`div`) mit den Verweisen auf die verfügbaren Bretter benutzt.

`sb_select`

Kennzeichnet ein Feld (`td`) zur Auswahl eines Steins.

`sb_piece`

Kennzeichnet ein Bild (`img`) eines Schachsteins.

`sb_sel`

Hebt ein ausgewähltes Feld (`td`) oder einen Knopf (`input`) hervor.

Benannte Vorlagen für den Aufruf durch Anpassungsdateien

Dadurch, dass Einfügungen in benannte Vorlagen erfolgen, ist es einfach, dieses Stylesheet anzupassen, indem es in ein Anpassungs-Stylesheet importiert wird.

`head.content.schach`

Die benannte Vorlage `head.content.schach` fügt Elemente im Kopf (`head`-Element) ein: Verweise auf die Dateien `SR_schach.css` und `SR_schach.js` sowie ein `script`-Element, das `windows.onload` eine Funktion zur Initialisierung zuweist. Der Verweis auf die CSS-Regeln `SR_schach.css` wird nicht eingefügt, wenn das `head`-Element ein `link`-Element mit dem Attributwert `id` = "`style_schach`" enthält. Die ECMAscript-Datei `SR_schach.js` wird nicht eingebunden, wenn das `head`-Element ein `script`-Element mit dem Attributwert `id` = "`script_schach`" enthält. Die Zuweisung einer Funktion an `window.onload` wird nicht eingefügt, wenn das `head`-Element ein `script`-Element mit dem Attributwert `id` = "`script_load`" enthält.

`body.content.schach.links`

Wenn das XHTML-Dokument ein Element („Öffner”) mit dem Attributwert `id` = "`sb_menu.links`" enhält, dann fügt die benannte Vorlage `body.content.schach.links` ein `div`-Element mit dem `id`-Wert `sb_divlinks` ein. Der Inhalt des eingefügten `div`-Elements ist das Wurzelelement der Datei `SR_bretter.xhtml`. Diese Datei enthält Verweise auf die verfügbaren Bretter, die der Besucher der Schachecke sehen soll.

Das eingefügte `div`-Element wird unterhalb das „Öffners” positioniert. Es ist anfangs verborgen. Ein Klick auf den „Öffner” zeigt es an oder verbirgt es wieder. Das Element wird auch verborgen, wenn der Mauszeiger „hinausläuft”. Dieses Verhalten definiert die Funktion `links` in der Datei `SR_schach.js`.

`div.idvalue.schach`

Herberts Schachecke: SR_schach.js

Die Funktionen der obersten Ebene

Die ECMAscript-Datei SR_schach.js definiert auf der obersten Ebene (window) drei Funktionen:

schach_onload ()

Diese Funktion initialisiert das Skript, sobald das Brett-Dokument vollständig geladen ist (window.onload). Die Funktion brett wird für jedes Brett-Element (class = "brett") aufgerufen. Die Funktion links wird mit dem Element mit dem id-Wert sb_menu.links als Parameter aufgerufen.

Wenn für das Brett-Dokument weitere Initialisierungen nötig sind, kann im Kopf (head) ein script-Element mit der ID script_load window.onload eine andere Funktion zuweisen.

links (*m*)

m ist das Element mit der ID sb_menu.links.

Ein Klick auf dieses Element zeigt die Liste der Verweise auf die verfügbaren „Bretter" an (div-Element mit der ID sb_divlinks) und verbirgt die Liste wieder. Die Liste wird auch wieder verborgen, wenn der Mauszeiger „hinausläuft" (mouseout).

Diese Funktion definiert dieses Verhalten.

brett (*elt*)

Diese Funktion definiert das Verhalten eines Brett-Elements. Sis ist komplex wie die Spielregeln des Schachspiels. Der folgende Abschnitt beschreibt einige Einzelheiten der Funktion.

Die Funktion brett (*elt*)

Diese Funktion ist der Kern dieses Skripts. Sie erlaubt dem Spieler zu ziehen, prüft die Züge des Spieler entsprechend den Schach-Spielregeln, kommuniziert mit dem Server, prüft die Spielsituationen, die der Server meldet, auf Matt, Patt, oder ob ein Matt überhaupt möglich ist, erfasst und prüft die Zeit, die ein Spieler für seine Züge benötigt, und erfüllt weitere Aufgaben. Die Aufgaben erfolgen zum Teil zeitlich parallel.

Auf der Ebene der Funktion sind die folgenden Variablen definiert:

br_elt

Repräsentiert das Brett-Element (div-Element), benutzt in der Funktion turn_board.

id

Der Wert des Attributs id des Brett-Elements. Er wird in den Funktionen turn_board, setp, put_piece, move_piece und clear_field benutzt und an envc als Parameter übergeben.

rm_u

Der Wert 1 bedeutet, dass wenigstens eine der Variable rm_m oder rm_r einen Wert größer als 0 hat. Der Wert wird in configure gesetzt und in ana_run gelesen.

rm_m

rm_m ist die maximale Anzahl aufeinanderfolgender umkehrbarer Züge, bis automatisch oder auf Forderung (rm_mr) Remis erkannt wird. Ein umkehrbarer Zug ist ein Zug, der weder ein Rochaderecht verwirkt noch einen Bauern versetzt oder einen Stein schlägt.

Ein Remis wegen zu vieler aufeinanderfolgender umkehrbarer Züge wird durch Zusammenarbeit erkannt: die Browser erkennen eigene unumkehrbare Züge und teilen dies in der Spielsituation dem Server mit.

Wenn rm_m = 0 ist, sind beliebig viele umkehrbare Züge erlaubt, ohne dass die Partie dadurch automatisch oder auf Forderung remis endet.

rm_mr

Der Wert 0 bedeutet, dass bei Überschreitung der Zugzahl rm_m (s. rm_n) die Partie automatisch remis endet, der Wert 1 bedeutet, dass in diesem Fall der Spieler Remis fordern kann.

rm_n

rm_n ist die aktuelle Anzahl aufeinanderfolgender umkehrbarer Züge des Gegners nach der letzten unumkehrbaren Änderung der Stellung.

rm_n wird bei einem eigenen unumkehrbaren Zug (irv_chg) = 0 gesetzt (on_sel_field). Bei einem gegnerischen Zug wird rm_n zunächst um 1 erhöht (acps). Wenn der Zug unumkehrbar ist, wird danach rm_n = 0 gesetzt (setp). Wenn rm_n größer ist als rm_m endet die Partie automatisch remis (rm_test, falls rm_mr 0 ist) oder kann der Spieler Remis fordern (rm_mr ist 1).

rm_r

rm_r ist die maximale Zahl, wie oft dieselbe Spielsituation in einer Partie eintreten darf, bevor die Partie automatisch remis endet (rm_rr = 0) oder der Spieler Remis durch Wiederholung fordern kann (rm_rr = 1).

Der Browser zählt nur die Spielsituationen, die der Server sendet. Wenn man als Spieler mehr als rm_r mal vor derselben Spielsituation steht, beendet der Browser die Partie automatisch remis (rm_test) oder erlaubt dem Spieler, Remis zu fordern.

Wenn rm_r = 0 ist, ist ein Remis durch Wiederholung nicht möglich.

rm_rr

Der Wert 0 bedeutet, dass die Partie automatisch remis endet, wenn der Spieler mehr als rm_r mal in derselben Spielsituation am Zug ist und rm_r nicht 0 ist. Der Wert 1 bedeutet, dass statt der automatischen Remis-Entscheidung der Spieler Remis fordern kann.

rm_ra

Der Wert 1 zeigt an, dass der Spieler Remis wegen Stellungswiederholung fordern kann. rm_ra wird in der Funktion rm_test = 1 gesetzt, in handle_remis ausgewertet und in acps = 0 gesetzt.

rm_h

Der Brower speichert die Spielsituationen, die der Server sendet, im Array rm_h. Dieses Array dient dazu, Remis durch Stellungswiederholung automatisch zu erkennen. Die gespeicherten Spielsituationen werden gelöscht, wenn man selbst einen unumkehrbaren Zug zieht (on_sel_field) oder bevor eine Stellung gespeichert wird, die durch einen unumkehrbaren Zug entstanden ist (setp).

rm_l

rm_l ist die Anzahl der gespeicherten Stellungen im Array rm_h. Um die gespeicherten Stellungen zu löschen, wird rm_l = 0 gesetzt.

Wenn acps eine neue Stellung empfängt, wird die Stellung unter rm_h[rm_l] gespeichert, rm_c[rm_l] = 1 gesetzt und rm_l um 1 erhöht.

rm_c

rm_c[i] gibt an, wie oft die Stellung rm_h[i] vorgekommen ist, s. rm_test.

tm_u

Diese und die folgenden Variablen, deren Namen mit tm_ beginnen, beziehen sich auf die Beschränkung der Bedenkzeit. tm_u zeigt an, ob überhaupt mit beschränkter Bedenkzeit gespielt wird. 0 bedeutet unbeschränkte

Bedenkzeit, `1` bedeutet beschränkte Bedenkzeit. Der Wert wird abhängig davon gesetzt, ob der Wert des Attributs `data-use` ein „Wort” der Form `time:`*`START`*`+`*`FISCHER`*`-`*`BRONSTEIN`* enthält.

`tm_g`

Die zu Anfang einer Partie verfügbare Zeit (*`START`*) in Millisekunden.

`tm_b`

Der Bronstein-Bonus (*`BRONSTEIN`*) je Zug in Millisekunden. Wenn ein Spieler zieht (am Ende einer Denk-Periode), wird diese Zeit von der Bedenkzeit für den Zug abgezogen. Die (vielleicht) verbleibende Restzeit wird von der insgesamt verfügbaren Zeit `tm_v` abgezogen.

Diese Variable wird in der Funktion `tm_start` benutzt sowie in der Funktion `tm_stop`, die sozusagen nach einem Zug „die eigene Uhr anhält”.

`tm_f`

Der Fischer-Bonus (*`FISCHER`*) je Zug in Millisekunden.

Jedesmal, wenn eine neue Spielsituation vorgegeben wird, (End-Spielsituationen ausgenommen), beginnt die neue Denk-Periode an (`acps`). Am Ende einer Denk-Periode wird der Fischer-Bonus zur verfügbaren Zeit `tm_v` addiert.

Diese Variable wird in der Funktion `tm_start` benutzt sowie in der Funktion `tm_stop`, die sozusagen nach einem Zug „die eigene Uhr anhält”.

`tm_v`

Die restliche verfügbare Bedenkzeit ohne Bronstein-Bonus (`tm_b`) und ohne den Fischer-Bonus (`tm_f`), benutzt in den Funktionen `tm_start` und `tm_stop`.

Der Anfangswert wird in der Funktion `reset` gesetzt.

`tm_a`

Die aktuelle Zeit (in Millisekunden-Auflösung) zu Beginn eines „Denkzeit-Intervalls”, des Denk-Zeitraums für einen Zug.

Der Wert wird in der Funktion `acps` an zwei Stellen auf die aktuelle Zeit gesetzt: bevor die Untersuchung der Stellung (`ana_run`) startet und nachdem die Stellung angezeigt ist, das Brett ggf. automatisch gedreht (`auto_view`) und der Farbbalken (`ampel`) umgeschaltet sind. Maßgeblich ist der spätere der beiden Zeitpunkte. Die Untersuchung der Stellung läuft asynchron, und nach der Untersuchung der Stellung startet die Zeitanzeige (`tm_start`), aber nur dann, wenn die Anfangszeit `tm_a` in diesem Moment nicht `0` ist. Die Zeitanzeige startet erst nach der Untersuchung der Stellung, weil sie mit einer Zeitkontrolle verbunden ist. Ein technisches Remis ist ein Remis, auch wenn die Bedenkzeit abläuft.

Der Wert `0` bedeutet, dass die eigene Bedenkzeit nicht läuft. Das wird in den Funktionen `tm_start` und `tm_stop` geprüft. Wenn `tm_a` = 0 ist, zeigt `tm_start` die aktuell verbleibende Bedenkzeit `tm_v` nur einmal an und aktualisiert die Anzeige nicht weiter. `tm_stop` macht in diesem Falle gar nichts. Im Spielmodus mit Zeitbeschränkung machen die Funktionen `on_sel_field` und `on_sel_piece` nichts, wenn `tm_a` = 0 ist.

`tm_a` wird in den Funktionen `tm_stop` und `reset` auf `0` gesetzt.

`tm_e`

Die aktuelle Zeit (in Millisekunden-Auflösung) am Ende eines Denkzeit-Intervalls. Der Wert wird gesetzt, wenn der Spieler ein Feld (`on_sel_field`) oder eine Auswahl-Figur zur Bauernumwandlung (`on_sel_piece`) wählt, und wird in der Funktion `tm_stop` ausgewertet.

`tm_c`

Die Zeit von einer Zeitkontrolle bis zur folgenden automatischen Zeitkontrolle, s. `tm_t`.

`tm_t`

`tm_t` ist der Timer („Zeitschaltuhr”) für die Aktualisierung der Anzeige der verbleibenden Bedenkzeit (`tm_start`). Er wird in `tm_start` angehalten und gesetzt und in `tm_stop` angehalten.

`tm_start`

Hält gegebenenfalls `tm_t` an und aktualisiert die angezeigte restliche Bedenkzeit einschließlich des Fischer-Bonus und des Bronstein-Bonus. Die Zeit wird auf ganze Sekunden abgerundet und in Stunden, Minuten und Sekunden angezeigt. Dabei prüft sie auch, ob die Bedenkzeit abgelaufen ist, und ruft in diesem Fall `tm_out` auf. Wenn die Bedenkzeit „läuft” (`tm_a` ist nicht `0`), wird `tm_t` zur regelmäßigen Aktualisierung gestartet.

`tm_start` wird in `reset` zur Anzeige der anfangs verfügbaren Zeit aufgerufen und nach der Untersuchung einer neuen Stellung (`ana_run`) zur laufenden Aktualisierung der Zeitanzeige. Diese Funktion setzt nicht die Anfangszeit eines Denkintervalls.

`tm_stop`

Diese Funktion bedeutet, dass ein Denkzeitintervall endet. Sie wird in `send` aufgerufen, bevor eine neue Spielsituation an den Server gesendet wird. Der Timer zur Aktualisierung der Zeitanzeige (`tm_t`) wird angehalten, der Fischer-Bonus zur verfügbaren Zeit addiert und die Dauer des Denkzeitintervalls abzüglich des Bronsteinbonus (`tm_b`) von der verfügbaren Zeit (`tm_v`) abgezogen. Falls die Zeit abgelaufen ist, wird `tm_out` aufgerufen. Statt der neuen Stellung wird dann die Zeitüberschreitung an den Server gesendet.

Falls aus irgendeinem Grund das Ende des Denkzeitintervalls (`tm_e`) noch nicht gesetzt sein sollte, setzt diese Funktion es auf die aktuelle Zeit.

`tm_out`

Wird in `tm_start` und `tm_stop` aufgerufen, wenn die Bedenkzeit abgelaufen ist. Das Spiel wird beendet (`mode` = 6), die Spielsituation „Zeitüberschreitung” als `posi`[3] gespeichert und `-0` als verbleibende Bedenkzeit angezeigt.

`sel_field`

`sel_field` ist das ausgewählte und hervorgehobene Feld (`td`-Element) des Schachbretts oder `null`.

Dem entspricht die Zahl `sel_fld`.

Die Funktion `on_sel_field` setzt `sel_field` und `sel_fld` „synchron”, `desel_field` setzt beide Variablen zurück auf den Wert `null` bzw. `0`.

`sel_field` wird nur in der Funktion `on_sel_field` benutzt.

`sel_fld`

Diese Zahl ist `0` oder bezeichnet das ausgewählte Feld `sel_field` im „88-Format” (s. `view`).

`sel_fld` wird in den Funktionen `on_sel_piece` und `on_sel_field` benutzt.

`sel_piece`

`sel_piece` ist das ausgewählte und hervorgehobene `td`-Element zur Auswahl eines Steins oder `null`.

`sel_piece` wird in der Funktion `on_sel_piece` gesetzt und in der Funktion `desel_piece` auf `null` zurückgesetzt.

`sel_piece` wird in den Funktionen `on_sel_piece` und `on_sel_field` benutzt.

`sel_btn`

`sel_btn` ist der ausgewählte und hervorgehobene Knopf (`input`-Element) oder `null`. Die Funktionen `new_setup` und `choose_pos` setzen `sel_btn`, `desel_btn` setzt `sel_btn = null`.

`re_sel`

Ein Wert des Attributs `class`, der dem regulären Ausdruck `re_sel` entspricht, kennzeichnet das HTML-Element als hervorgehoben. `re_sel` wird den Funktionen `sel_elt` und `desel_elt` benutzt.

`sel_elt` (*e*)

Kennzeichnet das HTML-Element *e* als hervorgehoben.

`sel_elt` wird in den Funktionen `new_setup`, `choose_pos`, `on_sel_piece` und `on_sel_field` aufgerufen.

`desel_elt` (*e*)

Hebt die Hervorhebung des HTML-Elements *e* auf.

`desel_elt` wird aufgerufen in den Funktionen `desel_field`, `desel_piece`, `desel_btn` und `on_sel_piece`.

`desel_field`

Hebt die Auswahl des Feldes `sel_field` auf.

`desel_field` wird aufgerufen in `turn_board`, `clear_board`, `on_sel_piece`, `acps` und `recv`.

`desel_piece`

Hebt die Auswahl des Steins `sel_piece` auf.

`desel_piece` wird aufgerufen in `clear_board`, `on_sel_piece`, `acps` und `recv`.

`desel_btn`

Hebt die Hervorhebung des Elements `sel_btn` auf.

`desel_btn` wird aufgerufen in `choose_pos`, `on_sel_piece`, `acps` und `recv`

`du`

Der Wert des Attributs `data-use`. Er wird zur Initialisierung von `active` und `connect` benutzt, zur Initialisierung der veränderbaren Einstellungen an `configure` übergeben, zur anfänglichen Ausrichtung des Bretts ausgewertet und vor dem erstmaligen Laden der Spielsituation vom Server geprüft, ob er das Wort `noninitld` enthält.

`active`

Diese und die folgenden Variablen entsprechen Konfigurations-Einstellungen im Attribut `data-use` des Brett-Elements..

Den Wert bestimmen die Konfigurations-Einstellungen `passive` und vorrangig `active`.

Wenn `active` = 0 ist, werden neben dem Brett nur die Knöpfe „Brett drehen” und „zur Analyse” angezeigt. Die Steine zur Auswahl werden verborgen. Weitere Einschränkungen verhindern, dass sich der Spieler aktiv beteiligen kann. Die Klick-Behandlungsfunktionen `on_sel_field` und `on_sel_piece` enden sofort, wenn `active` = 0 ist.

`active` wird in den Funktionen `configure`, `on_sel_field`, `on_sel_piece`, `switch_ampel`, `ana_run`, `ana`, `to_store`, `send`, `acps`, `recv` und `envh` (Schlüssel `enable`) ausgewertet.

`connect`

Der Wert ist `0`, wenn die Konfigurations-Einstellung das Wort `passive` enthält, sonst `1`

Der Wert `1` bedeutet, dass weitere Anfragen an den Server erfolgen, nachdem die Stellung erstmals geladen ist. Beim Wert `0` werden Züge weder an den Server gesendet, noch wird der Server nach Änderungen der Stellung gefragt (Analyse-Brett).

Wenn der Wert `0` ist, gibt es keine Zeitbeschränkung.

`connect` wird in den Funktionen `choose_pos`, `on_sel_piece`, `on_sel_field` und `recv` ausgewertet.

`inter`

Jede Anfrage belastet den Server und kostet den Spieler ein wenig Datenvolumen seines Internet-Tarifs. Deshalb sollten die automatischen Anfragen nur so schnell erfolgen, dass ein halbwegs flüssiges Spiel möglich ist. `inter` ist eine Zeitdauer in Millisekunden, die den Zeitabstand zwischen zwei automatischen Anfragen bestimmt (`recv`).

Für ein Kiebitz-Brett (s. `active`) erfolgen zwei Anfragen immer im Abstand von `inter` Millisekunden. Voreingestellt ist `3000`.

Sonst ist der voreingestellte Wert `1000`. Nachdem der Browser eine neue Stellung an den Server gesendet hat, erfolgen die automatischen Anfragen, innerhalb der ersten 30 Sekunden im Abstand von `inter` Millisekunden. Danach wird der zeitliche Abstand ein immer größeres Vielfaches von `inter` Millisekunden, bis er nach zwei Stunden das 900-fache von `inter` beträgt. In der Voreinstellung ist das eine Viertelstunde.

Nachdem der Server eine neue Stellung gesendet hat, ist man in einem normalen Spiel selbst am Zug. Die automatischen Anfragen erfolgen deshalb in größeren Zeitabständen unabhängig von `inter`: innerhalb der ersten 10 Minuten alle 30 Sekunden, danach werden die Zeitabstände größer, bis sie nach zwei Stunden 15 Minuten betragen.

Nur beim Lern-Brett (`snow`) erfolgen die automatischen Anfragen, nachdem eine neue Spielsituation übertragen ist, in den gleichen Zeitabständen, unabhängig davon, ob der Server (Gegner) oder der Browser (der Spieler) die geänderte Spielsituation sendet.

`doana`

Der Wert `1` (voreingestellt) bedeutet, dass eine empfangene Stellung (Zug des Gegners) geprüft wird, ob die eigene Seite Matt oder Patt ist oder ob überhaupt eine Seite genug Material hat mattzusetzen.

Der Wert `0` ist nur möglich, wenn auch `validate` = 0 ist. In diesem Fall wird die empfangene Stellung nicht geprüft. Das eigene Matt, Patt oder Remis durch unzureichendes Material wird nicht erkannt.

S. Einstellung `ana`

`validate`

Der Wert `0` bedeutet, dass nicht geprüft wird, ob ein Zug den Regeln entspricht. Die Funktion `on_sel_piece` erlaubt dann, für einen Bauern auf der letzten Reihe einen beliebigen Stein zu wählen, auch einen Stein der „falschen" Farbe oder einen zweiten König. Die Funktion `move` erlaubt es, jeden Stein auf ein beliebiges Feld zu setzen und jeden Stein zu schlagen, auch einen eigenen Stein.

Die folgenden Regeln für das En-passant-Schlagen gelten auch dann, wenn `validate` = 0 ist: Wenn ein Bauer genau zwei Felder nach vorn zieht, dann ist das Feld, das er überschreitet, das En-passant-Schlagfeld. Eine Bauernumwandlung ist nicht möglich, wenn ein Bauer mit einem Doppelschritt nach vorn auf die letzte Reihe zieht, wohl aber bei jedem anderen Schritt. Wenn im Gegenzug ein Bauer auf das En-passant-Schlagfeld zieht, dann wird der Bauer, der zuvor das En-passant-Schlagfeld überquert hat, vom Brett genommen.

Für eine Rochade gelten unabhängig von `validate` folgende Regeln: Wenn ein weißer König von e1 nach g1 zieht, dann wird ein weißer Turm von h1 nach f1 gesetzt. Wenn ein weißer König von e1 nach c1 zieht, dann wird ein weißer Turm von a1 nach d1 gesetzt. Entsprechende Regeln gelten für Schwarz.

Wenn `validate` = 1 ist, wird auch eine Stellung, die der Server sendet, geprüft (`ana_run`): das eigene Matt oder Patt oder Remis durch unzureichendes Material wird automatisch erkannt. Ob ein Matt oder Patt des Gegenspielers erkannt wird, liegt an den Einstellungen der Gegenseite.

S. Einstellung `val`

`snow`

Der Wert `1` („Lernbrett”) bedeutet, dass die Zeitabstände zwischen zwei automatischen Anfragen nicht davon abhängen, ob die aktuelle Stellung vom Server gemeldet worden ist (d.h. der Gegner hat gezogen) oder od man selbst die aktuelle Stellung herbeigeführt hat (`recv`).

S. Einstellung `snow`

`touch`

Der Wert `1` bedeutet, dass geprüft wird, ob der Stein auf einem „angeklickten” Feld überhaupt ziehen kann (`on_sel_field`). Wenn der Stein nicht ziehen kann, wird das Feld nicht ausgewählt.

Wenn `touch` = `0` ist, kann ein Stein ausgewählt werden, auch wenn der Stein gemäß den Regeln nicht ziehen kann. Wenn dazu auch `validate` = `0` ist, kann ein Stein jeder Farbe ausgewählt werden.

`touch` entspricht der Konfigurations-Option `touch`.

Der voreingestellte Wert ist `1`, wenn die Bedenkzeit beschränkt ist (`tm_u`) und eine Verbindung mit dem Server gehalten wird (`connect`).

`release`

Der Wert `1` (Voreinstellung) bedeutet: ein „angefasster” Stein (`sel_field`) wird automatisch losgelasen (die Auswahl aufgehoben), wenn ein anderes Feld angeklickt wird, das mit einem Stein derselben Farbe besetzt ist, und `strict` = 0 ist oder der zuerst ausgewählte Stein nicht ziehen kann (`a_c`), s. `on_sel_field`.

Der Wert `1` bedeutet, dass ein „angefasster” Stein erst durch einen zweiten Klick auf das Feld „losgelassen” werden muss, bevor ein anderer Stein ausgewählt werden kann.

`release` entspricht der Konfigurations-Option `release`. Benutzt in `on_sel_field`.

`strict`

Der Wert `1` entspricht der Regel „Gerührt - geführt”. Ein ausgewählter Stein (`sel_field`) muss gezogen werden, wenn ein Zug möglich ist (`a_c`). Der Wert `0` (Voreinstellung) bedeutet, dass man statt des ausgewählten Steins auch einen anderen Stein ziehen kann.

`strict` entspricht der Konfigurations-Option `strict`. Benutzt in `on_sel_field`.

`king`

In der Voreinstellung (Wert `1`) endet eine Partie mit einem Fehler, wenn in einer empfangenen Stellung ein König fehlt (`ana`). Beim Wert `0` läuft das Spiel auch ohne König weiter. So können Anfänger üben, mit zwei Türmen (ohne König) mattzusetzen.

`king` entspricht der Konfigurations-Option `king`. Benutzt in `ana`.

`posi`

Im Array `posi` werden Spielsituationen gespeichert.

`[0]`: die Anfangsstellung
`[1]`: die letzte empfangene Stellung (s. `undo`)
`[2]`: die gemerkte Stellung (s. `save_position`)
`[3]`: die End-Spielsituation im Fall `mode` = 6 oder eine Stellung aus dem lokalen Speicher (`from_store`)

`view`

Der Wert `0` bedeutet, dass das Feld a1 unten links angezeigt wird (aus der Sicht von Weiß), `1` bedeutet, dass das Feld a1 oben rechts angezeigt wird (aus der Sicht von Schwarz).

In der Funktion `put_piece` und anderen wird ein Feld durch Zahlen von 11 bis 88 (`fld`) bezeichnet. Die erste Ziffer im Bereich von 1 bis 8 steht für die Line a bis h, die zweite Ziffer für die Reihe. Das Attribut `id` des `td`-Elements unten links hat den Wert "*`id`*`_11`", das Attribut `id` des `td`-Elements oben rechts hat den Wert

"*id*_88". Dem Feld `fld` wird das `td`-Element mit dem `id`-Wert "*id_NN*" zugeordnet. Im Falle `view` = 0 steht *NN* für die Dezimaldarstellung von `fld`, im Falle `view` = 1 steht *NN* für die Dezimaldarstellung von 99 - `fld`.

`auto_view`

`auto_view` = 1 bedeutet, dass das Brett abhängig davon, welche Farbe am Zug ist, „gedreht" wird, wenn der Server eine neue Stellung liefert. Wenn Weiß am Zug ist, ist das Feld a1 unten links, wenn Schwarz am Zug ist, ist das Feld a1 oben rechts.

Anfangs ist `auto_view` = `active`. Die Funktion `turn_board` setzt `auto_view` = 0, wenn sie über die Schaltfläche „Brett drehen" aufgerufen wird. Die Funktionen `new_setup` („Neu aufstellen") und `set_initial_position` („Anfangsstellung") setzen `auto_view` = 1.

`auto_rem`

`configure` kann dieser Variablen den Wert `0` oder `1` (Voreinstellung) zuweisen.

`1` bedeutet, dass automatisch Remis angeboten wird (`remis = -1`), wenn in einer Spielsituation Mattsetzen nicht möglich ist (s. `setp`).

`turn_board`

Kehrt die Darstellung des Bretts um und setzt `auto_view = 0`, falls `turn_board` über den Knopf „Brett drehen" auf gerufen worden ist.

`configure` (*conf*)

Diese Funktion liest die Konfigurations-Einstellungen, die grundsätzlich im Verlauf einer Browser-Sitzung geändert werden können, aus der Konfigurations-Zeichenkette *conf*.

`configure` wird zur Initialisierung mit dem Wert des Attributs `data-use` als Parameter aufgerufen.

`kings_fld`

Ein Array mit zwei Komponenten: `kings_fld [0]` ist das Feld des weißen Königs, `kings_fld [1]` das Feld des schwarzen Königs im „88-Format" (s. `view`).

Die Werte werden in den Funktionen `put_piece` und `move_piece` gesetzt und in der Funktion `clear_board` auf `0` zurückgesetzt.

`ana` benutzt die Werte zur Prüfung einer neu empfangenen Spielsituation (Matt?), `move` nutzt die Stellung des „eigenen" Königs (des Königs der Farbe, die am Zug ist), um zu prüfen, ob ein Zug möglich ist.

`can_win`

Der Wert `0` bedeutet, dass der Spieler, der am Zug ist, nicht mattsetzen kann, weil ihm das nötige Material fehlt.

Die Funktion `clear_board` setzt den Wert `1`, `setp` setzt den Wert abhängig von der Stellung, `ana` prüft den Wert und beendet die Partie als Remis, wenn beide Seiten nicht mattsetzen können.

`can_los`

Der Wert `0` bedeutet, dass der Spieler, der am Zug ist, nicht mattgesetzt kann, weil der Gegenseite das nötige Material fehlt.

Die Funktion `clear_board` setzt den Wert `1`, `setp` setzt den Wert abhängig von der Stellung, `ana` prüft den Wert und beendet die Partie als Remis, wenn beide Seiten nicht mattsetzen können. Die Funktion `tm_out` beendet bei Zeitüberschreitung die Partie remis, wenn die Gegenseite nicht mattsetzen kann. Die Funktion `set_selfmade` aktiviert den Knopf „Aufgeben" nur dann, wenn die Gegenseite mattsetzen kann.

`b_undo`

Repräsentiert den Knopf „Zug zurücknehmen" und ist wie `b_resign`, `b_remis` und `ampel` nur dann nicht `null`, wenn `active` un `connect` beide nicht 0 sind und das Element auch wirklich da ist.

`b_resign`

Repräsentiert den Knopf „Aufgeben”

`b_remis`

Repräsentiert den Knopf „Remis anbieten / angeboten / annehmen” Def Knopf wird in der Funktion `set_selfmade` aktiviert oder deaktiviert.

`remis`

Die möglichen Werte sind:

`-1`: ich biete mit dem nächsten Zug Remis an. Der Knopf `b_remis` zeigt den Text „Remis angeboten” (Attribut `data-remis`).
`0`: ich (der Spieler) biete weder Remis an noch kann ich ein Remis-Angebot annehmen. `b_remis` zeigt den Text „Remis anbieten” (Attribut `data-offer`).
`1`: ich kann ein Remis-Angebot annehmen. `b_remis` zeigt den Text „Remis annehmen” (Attribut `data-accept`).
`2`: ich kann Remis fordern. `b_remis` zeigt den Text „Remis fordern” (Attribut `data-req`).

`remis` wird in der Funktion `set_remis` gesetzt und in den Funktionen `handle_remis` und `getp` gelesen.

`ampel`

Repräsentiert den Farbbalken (`p`-Element) zur Status-Anzeige.

`time`

Repräsentiert das Feld (`p`-Element) zur Anzeige der verbleibenden Bedenkzeit und ist nur dann nicht `null`, wenn `active`, `connect` und `tm_u` alle nicht 0 sind und das Element auch wirklich da ist.

`reset`

Diese Funktion setzt die verfügbare Zeit (`tm_v`) und die Zähler zur automatischen Remis-Entscheidung (`rm_n`, `rm_l`) auf ihre Anfangswerte zu Beginn einer Partie und setzt `irv_chg = 1`. Die Funktion wird aufgerufen,

wenn der Spieler den Knopf „Neu aufstellen” (`new_setup`) wählt,
wenn der Spieler den Knopf „Anfangsstellung” (`set_initial_position`) wählt,
wenn das Brett vom Aufstell-Modus (`mode` = 1) in den Spiel-Modus (2 oder 3) wechselt, weil ein Spieler zieht (`move`),
auf Anforderung der Umgebung (`envh` Schlüssel `reset`).

`waiting`

Der Wert `1` bedeutet, dass die Antwort auf eine Anfrage erwartet wird. Wenn `send` aufgerufen wird, während `waiting` = 1 ist, unterbleibt die Anfrage an den Server. Sonst setzen `send` oder `setup` den Wert auf `1`. `recv` setzt den Wert zurück auf `0`. `switch_ampel` schaltet den Farbbalken gelb, wenn `waiting` = 1 ist.

`tmpbreak`

Diese Variable ist dazu gedacht, die Kommunikation des Bretts mit dem Server vorübergehend zu unterbrechen. Sie wird auf Anforderung der Umgebung gesetzt (`envh`, Schlüssel `enable` / `break`) Bei einem positiven Wert ist der Server sozusagen abgeschaltet.

Wenn `tmpbreak` positiv ist, sendet `send` keine Anfrage an den Server, sondern erhöht `tmpbreak` um eins. Wenn die Unterbrechung endet (Schlüssel `enable` / `continue`), kann `envh` erkennen, ob die Unterbrechung eine Anfrage verhindert hat, und in diesem Fall eine neue Anfrage auslösen.

`changed`

Der Wert `1` bedeutet, dass die aktuell angezeigte Stellung gegenüber der Stellung, die zuletzt von `acps` vorgegeben worden ist, geändert und noch nicht bestätigt ist. Der Wert wird in der Funktion `switch_ampel` gesetzt, die den Farbbalken gelb schaltet, wenn `changed` den Wert `1` annimmt.

send setzt changed = 1 (durch einen Aufruf von switch_ampel), bevor eine neue Spielsituation an den Server gesendet wird. acps setzt changed = 0, recv setzt changed = 0, wenn der Server eine neue Kennung der Spielsituation gemeldet hat.

Wenn changed = 1 ist, bleiben ein Klick auf eine Schaltfläche zur Änderung der Stellung (choose_pos) oder ein Klick auf ein Feld des Bretts (on_sel_field) wirkungslos.

selfmade

Der Wert 1 bedeutet, dass die aktuelle Stellung durch eine eigene Aktion entstanden und nicht durch acps vorgegeben ist.

Wenn selfmade = 1 ist, ist es nicht möglich, aufzugeben (resign). Wenn selfmade = 0 ist, ist es nicht möglich, eigene Aktionen zurückzunehmen (undo). Abhängig vom Wert von selfmade werden die Knöpfe b_undo und b_resign aktiviert oder deaktivert. Dazu wird selfmade über die Funktion set_selfmade geändert.

acps setzt selfmade = 0, send setzt selfmade = 1, bevor eine neue Spielsituation an den Server gesendet wird.

Von selfmade hängen die Zeitabstände zwischen zwei automatischen Anfragen ab.

Der Wert 0 wird durch einen grünen Farbbalken (ampel), der Wert 1 durch einen roten Farbbalken angezeigt, sofern der Farbbalken nicht aus einem vorrangigen Grund gelb (Kommunikationsproblem) oder schwarz (Ende der Partie) ist (switch_ampel).

Die Funktion undo definiert eine eigene Aktion, die aber eine Stellung herstellt, die zuvor durch acps vorgegeben worden ist. Deshalb setzt undo selfmade = 0.

set_selfmade (*selfmade_neuer_wert*)

Setzt den Wert von selfmade, aufgerufen in undo, acps und send.

Der Knopf „Aufgeben” wird nur dann aktiviert, wenn die Gegenseite mattsetzen kann (s. can_los).

Der Knopf „Remis” wird aktiviert, wenn die Stellung nicht „selbstgemacht” ist.

switch_ampel (*changed_neuer_wert*)

Diese Funktion setzt den Wert von changed und aktualisiert den Farbbalken (ampel):

Schwarz bei Ende der Partie. In diesem Fall werden auch die Knöpfe b_undo, b_resign und b_remmis deaktiviert.
Gelb, wenn waiting oder changed = 1 sind.
Rot, wenn selfmade = 1 ist
sonst Grün

brett

brett ist ein „zweidimensionales” Array mit Indizes reihe, linie von 0 bis 7. reihe von 0 bis 7 entspricht den Reihen a bis h, linie entspricht der Linie „linie + 1”. Die Werte des Array sind leere Zeichenketten (für freie Felder) oder Zeichenketten *CP* aus zwei Buchstaben. Der erste Buchstabe *C* bezeichnet die Farbe eines Steins: w für Weiß oder b für Schwarz. Der zweite Buchstabe *P* bezeichnet die Art des Steins: p einen Bauern, r einen Turm, n einen Springer, b einen Läufer, q eine Dame und k einen König. Die Buchstaben orientieren sich an den englischen Bezeichnungen der Steine. brett repräsentiert immer die aktuelle Stellung. brett [reihe] [linie] zeigt an, ob das durch die Indizes bezeichnete Feld frei ist oder von einem Stein besetzt ist. brett [4][2] = "wk" bedeutet zum Beispiel, dass der weiße (w) König (k) auf dem Feld e3 steht.

Die sichtbare Stellung und brett werden dadurch synchron gehalten, dass die sichtbare Stellung und brett nur über die drei Funktionen put_piece, move_piece und clear_field geändert werden.

brett wird in den Funktionen is_attacked, can_move_or_take, ana, clear_board, getp und move ausgewertet.

`can_rk_wq`

Der Wert `1` bedeutet, dass Weiß das Recht zur langen Rochade nicht dauerhaft verwirkt hat (grundsätzliches Rochederecht). `0` bedeutet, dass Weiß das Recht zur langen Rochade dauerhaft verwirkt hat.

Die Funktion `clear_board` setzt diese Variable und die Variablen zu den übrigen Rochaderechten auf `1`, die Funktion `setp` zunächst auf `0` und dann entsprechend der einzustellenden Spielsituation. Die Funktion `move` setzt die Variable auf `0`, wenn der König zieht oder wenn ein Stein auf das Feld a1 zieht oder vom Feld a1 wegzieht. Wenn `validate` = 1 ist, benutzt `move` den Wert zur Prüfung, ob ein Zug möglich ist. `getp` liest die Variable als Teil der aktuellen Spielsituation.

Entsprechendes gilt für die anderen Variablen, die den grundsätzlichen Rochaderechten entsprechen.

`can_rk_wk`

Der Wert `1` bedeutet, dass Weiß das grundsätliche Recht zur kurzen Rochade hat.

`can_rk_bq`

Der Wert `1` bedeutet, dass Schwarz das grundsätliche Recht zur langen Rochade hat.

`can_rk_bk`

Der Wert `1` bedeutet, dass Schwarz das grundsätliche Recht zur kurzen Rochade hat.

`ep_fld`

Das En-passant-Schlagfeld. Nachdem ein Bauer einen Doppelschritt nach vorn gemacht hat, bezeichnet der Wert das Feld, das der Bauer überquert hat, im „88-Format" (s. `view`). Nach einem anderen Zug hat `ep_fld` den Wert 0.

Die Funktion `clear_board` setzt `ep_fld` = 0. `setp` setzt `ep_fld` entsprechend der Spielsituation. `getp` liest `ep_fld` als Teil der aktuellen Spielsituation, `move` setzt den Wert entsprechend dem Zug.

Die Funktion `ana` und die Hilfsfunktion `can_move_or_take` benutzen `ep_fld` zur Prüfung von Matt oder Patt. `move` benutzt den Wert zur Prüfung, ob ein Zug möglich ist.

`irv_chg`

Der Wert `1` bedeutet, dass die Stellung durch eine eigene Aktion unumkehrbar geändert ist. Eine solche Aktion kann ein unumkehrbarer Zug sein oder der Knopf „Neu aufstellen" (`new_setup`) oder „Anfangsstellung" (`set_initial_position`) oder der automatische Wechsel vom Aufstell-Modus (`mode = 1`) in den Spiel-Modus (`move`) oder eine Anforderung der Umgebung (`envh`, Schlüssel `reset`)

`irv_chg` wird in `clear_board` = 0 gesetzt, in `reset` auf den Wert `1`. `on_sel_field` löscht im Falle `irv_chg = 1` nach einem Zug die gespeicherten Spielsituationen (`rm_l`). `getp` wertet `irv_chg` aus.

`mode`

Die möglichen Werte sind:

`1`: Aufstell-Modus Der Spieler kann Steine auf das Brett stellen oder vom Brett nehmen.
`2`: Weiß ist am Zug
`3`: Schwarz ist am Zug
`4`: Weiß hat einen Bauern auf die 8. Reihe gesetzt und muss noch eine Umwandlungs-Figur wählen.
`5`: Schwarz hat einen Bauern auf die 1. Reihe gesetzt und muss noch eine Umwandlungs-Figur wählen.
`6`: Die Partie ist beendet. `posi [3]` ist die End-Spielsituation.

`pid`

Die Kennung einer Spielsituation: die Kennung, die der Server zuletzt gesendet hat.

`recv` setzt den Wert, und `send` sendet den Wert an den Server.

`timeac`

Die akkumulierte Zeit bis zu einer automatischen Anfrage seit der letzten Änderung der Spielsituation in Millisekunden. Von `timeac` hängt die Zeit bis zur nächsten automatischen Anfrage ab.

Die Funktionen `send` und `recv` setzen `timeac = 0`, wenn sie eine neue Spielsituation senden oder empfangen. `recv` addiert die vorangegangene Wartezeit auf.

`timer`

Der Timer („Zeitschaltuhr") für die nächste automatische Anfrage.

Die Funktion `recv` setzt `timer`, `clear_board` und `send` löschen den Timer.

`is_attacked` (*`fld`*, *`sf`*, *`tf`*, *`ep`*)

Diese Hilfsfunktion prüft, ob das Feld *`fld`* von einem Stein der Gegenfarbe, die nicht am Zug ist, bedroht ist, wenn das Feld *`sf`* und das Feld vor dem Feld *`ep`* frei sind und auf dem Feld *`tf`* ein eigener Stein steht.

Das ist die Situation, wenn ein eigener Stein vom Feld *`sf`* auf das Feld *`tf`* zieht. Falls ein Bauer en passant schlägt, ist *`ep`* das En-passant-Schlagfeld.

Die Parameter haben den Wert `0` oder bezeichnen ein Feld im „88-Format" (s. `view`).

Aufgerufen in `can_move_or_take`, `ana` und `move`.

`can_move_or_take` (*`ro`*, *`li`*, *`kf`*)

Die Parameter *`ro`* und *`li`* im Bereich von 0 bis 7 bezeichnen die Reihe und die Linie eines Feldes, *`kf`* das Feld des Königs im „88-Format".

Diese Funktion prüft, ob ein eigener Stein das bezeichnete Feld besetzen kann oder ob ein Bauer auf dem bezeichneten Feld en passant geschlagen werden kann. Diese Hilfsfunktion dient zum Test, ob durch die Besetzung des Feldes oder das Schlagen ein Schachgebot abgewehrt werden kann. Weil das En-passant-Schlagfeld nicht in der Wirkungslinie (gerade oder schräg) eines schachbietenden Steins liegen kann, wird nicht geprüft, ob ein Bauer durch En-passant-Schlagen das Feld besetzen kann.

`kf` dient dazu, Züge auszuschließen, die wegen einer Fesselung nicht möglich sind.

`can_move_or_take` wird in `ana` aufgerufen, wenn ein einfaches Schachgebot erkannt ist.

`a_d`

Diese Variable und die übrigen Variablen `a_*` beziehen sich auf die erste Untersuchung einer empfangenen Stellung.

Diese Variable zeigt an, ob die Werte `a_b` schon gültig sind (Wert `1`) oder noch nicht bestimmt (Wert `0`) sind. Erst wenn `a_b` gültige Werte hat, liefern die Funktionen `a_k` und `a_c` richtige Ergebnisse.

`a_d` wird in `ana_run` auf den Wert `0` und in `ana` auf den Wert `1` gesetzt.

`a_d` wird in der Funktion `on_sel_field` ausgewertet. Wenn der Wert dort nicht `1` ist, wird `on_sel_field` noch einmal zeitverzögert aufgerufen.

`a_b`

Das zweidimensionale Feld mit 3 x 3 Werten enthält Daten zu der Nachbarschaft des Königs. `0` bedeutet ein freies Feld, `1` ein Feld, das durch einen gegnerischen Stein bedroht ist, `2` ein Feld, das durch einen Stein der Farbe des Königs blockiert ist.

`a_b` wird in der Funktion `ana` bestimmt und in der Funktion `a_k` gelesen.

`a_s`

`a_s` ist die Anzahl der gegnerischen Steine, die dem eigenen König Schach bieten. Die möglichen Werte bei regelkonformem Spiel sind `0`, `1` und `2`.

a_s wird in der Funktion ana bestimmt und in der Funktion a_c gelesen.

a_k

a_k zeigt an, ob der König ziehen kann. Die möglichen Werte:

-1: noch nicht bestimmt
0: nein
1: ja

a_k wird in der Funktion ana_run mit dem Wert -1 belegt und in der Funktion a_t bestimmt und ausgewertet.

a_t

a_t testet, ob der König ziehen kann. a_t wird in a_c und ana aufgerufen

a_c (*sl*, *sr*)

sl und *sr* im Bereich von 0 bis 7 bezeichnen ein Feld. Das Ergebnis ist1 wenn auf dem Feld ein Stein steht, der ziehen kann, sonst 0.

a_c wird in ana und on_sel_field aufgerufen.

ana_run

Diese „Umschlagfunktion" kapselt den Aufruf von ana und, falls active nicht 0 ist, den folgenden Test des Partieverlaufs auf Remis (rm_test) und den anschließen Start der Anzeige der verbleibenden Bedenkzeit (tm_start).

Der Hauptzweck ist natürlich die laufende Anzeige der Zeit nach dem Abschluss der Untersuchung, die mit einer Prüfung auf Zeitüberschreitung verbunden ist.

acps ruft ana_run auf, nachdem eine neue Spielsituation angezeigt ist.

ana

Prüft eine vom Server empfangene Spielsituation auf Matt, Patt, unzureichendes Material und einige Fehler.

Dabei setzt sie einige Variablen, die benötigt werden, um später zu prüfen, ob in der gegebenen Situation ein Stein ziehen kann. a_d = 1 zeigt an, dass diese Variablen (a_*) gültige Werte haben.

Wenn die Funktion das Ende einer Partie erkennt und active nicht 0 ist, setzt sie mode = 6, speichert die End-Spielsituation in posi [3] und ruft send auf.

rm_test

Diese Funktion erkennt ein Remis aufgrund des Partieverlaufs. Sie wird in ana_run aufgerufen.

Wenn rm_m nicht 0 ist und rm_n größer ist als rm_m, endet die Partie im Falle rm_mr = 0 automatisch remis, weil zu viele Züge hintereinander erfolgten, ohne dass die Stellung unumkehrbar geändert ist. Andernfalls kann der Spieler Remis fordern.

Wenn mehr als eine Stellung in rm_h gespeichert sind, wird geprüft, ob die letzte Stellung rm_h[rm_l - 1] schon unter einem anderen Index i gespeichert ist. Wenn die Stellung schon unter rm_h[i] gespeichert ist, wird rm_l um 1 vermindert und rm_c[i] um 1 erhöht. Wenn dann rm_c[i] größer ist als rm_r, endet die Partie remis.

clear_board

Nimmt alle Steine vom Brett.

timer wird gelöscht, die Hervorhebung eines ausgewählten Feldes (desel_field) oder eines Auswahlsteins (desel_piece) wird aufgehoben, die grundsätzlichen Rochaderechte (s. can_rk_wq) werden gewährt, kings_fld, ep_fld, can_win, can_los und irv_chg werden auf ihre Anfangswerte gesetzt, mode auf den Wert 1 (Aufstell-Modus).

`clear_board` wird aufgerufen in `new_setup`, `show_result` und `setp`.

`new_setup`

Behandelt den Knopf „Neu aufstellen”.

Der Knopf wird hervorgehoben zum Zeichen, dass das Brett in den Aufstell-Modus kommt. Das Brett wird geleert (`clear_board`), das automatische Drehen eingeschaltet (`auto_view`) und die Zeit und Zähler auf die Werte zu Anfang einer Partie zurückgesetzt (`reset`).

`choose_pos` (*`event`*, *`ix`*)

Stellt die in `posi [`*`ix`*`]` gespeicherte Spielsituation auf, hebt den gewählten Knopf (`input`-Element) hervor (s. `sel_elt` und `sel_btn`), sendet die Spielsituation an den Server (`send`) und startet zeitverzögert die Untersuchung der Stellung (`ana_run`).

`event` ist das behandelte Eingabe-Ereignis.

`set_initial_position`

Behandelt den Knopf „Anfangsstellung”

Schaltet die automatische Ausrichtung des Bretts ein (`auto_view` = 1), stellt die Anfangsstellung auf (`choose_pos`) und setzt Zeit und Zähler zurück (`reset`).

Diese Funktion kann auch von `envh`, Schlüssel `set_initial_position` aufgerufen werden.

`undo`

Behandelt den Knopf „Zug zurücknehmen”.

Wenn die aktuelle Stellung nicht durch einen eigenen Zug entstanden ist (`selfmade`), macht diese Funktion nichts. Sonst stellt sie die letzte empfangene Stellung wieder her (`choose_pos`) und kennzeichnet sie als „nicht selbstgemacht” (`set_selfmade`).

`save_position`

Behandelt den Knopf „Stellung merken”

Speichert die aktuelle Stellung (`getp`) in `posi [2]`.

`set_saved_position`

Behandelt den Knopf „Gemerkte Stellung”

Stellt die gemerkte Stellung (`posi [2]`) wiederher (`choose_pos`).

`to_store`

Behandelt den Knopf „# Analyse„

Die aktuelle Stellung (`getp`) wird unter dem Schlüssel `pos` im lokalen Speicher `window.localStorage` gespeichert.

`from_store`

Behandelt den Knopf ”# Analyse„

Die im lokalen Speicher unter dem Schlüssel `pos` gespeicherte Stellung wird unter `posi[3]` gespeichert und angezeigt (`choose_pos`).

`resign`

Behandelt den Knopf ”Aufgeben„

Aufgeben ist nur möglich, wenn die aktuelle Stellung nicht selbst herbeigeführt ist. (`selfmade`)

`set_remis` (*r*)

Setzt den Wert der Variablen `remis` = *r* und stellt den Text des Remis-Knopfes (`b_remis`) ein.

`handle_remis`

Behandelt den Remis-Knopf.

Ein Remis-Angebot (`remis` = 1) wird angenommen. Falls Remis wegen Stellungswiederholung (`rm_ra`) oder wegen Überschreitung der Zugzahl (`rm_m` und `rm_n`) gefordert werden kann, endet die Partie remis. Sonst wird ein eigenes Remis-Angebot gegeben oder aufgehoben.

`show_result`

Behandelt einen Klick auf den (schwarzen) Farbbalken (`ampel`) im Fall `mode` = 6.

Es wird die symbolische Stellung angezeigt, die der End-Spielsituation in `posi[3]` entspricht.

`setp` (*pos*)

Zeigt die Spielsituation *pos* an.

Wenn *pos* eine End-Spielsituation ist, wird sie in `posi[3]` gespeichert und `mode` = 6 gesetzt. Die angezeigte Stellung wird weiterhin angezeigt.

Andernfalls wird die Stellung angezeigt. Dabei werden die Rochaderechte, und das En-passant-Schlagfeld (`ep_fld`) gesetzt. Es wird auch geprüft, ob jede Seite mattsetzen kann (`can_win` und `can_los`). Wenn die Stellung mit einem Remis-Angebot verbunden ist, wird dies angezeigt und `remis` = 1 gesetzt. Andernfalls wird `remis` abhängig von `auto_rem` und `can_win` gesetzt.

`getp`

Liefert die aktuelle Spielsituation.

Im Falle `mode` = 6 ist dies `posi[3]`, sonst gemäß den Variablen `brett`, `mode`, `ep_fld` `remis` und den Rochaderechten (`can_rk_wq` u.a.)

`put_piece` (*pc*, *fld*)

Setzt den Stein *pc* auf das Feld *fld*.

pc ist eine Zeichenkette von zwei Kleinbuchstaben, die einen Stein bezeichnen, s. `brett`. *fld* ist eine zweistellige Zahl von 11 bis 88, die im „88-Format” (s. `view`) ein Feld bezeichnet. Die Anzeige und `brett` werden „synchron” geändert. Wenn der gesetzte Stein ein König ist, wird `kings_fld` entsprechend geändert.

`put_piece` wird in den Funktionen `show_result`, `setp`, `move_piece` `on_sel_piece` (bei einer Bauern-Umwandlung), `move` und `on_sel_field` (im Aufstell-Modus) aufgerufen.

`move_piece` (*sfld*, *fld*, *p*)

Setzt einen Stein vom Feld *sfld* auf das Feld *fld*.

Die Parameter *sfld* und *fld* sind im „88-Format” (s. `view`). Wenn *p* definiert ist, dann muss der Stein von der Art *p* sein (s. `brett`). `brett` und die Anzeige werden „synchron” geändert.

`move_piece` wird in `move` aufgerufen.

`clear_field` (*fld*)

Wenn das Feld *fld* („88-Format”) nicht leer ist, wird der Stein auf diesem Feld entfernt. `brett` und die Anzeige werden „synchron” geändert.

`clear_field` wird in `clear_board` und in `move` aufgerufen.

`on_sel_piece`

Behandelt einen Klick auf einen Auswahl-Stein (Attribut `class` = "`sb_select`").

Die Wahl eines Auswahl-Steins wird in zwei Fällen behandelt: wenn ein Bauer auf der letzen Reihe durch eine Umwandlungsfigur zu ersetzen ist (`mode` = 4 oder `mode` = 5) oder im Aufstell-Modus (`mode` = 1).

Bei der Wahl einer Umwandlungsfigur wird zuerst die Endzeit eines „Denkintervalls" (`tm_e` gesetzt, falls die Bedenkzeit beschränkt ist (`tm_u`). Das Feld des Bauern ist in diesem Fall das hervorgehobene Feld (`sel_fld`). Im Fall `validate` = 1 wird geprüft, ob der gewählte Stein von der richtigen Farbe und weder ein König noch ein Bauer ist. Der gewählte Stein wird auf das Feld gesetzt (`put_piece`), die Auswahlen aufgehoben (`desel_piece`, `desel_field`) und die Stellung an den Server gesendet (`send`), falls nicht `connect` = 0 ist.

Wenn man im Aufstell-Modus auf den bereits ausgewählten Auswahl-Stein klickt, werden die Auswahl des Steins (`desel_piece`) und die Hervorhebung des Knopfes (`desel_btn`) aufgehoben und die aktuelle Stellung an den Server gesendet, sofern `connect` nicht 0 ist. Sonst wird statt des vielleicht bisher ausgewählten Steins (`sel_piece`) der angeklickte Auswahl-Stein neu ausgewählt.

`is_blocked` (*`sr`*, *`sl`*, *`tr`*, *`tl`*

Prüft, ob die Verbindung zwischen zwei Feldern durch einen Stein unterbrochen ist. Diese Hilfsfunktion wird aufgerufen, wenn `move` prüft, ob ein Zug den Regeln entspricht (s. `validate`).

Die Parameter sind Indizes im Bereich von 0 bis 7. *`sr`* und *`sl`* sind die Reihe und Linie des ersten Feldes (`brett[`*`sr`*`][`*`sl`*`]`), *`tr`* und *`tl`* sind die Reihe und Linie des zweiten Feldes.

`move` (*`sf`*, *`tf`*)

Zieht einen Stein vom Feld *`sf`* auf das Feld *`tf`*. Das Ergebnis bedeutet:

`0`: der Zug ist nicht möglich
`1`: der Zug ist ausgeführt
`2`: es ist noch eine Umwandlungs-Figur zu wählen

`sf` und *`tf`* sind das Ausgangsfeld und das Zielfeld im „88-Format". Wenn das Ausgangsfeld frei ist, ist das Ergebnis `0`. Sonst ist das Ergebnis nur dann `0`, wenn `validate` = 1 ist und der Zug nicht den Spielregeln entspricht.

Vom Aufstell-Modus (`mode = 1`) wechselt das Brett automatisch in den Zugmodus entsprechend der Farbe des Steins auf dem Ausgangsfeld (s. `reset`).

Wenn ein Bauer einen Doppelschritt nach vorn macht und das Zielfeld nicht das bisherige En-passant-Schlagfeld ist, wird das Feld, das der Bauer überschritten hat, das neue En-passant-Schlagfeld (`ep_fld`). Wenn ein Bauer auf das bisherige En-passant-Schlagfeld zieht, wird der Stein vor dem Zielfeld (aus der Sicht des Ziehenden) entfernt (`clear_field`).

Wenn ein Bauer einen Doppelschritt nach vorn macht, ist das Ergebnis `1` (Zug ausgeführt). Wenn andernfalls ein Bauer auf die letzte Reihe zieht, ist das Ergebnis `2` (Umwandlungsfigur wählen).

Wenn das Ausgangsfeld oder das Zielfeld ein Eckfeld ist, geht das entsprechende grundsätzliche Rochaderecht (s. `can_rk_wq`) verloren. Das Rochaderecht geht auch bei einem Zug des Königs verloren. Die Rochaderechte werden aber nicht geprüft, wenn `validate` = 0 ist.

Wenn der König von seinem Ausgangsfeld zwei Felder zur Seite zieht und in der Ecke, zu der sich der König bewegt hat, ein Turm der Farbe des Königs steht, dann wird der Turm auf das Feld auf der anderen Seite neben dem König gestellt.

Zur Überprüfung, ob ein Zug nach den Spielregeln möglich ist, werden die Hilfsfunktionen `is_blocked` und `is_attacked` aufgerufen.

`on_sel_field`

Behandelt einen Klick auf ein Feld des Schachbretts

Wenn die Bedenkzeit beschränkt ist (`tm_u`) und das Brett nicht im Aufstell-Modus ist (`mode`), wird das Ende des Denkzeit-Intervalls gesetzt (`tm_e`). Wenn die eigene Bedenkzeit nicht „läuft" (`tm_a`), geschieht nichts.

Es geschieht auch nichts, wenn die aktuelle Stellung noch nicht vom Server bestätigt ist (`changed`), eine Umwandlungsfigur zu wählen ist oder die Partie beendet ist (`mode`).

Wenn im Aufstell-Modus (`mode` = 1) ein Auswahlstein ausgewählt ist (`sel_piece`), wird ein solcher Stein auf das Feld gesetzt.

Wenn das Feld bereits ausgewählt ist (`sel_field`), wird die Auswahl aufgehoben (`desel_field`), sofern nicht `strict` = 1 ist oder der ausgewählte Stein nicht ziehen kann (`a_c`).

Wenn ein anderes Feld ausgewählt ist und `release` = 1 ist, das Feld mit einem Stein derselben Farbe wie die des ausgewählten Steins besetzt ist und `strict` nicht 1 ist oder der ausgewählte Stein nicht ziehen kann (`a_c`) und `touch` nicht 1 ist oder der neu angeklickte Stein ziehen kann, wird die Auswahl aufgehoben (`desel_field`) und das angeklickte Feld neu ausgewählt.

Wenn ein anderes Feld ausgewählt ist, wird sonst versucht, einen Stein vom ausgewählten Feld auf das neu gewählte Feld zu setzen (`move`) Wenn der Zug nicht möglich ist, geschieht nichts. Sonst wird die Feldauswahl aufgehoben. Wenn noch eine Umwandlungsfigur zu wählen ist, wird der entsprechende Modus gesetzt (`mode`), sonst die nach dem Zug erreichte Stellung an den Server gemeldet (`send`)

Wenn sonst das Feld frei ist, geschieht nichts.

Wenn sonst `touch` nicht 0 ist, wird geprüft, ob der Stein ziehen kann. Wenn das nicht der Fall ist, geschieht nichts.

Wenn sonst `validate` nicht 0 ist und die Farbe des Steins nicht am Zug ist, geschieht nichts.

Sonst wird das Feld neu ausgewählt und `mode` entsprechend der Farbe des Steins gesetzt.

`acps` (*p*)

„AcCept PoSition"

Diese Funktion stellt die neue Spielsituation *p* ein. Alle Auswahlen (Feld, Auswahlstein, Knopf) werden aufgehoben. Die Stellung wird als „fremdbestimmt" gekennzeichnet (`selfmade = 0`, s. `set_selfmade`), der „Remis-Zugzähler" `rm_n` erhöht und die Stellung angezeigt (`setp`). Eine normale Partie-Spielsituation (mit Zugpflicht für eine Seite) wird zur Erkennung von Remis durch Wiederholung gespeichert (`rm_r`, `rm_h`, `rm_c`, `rm_l`). Die Untersuchung der Stellung startet (`ana_run`), das Brett wird, falls nötig, „gedreht" (`auto_view`, `turn_board`), bei begrenzter Bedenkzeit wird der Fischer-Bonus (`tm_f`) zur verfügbaren Zeit (`tm_v`) addiert und die „Uhr" gestartet (`tm_a`). Die Stellung als ungeändert gekennzeichnet und der Farbbalken aktualisiert (`switch_ampel`). Wenn die Spielsituation nicht das Ende einer Partie kennzeichnet, wird sie für die Funktion „Zug zurücknehmen" (`undo`) gespeichert (`posi [1]`), andernfalls wird die Nachricht `game_over` gesendet.

`recv`

Behandelt das Ende einer Anfrage an den Server. Die Anfrage kann entweder durch Überschreiten der Antwortzeit oder durch eine Antwort des Servers enden.

Wenn die Antwortzeit überschritten ist, wird gleich noch einmal ein neue Anfrage gestartet (`send`. Weiter geschieht nichts.

Wenn der Server eine neue Spielsituation sendet, stellt `acps` die Spielsituation ein. Die Kennung (`pid`) der Spielsituation wird gespeichert und die akkumulierte Wartezeit (`timeac`) vor automatischen Aufrufen zurückgesetzt.

Wenn der Server die neue Kennung der Spielsituation (als „Empfangsbestätigung") sendet, werden alle Auswahlen (Feld, Auswahlstein, Knopf) aufgehoben, die Stellung als ungeändert (`changed`) gekennzeichnet und die Farbanzeige aktualisiert (`switch_ampel`).

Sonst („nichts Neues”) wird nur die Farbanzeige aktualisiert.

Wenn der Server keinen Fehler meldet und `active` und `connect` beide nicht `0` sind, wird die Wartezeit bis zur nächsten automatischen Anfrage bestimmt und zu `timeac` addiert.

Wenn die Partie nicht beendet ist und `connect` nicht `0` ist, startet zeitverzögert die nächste automatische Anfrage (`timer`).

`send` (*`c`*)

Sendet eine Anfrage an den Server.

`c` = 1 bedeutet, dass die Spielsituation lokal geändert ist.

Zuerst wird die „Zeitschaltuhr” (`timer`) für den automatischen Aufruf von `send` angehalten.

Wenn *`c`* nicht `0` ist und `active` logisch wahr ist (kein Kiebitz-Brett), wird die eigene „Schachuhr” angehalten (`tm_stop`), `selfmade` = 0 gesetzt und die „Ampel” aktualisiert (`switch_ampel`). Falls eine Partie endet, wird die Nachricht `game_over` gesendet (`envc`). Danach folgt die Nachricht `send_position`.

Jetzt wird geprüft, ob die Verbindung zum Server unterbrochen ist (`tmpbreak`). Wenn `tmpbreak` nicht Null ist, wird `tmpbreak` um `1` erhöht, und die Funktion `send` endet.

Die Anfrage-URI mit dem Query-String wird aufgebaut. Falls Stellung noch nicht durch den Server bestätigt ist, wird die akkumulierte Wartezeit `timeac` zurückgesetzt.

Wenn `waiting` = 0 ist, wird `waiting` = 1 gesetzt und die Anfrage gesendet.

`setup` (*`pos`*)

Sendet die Spielsituation *`pos`* unabhängig von der aktuellen Kennung (`pid`) an den Server, der sie als neue Spielsituation speichert. Die URI führt zum dem CGI-Skript `sb_pos`, das in der Antwort die neue Kennung der Spielsituation liefert.

Ein anstehender verzögerter Aufruf von `send` (`timer`) wird gelöscht. Falls noch eine Antwort des Servers aussteht (`waiting`), wird diese Funktion etwas später noch einmal aufgerufen. Sonst wird `waiting` = 1 gesetzt und die Anfrage gesendet. Die Funktion `recv` behandelt die Antwort des Servers.

Diese Funktion wird auf Anforderung der Umgebung aufgerufen (`envh`, Schlüssel `setup`). Sie ist nützlich für Vorgaben zum Beginn einer Partie.

`on_ampel`

Behandelt einen Mausklick auf den Farbbalken. Falls das Spiel beendet ist (`mode` = 6), wird die symbolische Stellung zum Ergebnis angezeigt. Wenn andernfalls die Stellung nicht lokal geändert ist (`changed`), wird die Spielsituation aktualisiert (`send`).

`envc` (*`id`*, *`key`*, *`val`*)

Wenn `envc` („environment: call”) nicht `null` ist, ist `envc` eine Funktion, mit der das Brett die Umgebung über bestimmte Ereignisse benachrichtigt. `envc` ist das Ergebnis des Aufrufs der Funktion `window.schach_neues_brett`, falls diese existiert (s. `envh`).

`id` ist der Wert des Attributs `id`, *`key`* eine Zeichenkette, die den Grund und den Zweck des Aufrufs bezeichnet, *`val`* ein Wert, dessen Bedeutung ebenso wie die Bedeutung des Ergebnisses von *`key`* abhängt.

`envh` (*`key`*, *`val`*)

Das Funktionenpaar `envc` und `envh` („environment: handle”) dient zur Kommunikation zwischen dem Brett und der Umgebung. Das Brett ruft `envc` auf, die Umgebung ruft `envh` auf.

Wenn die Funktion `window.schach_neues_brett` existiert, wird sie mit `envh` als drittem Parameter aufgerufen. So erfährt die Umgebung die Funktion `envh`, und das Brett erfährt die Funktion `envc`:

```
envc = window.schach_neues_brett (id, "new", envh)
```

Herberts Schachecke: die Kommunikation zwischen Umgebung und Brett

Die Kommunikation zwischen Brett und Umgebung erfolgt über zwei Funktionen: das Brett ruft `envc (id, key, data)` auf, um der Umgebung ein Ereignis mitzuteilen, die Umgebung ruft `envh (key, data)` auf, um das Brett zu „steuern".

Schlüssel für envh

Die Umgebung ruft die Funktion `envh` auf, um Informationen vom Brett zu bekommen oder das Brett zu steuern. Der erste Parameter *`key`* ist eine Zeichenkette, die die Bedeutung des Aufrufs festlegt (Schlüssel). Die Bedeutung des zweiten Parameters *`val`* hängt vom Schlüssel ab, ebenso die Bedeutung des Rückgabewertes. Die folgende Liste erklärt die Bedeutung der möglichen Schlüssel.

`response`

> *`val`* ist eine neue Spielsituation. `acps` stellt diese Spielsituation ein genauso wie eine Spielsituation, die der Server sendet.

`push`

> Sendet die im Parameter *`val`* übergebene Spielsituation an den Server. Der Server speichert die Spielsituation unabhängig von einer Kennung (`pid`, `SX_set [uri.dbk#uri.set]`). So können in bestimmten Anwendungen wie einem Demonstrationsbrett „Kiebitze" das Spielgeschehen verfolgen.

`setup`

> Stellt eine bestimmte Spielsituation auf und sendet sie an den Server. Die bisherige Kennung der Spielsituation wird nicht gesendet. Der Server antwortet mit der Kennung der neu gespeicherten Spielsituation, das Brett übernimmt die Kennung aus der Antwort des Servers. So kann eine „kontrollierten" Umgebung eine bestimmte Spielsituation vorgeben.
>
> Der Parameter *`val`* bestimmt die Spielsituation:
>
> `initial`
>
> > Die Anfangsstellung in der aktuellen Konfiguration.
>
> `current`
>
> > Die aktuelle Spielsituation
>
> *`POSITION`*
>
> > Sonst ist *`val`* die Spielsituation.

`reset`

> Setzt Zeit und Zähler auf die Ausgangswerte zu Beginn einer Partie, s. `reset`. Wenn *`val`* nicht `null` ist, wird `set_selfmade` mit *`val`* als Parameter aufgerufen. Eine Anwendung zeigt das Beispiel „Gewinner bleibt sitzen"

`configure`

> Falls *`val`* die leere Zeichenkette ist, ist das Ergebnis des Funktionsaufrufs eine Konfigurations-Zeichenkette, wie sie im Attribut `data-use` des Brett-Elements benutzt werden kann. Sie entspricht den wirksamen Einstellungen des Bretts.

Wenn *val* eine nicht leer Zeichenkette ist, dann wird *val* als Konfigurations-Zeichenkette interpretiert und die Einstellungen des Bretts geändert. Nicht alle Einstellungen können geändert werden.

`enable`

Ermöglicht dem Spieler unter Bedingungen, zu ziehen, und startet bei beschränkter Bedenkzeit die Uhr. Der zweite Parameter *val* legt eine Bedingung fest, unter der der Spieler ziehen kann oder nicht.

`w`

Der Spieler kann ziehen, wenn Weiß am Zug ist.

`b`

Der Spieler kann ziehen, wenn Schwarz am Zug ist.

`a`

Der Spieler kann ziehen, gleich ob Weiß oder Schwarz am Zug ist, oder ob nicht festgelegt ist, wer am Zug ist. Natürlich gelten die Spielregeln: wenn Weiß am Zug ist, kann der Spieler nur einen weißen Stein ziehen, wenn Schwarz am Zug ist, nur einen schwarzen Stein.

`s`

Spielende-Modus `mode` = 6. Nichts geht mehr.

`break`

Vorübergehend werden keine Anfragen des Bretts an den Server zugelassen. Die Variable `tmpbreak` wird mit `1` belegt. Mit dem zweiten Schlüssel `continue` werden Anfragen wieder zugelassen.

`continue`

Nachdem mit dem zweiten Schlüssel (Parameter *val*) `break` Anfragen des Bretts an den Server unterbunden sind, werden sie jetzt wieder zugelassen und gleich eine Anfrage gestellt, sofern durch die Unterbrechung eine Anfrage verhindert worden ist.

`start`

Setzt `tmpbreak = 0` und ruft `send` auf. So kann nach dem Ende einer Partie eine „Beobachtungsbrett" wieder aktiviert werden.

`query`

Der Rückgabewert ist eine aktuelle Einstellung des Bretts. Der zweite Parameter *val* bestimmt den Wert:

`time_difference`

Die Differenz zwischen der anfangs verfügbaren Zeit und der für den aktuellen oder den nächsten Zug verfügbaren Zeit in Millisekunden.

`total_bonus`

Die Summe von Fischer-Bonus und Bronstein-Bonus je Zug in Millisekunden.

`remaining_time`

Die für den aktuellen oder nächsten Zug verfügbare Bedenkzeit in Millisekunden. Wenn der Spieler am Zug ist, ist es die gesamte für den Zug verfügbare Bedenkzeit, nicht die verbleibende restliche Bedenkzeit. Der

Wert kann negativ sein, wenn der Spieler die Zeitbegrenzung überschritten hat. Dann kann er allerdings keinen weiteren Zug mehr machen.

`initial_time`

Die zu Anfang einer Partie verfügbare Zeit in Millisekunden. Im „normalen" Schach gibt es mitunter komplizierte Zeitbeschränkungen, die zum Beispiel für den ersten Zug höchstens 30 Minuten erlauben. Hier sind nur drei Werte zur Beschränkung der Bedenkzeit einzustellen: die anfangs verfügbare Zeit, der Fischer-Bonus, der vor jedem Zug zur verfügbaren Zeit addiert wird, und der Bronstein-Bonus, der nach jedem Zug von der für den Zug verbrauchten Zeit abgezogen wird.

`fischer_bonus`

Der Fischer-Bonus je Zug in Millisekunden.

`bronstein_bonus`

Der Bronstein-Bonus je Zug in Millisekunden.

`mode`

Der Wert der Variablen `mode`.

`technical`

Eine Bitfeld zur spieltechnischen Bewertung:

1: Gewinn ist möglich
2: Verlust ist möglich

Bei anderen Werten des Parameter *`val`* ist der Rückgabewert `null`.

`time_add`

Hier und in den folgenden `time_`*`XX`*-Schlüsseln ist *`val`* eine Zeitdauer in Millisekunden. *`val`* wird zu der verbleibenden Bedenkzeit `tm_v` addiert.

`time_grant`

Wenn die verbleibende Bedenkzeit (`tm_v`) kleiner ist als *`val`*, ist *`val`* die neue verbleibende Bedenkzeit. Nach dieser Mitteilung ist die neue verbleibende Bedenkzeit also mindestens *`val`*.

`time_limit`

Wenn die verbleibende Bedenkzeit (`tm_v`) größer ist als *`val`*, ist *`val`* die neue verbleibende Bedenkzeit. Nach dieser Mitteilung ist die neue verbleibende Bedenkzeit also höchstens *`val`*.

`time_set`

Wenn die neue verbleibende Bedenkzeit (`tm_v`) ist *`val`*,

`set_initial_position`

Wenn der zweite Parameter *`val`* logisch wahr ist, ist er die neue Anfangsstellung. Die Funktion `set_initial_position` wird aufgerufen, genauso als wenn der Knopf „Anfangsstellung" gewählt würde.

Schlüssel für envc

Das Brett ruft `envc` auf, um Nachrichten an die Umgebung zu senden. Der erste Parameter *`id`* ist immer die ID des Bretts (Attribut `id`), der zweite Parameter *`key`* ist eine Zeichenkette, die die Nachricht bestimmt (Schlüssel), die

Bedeutung des dritten Parameters *val* hängt vom Schlüssel ab, ebenso die Bedeutung des Rückgabewertes. In jedem Fall aber gilt, dass der Rückgabewert `0` immer für das normale Verhalten des Bretts steht. Natürlich kann `envc` durch Aufrufe von `envh` die Einstellungen des Bretts direkt ändern.

Die folgenden Liste erklärt die Bedeutung der möglichen Schlüssel.

`new`

Das XHTML-Dokument ist geladen, und die Funktion `brett` zur Initialisierung eines Bretts steht am Ende der Ausführung davor, beim Server die Spielsituation abzufragen. *val* ist die Funktion `envh`.

`reset`

Die Funktion `reset` setzt alle Variablen auf die Werte zu Beginn einer Partie. Der Parameter *val* zeigt an, ob die Bedenkzeit beschränkt ist (`tm_u`).

`set_initial_position`

Der Knopf „Anfangsstellung” ist gedrückt (Funktion `set_initial_position`). Bei einem anderen Rückgabewert als `0` kehrt die Funktion `set_initial_position` zurück, als wäre der Knopf nicht gedrückt worden.

`store_position`

Der Spieler wählt den Knopf „zur Analyse”. Der Parameter *val* enthält die aktuelle Spielsituation. Diese Aktion kann zum Beispiel in eine Aufzeichnung des Spielverlaufs aufgenommen werden. In einer aufbereiteten Darstellung des Spielverlaufs könnte dann ein Diagramm eingefügt werden.

Wenn der Rückgabewert logisch wahr ist, endet die Funktion `to_store` unmittelbar, ohne die Spielsituation lokal zu speichern.

`accept_position`

Die Funktion `acps` gibt eine neue Spielsituation vor, die normalerweise durch einen Zug des Gegenspielers entstanden ist. *val* ist die neue Spielsituation.

Die Nachricht `accept_position` wird gesendet, unmittelbar nachdem die neue Stellung angezeigt ist (`setp`). Die Analyse der Stellung ist noch nicht gestartet, das Brett ist nicht gedreht, die „Uhr” läuft noch nicht, der Farbbalken noch nicht aktualisiert. Wenn der Rückgabewert logisch wahr ist, endet die Funktion `acps` sofort.

`send_position`

Die Funktion `send` steht davor, eine neue Spielsituation an den Server zu senden. *val* ist die Spielsituation.

Beim Rückgabewert `1` sendet `send` die Spielsituation nicht. In diesem Fall kann `envh` mit dem Schlüssel `response` einen Antwortzug liefern.

`game_over`

Diese Nachricht wird gesendet, wenn eine Partie endet. Das kann geschehen, wenn die Gegenseite das Ende der Partie mitteilt (`acps`) oder bevor das Ende der Partie der Gegenseite mitgeteilt wird (`send`). Im zweiten Fall erfolgt die Nachricht `game_over` vor der Nachricht `send_position`. Wenn der Rückgabewert nicht Null ist, wird weder die Nachricht `send_position` gesendet, noch wird das Ende der Partie an den Server gesendet.

val ist der Code für den Ausgang der Partie.

Herberts Schachecke: Beispiele

Die hier beschriebenen Beispiele zeigen, wie das „Brett” in eine Umgebung eingebettet werden kann und mit der Umgebung kommunizieren kann.

Die Kommunikation zwischen Brett und Umgebung erfolgt über zwei Funktionen: das Brett ruft `envc (id, key, data)` auf, um der Umgebung ein Ereignis mitzuteilen, die Umgebung ruft `envh (key, data)` auf, um das Brett zu „steuern”.

Wenn die genannten Dateien nicht im Wurzelverzeichnis des Archivs liegen, dann finden Sie sie im Verzeichnis `doc/examples`. Um sie auszuprobieren, müssen Sie die Dateien in das Basisverzeichnis Ihrer Installation der Schachecke kopieren. Manche Verweise auf Beispiele finden Sie auch in meiner Schachecke (`http://herbaer.de/schach/ausprobieren/SD_more.xhtml`).

Einstellungen

Dateien `SB_confable.xhtml` und `SR_confable.js`

Die Datei `SB_confable.xhtml` enthält ein Formular zur Eingabe der Bedenkzeit, Regeln zur automatischen Remis-Entscheidung und zum Ziehen. Das Skript `SR_confable.js` „belebt” das Formular.

`SR_confable.js` definiert auf „Fenster”-Ebene die Funktion `schach_neues_brett`. Die Funktion `brett` in `SR_schach.js` ruft sie für jedes „Brett-Element” auf. In diesem Beispiel `SB_confable.xhtml` gibt es genau ein Brett und genau ein Formular. Im Hinblick auf mehrere Bretter in einem Dokument achte ich auf eine klare Zuordnung des Formulars zu dem Brett. Ich gebe dem Brett-Element die ID (Attribut `id`) `b1` und dem Formular die ID `b1_form`.

`schach_neues_brett` wird mit drei Parametern *`id`*, *`key`* und *`val`* aufgerufen. Grundsätzlich könnte diese Funktion alle Ereignisse behandeln, die das Brett meldet. Ich ziehe es vor, dass diese Funktion nur das Ereignis "`new`" (Parameter *`key`*) behandelt. So meldet sich sozusagen ein Brett an.

Der erste Parameter *`id`* ist die ID des Brett-Elements, hier "`b1`", der zweite Parameter *`key`* die Zeichenkette "`new`" und der dritte Parameter die Funktion `envh` des Bretts, über die sich die Umgebung an das Brett wendet. `schach_neues_brett` speichert diese Funktion in der Variablen `cb` („callback”).

`schach_neues_brett` speichert den Knopf „Neu aufstellen” (`btn_initpos`), positioniert das Formular über dem Brett, speichert die Eingabefelder des Formulars (`inp["`*`NAME`*`"]`, *`NAME`* steht für das Attribut `name` des Eingabefeldes) und ordnet den Formular-Knöpfen (`button`-Elementen) die Behandlungsfunktion `on_button` zu und gibt eine Funktion `handler` als Ergebnis zurück. Diese Funktion speichert das Brett in der Variablen `envc` und ruft sie auf, um Ereignisse an die Umgebung zu melden.

Wenn der Spieler den Knopf „Anfangsstellung” drückt, ruft das Brett die Funktion `handler` auf. Der erste Parameter *`i`* ist die ID des Bretts, der zweite Parameter *`k`* die Zeichenkette "`set_initial_position`" und der dritte Parameter in diesem Falle undefiniert.

Jetzt durchläuft das konfigurierbare Brett eine Folge von drei Zuständen. Technischer gesagt, durchläuft die Variable `handling` der Funktion `schach_neues_brett` die drei Werte `2`, `1` und wieder den Anfangswert `0`.

Angenommen, `handling` hat den Anfangswert `0`. Der Spieler drückt den Knopf „Anfangsstellung”, und die Funktion `handler` wird mit dem Schlüssel `set_initial_position` aufgerufen. `handler` liest die aktuellen Brett-Einstellungen (`configure`), stellt die Inhalte der Eingabefelder ein und zeigt das Formular an. `handling` bekommt den Wert `2`. Das bedeutet, dass das Formular angezeigt ist. Wenn der Spieler jetzt noch einmal den Knopf „Anfangsstellung” drückt, geschieht nichts, denn `handler` gibt sofort `1` zurück. Erst muss das Formular geschlossen werden.

Der Spieler prüft und ändert vielleicht die Einstellungen und drückt dann den Knopf „OK” oder „Abbrechen”. Beide Knöpfe rufen die Funktion `on_button` auf. Diese Funktion verbirgt erst einmal das Formular. Dann prüft sie, ob der Knopf „OK” (`name` = "`ok`") oder der andere Knopf („Abbrechen”) gedrückt worden ist.

Wenn der Knopf „Abbrechen” gedrückt worden ist, wird `handling = 0` gesetzt, und alles ist fertig.

Wenn der Knopf „OK” gedrückt worden ist, wird dem Brett eine Konfigurations-Zeichenkette entsprechend den Eingabefeldern gesendet (Schlüssel `configure`) und der Knopf „Anfangsstellung” sozusagen virtuell gedrückt.

- Und nichts geschieht. `handling` hat noch den Wert `2`, und `handler` gibt den Wert `1` zurück, damit eben nichts geschieht. Dann setze ich eben zuvor `handling = 0`,

- und das Eingabeformular erscheint gleich wieder, wenn ich den Knopf „OK” drücke. Auch nicht gut.

Es sind wirklich drei Zustände zu unterscheiden. Ich setze `handling = 1`, Bedeutung „Einstellungen bestätigt”. `handler` setzt dann `handling = 0` und liefert den Rückgabewert `0` („weitermachen”).

Zwei einstellbare Bretter

Dateien `SB_doppel.xhtml`, `SR_confable.xslt`, `SR_confable.css`, `SR_confable.js` und `SB_mixeddouble.xhtml`

Natürlich kann ein XHTML-Dokument zwei oder noch mehr „Bretter” enthalten. Auf zwei Brettern kann man dann zum Beispiel mit Zeitbeschränkung gegen sich selbst spielen. Die beiden Bretter verhalten sich genauso, als säßen die Spieler in Schermbeck und Holsterhausen: sie kommunizieren über den Server. Das Beispiel `SB_doppel.xhtml` enthält zwei Bretter in einem „flex”-Container. Wenn das Browserfenster breit genug ist, erscheinen die beiden Bretter nebeneinander, sonst untereinander. Aber wir haben doch alle breite Bildschirme?

Nun soll die Bedenkzeit bei beiden Bretter einstellbar sein wie in dem vorangehenden Beispiel. Wenn man nur ein einziges Formular einfügen will, dann muss man auf „Fenster-Ebene” nachverfolgen, für welches der beiden Bretter das Formular geöffnet ist und Konflikte so oder so verhindern. Das ist nicht schön. Und zweimal das gleiche Formular einfügen ist einfacher, aber auch nicht schön. Ich halte es für die richtige Lösung, die Vorlage für ein Brett um das Formular für die Einstellungen zu ergänzen. Wie das geht, zeigt die Datei `SR_confable.xslt`. Die benannten Vorlagen in `SR_schach.xslt` machen es einfach. Das ist auch eine Art Umgebung des Brettes.

Die CSS-Regeln für das Formular sind in die Datei `SR_confable.css` ausgelagert, die Skript-Datei `SR_confable.js` ist unverändert. Ich habe es mir einfach gemacht: die Datei `SR_confable.xslt` fragt, ob das Attribut `class` eines Brett-Elements genau die Zeichenkette „`brett confable`” enthält, nicht mit einem zusätzlichen Leerzeichen zwischen den Wörtern, nicht in umgekehrter Reihenfolge der Wörter. Aber ist kein ernstes Problem.

Und wenn nur bei einem Brett die Zeit einstellbar sein soll? Das „gemischte Doppel” (`SB_mixeddouble.xhtml`) ist kein Problem.

Mattsetzen üben

Dateien `SB_matt.xhtml`, `SR_matt.js`, `SB_mattsolo.xhtml`

Lernende sollen mattsetzen üben. Also bekommen sie Stellungen zum Üben. Das macht die Datei `SR_matt.js`. Diese Datei definiert die Funktion `schach_neues_brett` auf der „Fenster”-Ebene. Diese Funktion wird aufgerufen, wenn ein Schachbrett geladen ist. Der dritte Parameter ist die Funktion, mit der die Umgebung das Brett „anspricht” (`envh` in der Datei `SR_schach.js`), der Rückgabewert ist die Funktion, mit der das Brett die Umgebung „anspricht” (`handler`).

Wenn der Spieler den Knopf „Anfangsstellung” wählt, wird `handler` aufgerufen. Der zweite Parameter *k* hat den Wert "`set_initial_position`". `handler` setzt eine neue Anfangsstellung und neue Bedingungen, unter denen das Spiel automatisch remis endet (bei Wiederholung einer Stellung oder bei Überschreitung einer bestimmten Zahl von Zügen ohne unumkehrbare Änderung). Dazu ruft `handler` die „Rückruf-Funktion” mit dem Schlüssel "`configure`" auf und übergibt die Einstellungen als zweiten Parameter. Der Spieler braucht nichts weiter einzugeben. Deshalb gibt `handler` `0` zurück, damit die Behandlung des Knopfes „Anfangsstellung” normal weiterläuft.

Bei einem „Analysebrett” ohne Verbindung zum Server funktioniert das gut (`SB_mattsolo.xhtml`). Allerdings kennt das Analysebrett keine Remis-Bedingungen, denn nur die Stellungen, die der Server sendet, werden analysiert auf Matt, Patt oder andere „Endsituationen”.

Nun wollen wir mit einer Beschränkung der Zeit und der Zugzahl üben (`SB_matt.xhtml`). Da das ganz allein nicht so einfach funktioniert, üben wir mit einer Partnerin, die den schwarzen König zieht. Natürlich können wir auch ein zweites Browserfenster mit einem Schachbrett (zum Beispiel „Dauerschach” ohne Zeitbeschränkung) öffnen. Es funktioniert nicht. Denn „Anfangsstellung” sendet eine Stellung an den Server und setzt sozusagen die Uhr des Gegners in Gang, falls der mit Zeitbeschränkung spielt. Der Spieler selbst kann erst dann ziehen, wenn er eine „Antwort” bekommen hat. Die Antwort kann nur ein weißer Zug sein, weil in der vorgesetzten Stellung Weiß am Zug ist. Aber dann muss der schwarze König ziehen. Wir wollen doch mit den weißen Steinen mattsetzen!

Die Umgebung kann das Brett „freigeben”. Der Schlüssel ist "`enable`". Dem Spieler wird erlaubt, mit Weiß zu ziehen, aber erst nachdem die normale Aktion „Anfangsstellung” abgeschlossen ist. Die sendet nämlich die Stellung an den Server und legt das Brett erst einmal lahm. Deshalb wird `cb` verzögert aufgerufen.

Was wünscht man sich jetzt? Zum einen eine Antwort-Automatik, die in bestimmten Situationen (zum Beispiel bei einem einsamen König) oder auch immmer für die Partnerin automatisch zieht. Das ist im sozusagen ein Schachprogramm als Partnerin oder eine Partnerin mit Schachprogramm-Unterstützung.

Der zweite Wunsch ist eine Kurzschluss-Antwort, die bestimmte Situationen gar nicht erst an den Server sendet, sondern sofort antwortet. Das nächste Beispiel erfüllt diesen Wunsch.

Gegenzug-Automat

Dateien `SB_qr.xhtml`, `SR_qr.js`.

Dieses Beispiel sendet Spielsituationen nicht an den Server, sondern liefert gleich einen Antwortzug durch ECMAscript. Die Buchstaben `qr` stehen für „Quick Response”.

Wenn Sie `SB_qr.xhtml` laden, ist das Brett zunächst leer. Das Wort `noinitld` im Attribut `data-use` verhindert, dass die aktuelle Stellung anfangs vom Server geladen wird.

Wenn Sie „Anfangsstellung” wählen, wird die normale Anfangsstellung aufgebaut und „gesendet”. Bevor eine neue Spielsituation an den Server gesendet wird, wird die Umgebungs-Rückruffunktion (`envc`) mit dem Schlüssel `send_position` aufgerufen. Der Rückgabewert ist `1`, deshalb wird die Spielsituation nicht an den Server gesendet.

Schauen wir in die Datei `SR_qr.js`. Sie definiert die Funktion `schach_find_move`, die in einer Spielsituation einen Gegenzug findet und die neue Spielsituation liefert.

Hier interessiert die Funktion `schach_neues_brett`. Ihr Rückgabewert ist `handler`. Das Brett ruft diese Funktion in bestimmten Situationen auf. Eine solche Situation ist unmittelbar, bevor eine neue Spielsituation (normalerweise) an den Server gesendet wird. Der Schlüssel *k* ist in diesem Fall `send_position`. `handler` ruft zeitverzögert `response` auf und ergibt `1`. Deshalb wird die aktuelle Spielsituation nicht an den Server gesendet.

`response` lässt `schach_find_move` einen Antwortzug finden und sendet die dadurch entstehende neue Stellung an das Brett (Schlüssel `response`).

`schach_find_move` wählt aus allen möglichen Zügen irgendeinen Zug pseudozufällig aus. Natürlich ist diese Funktion kein starkes Schachprogramm. Aber hier geht es um die Funktionen des Bretts. In diesem einfachen Beispiel können Sie nur die schwarzen Steine setzen.

Mattsetzen wie der Blitz

Dateien `SB_mattqr.xhtml`, `SR_mattqr.js`.

Ein vorhergehendes Beispiel zeigt, wie automatisch eine Stellung vorgegeben wird, um das Mattsetzen zu üben. Mit Zeit- und Zugzahl-Beschränkung geht das leider nur mit einem Übungspartner. Dieses Beispiel kombiniert die automatische Vorgabe einer Stellung mit dem „Gegenzug-Automaten” des letzten Beispiels. So kann man auch allein probieren, ob man es schafft, im Blitztempo in 40 Zügen mit Springer und Läufer mattzusetzen.

Das frühere Beispiel startete damit, dass die Anfangsstellung an den Server gesendet wird. Jetzt stellen wir die Anfangsstellung direkt ein. Der Server ist am Spiel gar nicht mehr beteiligt.

Schauen wir zunächst auf die Kommunikation zwischen Umgebung und Brett in der Datei `SR_mattqr.js`. Diese Datei definiert zwei Funktionen `schach_find_move` und `schach_neues_brett`. Die erste Funktion

`schach_find_move` ist nahezu identisch mit dem früheren „Gegenzug-Automaten” (`SR_qr.js`). Das Ergebnis (`handler`) der Funktion `schach_neues_brett` kombiniert die beiden früheren Beispiele (`SR_matt.js` und `SR_qr.js`). Die Brett-Meldung „`send_position`” wird genauso behandelt wie zuvor.

Die Brett-Meldung „`set_initial_position`” wird anders behandelt ohne Beteiligung des Servers. Zunächst wird pseudo-zufällig eine der Steinkombinationen Springer und Läufer, zwei Läufer, Turm, Dame, Turm und Springer sowie Turm und Läufer jeweils natürlich mit König gewählt und die verfügbare Zeit sowie die Zugzahl bis zum automatischen Remis gesetzt (Schlüssel `configure`), Zeit und Zähler werden auf die Anfangswerte gesetzt (Schlüssel `reset`), und dann wird die Stellung an das Brett gesendet (Schlüssel `response`). Das ist alles, deshalb ist der Rückgabewert `1`.

Der König flieht

Die Funktion `schach_find_move` ist hier gegenüber der früheren Version in `SB_qr.xhtml` erweitert. Falls man (d.h. die Seite, die am Zuge ist) nur noch den König hat, bleibt nur die Frage: Welches Feld ist für den König das beste? Die Felder am Rand bekommen einen Punkt, deren Nachbarfelder zwei Punkte, deren Nachbarfelder drei, die Zentrumsfelder schließlich vier Punkte. Wenn die Gegenseite außer dem König nur noch einen Springer und einen Läufer hat, werden die Punkte anders vergeben. Die Eckfelder der Farbe des Läufers sind die schlechtesten Fluchtfelder, sie bekommen einen Punkt. Die angrenzenden Randfelder bekommen immer einen Punkt mehr, das Eckfeld von der „Gegenfarbe” des Läufers bekommt so acht Punkte. Die angrenzenden Felder zum Zentrum hin bekommen immer zwei Punkte mehr. Die Zentrumsfelder von der Farbe des Läufers bekommen so 10 Punkte, die Zentrumsfelder von der „Gegenfarbe” bekommen die Höchstpunktzahl 11. Soweit die „Vorabbewertung”.

Weiter wird unterschieden, ob das Feld durch einen gegnerischen Stein bedroht ist, ob das Feld frei ist oder ob ein eigener oder gegnerischer Stein auf dem Feld steht. Wenn das Feld bedroht ist, bekommt es 0 Punkte, wenn es frei ist, wird die Punktzahl der Vorabbewertung mit 4 multipliziert, wenn ein generischer Stein auf dem Feld steht (er ist nicht gedeckt!), wird die Punktzahl der Vorabbewertung mit 8 multipliziert. Allgemeiner hat man für jeden der sechs Fälle (bedroht / nicht bedroht, frei / eigener Stein / gegnerischer Stein) einen Faktor, mit dem die Vorab-Punktzahl multipliziert wird, und einen „Bonus”, der addiert wird, insgesamt 12 Zahlen, an denen man „drehen” kann. Hier ist der „Bonus” immer Null, der Faktor ist Null für ein bedrohtes Feld, `4` für ein freies Feld, `8` für ein Feld, auf dem eine ungedeckter gegnerischer Stein steht, und sonst `1`.

Nun ist ein Fluchtfeld so gut wie die angrenzenden Fluchtfelder. Zur Verbesserung der Bewertung werden in einem nächsten Schritt die vorhergehenden Bewertungen der Nachbarfelder addiert, jedenfalls im Prinzip. Die Nachbarfelder werden nicht alle gleich gewichtet. Es werden die genannten sechs Fälle unterschieden, und für jeden Fall gibt es einen Faktor und einen „Bonus”, also wieder 12 Zahlen, die die Bewertung variieren. Ich habe die folgenden Zahlen gewählt: Für bedrohte Felder ist der Faktor Null, für freie Felder `3`, für Felder mit einem gegnerischen Stein `5`, für Felder mit einem eigenen Stein `1`. Der Bonus ist für unbedrohte Felder Null, für freie Felder oder Felder mit einem gegnerischen Stein `1`, sonst Null.

Diese „Verbesserung” kann wiederholt werden. Aber Wiederholungen führen nicht zu einem besseren praktischen Ergebnis.

Mattsetzen mit Kiebitz

Dateien `SB_mattk.xhtml` und `SR_mattk.js`

Schafft es Herbert, in drei Minuten und maximal 40 Zügen ohne Stellungswiederholung mit Springer und Läufer mattzusetzen? Das vorangehende Beispiel soll so ausgebaut werden, dass die „Lästerer” das Spiel am Kiebitzbrett verfolgen können. Das Kiebitzbrett fragt den Server nur alle drei Sekunden nach einer neuen Spielsituation, deshalb überspringt das Kiebitzbrett manchen schnellen Zug.

In jedem Fall überspringt das Kiebitzbrett die Stellung nach einem Zug von Weiß, wenn dieser nicht die Übung beendet, denn der automatische Königszug erfolgt nahezu sofort. Die Stellung vor einem Königszug wird in diesem Beispiel gar nicht erst an den Server gesendet.

Die Datei enthält im `style`-Element im Kopf CSS-Regeln, die die Knöpfe „# Analyse”, „Neu aufstellen” und „Aufgeben” gar nicht erst anzeigen (`display: none;`). Diese Knöpfe haben die `id`-Attributwerte `b1_btn2`, `b1_btn3` und `b1_btn9`.

Zunächst wird dem Server die Anfangsstellung gesendet. Auf die Nachricht `set_initial_position` hin wird die Stellung nicht nur mit dem Schlüssel `response` an das Brett gesendet, sondern auch mit dem Schlüssel `push` an den Server.

Nun macht Weiß einen oder mehrere Züge, die die Übung nicht beenden, also weder matt setzen noch patt setzen. Der Kiebitz hat kaum eine Chance, die Stellung nach einem Zug von Weiß zu sehen, weil der „Zug-Automat" sofort antwortet. Deshalb wird die Stellung nach dem Zug von Weiß gar nicht erst an den Server gesendet, sondern die Stellung nach dem Antwortzug. Die Funktion `response` (Datei `SR_mattk.js`) sucht den Antwortzug und sendet die sich ergebende Spielsituation `r` an das Brett (`response`). Die Antwort beendet nicht die Übung, sie enthält nicht den Buchstaben `x`. `response` sendet die Antwort-Situation auch an den Server (`push`).

Wenn der Spieler die Zeit überschreitet, eine Stellung zu oft wiederholt, die maximale Zugzahl überschreitet oder nicht mehr genug Material hat, um mattzusetzen, sendet das Brett die Meldung `game_over`. Der dritte Parameter (hier *v*) ist die End-Spielsituation. Die Variable `game_over` hat noch den Wert 0. Die End-Spielsituation wird an den Server gesendet und die Übung ist beendet (Rückgabewert `1`).

Angenommen, der weiße Zug setzt matt oder patt. Die Funktion `schach_find_move` erkennt das und antwortet mit der entsprechenden End-Spielsituation, die den Buchstaben `x` enthält. Das Kiebitzbrett sendet zwar keine Spielsituation an der Server, analysiert aber eine empfangene Spielsituation wie andere Bretter. Es erkennt ein Matt oder Patt. Es ist nicht nötig, eine Matt- oder Patt-Meldung für das Kiebitzbrett zu senden. Statt dessen sendet die Funktion `response` dem Server die Schlussstellung, die das Brett gesendet hat. Dem Brett wird natürlich die End-Spielsituation gesendet, das dann mit der Meldung `game_over` antwortet. Wenn jetzt als Reaktion auf diese Meldung die End-Spielsituation an den Server gesendet wird, ersetzt sie auf der Seite des Servers die Schlussstellung. Der Kiebitz hat dann keine Möglichkeit, die Schlussstellung zu sehen. Hier kommt die Variable `game_over` zur Wirkung. Wenn `schach_find_move` mit einer End-Spielsituation antwortet, wird `game_over = 1` gesetzt. Die Reaktion auf die Meldung `game_over` des Bretts sendet dem Server nichts mehr, sondern gibt direkt `1` zurück. So kann der Kiebitz die Schlussstellung sehen. Natürlich wird bei einer neuen Übung (`set_initial_position`) wieder `game_over = 0` gesetzt.

Selbsttest

Dateien `SB_mattst.xhtml`, `SR_mattst.js`

In diesem Beispiel ist das Brett anfangs verborgen. Das liegt am Wort `hide` im Attribut `class` des Brett-Elements (`div`-Element mit der ID `b1`) zusammen mit der CSS-Regel `.hide { display: none; }`.

Interessanter ist, dass hier auch der Knopf „Anfangsstellung" (ID `b1_btn4`) verborgen ist. Seine Funktion übernimmt hier der Knopf „Neuen Test starten" (ID `btn_start`). Die Funktion `schach_neues_brett` in der Datei `SR_mattst.js` ordnet ihm die Funktion `on_start` zu. Diese setzt zunächst den Zugzähler `n_mv = 0`. Wir wollen nämlich zählen, wie viele Züge schließlich zum Matt führen (oder nicht). Dann wird der Variablen `rnd` zufällig eine Zahl von 0 bis 10 zugewiesen. Diese Zahl bestimmt, welche Steine zum Mattsetzen aufgebaut werden. Von ihr hängt auch die verfügbare Zeit und die Höchstzahl der Züge ab (`configure`). Alle Zähler werden auf die Anfangswerte zu Beginn eines Spiels gesetzt (`reset`). Das Brett wird geleert, indem mit dem Schlüssel `response` eine „leere" Stellung an das Brett gesendet wird. Das Brett ist jetzt im Aufbau-Modus. Der Spieler könnte jetzt eine Auswahlstein anklicken und aufs Brett stellen. Damit das verhindert wird, wird das Brett in den „Game-Over-Modus" gesetzt (Schlüssel `enable`, Parameter `s`). Das Brett wird angezeigt, die Erfolgsstatistik verborgen, und eine Sekunde später (`setTimeout`) der Test gestartet (`do_start`).

`do_start` sendet eine zufällige Stellung mit den Steinen gemäß der Variablen `rnd` an das Brett (`response`).

Wenn der Spieler zieht, sendet das Brett eine `send_position`-Nachricht. Der Zugzähler `n_mv` wird erhöht, die Funktion `response` sendet eine Antwort an das Brett. Der Rückgabewert auf die `send_position`-Nachricht ist 1. Deshalb wird die Spielsituation nicht an den Server gesendet.

`game_over`

Wenn das Spiel beendet ist, sendet das Brett eine `game_over`-Nachricht mit der End-Spielsituation als Parameter. Die Funktion `game_over` wertet den Test aus. Der Rückgabewert auf die `game_over`-Nachricht ist 1. Das

verhindert den weiteren normalen Ablauf auf der Seite des Bretts Andernfalls würde das Brett darauf eine `send_position`-Nachricht mit der End-Spielsituation senden, die Funktion `response` würde diese End-Spielsituation „zurückschicken" und das Brett noch einmal eine `game_over`-Nachricht senden.

Die Funktion `game_over` erstellt zu dem Test eine Liste mit vier Einträgen: der Stein-Kombination (ein Zahlencode von 0 bis 5), dem Ergebnis (0 Matt, 1 Patt, 2 Remis durch Materialverlust, 3 Zeitüberschreitung, 4 Wiederholung, 5 Überschreitung der Zugzahl), der Zahl der Züge und der verbrauchten Zeit in Millisekunden.

Um die verbrauchte Zeit zu bestimmen, fragt die Funktion das Brett nach den Zeit-Einstellungen (Schlüssel `query`, `time_difference`, `total_bonus`).

Diese Liste mit den vier Einträgen wird einer Liste der Ergebnisse aller „Selbsttests" hinzugefügt.

Zusätzlich wird eine zweidimensionales Feld `stats [art][i]` aktualisiert. `art` bezeichnet die Steinkombination.

`stats [art] [0]`

Die Gesamtzahl der Tests mit der Steinkombination

`stats [art] [1]`

Die Zahl der Tests mit der Steinkombination
, die mit Matt endeten (erfolgreiche Tests).

`stats [art] [2]`

Die Gesamtzahl der Züge der erfolgreichen Tests.

`stats [art] [3]`

Die in den erfolgreichen Tests insgesamt benötigte Bedenkzeit.

Die Funktion `game_over` zeigt abhängig vom Ausgang des Tests einen Text an. Bei einem Test, der mit Matt geendet hat, wird die Steinkombination, die Zahl der Züge und die benötigte Bedenkzeit angezeigt.

Der Knopf „Ihre Erfolgsstatistik" ist mit der Funktion `show_stats` verbunden. Sie bricht ein möglicherweise laufendes Testspiel ab, zeigt zusammengefasste Daten auf der Grundlage des Feldes `stats` an (`div`-Element mit der `id` `statistics`) und verbirgt das Brett.

Die Flucht des Königs ist hier weiter „verfeinert". Aus den möglichen Feldern, auf die der König ziehen kann, wird nur eines der beiden aussichtsreichsten Felder laut der Feldbewertung ausgewählt. Die möglichen Züge werden nach absteigender Bewertung sortiert (`qsort_ind`, „Quicksort" der Indizes). Die Funktion `choose_mv` nimmt die Zahl der Züge, die in die Zufallsauswahl kommen können, als Parameter.

Zum Test der Sortierung dient die Datei `qsort_test.js`.

Selbsttest-Ergebnisse speichern

Dateien `SB_mattsv.xhtml`, `SR_mattsv.js`, `hbs_mattsv.xslt`. Die Buchstabenkombination `sv` erinnert an das englische Wort „save".

Es ist ursprünglich nicht vorgesehen, dass ein Browser mittels Javascript Dateien im lokalen Dateisystem speichert. Die „Blob-URL"-Methode ist (noch) nicht standardisiert. Deshalb verwende ich hier „data:"-URL.

Die Ergebnisse der Selbsttests sollen im lokalen Dateisystem gespeichert werden. Der Knopf „Ergebnis speichern" (ID `btn_save`) ist mit der Funktion `save_results` verbunden. Diese erstellt eine „data:"-URL zum Download, und setzt die Attribute `href` und `download` im Download-Link (verborgenes `a`-Element mit der ID `a_save`) Der Wert des Attributs `download` ist ein vorgeschlagener Dateiname, unter dem die Ergebnisse gespeichert werden sollen. Der vorgeschlagene Dateiname beginnt mit `hbs_test_`, gefolgt von einem Zeitstempel und dem Suffix `.xml`. Das

Präfix `hbs_` steht für „HerBärs Schachecke”. Die Aktivierung (`click`) des Download-Links speichert die Daten und deaktiviert den Knopf „Ergebnis speichern”.

Der Knopf „Ergebnis speichern” wird zusammen mit dem Knopf „Ihre Erfolgsstatistik” in der Funktion `game_over` aktiviert, sobald ein einzelner Test endet.

Es gibt viele geeignete Dateiformate, die Ergebnisse der Einzeltests zu speichern. Ich habe ein einfaches XML-Format gewählt, das die Datei `rs.rng` beschreibt. Die Funktion `results_xml`, aufgerufen von `save_results`, stellt die Daten aus `results` zu einem XML-Dokument zusammen.

Der Browser erlaubt normalerweise nicht, dass eine lokale XML-Datei zur Darstellung auf ein Stylesheet im Internet verweist. Deshalb ist zum Beispiel die Datei `hbs_mattsv.xslt` herunterzuladen und im selben Verzeichnis wie die Ergebnis-XML-Datei zu speichern, und schon kann die Ergebnis-XML-Datei im Browser angezeigt werden. `hbs_mattsv.xslt` beschreibt auch die verwendeten XML-Elemente.

Der Server bekommt von den Tests und den Ergebnissen nichts mit, alle Ergebnisse bleiben lokal auf dem Rechner des Spielers.

Matt ohne König

Dateien `SB_moksv.xhtml` und `SR_moksv.js`. Die Buchstabenfolge `mok` steht für „Matt Ohne König”.

Ein Lernender (sind wir das nicht alle?) kann üben, ohne den eigenen König mattzusetzen. So prägen sich Mattbilder und manche Pattbilder ein. Dieses Beispiel entspricht fast völlig dem vorhergenden Beispiel, nur dass in den Stellungen der weiße König im Urlaub ist. Die Konfigurations-Option `-king` verhindert, dass ein Fehler gemeldet wird, wenn ein König fehlt.

Schnellschach-Partie speichern

Dateien `SB_schnellsv.xhtml` und `SR_schnellsv.js`. Die Buchstaben `sv` erinnnern an das englische Wort „save”.

Der Verlauf einer Partie soll aufgezeichnet und am Ende einer Partie im Dateisystem des Rechners des Spielers gespeichert werden.

Die Datei `SB_schnellsv.xhtml` definiert den Knopf „Spiel speichern” und ein unsichtbares `a`-Element mit der ID `a_save` („Download-Link”).

Der Knopf „Spiel speichern” ist mit der Funktion `on_save` verbunden. Wie im vorhergehenden Beispiel `SB_mattsv.xhtml` setzt sie die Attribute `href` und `download` des Download-Links und aktiviert ihn (`click`). Der Knopf „Spiel speichern” wird wieder deaktiviert. Er wird auch zu Anfang im Skript deaktiviert, weil mancher Browser beim Neuladen einer Seite den Aktivierungszustand von Eingabeelementen sonst beibehält.

Die Variable `started` in der Funktion `schach_neues_brett` zeigt an, ob überhaupt ein Spiel begonnen hat. Die Variable `game` enthält den bisherigen Spielverlauf.

Ein Spiels beginnt, wenn das Brett `reset` meldet. Die Variable `started` wird = 1 gesetzt und `game` initialisiert. Die erste Zeile gibt den URL-Pfad an (`window.location.path`), die zweite Zeile die Zeitbeschränkung, falls wie hier mit Zeitbeschränkung gespielt wird. Allerdings kann ein Spieler auch direkt ziehen, wenn erstmalig eine Stellung geladen ist. Deshalb ruft `schach_neues_brett` `handler` mit dem Schlüssel `reset` auf und benachrichtigt sich sozusagen selbst über den Beginn eines Spiels.

Die Nachricht `reset` bedeutet zunächst nur, dass der Spieler eine neue Partie beginnen möchte. Wenn der Server mit einer Stellung antwortet (Schlüssel `accept_position`), in der eine Seite am Zug ist, wird der Beginn der Partie bestätigt. In diesem Fall wird `started = 2` gesetzt. Wenn der Server mit einer anderen Spielsituation antwortet, wird wieder `started = 0` gesetzt. Eine solche „andere” Spielsituation kann eine Stellung im „Aufbau” sein, bei der nicht festgelegt ist, welche Seite am Zug ist, oder die End-Spielsituation der vorhergehenden Partie. Ob und welche Seite am Zug ist, ergibt die `query`-Nachricht mit dem Schlüssel `mode`.

Wenn das Brett eine neue Spielsituation sendet (Schlüssel `send_position`) oder empfängt (Schlüssel `accept_position`), wird die neue Spielsituation an den Spielverlauf (`game`) angehängt. Wenn es nach einem eigenen Zug eine Spielsituation sendet, wird zusätzlich auch die verbleibende Bedenkzeit angehängt.

Bei Ende des Spiels meldet das Brett `game_over`. Die `game_over`-Nachricht folgt nach der `accept_position`-Nachricht, aber geht der `send_position`-Nachricht voraus. Es gibt Gründe für die eine oder die andere Reihenfolge der Nachrichten. Ich bleibe bei der beschlossenen Reihenfolge. Falls `started` den Wert `2` hat, fügt die `game_over`-Nachricht die End-Spielsituation und die verbleibende Bedenkzeit an den Spielverlauf an, setzt `started = 0` und aktiviert den Knopf „Spiel speichern”. So wird in jedem Fall der vollständige Spielverlauf gespeichert.

Im Spielverlauf wird auch gespeichert, wenn der Spieler den Knopf „zur Analyse” wählt, um eine interessante Stellung zu kennzeichnen (Schlüssel `store_position`).

Erfolgsstatistik

Dateien `SB_schnellst.xhtml`, `SR_schnellst.js`, `hbs_games_ht.xslt`. Die Buchstaben `st` erinnern an „STatistik”.

Es ist schön, wenn man am Ende eines Spiel-Nachmittags sehen kann, wie viele Partien man gewonnen oder verloren und überhaupt gespielt hat. Wir ergänzen das letzte Beispiel (`SB_schnellsv.xhtml`) um eine solche Statistik.

Die Datei `SB_schnellst.xhtml` blendet durch CSS-Regeln zwei Knöpfe aus:

- Neu aufstellen (ID `br_btn3`)
- Anfangsstellung (ID `br_btn4`)

Den Knopf „Neu aufstellen” braucht man für das gewöhnliche Spiel nicht. Aber für den Knopf „Anfangsstellung” ist ein Ersatz nötig, mehr dazu später.

Zunächst ein Überblick über die Datei `SB_schnellst.xhtml`. Nach der Überschrift und einer Knopfreihe („Neues Spiel”, „Spiel speichern”, „Erfolgsstatistik”) folgen das Brett und der Statistik-Abschnitt (ID `d_stat`). Das Brett und der Statistik-Abschnitt sind anfangs verborgen, später wird immer nur einer der beiden Abschnitte angezeigt.

Der Knopf „Neues Spiel” (ID `btn_game`) ist mit der Funktion `on_game` (`SR_schnellst.js`) verbunden. Diese Funktion verbirgt den Statistik-Abschnitt (`d_stat`), zeigt das Brett (`e_br`) an und sendet dem Brett dann die Nachricht `set_initial_position`. Das ist also der Ersatz für den Knopf „Anfangsstellung”.

Die wesentlichen Erweiterungen der Funktion `schach_neues_brett` gegenüber dem letzten Beispiel sind:

- `col`: die Farbe, die der Spieler in der laufenden Partie führt
- `mvs`: die Zahl der Züge in der laufenden Partie
- `results`: eine Liste der Ergebnisse der bisherigen Partien

Eine Partie läuft erst dann ganz sicher, wenn der Server nach einem `reset` eine Stellung sendet (`accept_position`), in der eine Seite (Schwarz oder Weiß) am Zug ist. Der Spieler führt dann in der laufenden Partie diese Seite. Bei der Behandlung der Nachricht `accept_position` wird `col` mit dem Wert `w` (Weiß) oder `b` (Schwarz) belegt. Bei der Nachricht `reset` wird `col` wieder mit dem Anfangswert `u` (unbestimmt) belegt.

Die Zahl der Züge (`mvs`) wird bei der Nachricht `reset` auf den Anfangswert `0` gesetzt und bei der Nachricht `send_position` um `1` erhöht. Bei Ende der Partie geht die Nachricht `game_over` der Nachricht `send_position` voran, so dass die Information über das Ende der Partie nicht als Zug gezählt wird.

`results` ist die Liste der Ergebnisse der beendeten Partien. Jeder Eintrag in dieser Liste ist eine Liste mit vier Elementen:

der Farbe, die der Spieler führte (`col`)
der Seite, die die Partie beendete, also nicht den letzten Zug machte,
der Art oder dem Grund des Partie-Endes

und der Zahl der Züge (mvs).
Wenn eine laufende Partie (started = 2) endet, sendet das Brett game_over. Der Liste results wird dann ein neuer Eintrag hinzugefügt. Die zweite und die dritte Komponente des Eintrags ergeben sich aus der End-Spielsituation (Parameter *val*). Die Knöpfe „Erfolgsstatistik" und „Erfolgsstatistik speichern" (b_stsv) werden aktiviert. Der Knopf „Erfolgsstatistik speichern" ist im Statistik- Abschnitt enthalten. Der Knopf „Spiel speichern" wird natürlich auch aktiviert, genauso wie im vorhergehenden Beispiel.

Der Knöpfe „Erfolgsstatistik" und „Erfolgsstatistik speichern" funktionieren genauso wie die entsprechenden Knöpfe im Beispiel „Selbsttest-Ergebnisse speichern" („Ihre Erfolgsstatistik" und „Ergebnis speichern"). Nur ist der Knopf „Erfolgsstatistik speichern" hier nicht in der „Knopfzeile", sondern im Statistik-Abschnitt. Die Datei games.rng beschreibt das XML-Format der Erfolgsstatistik, die Datei hbs_games_ht.xslt erlaubt es, die Erfolgsstatistik im Browser anzusehen. Der vorgeschlagene Dateiname hat die Form hbs_games_*TIMEMS*.xml. *TIMEMS* steht für die aktuelle Systemzeit in Millisekunden. (s. Dateiformate)

Zug-Einstellungen

Dateien SB_schnellcf.xhtml, SR_schnellcf.js und SB_suw.xhtml

Dieses Beispiel zeigt dem Spieler mögliche Einstellungen zum Ziehen und erlaubt, verschiedene Einstellungen auszuprobieren.

Der Knopf „Zug-Einstellungen" unter dem Brett (ID btn_mv) zeigt den Abschnitt „Zug-Einstellungen" an. Die Markierungskästchen (input-Elemente) entsprechen den Konfigurations-Optionenpaaren -strict / strict, -touch / touch und -release / release. Der Name (name) der Markierungskästchen ist die Buchstabenfolge mv gefolgt von dem Namen der „positiven" Option (ohne Minuszeichen). Ein markiertes Kästchen entspricht der „positiven" Option, ein nicht markiertes Kästchen entspricht der „negativen" Option mit vorangestelltem Minuszeichen.

Dieses Beispiel ignoriert nach der Initialisierung Meldungen des Bretts vollständig. Die Funktion schach_neues_brett gibt eine „leere" Funktion zurück, die nichts tut. Aber während das Feld „Zug-Einstellungen" angezeigt wird, soll der Spieler keine unangenehmen Überraschungen erfahren wie einen Partieverlust durch Zeitüberschreitung. Deshalb können die Anfragen des Bretts an den Server vorübergehend eingestellt werden (envh mit dem Schlüsselpaar enable / break). Mit dem Schlüsselpaar enable / continue) sendet das Brett wieder Anfragen an den Server.

Die Datei SB_schnellcf.xhtml bietet Schnellschach. Wie wirken die Zug-Einstellungen, wenn die Spielregeln nicht gelten oder andere Spielregeln gelten, die nicht geprüft werden? Wir spielen schnell mal zur Entspannung Schaf und Wolf (SB_suw.xhtml). Hier sind die Reihen der Auswahlfiguren (ID b1_sw und b1_sb) ebenso wie einige Knöpfe verborgen.

Beim Schaf-und-Wolf-Spiel gelten die Schach-Spielregeln nicht alle (Einstellung -val). Die Zug-Einstellung touch hat zur Folge, dass eine empfangene Stellung (nach einem Gegenzug) entsprechend den Schachregeln auf Matt oder Remis geprüft wird. Wo kein König, da kein Matt - es ist nur zu verhindern, dass das Spiel abbricht, weil die Könige fehlen (-king). Solange es einen Bauern gibt, ist es kein „technisches Remis", und solange nicht alle Schafe oder der Wolf auf der Zielreihe stehen, ist ein Bauern-Schachzug nach vorn möglich, also kein Patt. Daher können beim Schaf-und-Wolf-Spiel alle Zug-Einstellungen genutzt werden. Probieren Sie es aus! Vorsicht: die Regellosigkeit (oder besser die Nichtprüfung oder Unkenntnis der Schaf-und-Wolf-Spielregeln) macht es möglich, dass ein Schaf ein anderes Schaf „vertreibt" (schlägt).

Kinder können sich von Anfang an an die Regel „gerührt - geführt" gewöhnen. Daher bevorzuge ich bei Schaf und Wolf die Zug-Einstellung strict.

Komplizierte Zeitbeschränkung

Dateien SB_cxtm.xhtml, SR_cxtm.js, SB_cxtmsv.xhtml und SR_cxtmv.js. Die Buchstabenfolge „cxtm" erinnert an „CompleXTiMe".

Die Schlüssel time_add, time_grant, time_limit und time_set erlauben komplizierte Beschränkungen der Bedenkzeit.

Ich wandle das Schnellschach ab: die ersten zehn Züge müssen im Blitz-Tempo erfolgen, und sobald erstmals eine Seite technisch nicht mehr mattsetzen kann, beträgt die verbleibende Bedenkzeit 2 Minuten zuzüglich 2 Sekunden Fischer-Bonus.

Zunächst zur ersten Phase der ersten zehn Züge im Blitztempo. Ich bevorzuge eine Blitz-Bedenkzeit von 220s bei 2s Fischer-Bonus. Für eine durchschnittliche Partie nehme ich 40 Züge an. Daher komme ich für die ersten zehn Züge auf 55s bei 2s Fischer-Bonus.

Nun zur zweiten Phase nach dem zehnten Zug. Die verfügbare Zeit für die ersten zehn Züge beträgt 75s. Der Spieler bekommt nach dem zehnten Zug eine Zeitgutschrift von 645s. Der Fischer-Bonus wird auf 6s erhöht. Der akkumulierte Bonus für die nächsten 30 Züge beträgt 30 x 6s = 180s. Die Bedenkzeit für die ersten vierzig Züge beträgt insgesamt 75s + 645s + 180s = 900s wie bei meinem bevorzugten Schnellschach.

Die Bedenkzeit von 120s bei 2s Fischer-Bonus für die mögliche Schlussphase reicht für mich normalerweise aus, mit Springer, Läufer und König mattzusetzen.

Nun zu den Dateien `SB_cxtm.xhtml` und `SR_cxtm.js`. In der Datei `SR_cxtm.js` interessieren die folgenden drei Variablen.

`started`

> Wie im Beispiel der Partie-Aufzeichnung wird der Beginn einer Partie erkannt, wenn der Server nach `reset` eine Zug-Stellung liefert (`accept_position`). Bei einer `reset`-Nachricht wird `started` vom Anfangswert `0` auf den Wert `1` gesetzt, bei einer `accept_position`-Nachricht von `1` bei einer Zugstellung auf `2`, sonst (Aufbau oder End-Spielsituation) wieder auf `0`. Auch bei „`game_over`" wird `started` auf `0` gesetzt.

`mvcnt`

> Der Zugzähler, an dem das Ende der Blitzphase erkannt wird. Bei einer `reset`-Nachricht wird `mvcnt = 0` gesetzt. Bei einer `send_position`-Nachricht wird `mvcnt` erhöht, sofern ein Spiel läuft (s. `started`). Nach dem zehnten Zug werden dem Spieler 645000 Millisekunden Bedenkzeit gutgeschrieben (`time_add`) und der Fischer-Bonus auf 6s gesetzt (`configure`), sofern nicht schon die Schlussphase (`ending`) erreicht ist.

`ending`

> Der Wert `1` zeigt an, dass das Spiel die Schlussphase erreicht hat. Bei einer `reset`-Nachricht wird `ending = 0` gesetzt. Beim Empfang einer Stellung (`accept_position`) werden Daten zum möglichen technischen Remis abgefragt. Der Schlüssel `query` mit dem zweiten Schlüssel `technical` liefert ein Bitfeld. Das 1-Bit zeigt an, ob der Spieler, der am Zug ist, theoretisch gewinnen kann, das 2-Bit zeigt an, ob sein Gegenspieler theoretisch gewinnen kann. Wenn nicht beide Bits gesetzt sind, wird die verbleibende Bedenkzeit auf 120000 Millisekunden gesetzt (`time_set`) und der Fischer-Bonus auf 2 Sekunden.

Die Dateien `SB_cxtmsv.xhtml` und `SR_cxtmsv.xhtml` kombinieren die „komplizierte" Zeitbeschränkung mit der Partieaufzeichnung aus einem früheren Beispiel.

Schwarz und Weiß

Dateien `SB_weiss.xhtml`, `SR_weiss.js`, `SB_schwarz.xhtml` und `SR_schwarz.js`,

Der Anfang einer Partie ist in den bisherigen Beispielen etwas holprig, weil am Anfang der eine Spieler nicht weiß, ob der andere überhaupt schon seinen Computer eingeschaltet hat, weil die Spieler erst aushandeln müssen, wer mit den weißen Steinen spielt und weil anfangs auch noch nicht klar ist, ob vielleicht schon andere Spieler an dem Brett spielen, obwohl das natürlich unwahrscheinlich ist. Wenn schon vorher festgelegt ist, wer mit den weißen Steinen und wer mit den schwarzen Steinen spielt, bleibt nur die Unsicherheit, wer als erster das Schachbrett geladen hat.

Der eine Spieler lädt `SB_weiss.xhtml`, der andere `SB_schwarz.xhtml`.

Schauen wir zuerst `SB_weiss.xhtml` an. Außer dem Knopf „Neu aufstellen" wird auch der Knopf „Anfangsstellung" nicht angezeigt. Diese Datei ist für eine einzelne Partie gedacht, bei der der Spieler die weißen Steine führt. Die zweite Besonderheit ist die Einstellung `noinitld`. Die anfangs im Server gespeicherte Spielsituation wird überhaupt nicht geladen.

Nun zur Datei `SR_weiss.js`. Zu Anfang wird dem Server die Anfangsstellung sozusagen „aufgezwungen” (`envh`-Schlüssel `setup` mit dem zweiten Schlüssel `initial`) und die Uhr gestellt (`reset`).

Die Variable `play` zeigt an, ob die Partie schon begonnen hat. Wenn sie begonnen hat (`play` ist nicht Null), läuft sie ohne weitere Eingriffe weiter. Am Anfang warten wir, dass der Server eine neue Spielsituation meldet (`envc`-Schlüssel `accept_position`). Wenn dann Weiß am Zug ist (`mode` = `2`), läuft die Partie (`play = 1`). Sonst ist etwas falsch gelaufen. Vielleicht hat sich ein Dritter eingemischt? Die „Umgebung” sendet den End-Spielcode `awxx`: Weiß bricht das Spiel ab.

Was geschieht, wenn beide Spieler `SB_weiss.xhtml` laden? Einer von beiden ist schneller. Der zweite liefert dem ersten eine Spielsituation, in der Weiß am Zug ist. Für den ersten Spieler beginnt die Partie. Er zieht und liefert dem zweiten eine Stellung, in der Schwarz am Zug ist. Der zweite bricht dann das Spiel ab.

Nun zur Datei `SB_schwarz.xhtml`. Im Vergleich zu `SB_weiss.xhtml` gibt es hier zusätzlich die Einstellung `view:b`. Sie bedeutet, dass die „schwarze” Brettseite unten angezeigt wird. Eine entsprechende Einstellung `view:w` in der Datei `SB_weiss.xhtml` ist nicht nötig, weil in der Voreinstellung die weiße Brettseite unten angezeigt wird.

Die Datei `SB_schwarz.js` startet ebenso wie `SB_weiss.js` damit, dass dem Server die Anfangsstellung „aufgeladen” wird. Dann gibt es zwei Möglichkeiten (von Fehlern abgesehen), wie es weitergehen kann. Im ersten Fall hat sich der Spielpartner mit den weißen Steinen schon angemeldet und antwortet mit seinem ersten Zug. Diese „schwarze” Brettseite empfängt eine Stellung, in der Schwarz am Zug ist (`mode` ist `3`). Das Spiel läuft normal weiter. Im zweiten Fall lädt der Spielpartner die `SB_weiss.xhtml` als zweiter. Diese schwarze Brettseite empfängt die Anfangsstellung, in der Weiß am Zug ist. Als Antwort gibt lädt sie dem Server wieder die Anfangsstellung auf. Jetzt kann Weiß ziehen, und das Spiel läuft.

Die Variable `play` kann hier drei Werte annehmen:

`0` bedeutet, dass noch keine Spielsituation vom Server empfangen worden ist.
`1` bedeutet, dass genau einmal eine Spielsituation vom Server empfangen worden ist und in dieser Spielsituation nicht Schwarz am Zug war.
`2` bedeutet, dass das Spiel läuft.

Im Falle `play = 0` kann in einer empfangenen Stellung Weiß oder Schwarz am Zug sein. Wenn Schwarz am Zug ist, läuft die Partie (`play = 2`). Wenn Weiß am Zug ist, wird weiter gewartet (`play = 1`). Im Falle `play = 1` muss in einer empfangenen Stellung Schwarz am Zug sein, sonst wird die Partie abgebrochen (End-Spielsituation `abxx`).

Was geschieht, wenn beide Spieler `SB_schwarz.xhtml` laden? Der zweite sendet dem ersten die Anfangsstellung, in der Weiß am Zug ist. Der erste sendet dem zweiten wieder die Anfangsstellung, der zweite sendet dem ersten zum zweiten Mal die Anfangsstellung, der erste bricht das noch nicht begonnene Spiel ab.

Als Entschädigung für den fehlenden Knopf „Anfangsstellung” gibt es einen Verweis „Neue Partie” sozusagen auf die andere Seite des Bretts. Der Verweis wird angezeigt, sobald eine Partie endet.

Gewinner bleibt sitzen

Dateien `SB_gbs.xhtml` und `SR_gb.js`

Dieses Beispiel zeigt, dass (und wie) die Schachecke zu einem Online-Turnier benutzt werden kann. Die Umgebung gibt vor, welche Spieler mit welcher Farbe gegeneinander spielen und erlaubt anderen, das Spiel zu beobachten. Am Ende einer Partie wird das Ergebnis der Umgebung gemeldet.

Ein Online-Turnier erfordert, dass die Teilnehmer sich anmelden. Da ich keinen juristischen Beistand zu meiner Website habe, ist eine Online-Anmeldung ausgeschlossen. Anders ist es, wenn ich Bekannten Zugangsdaten (Anmeldename und Kennwort) mitteilte. Das „Turnier” gibt es deshalb nur auf dem lokalen Rechner. Die Teilnehmer müssen sich an demselben Rechner und an demselben virtuellen „Ort” anmelden. Die Daten zum Turnierablauf bleiben im lokalen Speicher des Rechners. Der Austausch der Spielsituationen zwischen den Browerfenstern erfolgt über das Internet.

Das „Turnier” ist sehr einfach. Es läuft immer nur eine Partie. Der Gewinner kann die nächste Partie spielen, bis er eine Partie verliert. Für eine gewonnene Partie bekommt ein Teilnehmer drei „Wartepunkte”, für ein Remis zwei

„Wartepunkte” und für eine verlorene (aber immerhin gespielte) Partie einen „Wartepunkt”. Der „Sitzenbleiber” spielt gegen den Teilnehmer mit den wenigsten Wartepunkten. Bei gleichen Wartenpunkten spielt der Teilnehmen mit dem geringsten Erfolg. Ein Teilnehmer kann sich jederzeit anmelden und abmelden. Beim Anmelden bekommt er so viele Wartepunkte wie der angemeldete Teilnehmer mit den wenigsten Wartepunkten.

Die Funktion `Turnier` repräsentiert den „Turnier”-Ablauf. Zu jedem Browserfenster gibt es eine Instanz, aber alle teilen sich die Daten im lokalen Speicher des Browsers. Ich nutze hier den objektorientierten Stiel. Die Funktion dient als „Konstruktor” für das „Objekt” `turnier` mit den folgenden Methoden:

`login` (*`name`*)

Ein Teilnehmer meldet sich unter dem Namen *`name`* an.

`report_result` (*`name`*, *`res`*)

Der Teilnehmer *`name`* meldet das Partie-Ergebnis *`res`*: `0`: Schwarz gewinnt, `1`: Remis, `2`: Weiß gewinnt.

`logout` (*`name`*)

Der Teilnehmer *`name`* meldet sich ab.

`info` (*`tbody`*)

Liefert eine Übersicht über den aktuellen Stand (Teilnehmer, Punkte, Partien). *`tbody`* ist der Tabellen-Rumpf, der mit den Daten gefüllt wird.

info_lastgame ()

Spieler der letzten beendeten Partie und deren Ergebnis.

info_game ()

Spieler der anstehenden oder laufenden Partie und ggf die Bestätigungen der Spieler.

`results` (*`tbody`*)

Die Spieler der bisher beendeten Partien und deren Ergebnisse werden in den Tabellenrumpf *`tbody`* eingetragen.

`confirm` (*`name`*)

Der Spieler *`name`* meldet sich bereit zur anstehenden Partie.

`new_tournament` ()

Ein neues Turnier beginnt. Alle Daten zum Turnier im lokalen Speicher werden gelöscht.

In einer „echten” Umgebung würden derartige Methoden nicht unmittelbar antworten, sondern Daten über das Netz senden und „zurückrufen”, wenn die Antwort bereit steht. Aber auf die Einzelheiten des Turniers kommt es hier nicht an. In diesem Beispiel ist wesentlich, wie den Spielern ihre Rolle am Brett vorgegeben wird.

Die Funktion `brett` (Datei `SR_schach.js`) „modelliert” das Spielbrett mit den Steinen und der Uhr, die Funktion `Turnier` den Turnier-Ablauf. Die Funktion `schach_neues_brett` hat hier sozusagen die Rolle des Spielers: sie ergänzt das Brett um die Elemente, die den Spieler zum Turnier-Teilnehmer machen.

Der erste interessante Punkt in diesem Beispiel ist die Initialisierung. Das `script`-Element in der Datei `SB_gbs.xhtml`, das die Skript-Datei `SR_gbs.js` lädt, hat die ID `script_load`. Diese spezielle ID hat zur Folge, dass die Standard-Funktion `window.onload` nicht definiert wird. Statt dessen muss (oder sollte oder kann, wie auch immer) die Initialisierungsfunktion an anderer Stelle definiert werden, hier in der Datei `SR_gbs.js`. Zunächst wird `Turnier` als Konstruktor für `turnier` aufgerufen. Dann folgt die Initialisierung `schach_onload` wie bei den anderen Beispielen. Es gibt genau ein „Brett”-Element, daher wird `schach_neues_brett` genau einmal aufgerufen.

Das Brett-Element ist im Abschnitt mit der ID `d_brett` enthalten, der anfangs verborgen ist. Der Weg zur Anzeige des Bretts führt über den Abschnitt mit der ID `d_partie`. Die Funktion `on_partie` zeigt diesen Abschnitt an und aktualisiert ihn. Die Funktion `info_game` liefert die Namen der Spieler der anstehenden oder laufenden Partie und die Information, ob die Spieler sich bereit gemeldet haben (`confirm`). Nun kann es sein, dass die Paarung noch nicht feststeht, zum Beispiel weil sich ein vorgesehener Spieler abgemeldet hat oder überhaupt noch kein zweiter Spieler angemeldet ist. In diesem Fall wird `on_partie` zwei Sekunden später noch einmal aufgerufen.

Andernfalls stehen die vorgesehenen Spieler der anstehenden Partie fest. Wenn der angemeldete Spieler, der vor dem Browserfenster sitzt (`name`), nicht als Spieler der anstehenden Partie vorgesehen ist, kann er warten (Knopf „Warten", `on_wait`) oder die Partie beobachten (Knopf „Beobachten", `on_watch`). Die Funktion `on_wait` setzt `partie_waiting = 1` und ruft `on_partie` zur Aktualisierung auf. Falls `partie_waiting` = 1 ist, wird `on_partie` drei Sekunden später automatisch aufgerufen. Die Funktion `on_watch` setzt `playmode = 1` und zeigt das Brett an (`on_brett`).

Es bleibt der Fall, dass der angemeldete Spieler die anstehende Partie spielen soll. Er kann sich jetzt spielbereit melden (Knopf „Spielen", `confirm`). Wenn beide vorgesehenen Spieler sich spielbereit gemeldet haben, wird `playmode = 2` oder `playmode = 3` gesetzt. `2` bedeutet, dass der angemeldete Spieler die weißen Steine führt, `3` bedeutet, dass er die schwarzen Steine führt. Dann wird das Brett angezeigt (`on_brett`). Wenn sich der Gegner noch nicht spielbereit gemeldet hat, wird `on_partie` eine Sekunde später noch einmal aufgerufen.

Die Funktion `on_brett` zeigt das Brett an. Die Variable `playmode` hat beim Aufruf dieser Funktion einen der Werte `1` (beobachten), `2` (weiß) oder `3` (schwarz). Unter dem Brett (ID `brt`) gibt es eine Leiste (ID `p_gameover`) mit den drei Knöpfen „Partie", „Abmelden" und „Übersicht". Diese Leiste wird nur im Fall `playmode` = 1 angezeigt, sonst zunächst verborgen. Der Spieler hat so keine Möglichkeit, vom Brett zwischendurch zu einer anderen Ansicht zu wechseln, zum Beispiel den Tabellenstand anzusehen. (Natürlich kann er die Seite neu laden, aber wir halten uns an die Regeln.) Das wäre leicht zu machen, lenkt aber vom Wesentlichen ab.

Die Variable `play` „verfolgt" den Beginn einer Partie wie in den Beispielen Schwarz und Weiß. Hier wird `play = 0` gesetzt und so eine Partie begonnen, falls `playmode` nicht `1` ist. Das ist hier möglich, weil der Spieler das Brett während der Partie nicht „verlassen" kann.

Im Falle `playmode` = `1` (Beobachten) wird die Einstellung `"-active view:a"` an das Brett gesendet. `-active` verhindert, dass der Beeobachter ins Spiel eingreifen kann. `view:a` stellt ein, dass die Ansicht automatisch gedreht wird, solange der Beobachter nicht den Knopf „Brett drehen" drückt. `enable` mit dem zweiten Schlüssel `start` startet regelmäßige Anfragen an den Server nach der aktuellen Stellung, bis die Partie beendet ist oder bis der Beobachter die Ansicht wechselt. Die Funktionen `on_logout`, `on_partie` und `on_info` beenden die regelmäßigen Anfragen durch `enable` mit dem zweiten Schlüssel `s`.

Falls `playmode` `2` oder `3` ist, wird mit der Einstellung `active` das Brett aktiviert, so dass die Knöpfe „Remis", „Aufgeben", der Farbbalken und die Zeitanzeige wieder erscheinen und das Brett auf Klicks reagiert. Die Einstellung `view:b` beziehungsweise `view:w` zeigt das Brett mit der Farbe, die der Spieler führt, unten an. `reset` setzt alle Zähler auf die Werte zum Beginn einer Partie. `setup` mit dem zweiten Schlüssel `initial` stellt die Anfangsstellung auf und sendet sie an den Server.

Es ist nicht festgelegt, ob Schwarz oder Weiß zuerst das Brett sieht. Natürlich könnte die Umgebung (die „Turnierleitung") eine Reihenfolge vorgeben, aber der Aufwand zur Synchronisation wird eher größer. Vor dem Beginn einer Partie hat `play` den Wert `0`. Der Wert `2` bedeutet, dass die Partie läuft. In der ersten Spielsituation, die Weiß empfängt (`accept_position`), kann nur Weiß am Zug sein. Wenn nicht, dann hat vielleicht ein unvorsichtiger Besucher der Website zufällig den „Spielort" getroffen und etwas ausprobiert. In diesem Fall bricht Weiß das Spiel ab (Spielsituation `awxx`). Sonst wird `play = 2` gesetzt.

In der ersten Spielsituation, die Schwarz (`playmode` ist `3`) empfängt, kann dagegen Weiß am Zug sein, wenn nämlich Schwarz schneller ist und das Brett zuerst sieht. Schwarz antwortet mit der Anfangsstellung und setzt `play = 1`. Wenn in der nächsten Spielsituation, die Schwarz empfängt, wieder Weiß am Zug ist, bricht Schwarz das Spiel ab (Spielsituation `abxx`). Sonst wird `play = 2` gesetzt, und die Partie läuft.

Es bleibt nur noch nach der Partie (`game_over`) das Ergebnis zu melden (`report_result`). Der Einfachheit halber melden beide Seiten. Die „Turnierleitung" registriert die erste Ergebnismeldung und ignoriert die zweite. Nach der Partie werden die Knöpfe unter dem Brett angezeigt, damit es „legal" weitergehen kann.

Herberts Schachecke: `SR_matt.js`

Die Aufgabe

Die Funktion `schach_neues_brett` liefert eine Funktion, die das Brett bei bestimmten Ereignissen aufruft. Wenn der Spieler den Knopf „Anfangsstellung” wählt, werden neue Einstellungen, insbesondere eine Anfangsstellung, an das Brett gesendet und das Brett für einen weißen Zug freigegeben.

Variable und Funktionen

Die folgende Liste erklärt die Variablen und Funktionen innerhalb von `schach_neues_brett`.

`cb` (*`key`*, *`val`*)

> `cb` ist der dritte Parameter der Funktion `schach_neues_brett`. Diese Funktion sendet Einstellung an das Brett. Hier werden die Schlüssel (Werte für *`key`*) `configure` und `enable` benutzt.

`brett` [][]

> Die aufwendigste Aufgabe ist die pseudo-zufällige Positionierung des schwarzen Königs, nachdem die weißen Steine gesetzt sind. `brett` ist ein zweidimensionales Feld von Zahlen. Die Indizes von 0 bis 7 entsprechen den Linien und Reihen. Eine positive Zahl ist das Gewicht, das das Feld zur Positionierung des schwarzen Königs bekommt. Die Zahl `0` bedeutet, dass der schwarze König nicht auf das Feld gesetzt werden kann oder soll, zum Beispiel weil das Feld von einem weißen Stein bedroht ist. Die Zahl `-1` bedeutet, dass das Feld durch einen weißen Stein besetzt ist.

`brett_wght_center` ()

> Der schwarze König soll bevorzugt im Zentrum stehen. Diese Funktion setzt die anfänglichen Gewichte der Felder und gibt den Feldern im Zentrum ein größeres Gewicht.

`brett_add_around` (*`pl`*, *`pr`*)

> Ein aktiver schwarzer König greift einen weißen Stein an. Die Parameter *`pl`* und *`pr`* sind Linie und Reihe. Das bezeichnete Feld und die angrenzenden Felder bekommen ein größeres Gewicht. Auf dem bezeichneten Feld steht hier ein Läufer, Springer oder Turm.

`brett_clr_k` (*`pl`*, *`pr`*)

> Ein weißer Stein auf dem Feld mit den Koordinaten *`pl`* und *`pr`* bedroht normalerweise andere Felder, die der schwarze König nicht besetzen kann. Diese und die folgenden Funktionen `brett_clr_*` setzen das Gewicht der bedrohten Felder auf `0`.
>
> Diese Funktion nimmt an, dass auf dem bezeichneten Feld der weiße König steht („`k`”).

`brett_clr_b` (*`pl`*, *`pr`*)

> Setzt das Gewicht der angrenzenden Felder in diagonalen Richtungen bis zu einem Feld (ausschließlich), das durch einen weißen Stein besetzt ist, auf `0`. Der Buchstabe `b` erinnert an das englische Wort „Bishop” für Läufer.

`brett_clr_n` (*`pl`*, *`pr`*)

> Setzt das Gewicht der Felder, die ein Springer vom angegebenen Feld aus erreichen kann. Der Buchstabe `n` erinnert an den Buchstaben zur Kennzeichnung eines Springers in der englischen Notation.

`brett_clr_r` (*`pl`*, *`pr`*)

> Setzt das Gewicht der angrenzenden Felder in waagerechten und senkrechten Richtungen bis zu einem Feld (ausschließlich), das durch einen weißen Stein besetzt ist, auf `0`. Der Buchstabe `r` erinnert an das englische Wort „Rook” für Turm.

`brett_rnd88` ()

Diese Funktion liefert das pseudo-zufällige Feld gemäß den Gewichtungen in `brett` im „88-Format”.

`rndpos_nb` ()

Setzt den weißen König, Springer und Läufer gleichverteilt auf das Brett und dann den schwarzen König (`brett_rnd88`). Die Buchstaben `nb` erinnern an die Bezeichnungen für Springer und Läufer in englischen Notationen. Das Ergebnis ist die Stellung mit Weiß am Zug.

Diese Funktion nutzt die Hilfsfunktionen `brett_wght_center`, `brett_add_around`, `brett_clr_n`, `brett_clr_b` und `brett_rnd88`.

`rndpos_bb` ()

Setzt den weißen König und zwei verschieden-feldrige Läufer gleichverteilt auf das Brett und dann den schwarzen König (`brett_rnd88`). Die Buchstaben `bb` erinnern an zwei Läufer. Das Ergebnis ist die Stellung mit Weiß am Zug.

`rndpos_q` ()

Setzt den weißen König und die weiße Dame gleichverteilt auf das Brett und dann den schwarzen König (`brett_rnd88`). Der Buchstabe `q` erinnert an das englische Wort „Queen” für Dame. Das Ergebnis ist die Stellung mit Weiß am Zug.

Diese Funktion nutzt die Hilfsfunktionen `brett_wght_center`, `brett_clr_b`, `brett_clr_r` und `brett_rnd88`.

`rndpos_r` ()

Setzt den weißen König und einen weißen Turm gleichverteilt auf das Brett und dann den schwarzen König (`brett_rnd88`). Der Buchstabe `r` erinnert an das englische Wort „Rook” für Turm. Das Ergebnis ist die Stellung mit Weiß am Zug.

Diese Funktion nutzt die Hilfsfunktionen `brett_wght_center`, `brett_add_around`, `brett_clr_r` und `brett_rnd88`.

`rndpos_nr` ()

Setzt den weißen König, Springer und Turm gleichverteilt auf das Brett und dann den schwarzen König (`brett_rnd88`). Die Buchstaben `nr` erinnern an die Bezeichnungen für Springer und Turm in englischen Notationen. Das Ergebnis ist die Stellung mit Weiß am Zug.

Diese Funktion nutzt die Hilfsfunktionen `brett_wght_center`, `brett_add_around`, `brett_clr_n`, `brett_clr_r` und `brett_rnd88`.

`rndpos_br` ()

Setzt den weißen König, Läufer und Turm gleichverteilt auf das Brett und dann den schwarzen König (`brett_rnd88`). Die Buchstaben `br` erinnern an die Bezeichnungen für Springer und Turm in englischen Notationen. Das Ergebnis ist die Stellung mit Weiß am Zug.

Diese Funktion nutzt die Hilfsfunktionen `brett_wght_center`, `brett_add_around`, `brett_clr_b`, `brett_clr_r` und `brett_rnd88`.

`handler` (*i*, *k*, *v*)

Das Brett ruft diese Funktion bei bestimmten Ereignissen auf. Diese Funktion behandelt den Fall, dass *k* den Wert `set_initial_position` hat (Knopf „Anfangsstellung”).

Pseudo-zufällig liefert eine der Funktionen `rndpos_*` die nächste Stellung mit Weiß am Zug. Passend zur ausgewählten Funktion wird die Zahl der Züge bis zum automatischen Remis gewählt.

`cb` mit dem ersten Parameter "`configure`" sendet die Einstellungen an das Brett. Der zeitverzögerte Aufruf mit dem Schlüssel "`enable`" gibt das Brett für den Spieler für einen weißen Zug frei.

Das Ergebnis `0` bedeutet, dass die Behandlung des Knopfes „Anfangsstellung” normal weitergeht.

Ergebnis des Matt-Selbstests

Wurzelelement rs

rs — Ergebnisse der Einzeltests

Enthält: r (+)

Enthalten in: Wurzel

```
<element name="rs">
  <oneOrMore>
    <ref name="el_r"/>
  </oneOrMore>
</element>
```

r — Ergebnis eines Einzeltests

Enthält: p, e, m, t

Enthalten in: rs

```
<element name="r">
  <interleave>
    <ref name="el_p"/>
    <ref name="el_e"/>
    <ref name="el_m"/>
    <ref name="el_t"/>
  </interleave>
</element>
```

p — Kombination der Steine zum Mattsetzen

`0`

: Springer, Läufer und König

`1`

: zwei Läufer und König

`2`

: Turm und König

`3`

: Dame und König

`4`

: Springer, Turm und König

`5`

: Läufer, Turm und König

`10`

: Dame und Turm

`11`

: Dame und Läufer

`12`

: Dame und Springer

`13`

Zwei Türme

`14`

Turm und zwei Läufer

`15`

Turm, Läufer und Springer

`16`

Turm und zwei Springer

Erlaubte Werte: "0", "1", "2", "3", "4", "5", "10", "11", "12", "13", "14", "15", "16"

Enthalten in: r

```
<element name="p">
  <choice>
    <value>0</value>
    <value>1</value>
    <value>2</value>
    <value>3</value>
    <value>4</value>
    <value>5</value>
    <value>10</value>
    <value>11</value>
    <value>12</value>
    <value>13</value>
    <value>14</value>
    <value>15</value>
    <value>16</value>
  </choice>
</element>
```

e Ausgang des Einzeltests

`0`

Matt (Erfolg)

`1`

Patt

`2`

Technisches Remis: Matt nicht mehr möglich

`3`

Die verfügbare Bedenkzeit ist überschritten

`4`

Remis durch Wiederholung der Stellung

`5`

Die Zahl der Züge ist zu hoch

Erlaubte Werte: "0", "1", "2", "3", "4", "5"

Enthalten in: r

```
<element name="e">
  <choice>
    <value>0</value>
    <value>1</value>
    <value>2</value>
    <value>3</value>
    <value>4</value>
    <value>5</value>
  </choice>
</element>
```

m — Die Anzahl der Züge

Enthalten in: r

```
<element name="m">
  <data type="nonnegative_integer"/>
</element>
```

t — Die benötigte Bedenkzeit in Millisekunden

Enthalten in: r

```
<element name="t">
  <data type="integer"/>
</element>
```

Herberts Schachecke: Dateiformat des Spielverlaufs

Der Spielverlauf wird als Textdatei gespeichert, die aus einer Folge verschiedener Arten von Zeilen besteht. Zeilen, die nur aus Leerraum bestehen oder mit dem Zeichen „#” beginnen, sind Kommentare. Eine Zeile beginnt mit einem Kleinbuchstaben, dem ein Doppelpunkt folgt. Der führende Kleinbuchstabe kennzeichnet die Bedeutung der Zeile. Manche Zeilen enthalten allgemeine Informationen über das Spiel, andere Zeilen geben Ereignissse des Spiels in der zeitlichen Reihenfolge an.

a:*POSITION*

POSITION steht für eine Spielsituation, die das Brett empfangen hat (Gegenzug).

s:*POSITION*:*TIME*

POSITION steht für eine Spielsituation, die das Brett sendet (eigener Zug). *TIME* steht für die verbleibende Bedenkzeit im Millisekunden.

l:*URLPATH*

URLPATH steht für den URL-Pfad des Dokuments. Diese Zeile sollte es höchstens einmal geben.

t:*REMAINING*:*FISCHER*:*BRONSTEIN*

Diese Zeile gibt die Zeitbeschränkung für den weiteren Verlauf des Spiels an. Normalerweise gibt es nur eine solche Zeile am Anfang des Spielverlaufs, wenn nicht (zum Beispiel nach einer bestimmten Zahl von Zügen) die Zeitbeschränkung geändert wird. *REMAINING* ist die verbleibende Bedenkzeit ohne Bonus für die folgenden Züge, *FISCHER* der Fischer-Bonus je Zug, *BRONSTEIN* der Bronstein-Bonus je Zug. Alle Zeitangaben sind in Millisekunden.

m:

Der Spieler hat an dieser Stelle den Knopf „zur Analyse” gewählt.

o:*ENDSITUATION*:*TIME*

Die Partie ist beendet. *ENDSITUATION* steht für die Spielsituation, mit der die Partie endet, *TIME* für die verbleibende Bedenkzeit in Millisekunden. *TIME* kann nicht nur bei Ende durch Zeitüberschreitung negativ sein.

Ergebnisse einer Reihe von Partien

Wurzelelement	games

games

Daten zu einer Reihe von Partien.

Enthält: g (+), u (?)

Enthalten in: Wurzel

```
<element name="games">
  <interleave>
    <oneOrMore>
        <ref name="el_g"/>
    </oneOrMore>
    <optional>
        <ref name="el_u"/>
    </optional>
  </interleave>
</element>
```

u

Der URL-Pfad des „Bretts”

Enthalten in: games

```
<element name="u">
  <data type="string"/>
</element>
```

g

Daten einer Partie

Enthält: c, e, r, m

Enthalten in: games

```
<element name="g">
  <interleave>
    <ref name="el_c"/>
    <ref name="el_e"/>
    <ref name="el_r"/>
    <ref name="el_m"/>
  </interleave>
</element>
```

c

Die Farbe der Steine, die der Spieler geführt hat:

b Schwarz
w Weiß
u unbekannt (sollte normalerweise nicht vorkommen)

Erlaubte Werte: "b", "w", "u"

Enthalten in: g

```
<element name="c">
  <choice>
    <value>b</value>
    <value>w</value>
    <value>u</value>
  </choice>
</element>
```

e

Die Farbe, die „zuerst nicht mehr” gezogen hat oder ziehen konnte, und statt eines Zuges das Ende der Partie feststellte.

b Schwarz
w Weiß
u unbekannt (sollte normalerweise nicht vorkommen)

Erlaubte Werte: "b", "w", "u"

Enthalten in: g

```
<element name="e">
  <choice>
    <value>b</value>
    <value>w</value>
    <value>u</value>
  </choice>
</element>
```

r — Art und Grund des Partieendes (End-Spielsituation)

a Remis-Angebot angenommen
r Aufgabe
m Matt
p Patt
u Technische Remis / unzureichendes Material
w Remis durch Stellungswiederholung
e Remis durch Überschreitung einer Zugzahl (50-Züge-Regel)
t Verlust durch Zeitüberschreitung
g Remis durch Zeitüberschreitung
x Abbruch (nicht sportlich)

Erlaubte Werte: "a", "r", "m", "p", "u", "w", "e", "t", "g", "x"

Enthalten in: g

```
<element name="r">
  <choice>
    <value>a</value>
    <value>r</value>
    <value>m</value>
    <value>p</value>
    <value>u</value>
    <value>w</value>
    <value>e</value>
    <value>t</value>
    <value>g</value>
    <value>x</value>
  </choice>
</element>
```

m — Anzahl der Züge

Enthalten in: g

```
<element name="m">
  <data type="nonegative_integer"/>
</element>
```

Herberts Schachecke: Dateiformate

Hier sind die Dateinamenskonventionen und Verweise auf die Dateiformate in den Beispielen zusammengestellt.

Die Ergebnisse eines Matt-Selbstest (`SB_mattst.xhtml`) werden in einem XML-Format (rs) gespeichert. Der vorgeschlagene Dateiname hat die Form `hbs_tr_`*`TIMEMS`*`.xml`. Der Platzhalter *`TIMEMS`* steht für die aktuelle Systemzeit in Millisekunden-Auflösung. Die Buchstaben `hbs` stehen für `HerBerts Schachecke`, die Buchstaben `tr` erinnern an „`Test Results`”. Die Vorlage `hbs_sr_ht.xslt` stellt die Ergebnisse im Browser dar.

Der Verlauf einer Partie (Beispiel `SB_schnellsv.xhtml`) wird in einer zeilenbasierten Textdatei gespeichert. Der vorgeschlagene Dateiname hat die Form `hbs_game_`*`TIMEMS`*`.txt`.

Die Ergebnisse einer Reihe von Partien (Beispiel `SB_schnellst.xhtml`) werden in einem XML-Format (games) gespeichert. Der vorgeschlagene Dateiname hat die Form `hbs_games_`*`TIMEMS`*`.xml`. Die Vorlage `hbs_games_ht.xslt` stellt die Ergebnisse im Browser dar.

Verweise zum Download verschiedener Vorlagen sind in der Datei `SR_dllinks.xhtml` zusammengestellt. Dadurch wird ein Aufwand durch Umleitungen von verschiedenen „Spielorten” vermieden.

Herberts Schachecke: Gruppenschach

Schach zu viert

Angenommen, Peter, Paula, Maria und Josef haben sich an einem Brett verabredet. Die Option `snow` ist aktiv, so dass die anderen es sofort sehen, wenn jemand zieht. Die Synchronisation ist ohne eine übergeordnete „Spielerverwaltung" schon bei zwei Spielern nicht einfach, bei mehr Spielern wird sie zum Glücksspiel. Wenn alle synchronisiert sind, sieht der Spieler, der zuletzt erfolgreich eine neue Spielsituation hergestellt hat, den roten Balken, die anderen Spieler sehen den grünen Balken. Bei zwei Spielern erkennt der Spieler, dessen Balken von Rot auf Grün wechselt, dass nun beide synchronisiert sind und er nun ziehen kann. Bei mehreren Spielern kann sich anfangs nur ein Spieler sicher sein, die anderen können es vermuten.

Nehmen wir nun an, dass Peter die Anfangsstellung mit einem roten Balken sieht. Die anderen sehen einen grünen Balken. Wir nehmen nun an, dass die vier mit Zeitbeschränkung spielen. Peter mit dem roten Balken kann nicht ziehen, weil seine „Uhr" nicht läuft. Aber jeder der anderen drei kann den ersten Zug machen. Nehmen wir an, Maria ist die schnellste und zieht als erste. Sie bekommt den roten Balken. Die anderen drei sehen jetzt das Brett mit einem grünen Balken und den schwarzen Steinen unten zum Zeichen, dass Schwarz am Zug ist.

Jetzt ist Paula die schnellste und macht einen Zug mit Schwarz. Sie sieht jetzt den roten Balken. Die anderen drei sehen einen grünen Balken und die weißen Steine unten zum Zeichen, dass Weiß am Zug ist. So geht es weiter. Wenn Peter auf die Idee kommt, den Knopf „Brett drehen" zu drücken, dann wird das Brett gedreht, aber für den weiteren Verlauf der Partie das automatische Drehen ausgeschaltet. Wenn Peter mal unaufmerksam ist, kann er nicht sofort erkennen, welche Seite am Zug ist. Er kann aber versuchsweise einen weißen Stein anklicken. Wenn der Stein markiert ist, ist Weiß am Zug.

Die Zeit

Wer den grünen Balken sieht, sieht seine Zeit laufen. Aber nur wenn er wirklich zieht, wird die „verbrauchte" Zeit angerechnet. Wenn ein anderer Spieler ihm zuvorkommt, springt seine „Uhr" zurück auf die Zeit, die ihm verblieben war, als sein Balken zum ersten Mal auf Grün wechselte.

In einer früheren Version der Schackecke wurde der Fischer-Bonus dem Spieler jedesmal gutgeschrieben, wenn ihm eine neue Stellung präsentiert wurde. Das finde ich nicht richtig, wenn der Spieler gar nicht zieht. Die angezeigte Zeit ist immer die übertragene Restzeit zuzüglich des Fischer-Bonus und des Bronstein-Bonus abzüglich der für die „Denkperiode" verbrauchten Zeit, abgerundet auf ganze Sekunden. Wenn der Spieler zieht, wird der Bronstein-Bonus von der verbrauchten Zeit abgezogen, der Fischer-Bonus zur verbleibenden Zeit addiert und die verbrauchte Zeit (abzüglich des Bronstein-Bonus) von der verbleibenden Zeit abgezogen. Das Ergebnis ist die neue zu übertragende Restzeit.

Nun kann es auch bei der Option `snow` vorkommen, dass der Server erst als Antwort auf einen Zug meldet: „Danke, aber ein anderer Spieler hat schon gezogen". Das ist dann Pech für den Spieler, die verbrauchte Zeit kommt nicht wieder zurück.

Wenn einer der Spieler die Bedenkzeit überschreitet, ist die Partie beendet.