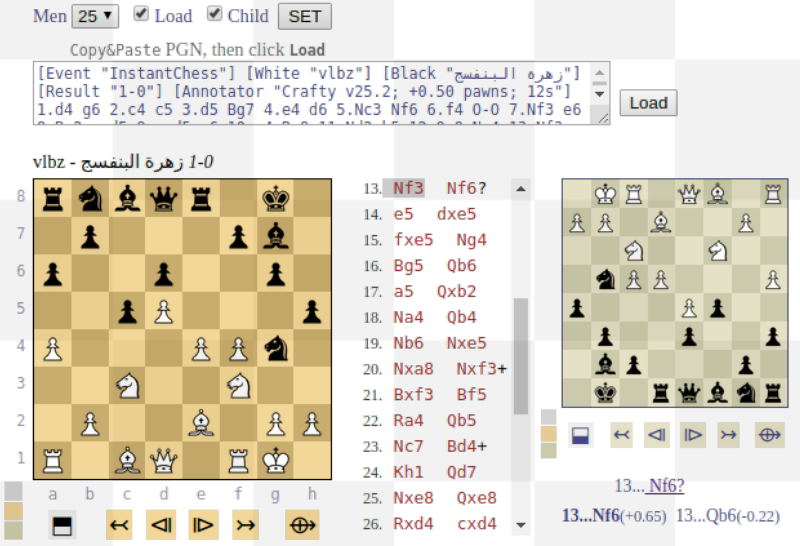
/

- **.git** — *a înregistra și controla (cu* GIT*) modificarea fișierelor, pe parcursul dezvoltării aplicației!*
- **CSS**
  - **brw.css** — *Poziționare câmpuri, piese, diviziuni*
    regula background-position: -K*100%
- **Men** sprite-*uri pentru piesele de șah*
  - **men20.png**
  - **men21.png**
  - **...** ♙♔♕♖♗♘♞♝♜♛♚♟
- **JS** (javaScript, ES6, jQuery; v. *Wikipedia*)
  - **jquery.js**
  - **widget.js** — *componenta "Widget" din* jQuery-UI
  - **brw.js** — *tabele precalculate* From-To,
    expresii-șablon pentru mutări, etc.;
    definește diviziunile și handler*-ele asociate;*
    analiza sintactică a textului PGN*;*
    codificarea binară a mutărilor;
    generatorul de mutări; verificarea legalității;
- **allMoves.html** — *produce tabelele prealabile* From-To
- **Test.html** — *instanțiază* pgnbrw() *în diverse circumstanțe*

Vlad Bazon – Modelarea în browser a partidei de șah

Vlad Bazon

Modelarea în browser a partidei de șah



« `$('tPGN2').pgnbrw({Men:25, Child:true});` »

Vlad Bazon

Modelarea în browser a partidei de șah

Prefață

Realizarea unei aplicații Web nebanale, ar fi „piatra de încercare” pentru orice programator în devenire. Ilustrăm procesul de realizare treptată a unei aplicații de *șah*, evidențiind progresiv și corelând cunoștințele necesare; subiectul exprimat este *partida de șah* furnizată într-un anumit format textual.

Folosim `HTML`, `CSS`, `javaScript` (cu facilități din `ES6` și `jQuery`) și ne ajutăm uneori de alte limbaje (în principal, pentru a constitui „automat” anumite structuri `HTML` și `CSS`). Subliniem că nu ne ocupăm direct, de un limbaj sau altul; *Wikipedia* va putea suplini defectul inerent al acestui studiu de a nu fi un manual.

Impresia ar fi aceea că aplicația și cartea evoluează cumva simultan – ceea ce nu este chiar fals: aplicația a fost realizată demult (v. `http://docerp.ro/`), dar montând acum povestea ei, profităm pe parcurs de șansa unor îndreptări, simplificări sau actualizări. În orice caz, plecăm aici de la lucruri cât se poate de simple și le dezvoltăm treptat, cât se poate de logic și firesc; reperele principale sunt următoarele:

– creem o infrastructură `DOM + CSS` bazată pe *poziționare* (a câmpurilor pe tabla de șah și a pieselor dintr-un `sprite` pe câmpurile tablei), care să depindă cât mai puțin de dimensiuni și de orientarea curentă a tablei; adăugăm (poziționând prin `CSS grid`) diviziunile necesare pentru redarea listei de mutări și a adnotărilor preluate din textul `PGN` al partidei;

– creem o a doua infrastructură, bazată pe reprezentarea `0x88` și pe o anumită codificare binară a mutărilor – permițând verificarea legalității mutărilor și constituirea unui tabel în care o „înregistrare” este poziția rezultată pe tabla de șah prin efectuarea mutării curente;

– asociem acestui tabel de „înregistrări” un mecanism de navigare, legând între ele cele două infrastructuri menționate.

Se cuvin mulțumiri lui *Mihai Bazon* (`https://github.com/mishoo`), pentru ideea pe care ne-am bazat aici, de a defini tabla de șah (cu orientarea și piesele curente) folosind selectori de poziționare `CSS` imbricați (în loc de o structură `<table>`, cum se obișnuiește).

Vlad Bazon, 2019

Cuprins

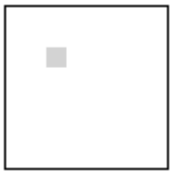
1 Poziționări, poziții și structurări

1.1 Plasarea unui pătrățel pe o tablă

Să zicem că avem un container `<div>` și vrem să așezăm înăuntru un „pătrățel" `<div>`, astfel încât colțul stânga-sus al acestuia să fie la un sfert din lățimea și respectiv, înălțimea tablei „părinte".

Realizăm aceasta setând proprietățile CSS[1] `position:`*relative* pentru container și `position:`*absolute* pentru elementul conținut:

```
<!DOCTYPE html>
<head> <meta charset="utf-8" />
    <style>
        .container {
            width:160px; height:160px;
            position: relative; left:10px;
            border: 1px solid black; }
        .field {
            width:20px; height:20px;
            position: absolute;
            top: 25%; left: 25%;
            background: lightgray; }
    </style>
</head>
<body>
    <div class="container">
        <div class="field"></div>
    </div>
</body>
```



Încărcând acest fișier HTML, browserul va înregistra proprietățile specificate în secțiunea `<style>` și va reda apoi cele două diviziuni din `<body>`, aplicându-le proprietățile de stilare referite prin atributul `class` al fiecăreia. Pătratul exterior `div.container` este poziționat față de marginile ferestrei browserului (`left:10px`, adică la 10px de marginea stângă); în schimb, cel interior `div.field` este poziționat față de marginile „părintelui" său.

Exerciții. Reformulați `<body>` (introducând și un element `<pre>`) astfel încât în fereastra browser-ului, `div.container` să apară alăturat textului-sursă.

[1] https://en.wikipedia.org/wiki/Cascading_Style_Sheets

1.2 Pătrățele aliniate orizontal

Să zicem acum că avem nu doar unul, ci chiar 8 pătrățele și vrem să le poziționăm unul lângă celălalt. Am putea repeta definițiile `.field` de 8 ori, modificând doar valoarea pentru `left` (zero pentru primul, apoi mărind cu câte 160/8=20px pentru următoarele pătrățele).

Dar avem o soluție mai elastică: asociem pătrățelelor o a doua clasă de proprietăți `.Col-N`, `N=1..8` – conținând numai `left`-ul fiecăruia (lăsând în `.field` numai `top`, care are aceeași valoare pentru toate):

```
<head>   <style>
    .container {
        width:176px; height:176px; /* 2px border la .field */
        position: relative;
        border: 1px solid black; }
    .field {
        width:20px; height:20px;
        position: absolute; top: 25%;
        border:1px solid white; background: lightgray; }
    .Col-1 { left: 0 }
    .Col-2 { left: 22px } /* width + border_orizontal */
    .Col-3 { left: 44px }
    .Col-4 { left: 66px }
    .Col-5 { left: 88px }
    .Col-6 { left: 110px }
    .Col-7 { left: 132px }
    .Col-8 { left: 154px }
</style>   </head>
<body>
<div class="container">
    <div class="field Col-1"></div>
    <div class="field Col-2"></div>
    <div class="field Col-3"></div>
    <div class="field Col-4"></div>
    <div class="field Col-5"></div>
    <div class="field Col-6"></div>
    <div class="field Col-7"></div>
    <div class="field Col-8"></div>
</div>
</body>
```

Pentru a putea distinge pătrățelele, am adăugat `border` în clasa `.field`; ca urmare, am modificat lățimea containerului: 160px + 8*(1px *border-stânga* + 1px *border-dreapta*) = 176px. Măsurând în pixeli (20px + 2px border = 22px), valorile `left` sunt 0px, 22px, 44px, etc.

Fiecare „pătrățel” inclus în `div.container` este caracterizat prin `field` (ceea ce asigură poziționarea lui față de marginile tablei și în plus, distanțează față de marginea de sus a acesteia) și prin `Col-N` (care distanțează față de marginea stângă a tablei).

Neajunsuri... (cu cât mai devreme, cu atât mai bine!)

Scriind fișierul redat mai sus, ne dăm seama de anumite inconveniente. Ar fi mai simplu să notăm "`Col2`" în loc de "`Col-2`"; dar mai important, decât să editățm prin *Copy&Paste* liniile de text `<div>` – ar fi mai convenabil să le *generăm* printr-un anumit program. Folosind AWK[2], am avea cea mai simplă (și directă) formulare:

```
awk 'BEGIN {for(i=1;i<=8;i++)
              print "<div class=\"field Col" i "\"></div>"}'
```

Tastând pe linia de comandă această secvență obținem cele 8 linii `<div class="field Col1"></div>` etc. și n-avem decât să le copiem de pe ecran în fișierul HTML respectiv.

Este drept că dacă se lucrează pe un sistem Windows, atunci apar inevitabil *alte* neajunsuri: cum „scriem” fișierul (oare, folosim celebrul „*procesor* de text”?), sau cum procurăm și instalăm „programul AWK”?
Preferăm să lucrăm pe un sistem xUbuntu-Linux, dispunând imediat de editoare de text (cu facilitățile cuvenite editării de „cod-sursă” într-un limbaj sau altul), de compilatorul `gcc` pentru C, de o gamă largă de interpretoare (bash, perl, python, awk, etc.), de pagini de *help* pentru toate cele (orientate pe conținut, nu reclamă) și de o multitudine de „programe utilitare” – toate acestea fiind așa de bune și de necesare, încât de-a lungul timpului multe au fost „clonate” și pentru sistemul comercial Microsoft-Windows.

Revenind la ale noastre, să observăm că pentru a distinge câmpurile este preferabil să le *colorăm* alternativ, în loc să le bordăm.
În plus, măsurarea în *pixeli* folosită mai sus este greu de întreținut (schimbând dimensiunile pentru `.container`, avem de modificat și valorile `left` din `.Col*`); să adoptăm în schimb o măsurare *procentuală*: împărțind dimensiunea 100% a tablei la 8 obținem (exact) 12.5% și avansăm la următoarea coloană adunând 12.5% la *left*-ul celei precedente.

Exercițiu. Dacă încă nu ați avut de-a face cu „Linux”... căutați și instalați `Ubuntu-Linux`; deschideți aplicația *Terminal* și tastați pe linia de comandă `man awk`. Deschideți și editorul de text `gedit`, formulați un program C și compilați-l folosind `gcc` (`gcc --help` sau *Google* vă va spune cum să faceți).

[2] https://en.wikipedia.org/wiki/AWK

1.3 Tabla de șah

O tablă de șah are 64 de pătrățele, dispuse pe 8 linii și 8 coloane. Eliminăm `top` din clasa `.field` și definim clasele `Row*`, similare cu `Col*` (și conținând `top` pentru fiecare *linie*). Pătrățelele tablei vor fi definite acum prin `<div class="field ColN RowN">`, unde `.field` setează `position:absolute` și indică browserului dimensiunile pătrățelului, iar `.ColN` și `.RowN` precizează "left" și "top". În loc de 64 definiții `.field` cu setări pentru "top" și "left", avem numai 17 definiții – una comună pentru `.field`, 8 definiții `Col*` ("left" pentru fiecare coloană) și 8 definiții `Row*` ("top" pentru fiecare linie):

```
<style>
    .container {
        width: 176px; height: 176px;
        position: relative;
        border: 1px solid black; }
    .field {
        width: 20px; height: 20px;
        position: absolute;
        padding: 1px; /* în loc de border */ }
    .WhiteField {background: white}
    .BlackField {background: lightgrey}
    .Col1 {left:    0%}  .Row1 {top:    0%}
    .Col2 {left: 12.5%}  .Row2 {top: 12.5%}
    .Col3 {left:   25%}  .Row3 {top:   25%}
    .Col4 {left: 37.5%}  .Row4 {top: 37.5%}
    .Col5 {left:   50%}  .Row5 {top:   50%}
    .Col6 {left: 62.5%}  .Row6 {top: 62.5%}
    .Col7 {left:   75%}  .Row7 {top:   75%}
    .Col8 {left: 87.5%}  .Row8 {top: 87.5%}
</style>
```

Bineînțeles că pentru a înscrie în fișier ultimele definiții CSS ne-am folosit iarăși de AWK (pentru a produce „automat” liniile respective):

```
awk 'BEGIN {
     for(i=1;i<=8;i++) {
        pos = sprintf("%4g%%", (i-1)*12.5);
        print(".Col" i " {left: " pos "}  .Row" i " {top: " pos "}")
     }}'
```

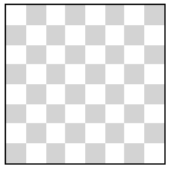
Cum adăugăm apoi, liniile `<div class="field ColN RowN"></div>`? Cel mai simplu ar fi să folosim iarăși AWK... Dar între secvența CSS tocmai generată prin AWK și secvența HTML formată din cele 64 de diviziuni pe care vrem să o generăm cumva acum, avem o diferență contextuală impor-

tantă: prima este fixată (în segmentul `<head>`), în timp ce a doua (`HTML`) ar trebui să poată fi generată oriunde s-ar dori, eventual în mai multe locuri din pagină.

HTML este destinat unui browser, iar acesta încorporează și un interpretor de limbaj javaScript (sau JS) – deci soluția firească pentru generarea de conținut `HTML` constă în a folosi JS (*nu* AWK).

Să prevedem în segmentul `<body>` un element `div.container` și un element `<script>` în care formulăm o funcție JS care să genereze cele 64 de „pătrățele" în containerul pentru care este invocată:

```
<body>
    <div class="container" id="tabla"></div>
    <script>
    function setChessTable() {
        var html = [];
        for(var row = 1; row <= 8; row++) {
            for(var col = 1; col <= 8; col++) {
                var fieldColor = (row + col) & 1 ?
                        "BlackField" : "WhiteField";
                html.push("<div class='field",
                                    " Row", row,
                                    " Col", col,
                                    " ", fieldColor,
                          "'></div>")
            }
        }
        return html.join('');
    }
    document.getElementById('tabla')
            .innerHTML = setChessTable();
    </script>
</body>
```



Am adăugat „pătrățelelor" și clasa `.BlackField`, sau `.WhiteField` – asigurând alternanța *background* a câmpurilor: câmpul este „negru" dacă suma indicilor 1..8 de linie și coloană este *impară*. Folosind instrumentul *Web Developer* al browser-ului, putem vedea că apelul `setChessTable()` a constituit în diviziunea indicată 64 de elemente `<div>`, fiecare având specificate câte patru clase de proprietăți CSS; de exemplu, câmpul aflat pe linia 5 și coloana 4 – socotind de sus în jos și de la stânga la dreapta – este definit prin `<div class="field Row5 Col4 BlackField"></div>`.

Funcția `setChessTable()` a permis constituirea *dinamică* a diviziunilor respective: fișierul încărcat de browser introducea un singur element vizibil, `<div id='tabla'>`, având conținutul vid; dar imediat după ce l-a încărcat,

browserul a trebuit să execute instrucțiunea din `<script>` – ceea ce a determinat execuția funcției `setChessTable()` și înscrierea rezultatului acesteia drept conținut al elementului identificat prin `'tabla'`.

Cu specificația modernă ES6[3], funcția s-ar formula mai bine astfel:

```
function setChessTable() {
    let html = [],  seq = [1,2,3,4,5,6,7,8];
    for(let row of seq)
        for(let col of seq) {
            let fieldColor = (row + col) & 1 ?
                             "BlackField" : "WhiteField";
            html.push(`<div class=
                  "field Row${row} Col${col} ${fieldColor}"
                       ></div>`);
        }
    return html.join('');
}
```

`let` – spre deosebire de `var` – creează variabilele în contextul *local* al funcției sau blocului respectiv; `for...of` iterează pe componentele unui obiect (aici, pe tabloul '`seq`'). Ambalând cu ` (caractere *backstick*, în loc de ghilimele sau apostrof), rezultă un „șablon literal" în care putem intercala `${`*variabilă*`}`, obținând substituirea variabilei indicate, prin valoarea curentă a acesteia.

Pentru a imita o „tablă de șah" veritabilă, vom avea de adăugat o diviziune dedesubt și una în stânga, pentru a nota coloanele și liniile; iar aceste două diviziuni pot fi și ele poziționate „absolut" față de tabla respectivă (putând refolosi definițiile `Col*` și `Row*`). Dar aspectul *dinamic* este cel mai important: în orice moment, există niște *piese* de șah care ocupă anumite câmpuri; trebuie prevăzută și cerința de a „inversa" tabla (comutând „sus–jos", notația și piesele).

Exercițiu. Mai sus, liniile au fost indexate (în mod implicit) de sus în jos-ul tablei – la fel cu liniile `<tr>` dintr-un element `<table>`, în HTML.
Modificați definițiile `.Row*` și deasemenea, funcția `setChessTable()`, încât liniile tablei să fie indexate de jos în sus.

Exercițiu. Angajând eventual AWK, formulați un fișier HTML care să redea folosind `<table>`, o tablă de șah.

Exercițiu. Formulați un program C++ care să producă un fișier HTML conținând elementele `<table>` necesare pentru redarea în fereastra browserului a celor 92 de soluții ale problemei damelor[4].

[3] https://en.wikipedia.org/wiki/ECMAScript
[4] https://en.wikipedia.org/wiki/Eight_queens_puzzle