

C.N. "Dr. Ioan Meșotă" Brașov
Profil matematică-informatică

Lucrare de atestat

AI SAH

Realizat de Chiribuță Robert Adrian
Profesor coordonator Popovici Maria
Clasa a XII-a B

Sesiunea Brașov Mai 2024

CUPRINS

1. MOTIVAREA ALEGERII TEMEI
2. LIMBAJE FOLOSITE
3. STRUCTURA PROIECTULUI
4. CERINȚE MINIME DE SISTEM
5. PRORGRAME UTILIZATE
6. BIBLIOGRAFIE

1 Motivarea alegerii temei

Am ales această temă din mai multe motive:

- În primul rând algoritmii și metodele de programare folosite în crearea unui site cu bot de șah sunt folosiți în multe domenii diferite din informatică și teoria jocurilor.
- De asemenea, este un proiect complex pe care mi-l doream să îl fac de ceva timp, care îți pune la încercare și în final dezvoltă abilitățile tehnice și de gestionare a timpului, având o dată fixă până când trebuie terminat. Și totodată te învață concepte și tehnici noi.

2 Limbaje folosite

- **HTML** sau HyperText Markup Language, este un limbaj de marcă utilizat pentru crearea și structurarea conținutului unei pagini web. Prin intermediul unui set de etichete și atributelor, HTML definește elementele de bază ale unei pagini web, cum ar fi titlurile, paragrafele, imagini și link-uri. Acesta permite dezvoltatorilor să organizeze și să formateze conținutul într-un mod structurat și semnificativ, esențial pentru crearea paginilor web ușor de înțeles și de navigat.
- **CSS**, sau Cascading Style Sheets, este un limbaj de stilizare folosit pentru a formata și a stiliza aspectul vizual al unei pagini web. Prin definirea regulilor de stil, precum culoarea, fontul, dimensiunea și alinierea elementelor HTML, CSS permite dezvoltatorilor să creeze aspectul și simțul dorit pentru site-uri web. Prin separarea conținutului și a prezentării, CSS îmbunătățește flexibilitatea și consistența în designul web, facilitând personalizarea și gestionarea aspectului unei pagini web în mod eficient.
- **TypeScript** TypeScript este un limbaj de programare care extinde limbajul JavaScript, adăugând tipurile statice la limbaj. Acesta oferă programatorilor posibilitatea de a defini tipuri pentru variabile, argumente de funcții și valori de returnare, permițând astfel detectarea erorilor în timpul dezvoltării și îmbunătățirea robusteții codului. Prin adăugarea tipurilor, TypeScript facilitează înțelegerea și menținerea codului, crește productivitatea și reduce numărul de erori în timpul executării.
- **Rust** este un limbaj de programare modern și eficient, proiectat pentru a oferi performanță și siguranță la nivel de sistem, fără a compromite abordarea ergonomică și ușurința de utilizare. Prin intermediul sistemului său de tipuri puternic, Rust permite scrierea de cod care este sigur din punct de vedere al memoriei, fără a fi necesare tehnici complexe de gestionare a resurselor.
- **WebAssembly (WASM)** WASM, sau WebAssembly, este un format binar și un limbaj de reprezentare a codului portabil, proiectat pentru a fi executat în medii web. Acesta oferă o alternativă eficientă și sigură la JavaScript pentru executarea codului în browser, permițând compilarea codului într-un format compact și performant. Datorită performanței ridicate și a interoperabilității cu limbaje precum C/C++ și Rust, WASM devine o alegere atractivă pentru crearea aplicațiilor web complexe și interactive, cum ar fi jocurile, aplicațiile de editare foto/video și multe altele.

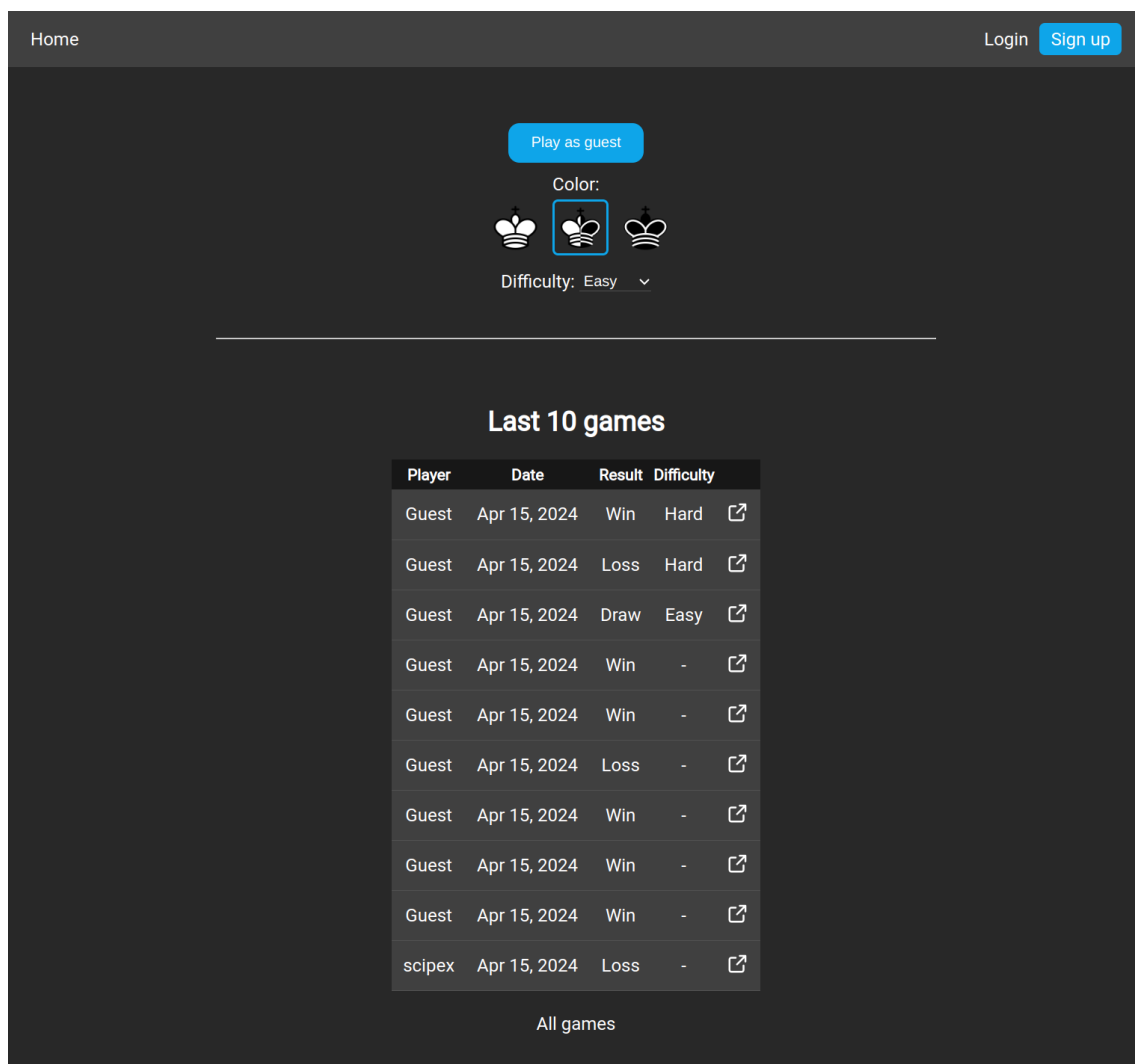
3 Structura Proiectului

Proiectul este compus din 3 componente:

1. Server (backend) în Rust
2. Interfața (frontend) în HTML, CSS, și TypeScript
3. Librăria de șah în Rust, compilată în WASM

3.1 Frontend

3.1.1 Pagina principala



Această pagină conține mai multe elemente:

- Bara de navigație, prezentă pe toate paginile cu un link către pagina principală în partea stângă, iar în partea dreaptă linkuri pentru înregistrare/autentificare, dacă utilizatorul nu este autentificat, sau un link pentru deconectare și unul pentru vizitarea profilului, dacă utilizatorul este autentificat

```
1 fn navbar(user: Option<User>) -> String {
2     html!(
3         <div class="navbar-wrapper">
4             <div class="navbar">
5                 <a href="/">"Home"</a>
6
7                 <div class="user">
8                     {match user {
9                         Some(user) => logged_in(user),
10                        None => not_logged_in(),
11                    }}
12                 </div>
13             </div>
14         </div>
15     )
16 }
17
18 fn logged_in(user: User) -> String {
19     html!(
20         <a href="/logout" class="login">"Logout"</a>
21         <a href={format!("/users/{})", &user.username} class="
22 profile">{&user.username}</a>
23     )
24 }
25
26 fn not_logged_in() -> String {
27     html!(
28         <a href="/login" class="login">"Login"</a>
29         <a href="/register" class="register">"Sign up"</a>
30     )
31 }
```

- Meniu pentru începerea unui nou joc, realizat printr-un form HTML

```
1 fn new_game(button_text: &str) -> String {
2     html!(
3         <form action="/new-game" class="gameopts">
4             <button class="newgame" type="submit">{button_text}</
5 button>
6             <div class="coloropt">
7                 <p style="margin: 0">"Color:"</p>
8                 <div style="display: flex">
9                     {color_select("white", "checked")}
10                    {color_select("random", "")}
11                    {color_select("black", "")}
12                </div>
13            </div>
14        </form>
15    )
16 }
```

```

12         </div>
13         <div class="divopt">
14             <p style="margin: 0">"Difficulty:"</p>
15             <select name="difficulty" class="difficulty">
16                 <option value="0">"Easy"</option>
17                 <option value="1">"Medium"</option>
18                 <option value="2">"Hard"</option>
19             </select>
20         </div>
21     </form>
22 )
23 }
24
25 fn color_select(color: &str, checked: &str) -> String {
26     html!(
27         <div class="tooltip">
28             <input type="radio" name="color" id=color value=color
29             style="appearance: none" {checked} />
30             <label for=color>
31                 <img class="colorselect" src=format!("/assets/
32 select-{}.png", color) />
33             </label>
34             <span class="tooltiptext">{color}</span>
35         </div>
36     )
37 }

```

- Tabel cu ultimele 10 jocuri

Generat pe server

```

1 pub fn games_list(games: Vec<Game>) -> String {
2     html! {
3         <table class="games">
4             <tr class="games-header">
5                 <th style="padding: 5px 0px;">"Player"</th>
6                 <th>"Date"</th>
7                 <th>"Result"</th>
8                 <th>"Difficulty"</th>
9                 <th></th>
10            </tr>
11            {games.into_iter().map(game_html).collect::<String>()}
12        </table>
13    }
14 }

```

```

1 pub fn game_html(game: Game) -> String {
2     let date_format = time::format_description::parse("[month repr
3 :short] [day], [year]").unwrap();
4     let date = game.played_at.format(&date_format).unwrap().
5     to_string();
6     let difficulty = game.difficulty.clone().unwrap_or("-".into())
7     ;
8 }

```

```

5
6     html! {
7         <tr class="game" onclick={format!("location.href='/games
8         /{}';", game.id)}>
9             <td>{player(&game)}</td>
10            <td>{date}</td>
11            <td>{game.result}</td>
12            <td>{difficulty}</td>
13        </tr>
14    }
15
16 fn player(game: &Game) -> String {
17     match &game.player {
18         Some(player) => html! {
19             <a class="player" href={format!("/users/{}", player)}>
20                 {player}</a>
21             },
22         None => html! {
23             "Guest"
24         },
25     }
26 }

```


3.1.2 Pagina cu istoricul jocurilor

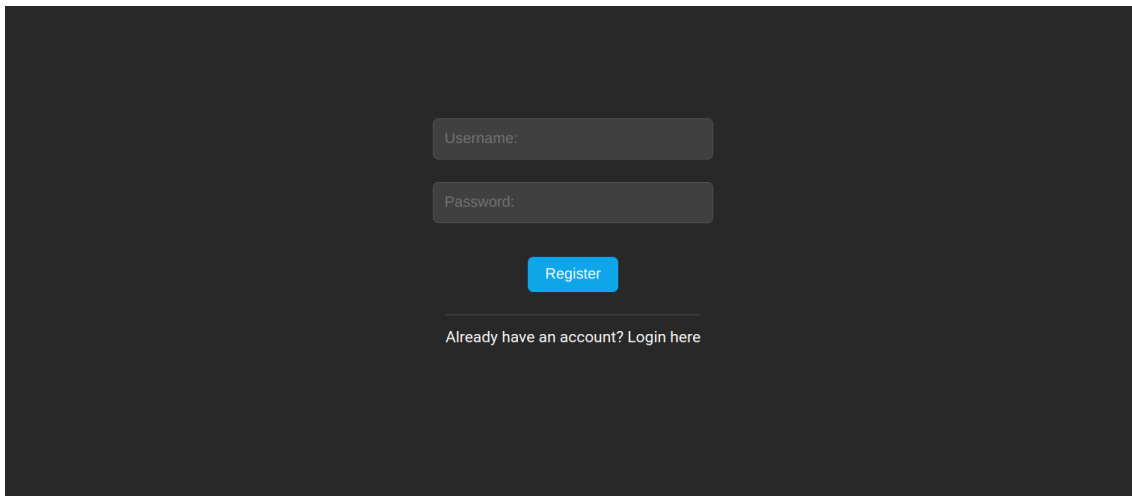
Home Login Sign up

All games played

Player	Date	Result	Difficulty	
Guest	Apr 15, 2024	Win	Hard	
Guest	Apr 15, 2024	Loss	Hard	
Guest	Apr 15, 2024	Draw	Easy	
Guest	Apr 15, 2024	Win	-	
Guest	Apr 15, 2024	Win	-	
Guest	Apr 15, 2024	Loss	-	
Guest	Apr 15, 2024	Win	-	
Guest	Apr 15, 2024	Win	-	
Guest	Apr 15, 2024	Win	-	
Guest	Apr 15, 2024	Win	-	
scipex	Apr 15, 2024	Loss	-	
Gigi	Apr 14, 2024	Win	-	
Gigi	Apr 14, 2024	Win	-	
scipex	Apr 14, 2024	Loss	-	
Guest	Apr 14, 2024	Win	-	
Guest	Apr 14, 2024	Draw	-	
Guest	Apr 14, 2024	Win	-	
Guest	Apr 14, 2024	Win	-	
Guest	Apr 13, 2024	Win	-	
scipex	Apr 06, 2024	Win	-	
scipex	Apr 06, 2024	Loss	-	
Guest	Apr 06, 2024	Loss	-	
Guest	Apr 06, 2024	Loss	-	

Pagina este generată la fel ca tabelul de pe pagina principală, dar cu mai mult de 10 elemente

3.1.3 Pagina de înregistrare



Username:

Password:

Register

Already have an account? [Login here](#)

```
1 fn register_form(  
2     username_value: Option<&str>,  
3     username_error: Option<&str>,  
4     password_error: Option<&str>,  
5 ) -> String {  
6     html! (  
7         <form action="/register" class="register-form" method="post">  
8             <div class="field-div">  
9                 <input  
10                     type="text"  
11                     id="username"  
12                     name="username"  
13                     placeholder="Username: "  
14                     class="field"  
15                     value={username_value.unwrap_or("")}  
16                     required  
17                 />  
18                 <span class="error">{username_error.unwrap_or("")}</span>  
19             </div>  
20             <div class="field-div">  
21                 <input  
22                     type="password"  
23                     id="password"  
24                     name="password"  
25                     placeholder="Password: "  
26                     class="field"  
27                     required  
28                 />  
29                 <span class="error">{password_error.unwrap_or("")}</span>  
30             </div>  
31             <button type="submit" class="register-btn">"Register"</button>
```

```

32     </form>
33
34     <div class="below-form">
35         <a href="/login" class="login">"Already have an account? Login
here"</a>
36     </div>
37     )
38 }

```

Form-ul de înregistrare are și câmpuri pentru erori, în cazul în care username-ul este prea scurt sau există un cont cu același username, sau în cazul în care parola este prea scurtă.

3.1.4 Pagina de autentificare

The image shows a login form on a dark background. It consists of two text input fields, one for 'Username:' and one for 'Password:'. Below these fields is a blue button labeled 'Login'. At the bottom of the form, there is a link that says 'Don't have an account? Register here'.

```

1 fn login_form(
2     username_value: Option<&str>,
3     username_error: Option<&str>,
4     password_error: Option<&str>,
5 ) -> String {
6     html! {
7         <form action="/login" class="register-form" method="post">
8             <div class="field-div">
9                 <input
10                     type="text"
11                     id="username"
12                     name="username"
13                     placeholder="Username: "
14                     class="field"

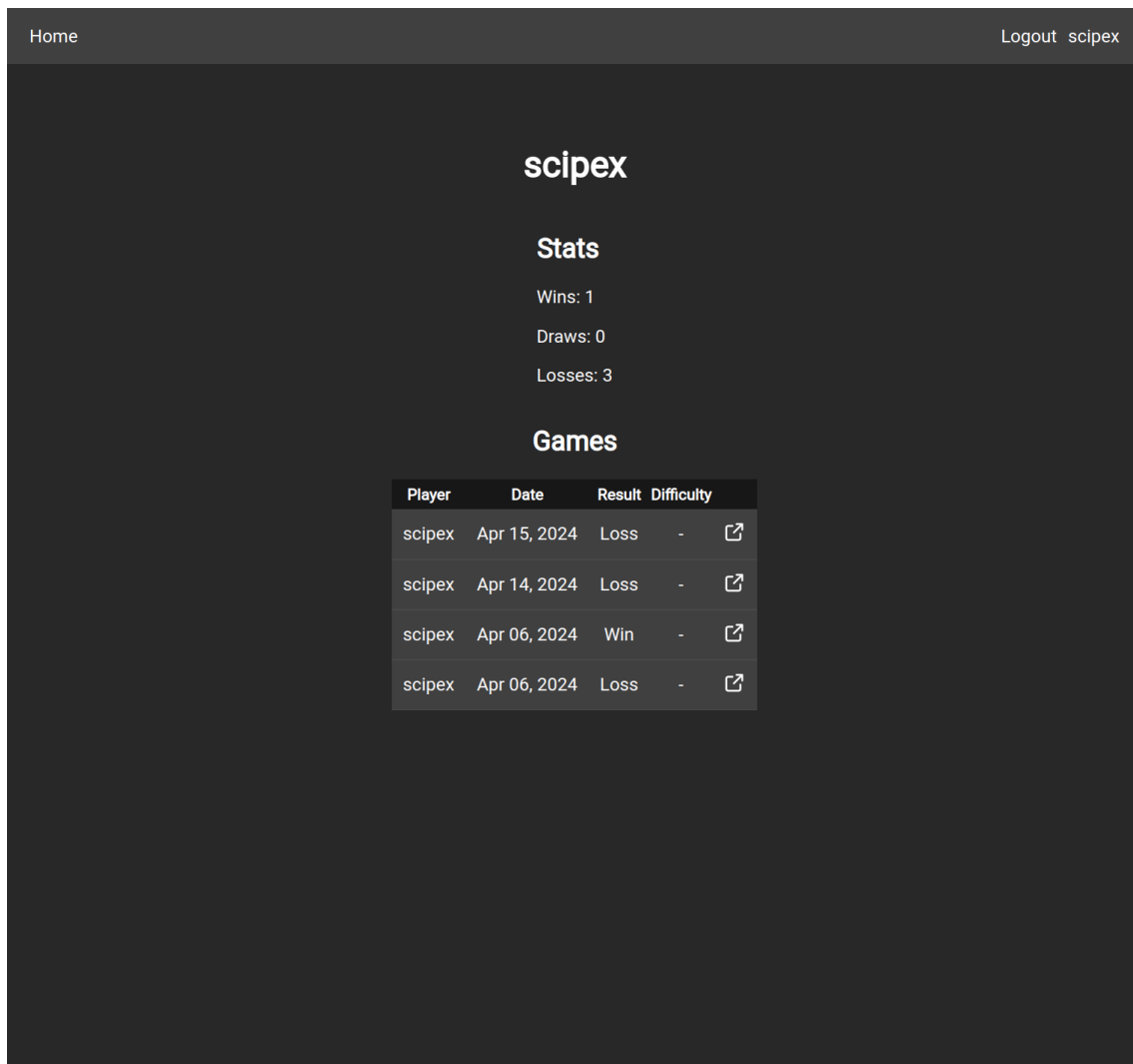
```

```

15         value={username_value.unwrap_or("")}
16         required
17     />
18     <span class="error">{username_error.unwrap_or("")}</span>
19 </div>
20 <div class="field-div">
21     <input
22         type="password"
23         id="password"
24         name="password"
25         placeholder="Password: "
26         class="field"
27         required
28     />
29     <span class="error">{password_error.unwrap_or("")}</span>
30 </div>
31 <button type="submit" class="register-btn">"Login"</button>
32 </form>
33
34 <div class="below-form">
35     <a href="/register" class="login">"Don't have an account?
36     Register here"</a>
37 </div>
38 }

```

3.1.5 Pagina cu profilul utilizatorilor



```
1 <div class="content">
2   <h1>{&user.username}</h1>
3   {stats_html(stats)}
4   <h2>"Games"</h2>
5   {games_list(games)}
6 </div>
```

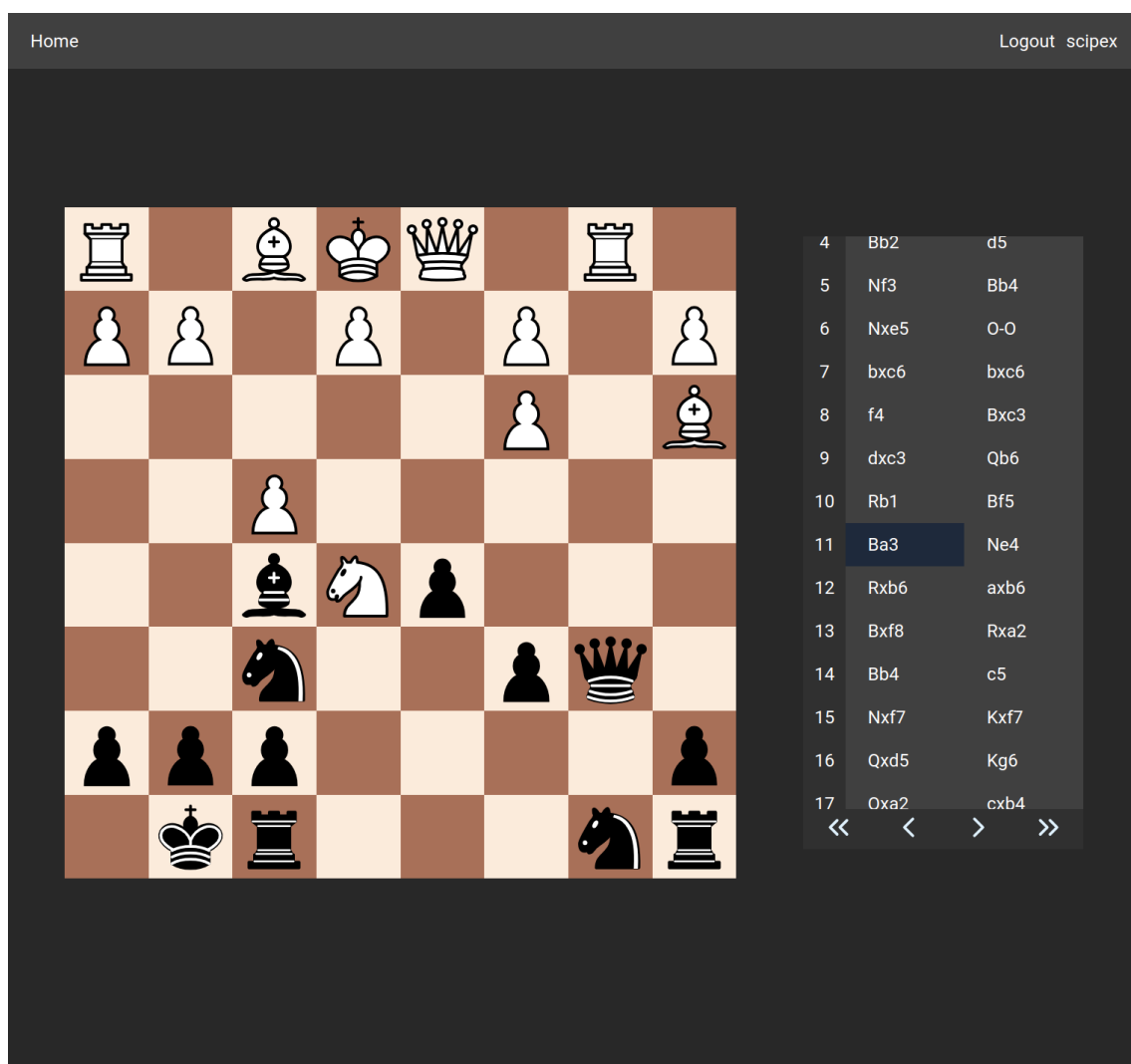
```
1 fn stats_html(stats: [i64; 3]) -> String {
2   html! (
3     <div class="stats">
4       <h2>"Stats"</h2>
5       <p>
6         "Wins: " {stats[0]}
7       </p>
8       <p>
9         "Draws: " {stats[1]}
```

```

10     </p>
11     <p>
12         "Losses: " {stats[2]}
13     </p>
14 </div>
15 )
16 }

```

3.1.6 Pagina jocului



Pagina jocului folosește TypeScript și librăria Lit. Este împărțită în 2 secțiuni:

- Tabla de joc care interacționează atât la mouse cât și la ecranele tactile

```

1 <div
2   class="container"
3   @mousedown=${this.mouse_down}
4   @mouseup=${this.mouse_up}
5   @mousemove=${this.mouse_move}
6   @contextmenu=${this.right_click}
7 >
8   ${this.promotion_menu()}
9   <div class="board" ${ref(this.board_ref)}>
10     ${Array.from(Array(64).keys()).map((i) => this.board_tile(i))}
11   </div>
12
13   <div class="piece-hover" ${ref(this.piece_hover_ref)}></div>
14 </div>
15
16
17 board_tile(i: number) {
18   const x = this.flip ? i % 8 : 7 - (i % 8);
19   const y = this.flip ? 7 - Math.floor(i / 8) : Math.floor(i / 8);
20   const color = (x + y) % 2 === 0 ? "black" : "white";
21   const idx = x + y * 8;
22
23   let piece_style = styleMap({});
24   if (this.pieces.has(idx)) {
25     const p = this.pieces.get(idx)!;
26
27     piece_style = styleMap({
28       "background-image": `url(${piece_asset(p.kind, p.color)})`,
29       "background-size": "contain",
30     });
31   }
32
33   return html`<div
34     id="tile-${idx}"
35     class="tile ${color}"
36     style=${piece_style}
37     @mouseenter=${(e: MouseEvent) => this.tile_mouseenter(e, idx)}
38     @mouseleave=${(e: MouseEvent) => this.tile_mouseleave(e, idx)}
39   >
40     <div class="legal_move" id="move-${idx}"></div>
41   </div>`;
42 }

```

- Meniul din dreapta care conține lista cu mutări și butoane pentru a naviga la începutul jocului, la mutarea precedentă, la mutarea următoare și la ultima mutare, în această ordine

```

1 <div>
2   <div class="moves-wrapper" ${ref(this.moves_ref)}>
3     <div class="moves">
4       ${this.move_pairs().map((pair, i) => this.render_pair(pair,
5         i))}

```

```

5     </div>
6 </div>
7 <div class="controls">
8     <button class="control" @click=${() => this.handle_ply_select
9         (0)}>
10         ${chevron_dleft(30)}
11     </button>
12     <button
13         class="control"
14         @click=${() => {
15             if (this.drawn_ply > 0) {
16                 this.handle_ply_select(this.drawn_ply - 1);
17             }
18         }}
19     >
20         ${chevron_left(30)}
21     </button>
22     <button
23         class="control"
24         @click=${() => {
25             if (this.drawn_ply < this.moves.length) {
26                 this.handle_ply_select(this.drawn_ply + 1);
27             }
28         }}
29     >
30         ${chevron_right(30)}
31     </button>
32     <button
33         class="control"
34         @click=${() => this.handle_ply_select(this.moves.length)}
35     >
36         ${chevron_dright(30)}
37     </button>
38 </div>
39
40
41 render_pair(pair: MovePair, idx: number) {
42     return html`
43         <div class="idx">${idx + 1}</div>
44         <div
45             class=${(this.drawn_ply === idx * 2 + 1 ? "selected" : "") +
46             " white"}
47             @click=${() => this.handle_ply_select(idx * 2 + 1)}
48         >
49             ${move_to_str(pair.white)}
50         </div>
51         <div
52             class=${(this.drawn_ply === idx * 2 + 2 ? "selected" : "") +
53             " black"}
54             @click=${() => this.handle_ply_select(idx * 2 + 2)}

```



```

53     >
54     ${pair.black ? move_to_str(pair.black) : ""}
55   </div>
56   `;
57 }

```

3.1.7 Stilurile CSS

```

1  body {
2    margin: 0;
3    height: 100%;
4    background-color: #282828;
5    color: white;
6    font-family: Roboto;
7  }
8
9  a,
10 button {
11   text-decoration: none;
12   color: white;
13 }
14
15 .navbar-wrapper {
16   width: 100%;
17   height: 50px;
18   background-color: #404040;
19   position: absolute;
20   top: 0;
21 }
22
23 .navbar {
24   height: 100%;
25   display: flex;
26   justify-content: space-between;
27   align-items: center;
28   padding: 0 20px;
29 }
30
31 .user {
32   display: flex;
33   align-items: center;
34   gap: 10px;
35 }
36

```

```

37 .login:hover {
38     color: #0ea5e9;
39 }
40
41 .register {
42     background-color: #0ea5e9;
43     padding: 5px 10px;
44     border-radius: 5px;
45 }
46
47 .content {
48     padding-top: 100px;
49     max-width: 800px;
50     margin: 0 auto;
51     color: white;
52     display: flex;
53     flex-direction: column;
54     align-items: center;
55 }
56
57 .game {
58     cursor: pointer;
59 }
60
61 .game:hover {
62     background-color: #525252;
63 }
64
65 .newgame {
66     padding: 10px 20px;
67     border-radius: 10px;
68     background-color: #0ea5e9;
69     cursor: pointer;
70     border: none;
71     text-decoration: none;
72     color: white;
73 }
74
75 .field {
76     background-color: #404040;
77     color: white;
78     border: 1px solid #525252;
79     border-radius: 5px;
80     padding: 10px;
81     width: 100%;
82     box-sizing: border-box;
83 }
84
85 .register-form {
86     display: flex;
87     flex-direction: column;

```

```

88     align-items: center;
89     gap: 20px;
90     width: 250px;
91 }
92
93 .register-btn {
94     background-color: #0ea5e9;
95     padding: 8px;
96     border-radius: 5px;
97     cursor: pointer;
98     border: none;
99     color: white;
100    width: 80px;
101    margin-top: 10px;
102 }
103
104 .error {
105     color: red;
106     display: inline-block;
107     font-size: 12px;
108 }
109
110 .field-div {
111     width: 100%;
112 }
113
114 .below-form {
115     margin-top: 20px;
116     padding-top: 10px;
117     border-top: 1px solid #525252;
118     font-size: 14px;
119 }
120
121 .games {
122     text-align: center;
123     border: 0px;
124     background-color: #404040;
125     border-spacing: 0px;
126 }
127
128 .games-header {
129     font-size: 14px;
130     background-color: #171717;
131 }
132
133 td {
134     border-bottom: 1px solid #525252;
135     padding: 10px 10px;
136 }
137
138 .player:hover {

```

```

139     text-decoration: wavy underline;
140 }
141
142 .gameopts {
143     display: flex;
144     gap: 10px;
145     flex-direction: column;
146     align-items: center;
147 }
148
149 .coloropt {
150     display: flex;
151     gap: 4px;
152     flex-direction: column;
153     align-items: center;
154 }
155
156 .coloropt input:checked+label>.colorselect {
157     border: 2px solid #0ea5e9;
158     border-radius: 5px;
159     box-sizing: border-box;
160 }
161
162 .colorselect {
163     cursor: pointer;
164 }
165
166 img {
167     width: 50px;
168     height: 50px;
169 }
170
171 .tooltip {
172     position: relative;
173     display: inline-block;
174 }
175
176 .tooltip .tooltiptext {
177     visibility: hidden;
178     background-color: #52525b;
179     color: white;
180     text-align: center;
181     border-radius: 6px;
182     padding: 5px;
183     position: absolute;
184     z-index: 1;
185     bottom: -100%;
186     left: 0;
187     opacity: 0;
188     transition: opacity 0.3s;
189     transition-delay: 0.7s;

```

```
190 }
191
192 .tooltip:hover .tooltiptext {
193     visibility: visible;
194     opacity: 1;
195 }
196
197 .tooltip .tooltiptext::after {
198     content: " ";
199     position: absolute;
200     bottom: 100%;
201     left: 50%;
202     margin-left: -5px;
203     border-width: 5px;
204     border-style: solid;
205     border-color: transparent transparent #52525b transparent;
206 }
207
208 .divopt {
209     display: flex;
210     gap: 4px;
211     align-items: center;
212 }
213
214 .difficulty {
215     background-color: #282828;
216     color: white;
217     border: 0;
218     border-bottom: 1px solid #525252;
219 }
```

3.2 Backend

3.2.1 Baza de date

Baza de date conține următoarele tabele:

- Tabelul User

```
1 CREATE TABLE users (  
2   id INT PRIMARY KEY AUTO_INCREMENT,  
3   username VARCHAR(256) NOT NULL UNIQUE,  
4   password VARCHAR(256) NOT NULL,  
5   token VARCHAR(36) UNIQUE,  
6 );
```

- Tabelul Games

```
1 CREATE TABLE games (  
2   id INT PRIMARY KEY AUTO_INCREMENT,  
3   player VARCHAR(64) REFERENCES users(id),  
4   moves JSON NOT NULL,  
5   result VARCHAR(256) NOT NULL,  
6   difficulty VARCHAR(8),  
7  
8   played_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP  
9 );
```

3.2.2 Autentificare și înregistrare

Înregistrarea se face în mai mulți pași:

1. Validarea datelor
 - Usernameul trebuie să fie unic
 - Parola trebuie să aibă minim 4 caractere
2. Criptarea parolei
3. Inserarea în baza de date a userului
4. Generarea unui token unic pentru cookie

```
1 pub async fn post(  
2     State(state): State<AppState>,  
3     Form(mut data): Form<RegisterForm>,  
4 ) -> impl IntoResponse {  
5     // Validarea lungimii parolei  
6     if data.password.len() < 4 {  
7         return Html(render_index(html!(  
8             <div class="content">  
9                 {register_form(None, None, Some("Password must be at  
10                    least 4 characters long"))}  
11             </div>  
12         )))  
13         .into_response();  
14     }  
15     //Criptarea parolei  
16     data.password = {  
17         let salt = SaltString::generate(&mut OsRng);  
18         let argon2 = Argon2::default();  
19         argon2  
20             .hash_password(data.password.as_bytes(), &salt)  
21             .expect("Failed to hash password")  
22             .to_string()  
23     };  
24     // Inserarea în baza de date  
25     let q = sqlx::query!(  
26         r#"  
27         INSERT INTO users (username, password)  
28         VALUES (?, ?)  
29         "#,  
30         data.username,  
31         data.password,  
32     )  
33     .execute(&state.pool)  
34     .await;  
35     let id = sqlx::query!(
```

```

36         r#"
37         SELECT id FROM users
38         WHERE username = ?
39         "#,
40         data.username,
41     )
42     .fetch_one(&state.pool)
43     .await
44     .unwrap()
45     .id;
46
47     // Generarea unui token unic
48     let token = update_token(&state.pool, id).await.unwrap();
49
50     match q {
51         Ok(_) => {
52             let cookie = Cookie::build(("SESSION", token.to_string()))
53                 .path("/")
54                 .max_age(Duration::days(2))
55                 .build();
56             let mut headers = HeaderMap::new();
57             headers.insert(SET_COOKIE, cookie.to_string().parse().
unwrap());
58             (headers, Redirect::to("/")).into_response()
59         }
60
61         Err(e) => {
62             // Daca inserarea în baza de date a esuat deoarece
63             // usernameul nu este unic se afiseaza pagina de
64             // înregistrare cu eroare
65             if let sqlx::Error::Database(db_err) = &e {
66                 if db_err.kind() == sqlx::error::ErrorKind::
UniqueViolation {
67                     return Html(render_index(html!(
68                         <div class="content">
69                             {register_form(None, Some("Username
already exists"), None)}
70                         </div>
71                     )))
72                     .into_response();
73                 }
74             }
75         }
76     }
77 }

```


Autentificarea se face în mod asemănător cu înregistrarea:

1. Validarea lungimii parolei
2. Verificarea usernameului în baza de date

```
1 pub async fn post(State(state): State<AppState>, Form(data): Form<
  LoginForm>) -> impl IntoResponse {
2   // Validarea lungimii parolei
3   if data.password.len() < 4 {
4       return Html(render_index(html!(
5           <div class="content">
6               {login_form(None, None, Some("Password must be at least
7               4 characters long"))}
8           </div>
9       )))
10      .into_response();
11  }
12
13  let q = sqlx::query_as!(
14      User,
15      r#"
16      SELECT * FROM users
17      WHERE username = ?
18      "#,
19      data.username
20  )
21  .fetch_one(&state.pool)
22  .await;
23
24  match q {
25      Ok(user) => {
26          // Validarea parolei
27          let password = Argon2::default()
28              .verify_password(
29                  data.password.as_bytes(),
30                  &PasswordHash::new(&user.password).unwrap(),
31              )
32              .is_ok();
33          if password {
34              let mut token = user.token;
35              if token.is_none() {
36                  token = update_token(&state.pool, user.id).await;
37              }
38
39              let cookie = Cookie::build(("SESSION", token.unwrap().
40              to_string()))
41                  .path("/")
42                  .max_age(Duration::days(2))
43                  .build();
44              let mut headers = HeaderMap::new();
```

```

43         headers.insert(SET_COOKIE, cookie.to_string().parse().
unwrap());
44         (headers, Redirect::to("/")).into_response()
45     } else {
46         Html(render_index(html!(
47             <div class="content">
48                 {login_form(Some(&data.username), None, Some("
Invalid password"))})
49             </div>
50             )))
51         .into_response()
52     }
53 }
54
55 Err(e) => {
56     // Daca usernameul nu exista se afiseaza pagina de
57     // autentificare cu eroare
58     if let sqlx::Error::RowNotFound = e {
59         return Html(render_index(html!(
60             <div class="content">
61                 {login_form(None, Some("Username dosen't
exist"), None)})
62             </div>
63             )))
64         .into_response();
65     }
66 }
67 }
68 }

```

3.2.3 Trimiterea jocurilor către server

La finalul jocului frontendul face un request POST către backend care conține datele despre joc, encodeate în format JSON:

- Rezultatul jocului: remiză, câștig sau pierdere
- Dificultatea la care a fost setat algoritmul
- Un array ce conține mutările care s-au făcut în joc, în ordine

Funcția din TypeScript care face requestul

```
1 send_game_to_server() {
2   let result = "Draw";
3   if (this.game.game_state() = GameState.Checkmate) {
4     if (this.game.side_to_move() = this.bot_color) {
5       result = "Win";
6     } else {
7       result = "Loss";
8     }
9   }
10
11   fetch("/api/submit_game", {
12     method: "POST",
13     headers: {
14       "Content-Type": "application/json",
15     },
16     body: JSON.stringify({
17       result: result,
18       moves: this.game.moves_server(),
19       difficulty: this.difficulty,
20     }),
21   });
22 }
```

În backend, sunt decodate datele, procesate și adăugate în baza de date. Dacă utilizatorul este autentificat se ține minte în baza de date, dacă nu se trece ca Guest.

```
1 async fn submit_game(
2   State(state): State<AppState>,
3   cookies: TypedHeader<Cookie>,
4   Json(data): Json<GameDataJson>,
5 ) {
6   let user = get_user_token(cookies);
7   let difficulty = match data.difficulty {
8     0 => "Easy",
9     1 => "Medium",
10    2 => "Hard",
11    _ => "-",
12  };
13
14  match user {
15    Some(id) => {
16      sqlx::query!(
```

```

17         "INSERT INTO games (player, moves, result, difficulty)
VALUES (?, ?, ?, ?)",
18         id,
19         data.moves,
20         data.result,
21         difficulty
22     )
23     .execute(&state.pool)
24     .await
25     .unwrap();
26 }
27 None => {
28     sqlx::query!(
29         "INSERT INTO games (moves, result, difficulty) VALUES
(? , ? , ?)",
30         data.moves,
31         data.result,
32         difficulty
33     )
34     .execute(&state.pool)
35     .await
36     .unwrap();
37 }
38 }
39 }

```

3.3 Librăria

Librăria de șah are două elemente principale folosite în frontend: structura *Game* și funcția *bot_move*.

Structura *Game* conține toate informațiile despre jocul curent și este definită astfel:

```
1 struct Game {  
2     board: Board,  
3     moves: Vec<Move>,  
4     fifty_move_rule: u8,  
5     game_state: GameState,  
6     board_history: Vec<Board>,  
7 }
```

- Câmpul *board* conține configurația curentă de piese
- Câmpul *moves* conține un vector cu mutările jucate până acum
- Câmpul *fifty_move_rule* numără câte mutări au trecut de la ultima mutare de pion sau captură (dacă ajunge la 50 jocul se declară remiză)
- Câmpul *game_state* memorează stadiul curent al jocului și poate lua una dintre următoarele valori:
 - InProgress
 - Checkmate
 - Stalemate (jucătorul curent nu poate face nicio mutare, dar nu este în șah. Jocul se declară remiză)
 - DrawByRepetition (dacă se repetă aceeași poziție a pieselor de 3 ori jocul se declară remiză)
 - DrawByFiftyMoveRule
 - DrawByInsufficientMaterial (dacă niciunul din jucători nu are destule piese pentru a da mat jocul se declară remiză)
- Câmpul *board_history* conține configurațiile pentru fiecare mutare. Este folosit pentru verificarea regulii de remiză prin repetiție și pentru a afișa mai ușor istoricul jocului în frontend

Tabla de șah este alcătuită din 64 de pătrate, același număr ca numărul de biți pe care îl folosesc procesoarele moderne. Astfel se poate asocia fiecărui pătrat un bit unic într-o variabilă pe 64 de biți (de exemplu *unsigned long long* în c++ sau *u64* în Rust), unde valoarea de 1 înseamnă că pătratul este ocupat de o piesă, iar 0 că este liber.

Nu este de ajuns, însă, să se memoreze toată tabla într-un singur număr, deoarece se pierde informația despre tipul fiecărei piese, deci se țin minte 7 numere pentru fiecare culoare (pentru pionii, cai, nebuni, ture, regine, rege și unul cu toate piesele combinate).

Această reprezentare se numește *BitBoards* și este eficientă din punct de vedere al memoriei, dar, mai important, din punct de vedere al timpului de executare, deoarece pentru generarea mutărilor sau obținerea informațiilor despre tablă se folosesc operații pe biți.

Așadar structura *Board* este definită astfel (tipul *BitBoard* este un număr pe 64 de biți):

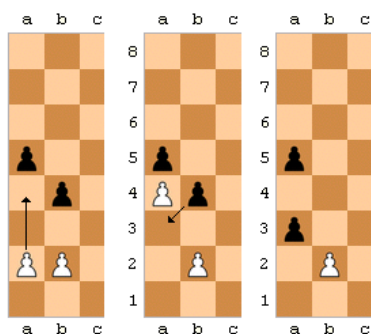
```

1 struct Board {
2     w_pawn: BitBoard,
3     w_knight: BitBoard,
4     w_bishop: BitBoard,
5     w_rook: BitBoard,
6     w_queen: BitBoard,
7     w_king: BitBoard,
8     b_pawn: BitBoard,
9     b_knight: BitBoard,
10    b_bishop: BitBoard,
11    b_rook: BitBoard,
12    b_queen: BitBoard,
13    b_king: BitBoard,
14
15    w_occ: BitBoard,
16    b_occ: BitBoard,
17    occ: BitBoard,
18
19    side_to_move: Color,
20    in_check: bool,
21
22    en_passant: Option<Square>,
23    can_castle: u8,
24 }

```

Pe lângă cele 14 numere precizate mai sus, structura mai are câteva câmpuri:

- Câmpul *occ* conține numărul care reprezintă toată tabla
- Câmpul *side_to_move* conține jucătorul curent (*White* sau *Black*)
- Câmpul *in_check* ține minte dacă jucătorul curent este în șah
- Câmpul *en_passant* ține minte dacă în ultima mutare un pion a avansat cu 2 pătrate și pătratul de pe care a plecat, pentru o regulă mai puțin cunoscută numita en passant (dacă un pion s-a mutat 2 pătrate și ajunge în dreptul unui pion inamic poate fi capturat de oponent **doar** pentru următoarea mutare)



- Câmpul *can_castle* ține minte folosind 2 biți pentru fiecare jucător dacă poate face rocada, câte un bit pentru fiecare direcție.

Funcția *bot_move* calculează cea mai bună mutare pentru un *Board* și o dificultate anume pe care le primește ca parametrii. Algoritmul de căutare a celei mai bune mutări este compus din 3 părți:

1. Generarea mutărilor legale
2. Evaluarea unei poziții
3. Căutarea celei mai bune mutări

3.3.1 Generarea mutărilor legale

Generarea mutărilor este o parte esențială, deci trebuie să fie cât mai rapidă, astfel pentru fiecare tip de piesă și fiecare pătrat se precalculează un tabel care conține un *BitBoard* cu valori de 1 în pătratele care sunt atacate de fiecare piesă pe pătratul respectiv.

Această metodă funcționează pentru pioni, cai și regi dar nu și pentru ture, nebuni și regine deoarece ele pot fi blocate de alte piese. Se observă că regina funcționează ca o tură combinată cu un nebun, deci nu avem nevoie să generăm nimic în plus.

Pentru ture și nebuni se folosește o metodă numită *Magic BitBoards* care constă în generarea unui *BitBoard* pentru fiecare aranjament piese care pot bloca piesa pentru care generăm. La prima vedere pare o metodă inefficientă, dar tura atacă 14 pătrate pe o tablă goală iar nebunul maxim 13. Aceste numere se pot reduce la 12, respectiv 9 observând că pătratele de la marginea tablei nu schimbă *BitBoardul* generat indiferent de starea de ocupație, ceea ce înseamnă că pentru fiecare pătrat trebuie să generăm $2^{12} + 2^9 = 4608$ de numere, un număr foarte mic.

Rezultatele generării sunt memorate într-o structură numită *Precalc*:

```
1 pub struct Precalc {
2     pub bishop_magic: Box<[[BitBoard; 512]; 64]>,
3     pub rook_magic: Box<[[BitBoard; 4096]; 64]>,
4     pub bishop: Box<[(BitBoard, u32); 64]>,
5     pub rook: Box<[(BitBoard, u32); 64]>,
6     pub pawns: Box<[[BitBoard; 2]; 64]>,
7     pub knight: Box<[BitBoard; 64]>,
8     pub king: Box<[BitBoard; 64]>,
9 }
```

3.3.2 Evaluarea unei poziții

Evaluarea unei poziții este destul de rudimentară, însă suficientă. Astfel fiecare piesă are o valoare de bază:

- pionul 100
- calul și nebunul 300
- tura 500
- regina 900
- regele 10000

De asemenea pentru fiecare pătrat de pe tablă există un bonus (pozitiv sau negativ) care îndrumă algoritmul să pună piesele pe pătrate în general bune.

Calculul scorului unei poziții se face astfel:

1. Se adună la scor suma dintre valorile pieselor jucătorului curent și bonusul pentru tipul de piesă și pătrat
2. Se scad din scor suma dintre valorile pieselor oponentului și bonusul pentru tipul de piesă și pătrat

```

1 fn evaluate(board: &Board) -> i32 {
2     let score = |c: Color| {
3         let mut s = 0;
4         for p in PIECE_KINDS.into_iter() {
5             let mut b = board.board(Piece::new(p, c)).0;
6             while b != 0 {
7                 let square = b.trailing_zeros() as usize;
8                 b &= b - 1;
9                 s += Self::piece_value(p);
10                s += if c == Color::White {
11                    Self::piece_square_value(p, square)
12                } else {
13                    Self::piece_square_value(p, Self::mirror_square(
14                        square))
15                };
16            }
17        }
18    };
19
20    let eval = score(Color::White) - score(Color::Black);
21    if board.side_to_move == Color::White {
22        eval
23    } else {
24        -eval
25    }
26 }
27
28
29 fn piece_value(piece_kind: PieceKind) -> i32 {
30     match piece_kind {
31         PieceKind::Pawn => 100,
32         PieceKind::Knight => 300,

```



```

33     PieceKind::Bishop => 300,
34     PieceKind::Rook   => 500,
35     PieceKind::Queen  => 900,
36     PieceKind::King   => 10000,
37 }
38 }
39
40 const PAWN_SCORE: [i32; 64] = [
41     0,  0,  0,  0,  0,  0,  0,  0,
42     0,  0,  0, -10, -10,  0,  0,  0,
43     0,  0,  0,  5,  5,  0,  0,  0,
44     5,  5, 10, 20, 20,  5,  5,  5,
45    10, 10, 10, 20, 20, 10, 10, 10,
46    20, 20, 20, 30, 30, 30, 20, 20,
47    30, 30, 30, 40, 40, 30, 30, 30,
48    90, 90, 90, 90, 90, 90, 90, 90,
49 ];
50
51 const KNIGHT_SCORE: [i32; 64] = [
52    -5, -10,  0,  0,  0,  0, -10, -5,
53    -5,  0,  0,  0,  0,  0,  0, -5,
54    -5,  5, 20, 10, 10, 20,  5, -5,
55    -5, 10, 20, 30, 30, 20, 10, -5,
56    -5, 10, 20, 30, 30, 20, 10, -5,
57    -5,  5, 20, 20, 20, 20,  5, -5,
58    -5,  0,  0, 10, 10,  0,  0, -5,
59    -5,  0,  0,  0,  0,  0,  0, -5,
60 ];
61
62 const BISHOP_SCORE: [i32; 64] = [
63     0,  0, -10,  0,  0, -10,  0,  0,
64     0, 30,  0,  0,  0,  0, 30,  0,
65     0, 10,  0,  0,  0,  0, 10,  0,
66     0,  0, 10, 20, 20, 10,  0,  0,
67     0,  0, 10, 20, 20, 10,  0,  0,
68     0,  0,  0, 10, 10,  0,  0,  0,
69     0,  0,  0,  0,  0,  0,  0,  0,
70     0,  0,  0,  0,  0,  0,  0,  0,
71 ];
72
73 const ROOK_SCORE: [i32; 64] = [
74     0,  0,  0, 20, 20,  0,  0,  0,
75     0,  0, 10, 20, 20, 10,  0,  0,
76     0,  0, 10, 20, 20, 10,  0,  0,
77     0,  0, 10, 20, 20, 10,  0,  0,
78     0,  0, 10, 20, 20, 10,  0,  0,
79     0,  0, 10, 20, 20, 10,  0,  0,
80    50, 50, 50, 50, 50, 50, 50, 50,
81    50, 50, 50, 50, 50, 50, 50, 50,
82 ];
83

```

```

84 const KING_SCORE: [i32; 64] = [
85     0,  0,  5,  0, -15,  0, 10,  0,
86     0,  5,  5, -5, -5,  0,  5,  0,
87     0,  0,  5, 10, 10,  5,  0,  0,
88     0,  5, 10, 20, 20, 10,  5,  0,
89     0,  5, 10, 20, 20, 10,  5,  0,
90     0,  5,  5, 10, 10,  5,  5,  0,
91     0,  0,  5,  5,  5,  5,  0,  0,
92     0,  0,  0,  0,  0,  0,  0,  0,
93 ];
94
95 fn piece_square_value(piece: PieceKind, square: usize) -> i32 {
96     match piece {
97         PieceKind::Pawn => Self::PAWN_SCORE[square],
98         PieceKind::Knight => Self::KNIGHT_SCORE[square],
99         PieceKind::Bishop => Self::BISHOP_SCORE[square],
100         PieceKind::Rook => Self::ROOK_SCORE[square],
101         PieceKind::Queen => 0,
102         PieceKind::King => Self::KING_SCORE[square],
103     }
104 }

```

3.3.3 Căutarea celei mai bune mutări

Șahul este un joc cu suma nulă, adică câștigul unui jucător este perfect echilibrat de pierderea celuilalt (dacă o mutare îi oferă unui jucător scorul de 200, pentru oponent mutarea va avea scorul -200), ceea ce permite folosirea unui algoritm de căutare numic *Negamax*.

Algoritmul constă în generarea unui arbore în care muchiile reprezintă câte o mutare legală, iar nodurile pozițiile după efectuarea mutărilor din rădăcină până la nod până la o adâncime fixă. Acest arbore se parcurge într-un mod similar cu algoritmul DFS, iar fiecărui nod i se atribuie un scor egal cu:

- Rezultatul funcției de evaluare, dacă nodul este o frunză
- Maximul dintre scorurile descendenților, cu semn opus (deoarece o mutare bună pentru oponent este o pierdere pentru jucătorul curent), dacă nodul nu este o frunză

La final vom ști din rădăcină ce mutare este optimă pentru jucătorul curent.

În mod general, acest algoritm se implementează astfel:

```

1 int negaMax( int depth ) {
2     if ( depth == 0 ) return evaluate();
3     int max = -INF;
4     for ( all moves ) {
5         score = -negaMax( depth - 1 );
6         if( score > max )

```

```

7         max = score;
8     }
9     return max;
10 }

```

În practică acest algoritm este foarte ineficient deoarece caută un număr foarte mare de noduri, și se poate optimiza prin nenumărate moduri. Funcția de căutare folosește următoarele optimizări:

- Alpha-beta pruning
- Tabele de transpoziții
- Move ordering
- Iterative deepening
- Principal Variation

De asemenea, pentru șah nu este de ajuns căutarea până la o adâncime fixă, trebuie continuată căutarea până la poziții unde nu există mutări care capturează piese.

De exemplu, în cazul în care căutarea ar ajunge în poziția de mai jos și se mai poate evalua o singură mutare în adâncime, mutarea regină la b6 ar fi considerată o mutare foarte bună deoarece capturează un nebun, însă la următoarea mutare regina poate fi capturată de pionul de la a7. Această îmbunătățire se numește *quiescence search*.



Alpha-beta pruning

Această optimizare este cea mai importantă și cea care permite implementarea tuturor restul optimizărilor și constă în adăugarea a 2 parametrii alpha și beta care reprezintă scorul minim pe care jucătorul curent îl poate obține, respectiv scorul maxim pe care oponentul îl poate obține. Folosind aceste 2 numere se pot reduce semnificativ numărul de noduri căutate, dar obținând garantat același rezultat.

După ce se calculează scorul unei mutări posibile se verifică dacă este mai mare sau egal decât beta. Dacă da atunci nu mai are sens căutarea acestui nod deoarece oponentul poate să facă o mutare care garantează un scor mai mare sau egal decât orice altă mutare găsită până acum, deci este o mutare proastă pentru jucătorul curent și se returnează beta.

Implementarea generală este asemănătoare cu cea de la *negamax*:

```
1 int negaMax(int depth, int alpha, int beta) {
2     if (depth == 0) return evaluate();
3     for (all moves) {
4         score = -negaMax(-beta, -alpha, depth-1);
5         if (score >= beta) {
6             return beta;
7         }
8         if (score >= alpha) {
9             alpha = score;
10        }
11    }
12    return alpha;
13 }
```

Move ordering

Optimizarea precedentă funcționează mai bine atunci când găsește o mutare bună mai repede, astfel se pot ordona mutările înainte de a fi parcurse. Ordonarea se poate face atribuind fiecărei mutări un scor general și apoi sortarea în ordine descrescătoare după scor.

O metodă simplă de atribuire a scorului general este

- Dacă mutarea este o captură: $10 * p2 - p1$, unde $p1$ este valoarea piesei cu care se capturează și $p2$ este valoarea piesei capturate
- 0, în caz contrar

De exemplu dacă avem mutările:

- Capturarea unei ture cu un cal
- Mutarea unui cal pe un spațiu gol
- Capturarea unei ture cu un pion

După sortare se vor căuta în ordinea:

- Capturarea unei ture cu un pion
- Capturarea unei ture cu un cal
- Mutarea unui cal pe un spațiu gol

Iterative deepening și principal variation

Iterative deepening este o tehnică de a gestiona timpul în care algoritmul găsește o mutare și constă în rularea căutarea de mai multe ori, începând de la adâncimea maximă 1 până la limita fixată, iar după fiecare căutare completă, dacă a trecut timpul alocat găsirii acestei mutări se ignoră rezultatul curent și se returnează cel precedent. (Pentru a opri căutarea curentă la timp se verifică periodic în funcția de *negamax* dacă a trecut timpul).

Doar această tehnică nu este o optimizare, ci din potrivă se caută aceleași noduri de mai multe ori. Optimizarea se face prin tehnica *principal variation* care constă în căutarea mutărilor care au dus la cel mai bun scor iterația trecută înaintea tuturor celorlate mutări (acestor mutări li se atribuie un scor mai mare decât celorlalte în funcția de sortare), deoarece cea mai bună mutare în căutarea până la adâncimea k este probabil o mutare bună și la căutarea $k+1$.

Tabele de transpoziții

Să considerăm poziția de mai jos. Există 4 serii distincte de mutări prin care se poate ajunge în acest punct (numite transpoziții) și după fiecare se vor evalua aceleași mutări, ceea ce este inefficient.

Astfel după ce am evaluat o poziție se memorează într-un tabel (reprezentat printr-un vector mare) adâncimea la care s-a făcut evaluarea, scorul rezultat și punctul din funcția de *alpha-beta* unde s-a memorat rezultatul (în if-ul cu beta, în if-ul cu alpha sau la final)

Acum, în funcția de *alpha-beta* înainte de for se verifică dacă în tabelul de transpoziții există o intrare cu poziția dată cu adâncimea mai mare sau egală decât adâncimea curentă și care să satisfacă limitele impuse de alpha și beta din căutarea curentă. În cazul în care există se returnează scorul ținut minte în tabel, altfel se continuă cu căutarea normală.

Pentru memorarea unei poziții într-un array trebuie generat un index aproape unic folosind *hashuri Zobrist* care funcționează astfel:

- Se generează mai multe numere la întâmplare (pe 64 de biți) care rămân aceleași pe parcursul căutării
 - Câte 12 pentru fiecare tip de piesă pentru fiecare pătrat (720 de numere)
 - 16 numere pentru fiecare stare de rocadă (sunt 4 rocade, deci 2^4 numere)
 - 8 numere pentru fiecare coloană pe care se poate face mutarea en passant
 - 1 număr pentru culoarea jucătorului
- Transformarea unei poziții într-un index se face astfel:
 1. Începem cu indexul egal cu 0

2. Pentru fiecare piesă de pe tablă se face xor între index și numărul generat pentru piesă și poziția ei pe tablă
3. Se face xor între index și starea de rocadă
4. Dacă se poate juca en passant, se face xor între index și coloană
5. Dacă jucătorul curent are piesele cu culoarea neagră se face xor între index și numărul pentru culoare
6. Se returnează indexul obținut

Indexul rezultat va fi un număr pe 64 de biți, posibil foarte mare, și nu poate fi folosit direct ca index, astfel tabelul de transpoziții va avea o mărime fixă, iar indexul în tabel se va calcula prin indexul generat modulo mărimea tabelului.

3.3.4 Implementarea algoritmului

Funcția *bot_move*

```

1 pub fn bot_move(board: Board, difficulty: Difficulty) -> Move {
2     let (depth, tt_size, max_time) = match difficulty {
3         Difficulty::Easy => (5, 10000000, 3000),
4         Difficulty::Medium => (7, 10000000, 3000),
5         Difficulty::Hard => (30, 10000000, 10000),
6     };
7
8     let mut bot = Bot::new(depth, tt_size, max_time);
9     let m = bot.make_move(board.0, difficulty == Difficulty::Hard);
10
11     return select_move(m, difficulty);
12 }

```

Structura *Bot*

```

1 pub struct Bot {
2     depth: i32,
3
4     transposition_table: TranspositionTable,
5     zobrist: Zobrist,
6
7     time: u64,
8     should_stop: bool,
9     start: Instant,
10
11     pv_table: Vec<Box<[Move]>>,
12     pv_len: Vec<usize>,
13 }

```

Funcția *make_move* definită pentru un bot returnează un array care poate să conțină:

- Un singur element: cea mai bună mutare, dacă este setată dificultatea grea

- Toate mutările legale din poziția inițială, sortate după scor, pentru celelalte dificultăți

```

1 pub fn make_move(&mut self, board: Board, hard_diff: bool) -> Box<([
  Move, i32])> {
2     if hard_diff {
3         let mut moves = vec![];
4         self.start = Instant::now();
5         for depth in 1..=self.depth {
6             let s = self.search(board.clone(), -500_000, 500_000, depth
, 0, true);
7             self.reached_depth = depth;
8             if self.should_stop {
9                 break;
10            }
11            moves = vec![(self.pv_table[0][0], s)];
12        }
13        moves.into()
14    } else {
15        let mut moves = vec![];
16        for depth in 1..=self.depth {
17            let mut new_moves = vec![];
18            let m = legal_moves(&board);
19            for m in m.iter() {
20                let mut board = board.clone();
21
22                board.make_move(m);
23                let score = -self.search(board, -500_000, 500_000,
depth, 1, true);
24
25                new_moves.push((m.clone(), score));
26            }
27
28            self.reached_depth = depth;
29            if self.should_stop {
30                break;
31            }
32            moves = new_moves;
33        }
34        moves.sort_by(|a, b| b.1.cmp(&a.1));
35        moves.into()
36    }
37 }

```

Celelalte funcții folosite în căutare, care implementează algoritmi și tehnicile prezentate anterior:

```

1 pub fn search(
2     &mut self,
3     board: Board,
4     mut alpha: i32,
5     beta: i32,
6     depth: i32,

```

```

7     ply: i32,
8     mut follow_pv: bool,
9 ) -> i32 {
10    self.nodes_searched += 1;
11    if self.nodes_searched % 5000 == 0 {
12        self.check_time();
13        if self.should_stop {
14            return 0;
15        }
16    }
17
18    self.pv_len[ply as usize] = 1;
19
20    if depth == 0 {
21        return self.quiescence(board, alpha, beta, ply);
22    }
23
24    let hash = self.zobrist.hash(&board);
25    if let Some(score) = self.transposition_table.probe(hash, depth,
alpha, beta) {
26        return score;
27    }
28
29    let moves = legal_moves(&board);
30    if follow_pv {
31        follow_pv = false;
32        for m in moves.iter() {
33            if m == &self.pv_table[ply as usize][0] {
34                follow_pv = true;
35                break;
36            }
37        }
38    }
39
40    let moves = self.sorted_moves(ply, &board, follow_pv);
41    if moves.is_empty() {
42        return if board.in_check { -490_000 + ply } else { 0 };
43    }
44
45    let mut tt_entry_kind = TranspositionKind::Alpha;
46
47    let next_depth = if board.in_check { depth } else { depth - 1 };
48    for m in moves.iter() {
49        let mut b = board.clone();
50        b.make_move(m);
51        let score = -self.search(b, -beta, -alpha, next_depth, ply + 1,
follow_pv);
52
53        if self.should_stop {
54            return 0;
55        }

```



```

56         if score >= beta {
57             self.transposition_table
58                 .store(hash, depth, beta, TranspositionKind::Beta, *m);
59             return beta;
60         }
61
62         if score > alpha {
63             tt_entry_kind = TranspositionKind::Exact;
64             alpha = score;
65
66             let p = ply as usize;
67             self.pv_table[p][0] = *m;
68             for i in 0..self.pv_table[p + 1].len() {
69                 self.pv_table[p][i + 1] = self.pv_table[p + 1][i];
70             }
71             self.pv_len[p] = self.pv_len[p + 1] + 1;
72         }
73     }
74 }
75
76 self.transposition_table.store(
77     hash,
78     depth,
79     alpha,
80     tt_entry_kind,
81     self.pv_table[ply as usize][0],
82 );
83
84 alpha
85 }
86
87 fn check_time(&mut self) {
88     let ms = Instant::now() - self.start;
89     let ms = ms.as_millis() as u64;
90     if ms >= self.time {
91         self.should_stop = true;
92     }
93 }
94
95 fn quiescence(&mut self, board: Board, mut alpha: i32, beta: i32, ply:
i32) -> i32 {
96     if self.nodes_searched % 5000 == 0 {
97         self.check_time();
98         if self.should_stop {
99             return 0;
100         }
101     }
102
103     let score = Self::evaluate(&board);
104     if score >= beta {
105         return beta;

```

```

106     }
107     alpha = alpha.max(score);
108
109     let moves = self.sorted_moves(ply, &board, false);
110     for m in moves.into_iter().filter(|m| m.capture()) {
111         let mut b = board.clone();
112         b.make_move(m);
113         let score = -self.quiescence(b, -beta, -alpha, ply + 1);
114
115         if self.should_stop {
116             return 0;
117         }
118
119         if score >= beta {
120             return beta;
121         }
122         alpha = alpha.max(score);
123     }
124
125     alpha
126 }
127
128 fn move_score(&self, ply: i32, m: &Move, board: &Board, score_pv: bool)
129     -> i32 {
130     if score_pv && m == &self.pv_table[ply as usize][0] {
131         return 10_000;
132     }
133
134     if matches!(m.special(), Some(SpecialMove::EnPassant)) {
135         return Self::capture_value(PieceKind::Pawn, PieceKind::Pawn);
136     }
137
138     if let Some(p) = board.piece(m.to().idx()) {
139         Self::capture_value(m.piece().kind, p.kind)
140     } else {
141         0
142     }
143 }
144
145 fn sorted_moves(&self, ply: i32, board: &Board, score_pv: bool) -> Box<
146     [Move]> {
147     let moves = legal_moves(board);
148     let mut moves = moves
149         .into_iter()
150         .map(|m| (m, self.move_score(ply, m, board, score_pv)))
151         .collect::<Vec<_>>();
152     moves.sort_by(|a, b| b.1.cmp(&a.1));
153     moves.into_iter().map(|(m, _)| *m).collect()
154 }
155
156 fn capture_value(p1: PieceKind, p2: PieceKind) -> i32 {

```

```

155 10 * Self::piece_value(p2) - Self::piece_value(p1)
156 }

```

Funcția *select_move* folosită în *bot_move* selectează care dintre mutările returnate de structura *bot* să fie jucată în funcție de dificultate:

- Pentru ușor și mediu:
 1. În cazul în care prima mutare are scorul foarte mare (algoritmul a găsit o serie de mutări care duc la mat forțat) este aleasă aceasta
 2. Altfel pentru fiecare mutare este calculat diferența dintre scorul acesteia și scorul primei mutări
 3. Se alege la întâmplare, cu probabilitățile setate pentru fiecare dificultate care să fie diferența maximă de scor dintre mutarea care va fi jucată și cea mai bună mutare, apoi se alege una dintre mutările care nu depășește diferența

Astfel pe aceste dificultăți se vor juca intenționat mutări mai slabe pe lângă limitele mai restrânse de timp și adâncime.

- Pentru greu este returnată o singură mutare care se returnează mai departe

```

1 fn select_move(moves: Box<[(Move, i32)]>, difficulty: Difficulty) ->
  Move {
2     match difficulty {
3         Difficulty::Easy => {
4             if moves[0].1 >= 400_000 {
5                 return moves[0].0.into();
6             }
7
8             let score_brackets = [50, 100, 200, 500, 2000];
9             let probabilites = [0.35, 0.25, 0.2, 0.15, 0.05];
10
11             let moves_per_bracketed: Vec<_> = score_brackets
12                 .iter()
13                 .map(|b| {
14                     moves
15                         .iter()
16                         .filter(|(_, s)| moves[0].1 - *s <= *b)
17                         .collect::<Vec<_>>()
18                 })
19                 .collect();
20
21             loop {
22                 let mut r: f64 = rand::thread_rng().gen();
23                 console_log!("Generated {r}");
24                 let mut bracket = 0;
25                 for i in 0..probabilites.len() {
26                     r -= probabilites[i];
27                     bracket = i;
28                     if r <= 0. {
29                         break;
30                     }
31                 }
32             }
33         }
34     }
35 }

```

```

31         }
32
33         if !moves_per_bracked[bracket].is_empty() {
34             let idx = rand::thread_rng().gen_range(0..
moves_per_bracked[bracket].len());
35             return moves[idx].0;
36         }
37     }
38 }
39 Difficulty::Medium => {
40     if moves[0].1 >= 400_000 {
41         return moves[0].0.into();
42     }
43
44     let score_brackets = [20, 50, 100, 200, 400];
45     let probabilites = [0.5, 0.3, 0.1, 0.08, 0.02];
46
47     // ...
48     // Acelasi cod ca la dificultatea easy
49 }
50 Difficulty::Hard => moves[0].0;
51 }
52 }

```

4 Cerințe minime de sistem

Hardware

- Procesor(CPU) 1GHz+
- Memorie(RAM) 4GB+

Software

- Sistem de operare: Windows 7+, Linux, MacOS 18+, Android 7+, IOS 8+
- Browser: Chrome 57+, Safari 11+, Firefox 52+

5 Programe utilizate

- Neovim

6 Bibliografie

- **Librării externe utilizate pentru frontend**

- Lit <https://lit.dev/>

- **Librării externe utilizate pentru backend**

Server HTTP

- axum <https://github.com/tokio-rs/axum>
- tokio <https://tokio.rs/>

Client SQL

- sqlx <https://github.com/launchbadge/sqlx>

Generator de HTML

- html-to-string-macro <https://crates.io/crates/html-to-string-macro>

- **Librării externe utilizate pentru librăria de şah**

- js-sys <https://crates.io/crates/js-sys>
- wasm-bindgen <https://github.com/rustwasm/wasm-bindgen>
- web-sys <https://crates.io/crates/web-sys>

- **Resurse**

- <https://www.chessprogramming.org/>