

UNIVERSITATEA BABEŞ-BOLYAI CLUJ-NAPOCA
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
SPECIALIZAREA INFORMATICĂ

LUCRARE DE LICENȚĂ

Minimax: Principii, concepte și aplicații

Conducător Științific

Lector Univ. Dr. CZIBULA Istvan

Absolvent
Dumitru Vlad Ștefan

2024

Abstract

This paper emphasizes the importance of learning a chess opening from the very beginning of a game. It acknowledges all the challenges of learning this complex game and aims to overcome these difficulties by introducing the Chess Openings platform. This user-oriented platform offers accessible courses that cover all the necessary aspects of learning and practicing chess openings, providing real game situation examples.

The Chess Openings platform stands out particularly for its implementation of a chess engine based on the MiniMax algorithm with Alpha-Beta Pruning. This implementation allows users to test their knowledge gained from the courses, thereby developing their skills and preparing them for confrontations with other players. The paper also explores the implementation of the MiniMax algorithm, comparing it with other fundamental algorithms for developing chess engines. By integrating the MiniMax algorithm into the Chess Openings application, it demonstrates its practical use, highlighting its simplicity and efficiency.

The research aims to contribute to chess education by offering an innovative learning environment through the Chess Openings application while advancing the field of chess algorithm development.

This paper is structured into five chapters, covering: the essential principles of chess, other algorithms for developing chess engines, detailed information about the MiniMax algorithm, the application development process, and the conclusions drawn from this development.

This paper is the result of my work, addressing the mentioned research and developing the Chess Openings application to enhance the chess learning process.

Cuprins

Capitolul 1 - Introducere.....	1
Capitolul 2 - Principii de șah.....	3
2.1 Tabla de șah.....	3
2.2 Piese.....	4
2.3 Notarea.....	6
2.4 Moduri în care se poate termina un meci.....	7
Capitolul 3 - Motoare de șah.....	9
3.1 Motoare de șah deja existente.....	9
3.2 Algoritmi folosiți pentru dezvoltarea motoarelor de sah.....	9
3.2.1 Monte Carlo Tree Search (MCTS).....	10
3.2.2 Principal Variation Search (PVS).....	11
3.2.3 NegaScout (Null Move Pruning).....	12
Capitolul 4 - Algoritmul Minimax.....	14
4.1 Introducerea algoritmului.....	14
4.2 Alpha beta pruning.....	15
4.3 Funcția euristică.....	16
4.4 Detalii de implementare.....	17
4.5 Minimax în jocul de șah.....	18
4.6 Eficiența algoritmului.....	18
4.7 Avantaje.....	18
4.8 Limitări.....	19
Capitolul 5 - Studiu de caz.....	20
5.1 Introducere Chess Openings.....	20
5.1.1 Actori.....	20
5.1.2 Cazuri de utilizare.....	21
5.1.3 Diagrama de clase.....	22

5.1.4 Aplicații similare.....	25
5.2 Tehnologii.....	25
5.2.1 Server.....	25
5.2.1.1 Java.....	25
5.2.1.2 Versiuni de Java.....	26
5.2.1.3 Spring.....	28
5.2.1.4 Spring Data.....	29
5.2.1.5 Spring Boot.....	30
5.2.1.6 JUnit Testing.....	30
5.2.2 Client.....	31
5.2.2.1 HTML.....	31
5.2.2.2 CSS.....	31
5.2.2.3 JavaScript.....	32
5.3 Acces și salvare de date.....	32
5.3.1 Bază de date.....	32
5.3.2 Acces la date.....	34
5.4 Ghid utilizare aplicație.....	35
5.5 Arhitectura și design patterns.....	41
5.5.1 Şablonul de proiectare MVC.....	42
5.5.2 Şablonul de proiectare Factory.....	42
5.5.3 Şablonul de proiectare Combinator.....	43
5.6 Testare.....	44
5.7 Docker.....	46
5.7.1 Dockherizarea aplicației.....	48
Capitolul 6 - Concluzii.....	51
Bibliografie.....	53

Capitolul 1 - Introducere

Într-o lume în care jocul de șah, devine din ce în ce mai popular, învățare acestuia a devenit o adevărată provocare dacă nu ai resursele necesare, sau chiar timpul de a studia în detaliu fiecare combinatie și mutare ca un adevărat grandmaster. Învățarea jocului de șah, este foarte importantă și benefică pentru o persoană, deoarece dezvolta abilități cognitive și oferă avantaje pe termen lung. Șahul stimulează gândirea critică și creativă, antrenând mintea să analizeze situațiile complexe și să anticipateze posibile consecințe. De asemenea îmbunătășește capacitatea de rezolvare a problemelor și abilitățile de planificare strategică, deoarece jucătorii trebuie să gândească mai multe mutări înainte și să-și adapteze strategiile în funcție de acțiunile adversarului. Astfel, învățarea șahului contribuie semnificativ la dezvoltarea personală și intelectuală a unei persoane.

Pentru asta am creat aplicația Chess Openings care este aplicația perfectă pentru pasionații de șah, indiferent dacă sunt începători curioși să învete sau jucători experimentați dornici să-și perfecționeze abilitățile. Această aplicație inovatoare combină instruirea în deschiderile de șah cu un motor de joc implementat folosind algoritmul MiniMax, oferind o experiență captivantă și educativă.

Cu ajutorul Chess Openings, puteți explora o gamă largă de deschideri de șah, de la cele clasice la cele mai moderne și complexe. Fie că doriți să vă îmbunătățiți jocul în deschidere sau să învățați strategii avansate, aplicația vă pune la dispoziție resurse bogate și instrucțiuni detaliate pentru a vă ghida în această călătorie.

Unul dintre aspectele distinctive ale Chess Openings este motorul său de joc implementat cu algoritmul MiniMax. Acest algoritm, folosit în inteligența artificială, analizează mișările posibile într-un joc și își optimizează strategia pentru a obține cel mai bun rezultat posibil, luând în considerare atât mutările proprii, cât și cele ale adversarului. Astfel, utilizatorii pot juca împotriva unui adversar virtual antrenat, care oferă provocări realiste și oportunități de învățare continuă.

Indiferent dacă sunteți la început de drum în lumea șahului sau sunteți jucători avansați în căutare de noi provocări, Chess Openings vă oferă resursele și instrumentele necesare pentru a vă dezvolta abilitățile și a vă bucura de această minunată artă a gândirii strategice.

Utilizarea instrumentelor IA generative

În timpul elaborării acestei lucrări, am folosit SciSpace pentru a găsi mai ușor documentații despre tema aleasă, și de asemenea ChatGbt pentru a refraza limbajul scris de mine, într-unul mai profesional. După utilizarea acestui serviciu, am revizuit și editat conținutul generat și îmi asum întreaga responsabilitate pentru conținutul lucrării.

Capitolul 2 - Principii de șah

Şahul este unul dintre cele mai populare jocuri de strategie din lume, cu o complexitate fascinantă și o istorie bogată. Originar din India, șahul a evoluat de-a lungul timpului, devenind un simbol al inteligenței și al capacitatei strategice. Este un joc care se joacă în 2 persoane, amândoi având câte 16 piese, încercând să-și doboare regele advers. Cu un set complex de reguli și tactici, șahul încurajează gândirea strategică, anticiparea mișcărilor adversarului și luarea deciziilor în funcție de poziționarea pieselor pe tabla. Este un joc captivant, care a inspirat generații întregi de jucatori, de la amatori la mari maeștri, să exploreze și să îmbunătățească continuu abilitățile lor strategice și tactice.

2.1 Tabla de șah

O tablă de șah este compusă din 64 de pătrățele, dispuse într-o matrice de 8 pe 8. Aceste pătrățele sunt colorate alternativ în alb și negru. O tablă are de asemenea coordonate, linile verticale sunt numerotate de la a la h, iar cele orizontale de la 1 la 8, asta însemnând că fiecare căsuță de pe tabla de șah, poate fi identificată, combinând numărul liniei cu cel al coloanei. De exemplu în imaginea 2.1, căsuța selectată se poate identifica ca fiind e4, deoarece se află pe linia 4 și coloana e, adică coloana 5, astfel devenind căsuța e4. Regula este ca tot timpul coloana se pune prima, și după linia [LR23].

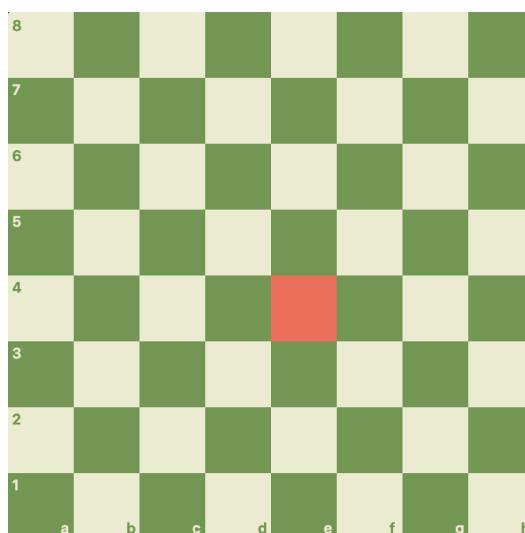


Figura 2.1.1 Tabla de șah cu căsuța e4.[Tce4]

La început, după adaugarea pieselor, alb începe de pe liniile 1 și 2, iar negru de pe liniile 7 și 8. Fiecare culoare are câte 16 piese, adică în total 32 de piese care se află pe tabla. Piesele trebuie aranjate în felul următor. Fiecare jucător are câte o linie de pioni, cu o linie de piese majore în spatele lor. Piesele majore trebuie aranjate în felul următor, de la stanga la dreapta: o tură, un cal, un nebun, o regină, un rege, un nebun, un cal și o tură. În timpul meciului, jucătorii mută pe rând, începând cu jucătorul alb [LR23].



Figura 2.1.2 Poziția de start

2.2 Piese

Regina, este cea mai valoroasă piesă de șah, valorând 9 puncte. Fiecare jucător are o singură regină, care poate să se mute sus, jos, stânga, dreapta și pe diagonală, și se poate muta câte căsuțe se dorește, dacă acele căsuțe nu sunt blocate de alte piese. Regina nu poate sări peste alte piese, chiar dacă acele piese sunt de aceeași culoare. Numai ea se poate muta în toate aceste direcții, putând să se mute pe o tablă goală, un număr maxim de 27 de căsuțe, pormind de la o anumită poziție [LR23].

Tura, este a doua cea mai valoroasă piesă, valorând 5 puncte. Fiecare jucător începe având 2 ture pe tablă așezate în colțuri opuse pe aceeași linie. Tura se poate muta sus, stânga, dreapta și jos,

Capitolul 2 - Principii de șah

câte căsuțe se dorește, având aceleași restricții ca și regina. Pe o tablă goală, aceasta se poate muta un număr maxim de 14 căsuțe, pornind de la o anumită poziție [LR23].

Nebunul, valorează 3 puncte, fiecare jucător începând cu câte 2 nebuni, unul care se deplasează pe căsuțele negre, iar altul care se deplasează pe căsuțele albe. Nebunul se poate muta numai pe diagonală, câte casute dorește. Acesta se poate muta un număr maxim de 13 casute, pornind de la o anumită poziție [LR23].

Calul, valorează de asemenea 3 puncte, fiecare jucător începe cu câte 2 cai. Calul se mută într-un mod unic, acesta fiind singura piesă care se poate muta așa pe tabla, acesta se mută în forma de L pe tablă, putând sări peste piese, indiferent că este piesă adversă sau piesă de aceeași culoare. Deoarece este o piesă care poate sări peste alte piese, este singura, care la începutul meciului se poate muta, în afară de pioni. Pe o tabla goală, calul se poate muta un număr maxim de 8 căsuțe [LR23].

Pionul, este valorat la un punct, iar fiecare jucător începe cu câte 8 pioni. Acestea au 5 reguli asociate. Pionii se pot muta numai în față, câte o căsuță, dar dacă acesta se află pe linia de început, se poate muta câte 2 căsuțe. Un pion de asemenea se poate muta o căsuță pe diagonală în față, dacă pe acea căsuță se află o piesă adversă, astfel capturând acea piesă, astăzintă că pionul nu se poate muta pe diagonală dacă acolo se află o piesă de aceeași culoare, sau dacă căsuța este goală. O regulă foarte importantă în șah, care face pionul o piesă foarte valoros este faptul că, atunci când pionul ajunge pe ultima linie, adică linia pe care începe regele advers, acesta se va schimba într-o alta piesă la alegere, între regină, tură, nebun sau cal. De asemenea pionul mai are o regula, care se întâlnește cel mai rar, numită "en-passant", adică dacă un pion se află pe linia 5, dacă este de culoare albă, sau 4, dacă este de culoare neagră, iar pionul advers, fie cel din stanga sau din dreapta pionului, se află pe linia de start, și acesta se mută 2 căsuțe, pionul nostru, cel de pe linia 4 sau 5, poate să captureze pionul advers, și se mută o căsuță în față pe diagonală, adică pe scurt se poate captureaza pionul advers ca și cum s-ar fi mutat o singură căsuță [LR23].

Capitolul 2 - Principii de șah

Regele, cea mai importantă piesă de șah, acesta nu are o valoare, deoarece nu poate fi capturat, și trebuie protejat cu orice preț. Fiecare jucător începe cu un singur rege. Jocul de șah este câștigat sau pierdut în funcție de mutările posibile ale regelui. Un rege se poate muta câte o singură căsuță în toate direcțiile, față, spate, stânga, dreapta și pe diagonală. Dar, dacă regele se află pe poziția sa de start, și nu a fost mutat deloc, iar și o tură, oricare din cele 2, se află de asemenea pe pozițiile lor de start și nu au fost mutate de loc, se poate efectua o rocadă, adică regele și tura, se mută împreună, adică regele se mută 2 căsuțe în stânga sau dreapta, depinzând în ce direcție se face rocadă, iar tura se va muta în căsuța din stânga regelui, dacă se face rocadă în partea dreapta (rocada mica), sau în căsuța din dreapta regelui, dacă se face rocadă în partea stanga (rocada mare). Există cîteva restricții pentru efectuarea unei rocade, trebuie să nu existe nicio piesă între rege și tură, regele și tura, nu au fost mutate deloc, iar regele nu este în șah. De asemenea regele poate să fie pus în șah, adică dacă o piesă se poate muta pe căsuța regelui, în acest moment, jucătorul al cărui rege este pus în șah, nu poate muta nici o piesă, decât cele care scot regele din șah, adică opresc piesa adversă sa vada regele. [LR23]

2.3 Notarea

Notarea de șah, descrie cum se mută piesele într-un meci, astfel toate mutările sunt salvate și descrise. În tabelul 2.3 se poate observa cum diferite piese sunt notate.

Pion (Pawn)	Pentru pioni, numai căsuța în care se mută este notată, de exemplu dacă un pion a fost mutat pe e5, înseamnă ca pionul a fost mutat de pe e4, dacă acesta este pion alb. Dacă un pion capuseaza o piesă, se include de pe ce coloana a venit de exemplu dacă pionul de pe e4 capuseaza o piesă pe d5, se notează exd5
Nebun (Bishop)	B
Cal (Knight)	N

Capitolul 2 - Principii de șah

Tură (Rook)	R
Regina (Queen)	Q
Rege (King)	K

Tabelul 2.3 Notarea pieselor de șah

Notariile vin din limba engleza, se poate observa ca pentru cal se notează cu N și pentru rege cu K, asta pentru că să se deosebească piesele. Pentru notarea unei poziții pentru o piesă în afară de pion se procedează astfel, se pune litera corespunzătoare piesei, și căsuța în care a ajuns, de exemplu dacă mutăm nebunul de pe căsuța e4 pe căsuța d5, se notează Bd5. Dacă o piesă este capturată se pune un x înaintea căsuței, dacă același nebun se mută pe d5 și capturează o piesă se notează Bxd5. Dacă o mutare pune regele în șah se notează cu + la final, dacă de exemplu același nebun da șah la rege mutându-se pe căsuța d5 se va nota Bd5+, iar dacă în același timp se capturează o piesă se notează așa Bxd5+. Dacă e șah mat se notează Bd5#. Când un pion promovează se va insera un = și litera piesei în care se schimbă, de exemplu dacă pionul promovează de pe g7 pe g8 și se transformă într-o regină, se va nota g8=Q. Dacă se face rocadă mică, se va nota O-O, iar dacă se face rocadă mare se va nota O-O-O. Mai este o regulă, dacă 2 piese se pot muta pe aceeași căsuță, se va inseră și coloana de pe care pleacă piesa respectivă, de exemplu dacă avem 2 cai albi, unul pe casuta d4 și celalat pe căsuța f4, amândoi se pot muta pe e6, iar în meci se mută calul de pe d4, se va nota Kde6. [LR23]

2.4 Moduri în care se poate termina un meci

Un meci de șah, se poate avea 3 rezultate egal, victorie sau înfrângere. Un jucător poate câștiga în 2 moduri fie prin șah mat, sau dacă oponentul renunță la meci. Șah mat este atunci când regele advers este pus în șah, și nu poate bloca acel șah sau nu se poate muta în nici o căsuță. Adică pe scurt, oriunde se mută regele acesta poate fi capturat de o piesă adversă.

Egal se poate face în 4 moduri, stalemate, egal prin repetiție, regula de 50 de mutări, sau egal prin comun acord. Stalemate înseamnă ca adversarul nu are nici-o mutare posibilă, dar nu este pus în șah. Egal prin repetiție înseamnă ca se repeta aceeași poziție de joc de 3 ori în același meci. Adică dacă amândoi jucătorii repeta aceeași mutare de 3 ori în același meci jocul se va

Capitolul 2 - Principii de șah

termina la egal. Regula de 50 de mutări, înseamnă că dacă într-un interval de 50 de mutări consecutive, nu s-a făcut nicio capturare sau nu s-a mutat niciun pion, jocul se va termina la egal. Egalul prin comun acord înseamnă ca amandoi jucătorii decid că jocul este egal, astfel terminandu-se [LR23].

Capitolul 3 - Motoare de șah

3.1 Motoare de sah deja existente

Există diferite motoare de șah, fiecare folosind algoritmi și funcționalități diferite. Aceste motoare analizează pozițiile de șah pentru a determina cele mai puternice mișcări, folosind adesea algoritmul minimax, cu îmbunătățiri precum tăierea Alpha Beta. Unele motoare, cum ar fi Dev Zero, sunt proiectate pentru accesibilitatea universală, permitând diferite niveluri de calificare să joace concomitent. În special, Stockfish și Leela Chess Zero sunt motoare proeminente cu metode de joc distințe. Disponibilitatea motoarelor de șah pentru smartphone-uri și tablete le-a sporit accesibilitatea, permitând utilizatorilor să joace împotriva diferitelor motoare, fără să fie nevoie să învețe interfața unică a fiecărui. Cercetătorii au analizat chiar milioane de jocuri de șah, folosind motoare precum Stockfish, subliniind rolul acestora în aplicații precum detectarea înșelăciunii și evaluarea abilităților [AKMP18, NUGP21]. Probabil, cel mai performant motor de șah, în momentul actual este Stockfish. Acesta este un program open source, care utilizează algoritmi avansați de inteligență artificială și tehnici de calcul paralel pentru a analiza pozițiile de șah găsite și a găsi cele mai bune mișcări posibile. Este recunoscut pentru puterea să de calcul excepțională, fiind deseori folosit în competiții de șah de nivel înalt și în analiza partidelor de către jucătorii profesionisti și amatorii.

3.2 Algoritmi folosite pentru dezvoltarea motoarelor de sah

Diverse algoritmi sunt utilizati în dezvoltarea motoarelor de șah. Algoritmul Minimax, combinat cu tăierea alpha beta, este utilizat în mod obișnuit pentru a reduce sarcina de calcul și a spori eficiența. În plus, estimatorii bazați pe învățarea automată sunt integrați cu tehnici de inteligență artificială pentru a construi sisteme avansate de AI de șah, obținând o precizie ridicată în prezicerea „mișcărilor bune”. Unele motoare precum Dev Zero utilizează algoritmul MiniMax și tăierea alpha beta împreună cu funcții euristice bazate pe parametri precum mobilitatea pieselor și evaluarea plăcii pentru a selecta mișcări optime.[MDCB23, Ceba23].

3.2.1 Monte Carlo Tree Search (MCTS)

Monte Carlo Tree Search (MCTS) este un algoritm puternic utilizat în diferite scenarii de luare a deciziilor. Acesta permite abordări inovatoare de autoînvățare, care integrează o rețea neuronală pentru a îmbunătății eficiența problemelor. În plus MTCS constrâns (C-MTSC) a fost dezvoltat pentru a gestiona sarcinile de luare a deciziilor cu constrangeri dure, asigurand îndeplinirea constrângerilor de costuri, obținând în același timp recompense și eficiente mai mari [WLY23, DPK23].

Algoritmul constă în 4 pași principali (Selecția, Expansiunea, Roll-out și Back-propagation). Acești pași se repetă iterativ până o condiție de oprire apare. În timpul selecției, plecând de la rădăcină, arborele este descendant. Drumul de la rădăcină la o frunză, este determinat folosind o politică de selecție care sugerează cele mai bune mișcări. Apoi, de la frunza selectată, un nod este adăugat în arbore (starea de extensie). Ramura care este expandată este selectată aleator, în timp ce evită repetiția a nodurilor existente. Al treilea pas, roll-out, este executat, în timp ce, noduri sunt generate și sunt temporal incorporate în soluție, dar nu sunt permanent adăugate la arbore. La final, rezultatul de la roll-out este propagat (pasul back-propagation) de la nodul selectat pentru expansiune, înapoi la rădăcina arborelui. În acest mod, agentul este informat de calitatea iterăției de căutare, pe care a performat-o. În figura 3.2.1, se poate observa pseudocodul de la MTCS.

Monte Carlo Tree Search Algorithm

Input:

- PD: problem's dimension
- FG: fitness goal
- RL: roll-out length

Output:

- the best solution found, n^*
- the maximum performance level reached, f
- the total number of roll-outs performed, t

```

1  create root node  $n_r$  and add it to the tree
2  make  $n_r$  actual node n
3   $n^* = \text{null}$ 
4   $f = 0$ 
5   $t = 0$ 
6  while(move_available(n))
    // descend the tree to a Leaf
7      while(!is_leaf(n))
8          n = best_child(n)
9      end
10     n = add_random_child_node(n)
11     RL = rand(PD, n)
12     f,  $n' = \text{roll\_out}(n, RL)$ 
13     t++
14     back_propagate(n, f)
15     if(f > FG)
16          $n^* = n'$ 
17         break
18     end
19     n =  $n_r$ 
20 end
21 return  $n^*, f, t$ 

```

Figura 3.2.1 MCTS' pseudo-code [Mtcs]

3.2.2 Principal Variation Search (PVS)

Algoritmul Principal Variation Search (PVS) este o tehnică metaeuristică hibridă utilizată într-o varietate de provocări de optimizare. PVS integrează un algoritm local de căutare pentru aranjarea orizontală cu o metodă precisă pentru poziționarea verticală, îmbunătățind constant soluțiile până la atingerea optimității sau a unei constrângeri de timp predefinite. În domeniul șahului Amazon, PVS este îmbogățit prin includerea tabelelor de transpunere, euristică istorică și procesare paralelă, rezultând o eficiență sporită de cădere și o utilizare sporită a resurselor CPU,

permisând astfel niveluri de explorare mai profunde și strategii de joc îmbunătățite. PVS oferă o metodologie strategică care armonizează exploatarea locală cu explorarea globală, oferind rezultate optime, de calitate superioară. În general, PVS prezintă eficacitatea competitivă în ceea ce privește precizia și robustețea optimizării, depășind algoritmii metaeuristici alternativi în sarcinile legate de identificarea parametrilor. [LJT21, LPV20]

3.2.3 NegaScout (Null Move Pruning)

NegaScout este o îmbunătățire a algoritmului Alpha beta pruning, care incorporează tăierea mișcării nenele. Tăiera mutării nulă este o tehnică utilizată pentru a îmbunătăți eficiența algoritmilor de căutare a arborelui de joc, presupunând că omiterea rândului unui jucător nu ar schimba semnificativ evaluarea poziției. S-a demonstrat că aceste modificări depășesc metodele standard, în special în pozițiile zugzwang, unde NegaScout rămâne eficient. Implementarea acestor îmbunătățiri în programele existente este relativ simplă, necesitând doar ajustări minore ale codului. [VST20, RCH13]

După cum se observă în figura 3.2.3, algoritmul NegaScout își desfășoară funcționarea în câțiva pași principali, fiecare având un rol specific în procesul său de căutare și evaluare a pozițiilor într-un joc. Primul pas constă în căutarea recursivă a arborelui de joc folosind tehnica negamax, care presupune explorarea tuturor mutărilor posibile și estimarea valorii acestora. Apoi, tăiera cu mișcare nulă este implementată pentru a elimina ramurile specifice din arbore care nu contribuie la căutarea optimă. În acest pas, se consideră faptul că unele poziții sunt atât de bune încât adversarul nu ar face o mutare nulă, ceea ce permite să fie eliminată explorarea suplimentară a acestor poziții. Al treilea pas este reprezentat de fereastra de căutare, care restricționează intervalul de valori pentru a optimiza căutarea. Prin folosirea acestei ferestre, se poate realiza o evaluare mai eficientă a pozițiilor, evitând explorarea inutilă a unor noduri. Ultimul pas constă în actualizarea valorilor și propagarea acestora înapoi în arbore, astfel încât să se obțină o estimare cât mai precisă a valorii poziției rădăcinii. Prin iterarea acestor pași și aplicarea lor în mod eficient, algoritmul NegaScout reușește să găsească o strategie optimă într-un timp mai scurt decât alte metode de căutare.

```

// Search game tree to given depth, and return evaluation of
// root node.
int NegaScout(gamePosition, depth, alpha, beta)
{
    if (depth=0 || game is over)
        // evaluate leaf gamePosition from
        // current player's standpoint
        return Eval (gamePosition);
    // present return value
    score = - INFINITY;
    n = beta;
    // generate successor moves
    moves = Generate(gamePosition);
    // look over all moves
    for i =1 to sizeof(moves) do
    {
        // execute current move
        Make(moves[i]);
        // call other player, and switch sign of
        // returned value
        cur = -NegaScout(gamePosition, depth-1,
                          -n, -alpha);
        if (cur > score) {
            if (n = beta ) OR (d <= 2)
                // compare returned value and
                // score value, update it if
                // necessary
                score = cur;
            else
                score = -NegaScout
                    (gamePosition,depth-1,
                     -beta, -cur);
        }
        // adjust the search window
        if (score > alpha) alpha = score;
        // retract current move
        Undo(moves[i]);
        // cut off
        if (alpha >= beta) return alpha;
        n = alpha+1;
    }
    return score;
}

```

Figura 3.2.3 NegaMax Algorithm Pseudo Code [Napc]

Capitolul 4 - Algoritmul Minimax

4.1 Introducerea algoritmului

Minimax este un algoritm utilizat în domeniul informaticii, în special în problemele de decizie și în jocuri care se joacă în 2 jucători, cum ar fi jocul de șah, tic tac toe, etc. Scopul acestui algoritm este de a determina cea mai bună mutare posibilă pentru un anumit jucător, analizând toate mutările posibile. Atunci când oricare dintre cei 2 jucători câștigă sau jocul se ajunge la egal, se va asigna o valoare a tablei de joc, ca să se indice situația jocului.

Algoritmul se bazează pe ideea de a simula toate posibilitățile de joc până la o adâncime dată în arborele de decizie, evaluând fiecare stare a jocului și alegând cea mai bună mișcare pentru jucătorul curent (Figura 4.1). Minimax încearcă să maximizeze rezultatul pentru jucătorul curent și să-l minimizeze pentru adversar. Dacă o stare a tablei de joc este în favoarea jucătorului maximal, se va asigna stării o valoare pozitivă, altfel se va asigna o valoare negativă [XYma19].

În Figura 4.1 este prezentat un arbore rezolvat cu minimax. Când este rândul lui Max, acesta are 2 posibile mutări, acesta poate să meargă pe oricare dintre aceste 2 mutări. După aceea este rândul lui Min, care la rândul său are de asemenea 2 mutări posibile, rezultând astfel în 4 posibile combinații la final. După aceea se asignează pentru fiecare posibilă situație o valoare conform funcției de evaluare euristice (4.3). Astfel jucătorul Min are de ales din nodurile finale, iar acesta va alege cel mai mic număr, adică pentru nodul 1 jucătorul Min are de ales cel mai mic număr dintre 10 și -20, astfel jucătorul min va alege nodul -20, la fel și pentru nodul numărul 2 în care Min, va trebui să aleagă cel mai mic număr dintre 15 și -10, astfel se va asigna valoarea -10 nodului 2. La final jucătorul Max, va trebui să aleagă cel mai mare număr dintre nodurile rezultate după alegerile de la jucătorul Min. Astfel Max va trebui să aleagă cel mai mare număr dintre -20 și -10, astfel Max va alege -10, rezultând astfel că mutarea cu valoarea -10 este cea mai bună mutare pentru jucătorul curent.

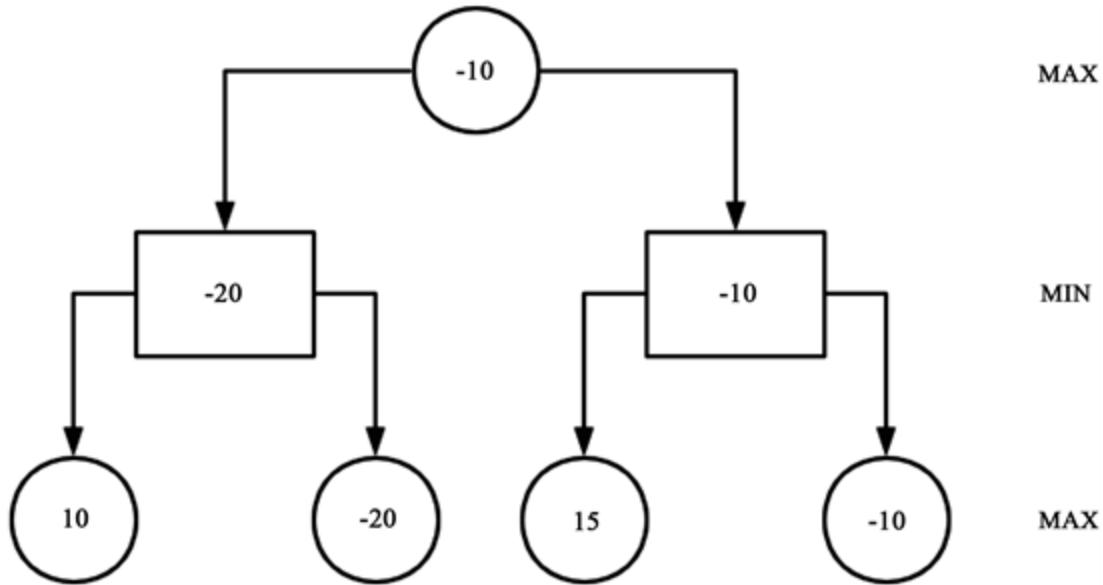


Figura 4.1: O posibilitate de arbore de joc [Apgt]

4.2 Alpha beta pruning

Alpha beta pruning este o tehnică de optimizare utilizată în algoritmul minimax pentru a căuta cea mai bună acțiune posibilă. Ajută să reducă timpul computațional eliminând noduri care nu sunt necesare în arborele de căutare. Deoarece algoritmul minimax este un algoritm de backtracking recursiv care selectează următoarea acțiune într-un joc de 2 persoane, cu cât apar mai multe noduri în arbore, cu atât durează mai mult căutarea, alpha beta pruning previne considerarea de stări care nu vor fi alese, făcând căutarea mai rapidă și mai eficientă [GAS23, SMB22].

În figura 4.2 este prezentat un exemplu de arbore minimax rezolvat cu alpha beta pruning. Arborele de minimax ramane la fel, doar ca atunci cand este randul lui Max, pentru nodul D se calculează normal, se alege cel mai mare număr dintre 3 și 5, adică 5 și se asigneaza valoarea 5 nodului D. În cazul nodului E, nu se va mai evalua și nodul cu valoarea 9 deoarece, beta are valoarea 5 calculata la nodul D, și cum se verifica nodul cu valoarea 6, alfa devine 6, iar conform algoritmului alpha beta, 5 este deja mai mic decat 6, deci nu mai are sens sa se intre și pe ramura cu 9. Apoi se calculează valoarea minimă pentru nodul B, deoarece este randul jucătorului Min, iar acesta primește valoarea 5 deoarece $5 < 6$. Acum se va evalua nodul F, unde se va alege cea

mai mare valoare dintre 1 și 2, adică 2 și se va asigna valoarea 2 la nodul F. F va returna valoarea 2 la C deoarece alfa va fi 5 (calculat la nodul B) iar cum beta este 2, calculat la nodul F, nu se vor mai calcula și restul nodurilor, adică nodul returnat va fi cel cu valoarea 5.

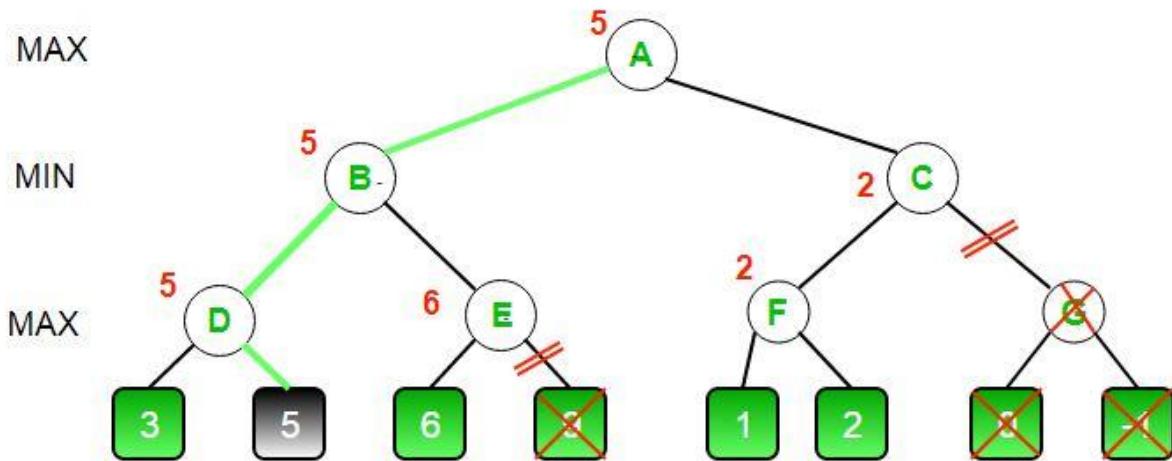


Figura 4.2 Minimax Algorithm in Game Theory | Set 4 (Alpha-Beta Pruning) [Magt]

4.3 Funcția euristică

Funcțiile euristice de evaluare sunt utilizate în algoritmi MiniMax pentru evaluarea situației curente ale jocului. Decizia finală este dependentă în mare parte de aceasta evaluare, și cât de bine este realizată. Astfel pentru a avea un rezultat cât mai bun, funcția euristică trebuie realizată cât mai bine [XYma19].

În jocurile de șah, funcția euristică poate să conțină diferite situații de joc din viața reală pentru a putea determina cea mai bună mutare, de exemplu: numărul de puncte pentru fiecare jucător, numărul de posibile mutări, structura pionilor, siguranța regelui, posibile mutări de a da șah mat, și multe altele. Funcția euristică poate fi folosită pentru tot felul de jocuri care se pot juca în 2 persoane, de aceea este un pilon foarte important pentru algoritmul minimax, fără de care ar fi foarte greu să se poată face evaluarea mutării și determinarea celei mai bune mutări.

4.4 Detalii de implementare

In figura 4.4 se poate observa o implementare algoritmului minimax. După cum se observă, conform explicațiilor de la capitolul 4.1, la început se verifică condiția de oprire, în cazul acesta dacă în poziția actuală s-a terminat jocul sau dacă adâncimea algoritmului a ajuns la 0. Dacă una din cele 2 condiții este adevărată se returnează valoarea calculată cu funcția euristică. Dacă este jucătorul maximal, se inițializează valoarea maximă, cu cea mai mică valoare posibilă. După care se parcurge fiecare poziție posibilă, și se apelează funcția minimax, cu o adâncime mai mică decât cea curentă, cu alfa și beta curente (pentru tăierea alpha beta), se cere calcularea pentru jucătorul minimal, și cu mutarea curentă, pentru a putea calcula valoarea acelei poziții. Procesul se repetă până când s-a ajuns la condiția de oprire. După returnarea rezultatului, se ia cel mai mare număr, dintre valoarea maximă și cea calculată curent pentru poziția cerută. Apoi se salvează valoarea maximă dintre alfa și evaluarea curentă. Dacă beta este mai mic sau egal cu alfa, atunci algoritmul se oprește, conform tăierii alpha beta pruning. La final se returnează valoarea calculată cu funcția euristică.

Dacă este jucător minimal se repetă aceeași pași ca și la jucătorul maximal, cu modificarea că se inițializează valoarea minimă, cu cea mai mare valoare posibilă, în cazul acesta +infinit.

```
def minimaxAlphaBeta(depth, alpha, beta, maximizingPlayer,
gamePosition):
{
    if(game over in gamePosition or depth==0):
        return gamePosition.value

    if maximizingPlayer:
        maxEval=-INFINITY
        for child in gamePosition.children:
            eval=minimax(depth-1, alpha, beta, false, child)
            maxEval=max(maxEval,eval)
            alpha=max(alpha,eval)
            if (beta <= alpha):
                break
        return maxEval
    else:
        minEval=+INFINITY
        for child in gamePosition.children:
            eval=minimax(depth-1, alpha, beta, true, child)
            minEval=min(minEval,eval)
            beta=min(beta,eval)
            if (beta <= alpha):
                break
        return minEval
}
```

Figura 4.4 Minimax Algorithm in Artificial Intelligence – ITYUKTA [MAI]

4.5 Minimax în jocul de sah

Algoritmul minimax este o tehnică fundamentală utilizată în motoarele de șah pentru determinarea celei mai bune mutări analizând rezultatele potențiale. În contextul proiectării AI de șah, este utilizată o combinație de estimatori de învățare automată și AI, astfel reducând sarcina hardware și sporind eficiența. Mai mult minimax este studiat în larg, în teoria jocurilor, unde proprietățile și problemele sale de corectitudine sunt exploataate arătându-și astfel semnificația în procesele de luare a deciziilor. În general, minimax, împreună cu tehnici, precum alpha beta (vezi 4.2), joacă un rol în dezvoltarea motoarelor de șah [MDCB23].

Deoarece pentru un joc de șah, arborele de căutare devine foarte mare, astfel crescând timpul de căutare pentru fiecare mutare, se utilizează tehnica alpha beta pruning, pentru a elimina din nodurile considerate inutile conform funcției de evaluare euristice (vezi 4.3) [BJDA23].

4.6 Eficiența algoritmului

Algoritmul prezintă o complexitate de $O(b^*d)$, unde b este numărul mediu de posibile mutări, iar d este adâncimea pe care o alegem în arbore. Folosind alpha beta pruning, complexitatea a scăzut la $O(b^*(d/2))$. În comparație cu alte algoritme de Ai, folosite pentru determinarea celei mai bune mutari de șah, cum ar fi MCTS (Monte Carlo Tree Search), care are complexitatea $O(\log(n))$, sau Neural Networks care are complexitatea $O(n)$, se observă faptul că minimax nu este cel mai bine performant, pentru aplicații complexe cum ar fi jocul de șah.

4.7 Avantaje

Algoritmul Minimax prezintă mai multe avantaje, inclusiv simplitatea și eficiența. Algoritmul este simplu de implementat și de înțeles, oferind rezultate fiabile fără a necesita eforturi sau resurse substanțiale. În plus, funcționează rapid și eficient, incorporând tehnica de tăiere alpha beta pentru a diminua semnificativ tensiunea hardware prin eliminarea mișcărilor irelevante. În domeniul sistemelor de vot, algoritmul se remarcă prin capacitatea să de a selecta în mod constant câștigătorii optimi, simplitatea și rezistența la schemele strategice de vot. În sistemele informatiche, împreună cu tăierea alpha beta, ajută la luarea deciziilor strategice prin anticiparea mișcărilor adversarilor și exploatarea eficientă a posibilelor rezultate. Mai mult în sarcinile de

Capitolul 4 - Algoritmul Minimax

interpolare spațială, abordarea minimax în operațiile kriging, poate îmbunătăți acuratețea rezultatelor prin prezicerea efectiva a greutăților semivariogramelor și depășind metode tradiționale și abordările de învățare automată [RBD22, SMB22].

4.8 Limitări

Minimax prezintă probleme precum ratele de convergență inferioare, complexitatea computațională și restricțiile privind tipurile de probleme. Abordarea clasică minimax poate duce la performanțe suboptime în ceea ce privește convergența. În plus, algoritmii existenți minimax de optimizare, se pot confrunta cu probleme fundamentale, cum ar fi costurile de calcul ridicate și eficiența scăzută a optimizării, împreuna cu restricții privind tipurile de probleme pe care le pot aborda efectiv. Abordările convenționale minimax se pot confrunta din cauza complexității minimizării complete, ceea ce duce la limitări în obținerea soluțiilor optime [WLPG23, MJAK21].

Capitolul 5 - Studiu de caz

5.1 Introducere Chess Openings

Pentru începători, învățarea jocului șah, este o provocare, datorită numeroaselor reguli și tacticii care trebuie stăpâname. Există numeroase platforme care pot ajuta incepatorii de șasă învețe, să se dezvolte, dar aceste platforme nu au explicate deschiderile de șah pe larg pentru începători. De aceea Chess Openings, a fost dezvoltată special cu acest scop în minte, pentru a ajuta începătorii să învețe deschiderile de șah, și să le exerseze împotriva unui motor pentru a putea vedea cât de bine le stăpanesc și cât de puternice sunt. Aplicația oferă o mare varietate de lecții de deschideri de șah, fiecare având explicații video și scrise, urmate de lecții și challenge-uri, pentru a-și putea testa fiecare utilizator, cunoștințele dobândite din explicațiile anterioare, și pentru a înțelege de ce e bună acea deschidere. Cursurile sunt făcute în aşa mod, încât până și cei mai buni jucători de șah, își pot reîmprospăta cunoștințele, parcurgând și rezolvând challenge-urile de la fiecare curs. Aceste challenge-uri sunt construite în funcție de cele mai comune scenarii la aceste deschideri de șah, pentru a putea fiecare jucător să se familiarizeze cu posibilitățile de joc, și oportunitățile de a pedepsi adversarul pentru o mutare proastă. După terminarea cursurilor, fiecare utilizator are opțiunea de a provoca ai-ul, ca să-și poată demonstra abilitățile.

Ai-ul de șah, este dezvoltat în aşa mod, încât să ajute incepatorii să învețe deschiderile, și principiile de șah, jucând pentru nivelul unui începător. Dezvoltat folosind algoritmul MiniMax cu alfa beta pruning, motorul de șah, reușește să provoace jucătorii începători, ajutându-i să-și dezvolte cunoștințele și strategiile de joc, pentru a le putea după să le pună în practică împotriva altor jucători.

5.1.1 Actori

Utilizatorii aplicației Chess Openings, după înregistrarea în aplicație au acces la o multitudine de funcționalități, care sunt descrise mai pe larg în capitolul 5.1.2. Aceștia se vor putea bucura de cursuri de deschidere de cel mai înalt nivel, de asemenea de un ai de șah, care îi va ajuta să învețe tacticii și să își dezvolte gândirea pentru atunci când vor decide să joace împotriva altor jucători. De asemenea vor putea să vadă toate jocurile complete, și vor putea continua unele meciuri pe care nu le-au terminat.

5.1.2 Cazuri de utilizare

În figura 5.1.2 se poate vedea diagrama de cazuri de utilizare ale unui jucător din aplicația Chess Openings

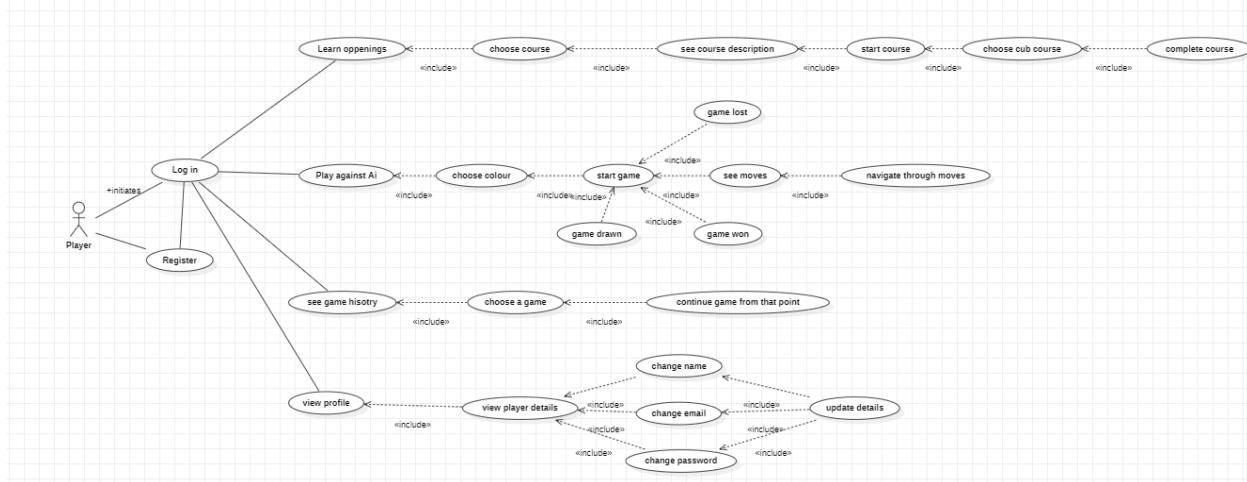


Figura 5.1.2 Diagrama de cazuri de utilizare

Jucătorul când deschide aplicația, va avea 2 opțiuni, LogIn sau Register. Cand jucatorul alege opțiunea de register, se va deschide o pagina, în care jucătorul este nevoit să-și introducă datele (numele, username-ul, e-mailul, parola). După completarea acestor date, jucătorul va fi trimis la pagina de LogIn, unde va fi nevoie să-și introducă, username-ul sau e-mailul, și parola. După logarea în aplicatie, se va deschide pagina de porinre, cu cele 3 optiuni: Learn Openings, Play against ai si Game History.

Pentru opțiunea de Learn openings, se va deschide o pagina nouă în care vor fi afișate toate cursurile disponibile. Jucătorul va alege un curs pe care vrea să îl învețe, iar în momentul acela, se va deschide o pagină nouă, în care se va explica deschiderea, scris și cu un video. La momentul în care jucătorul crede că a înțeles deschiderea, va apăsa pe butonul start course, iar o nouă pagină se va deschide, unde acesta va exersa ceea ce a învățat în mai multe sub cursuri.

Pentru opțiunea de Play against ai, se va deschide o pagina nouă, în care jucătorul va alege cu ce culoare vrea să joace, după care își va pune la încercare cunoștințele împotriva unui motor de şah dezvoltat în cadrul aplicației.

Pentru opțiunea de Game history, se va deschide o pagina nouă, în care vor apărea toate jocurile jucate de acel jucător, acesta putând să vadă la ce mutare a rămas, statusul jocului, randul căruia

jucător este, și cu ce culoare a jucat în acel meci. Jucătorul are opțiunea de a continua jocul din acel punct.

Pentru opțiunea view profile, se va deschide o pagina nouă, în care vor apărea toate datele cu care jucătorul și-a creat contul, având opțiunea de a-și schimba numele, emailul sau parola.

5.1.3 Diagrama de clase

În figura 5.1.4 se află diagrama de clase a aplicației Chess Openings. În interiorul diagramei se prezintă clasele Service și cele de tip controller pentru a arăta cum comunica aplicația server cu cea client.

În diagrama se află următoarele clase:

- CourseStartedByPlayerController, în care serverul adaugă în baza de date, cursul început de utilizatorul curent, verifica dacă mutarea făcută de jucător în interiorul cursului este legală și cea corectă, mută mutarea de computer, oferă sfaturi în cazul în care jucătorul este blocat, resetează tabla și returnează toate sub cursurile terminate. Aceasta clasa comunică la rândul ei cu CourseStartedByPlayerService, CourseService, PlayerService și SubCourseService.
- GameController, în care serverul adaugă în baza de date un joc nou început de către jucător, returnează lista de jocuri jucate returnează mutările efectuate în timpul jocului, permite navigarea prin mutările deja efectuate. Acest controller comunică cu GameService și BoardService.
- BoardController, în care serverul adaugă table de șah în baza de date, preia și trimită configurația unei table la client. Acest controller comunică la rândul lui cu clasa BoardService.
- SubCourseController, în care serverul preia toate subcursurile sau unul specific din baza de date. Clasa comunică la rândul ei cu clasele SubCourseService, BoardService.
- MovePieceController, în care serverul verifică dacă o mutare efectuată de client este legală și mută fiecare tip de piesă, verifică dacă meciul s-a terminat, returnează toate măsurile pentru o piesă specifică. Clasa comunică la rândul ei cu PieceService, BoardService și GameService
- RegisterController, în care serverul înregistrează un utilizator în baza de date și confirmă tokenul trimis pentru verificarea emailului. Clasa comunica cu PlayerService.

Capitolul 5 - Studiu de caz

- PlayerController, în care serverul returnează jucătorul după username or email și actualizează jucătorul. Clasa comunică cu PlayerService.
- LogInController, în care serverul autentifică utilizatorul și generează un token de sesiune. Clasa comunica cu PlayerService.
- AiController, în care serverul pune ai-ul de șah să caute cea mai bună mutare, după care o returnează la client. Clasa comunică la rândul ei cu MiniMax.
- CourseController în care severul returnează un curs, sau toate cursurile din baza de date. Aceasta clasă comunică la randul ei cu CourseService.
- BoardService, în care serverul căuta o tabla de șah după id, o salvează în baza de date și o actualizează.
- PieceService, în care serverul verifică dacă o mutare este corectă pentru anumite piese, transformă o mutare în notația standard de șah, face o mutare pe tabla de șah, și o refac.
- CourseService, în care serverul caută un curs după id, după nume și returnează toate cursurile disponibile.
- CourseStartedByPlayerService, în care serverul adaugă un curs început pentru jucătorul curent, schimba statusului cursului, returnează un anumit curs.
- SubCourseService, în care serverul returnează toate subcursurile din baza de date, sau doar un subcurs anume după denumirea acestuia.
- GameService, în care serverul caută un meci după id, adaugă un nou meci în baza de date, returnează un meci după id-ul jucatorilor și a tablei de joc, sau doar după id-ul jucatorului și actualizează datele jocului.
- PlayerService, în care severul salvează și validează datele unui jucător, cauta un jucător după email sau username și parola.
- MiniMax, în care serverul executa algoritmul MiniMax pentru căutarea celei mai bune mutări din poziția respectivă. Clasa comunică la randul ei cu Evaluation.
- Evaluation, în care serverul evaluează o mutare după anumiți factori, de exemplu numărul de mutări disponibile, atacuri, numărul de puncte de pe tabla, și multe altele.

Capitolul 5 - Studiu de caz

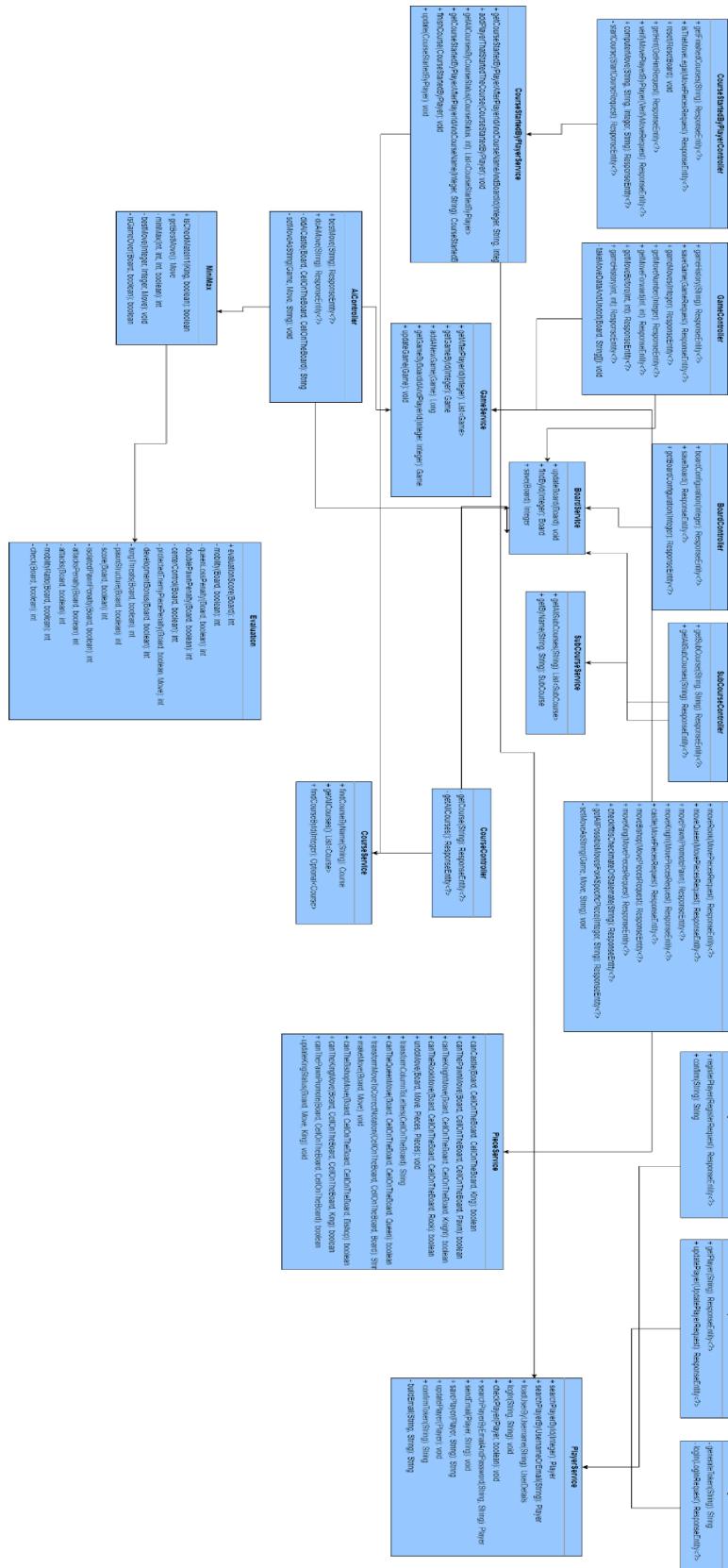


Figura 5.1.3 Diagrama de clase

5.1.4 Aplicații similare

Există o multitudine de aplicații similare cu Chess openings, cum ar fi Chess.com, Litchess. Chess.com, cea mai populară aplicație de șah, oferă mulți roboți contra care poți juca șah, de diferite nivele, și oferă și lecții de deschideri, oferind explicații și oferind suport online pentru concurarea împotriva altor jucători, fiecare jucător putând astfel să-și demonstreze abilitățile. Litchess, este o aplicație similară lui Chess.com, oferind roboți de diferite nivele, cursuri de deschidere, postate de alți utilizatori, și de asemenea oferă și suport online pentru concurarea împotriva altor utilizatori.

Chess openings, se remarcă prin faptul că, cursurile sunt gratuite, și explicate mai în detaliu, oferind mai multe subcursuri cu chalengeri pentru jocul de deschidere, astfel constituind fundația unui meci bun de șah, de la început, pentru utilizatorii începători. Platforma este foarte ușor de folosit, cu o interfață foarte bine construită, pentru a putea oferi servicii de cel mai înalt nivel. Oferă cursurile dezvoltate la cel mai înalt nivel pentru a spori cunoștințele utilizatorilor, și a accelera progresul acestora în meciul jucat contra un alt adversar. Aplicația oferă un istoric al tuturor jocurilor jucate împotriva ai-ului, astfel fiecare utilizator poate să țină cont de progresul sau, și poate să aleagă să continue un meci de unde a fost lăsat.

5.2 Tehnologii

5.2.1 Server

5.2.1.1 Java

Java este un limbaj de programare versatil bazat pe principii orientate pe obiecte care a avut un impact semnificativ asupra domeniului tehnologiei internetului. Limbajul aderă la conceptul fundamental de „scrie o dată, rulează oriunde”, permitând software-ului să funcționeze pe diverse platforme fără a fi nevoie de recompilare. Creat inițial de James Gosling în limitele Sun Microsystems, portabilitatea Java este posibilă prin compilarea codului în bytecode Java, care poate fi executat pe diverse platforme utilizând o mașină virtuală Java.

Java MVC, care este compatibil cu Jakarta EE și implementează arhitectura model-view-controller, prezintă o alternativă la tehnologia Faces și este reglementată de

JSR-371, valorificând proiectul Eclipse Krazo. De la înființarea sa în 1995, Java a suferit o evoluție continuă sub supravegherea Sun Microsystems și ulterior Oracle Corporation. [GOR18] Java este un limbaj de programare cu scop general creat pentru a permite dezvoltatorilor să creeze cod care poate rula pe orice platformă fără a fi nevoie de recompilare, principiu denumit „scrie o dată, rulează oriunde” (WORA). Compilarea aplicațiilor Java are ca rezultat bytecode, capabil de execuție pe orice mașină virtuală Java (JVM) pe diferite platforme hardware. În timp ce sintaxa Java seamănă cu C și C++, oferă mai puține funcționalități de nivel scăzut în comparație cu aceste limbi. Cu toate acestea, Java se mândrește cu caracteristici dinamice, cum ar fi modificarea codului de rulare și reflectarea, care sunt de obicei absente în limbile tradiționale compilate. Instrucțiunile Java Bytecode seamănă foarte mult cu codul mașinii și necesită utilizarea Java Runtime Environment (JRE) pentru execuție. Java este construit ca un limbaj de programare orientat pe obiecte, aliniat la principiile Programării Orientate pe Obiect (OOP). Aceste principii cuprind incapsularea, moștenirea, polimorfismul și abstractizarea.

- Incapsularea implică gruparea datelor și a metodelor conexe într-o unitate coezivă cunoscută sub numele de clasă, facilitând organizarea codului, gestionabilitatea și simplitatea, reducând în același timp complexitatea.
- Moștenirea permite crearea unei noi clase derivate dintr-o clasă existentă, moștenind atributele și comportamentele sale, permitând în același timp adăugarea de noi proprietăți și comportamente.
- Polimorfismul permite interschimbabilitatea obiectelor din diferite clase, atât timp cât au o interfață comună, sporind flexibilitatea proiectării sistemului și simplificarea codului.
- Abstractizarea implică ascunderea detaliilor de implementare, expunând în același timp numai caracteristicile esențiale ale obiectului utilizatorului, sporind claritatea codului și reducând complexitatea. Java folosește clase, obiecte, moștenire, interfețe și alte elemente OOP pentru a întruchipa aceste principii.

În Java aceste principii sunt implementate prin intermediul claselor, obiectelor, interfețelor și altelor feature-uri de OOP.

5.2.1.2 Versiuni de Java

Limbajul de programare Java a cunoscut numeroase modificări de la înființarea sa în anii 1990, împreună cu numeroase incorporări de clase noi și adăugări la biblioteca standard. Progresul

Capitolul 5 - Studiu de caz

limbajului Java a fost supravegheată de către Java Community Process (JCP), folosind Java Specification Requests (JSR) pentru a sugera și defini toate modificările aplicate platformei Java. Oracle a contribuit semnificativ la avansarea limbajului, în special cu versiunea java 11 servind ca ediție de asistență pe termen lung (LTS), primind suport extins (începând cu momentul actual, versiunea 11 este susținută până în septembrie 2026). În plus, există și alte iterații de java, cum ar fi Java SE 17, clasificate și ca LTS și programate pentru suport prelungit.

În prezent, cea mai recentă iterație de java este Java 22, care și-a făcut debutul pe 19 martie 2024 [Jdk22].

Aplicația software furnizată este dezvoltată folosind Java 21 în implementarea serverului, iar secțiunea ulterioară evidențiază unele dintre îmbunătățirile majore introduse limbii, conform documentației oficiale [Jdk21]. Aceste îmbunătățiri, implementate în java 21, oferă numeroase avantaje pentru dezvoltatori, cuprinzând securitatea codului, performanța și compatibilitatea.

- Thread-uri virtuale (Virtual Threads): caracterizate ca fiind ușoare, diminuează semnificativ nivelul de efort necesar în dezvoltarea, întreținerea și monitorizarea aplicațiilor concurente cu randament ridicat.
- Colecții secvențială (Sequenced Collections): Sunt propuse interfețe noi pentru a delimita colecțiile cu o secvență specificată de întâlniri. Fiecare dintre aceste colecții este caracterizată de un element inițial distinct definit, elemente ulterioare și culminează cu elementul final. În plus, oferă interfețe consistente de programare a aplicațiilor pentru recuperarea elementelor primare și finale, precum și pentru manipularea elementelor într-o secvență inversă.
- Modele de înregistrare (Record Patterns): Creșterea limbajului de programare Java prin încorporarea modelelor de înregistrare facilitează deconstrucția valorilor înregistrărilor. Cuibărirea tiparelor de înregistrare și a tiparelor de tip permite o abordare robustă, declarativă și configurabilă a traversării și manipulării datelor.
- Potrivirea modelului pentru switch (Pattern Matching for switch): Creșterea limbajului de programare Java prin încorporarea potrivirii modelelor pentru expresii și instrucțiuni comutatoare. Extinderea potrivirii tiparelor pentru comutare permite evaluarea unei expresii împotriva diferitelor modele, fiecare legat de o anumită acțiune, facilitând astfel exprimarea concisă și sigură a interogărilor complexe bazate pe date.

- Clase neidentificate și metode principale de instanță (Unnamed Classes and Instance Main Methods): Propagating evoluția limbajului de programare Java pentru a facilita inițierea programatorilor începători fără înțelegerea prealabilă a caracteristicilor adaptate software-ului complex. În loc să recurgă la o versiune alternativă de Java, indivizii au posibilitatea să formuleze declarații concise pentru aplicații de clasă solitară și, ulterior, să progreseze către integrarea funcționalităților sofisticate în programele lor pe măsură ce competența lor avansează.
- API-ul mecanismului de încapsulare a cheilor (Key Encapsulation Mechanism API): este conceput pentru a prezenta o interfață pentru mecanismele de încapsulare a cheilor (KEMs), care sunt instrumente criptografice utilizate pentru protejarea cheilor simetrice prin aplicarea criptografiei cu cheie publică.

5.2.1.3 Spring

Spring, cunoscut pe scară largă în lumea Java pentru utilizarea sa extensivă, în special în dezvoltarea de aplicații web și enterprise, s-a remarcat prin adaptabilitatea sa remarcabilă. Acționând ca o platformă cheie pentru gestionarea injectiei de dependențe, Spring oferă o gamă variată de opțiuni de configurare, inclusiv XML, Adnotări și JavaConfig. De-a lungul timpului, Spring a evoluat semnificativ pentru a se adapta cerințelor dinamice ale aplicațiilor de afaceri contemporane, integrând funcționalități avansate, cum ar fi îmbunătățiri de securitate, suport pentru depozitele de date NoSQL, manipularea eficientă a datelor mari, facilități simplificate pentru procesarea loturilor și interoperabilitate cu diverse sisteme. Importanța sa centrală este evidențiată prin rolul său în accelerarea dezvoltării aplicațiilor, prin furnizarea de resurse care facilitează integrarea, optimizarea codului și implementarea metodologiilor eficiente de dezvoltare. Arhitectura Spring, definită prin concepte precum Inversion of Control (IoC), Aspect-Oriented Programming (AOP) și mecanisme riguroase de gestionare a tranzacțiilor, îl plasează în centrul atenției ca un instrument esențial și indispensabil pentru dezvoltatorii Java. IoC permite developilor să elimine dependențele harcodate dintre obiecte în aplicație, în timp ce AOP permite modularizarea de probleme, cum ar fi logarea, securitatea și managementul de tranzacții. [FOG22]

Inversiunea controlului (Inversion of Control - IoC) și Injecția dependențelor (Dependency Injection - DI) sunt două concepte legate în ingineria software, care sunt adesea folosite împreună și sunt componente importante ale framework-ului Spring.

IoC se referă la conceptul de decuplare a managementului ciclului de viață al unei entități de entitatea însăși. Din punct de vedere istoric, o entitate a fost însărcinată cu crearea și gestionarea propriilor dependențe; cu toate acestea, IoC transferă această responsabilitate către un element distinct, cunoscut sub numele de container sau fabrică. În consecință, entitatea își dobândește dependențele prin injecție, mai degrabă decât prin instanțiere directă. Această metodologie facilitează o bază de cod mai adaptabilă și compartimentată, permitând entităților să sufere modificări sau ajustări fără a afecta sistemul în ansamblu.[JSW21]

Injecția de dependență este o instanțiere particulară a IoC în care o entitate este „injectată” cu dependențele sale în timpul rulării, mai degrabă decât să le construiască intern. Pentru a realiza acest lucru pot fi utilizate diverse abordări, cum ar fi injecția constructorului, injecția de setare sau injecția pe câmp. Responsabilitatea de a produce dependențe și de a le injecta în entitate revine containerului sau fabricii, care este determinată de configurația sistemului. În consecință, acest lucru favorizează cuplarea liberă între entități, deoarece nu li se cere să dețină cunoștințe cu privire la complexitatea modului în care dependențele lor sunt formate sau gestionate.

5.2.1.4 Spring Data

Spring data, este o componentă în cadrul Spring, care oferă o abordare unificată și ușor de folosit pentru accesul în cadrul aplicațiilor Java. Obiectivul principal, este de a eficientiza procesul de dezvoltare al aplicațiilor, prin furnizarea unui model de programare standardizat, și o abstractizare care se întinde pe diverse tehnologii de acces la date, precum baze de date relaționale, NoSQL și depozite semnificative de date.

Spring data oferă un set de interfețe comune și metode pentru interacțiunea cu diferitele depozite de date, cum ar fi repository-urile. Acestea se integrează perfect cu tehnologiile populare de acces la date, precum Spring JPA (Java Persistence API), care se bazează pe specificația standard JPA, și oferă funcționalități suplimentare și caracteristici de conveniență, cum ar fi implementările automate de repositories, metode de interogare, suport pentru paginare și funcționalități de audit.

Spring Data JPA, oferă compatibilitate cu mai mulți furnizori, cum ar fi Hibernate, EclipseLink și OpenJPA. Prezintă o abordare consistentă și unitară pentru lucru cu JPA în diferiți furnizori, permitând dezvoltatorilor să treacă ușor între aceștia sau să ii combine după nevoie.

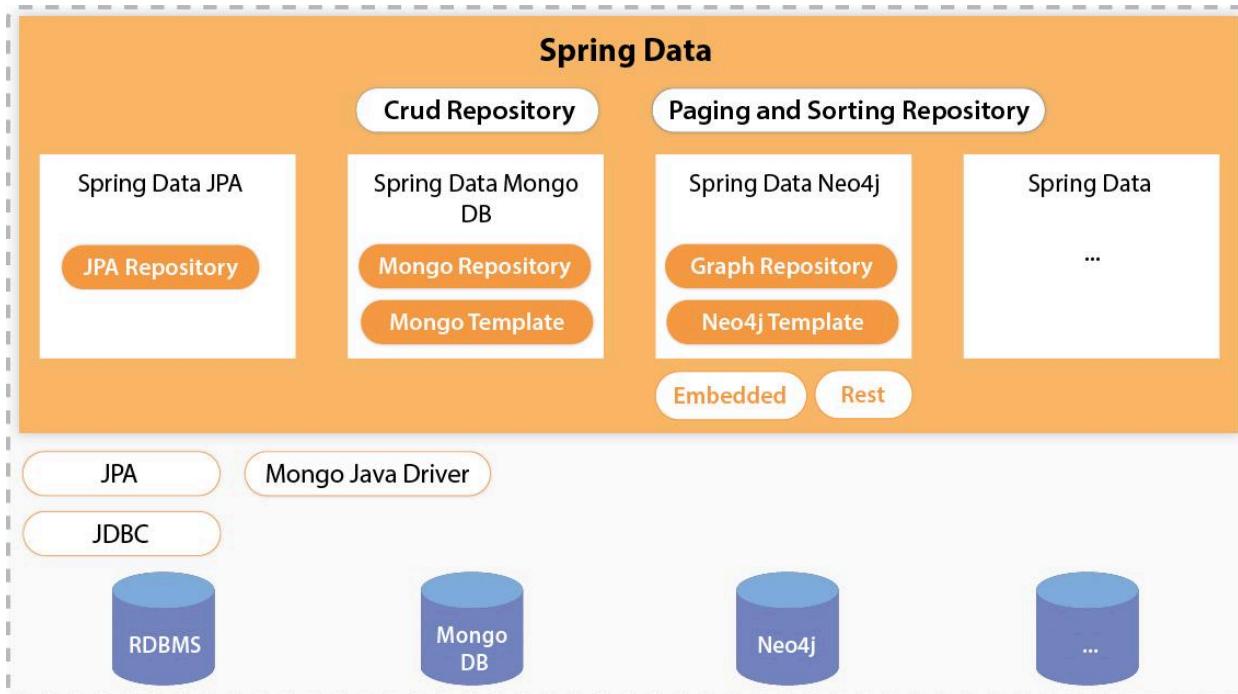


Figura 5.2.1.4 Spring Data [Sd]

5.2.1.5 Spring Boot

Pentru configurarea unei aplicații Spring, este nevoie de un proces care durează destul de mult timp, și este nevoie de crearea unor fișiere de configurare și instalarea și setarea unor tool-uri suplimentare. Folosind Spring Boot se scăpa de acest proces de configurare, și proiectul se poate executa imediat, folosind abordarea convenție peste configurare. Mai mult de atât, Spring Boot gestionează toate dependențele necesare, asigurând compatibilitatea între două versiuni diferite.

5.2.1.6 JUnit Testing

Pentru asigurarea corectitudinii funcționalităților aplicației, sunt folosite teste, ca să fim siguri că toate cerințele au fost validate înainte de a trimite aplicația în producție. Unit tests sunt folosite pentru a valida corectitudinea serviciilor implementate, iar JUnit 5 este folosit foarte des pentru

acest scop. Aceste teste sunt fundamentale pentru a verifica ușor și rapid dacă rezultatul este acela dorit sau dacă este predispus la erori. JUnit este un framework de test pentru Java care oferă o opțiune elegantă pentru testarea funcționalităților.

5.2.2 Client

5.2.2.1 HTML

HTML (HyperText Markup Language) este un limbaj fundamental în crearea paginilor web. Dezvoltat în 1990 de Tim Berners-Lee, HTML structurează documente web pentru a fi afișate în browser. Aceasta definește aspectul și formatul conținutului de o pagina web. HTML funcționează împreună cu tehnologii precum CSS pentru styling și JavaScript pentru interactivitate. Cea mai recentă versiune, HTML5, care a fost folosită pentru dezvoltarea aplicației Chess Openings, permite accesul la modelele de obiecte ierarhice prin JavaScript, și permite capacitați îmbunătățite de structurare. Înțelegerea HTML este crucială pentru dezvoltarea web, deoarece formează coloana vertebrală a site-urilor web prin organizarea conținutului și furnizarea structurii pentru design și funcționalitate. [JM23]

Este important de menționat faptul ca HTML nu este un limbaj de programare, este un limbaj markup, care funcționează ca un sistem pentru identificarea și descrierea diferitor elemente ale unui document, cum ar fi titluri, paragrafe, liste. Documentele HTML sunt compuse dintr-o serie de tag-uri, care definesc structura unui conținut. Tagurile sunt cuprinse în interiorul parantezelor (< și >) și sunt compuse dintr-un nume pentru acel tag, și una sau mai multe atribute care produc mai multe informații despre acel tag. În figura 5.2.2.1 este prezentată o bucată de cod HTML.

```
<!DOCTYPE html>
<html lang="en">

<head>
    <title id="pageTitle"> </title>
    <link href="chessBoard.css" rel="stylesheet" />
</head>
```

Figura 5.2.2.1 Exemplu tag HTML

5.2.2.2 CSS

CSS (Cascading Style Sheets) este un element esențial în dezvoltarea web, utilizat pentru stilizarea paginilor web. CSS oferă dezvoltatorilor posibilitatea de a controla aspectul și designul elementelor HTML de pe o pagina web. CSS poate fi scris într-un fișier separat, sau în interiorul

targului definit în HTML. Dezvoltatorii moderni preferă folosirea unor framework-uri, precum Bootstrap, care produc elemente CSS deja scrise, ca să accelereze dezvoltarea și consistența asupra mai multor pagini web.

5.2.2.3 JavaScript

JavaScript, numit inițial LiveScript, a fost creat de Brendan Eich la Netscape la mijlocul anilor 1990 și ulterior redenumit JavaScript în 1995. De-a lungul anilor, JavaScript a evoluat semnificativ, devenind un limbaj versatil cu biblioteci pentru diverse aplicații pe partea client și server, desktopuri și dispozitive mobile. Este un limbaj dinamic funcțional orientat pe obiecte potrivit pentru îmbunătățirea paginilor web și dezvoltarea diverselor aplicații web, inclusiv simulări care pot rula pe diferite dispozitive și infrastructuri cloud. Pentru dezvoltatorii ABAP, învățarea JavaScript implică înțelegerea elementelor sale de bază, diferențele față de ABAP și crearea de programe simple. Funcțiile moderne JavaScript au fost, de asemenea, introduse pentru a îmbunătăți lizibilitatea și ușurința de codare, făcându-l un instrument valoros pentru educația generală de programare. [Tjs23]

5.3 Acces și salvare de date

5.3.1 Bază de date

Pentru construirea bazei de date, am folosit PostgreSQL. PostgreSQL este sistem de gestionare a bazelor de date relaționale cu obiecte, este o soluție puternică și versatilă pentru diverse aplicații. Dezvoltat pe baza Postgres v.4.2 la Universitatea din California, Berkeley, PostgreSQL oferă support SQL, concepte relaționale de obiecte și extensibilitate pentru tipuri de date personalizate și metode de acces.

Conform standardelor de construcție pentru o baza de date, fiecare tabel este denumit în concordanță cu denumirea entității pe care o salvează. De exemplu pentru tabela player, se vor salva toți jucătorii. Fiecare tabelă are o cheie primară, rolul acesteia fiind să identifice fiecare intrare din tabel. Când o coloană face referire la o cheie primară dintr-un alt tabel, astfel devenind o cheie străină, se va numi NumeleTabelei_id, de exemplu pentru tabelul game, facem referire la id-ul unui jucător, astfel în tabela game se va numi player_id.

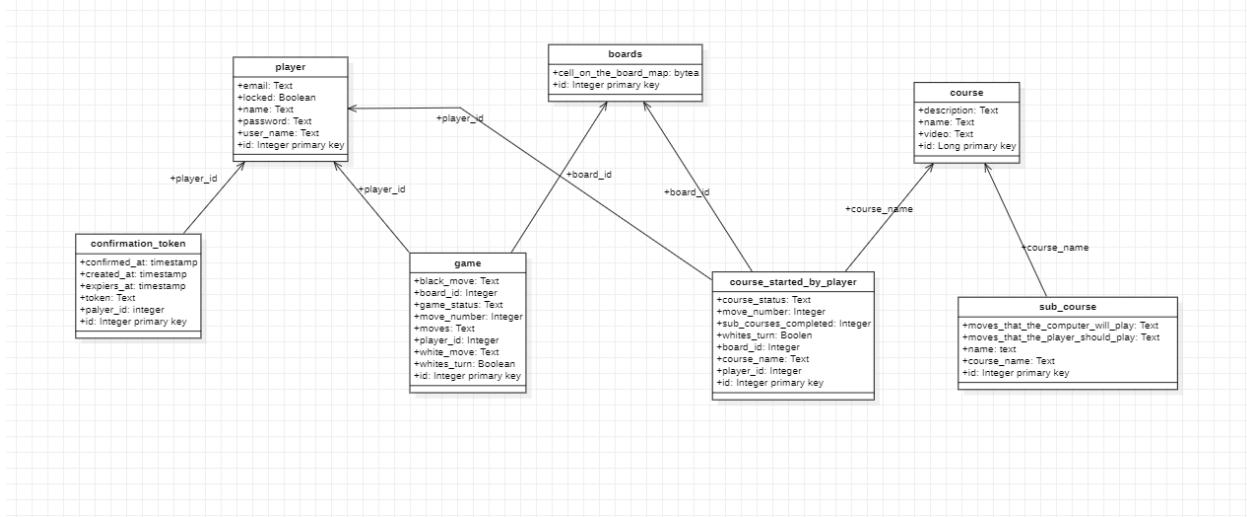


Figura 5.3.1 Diagrama bazei de date

O scurtă explicație pentru fiecare tabel:

1. Player

Tabelul player, este construit din coloanele email, pentru emailul jucătorului, locked, folosit pentru a stabili dacă jucătorul și-a confirmat emailul, name, numele jucătorului, password, parola acestuia, user_name și id.

2. confirmation_token

Aceasta tabelă conține toate tokenurile trimise pentru confirmarea emailului, este construită din coloanele confirmed_at, pentru a se salva data și ora la care s-a confirmat emailul, created_at, pentru salvarea datei și orei cand s-a creat tokenul de confirmare pentru emailul jucătorului, expires_at, data și ora la care expira tokenul, token, player_id, id-ul jucătorului căruia i s-a assignat tokenul, id.

3. boards

Tabela contine toate tabele de joc folosite, fie în cursuri, fie în meciuri, este construită din cell_on_the_board_map, care este defapt tabla, care contine cte o celulă pentru fiecare celulă din tabla, continand proprietățile pieselor care se află pe acea celulă, și id.

4. game

Reține toate meciurile jucate, este construită din black_move, salvând mutările jucate de jucătorul negru, board_id, care salvează o referință la tabla pe care se joacă, game_status, statusul jocului, care poate fi STARED, WHITE_WON, BLACK_WON, DRAW,

STALEMATE, move_number, numărul mutării la care se află cei 2 jucători, moves, toate mutările jucate de cei 2 jucătorii, player_id, o referință către jucătorul care joacă, white_move, toate mutările jucătorului alb, whites_turn, ca se știe a cărui jucător este randul să joace, id.

5. course

Reține toate cursurile care sunt disponibile în aplicație, este compusă din, description, descrierea cursului, name, numele cursului, video, un video care explica cursul, id.

6. sub_course

Reține toate sub cursurile, ale unui curs, și este construită din coloanele moves_that_the_computer_will_play, ce mutrai va muta computerul, moves_that_the_player_should_play, ce mutări ar trebui să facă jucătorul, name, numele sub cursului, course_name, care face referire la cursul din care face parte subcursul, id.

7. course_stared_by_player

Este compusă din toate cururile începute de un anumit jucător, și este construită din course_status, care poate avea valorile COMPLETED, InPROGRESS, UnOPEN, move_number, numărul mutării la care se află cursul, sub_courses_completed, numărul de subcursuri pe care le-a completat jucătorul din cursurile curente, whites_turn, să se salveze a cui e randul să mute, board_id, să se salveze referinta tablei pe care se executa cursul, course_name, se salvează o referinta a cursului care se completează, player_id, referinta asupra jucătorului care completeaza cursul, id.

5.3.2 Acces la date

Accesul la baza de date, a fost făcut folosind Spring JPA, care este explicitat în secțiunea 5.2.1.4. Acesta produce o implementare Spring pentru a abstractiza repository-ul care este un element fundamental în Domain-Driven Design (DDD), care este o tehnică folosită în dezvoltarea de aplicații software. Spring JPA gestionează transparent toate implementările disponibile JPA, și facilitează operații CRUD care sunt foarte ușor de executat.

Java Database Connectivity (JDBC) este un API special construit pentru limbajul Java care definește modul în care un utilizator poate accesa o bază de date. Parte din platforma Oracle Java Standard Edition, JDBC oferă o gamă de metode pentru interogarea și actualizarea datelor în

baze de date relaționale. În plus, puntea JDBC ODBC permite conexiunea la orice sursă de date accesibilă prin ODBC din mediul de găzduire al mașinii virtuale Java.

Cu JDBC, o aplicație poate utiliza simultan mai multe implementări. API-ul oferă o modalitate de încărcare dinamică a pachetelor Java dorite și înregistrarea lor cu managerul de drivere JDBC. Managerul de drivere funcționează ca o fabrică de conexiuni pentru crearea conexiunilor JDBC. Conexiunile JDBC permit generarea și executarea de instrucțiuni, inclusiv instrucțiuni de actualizare cum ar fi SQL CREATE, INSERT, UPDATE și DELETE, precum și instrucțiuni de interogare cum ar fi SELECT. Mai mult, este fezabil să se apeleze proceduri stocate printr-o conexiune JDBC.

5.4 Ghid utilizare aplicatie

Aplicația Chess Openings este concepută pentru a fi cât mai ușor de folosit pentru orice fel de utilizator indiferent de vârstă, astfel putând să ajute pe toată lumea să își dezvolte cunoștințele de șah, astfel devenind un jucător mai bun.

Procesul de logare în aplicație

La începutul aplicației, se află o pagina de login pentru a permite utilizatorilor să-și creeze cont, astfel putând să-și urmărească progresul. Crearea de cont se face folosind adresa de email, și parole securizate, astfel definind un mediu securizat. Prin logarea în aplicație, jucătorul deblochează acces la o multitudine de funcționalități, precum profile individuale, abilitatea de a studia diferite deschideri de șah, și abilitatea de a juca împotriva unui motor de șah.

Funcția de autentificare în aplicație, este reprezentată în figura 5.4.1, își asumă un rol foarte important, pentru securitatea jucătorului. Rolul său principal este de a verifica identitatea jucătorului, astfel căutând să folosească aplicația corespunzător, și blocând astfel accesul neautorizat. Pentru a putea intra în aplicație, jucătorii sunt nevoiți să-și introducă numele, emailul, username-ul și parola.

Procesul de log in este primul pas în autentificarea în aplicație, pentru un jucător. După cum este ilustrat în figura 5.4.2, pagina de login pune jucătorii să-și introducă emailul sau username-ul și parola, pentru a putea primii acces în aplicație.

Capitolul 5 - Studiu de caz

Pentru a putea fortifica securitatea de date ale jucatorului, este necesara incoroprarea unor masuri suplimentare, pe langa standardul de conectare folosind username, email și parola. Criptarea datelor apare ca fiind o opțiune pivotală, în salvarea acestor date în baza de date, sau în transmiterea între server și partea de client. Acest layer suplimentar de protectie asigura confidentialitatea datelor, astfel acestea devenind indescifrabile fără cheia de criptare folosită, chiar și în cazul de acces neautorizat.

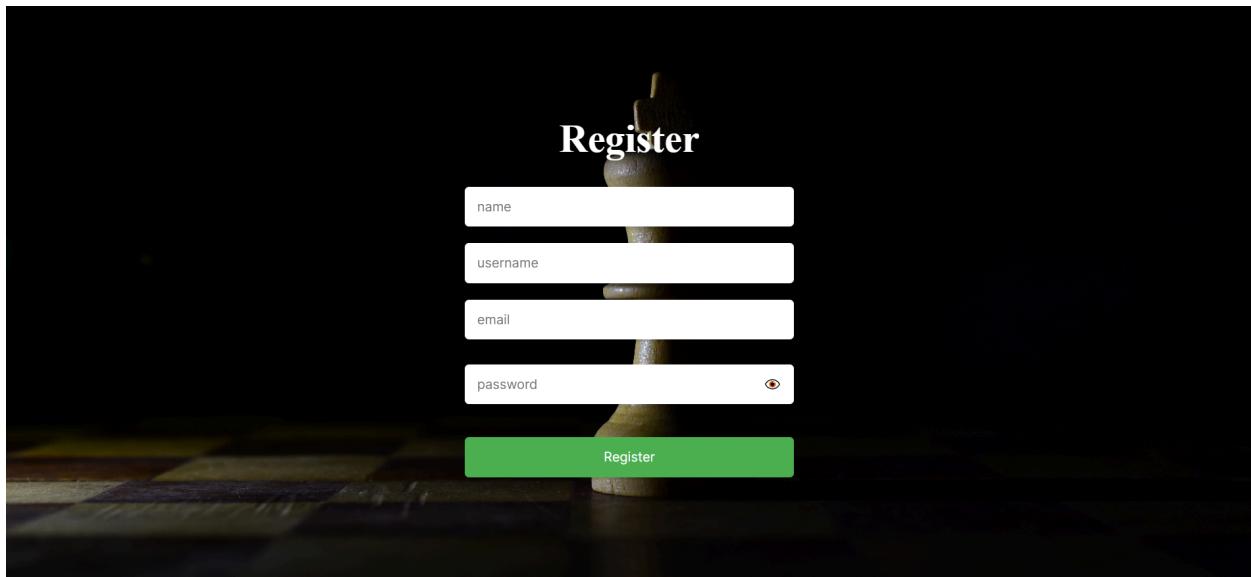


Figura 5.4.1 Pagina de register

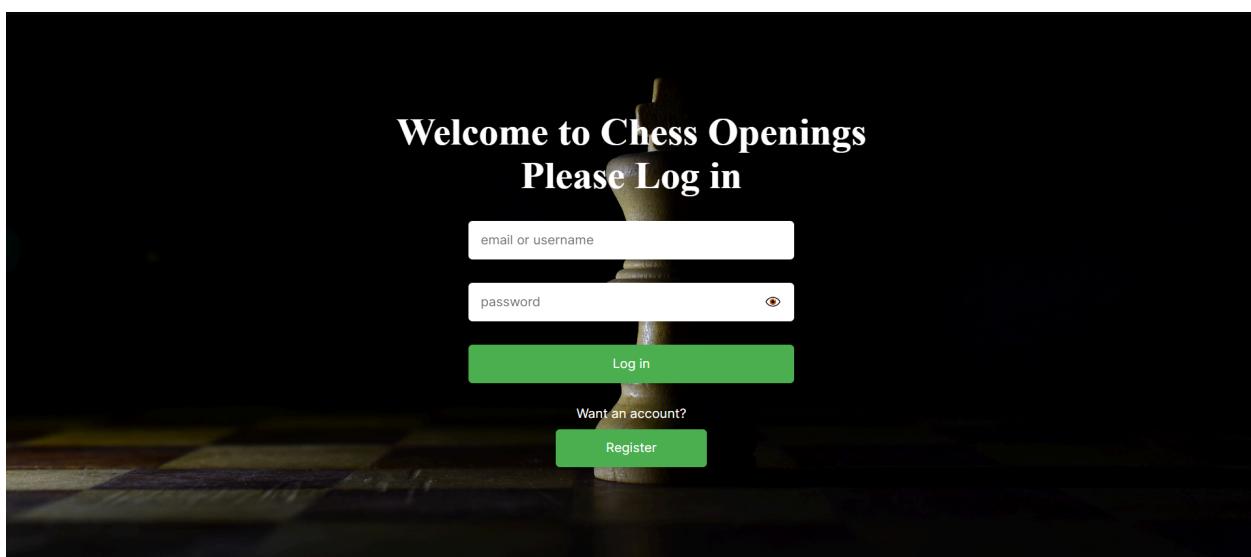


Figura 5.4.2 Pagina de login

Pagina de pornire

Dupa logarea in aplicatie, se va deschide pagina de pornire care va oferi utilizatorului cele 3 optiuni ale aplicatiei: Învățarea de deschiderilor, jucarea impotriva Ai-ului si vizualizarea istoricului de meciuri. In figura 5.4.3 se poate observa pagina de pornire a aplicatiei.

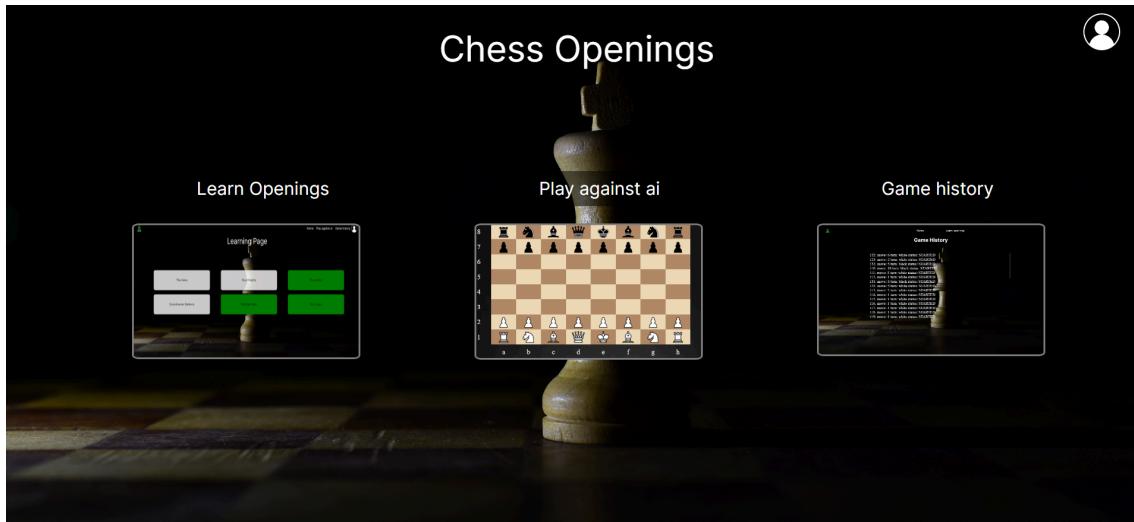


Figura 5.4.3 Pagina de pornire a aplicatiei

Alegerea cursului

Aplicația oferă utilizatorilor o multitudine de cursuri de deschideri de șah, construite în aşa fel încât să ajute jucătorii de orice nivel să își dezvolte cunoștințele. Jucătorii pot căuta foarte ușor ce tip de curs vor să învețe. După alegerea unui curs o pagina de descriere a acelui curs este prezentată, în care jucătorul poate să citească despre acea deschidere, cât și să urmărească un video explicativ despre acea deschidere. După parcurserea deschiderii cursului, se va deschide o nouă pagină, în care jucătorul va avea opțiunea să pună în practică ceea ce a invatat din pagina anterioară, parcurgând subcursuri care pun în practică diferite scenarii de joc, care se pot întâlni în momentul în care se joacă acea deschidere, astfel pregătind jucătorul pentru eventualele pericole, și cum să le contraatace și să pedepsească adversarul. Dacă la un moment dat, jucătorul nu mai știe ce trebuie să mute, poate să apese pe butonul de Hint, care îl va debloca, putând astfel să progreseze fără a se întoarce la pagina de învățare. La finalizarea cursului, aplicația va marca cursul terminat ca fiind făcut, iar cursul în cauză se va colora în culoarea verde în pagina de cursuri.

Capitolul 5 - Studiu de caz

În figura 5.4.4 se poate observa pagina de cursuri, dintre care poate să aleagă jucătorul, figura 5.4.5 prezintă pagina informativă a fiecărui curs, care de asemenea produce parte teoretică a cursului, iar în figura 5.4.6 se prezintă pagina în care jucătorul poate să-și exerceze cunoștințele dobândite anterior.

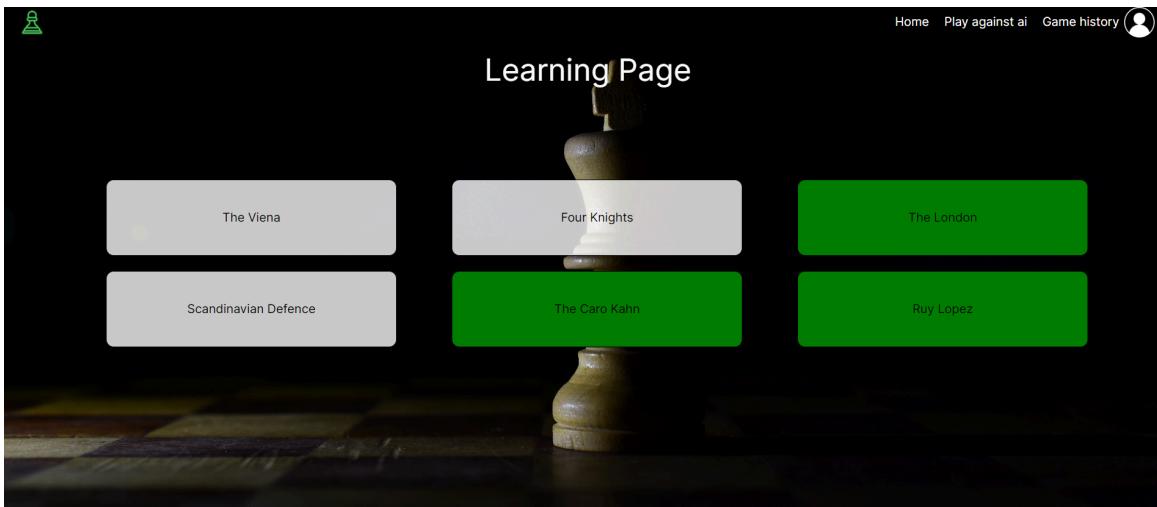


Figura 5.4.4 Pagina de cursuri

A screenshot of a chess informative page for the Vienna Game. It includes a detailed text description of the opening, a video player showing a video of a player explaining the game, and a 'Start course' button.

Figura 5.4.5 Pagina informativă a cursului

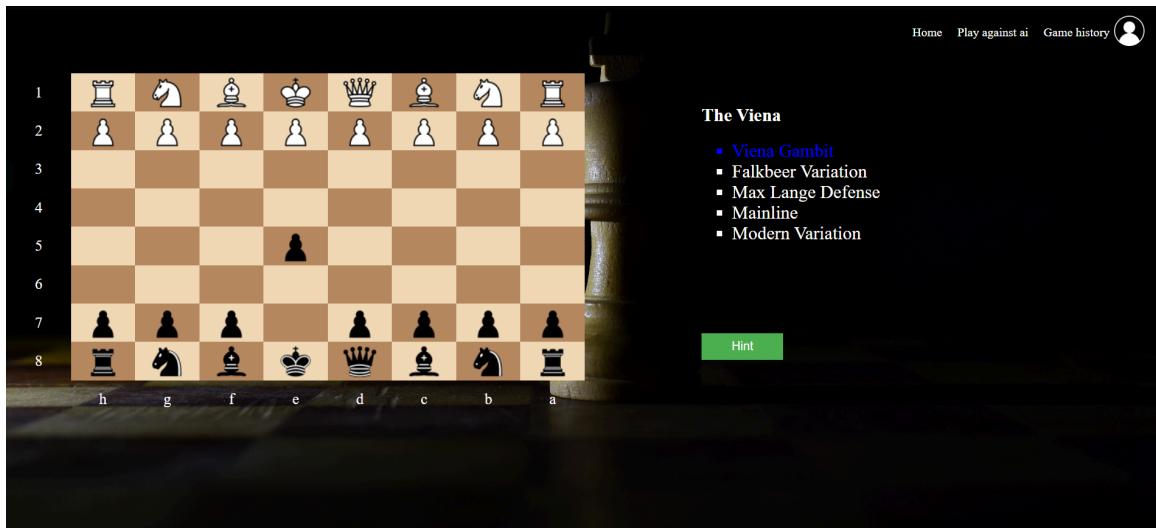


Figura 5.4.6 Pagina practică

Jucarea improtva Ai-ului

Chess openings, se remarcă prin motorul de șah pe care îl are implementat, ajutând jucătorii să își pună în valoare abilitățile dobândite în urma studierii cursurilor. Ai-ul este construit în aşa fel încât să ajute începătorii să își dezvolte abilitățile și să nu fie puși împotriva unui motor care joacă de nivelul unui maestru de șah. Înainte de începerea meciului, jucătorul va trebui să aleagă cu ce culoare vrea să joace, aşa cum este ilustrat în figura 5.4.7. În figura 5.4.8 se poate observa pagina în care jucătorii își pot pune la încercare cunoştințele. De asemenea se mai poate observa, pentru fiecare piesă apăsată, aplicația arată unde se pot muta fiecare piesă, iar în partea dreaptă a paginii se pot vedea toate mutările făcute până în acel moment, de asemenea având opțiunea să navigheze printre acele mutări, în diferite stări ale jocului.

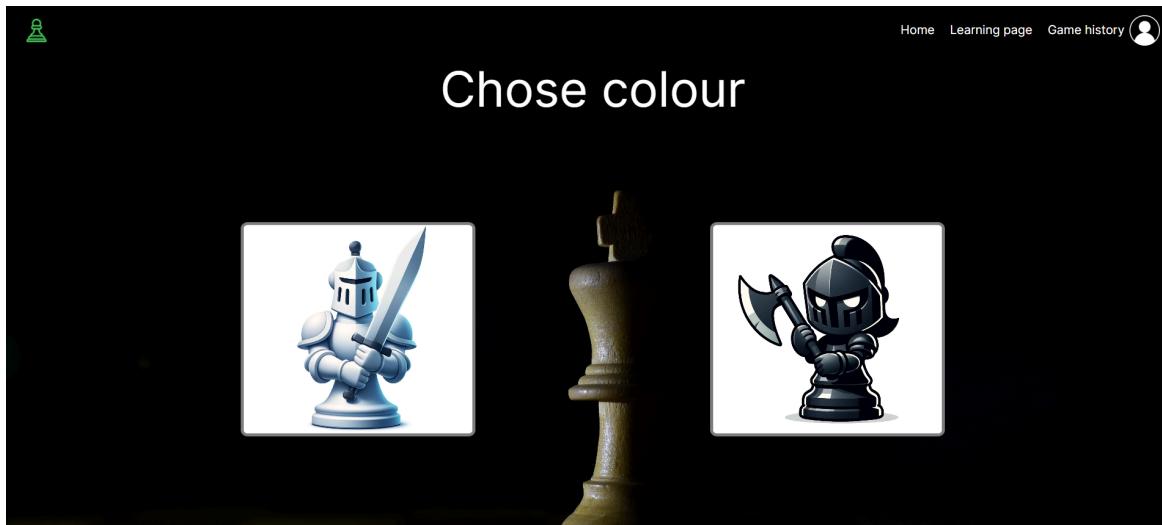


Figura 5.4.7 Alegerea culorii



Figura 5.4.8 Pagina de joc importiva ai-ului

Istoricul jocurilor

Aplicația, mai permite utilizatorului să își vadă istoricul jocurilor, jucate împotriva ai-ului. De asemenea acesta mai permite reluarea meciurilor, din acel moment al jocului, la o simplă apăsare pe meciul, din istoric, care se dorește a fi reluat, sau doar să se poată vedea meciul, cum a fost jucat, și ce mutari bune, sau proaste au fost făcute de-a lungul acestuia. În figura 5.4.7 se poate observa pagina în care jucătorii pot să își vadă meciurile jucate, împreună cu mutarea la care s-a oprit jocul, randul cărui jucător este să mute, statusul meciului, care poate să fie STARTED, WHITE WON, BLACK WON, STALEMATE, DRAW, și de asemenea culoarea cu care a jucat jucătorul acel meci.



Figura 5.4.7 Pagina de istoric a jocurilor

Pagina de profil

Jucătorii au opțiunea de a-și vedea detaliile cu care și-au creat profilul, inclusiv opțiunea de a-și schimba acele date, în afară de username, care nu este permis pentru a fi schimbă. În figura 5.4.8 se poate vedea pagina de profil a jucătorului, cu datele sale, și cu opțiunea de a se schimba datele. Parola nu va putea fi vizibilă, din motive de securitate.

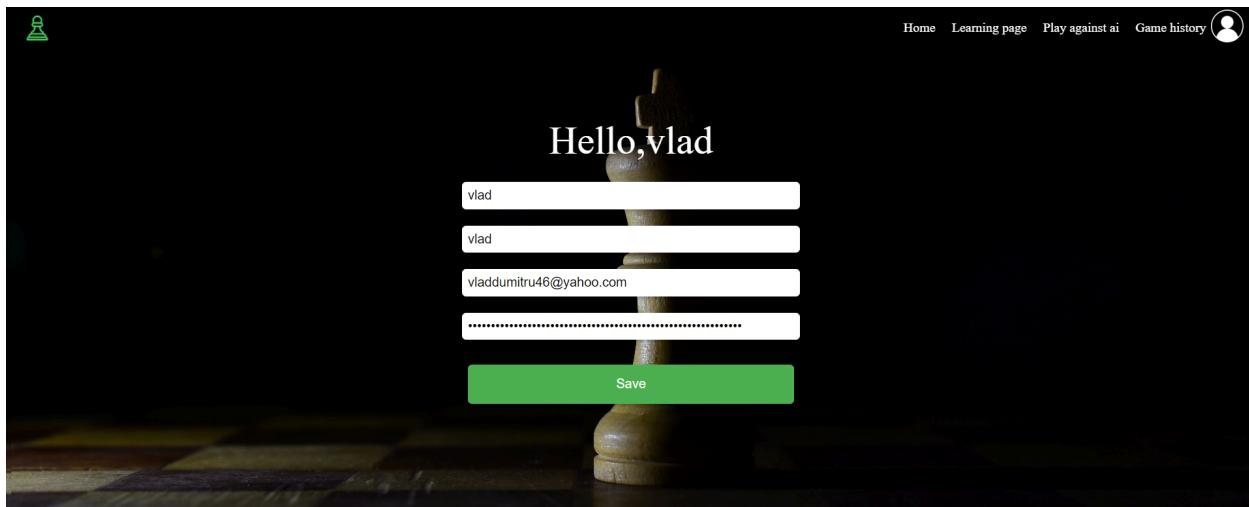


Figura 5.4.7 Pagina de profil a utilizatorului.

5.5 Arhitectura și design patterns

Pentru a face arhitectura aplicației și întreținerea acesteia, Chess Openings folosește un număr de şabloni de proiectare (design patterns). Aceste şabloni, oferă cea mai bună soluție în

dezvoltare software și soluții testate care apar în problemele comune în timpul dezvoltării unei aplicații. Prin urmare, programul scris este mai ușor de înțeles, mai ușor de întreținut, și mai scalabil. Sabloanele de proiectare, ajuta programatorii să creeze programe flexibile și de încredere, care sunt esențiale pentru a asigura succesul de viitor al aplicațiilor soft.

5.5.1 Șablonul de proiectare MVC

Şablonul MVC (Model - View - Controller) este o arhitectură software populară utilizată pentru a separa preocupările aplicației în trei componente distincte: Model, View, Controller.

Model: reprezintă datele aplicației și logica de afaceri. Este responsabil pentru stocarea, recuperarea și manipularea datelor.

View: afișează datele modelului utilizatorului. Este responsabil pentru formatarea și prezentarea datelor într-un mod ușor de înțeles de către utilizator.

Controller: acționează ca un intermediar între model și view. Primește solicitări de la utilizator prin interfața vizuală, le trimită modelului pentru procesare și actualizează vizualizarea cu rezultate.

În cadrul aplicației Chess Openings, am folosit şablonul MVC, pentru ca crea o arhitectură bine organizată și menținabilă. Implementarea acestuia, aduce multe beneficii. Mai întâi separa diferite probleme, permitând programatorilor să se focuseze pe un task specific. Aceasta separare îmbunătățește organizarea codului și întreținerea acestuia. De asemenea mai promovează modularitatea și reutilizarea, permănd dezvoltarea individuală și extinderea usoara a funcționalităților.

Şablonul MVC de asemenea face experiența cu utilizatorul mai bună, prin separarea logicii de prezentare de datele din server. Utilizatorii pot interacționa cu partea de View, fără a afecta direct partea de Model, astfel asigurând integritatea datelor.

5.5.2 Șablonul de proiectare Factory

Şablonul de proiectare Factory este un model de proiectare de creație, care oferă o interfață pentru crearea de instanțe ale claselor. Este utilizat pentru a ascunde complexitatea creării obiectelor și pentru a oferi o interfață unificată pentru crearea de obiecte dintr-o familie de clase înrudite.

Sablonul factory aduce o multitudine de beneficii precum:

Decuplare: Factory decuplează codul client de implementare specifică a claselor pe care le crează. Acest lucru face ca programul client să fie mai curat și mai ușor de înțeles.

Encapsulare: Sablonul encapsulează logica de creare a obiectelor, prin ascunderea acesteia de codul client.

Reutilizare: Factory poate fi utilizată pentru reutilizarea codului de creare a obiectelor.

Scalabilitatea: Sablonul poate fi utilizat pentru a crea obiecte într-un mod scalabil.

În cadrul aplicației Chess Openings am folosit sablonul Factory pentru a imbunătăți codul, scapând astfel de codul duplicat, și pentru întreținerea ușoară a acestuia. În figura 5.5.2 se poate observa un exemplu de cum a fost folosit sablonul factory în cadrul aplicației.

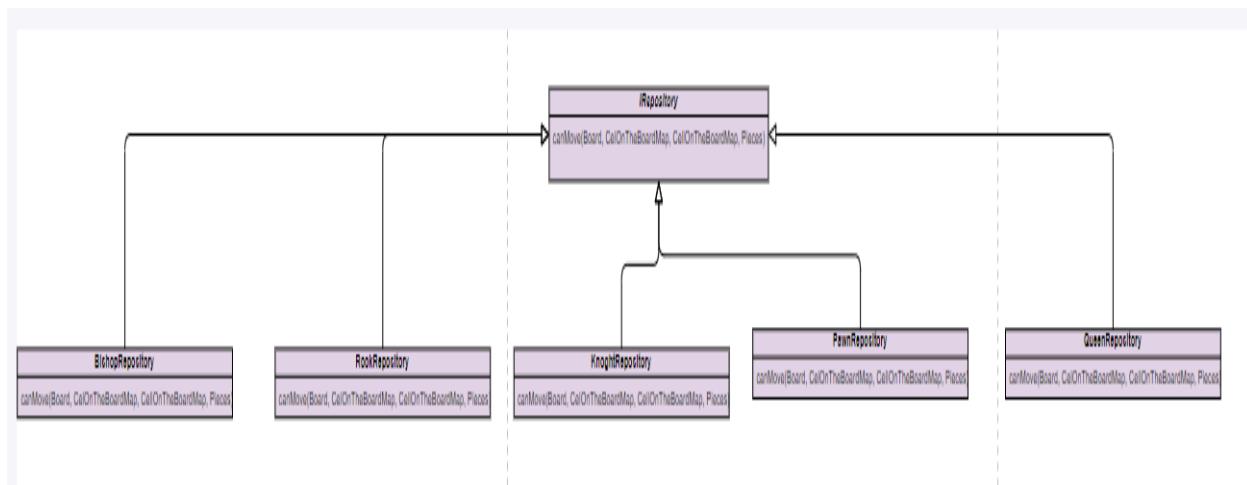


Figura 5.5.2 Exemplu de folosire a sablonului Factory în cadrul aplicației

5.5.3 Sablonul de proiectare Combinator

Sablonul de proiectare Combinator este un model de proiectare software care oferă o modalitate flexibilă de a combina funcții simple pentru a crea funcții mai complexe. Se bazează pe ideea de a compune funcții pure (fără efecte secundare) pentru a obține funcții noi cu același comportament pur.

Acesta vine cu o multitudine de beneficii, precum:

Flexibilitate: Permite combinarea facilă a funcțiilor simple pentru a crea funcții complexe cu o gamă largă de comportamente.

Reutilizare: Funcțiile pure pot fi reutilizate cu ușurință în diferite contexte.

Testabilitate: Funcțiile pure sunt mai ușor de testat, deoarece nu au efecte secundare.

Compoziție: Permite construirea de funcții complexe din funcții simple într-un mod modular.

În cadrul aplicației Chess Openings am folosit sablonul de proiectare Combinator pentru ca crea o validare a datelor introduse de către utilizator în momentul cand se înregistrează în aplicație. In figura 5.4.3.1 se poate observa modul in care a fost folosit sablonul Combinator in aplicatie, iar in figura 5.4.3.2 se poate observa cum a fost creat sablonul Combinator.

```
public class PlayerValidator {  
  
    7 usages  
    public void validatePlayer(Player player) throws PlayerValidationException {  
        ValidationResult result = PlayerValidationInterface  
            .isEmailValid()  
            .and(PlayerValidationInterface.isUserValid())  
            .and(PlayerValidationInterface.isPasswordValid())  
            .apply(player);  
    }  
}
```

Figura 5.4.3.1 Modul de folisire a sablonului Combinator in aplicatie

```
3 usages  
public interface PlayerValidationInterface extends Function<Player, ValidationResult> {  
  
    2 usages  
    default PlayerValidationInterface and(PlayerValidationInterface other) {  
        return player -> {  
            ValidationResult result = this.apply(player);  
            return result.equals(ValidationResult.SUCCESS) ? other.apply(player) : result;  
    };  
}
```

Figura 5.4.3.2 Crearea operatorului “and” esențial pentru crearea sablonul Cobniator

5.6 Testare

Validarea datelor este un proces esențial pentru asigurarea corectitudinii și fiabilității informațiilor introduse de utilizator într-un sistem înainte de a fi procesate ulterior. Prin aplicarea unor criterii riguroase și reguli de verificare, acest proces se asigură că datele integrate sunt precise și securizate. În cadrul proiectului de front-end, o atenție deosebită este acordată validării

Capitolul 5 - Studiu de caz

datelor introduse de utilizator înainte de a fi prelucrate. În situația în care un utilizator încearcă să își creeze un cont cu detalii incomplete sau incorecte, precum parole neconcordante sau adrese de email invalide, aceștia vor primi prompt un mesaj de eroare informativ indicând natura invalidă a datelor introduse.

În ceea ce privește proiectul backend, s-au implementat teste unitare pentru toate clasele de servicii folosind framework-ul JUnit. Aceasta este un cadru de testare unitară popular în mediul Java, cunoscut pentru simplitatea și eficiența sa.

Pentru a facilita procesul de testare unitară, s-a folosit și Mockito, un cadru de simulare pentru Java. Mockito permite crearea de obiecte simulate care să simuleze comportamentul dependențelor, permitând astfel testarea izolată a claselor de servicii fără a invoca funcțiile externe reale. Această abordare contribuie la crearea unor teste fiabile și repetabile prin reducerea dependențelor și asigurarea consistenței rezultatelor testelor.

În conformitate cu modelul Arrange, Act, Assert (AAA), fiecare test unitar a fost structurat în trei secțiuni distincte:

- Arrange: Inițializarea obiectelor și dependențelor necesare, configurarea obiectelor simulate și furnizarea datelor de testare pentru a configura starea inițială dorită pentru metoda testată.
- Act: Invocarea metodei testate cu datele de testare pregătite și dependențele asociate.
- Assert: Compararea rezultatului execuției metodei cu rezultatul așteptat și efectuarea aserțiunilor necesare pentru a verifica comportamentul corect al clasei de serviciu și producerea rezultatelor dorite.

Prin adoptarea acestui model, testele unitare sunt organizate în mod coerent și oferă o separare clară între fazele de pregătire, execuție și verificare. Aceasta îmbunătățește lizibilitatea și ușurința de întreținere a testelor, facilitând identificarea și remedierea problemelor în timpul procesului de dezvoltare.

În figura 5.6 se poate observa un exemplu de cum a fost folosit Junit cu Mockito pentru a testa o funcționalitate a aplicației.

```
@Test
void testFindCourseById() {
    // Arrange
    int courseId = 1;
    Course expectedCourse = new Course( name: "Math", description: "description");
    expectedCourse.setId((long) courseId);

    when(courseRepository.findById(courseId)).thenReturn(Optional.of(expectedCourse));

    // Act
    Optional<Course> actualCourse = courseService.findCourseById(courseId);

    // Assert
    Assertions.assertEquals(expectedCourse, actualCourse.orElse( other: null));
    verify(courseRepository, times( wantedNumberOfInvocations: 1)).findById(courseId);
}
```

Figura 5.6 Exemplu de testare al unei funcționalități

5.7 Docker

Docker este un software care creează, orchestrează și gestionează containerele care pot rula pe Linux și Windows.

Containerele docker permit execuția unui server proxy care traduce cereri de tipul HTTP/1.1 de la clientul web în cereri HTTP/2 folosind serviciile gRPC.

Docker este o unealtă care facilitează incapsularea unui proces creând artefakte distribuibile pentru orice tip de aplicație care poate fi reutilizată în execuția oricărui tip de mediu.

Containerele au fost disponibile în distribuțiile Linux de mulți ani, dar au fost rar folosite din cauza complexității necesare pentru gestionarea lor. Docker a realizat un progres remarcabil în valorificarea containerelor Linux prin combinarea unui format standardizat de "ambalare" cu ușurință în utilizare. Acest lucru ajută echipele de dezvoltare de pretutindeni să beneficieze de portabilitatea, integrarea și dezvoltarea simplificată a aplicațiilor, aşa cum erau inițial destinate pentru containerele Linux, fără succes real din cauza straturilor de complexitate. Docker aduce, de asemenea, avantaje în ceea ce privește scalarea performanței: reducând amprenta aplicației prin containere formatare, dependențele la nivel de sistem sunt minime, rezultând adesea în reducerea a câteva sute sau chiar zeci de megabytes. În plus, pornirea unui container durează

milisecunde, o diferență remarcabilă comparativ cu minutele necesare în mod obișnuit pentru mașinile virtuale.

Cu toate acestea, cea mai mare inovație și cel mai semnificativ impact adus de Docker și containerele Linux este schimbarea fundamentală în consumul de aplicații și servicii. Aplicațiile monolitice pot fi împărțite în zeci sau chiar sute de aplicații mici, construite pentru scopuri specifice, care, atunci când sunt conectate împreună, îndeplinesc aceeași funcție ca și aplicația originală. Avantajul este că aceste bucăți pot fi recrise, reutilizate și gestionate mult mai eficient decât aplicațiile monolitice.

Docker este o aplicație de tip client server. Clientul docker comunica cu serverul docker, cunoscut ca și "Docker daemon", care la rândul lui se ocupă de toate cererile [NP20].

Imagini Docker

Anterior, era necesar să "ambalezi" nu doar aplicația în sine, ci și multe dintre dependențele sale, inclusiv librăriile și serverele daemon. Cu toate acestea, era dificil să te asiguri că mediile de execuție erau cu adevărat identice. Cu Docker, aplicația este implementată împreună cu toate fișierele necesare pentru execuția sa. Imaginile stratificate ale lui Docker fac acest proces eficient, asigurând că mediul de rulare corespunde cu cel așteptat.

Fiecare container în Docker se bazează pe o imagine, care servește ca fundație pentru tot ceea ce este implementat și executat. Pentru a lansa un container, poți descărca fie o imagine existentă dintr-un depozit public, fie să creezi propria ta imagine personalizată. Imaginile Docker constau din unul sau mai multe straturi ale unui sistem de fișiere, care sunt asociate în mod tipic cu fiecare pas de construire folosit pentru a crea imaginea.

Imaginile sunt blocurile esențiale pentru utilizarea Docker. Acestea urmează un format stratificat, utilizând sisteme de fișiere Union, care sunt construite pas cu pas folosind o serie de instrucțiuni.

De exemplu:

- Adaugă un fișier
- Rulează o comandă
- Deschide un port

Astfel, acestea pot fi considerate "codul sursă" pentru containere. Ele sunt portabile, ușor de trimis mai departe, stocabile și actualizabile [NP20].

Containere Docker

Containerele sunt o modalitate foarte simplă și portabilă de a stoca o aplicație și dependențele sale.

Containerele reprezintă incapsularea unei aplicații și a dependențelor sale. La prima vedere, ele pot părea foarte asemănătoare cu mașinile virtuale, la fel ca VM-urile, un container conține o instanță izolată a unui sistem de operare care poate fi utilizat pentru a rula aplicații. Totuși, containerele oferă mai multe avantaje care permit utilizarea unor cazuri dificile sau imposibile cu VM-urile tradiționale:

- Containerele împart resursele cu sistemul de operare gazdă, ceea ce le face oarecum mai eficiente, în funcție de caracteristicile aplicației în sine. Containerele pot fi pornite și opriate într-o fracțiune de secundă, iar pentru aplicațiile care rulează în containere, există mult mai puțin overhead comparativ cu aplicațiile care rulează nativ pe sistemul de operare gazdă.
- Portabilitatea containerelor are potențialul de a elimina o întreagă clasă de erori cauzate de schimbările subtile ale mediului de rulare.
- Natura ușoară a containerelor înseamnă că dezvoltatorii pot rula zeci de containere simultan, permitând emularea unui sistem distribuit gata de producție.

Containerele au numeroase avantaje pentru utilizatorii finali, clienți și dezvoltatori, pe lângă posibilitatea implementării în cloud. Utilizatorii pot descărca și rula aplicații complexe fără a fi nevoie să petreacă mult timp cu probleme de configurare, instalare sau modificări necesare sistemelor lor. În mod similar, dezvoltatorii acestor aplicații pot evita problemele create de diferențele în mediile utilizatorilor și de disponibilitatea dependențelor necesare.

Toate containerele utilizează un singur nucleu, iar izolarea este implementată complet în cadrul acestuia, un proces cunoscut sub numele de virtualizare a sistemului de operare. Principalele avantaje se concentrează pe eficiența resurselor, deoarece nu este nevoie de un întreg sistem de operare pentru fiecare funcție izolată [NP20].

5.7.1 Dockherizarea aplicației

Strategia de deploy a aplicației Chess Openings se învârte în jurul alegerii Docker ca platformă pentru deploy. Docker oferă o multitudine de beneficii care o fac o alegere excelentă pentru a deploia aplicații.

Capitolul 5 - Studiu de caz

Pentru a deploia o aplicație backend, ea se containerizează prin crearea unui Dockerfile, care definește toate configurațiile și dependențele necesare. Fișierul Dockerfile specifică imaginea de bază, care servește ca mediul de rulare pentru aplicație. Este aleasă o imagine OpenJDK potrivită, astfel încât să fie la fel cu versiunea de Java folosită în aplicație. În imaginea 5.6.1.1 se poate vedea Dockerfile pentru aplicația de backend.

Pentru aplicația de frontend, se urmează aceeași strategie că și pentru aplicația de backend, unde se alege o imagine potrivită pentru aplicația de front end. În imaginea 5.6.1.2 se poate vedea Dockerfile pentru aplicația de frontend.

```
1 ► FROM openjdk:21
2
3 WORKDIR /app
4
5 COPY core/target/Core-0.0.1-SNAPSHOT.jar app.jar
6
7 EXPOSE 8080
8
9 ENTRYPOINT ["java", "-jar", "app.jar"]
10
```

Imaginea 5.6.1.1 Dockerfile pentru backend

```
1 ► FROM nginx:alpine
2 COPY . /usr/share/nginx/html
3 EXPOSE 80
4 CMD ["nginx", "-g", "daemon off;"]
5
```

Imaginea 5.6.1.2 Dockerfile pentru frontend

Implementarea aplicației folosind Docker oferă mai multe avantaje. În primul rând, Docker asigură portabilitatea, garantând o implementare consistentă în diverse medii, indiferent de variațiile infrastructurii de bază. Acest lucru elimină problemele de compatibilitate și garantează un comportament consistent al aplicației. În al doilea rând, Docker permite o scalabilitate ușoară,

Capitolul 5 - Studiu de caz

permîțând replicarea și distribuția fără efort a instanțelor aplicației, acordând eficient diverse sarcini de lucru. Mai mult, containerele Docker oferă izolare, creând un mediu de execuție controlat și sigur pentru aplicație. În final, Docker simplifică gestionarea aplicației și controlul versiunilor, permîțând actualizări și rollback-uri eficiente ale implementărilor, după cum este necesar.

Capitolul 6 - Concluzii

În concluzie, această lucrare a adus contribuții semnificative în dezvoltarea și implementarea aplicației Chess Openings, o platformă user-friendly creată pentru a învăța jucătorii de șah deschideri și pentru a-și dezvolta gândirea strategică pe parcursul unui meci de șah. Lucrarea include o explorare detaliată asupra diferenților algoritmilor de dezvoltare a motoarelor de șah, cum ar fi Monte Carlo Tree Search, Principal Variation Search și NegaScout, punând un accent deosebit pe algoritmul MiniMax în combinație cu AlphaBeta, care au fost utilizati pentru dezvoltarea motorului de șah din cadrul aplicației.

Contribuțiiile acestei lucrări depășesc simpla implementare a motorului de șah bazat pe algoritmul MiniMax. Cursurile de deschideri de șah sunt structurate astfel încât să fie accesibile tuturor tipurilor de jucători, indiferent de nivelul lor de cunoștințe și experiență. La început, cursurile se concentrează pe aspectele teoretice ale deschiderilor, urmate de exerciții practice ce simulează cele mai frecvente scenarii întâlnite în timpul jocului, când se joacă acea deschidere specifică.

Mai mult decât atât, implementarea cu succes a algoritmului MiniMax în aplicația Chess Openings demonstrează avantajele acestui algoritm, care este ușor de implementat și de înțeles, oferind rezultate eficiente și de calitate. Algoritmul MiniMax, împreună cu tehnica Alpha Beta Pruning, a fost dezvoltat până la punctul în care poate oferi soluții optime, aducându-l aproape de rezolvarea completă a jocului de șah, ceea ce îl face aproape de neînvins.

Pentru viitor, aplicația Chess Openings poate fi extinsă pentru a include suport online, permitând utilizatorilor să joace împotriva altor jucători sau chiar împotriva prietenilor lor, creând astfel un club social virtual. De asemenea, pot fi implementate motoare de șah de diferite niveluri, adaptate fiecărui tip de utilizator în funcție de nivelul și experiența acestuia, oferind astfel o experiență personalizată și progresivă de învățare.

Această lucrare a reușit să implementeze și să dezvolte cu succes aplicația Chess Openings, ca o platformă orientată spre utilizator pentru învățarea deschiderilor și principiilor de șah.

Implementarea algoritmului MiniMax pentru dezvoltarea și implementarea inteligenței artificiale de șah, împreună cu cercetarea teoretică aprofundată, a condus la crearea unui motor de șah performant. Contribuțiiile acestei lucrări au avut implicații semnificative în dezvoltarea utilizării

Capitolul 6 - Concluzii

algoritmului MiniMax, demonstrând simplitatea și eficiența acestuia în crearea și menținerea motorului de șah, precum și în alte tipuri de jocuri strategice.

Cu cercetări și dezvoltări viitoare, aplicația Chess Openings poate continua să evolueze, având potențialul de a aduce un impact pozitiv considerabil în domeniul educației șahului. Aceasta poate deveni un instrument esențial pentru jucătorii de toate nivelurile, contribuind la creșterea și rafinarea abilităților lor strategice și tactice.

Bibliografie

- [AKMP18] Anushka, Nair., Kanksha, Marathe., Suvarna, Pansambal. (2018). Literature Survey of Chess Engines. International journal of engineering research and technology, 5(1).
- [Appt] A possible game tree, <https://html.scirp.org/file/1-9601415x4.png>.
- [Ceba23] (2023). Chess Engine Based on AI: A Survey. Indian Scientific Journal Of Research In Engineering And Management.
- [DPK23] Dinesh, Parthasarathy., G., Kontes., Axel, Plinge., Christopher, Mutschler. (2023). C-MCTS: Safe Planning with Monte Carlo Tree Search.
- [FOG22] Felipe, Oliveira, Gutierrez., Joseph, B., Ottinger. (2022). Introducing Spring Framework 6.
- [GAS23] Garg, A., Shrotriya, A. (2023). Chess Board: Performance of Alpha–Beta Pruning in Reducing Node Count of Minimax Tree. In: Senju, T., So–In, C., Joshi, A. (eds) Smart Trends in Computing and Communications. SMART 2023. Lecture Notes in Networks and Systems, vol 645. Springer, Singapore.
- [GOR18] Gerard, O'Regan. (2018). Java Programming Language.
- [Jdk21] jdk, JDK 21 <https://openjdk.org/projects/jdk/21/>.
- [Jdk22] jdk, JDK 22 <https://openjdk.org/projects/jdk/22/>.
- [JM23] Janakiraman, Manokaran. (2023). Introduction to HTML. 31-39.
- [JSW21] James Shore and Shane Warden. The art of agile development. "O'Reilly Media, Inc.", 2021.
- [LJT21] Liming, Sun., Jingbo, Wang., Lan, Tang. (2021). A Powerful Bio-Inspired Optimization Algorithm Based PV Cells Diode Models Parameter Estimation. Frontiers in Energy Research.
- [LPV20] Jaydeep, Patel., Vimal, Savsani., Vivek, Patel., Rajesh, Patel. (2020). Exploring the Effect of Passing Vehicle Search (PVS) for the Wind Farm Layout Optimization Problem.
- [LR23] Levy Rozman, How to Win at Chess: The Ultimate Guide for Beginners and Beyond, 24 Octombrie 2023, Ten Speed Press.
- [Magt] Minimax Algorithm in Game Theory | Set 4 (Alpha-Beta Pruning) - GeeksforGeeks, https://media.geeksforgeeks.org/wp-content/uploads/MIN_MAX3.jpg.

- [MAI] Minimax Algorithm in Artificial Intelligence – ITYUKTA,
<https://www.google.com/url?sa=i&url=https%3A%2F%2Fnitk.acm.org%2Fblog%2F2022%2F01%2F16%2Fminimax-algorithm-in-artificial-intelligence%2F&psig=AOvVaw2nW6cfceFXU-E7iJXLRniL&ust=1712576365893000&source=images&cd=vfe&opi=89978449&ved=0CBIQjRxqFwoTCKiW4aCCsIUDFQAAAAAdAAAAABAn>.
- [MDCB23] J. Madake, C. Deotale, G. Charde and S. Bhatlawande, "CHESS AI: Machine learning and Minimax based Chess Engine," 2023 International Conference for Advancement in Technology (ICONAT), Goa, India, 2023, pp. 1-6.
- [MJAK21] Manish, Jain., Arun, Kumar., Rajendar, Bahl. (2021). A Novel Minimax Algorithm for Multi-channel Active Noise Control System.
- [Mtcs] MCTS' pseudo-code
<https://link.springer.com/article/10.1007/s00366-021-01338-2/figures/2>.
- [Napc] NegaMax Algorithm Pseudo Code
<https://www.researchgate.net/publication/262672371/figure/fig5/AS:393455634272260@1470818541653/NegaScout-Algorithm-Pseudo-Code-Using-the-Minimal-Window-Search-Principle.png>
- [NP20] Nigel Poulton. Docker Deep Dive: Zero to Docker in a single book. NIGEL POULTON LTD, 2020.
- [NUGP21] Nilam, Upasani., Ansh, Gaikwad., Arshad, Patel., Nisha, Modani., Prashanth, Bijamwar., Sarvesh, Patil. (2021). Dev-Zero: A Chess Engine.
- [RBD22] Richard B. Darlington. The Case for Minimax-TD, 16 Septembrie 2022, PREPRINT (Version 1) available at Research Square.
- [RCH13] Robert, C., Holte. (2013). Move Pruning and Duplicate Detection.
- [SD] Spring Data
<https://www.google.com/url?sa=i&url=https%3A%2F%2Finnovationm.co%2Fspring-data-jpa%2F&psig=AOvVaw3KzsHWrWZom4hzkA9YEjfl&ust=1713511192436000&source=images&cd=vfe&opi=89978449&ved=0CBIQjRxqFwoTCIjdnuGcy4UDFQAAAAAdAAAAABAr>.
- [SMB22] Shevtkar, Sumit & Malpe, Mugdha & Bhaila, Mohammed. (2022). Analysis of Game Tree Search Algorithms Using Minimax Algorithm and Alpha-Beta Pruning. International Journal of Scientific Research in Computer Science, Engineering and Information Technology.

Bibliografie

[SMB22] Prof., Sumit, S, Shevtkar., Mugdha, Malpe., Mohammed, Bhaila. (2022). Analysis of Game Tree Search Algorithms Using Minimax Algorithm and Alpha-Beta Pruning. International journal of scientific research in computer science, engineering and information technology.

[Tce4] The chessboard with the e4-square highlighted.

https://images.chesscomfiles.com/uploads/v1/images_users/tiny_mce/ColinStapczynski/phpmgbCBz.png.

[TJS23] Toko. (2023). Introduction to JavaScript.

[VST20] Victor, Sanh., Thomas, Wolf., Alexander, M., Rush. (2020). Movement Pruning: Adaptive Sparsity by Fine-Tuning.

[WLPG23] Winfried, Lohmiller., Philipp, Gassert., Jean-Jacques, E., Slotine. (2023). MinMax Networks.

[WLY23] Wei, Li., Yi, Liu., Yanhong, Ma., Kang, Xu., Jiang, Qiu., Zhongxue, Gan. (2023). A self-learning Monte Carlo tree search algorithm for robot path planning. Frontiers in Neurorobotics.

[XYMA19] Xiyu Kang, Yiqi Wang, Yanrui Hu, Research on Different Heuristics for Minimax Algorithm Insight from Connect-4 Game, Mai 2019.