2. Prezentarea aplicației dezvoltate

În acest capitol se vor trece în revistă cerințele sistemului dezvoltat, pașii care au contribuit la implementarea aplicației, realizarea componentelor software, implementarea algoritmilor și arhitectura ce stă la baza aplicației.

2.1. Dezvoltarea cerințelor aplicației

Scopul lucrării este de a dezvolta un motor de șah capabil să caute o mutare pentru o anumită configurație a unei table de șah pentru a permite jucarea unui meci de șah. Se pune accentul pe gradul de reutilizare al componentelor dezvoltate, astfel pentru implementarea mai multor jocuri de șah se va putea folosi același motor de șah. Se vor trece în revistă pașii necesari pentru implementarea unui joc și exemplificarea unei astfel de implementări pentru platforma desktop .NET. Pentru implementarea unui astfel de sistem este necesară definirea unor valori de intrare pentru motorul de șah, pe baza cărora motorul să poată evalua pozițiile și să ofere o mutare. Se folosește notația FEN (Forsyth–Edwards Notation) pentru reprezentarea stării unei table de șah la un moment dat, scopul notației este de a oferi toate informațiile necesare pentru începerea unui meci de șah cu orice configurație corectă. O astfel de notație FEN este compusă din șase câmpuri în felul următor[1]:

1) Plasare pieselor(din perspectiva jucătorului alb). Fiecare rând este descris, pornind pe coloane în funcție de piesa care ocupă pătratul se notează piesa(P=pion, N=Cal, B=Nebun, R=Tură, Q=Regină, K=Rege pentru jucătorul alb, iar pentru jucătorul negru se folosesc litere mici) sau numărul de casuțe libere. Rândurile sunt separate folosind caraterul ”/”;
2) Culoarea care urmează să mute: ”w” - alb, ”b” - negru;
3) Posibilitatea de a face rocadă: ”K” - rocadă pe partea regelui alb, ”Q” - rocadă pe partea reginei albe, aceeași reprezentare pentru jucătorul negru folosind litere mici. Lipsa unei posibilități de rocadă se face folosind caracterul ”-”;
4) En Passant se reprezintă folosind ”-” dacă lipsește sau notația algebrică a

pătratului;

5) numărul de jumătăți de mutări de la ultima capturare al unei piese sau al unei promovări de pion;
6) numărul de mutări întregi de la începutul jocului.

Notația ”rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1” se reprezintă ca în figura de mai jos(figura 3).

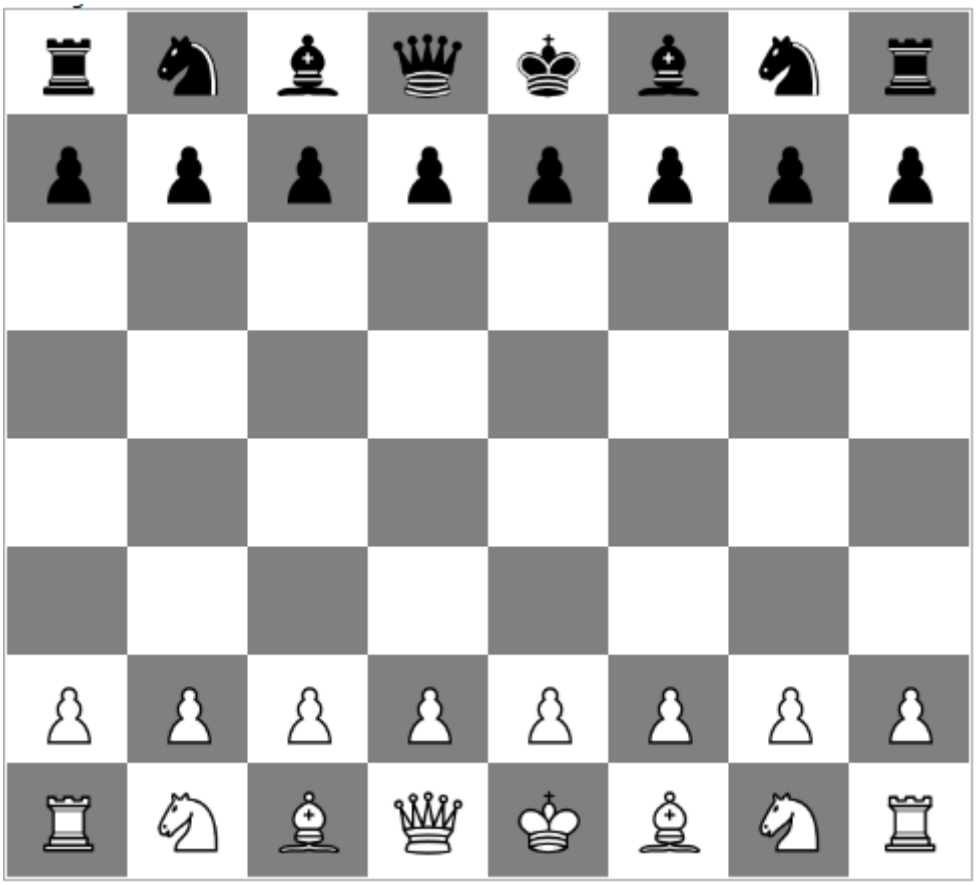

Figura 3

Configurația unei table de șah pentru începutul de meci

Aplicația trebuie să includă o interfață care să permită jucarea unui meci șah cu un adversar uman sau artificial, în funcție de preferință. Trebuie permisă posibilitatea de salvare și de continuare a unui meci. În cazul în care se dorește jucarea unui meci

cu un adversar inteligent, aplicația trebuie să permită selectarea nivelului de dificultate.

În ceea ce privește aplicația, ea trebuie să recunoască și să respecte regulile de șah pentru toate piesele și scenariile posibile. Pentru fiecare piesă trebuie permisă mutarea doar pe pozițiile legale. Trebuie identificată mutarea ce provoacă rocada și mutările ce împiedică această acțiune. În cazul în care are loc promovarea unui pion utilizatorului trebuie să i se permită alegerea piesei ce va înlocui pionul. Adversarul trebuie să aibă posibilitatea să atace poziția En Passant(figura 6), dacă ea este prezentă.

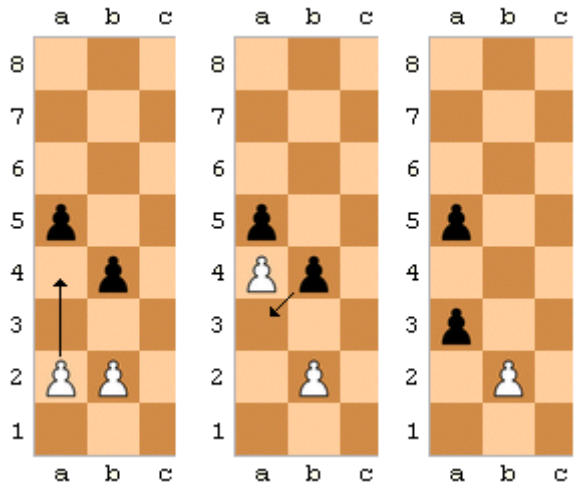

Figura 4

Poziția En Passant

Aplicația trebuie să identifice o poziție de șah și să anunțe jucătorul în cazul unei mutări neregulamentare, iar în cazul poziției de șah mat jocul trebuie încheiat și câștigătorul trebuie anunțat.

Utilizatorul aplicației are posibilitatea de a alege dacă dorește să își creeze un utilizator sau poate juca un meci ca și invitat. Creare contului se va face folosind un nume de utilizator și o parolă, pe care le va putea utiliza anterior pentru autentificare și va avea posibilitatea de a salva un meci sau de a încărca un meci deja salvat.

2. Prezentarea aplicației dezvoltate

2.2. Proiectarea sistemului

2.2.1. Motorul de șah

Motorul de căutare al unei mutări de șah este realizat folosind limbajul de programare C în mediul de dezvoltare Visual Studio Community Edition 2013. La baza implementării algorimului de căutare stă algorimul *Negamax*[4](figura 2). Căutarea *Negamax* este o variație al algorimului *Minimax*[6] care se bazează pe teoria jocului cu sumă zero între doi adversari. Pentru a simplifica implementarea algoritmului minimax, algoritmul *NegaMax* se bazează pe faptul că max(a, b) = −min(−a, −b)[7]. Astfel, valoarea unei poziții pentru un jucător A este negarea valorii jucătorului B ceea ce înseamnă că un jucător caută o mutare ce maximizează negativul valorii poziției rezultate mutării anterioare: această poziție succesor a fost, prin definiție, favorizată de adversar. Justificarea propoziției anterioare este valabilă indiferent de jucătorul al cărui rând este să mute. Asta însemnă ca o singură procedură poate fi folosită pentru a evalua ambele cazuri.

```
funcție negamax(nod, adâncime, culoare)
     dacă adâncime = 0 sau nod este nod terminal
          returnează culoare * valoarea euristică a nodului
     ceaMaiBunăValoare := -∞
     pentru fiecare copil al nod
          val := -negamax(copil, adâncime - 1, -culoare)
          ceaMaiBunăValoare := max(ceaMaiBunăValoare, val)
     returnează ceaMaiBunăValoare
```

Figura 5

Algoritmul negamax

Optimizările pentru algoritmul minimax sunt, de asemenea, la fel de aplicabile în cazul algoritmului Negamax. Alfa-beta tăiere poate reduce numărul de noduri pe care algoritmul negamax îi evaluează într-un arbore de căutare într-o manieră similară

în utilizarea sa cu algoritmul minimax. Pseudocodul pentru căutarea negamax cu adâncime limitată cu alfa-beta tăiere este prezentat în figura de mai jos(figura 6).

```
funcție negamax(nod, adâncime, α, β, culoare)
     dacă adâncime = 0 sau nod este nod terminal
          returnează culoare * valoarea euristică a nodului
     ceaMaiBunăValoare := -∞
     noduriCopii := Generează Mutări(nod)
     noduriCopii := OrdonareMutări(noduriCopii)
     pentru fiecare copil in noduriCopii
          val := -negamax(copil, adâncime - 1, -β, -α,
-culoare)
          ceaMaiBunăValoare := max(ceaMaiBunăValoare, val)
          α := max(α, val)
          if α ≥ β
               întrerupe
     returnează ceaMaiBunăValoare
```

Figura 6

Algoritmul negamax cu tăieri alfa beta

Pentru reprezentarea pătratelor tablei de șah este suficient un vector de 64 de elemente numere întregi, unde fiecare element are o valoare între 0 – 12 în funcție de piesa care se află pe pătratul respectiv. În momentul în care vrem să generăm mutări pentru piese trebuie să extindem acest vector de 64 de piese și să adăugăm o zonă care va putea identifica dacă piesa ce urmează a fi mutată este încă pe table de șah. În consecință vectorul de 8 x 8 elemente va avea două rânduri în plus înainte și după și câte o coloană în lateral(figura 7). Asta va permite o identificare mult mai ușoară a unei mutări neregulamentare. Deci vectorul va avea în total 120 de elemente, pozițiile nececare pentru identificarea mutărilor în afara tablei de șah sunt marcate cu gri în figura de mai jos. Vor fi folosite funcții de conversie pentru datele primite de la

utilizator astfel încât ele să corespundă vectorului folosit.

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89
90	91	92	93	94	95	96	97	98	99
100	101	102	103	104	105	106	107	108	109
110	111	112	113	114	115	116	117	118	119

Figura 7

Structura vectorului care reprezintă dispunerea pieselor pe o tablă de șah

Având reprezentarea tablei de șah și algorimul de căutare definit putem genera mutări pentru o anumită configurare. Mutările generate trebuie evaluate, este important pentru căutare ca mutările să fie evaluate într-o anumită ordine, astfel mutările mai „importante” să aibă prioritate la evaluare. Asta presupune faptul că în momentul în care o mutare este generată ea trebuie asociată cu un anume scor, în funcție de tipul de mutare. Capturările vor primi un scor mare, pe principiul celei mai valoroase victime cu cel mai puțin valoros atacator. Astfel un pion care atacă o regină va primi o valoare

mai mare decât un cal care atacă o regină. Se vor prioritiza și mutările care au avut succes în iterații anterioare. Motorul de șah va folosi algorimul negamax cu tăieri alfa beta împreună cu structura de 120 de element și prioritizarea mutărilor pentru a genera o mutare care are cel mai bun scor(figura 8).

```
funcție căutarePoziții(tabla)
     celMaiBunScor = negamax(tabla, -∞, ∞, adâncime, culoare)
     ceaMaiBunăMutare = mutări[0]
returnează ceaMaiBunăMutare
```

Figura 8

Algoritmul de căutare

Observăm în algorimul de deasupra faptul că funcția are o singură iterație care va merge până la adâncimea specificată. Acest algoritm poate fi optimizat prin adăugarea unui istoric al mutărilor care au cauzat tăieri alfa beta și introducerea unei iterări progresive al adâncimii, ceea ce va duce la un timp mai lung de calcul, dar va crește semnificativ calitatea mutărilor generate, deoarece acestea vor fi bazate pe experiența căutării anterioare(figura 9).

```
funcție căutarePoziții(tabla)
     pentru adâncime = 1 până la maxAdâncime
          celMaiBunScor = negamax(tabla, -∞, ∞, adâncime,
culoare)
          ceaMaiBunăMutare = mutări[0]
returnează ceaMaiBunăMutare
```

Figura 9

Algoritmul de căutare cu iterare progresivă în adâncime