Programare orientată pe obiecte

Tema - Chess

Partea a II-a

Data postării: 08.12.2025
Ultima actualizare: 08.12.2025, 12:00

Deadline: 14.01.2026 ora 23:55

Echipă temă: Carmen ODUBĂȘTEANU, Ștefan TURCU, Alexandru TUDOR

Facultatea de Automatică şi Calculatoare

Universitatea Națională de Știință și Tehnologie Politehnica București

Anul universitar 2025 - 2026

Seria CC

1 Obiective

În urma realizării acestei teme, studentul va fi capabil să:

- aplice corect principiile programării orientate pe obiecte studiate în cadrul cursului;
- utilizeze șabloane de proiectare pentru a rezolva probleme comune de design software;
- dezvolte o interfață grafică folosind Java Swing;
- transpună o problemă din viața reală într-o aplicație completă cu interfață grafică.

Șahul este un joc de strategie pe tablă, o competiție între două persoane. Fiecare jucător controlează o armată de 16 piese, care includ un rege, o regină, doi nebuni, doi cai, două turnuri (ture) și opt pioni. Jocul se desfășoară pe o tablă pătrată de 8x8, formată din 64 de pătrate alternante de culori diferite (alb și negru).

Proiectul propune crearea unui astfel de joc folosind principiile programării orientate pe obiect și funcționalitățile puse la dispoziție de limbajul de programare JAVA, aplicând noțiunile studiate în cadrul orelor de curs și laborator.

2 Reguli de joc

Obiectivul principal este de a da șah mat regelui adversarului, adică să-l pui într-o poziție în care este atacat și nu poate scăpa.

Fiecare tip de piesă se mișcă diferit:

- **King**: Se mișcă un pătrat în orice direcție. Noua poziție nu trebuie să fie ocupată de o piesă de aceeași culoare și nu trebuie să reprezinte o amenințare (să nu fie în șah).
- **Queen**: Se mișcă în orice direcție, pe oricâte pătrate. Nu poate sări peste alte piese.
- **Rook**: Se mișcă pe orizontală sau verticală, pe oricâte pătrate. Nu poate sări peste alte piese.
- **Bishop**: Se mișcă pe diagonală, pe oricâte pătrate. Nu poate sări peste alte piese.
- **Knight**: Se mișcă în formă de „L" – două pătrate într-o direcție și apoi un pătrat perpendicular pe aceasta. Calul este singura piesă care poate sări peste alte piese.
- **Pawn**: Se mișcă înainte un pătrat (sau două pătrate la prima mișcare), dar capturează piesele adversarului doar diagonal. Nu poate sări peste alte piese.

Pentru a simplifica jocul, nu vom lua în considerare mutările speciale care există în jocul original. Pentru mai multe detalii legate de regulile jocului, puteți accesa **această adresă**.

Tabla de șah are o structură matriceală, fiind împărțită în 64 de pătrate. Fiecare pătrat este identificat prin combinarea unei litere (de la „A" la „H") pentru coloane și a unui număr (de la 1 la 8) pentru rânduri. Coloanele sunt numerotate de la stânga (A) la dreapta (H), iar rândurile sunt numerotate de jos (1) în sus (8).

Pătratele de pe tabla de șah sunt alternant colorate în alb și negru, începând cu un pătrat de culoare închisă în colțul din stânga jos (A1). Această alternanță de culori este păstrată pe întreaga tablă.

În starea inițială a jocului, fiecare jucător dispune de două rânduri de piese. Primul rând (rândul 1 pentru echipa albă și rândul 8 pentru echipa neagră) include piesele majore: turnurile în colțuri, caii lângă turnuri, nebunii lângă cai, regina pe pătratul de culoarea sa și regele pe pătratul rămas. Al doilea rând (rândul 2 pentru ehipa albă și rândul 7 pentru echipa neagră) este ocupat de pionii fiecărui jucător.

Notația algebrică este folosită pentru a descrie mișcările pieselor pe tabla de șah, identificând fiecare pătrat prin litera coloanei și numărul rândului corespunzător. De exemplu, colțul din stânga jos este „A1", iar colțul din dreapta sus este „H8". Pentru a muta o piesă, spre exemplu un pion, care se află la poziția A2, la poziția A3, se va folosi notația A2-A3.

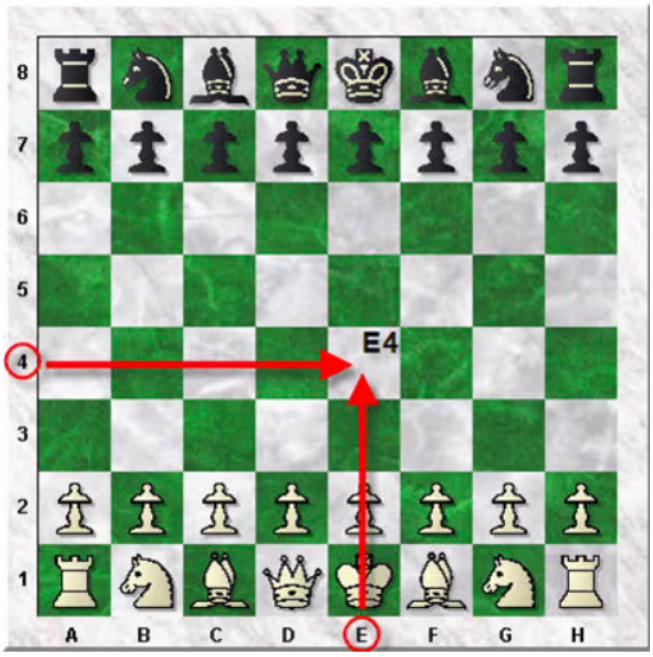

Figura 1: Reprezentarea tablei de șah

3 Flow-ul jocului

Jocul trebuie să urmărească pașii de mai jos (acești pași vor fi orchestrați, în principal, de metoda `Main.run()`, care va apela metode din clasele `User` și `Game`):

1. Utilizatorul alege în ce mod va folosi aplicația - terminal sau interfață grafică. **La testare/prezentare se va folosi doar interfața grafică. Deci puteți sări peste acest pas și să deschideți jocul direct în interfață.**
2. **Pagina de autentificare:** Utilizatorul trebuie să completeze credențialele (email și parolă) pentru a se autentifica în cont. Dacă autentificarea eșuează, va fi afișat un mesaj de eroare și utilizatorul poate reîncerca. De asemenea, aplicația trebuie să ofere posibilitatea creării unui cont nou printr-un buton dedicat.
3. **Meniul principal:** După autentificare, utilizatorul ajunge în meniul principal unde are următoarele opțiuni:
   - **Joc nou (Player vs. Computer):**
     - Se va introduce alias pentru Player și se va alege o culoare (alb/negru);
     - Se apelează metoda `Game.start()` și se deschide pagina de joc.
   - **Vizualizare jocuri în progres:**
     - Se va afișa o listă cu jocurile salvate ale utilizatorului;
     - Utilizatorul poate selecta un joc pentru a-l continua (`Game.resume()`) sau pentru a-l șterge din listă.
   - **Informații cont:** Afișarea punctajului total și a numărului de jocuri jucate.
   - **Delogare:** Revenire la pagina de autentificare.

3.1 Desfășurarea jocului

Jocul (o instanță a clasei `Game`, cu tabla de tip `Board` și jucătorii de tip `Player`) se desfășoară în felul următor:

1. Se afișează tabla de șah în interfața grafică, sub forma unui grid 8x8, din perspectiva utilizatorului (piesele utilizatorului jos, ale computerului sus). Fiecare pătrat este reprezentat grafic și conține piesa corespunzătoare (text sau icon).
2. Utilizatorul interacționează cu jocul prin interfața grafică:
   - **Selectare piesă:** click pe unul dintre pătratele care conțin o piesă proprie. Tabla va evidenția mutările posibile (de exemplu, colorând pătratele disponibile).
   - **Efectuare mutare:** click pe unul dintre pătratele evidențiate pentru a muta piesa. Mutarea este validată și executată, iar tabla se actualizează automat. Fiecare mutare este salvată în istoricul jocului prin metoda `Game.addMove()`.
   - **Renunță (Resign):** apăsarea unui buton dedicat pentru a accepta înfrângerea. Se aplică regulile de punctaj și se afișează pagina de final.
   - **Salvează și ieși:** apăsarea unui buton pentru a salva starea curentă a jocului și a reveni la meniul principal. Jocul poate fi reluat ulterior.
3. După ce jucătorul a făcut o mutare validă, **calculatorul va alege automat o piesă și o mutare random din cele posibile**, pe baza metodei `getPossibleMoves()`. Tabla se actualizează pentru a reflecta mutarea computerului, iar rândul revine la utilizator.
4. Pe parcursul jocului, interfața afișează permanent:
   - Cui îi aparține rândul curent (Player / Computer);
   - Piesele capturate de fiecare parte;
   - Punctajul curent acumulat în joc;
   - Mesaje de stare (*"You are in check"*, *"Checkmate"*, etc.).

Observații

- Toate acțiunile din joc trebuie să fie însoțite de feedback vizual și/sau mesaje explicative în interfața grafică.
- Asigurați-vă că tratați **TOATE** excepțiile care pot apărea. De exemplu, dacă utilizatorul încearcă o mutare invalidă, afișați un mesaj clar în interfață, fără oprirea neașteptată a programului.

3.2 Finalul jocului

Jocul se poate termina atunci când:

- **Jucătorul salvează și iese.** Jocul este adăugat în lista de jocuri aflate în desfășurare și poate fi reluat ulterior din meniul principal.
- **Jucătorul renunță (Resign).** Jocul este câștigat automat de computer. Se aplică penalizarea de punctaj (-150 puncte).
- **Egalitate.** Considerăm că oponentul Computer renunță atunci când se ajunge într-un caz de egalitate (aceeași poziție repetată de 3 ori consecutiv). Utilizatorul primește bonus (+150 puncte).
- **Șah-mat.** Unul dintre jucători reușește să aducă în șah-mat regele adversarului. Se aplică regulile de punctaj (+300 pentru victorie, -300 pentru înfrângere).

La finalul jocului, se afișează **pagina de final** care conține:

- Rezultatul jocului (Victorie / Înfrângere / Egalitate);
- Punctele câștigate sau pierdute în acest joc;
- Punctajul total actualizat al utilizatorului;
- Butoane pentru a reveni la meniul principal sau a închide aplicația.

3.3 Puncte acumulate în joc

Utilizatorul (presupunem că Player1 este utilizatorul autentificat și Player2 computerul) deține în cont X puncte (0 la crearea unui utilizator nou). Acestea se modifică la finalizarea fiecărui joc, după cum urmează. Notăm cu Y totalul punctelor acumulate în timpul jocului curent (din piesele capturate, conform tabelului de mai jos).

- Dacă Player2 renunță (egalitate), jocul se termină, iar la punctele actuale X ale lui Player1 se adaugă punctele acumulate în timpul jocului Y și se adaugă 150 de puncte. Dacă Player1 renunță, se scad 150 de puncte.

$$X_{\text{nou}} = X + Y \pm 150$$

(se adună $+150$ la victorie, se scad -150 la renunțare).

- Dacă Player1 aduce în șah-mat regele adversarului, jocul se termină, iar la punctele actuale X ale lui Player1 se adaugă punctele acumulate în timpul jocului Y și se adaugă 300 de puncte. Dacă Player2 aduce în șah-mat regele lui Player1, se scad 300 de puncte.

$$X_{\text{nou}} = X + Y \pm 300$$

(se adună $+300$ la victorie prin șah-mat, se scad -300 la înfrângere).

Pentru fiecare piesă capturată, se vor adăuga la punctaj sumele reprezentate în tabel:

Piesa	**Puncte**
Queen	90
Rook	50
Bishop	30
Knight	30
Pawn	10

4 Șabloane de proiectare

În dezvoltarea jocului va trebui să integrați **4 șabloane de proiectare**. Aceste șabloane sunt alese pentru a se potrivi natural cu arhitectura jocului de șah și pentru a demonstra bune practici în programarea orientată pe obiecte.

Observații

Dacă vreți să citiți mai multe despre șabloane (când se folosesc, avantaje, exemple de cod), puteți folosi această resursă: documentație despre design patterns în Java.

Imaginile utilizate pentru șabloanele de proiectare sunt preluate de pe refactoring.guru/design-patterns.

4.1 Singleton Pattern

Singleton este un design pattern din categoria **Creational Patterns**, folosit pentru a asigura că o clasă are o singură instanță și pentru a oferi un punct de acces global la acea instanță. Acest pattern este utilizat în situații în care este necesar un obiect unic care să fie accesibil de oriunde în program, precum un manager de configurare sau un punct central de control al aplicației.

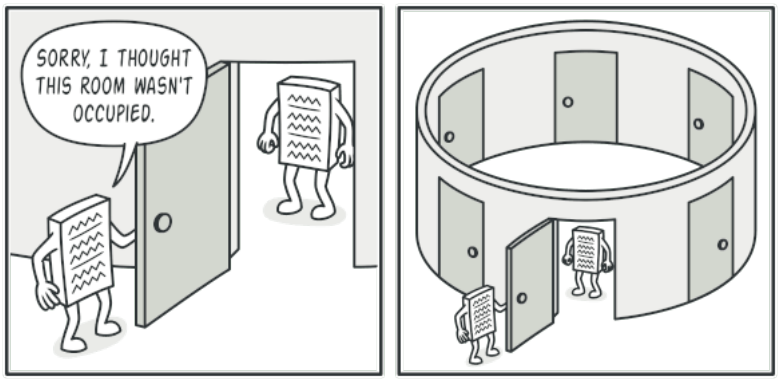

Figura 2: Acces la instanța centrală - Modelul Singleton

Toate implementările șablonului Singleton urmează aceeași structură:

- Clasa definește constructorul ca fiind **private**, astfel încât nu poate fi instanțiată din afara clasei.
- Clasa furnizează o metodă **statică** ce are ca scop returnarea instanței unice a clasei.
- Instanța unică a clasei este stocată într-o variabilă statică, ceea ce permite ca aceasta să fie accesibilă de oriunde.

Lazy initialization (sau inițializarea întârziată) este o tehnică de programare prin care crearea și inițializarea unui obiect sunt amânate până când este efectiv necesar în timpul execuției. Această abordare este folosită pentru a economisi resurse și pentru a reduce timpul de inițializare al aplicației, creând obiectele doar atunci când sunt folosite pentru prima dată.

Observații

Trebuie să utilizați acest șablon, împreună cu inițializarea întârziată, pentru a restricționa numărul de instanțieri ale clasei **Main**. Clasa Main gestionează utilizatorii, jocurile și sesiunea curentă, fiind esențial să existe o singură instanță care coordonează întreaga aplicație.

4.2 Factory Pattern

Factory este un design pattern din categoria **Creational Patterns**, folosit pentru a crea obiecte fără a specifica exact clasa de instanțiere. Este utilizat în situații în care crearea obiectelor poate fi delegată unei metode specializate, iar tipul exact al obiectului depinde de anumite condiții sau de configurațiile programului.

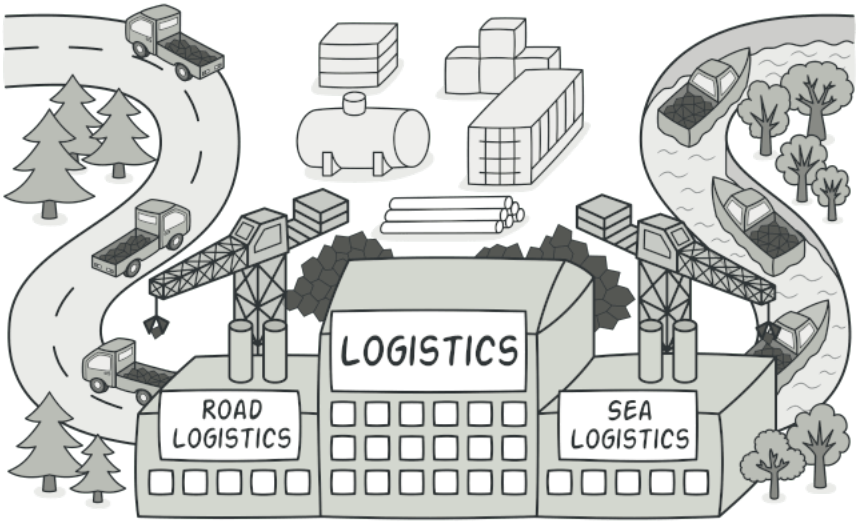

Figura 3: Crearea obiectelor fără a expune detalii - Modelul Factory

.

Observații

În cadrul jocului de șah, acest pattern este deosebit de util deoarece avem mai multe tipuri de piese (King, Queen, Rook, Bishop, Knight, Pawn) care extind aceeași clasă de bază `Piece`
Acest șablon va fi folosit pentru:

- Instanțierea pieselor la inițializarea tablei de șah (metoda `Board.initialize()`);
- Încărcarea pieselor din fișierul `games.json`;
- Promovarea pionului - când un pion ajunge pe ultima linie și trebuie transformat într-o altă piesă (Queen, Rook, Bishop sau Knight).

4.3 Strategy Pattern

Strategy este un design pattern din categoria **Behavioral Patterns**, care permite definirea unei familii de algoritmi, încapsularea fiecăruia și posibilitatea de a le interschimba. Acest pattern permite algoritmului să varieze independent de clienții care îl folosesc.

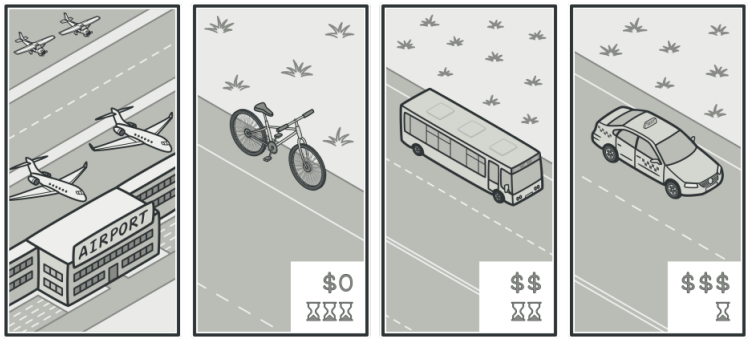

Figura 4: Mai multe strategii pentru a ajunge la aeroport - Modelul Strategy

În jocul de șah, Strategy Pattern este ideal pentru două aspecte importante:

1. Logica de mișcare a pieselor - fiecare tip de piesă are reguli diferite de deplasare. În loc să folosim lanțuri complexe de `if-else` sau `switch`, fiecare piesă utilizează propria strategie de mișcare.

2. Calculul punctajului - dferite situații necesită calcule diferite de punctaj - capturarea pieselor, câștigarea jocului, pierderea jocului, etc.

Observații

Acest șablon va fi folosit pentru:

- **Strategii de mișcare** - fiecare tip de piesă implementează propria logică de deplasare. Puteți defini o interfață `MoveStrategy` cu metoda `getPossibleMoves(Board board, Position currentPos)` și câte o implementare pentru fiecare tip de piesă (KingMoveStrategy, QueenMoveStrategy, etc.);
- **Strategii de punctaj** - pentru calculul punctelor la capturarea pieselor și la finalizarea jocului.

Exemplu pentru strategia de mișcare:

```
public interface MoveStrategy {
    List<Position> getPossibleMoves(Board board,
                                    Position from);
}
public class RookMoveStrategy implements MoveStrategy {
    @Override
    public List<Position> getPossibleMoves(Board board,
                                           Position from) {
        List<Position> moves = new ArrayList<>();
        // Mișcare pe orizontală și verticală
        int[][] directions = {{0,1}, {0,-1}, {1,0}, {-1,0}};
        for (int[] dir : directions) {
            // ... adaugă pozițiile valide
        }
        return moves;
    }
}
```

4.4 Observer Pattern

Observer este un design pattern din categoria **Behavioral Patterns**, care definește o dependență de tip one-to-many între obiecte, astfel încât atunci când un obiect își schimbă starea, toți dependenții săi sunt notificați și actualizați automat.

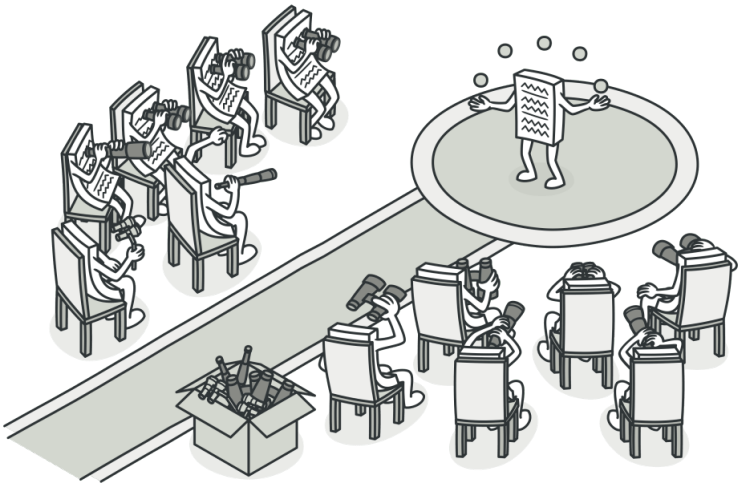

Figura 5: Notificarea automată a observatorilor - Modelul Observer

În jocul de șah, multe componente trebuie să reacționeze la schimbările de stare ale jocului: interfața grafică trebuie actualizată, istoricul mutărilor trebuie înregistrat, punctajul trebuie recalculat, iar verificările de șah/șah-mat trebuie efectuate.

Observații

Acest șablon va fi folosit pentru a notifica diferite componente atunci când:

- O mutare a fost efectuată pe tablă;
- O piesă a fost capturată;

Exemplu de implementare:

```
public interface GameObserver {
    void onMoveMade(Move move);
    void onPieceCaptured(Piece piece);
    void onPlayerSwitch(Player currentPlayer);
}
```

Observatorii pot fi: interfața grafică (pentru actualizarea afișării), un logger (pentru istoricul mutărilor), un manager de scor, etc.

5 Interfață grafică

Va trebui să realizați o interfață grafică pentru jocul de șah, folosind pachetul `Swing`. Interfața trebuie să cuprindă **obligatoriu** următoarele ecrane/pagini:

- **Pagină de autentificare (login / creare cont)**
  - Utilizatorul își introduce email-ul și parola și se autentifică în cont.
  - Din aceeași pagină se poate ajunge la crearea unui cont nou (apelând logică de tip `Main.newAccount`).
  - După autentificare, utilizatorul ajunge în meniul principal, unde poate vedea jocurile în progres și poate porni jocuri noi.
- **Meniul principal**
  - Permite:
    - începerea unui joc nou (Player vs Computer, alegerea culorii, alias-ului etc.);
    - continuarea unui joc existent din lista de jocuri asociate utilizatorului;
    - vizualizarea rapidă a unor informații despre cont (puncte totale, număr de jocuri, etc.).
- **Paginile de joc (ecranul principal de șah)**
  - Afișează tabla de șah sub forma unui **grid** 8×8, orientată din perspectiva utilizatorului (utilizatorul jos, adversarul sus).
  - Fiecare căsuță de pe tablă este reprezentată grafic (de exemplu, folosind `JButton` sau `JLabel`) și conține piesa corespunzătoare, prin text sau icon.
  - Utilizatorul poate selecta o piesă (click pe pătratul cu piesa) și apoi o poziție țintă; tabla evidențiază mutările posibile (de exemplu, prin colorarea pătratelor disponibile).
  - Sunt afișate în permanență informații utile, de exemplu:
    - cui îi aparține rândul curent (Player / Computer);
    - piesele capturate de fiecare parte;
    - punctajul curent acumulat în joc (valoarea pieselor capturate, Y);
    - eventual mesaje de stare: *"You are in check"*, *"Invalid move"*, *"Checkmate"* etc.
  - Trebuie să existe controale clare pentru acțiuni precum:
    - *Renunță* (resign);
    - *Salvează și ieși* (jocul rămâne în lista de jocuri în desfășurare);
    - *Înapoi la meniul principal*.
- **Pagina finală (rezumatul jocului)**
  - Este afișată la terminarea jocului (victorie, înfrângere sau egalitate).
  - Conține cel puțin:
    - rezultatul (de ex. *"Victorie prin șah-mat"*, *"Înfrângere"*, *"Egalitate"*);
    - punctele câștigate sau pierdute în acest joc;
    - punctajul total actualizat al utilizatorului (X_{nou});
    - opțiuni pentru întoarcerea la meniul principal sau închiderea aplicației.

Observații

- **SFATURI pentru implementarea tablei:**
  - Puteți folosi o matrice de `JButton[8][8]` pentru a reprezenta cele 64 de pătrate;
  - Alternați culorile de fundal (alb/negru sau alte nuanțe) folosind `setBackground()`;
  - Pentru piese, puteți folosi caractere Unicode (ex: U+2654 pentru regele alb) sau imagini PNG;
  - Evidențiați mutările posibile schimbând culoarea de fundal a pătratelor valide când o piesă este selectată;
  - Păstrați o referință la butonul selectat pentru a ști de unde se mută piesa.
- Pentru grafica pieselor (iconițe pentru rege, regină, pion etc.) puteți folosi seturi de iconițe gratuite sau imagini proprii. Important este ca piesele să fie ușor de diferențiat.
- Se vor puncta creativitatea și design-ul interfeței, precum și folosirea de imagini sugestive și mesaje clare pentru utilizator.
- Sunteți liberi să adăugați orice funcționalități suplimentare pe care le considerați utile din punct de vedere al interfeței grafice (ex: evidențierea ultimei mutări, cronometru simplu, log de mutări afișat într-un panou lateral etc.).
- Puteți utiliza biblioteci suplimentare pentru lucrul cu imagini sau layout-uri, dar baza interfeței trebuie să fie `Swing`. Orice lucru nou pe care îl utilizați trebuie menționat în README (ce ați folosit și în ce mod).

Atenție!

Folosirea jocului în terminal este permisă **doar** pentru acțiuni auxiliare (de exemplu, afişarea unor mesaje de debug sau selectarea modului de rulare). Interacţiunile principale ale utilizatorului cu jocul (autentificare, alegerea jocului, efectuarea mutărilor, încheierea partidei) trebuie să se poată realiza prin interfaţa grafică. Orice funcţionalitate implementată exclusiv în terminal nu va fi punctată la această secţiune.

6 Exemplu de interfață grafică

Mai jos sunt prezentate câteva exemple de interfață grafică pentru a vă oferi o idee despre cum ar putea arăta aplicația. **Nu este obligatoriu** să urmați exact acest design - puteți fi creativi și să vă creați propria interfață, atâta timp cât respectați funcționalitățile cerute.

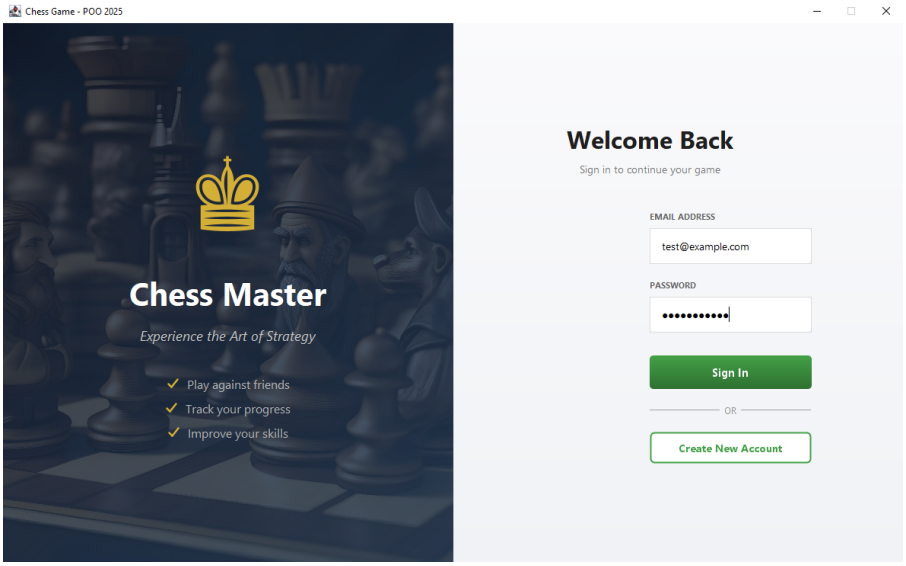

Figura 6: Pagina de autentificare - Login și înregistrare cont nou

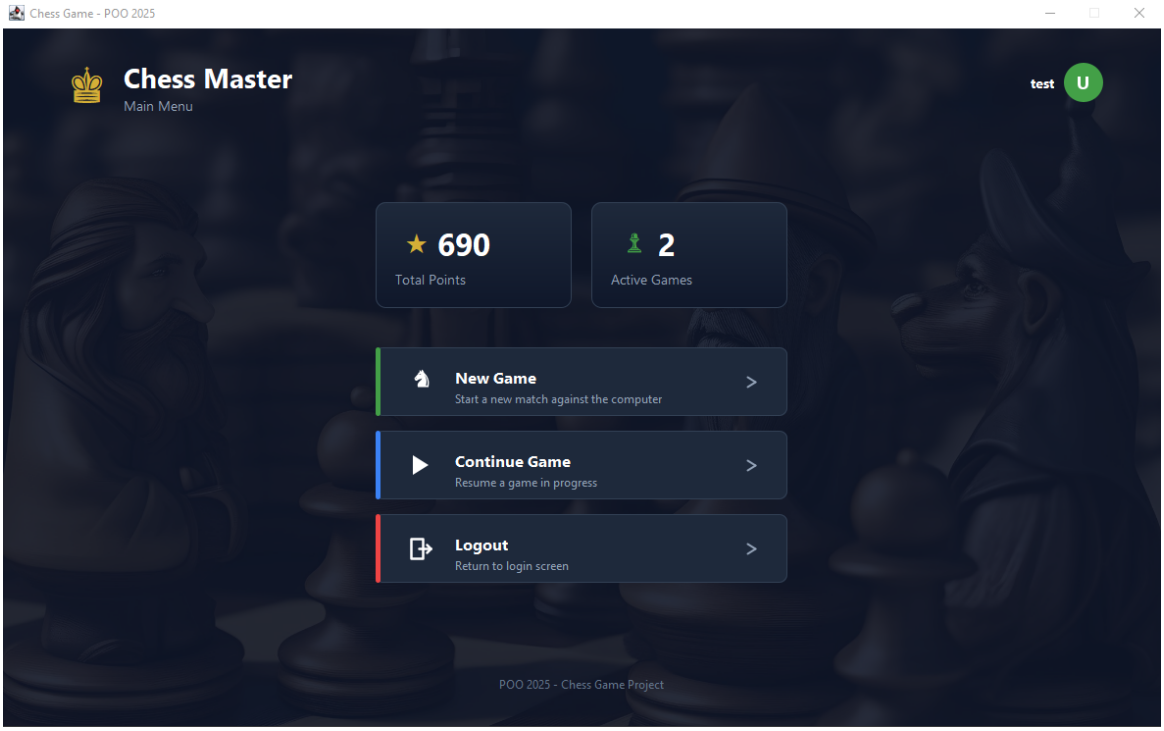

Figura 7: Meniul principal - Joc nou, continuare joc, delogare

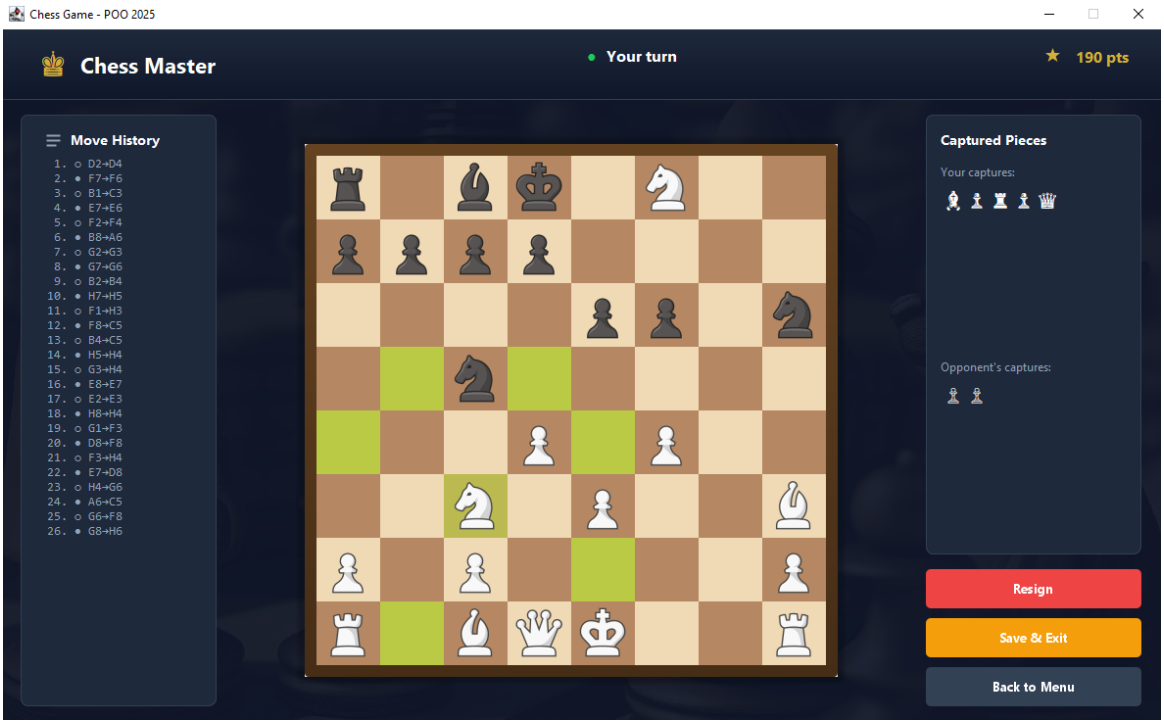

Figura 8: Pagina de joc - Tabla de șah, istoricul mutărilor, piese capturate, punctaj

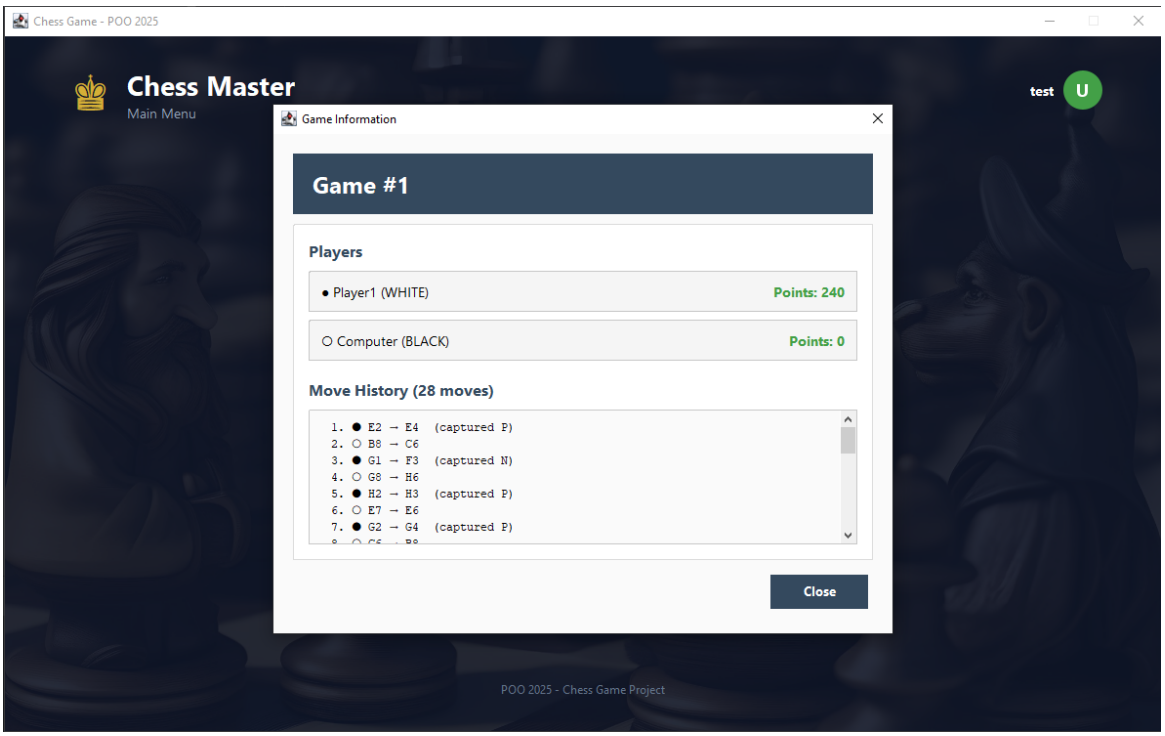

Figura 9: Informații despre un joc salvat - Jucători, punctaj, istoric mutări

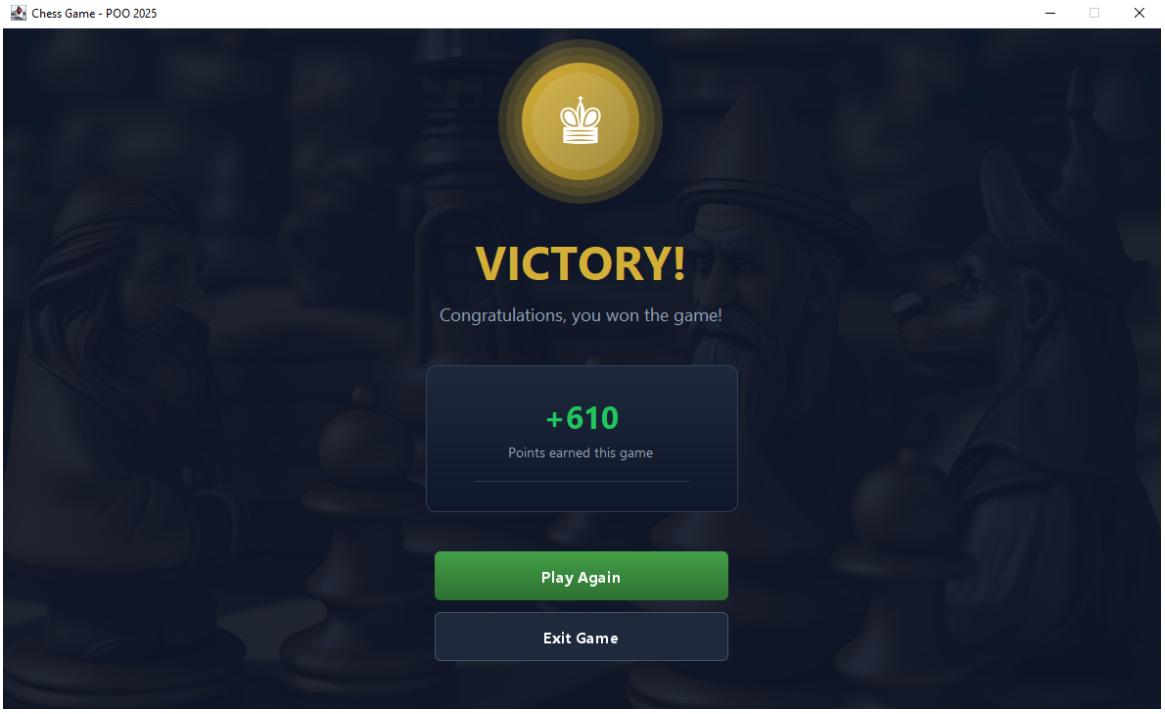

Figura 10: Pagina de final - Afișarea rezultatului și a punctelor câștigate

7 Punctaj

Șablon de proiectare	Puncte
Singleton	0.05
Factory	0.05
Strategy	0.1
Observer	0.1
Total	**0.3**

Interfață grafică	Puncte
Pagina de autentificare	0.05
Meniul principal	0.05
Pagina de joc	0.25
Pagina de sfârșit	0.05
Total	**0.4**

Atenție!

- Tema (a II-a parte) valorează **0.7 puncte** din nota finală a disciplinei POO!
- **Partea a II-a a temei este dependentă de prima parte. Nu veți primi punctaj pe partea a II-a a temei dacă prima parte nu a fost implementată.**
- Tema este **individuală**! Toate soluțiile trimise vor fi verificate, folosind o unealtă pentru detectarea plagiatului.
- **Se vor puncta DOAR funcționalitățile care pot fi testate.** Acest lucru înseamnă că va trebui să aveți o clasă de test prin care **să oferiți** posibilitatea **testării** tuturor funcționalităților pe care le-ați implementat, **altfel nu veți primi punctaj**.
- Tema se va încărca pe site-ul de cursuri până la termenul specificat în pagina de titlu. Se va trimite o arhivă **.zip** ce va avea un nume de forma **grupa_Nume_Prenume_tema2.zip** (ex. 326CC_Popescu_Andreea_tema2.zip) și care va conține următoarele:
  - un folder **SURSE** ce conține doar sursele Java;
  - un folder **PROIECT** ce conține proiectul în mediul ales de voi (de exemplu NetBeans, Eclipse, Intellij IDEA);
  - un fișier **README.pdf** în care veți specifica numele, grupa, gradul de dificultate al temei, timpul alocat rezolvării și veți specifica, pe scurt, în ce a constat dificultatea implementării.
- Lipsa fișierului README sau nerespectarea formatului impus pentru arhivă duce la o **depunctare** de **0.1** (fiecare) din punctajul temei.