UNIVERSITATEA TRANSILVANIA BRAȘOV
FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

PROIECT DE SEMESTRU

MEDII ȘI INSTRUMENTE DE PROGRAMARE

Sistem Software de Simulare Backgammon

Arhitectură Java Swing și Managementul Stărilor de Joc

STUDENT:
GRĂMADĂ
ADRIAN-GHEORGHIȚĂ

PROFESOR:
BOCU RĂZVAN

2026

Cuprins

Capitolul 1

Introducere și Analiza Domeniului

1.1 Motivarea alegerii temei

Backgammon-ul reprezintă unul dintre cele mai vechi jocuri din lume, a cărui complexitate derivă din echilibrul dintre probabilitatea matematică și decizia strategică. Alegerea acestui proiect pentru disciplina "Medii și Instrumente de Programare" se fundamentează pe necesitatea aplicării unor concepte avansate de Programare Orientată pe Obiect (POO) într-un context interactiv și dinamic.

Aplicația demonstrează modul în care un set de reguli abstracte poate fi transpus într-un sistem digital robust, capabil să gestioneze stări complexe și interacțiuni în timp real.

1.2 Istoric și Reguli Fundamentale

Sistemul implementat respectă normele internaționale de Backgammon:

- **Tabla de joc:** Compusă din 24 de puncte (triunghiuri înguste) organizate în patru cadrane.
- **Zarurile:** Determinantul numărului de spații pe care o piesă le poate parcurge.
- **Logica de "Hitting":** O piesă singură pe un punct poate fi lovită și trimisă pe bară.

Figura de mai sus ilustrează starea inițială a tablei de joc, implementată conform așezării standard din Backgammon. Punctele sunt reprezentate grafic prin triunghiuri alternante cromatic, iar piesele sunt grupate strategic: de exemplu, Albul (piesele deschise) are 5 piese pe punctul 6 (index 5) și 2 piese pe punctul 24 (index 23), în timp ce Negrul (piesele închise) are o distribuție simetrică. Central, se observă axa de simetrie a tablei, care separă cadranul interior de cel exterior.

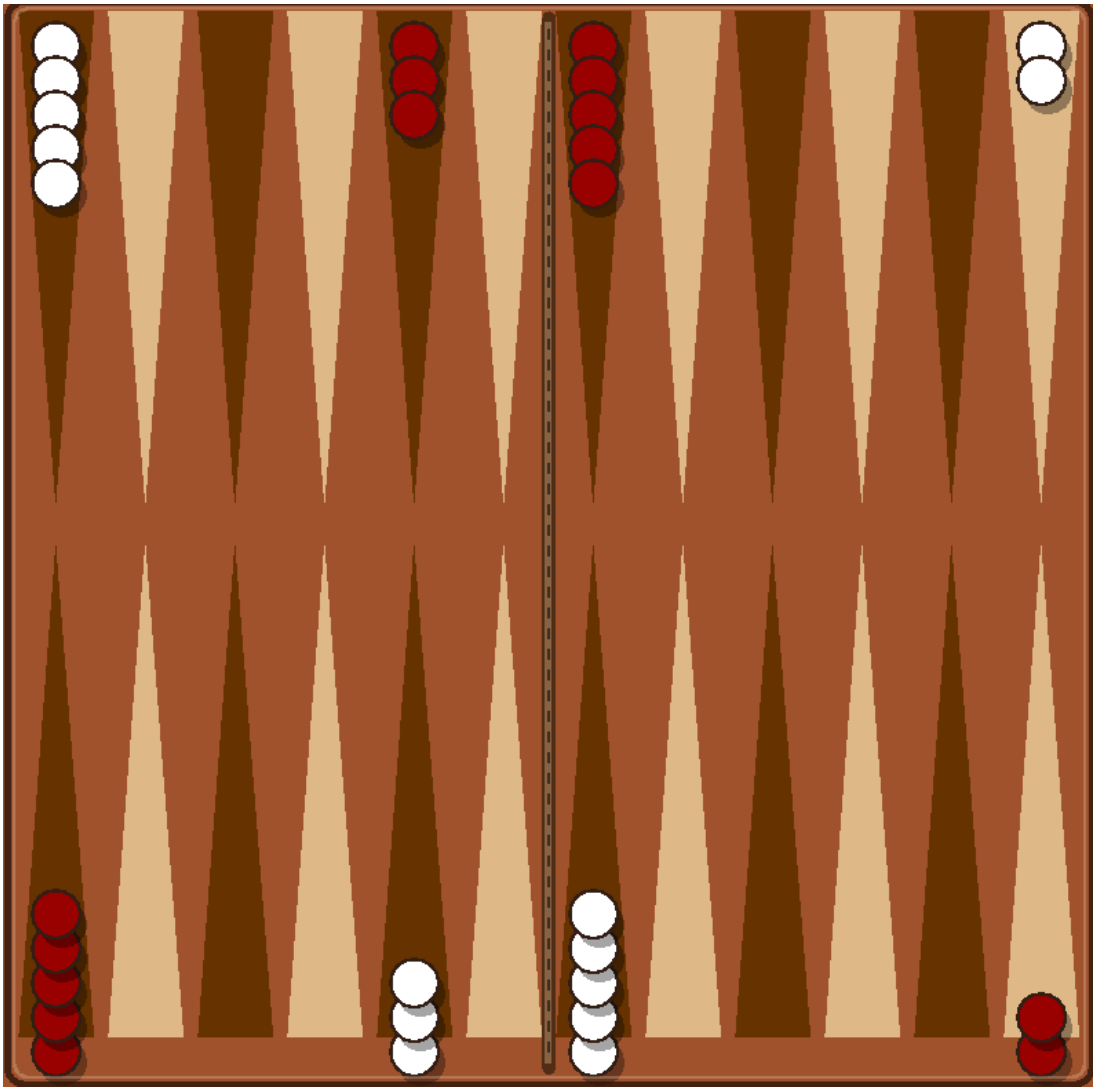

Figura 1.1: Configurația inițială a tablei de joc.

Capitolul 2

Arhitectura Sistemului Software

2.1 Paradigma Model-View-Controller (MVC)

Structura claselor reflectă separarea responsabilităților:

- **Modelul:** Clasele `Board`, `Point` și `Dice` stochează datele.
- **Vizualizarea:** `BackgammonUI` gestionează randarea grafică.
- **Controlerul:** Clasa `Game` orchestrează interacțiunea.

Figura 2.1: Panoul lateral pentru gestiunea pieselor de pe bară și a celor scoase.

Acest detaliu din interfață (Figura 2.2) prezintă panoul lateral responsabil pentru starea globală a pieselor. Acesta afișează în timp real numărul de piese aflate pe bară (cele lovite) și numărul de piese scoase de pe tablă (born off) pentru fiecare jucător.

Butonul „Select Bar” permite jucătorului să își selecteze piesele lovite pentru a reintra în joc, iar „Bear Off” este utilizat în faza finală pentru a elimina piesele din casă.

2.2 Gestiunea Stărilor (State Management)

Sistemul monitorizează în permanență starea curentă a jocului. Bara de control superioară afișează jucătorul curent și starea zarurilor.

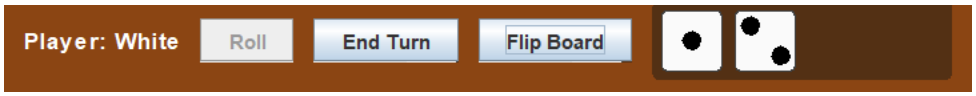

Figura 2.2: Interfața de control superioară cu zarurile afișate grafic.

În Figura 2.3 se observă modul în care este comunicată starea curentă a rundei. Jucătorul activ este indicat textual („Player: White”), iar valorile zarurilor sunt randate grafic sub forma unor cuburi clasice. Zarurile gri indică faptul că acestea au fost deja utilizate sau nu au fost încă aruncate, în timp ce zarurile complet vizibile indică mutările disponibile.

Capitolul 3

Implementarea Componentelor Logice

3.1 Logica Tablei și a Punctelor

Fiecare punct este o instanță a clasei `Point`. Metoda `add` asigură că nu se pot amesteca piese de culori diferite pe același punct, cu excepția situației de "hit".

3.2 Algoritmul de Calcul al Mutărilor

Calculul indexului de destinație este influențat de setarea `whiteMovesIncreasing`.

$$toIndex = \begin{cases} fromIndex + die, & \text{player} = 1 \\ fromIndex - die, & \text{player} = \text{-1} \end{cases} \tag{3.1}$$

3.3 Logica Barei (Bar)

Piesele lovite sunt trimise pe bară și blochează orice altă mutare a jucătorului până când acestea reintră în joc.

Figura 3.1 demonstrează situația în care o piesă Albă a fost lovită și se află în zona centrală (Bar). În acest stat, logica aplicației restricționează selectarea oricărei alte piese de pe tablă, forțând jucătorul să folosească butonul „Select Bar" pentru a încerca reintrarea în casa adversarului folosind zarurile disponibile.

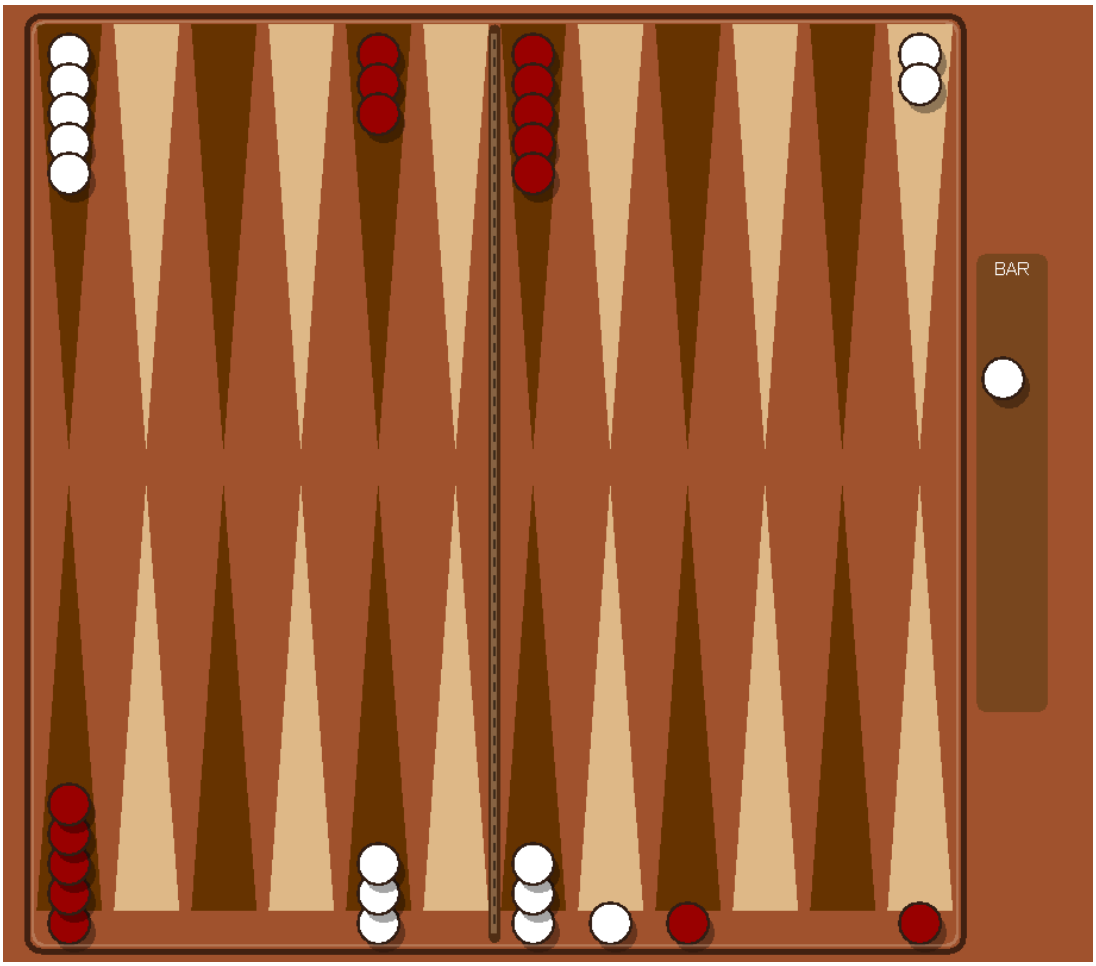

Figura 3.1: Vizualizarea unei piese albe aflată pe bară.

Capitolul 4

Proiectarea Interfeței cu Utilizatorul

4.1 Randarea Custom cu Graphics2D

`BoardCanvas` desenează triunghiurile folosind obiecte de tip `Polygon`. Această abordare permite redimensionarea tablei fără pierderea calității vizuale.

4.2 Interacțiunea prin MouseListener

La selectarea unei piese, aplicația marchează destinațiile legale cu puncte verzi.

În Figura 4.1 este prezentat sistemul de feedback vizual pentru utilizator. La selectarea unei piese albe (evidențiată printr-un contur roșu în jurul triunghiului sursă), motorul de joc calculează automat toate pozițiile de destinație permise pe baza zarurilor curente. Acestea sunt marcate cu buline verzi în interiorul triunghiurilor țintă, simplificând decizia jucătorului și prevenind mutările invalide.

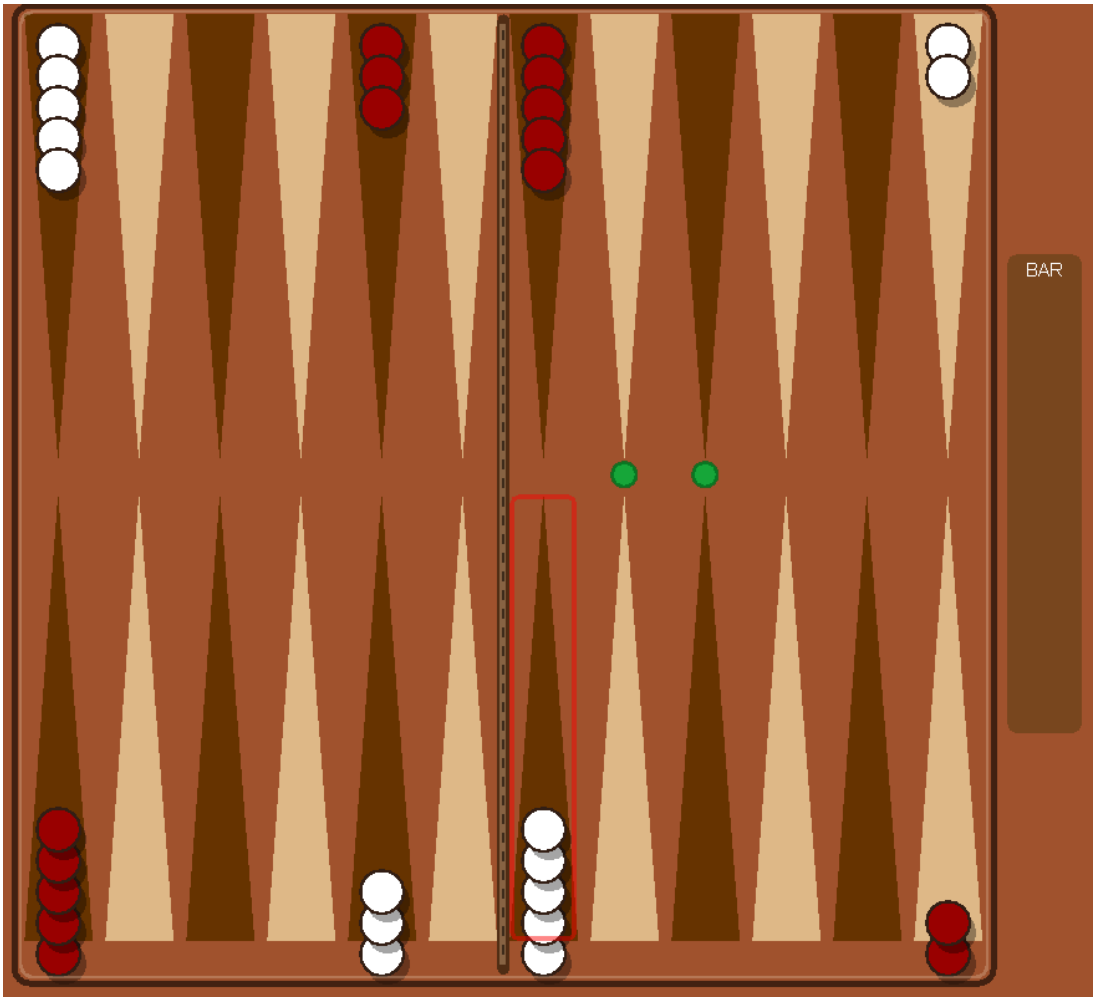

Figura 4.1: Evidențierea mutărilor posibile (puncte verzi) după selectarea unei piese.

Capitolul 5

Manual de Utilizare și Feedback

5.1 Efectuarea unei mutări

Utilizatorul trebuie să apese pe **"Roll"** pentru a activa runda.

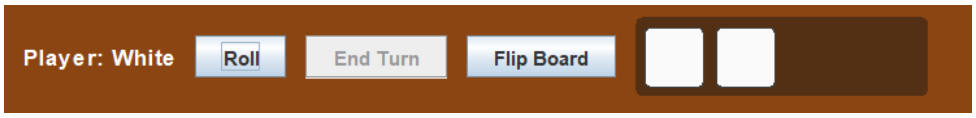

Figura 5.1: Starea interfeței înainte de aruncarea zarurilor.

Imaginea de mai sus (Figura 5.1) prezintă starea de așteptare a sistemului. Butonul „Roll" este activat, în timp ce „End Turn" este dezactivat, forțând ordinea logică a jocului. Utilizatorul poate observa cine urmează să mute, dar nu poate interacționa cu piesele de pe tablă până când zarurile nu generează valorile de mișcare.

5.2 Finalul jocului

Când un jucător scoate toate cele 15 piese, sistemul afișează un dialog de confirmare și blochează mutările ulterioare.

Figura 5.2 surprinde momentul finalizării unei partide. Un dialog de tip JOptionPane informează jucătorii despre rezultat („White wins!"). În acest punct, logica de joc trece în starea „gameOver", dezactivând butoanele de acțiune pentru a marca sfârșitul sesiunii curente.

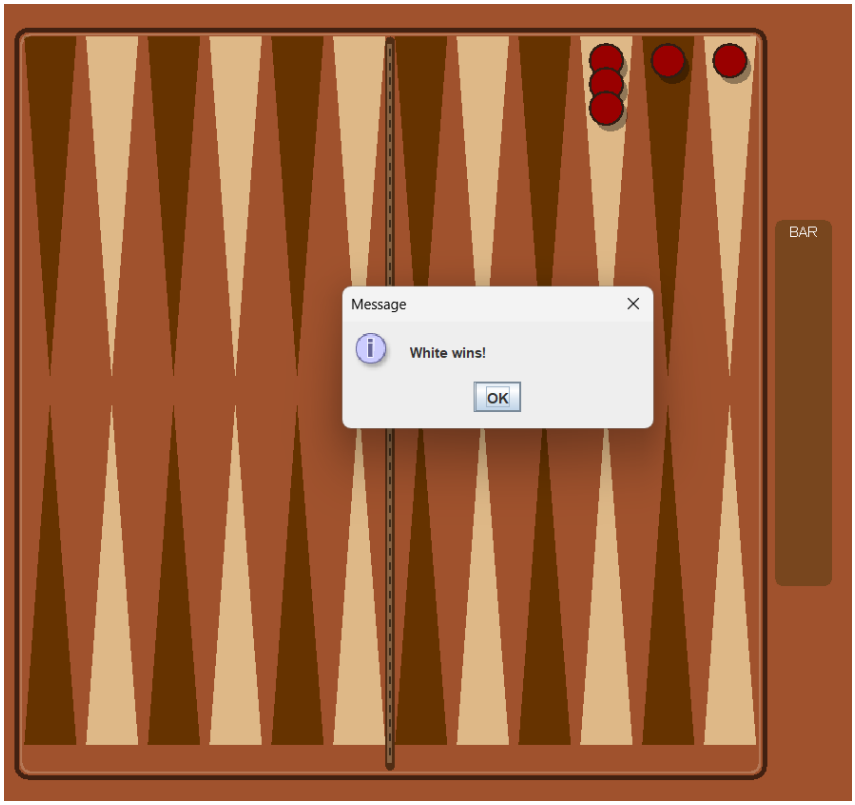

Figura 5.2: Mesaj de confirmare a victoriei jucătorului alb.

Capitolul 6

Bibliografie și Resurse

[1] **Oracle Java SE 25 Documentation**: https://docs.oracle.com/en/java/javase/25/

[2] **Java Swing API Reference**: https://docs.oracle.com/javase/8/docs/api/javax/swing/package-summary.html

[3] **Tesauro, G. - TD-Gammon Research**: https://bkgm.com/articles/tesauro/tdgammon.html

[4] **USBGF - Official Backgammon Rules**: https://usbgf.org/learn-backgammon/rules-of-backgammon/

[5] **Maven Project Documentation**: https://maven.apache.org/guides/index.html

[6] **GeeksforGeeks - Game Theory**: https://www.geeksforgeeks.org/minimax-algorithm-in-

Capitolul 7

Analiza Codului Sursă

În acest capitol sunt analizate componentele software critice care asigură funcționarea sistemului Backgammon.

7.1 Gestiunea Punctelor Individuale

Clasa `Point.java` este responsabilă pentru încapsularea stării unui singur triunghi de pe tablă. Aceasta reține proprietarul curent și numărul de piese.

```
        public void add(int owner) {
            if (count == 0) {
                this.owner = owner;
                this.count = 1;
            } else if (this.owner == owner) {
                this.count++;
            }
        }

```

Listing 7.1: Metoda de adăugare a pieselor în clasa Point

Logica de mai sus previne adăugarea directă a unei piese peste una adversă fără a trece prin logica de „hit” din clasa `Board`.

7.2 Logica Centrală a Tablei

Clasa `Board.java` implementează regulile de bază ale jocului. Metoda `isMoveLegal` verifică validitatea unei mutări înainte ca aceasta să fie executată.

```
        public boolean allInHome(int player) {
            int inHome = 0;
            if ((player == 1) == whiteMovesIncreasing) {
                for (int i = 18; i <= 23; i++)
                if (points[i].owner == player) inHome += points[i
    ].count;
            } else {
                for (int i = 0; i <= 5; i++)
                if (points[i].owner == player) inHome += points[i
    ].count;
```

```
            }
            int bornOff = (player == 1) ? whiteBornOff :
    blackBornOff;
            int bar = (player == 1) ? whiteBar : blackBar;
            return (inHome + bornOff) == 15 && bar == 0;
        }

```

Listing 7.2: Verificarea condiției pentru scoaterea pieselor

Această funcție este crucială deoarece condiționează faza de final („bearing off"), asigurându-se că toate cele 15 piese ale jucătorului sunt în zona casei și că nicio piesă nu a rămas blocată pe bară.

7.3 Controlul Fluxului de Joc

Clasa `Game.java` acționează ca un mediator. Ea calculează destinațiile legale pentru o selecție dată, combinând logica tablei cu zarurile disponibile.

```
        public List<Integer> getLegalDestinations(int fromIndex)
    {
            List<Integer> res = new ArrayList<>();
            if (!dice.hasMoves()) return res;
            for (int d : dice.getRemaining()) {
                int toIndex = board.getToIndex(currentPlayer,
    fromIndex, d);
                if (board.isMoveLegal(currentPlayer, fromIndex, d
    )) {
                    res.add(toIndex < 0 || toIndex > 23 ? -2 :
    toIndex);
                }
            }
            return res;
        }

```

Listing 7.3: Calculul destinațiilor legale

Această metodă permite interfeței grafice să randeze bulinele verzi menționate anterior, oferind o experiență de utilizare fluidă și corectă.

7.4 Interfața Grafică și Maparea Coordonatelor

Clasa `BackgammonUI.java` conține logica de randare. Un aspect tehnic important este maparea indexului logic al punctului la ordinea vizuală, care se poate schimba prin funcția „Flip Board".

```
        private int areaIndexToGameIndex(int areaIndex) {
            if (!flip) {
                if (areaIndex < 12) return 23 - areaIndex;
                return areaIndex - 12;
            } else {
```

```
6                   if (areaIndex < 12) return 12 + areaIndex;
7                   return 11 - (areaIndex - 12);
8               }
9           }
10
```

Listing 7.4: Inversarea vizuală a indexării