Dame-Spiel Programmierung

Caius-Ange Tawotchuen , Technische Hochschule Mittelhessen

Inhaltsverzeichnis

1. Allgemeine Einführung
2. Beschreibung der Hauptkomponenten
    - Spiellogik: Hauptklasse Spielfeld
    - Grafische Darstellung: Klasse DameView
    - Spielsteuerung: Klasse DameSpiel
3. Erklärung der wichtigsten Methoden
    - Methoden in DameSpiel
    - Methoden in Spielfeld
    - Methoden in DameView
4. Datenstruktur und Spielmechanik
5. KI und Minimax-Strategie
6. Ausführung und Benutzerinteraktion
7. Features

---

1. Allgemeine Einführung

Dieses Projekt simuliert ein Dame-Spiel mit zwei Hauptaspekten:

Spiellogik (Regeln, Bewegung, Punktestand): Implementiert in der Klasse Spielfeld.

Grafische Darstellung (Brett und Spielfiguren): Implementiert in der Klasse DameView, die Turtle-Grafik verwendet.

Steuerung und Integration: Die Klasse DameSpiel verbindet die Spiellogik mit der Darstellung und ermöglicht Interaktionen.

2. Beschreibung der Hauptkomponenten

a. Spiellogik: Hauptklasse Spielfeld

Zentrale Verwaltung des Spielfeldes und der Spielregeln.

Attribute :

feld: Ein 1D-Array, das die Felder des Spielfeldes darstellt (64 Felder).

spieler1, spieler2: Die beiden Spieler.

turn: Bestimmt, welcher Spieler am Zug ist (1 für weiß, -1 für schwarz).

Wichtige Methoden:

initialisieren(): Initialisiert das Spielfeld.

bewegenAusuehren(int startIndex, int zielIndex): Führt eine Bewegung aus.

steinBewegen(int startIndex, int zielIndex): Bewegt eine Spielfigur oder führt einen Schlagzug aus.

zeigeZugMoeglichkeit(int startIndex, int zielIndex): Listet alle möglichen Züge auf.

wechselSpieler(): Wechselt den aktiven Spieler.

istSpielAmEnde(): Überprüft, ob das Spiel beendet ist.

b. Grafische Darstellung: Klasse DameView

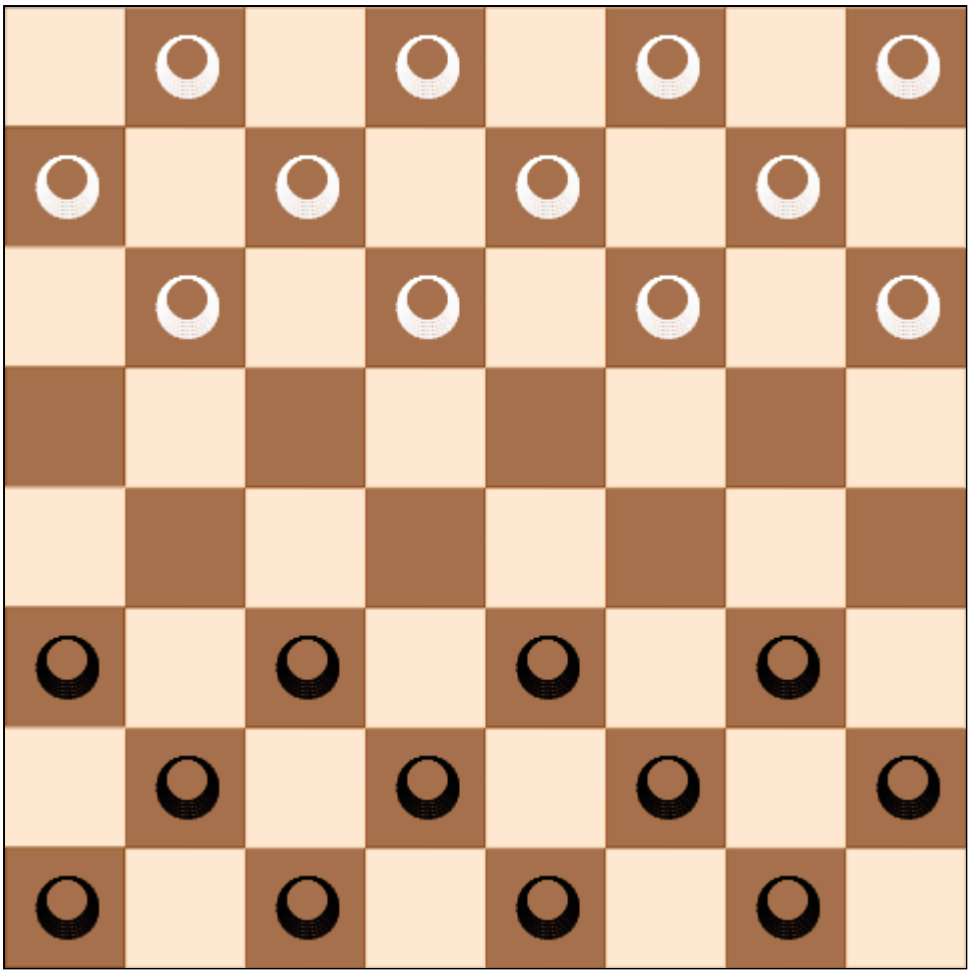
Zeichnet das Spielfeld und die Spielfiguren mithilfe einer Turtle-Grafikbibliothek.

Hauptfunktionen:

-Zeichnen des Spielfelds mit hellbraunen und dunklen Quadraten.

-Darstellung von Spielfiguren (Steine und Damen) in schwarz und weiß.

-Dynamische Aktualisierung des Bretts nach jeder Bewegung.



Spieler 1: Mensch	Punkte: 0	Steine: 12
Spieler 2: KI	Punkte: 0	Steine: 12

c. Spielsteuerung: Klasse DameSpiel

Verbindet die Spiellogik (Spielfeld) mit der Darstellung (DameView).

Verantwortlich für:

-Initialisierung des Spiels.

-Verarbeitung von Spielerbewegungen.

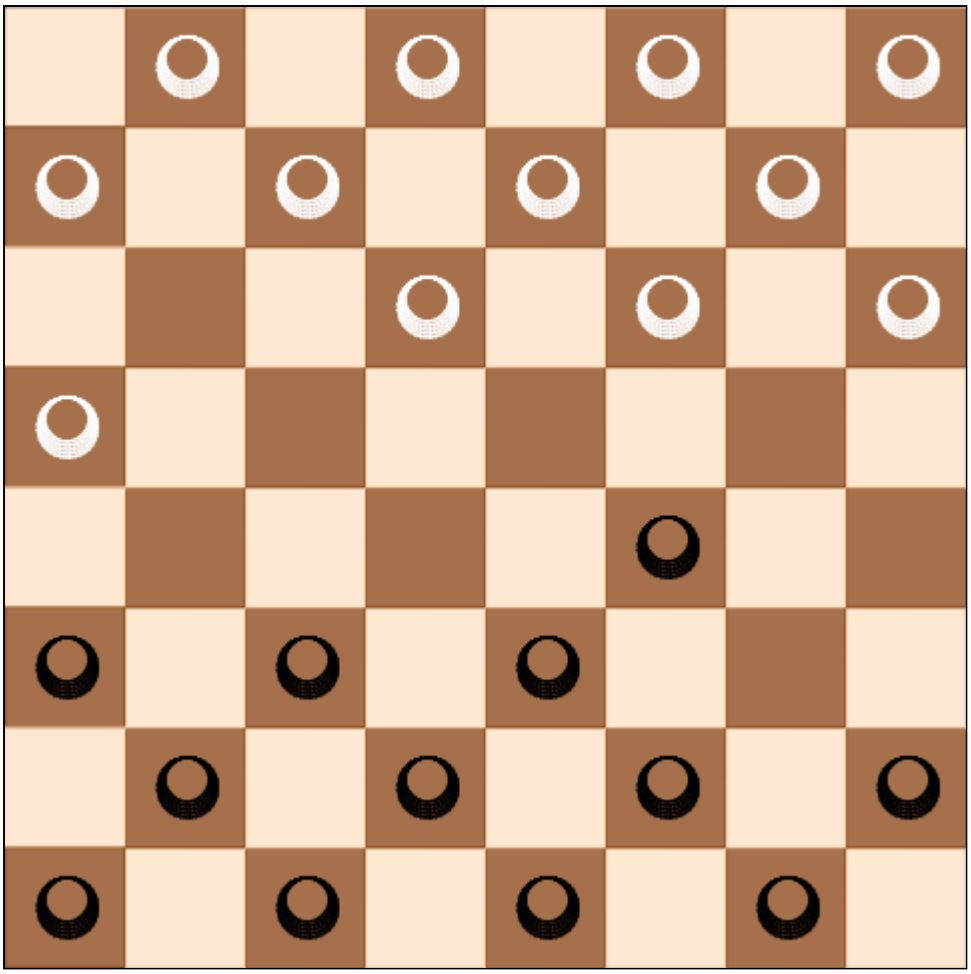
-Aktualisierung der grafischen Darstellung basierend auf Änderungen im Spielfeld.

3. Erklärung der wichtigsten Methoden

a. Methoden in DameSpiel

steinBewegenAusfuehren(int startIndex, int zielIndex):Führt eine Bewegung auf dem Spielfeld aus,ruft bewegenAusuehren aus Spielfeld auf, um die Logik zu validieren. und aktualisiert anschließend die grafische Darstellung durch druckeBrett

```java
public void steinBewegenAusfuehren(String startIndex, String z
    int start = konvertierePosition(startIndex );
    int ziel = konvertierePosition(zielIndex);

    spielfeld.bewegenAusfuehren(start, ziel);
    this.view.druckeBrett(this.spielfeld);
}

public int konvertierePosition(String position) {
    if (position == null || position.length() < 2) {
        throw new IllegalArgumentException("Ungültige Positior
    }

    char spalteChar = position.charAt(0); // Buschstabe
    char zeileChar = position.charAt(1); // Zahl

    int spalte = Character.toLowerCase(spalteChar) - 'a';
    if (spalte < 0 || spalte >= 8) {
        throw new IllegalArgumentException("Ungültige Spalte:
    }

    int zeile = Character.getNumericValue(zeileChar) - 1;
    if (zeile < 0 || zeile >= 8) {
        throw new IllegalArgumentException("Ungültige Zeile: '
    }

    return zeile * 8 + spalte;
}

public String toString() {
    return spielfeld.toString();
}
```



b. Methoden in Spielfeld

-Initialisierung

initialisieren(): Platziert die Figuren auf dem Spielfeld. Die ersten drei und letzten drei Reihen enthalten die jeweiligen Figuren.

-Bewegungsverwaltung

bewegenAusuehren(int startIndex, int zielIndex): Überprüft und führt eine Bewegung aus. Wenn die KI am Zug ist, wird die Methode MinimaxSpieler.findeBestenZug(Spielfeld spielfeld) aufgerufen. Handhabt auch Mehrfachschläge.

```
public boolean bewegenAusfuehren(int startIndex, int zielIndex
    // Ki ist dran
    if (getAktuellerSpieler().isKI()) {
        System.out.println("KI ist dran...");
        int[] besterZug = MinimaxSpieler.findeBestenZug(this);
```

```
        if (besterZug != null) {
            try {
                steinBewegen(besterZug[0], besterZug[1]);
                String startPosition = indexZuString(besterZug
                String zielPosition = indexZuString(besterZug
                System.out.println("KI zieht von " + startPosi
                wechselSpieler();
                System.out.println(this);
                if (istSpielAmEnde()) {
                    System.out.println("Das Spiel ist bereits

                    if (getGewinner().getPunkte() > best.getBe
                        best.punktestandSpeichern(getGewinner
                        System.out.println("Herzlichen Glueckw
                    }
                }
                return true;
            } catch (IllegalArgumentException e) {
                System.out.println("KI-Fehler: " + e.getMessag
                return false;
            }
        } else {
            System.out.println("Die KI kann keinen gueltigen Z

            return false;
        }
    }
    // Mensch ist dran
    else {
        try {
            steinBewegen(startIndex, zielIndex);
            wechselSpieler();
            if (!istSpielAmEnde() && getAktuellerSpieler().isK
                return bewegenAusfuehren(0, 0);
            }
            System.out.println(this);
            if (istSpielAmEnde()) {
                System.out.println("Das Spiel ist bereits been

                if (getGewinner().getPunkte() > best.getBester
                    best.punktestandSpeichern(getGewinner().ge
                    System.out.println("Herzlichen Glueckwunsc
                }
            }
```

```
                return true;

            } catch (IllegalArgumentException e) {
                System.out.println("Ungueltiger Zug: " + e.getMes
                return false;
            }
        }
    }
```

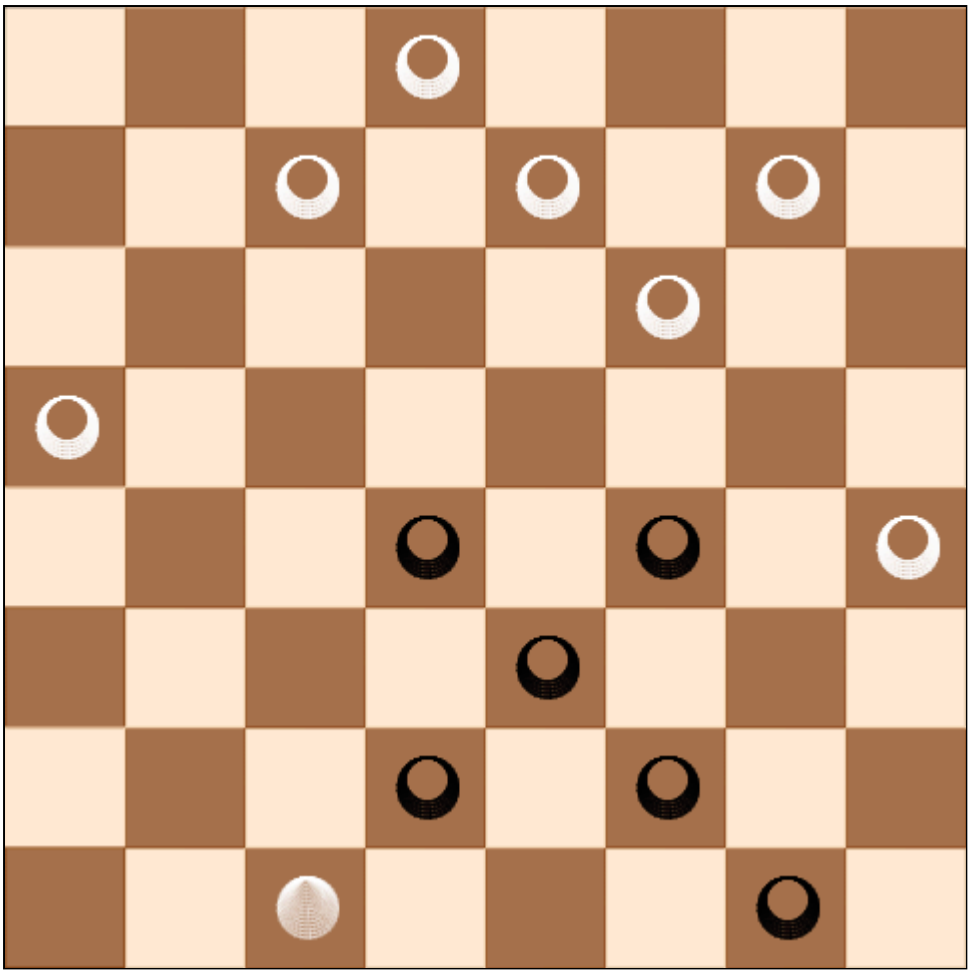
-Schlagverwaltung

steinSchlagen(int startIndex, int zielIndex) : ist dafür verantwortlich, zu prüfen und auszuführen, ob eine Spielfigur (ein Stein oder eine Dame) einen gültigen Schlagzug durchführen kann. Ein Schlagzug bedeutet, dass eine gegnerische Figur übersprungen und entfernt wird.

```
    public boolean steinSchlagen(int startIndex, int zielIndex) {
        int stein = feld[startIndex];
        boolean istKoenigen = Math.abs(stein) == 2;
        boolean schlag = false;
        if (istKoenigen) {
            schlag = dameSchlagen(startIndex, zielIndex);
        } else {
            schlag = normalerSteinSchlagen(startIndex, zielIndex);
            if (schlag) {
                dameBeforderungPruefen(zielIndex);
            }
        }

        return schlag;
    }
```

-Dame-Beförderung

dameBeforderungPruefen(int zielIndex): Überprüft, ob eine Figur das gegenüberliegende Ende des Spielfelds erreicht und zur Dame befördert wird.



Spieler 1: Mensch	Punkte: 24	Steine: 8
Spieler 2: KI	Punkte: 16	Steine: 6

c. Methoden in DameView

Verwendet zeichneStein und zeichneKoenigen, um normale Figuren und Damen darzustellen.

-zeichneBrett(Turtle t, int quadratGroesse)

Zeichnet ein 8x8-Brett mit abwechselnd hellbraunen und dunklen Quadraten. Setzt das Farbschema für die Quadrate: Hellbraun (255, 228, 196) und Dunkelbraun (139, 69, 19).

-zeichneStein(Turtle t, int index, int quadratGroesse, int r, int g, int b) und zeichneKoenigen(Turtle t, int index, int quadratGroesse, int r, int g, int b)

Zeichnen Spielfiguren als Kreise und Farbparameter definieren, ob die Figur schwarz oder weiß ist.

-druckeBrett(int[] brettZumDrucken)

Zeichnet das Spielfeld und die Spielfiguren basierend auf dem aktuellen Spielstatus. Unterscheidet zwischen: Schwarzen Figuren (-1, -2). Weißen Figuren (1, 2).

```
void druckeBrett(Spielfeld spielfeld) {
    if (turtle == null) {
        turtle = initTurtle();
        zeichneInfo(spielfeld);

    } else {
        turtle.reset();
        turtle = initTurtle();
    }
    if(!spielfeld.istSpielAmEnde()){
        infoTurtle.reset();
        zeichneInfo(spielfeld);
    }else{
        zeichneInfo(spielfeld);
    }
    zeichneBrett();
    steinSetzen(spielfeld);
}
```

4. KI und Minimax-Strategie

Der Minimax-Algorithmus:

Durchsucht alle möglichen Spielzüge bis zu einer vorgegebenen Tiefe (MAX_TIEFE). Bewertet Spielzustände mithilfe der Methode bewerteStellung, wobei berücksichtigt wird: Anzahl verbleibender Figuren. Nähe zu Beförderungen. Schutz durch verbündete Figuren.

- ## Klasse MinimaxSpieler

Implementiert die KI mithilfe des Minimax-Algorithmus.

Attribute:

MAX_TIEFE: Maximale Suchtiefe.

Wichtige Methoden:

findeBestenZug(Spielfeld): Findet den besten möglichen Zug für die KI.

minimax(Spielfeld spielfeld, int tiefe, boolean istMaximierend, int alpha, int beta): Bewertet Spielzustände rekursiv.

bewerteStellung(Spielfeld): Bewertet eine gegebene Spielstellung.

```
    private static int minimax(Spielfeld spielfeld, int tiefe, boo
        if (tiefe == 0 || spielfeld.istSpielAmEnde()) return bewer

        int bestWert = isMax ? Integer.MIN_VALUE : Integer.MAX_VAL
        for (int i = 0; i < 64; i++) {
            if (!istSpielerStein(spielfeld, i)) continue;
            for (int ziel : getGueltigeZuege(spielfeld, i)) {
                int wert = berechneMinimax(spielfeld, i, ziel, tie
                bestWert = isMax ? Math.max(bestWert, wert) : Matl
                if (isMax) alpha = Math.max(alpha, wert);
                else beta = Math.min(beta, wert);
                if (beta <= alpha) return bestWert;
            }
        }
        return bestWert;
    }
```

5. Ausführung und Benutzerinteraktion

Initialisierung: Die Klasse DameSpiel erstellt ein Spielfeld-Objekt und ein DameView-Objekt. Das Spielfeld wird mit DameView gerendert.

Spielerbewegungen: Bewegungen werden in DameSpiel ausgelöst. Die Spiellogik in Spielfeld überprüft die Gültigkeit und führt Bewegungen aus. Nach jeder Bewegung aktualisiert DameView das Spielfeld grafisch.

6. Datenstruktur und Spielmechanik

Spielfeld: Ein 1D-Array (feld) mit 64 Feldern. Indizes von 0 bis 63 repräsentieren die Felder linear.

Figuren:Positive Werte für weiße Figuren, negative Werte für schwarze Figuren.

Normale Steine haben absolute Werte von 1 und **Damen** haben absolute Werte von 2.

7. Features

Die Klasse BestenPunktestandVerwalten

verwaltet das Lesen, Schreiben und Aktualisieren des höchsten Punktestands eines Spiels durch Interaktion mit einer Textdatei, die in der Regel als “BestScore.txt” benannt ist. Bei ihrer Erstellung initialisiert sie den höchsten Punktestand (besterPunktestand) auf 0 und versucht, einen vorhandenen Punktestand über die Methode punktestandLesen() aus der Datei zu lesen. Wenn ein neuer Punktestand den aktuellen höchsten Punktestand übertrifft, aktualisiert die Methode punktestandSpeichern(int) die Datei und den Punktestand im Speicher. Die robuste Fehlerbehandlung stellt sicher, dass die Anwendung auch dann funktioniert, wenn die Datei fehlt oder beschädigt ist. Diese Klasse ist essenziell, um die höchsten Punktestände zwischen Spielsessions zu speichern und zu vergleichen.

```
class BestenPunktestandVerwalten {
    int besterPunktestand;
    String datei = "Bestscore.txt";

    BestenPunktestandVerwalten(String datei) {
        this.datei = datei;
        besterPunktestand = 0;
        punktestandLesen();
    }

    // Methode zum Lesen des besten Scores aus der Datei
    int punktestandLesen() {
        try {
            BufferedReader br = new BufferedReader(new FileReader
            String zeile = br.readLine();
            br.close();
            if (zeile != null && !zeile.isEmpty()) {
                besterPunktestand = Integer.parseInt(zeile);
            } else {
                besterPunktestand = 0;
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        return besterPunktestand;
    }

    // Methode zum Speichern eines neuen Scores, wenn es ein besse
```

```
    void punktestandSpeichern(int neuerPunktestand) {
        if (neuerPunktestand > besterPunktestand) {
            besterPunktestand = neuerPunktestand;
            try {
                BufferedWriter wr = new BufferedWriter(new FileWr
                wr.write(Integer.toString(besterPunktestand));
                wr.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    int getBesterPunktestand() {
        return this.besterPunktestand;
    }
}
```

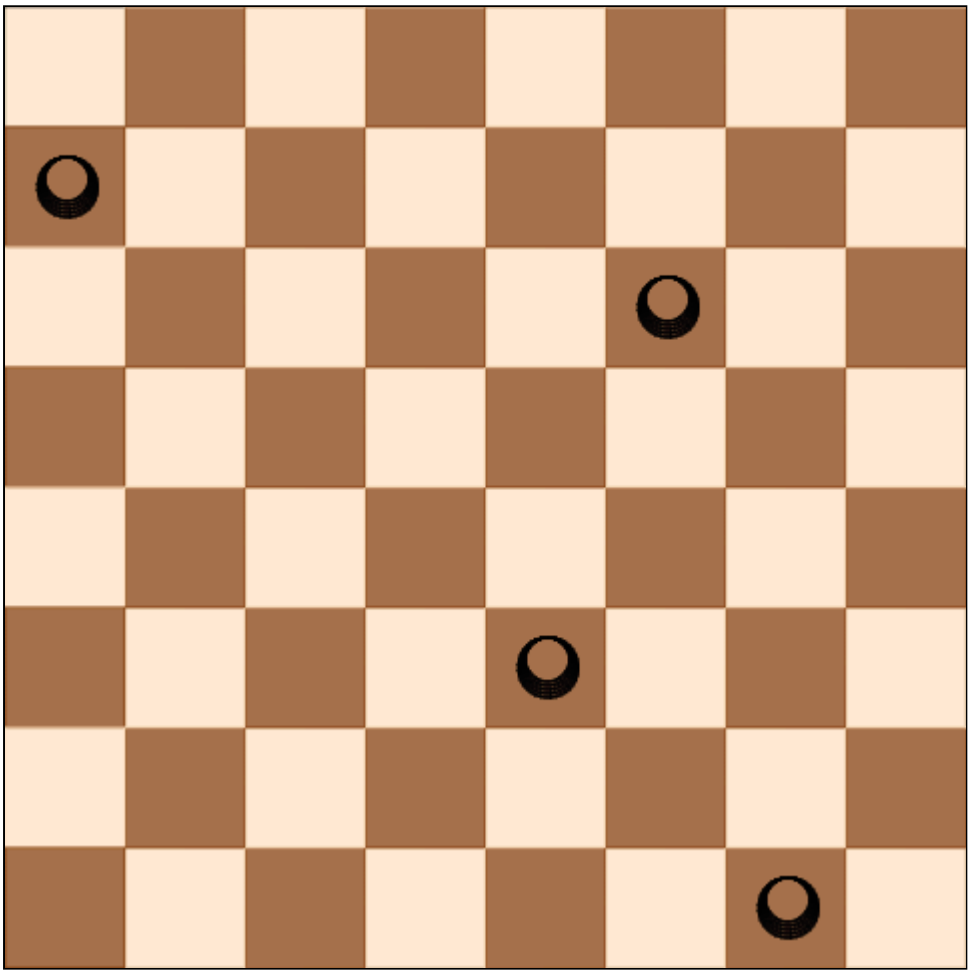
Die Methode zeigeZugMoeglichkeit

berechnet alle möglichen Züge für eine Spielfigur an einem bestimmten Index auf dem Spielfeld. Sie überprüft zunächst, ob der Index gültig ist und eine Figur auf dem angegebenen Feld steht. Danach unterscheidet sie zwischen normalen Steinen und Damen: Damen können sich diagonal in alle Richtungen bewegen, während normale Steine sich je nach Farbe nur vorwärts oder rückwärts diagonal bewegen können. Zusätzlich prüft die Methode auf Schlagzüge, bei denen gegnerische Figuren übersprungen werden können, falls das nächste Feld hinter dem Gegner frei ist. Die möglichen Zielpositionen werden gesammelt und als Array zurückgegeben. Bei ungültigen Eingaben wird eine IllegalArgumentException ausgelöst. Diese Methode dient zur Anzeige und Validierung von Zügen im Spiel.

```
    public int[] zeigeZugMoeglichkeit(int index) {
        ArrayList<Integer> moeglicheBewegung = new ArrayList<>();
        if (!istImSpielfeld(index)) {
            throw new IllegalArgumentException("Der ausgewaehlte St
        }
        int stein = feld[index];
        if (stein == 0) {
            throw new IllegalArgumentException("Das ausgewaelte Fel
        }
        boolean istKoenigen = Math.abs(stein) == 2;
        if (istKoenigen) {
            int[] dx = { -1, 1 };
```

```
        int[] dy = { -1, 1 };

        for (int dirX : dx) {
            for (int dirY : dy) {
                int currentX = index / 8;
                int currentY = index % 8;

                while (true) {
                    int nextX = currentX + dirX;
                    int nextY = currentY + dirY;

                    if (nextX < 0 || nextX >= 8 || nextY < 0 |
                        break;
                    int nextIndex = nextX * 8 + nextY;
                    if (feld[nextIndex] == 0) {
                        moeglicheBewegung.add(nextIndex);
                        currentX = nextX;
                        currentY = nextY;
                    } else if (feld[nextIndex] * stein < 0) {
                        int sautX = nextX + dirX;
                        int sautY = nextY + dirY;

                        if (sautX >= 0 && sautX < 8 && sautY >=
                            int sautIndex = sautX * 8 + sautY;
                            if (feld[sautIndex] == 0) {
                                moeglicheBewegung.add(sautIndex
                                currentX = sautX;
                                currentY = sautY;
                                continue;
                            }
                        }
                        break;
                    } else {
                        break;
                    }
                }
            }
        }
    } else {
        int richtung = (stein > 0) ? 1 : -1;
        int[] diagonalen = { 7 * richtung, 9 * richtung };

        for (int delta : diagonalen) {
            int zielIndex = index + delta;
            if (istImSpielfeld(zielIndex) && feld[zielIndex] ==
```

```
                    moeglicheBewegung.add(zielIndex);
                } else if (istImSpielfeld(zielIndex) && feld[zielI
                    // Moegliche Schlag
                    int ueberspringIndex = zielIndex + delta;
                    if (istImSpielfeld(ueberspringIndex) && feld[ue
                        moeglicheBewegung.add(ueberspringIndex);
                    }
                }
            }
        }
        return moeglicheBewegung.stream().mapToInt(i -> i).toArray
    }
```



```
Spieler 1: Mensch     Punkte: 32    Steine: 1
Spieler 2: KI         Punkte: 44    Steine: 4
```

```
Das Spiel ist bereits beendet.
Der Gewinner ist spieler_2   Punkte : 48   Steine: 4
Der Besterpunktestand: 48
```