

Aufgabenblatt 4

Alpha-Beta Suche

Abgabe (bis 28.05.2019 19:59 Uhr)

Die folgenden Dateien müssen für eine erfolgreiche Abgabe im git Ordner eingereicht sein:

Blatt04/src/Board.java Aufgabe 1.1

Blatt04/src/TicTacToe.java Aufgabe 1.2 und 1.3

Als Abgabe wird jeweils nur die letzte Version im git gewertet.

Aufgabe 1: Tic-Tac-Toe (Hausaufgabe)

Es soll ein Programm geschrieben werden, das beliebige Stellung im Tic-Tac-Toe Spiel bewertet. Bei Gewinnpositionen ist zu berücksichtigen, nach wieviel Zügen der Gewinn (bei optimal spielender Gegnerin) erreicht werden kann. Analoges gilt für Verlustpositionen. Eine genauere Erklärung wird unten gegeben.

Tic-Tac-Toe wird üblicherweise auf einem 3×3 Brett gespielt. Startend von einem leeren Brett markieren die Spielenden abwechselnd jeweils ein freies Feld mit 'x' bzw. 'o'. Es gewinnt, wer zuerst eine Dreierreihe (horizontal, vertikal oder diagonal) vervollständigt, siehe Abb. 1.

Die Implementation soll beliebige $n \times n$ Bretter zulassen, wobei $n \leq 10$ vorausgesetzt werden kann und das Spiel gewonnen ist, wenn n gleiche Steine eine Reihe bilden.

1.1 Implementation der Klasse Board (40 Punkte)

Betrachten Sie die gegebene Klasse `Position` und implementieren Sie die Klasse `Board` gemäß der folgenden API:

public class Board		
	<code>Board(int n)</code>	Erzeugt Board Objekt, ein Spielbrett der Größe $n \times n$ für $1 \leq n \leq 10$.
int	<code>nFreeFields()</code>	Gibt die Anzahl freier Felder zurück.
void	<code>setField(Position pos, int token)</code>	Setzt token auf Spielfeld pos, wobei token die Werte 0 (freies Feld), -1 (Spielstein 'o') oder 1 (Spielstein 'x') sein kann.
int	<code>getField(Position pos)</code>	Gibt Inhalt des Feldes pos zurück.
void	<code>doMove(Position pos, int player)</code>	Führt einen Spielzug von Spielerin player (1 oder -1) auf Feld pos aus.
void	<code>undoMove(Position pos)</code>	Macht den Zug auf Feld pos rückgängig, indem sie das Feld pos leert.
<code>Iterable<Position></code>	<code>validMoves()</code>	Gibt alle Felder, auf die gezogen werden kann, zurück.
boolean	<code>isGameWon()</code>	Wurde das Spiel durch den letzten Zug per <code>doMove()</code> gewonnen?

Bemerkungen:

- Der Konstruktor `board(int n)` sollte eine `InputMismatchException` werfen, wenn die Dimensionen des Spielbretts keinen Sinn ergeben.

- Die Methoden `setField()` und `getField()` müssen eine `InputMismatchException` werfen, wenn die Eingabeargumente keine validen Werte haben.
- Die Methode `undoMove()` braucht kein Gedächtnis zu haben.
- Die Methode `validMoves()` braucht nicht zu prüfen, ob das Spiel bereits beendet ist. Sie kann einfach alle Positionen freier Felder zurückgeben.
- Aus Effizienzgründen sollte die Anzahl der freien Felder oder die Anzahl der gesetzten Steine in einer Variable gespeichert werden, die in der Methode `setField()` aktualisiert wird. Dann müssen nicht bei jedem Aufruf von `nFreeFields()` alle Felder durchsucht werden.
- Sie sollten auch überlegen, wie Anfragen von `isGameWon()` effizient beantwortet werden können.
- Zum Debuggen kann es hilfreich sein, zusätzlich eine Methode `print()` zu implementieren, die das aktuelle Brett ausgibt.

1.2 Implementation der Klasse `TicTacToe` (50 Punkte)

Implementieren Sie die Alpha-Beta-Suche zur Bewertung einer Spielsituation im Tic-Tac-Toe Spiel als statische Methode

```
public static int alphaBeta(Board board, int player)
```

der Klasse `TicTacToe`. Dabei wird die Spielsituation als `board` übergeben, und die Position soll aus Sicht der Spielerin `player` (1 oder -1) bewertet werden. Gemäß der Motivation in der Einleitung benutzen wir die folgende Bewertung einer Spielstellung, siehe auch Abb. 1:

0	offene Position: keine der Spielerinnen kann einen Sieg erzwingen
$p + 1$	mit $p \geq 0$: Gewinnposition für <code>player</code> , wobei das Spiel mit p freien Feldern gewonnen werden kann
$-p - 1$	mit $p \geq 0$: Gewinnposition für <code>-player</code> (Gegenspielerin von <code>player</code>), wobei das Spiel mit p freien Feldern gewonnen werden kann

Es ist sinnvoll, die eigentliche Methode mit der Signatur

```
public static int alphaBeta(Board board, int player,  
                           int alpha, int beta, int depth)
```

zu implementieren, und diese aus der Methode mit der zuerst genannten Signatur mit geeigneten Werten für `alpha`, `beta` und `depth` aufzurufen.

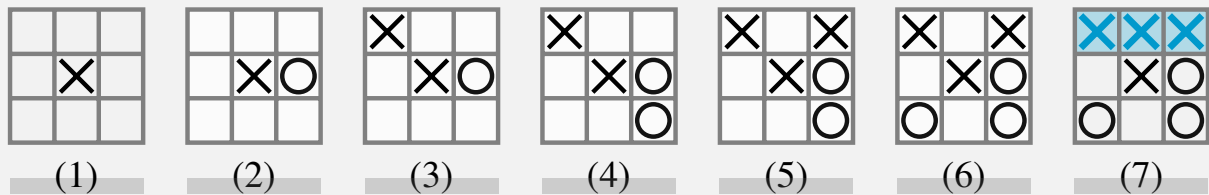
1.3 Bewertung aller möglichen Züge (10 Punkte)

Implementieren Sie in der Klasse `TicTacToe` eine statische Methode

```
public static void evaluatePossibleMoves(Board board, int player)
```

die zu einer gegebenen Spielposition `board` und Spielerin `player`, alle Zugmöglichkeiten mit der `alphaBeta()` Methode bewertet und wie in dem folgenden Beispiel auf dem Bildschirm ausgibt:

Abbildung 1: Tic-Tac-Toe Spiel



Es wird abwechseln gezogen, mit dem Ziel, eine Dreierreihe horizontal, vertikal oder diagonal zu besetzen. Hier endet das Spiel nach sieben Zügen. Da 2 Felder freibleiben, bekommt die Endstellung (7) in unserer Bewertung 3 Punkte und somit gilt dies aus Sicht von 'x' auch für Stellung (6). Ebenso bekommt Stellung (5) aus Sicht von 'o' -3 Punkte, da ein Sieg von 'x' mit 2 freien Feldern nicht vermieden werden kann. Stellung (1) hat hingegen aus Sicht von 'o' 0 Punkte und nicht -3, da es einen besseren Zug für 'o' gibt, mit dem ein Unentschieden erreicht werden kann.

Beispiel 1: Bewertung aller Zugmöglichkeiten

Evaluation for player 'x':

```
0  o  3
0  3 -2
3  0  x
```

Die Spielsituation mit jeweils einem Zug von 'x' und 'o' ist vorgegeben, 'x' ist am Zug. Die besetzten Felder werden entsprechend ausgegeben. Für die freien Felder wird die Zugbewertung angezeigt.

Die Ausgabe der Bewertung muss über System.out erfolgen und genau wie in dem Beispiel formatiert sein (automatisierte Überprüfung). Bei dem Spielbrett gilt der x Wert der Position in waagerechter Richtung. Das 'o' in Beispiel 1 ist also auf Position(1, 0).

Nutzen Sie die Methode, um die folgenden Fragen zum 3 × 3 Tic-Tac-Toe zu beantworten (wird nicht bewertet):

- Gibt es einen ungünstigen ersten Zug, der die zweite Spielerin in eine Gewinnposition versetzt?
- Welcher erste Zug ist am günstigsten insofern, als dass er der zweiten Spielerin möglichst viele Verlustpositionen bereitet.

Hinweise:

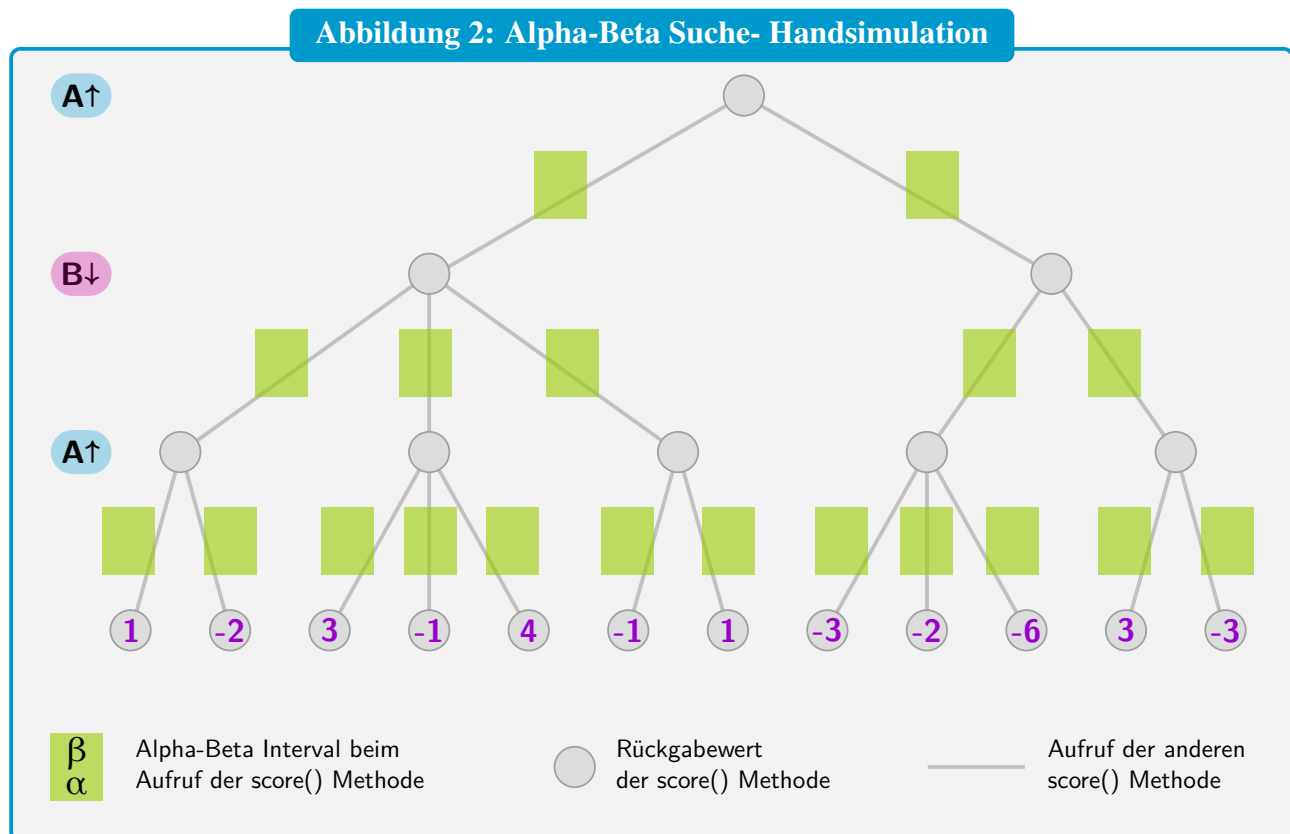
Zum Testen und Debuggen Ihrer Implementation sollten Sie für jede Klasse eine main() Methode schreiben, die ein Objekt der Klasse erzeugt und die unterschiedlichen Methoden aufruft. Alternativ können Sie auch die mitgegebenen JUnit-Tests schreiben, um die einzelnen Methoden zu testen. Für die Board Klasse können Sie z.B. ein leere Brett erzeugen, Züge ausführen, rückgängig machen, und anhand der Ausgabe der Zwischenergebnisse mit Board.print() die korrekte Funktionsweise überprüfen.

Für die TicTacToe Klasse können Sie ein Brett erzeugen, mehrere Züge ausführen und dann die alphaBeta() Methode aufrufen. Erzeugen Sie die Spielpositionen so, dass Sie den korrekten Rückgabewert kennen. Fangen Sie mit einer Position an, in der die ziehende Spielerin direkt mit dem

nächsten Zug gewinnt. Funktioniert Ihr Programm in diesem Fall, testen Sie noch eine andere direkte Gewinnposition und fahren Sie dann mit schrittweise schwierigeren Spielpositionen fort.

Sie können alphaBeta auch für ein leeres 2×2 Brett aufrufen. Welche Ausgabe erwarten Sie?

Aufgabe 2: Alpha-Beta Suche (Tut)



2.1 Wie funktioniert die Alpha-Beta Suche? Was beschreiben α und β ? Was beschreibt jeweils der α - oder β -Cutoff?

2.2 Vervollständigen Sie Abbildung 2.

Aufgabe 3: Branch and Bound (Tut)

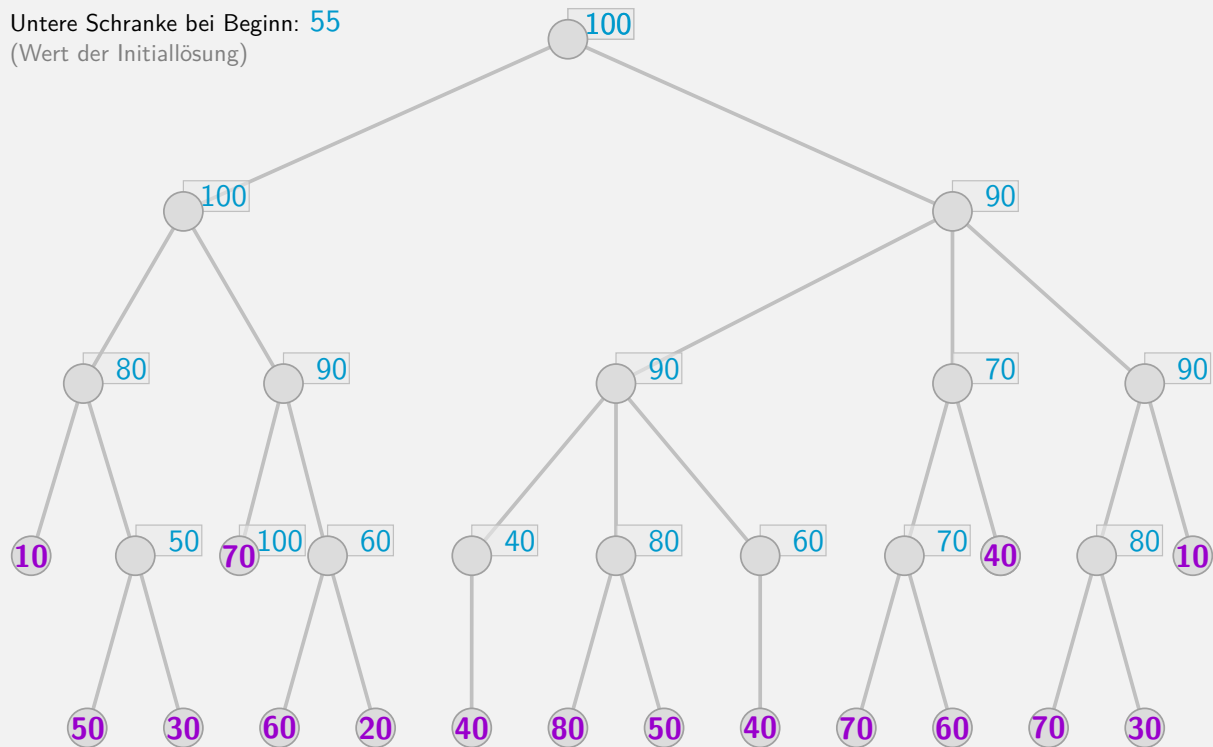
3.1 Welche Konzepte von Algorithmen haben Sie bereits kennengelernt für Bäume der Teillösungen?

3.2 Wie funktioniert Branch and Bound? Erklären Sie die grundlegenden Ideen.

3.3 Vervollständigen Sie Abbildung 3.

Abbildung 3: Branch and Bound- Handsimulation

Untere Schranke bei Beginn: 55
(Wert der Initiallösung)



100 Obere Schranke (*bound*)
der Teillösung

70 Wert der
Lösung

● Rückgabewert der rekursiven
Branch-and-Bound Methode