Programmierung II

Objektorientiertes Programmieren in Java

Universität Passau
Fakultät für Informatik und Mathematik
Innstraße 33
94032 Passau

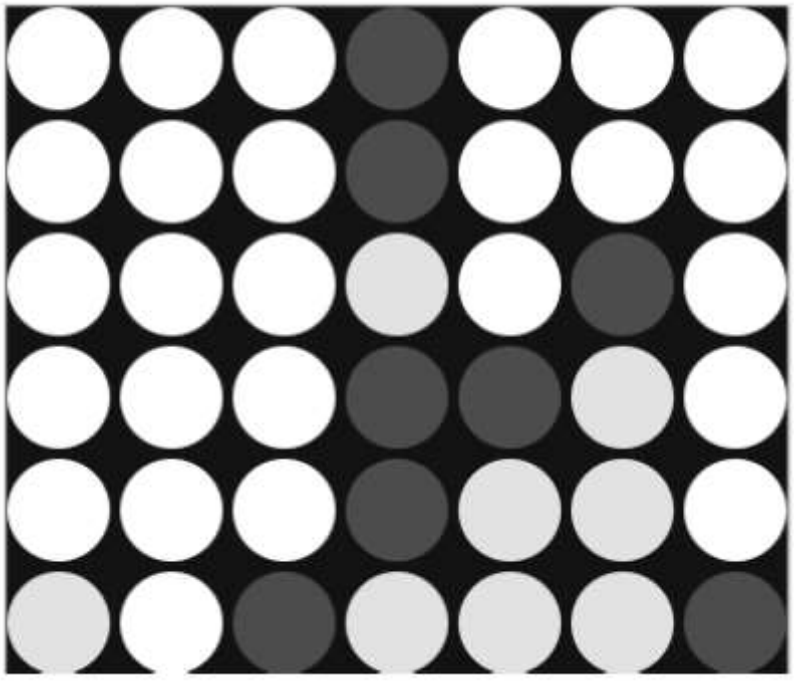
Telefon: +49 851 509-30 35
Telefax: +49 851 509-37 30 35
E-Mail: chris@infosun.fim.uni-passau.de
WWW: http://www.infosun.fim.uni-passau.de/~chris/

Aufgabe 2: Vier gewinnt

"Vier Gewinnt", auch als "Connect Four" oder "Captain's mistress" bekannt, ist ein Brettspiel für zwei Spieler. Das spielt wurde im Jahr 1974 von der Firma Milton Bradley veröffentlicht. Neben dem klassischen Vier Gewinnt gibt es noch Erweiterungen, wie z.B. ein dreidimensionale Varianten, oder Varianten in denen sich die "Richtung" der Gravitation ändern kann, was z.B. durch Drehen des Spielbretts erreicht wird.

In dieser Aufgabe soll die klassische Variante programmiert werden, in der ein menschlicher Spieler interaktiv gegen einen Computergegner antreten kann.

Das Spielprinzip



Das Spielbrett besteht aus 6 × 7 Feldern (**wobei diese Einschränkung der Spielfeldgröße nicht für die zu implementierende Aufgabenstellung gilt!**). In jeder der sieben Spalten des Spielfeldes können die Spieler abwechselnd ihre Spielsteine einwerfen. Der aktuell platzierte Spielstein kommt dabei in der tiefsten noch freien Zelle der jeweiligen Spalte zum liegen - ist eine Spalte bereits vollständig gefüllt, so dürfen dort keine Spielsteine mehr platziert werden.

Gewinner ist derjenige, der zuerst eine durchgängige "Vierer-Gruppe" aus seinen Spielsteinen gesetzt hat. Eine Vierer-Gruppe kann dabei waagrecht, senkrecht oder diagonal verlaufen. Ist das Spielfeld komplett gefüllt, ohne dass einer der Spieler ein Vierer-Gruppe erstellt hat, so endet das Spiel unentschieden.

Aufgabenstellung

Lösungsstrategien

Das Spiel soll interaktiv gegen einen Computer-Gegner gespielt werden können. Daher müssen geeignete Spielzüge für den Computergegner ermittelt werden.

Da Vier Gewinnt ein Spiel mit vollständiger Information ist, könnte theoretisch der gesamte Suchraum für den besten Zug berechnet werden - dies erweist sich allerdings als nicht praktikabel (naiv, ca. 10^{20} Zustände, genauer bis zu $7{,}1 \cdot 10^{13}$ Zustände).

Stattdessen wird für diese Aufgabe eine einfache aber gute Heuristik angewandt. Ausgehend vom aktuellen Spielstand bzw. Spielbrett werden mehrere Züge des Computergegners und des menschlichen Spielers vorausberechnet und in einem Spielbaum verwaltet. Jeder dieser Spielstände wird anhand einer gegebenen Formel bewertet, und der beste Zug wird daraus ermittelt.

Grundvoraussetzung für die Bewertung des Spielstands ist die Identifikation von vorhandenen Gruppen der jeweiligen Spieler. Für die Identifikation von Gruppen sind folgende einfache Regeln zu befolgen.

Gruppenidentifikation

- eine Gruppe besteht aus mindestens zwei und maximal vier Steinen eines Spielers, die lückenlos angeordnet sind
- die Gruppen können horizontal, vertikal oder diagonal auf/absteigend angeordnet sein
- ein Spielstein kann Bestandteil von mehreren Gruppen sein, jedoch nur wenn diese in verschiedenen Richtungen angeordnet sind
- es zählen immer nur maximal-große Gruppen, d.h. die Folge "X X X" entspricht einer (horizontalen) Dreier Gruppe, und nicht etwa zwei zweier Gruppe und einer Dreier Gruppe

Das folgende Beispiel sollte den Sachverhalt verdeutlichen:

```
. . . . . . .
. . . . . . .
. . . X . . .
. . . O . . .
. . . O O X .
. . . O O X X
```

- X hat drei Zweiergruppen - eine horizontale, eine vertikale, und eine diagonal-absteigende
- O hat zwei horizontale, eine vertikale, eine diagonal-aufsteigende, und zwei diagonal-absteigende Zweiergruppen, sowie eine vertikale Dreiergruppe.

Basierend auf diesen Informationen kann nun die Bewertungsformel eingeführt werden.

Bewertungsformel

Die Bewertung des Spielstands bzw. Spielbretts erfolgt anhand der folgenden drei Größen:

1. Gruppengrößen
   - Anzahl der Gruppen einer bestimmten Größe
   - $P = 50 + m_2 + 4 \cdot m_3 + 5000 \cdot m_4 - h_2 - 4 \cdot h_3 - 500000 \cdot h_4$
   - wobei m_x angibt, wie viele Gruppen der Größe x die Maschine besitzt, analog dazu h_x für den menschlichen Spieler
   - Interpretation: der Sieg des Gegner wiegt schwerer, d.h. der Computergegner versucht eine Niederlage zu verhindern

2. Platzierung in inneren Spalten
   - Treppenfunktion über Anzahl Spielsteine in Spalte
   - $Q = m_2 + 2 \cdot m_3 + 3 \cdot m_4 + 2 \cdot m_5 + m_6 - h_2 - 2 \cdot h_3 - 3 \cdot h_4 - 2 \cdot h_5 - h_6$
   - wobei m_x angibt, wie viele Spielsteine die Maschine in Spalte x platziert hat, analog dazu h_x für den menschlichen Spieler
   - Interpretation: Spalten in der Mitte werden bevorzugt, da diese eher an mehreren Gruppen beteiligt sein können als solche am Rand

3. Sofortiger Sieg
   - $R = 5000000$, wenn der Computergegner mit seinem aktuellem Zug den Sieg erringen kann, 0 sonst
   - Interpretation: jegliche taktischen Überlegungen hinfällig wenn Sieg sofort möglich, daher übersteigt dann dieser Wert alle negativen Bewertungsgrößen

Insgesamt errechnet sich nun der "Wert" eines Spielfelds folgendermaßen:

$\text{Bewertung}_{\text{Spielbrett}} = P + Q + R$

Natürlich ist es nicht ausreichend jeweils nur einen einzigen Spielstand zu bewerten. Stattdessen muss man eine Folge von aufeinander aufbauenden Spielzügen betrachten und bewerten, um einen möglichst günstigen Zug auswählen zu können. Es bietet sich daher an, diese Information in einem "Spielbaum" abzulegen.

Spielbaum

Der Fortschritt des Spiels Vier Gewinnt kann als Spielbaum repräsentiert werden. In der Wurzel wird dabei der aktuelle Spielstand gespeichert - das entspricht dem Spielstand nachdem der Mensch seinen Spielstein gesetzt hat, bzw. bevor die Maschine am Zug ist.

Jeder der Knoten im Vier Gewinnt Spielbaum hat bis zu sieben Kindknoten, jeweils einen für eine noch nicht voll besetzte Spalte des aktuellen Spielbretts. Der Index des Kindknoten gibt dabei an, in welche Spalte der nächste Spielstein eingefügt wurde. Hat z.B. ein Kindknoten den Index 5, so bedeutet das, dass ausgehend vom Spielstand im Vaterknoten nun ein weiterer Spielstein in Spalte fünf liegt.

Wie bereits erwähnt, repräsentiert die Wurzel den Spielstand nach dem Zug des menschlichen Spielers. Alle Knoten der Tiefe "1" spiegeln die möglichen Spielstände nach dem Zug des Computergegners wieder. Auf Tiefe "2" folgt darauf aufbauend dann wieder all diejenigen Spielstände, in denen der menschliche Spieler einen weiteren Spielstein gesetzt hat.

Daraus ergibt sich offensichtlich, dass, je tiefer der Baum ist, desto mehr Spielinformation steht dem Computergegner zur Verfügung, und umso besser wird dieser spielen - da aber für jeden Knoten im Baum das Spielbrett geklont werden muss, ergibt sich hier ab einer Tiefe von sechs bereits ein Speicherengpass und eine erhebliche Laufzeitverlängerung. Ein relatives Verfahren von Spielzug zu Spielzug würde hier eine effizientere Implementierung erlauben, was allerdings nicht nötig ist.

Der Spielbaum soll stets vollständig bis zur vorgegebenen Tiefe aufgebaut werden (soweit noch freie Felder vorhanden sind). Das Erstellen von Kindknoten eines Knotens *n* erfolgt also unabhängig davon, ob im durch *n* repräsentierten Spielzustand einer der beiden Spieler den Sieg erreicht hat. Dies hat Auswirkungen auf die Bewertung von Zügen.

Die folgenden interaktiven Bäume geben die Spielbäume aus dem ersten öffentlichen Test wieder:

- ein neues Spiel wird gestartet

```
java Shell
```

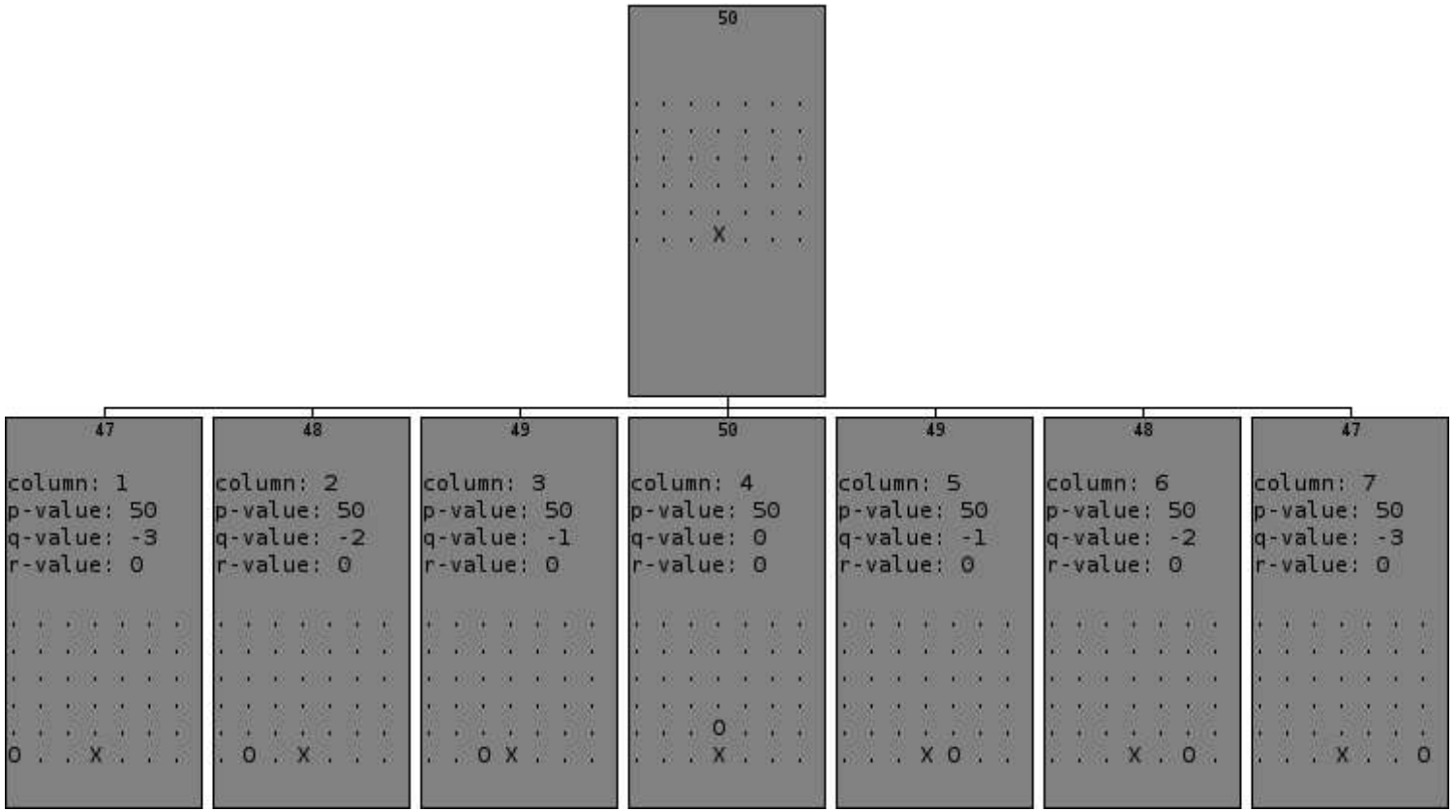
- Level wird auf "1" gesetzt, d.h. der Spielbaum wird die Tiefe "1" haben

```
connect4> level 1
```

- der menschliche Spieler setzt seinen Spielstein in Spalte "4"

```
connect4> move 4
```

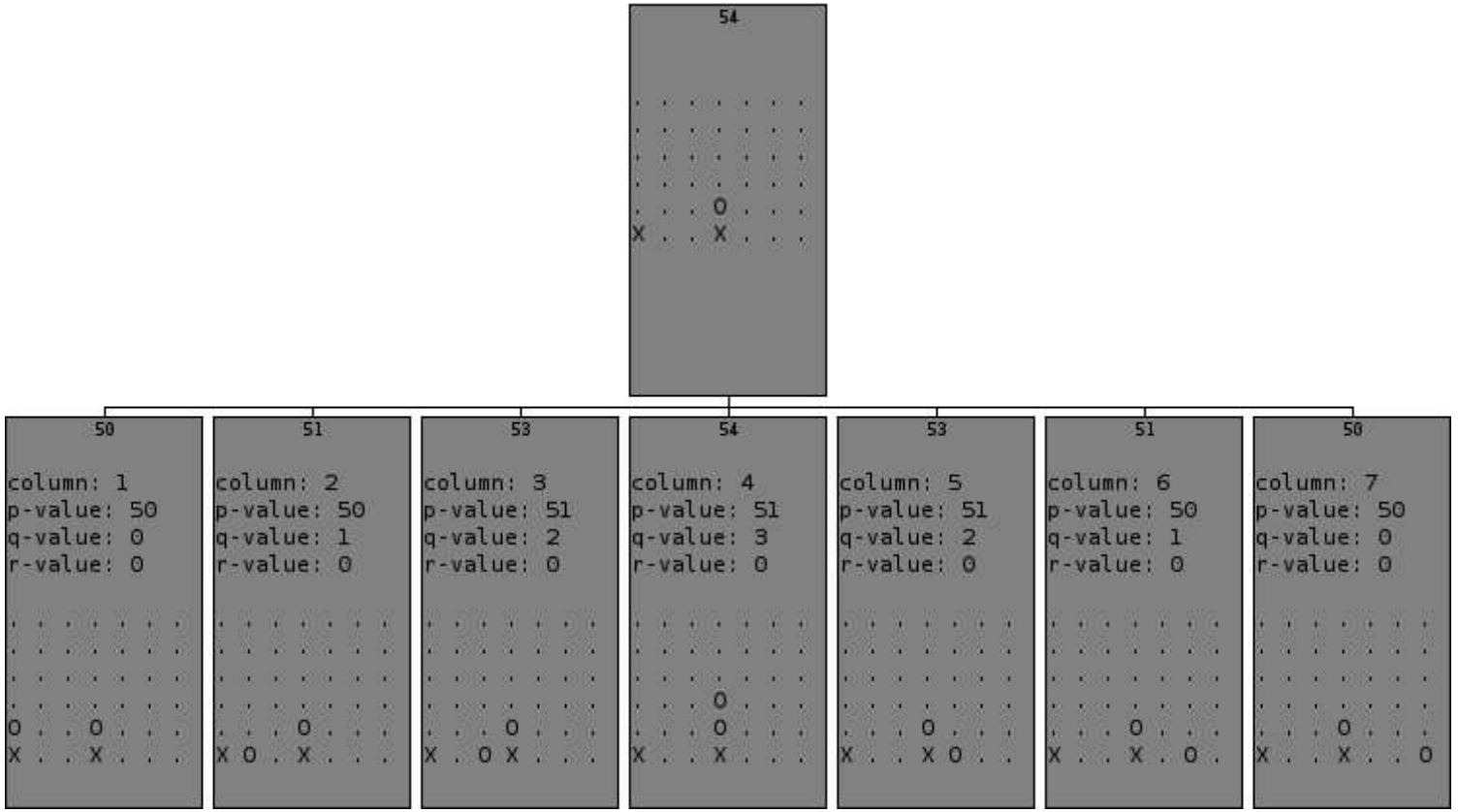
- der Computergegner entscheidet sich anhand des unten stehenden Spielbaums ebenfalls für Spalte "4"



- als nächstes setzt der menschliche Spieler seinen Spielstein in Spalte "1"

```
connect4> move 1
```

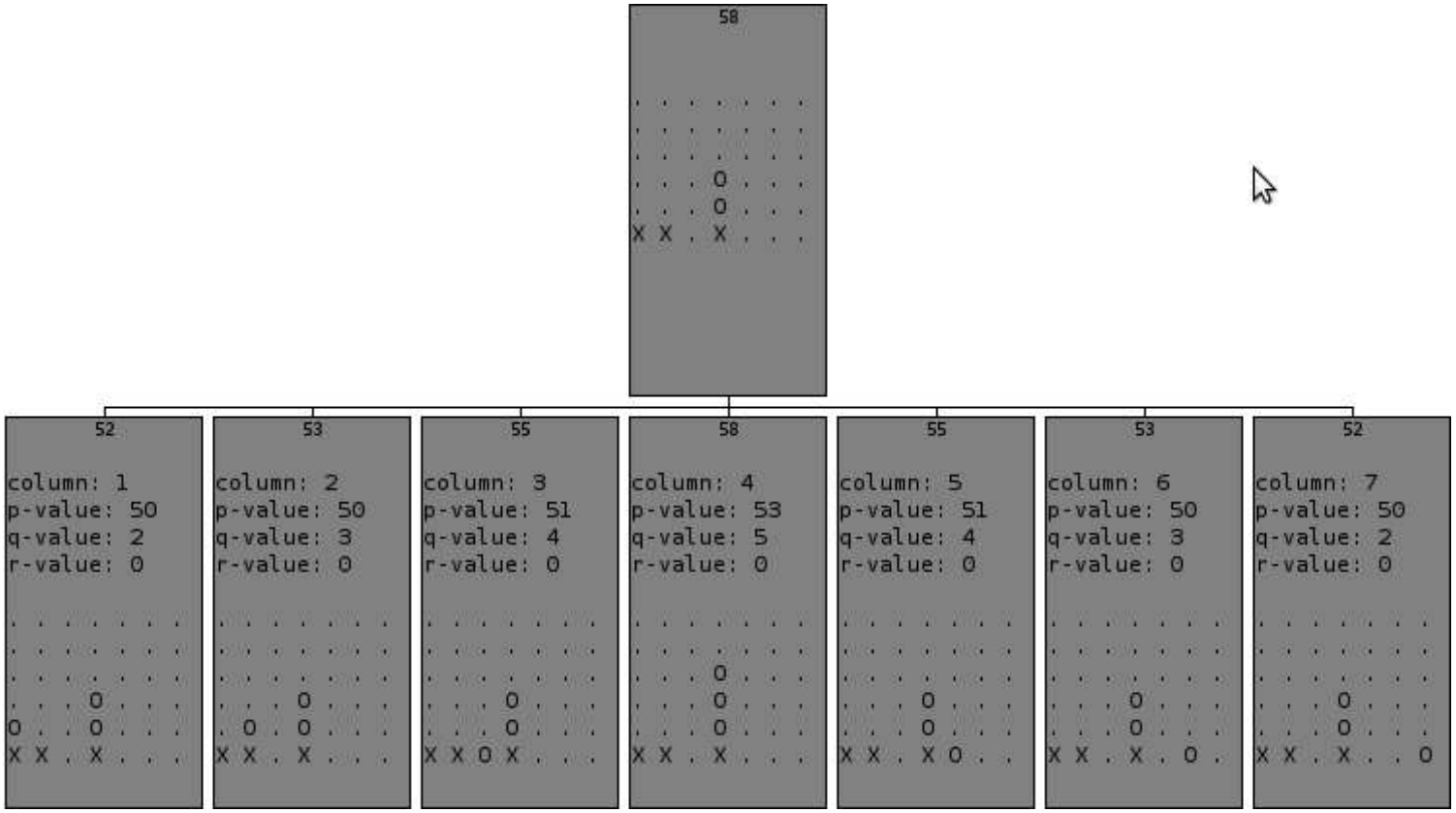
- der Computergegner entscheidet sich anhand des unten stehenden Spielbaums wiederum für Spalte "4"



- als nächstes setzt der menschliche Spieler seinen Spielstein in Spalte "2"

```
connect4> move 2
```

- der Computergegner entscheidet sich anhand des unten stehenden Spielbaums wiederum für Spalte "4"



- durch Platzieren des Spielsteins in Spalte "3" gewinnt der menschliche Spieler das Spiel, und das Spiel ist zu Ende

```
connect4> move 3
```

Die Spielbäume für den zweiten öffentlichen Test finden Sie hier.

Minimax-Algorithmus

Basierend auf dem erzeugten Spielbaum kann mit Hilfe des "Minimax-Algorithmus" der beste Zug in einem bottom-up Verfahren ermittelt werden.

Dabei ist folgendermaßen vorzugehen.

- Bewerte den Spielstand jedes Blattknotens mittels der Bewertungsformel

- Bewerte den Spielstand jedes inneren Knotens mittels der Bewertungsformel und addiere das Extremum seiner direkten Kindkoten
  - Tiefe gerade: Minimalwert der direkten Kindknoten
  - Tiefe ungerade: Maximalwert der direkten Kindknoten

- bevorzugte Spalte des Computergegners ergibt sich aus dem Spaltenindex desjenigen Kindes der Wurzel, dass den maximalen Wert besitzt

Interpretation: Der Computergegner wählt den für sich maximal besten Zug (Maximalwert der direkten Kindknoten) und geht davon aus, dass der menschliche Gegner ebenso den maximal besten Zug für sich wählt - dieser Zug ist aber maximal schlecht für den Computergegner, daher wird hier der Minimalwert der direkten Kindknoten gewählt.

Wichtig: Die Ergebnisse müssen für den Praktomaten überprüfbar sein, daher ergeben sich folgende Einschränkungen

- die Bewertungsformel darf keinesfalls abgeändert werden

- haben zwei Kindknoten laut der Bewertungsformel den selben Wert, so muss immer die linkeste Spalte gewählt werden (Collections.max() garantiert nicht, dass das "linkeste" Maximum gewählt wird)

- Es wird stets der komplette Spielbaum in der Auswertung miteinbezogen.

Hinweise zur Implementierung

Die Kommandoschnittstelle

Prompt: connect4>

Folgende Kommandos sollen unterstützt werden:

- NEW Starte neues Spiel
  - Beim ersten Spiel darf immer Mensch eröffnen
  - Ansonsten bleibt der Starter gleich wie beim vorherigen Spiel
  - Maschine zieht bei Bedarf sofort

- LEVEL Setze das Schwierigkeitslevel
  - Mindestens 1, . . . , 5
  - Default ist 4

    - Ändern während des Spiels mgl.

- SWITCH Wechsle den Eröffner und starte ein neues Spiel
    - Maschine zieht bei Bedarf sofort

- MOVE c Mensch zieht
    - Nur mgl. wenn er auch dran ist
    - Wirft Spielstein in Spalte c (Spalten beginnen bei 1)
    - Meldung bei Sieg
    - Ansonsten zieht jetzt automatisch Maschine
    - Ebenfalls Meldung bei Sieg

- WITNESS Gib Zeugen (= 4er-Gruppe) des Siegers aus
    - Nur erlaubt wenn aktuelles Spiel durch einen Spieler gewonnen wurde
    - Komma-separierte Liste von Koordinaten (r1, c1), (r2, c2), ... der Spielsteine die zum Sieg führten
    - Lexikographisch aufsteigend sortiert
    - Koordinatenursprung (1, 1) links unten

- PRINT Gib Brett als Zeilen × Spalten Matrix aus
    - Mit . für unbesetzte Slots
    - X für Slots die vom Menschen besetzt sind
    - O für Slots die vom Computer besetzt sind
    - Spalten sind per Whitespace ' ' getrennt
    - Zeilen nicht

- Allgemeine Kommandos
    - HELP Gibt einen sinnvollen Hilfetext aus
    - QUIT Beendet das Programm

Es genügt jeweils auf den ersten Buchstaben des jeweiligen Befehls ab zuprüfen, die Groß- und Kleinschreibung kann dabei ebenfalls außer Acht gelassen werden. Es gibt die Meldungen `"Congratulations! You won."`, `"Sorry! Machine wins."` und `"Nobody wins. Tie."` beim jeweiligen regulären Spielende.

Mit Hilfe dieser Befehlen kann das Spiel dann von der Kommandozeile gestartet und betrieben werden.

Testablauf öffenlicher Test #1

```
user@machine:~$ java Shell
connect4> level 1
connect4> move 4
connect4> move 1
connect4> move 2
connect4> move 3
Congratulations! You won.
connect4> witness
(1, 1), (1, 2), (1, 3), (1, 4)
```

Testablauf öffenlicher Test #2

```
connect4> switch
connect4> level 3
connect4> move 7
connect4> move 6
connect4> move 4
connect4> level 2
connect4> move 6
connect4> move 2
connect4> move 7
connect4> move 3
connect4> move 5
connect4> move 6
Sorry! Machine wins
connect4> witness
(1, 3), (2, 4), (3, 5), (4, 6)
```

Implementierung

- Implementierung des Spielbrettes als zweidimensionales Array, bestehend aus n × m Feldern
- klonen des Arrays für jeden Zustand im Spielbaum
- alternativ: relative Methode, die nur Differenz zum vorherigen Zustand abbildet
- Interface für die Implementierung des Spielbretts - Download des Interfaces (die Verwendung ist verpflichtend!):

```
public interface Board extends Cloneable {

  int ROWS = 6; // number of rows
  int COLS = 7; // number of columns
  int CONNECT = 4; // min. group size to win

  Player getFirstPlayer(); // get starter (set via constructor)

  Board move(int col); // make a human move

  Board machineMove(); // make a machine move

  setLevel(int level); // set the machine skill

  boolean isGameOver(); // game over check

  Player getWinner(); // the winner if any

  Collection<Coordinates2D> getWitness(); // a winning group

  Player getSlot(int row, int col); // content of slot

  Board clone(); // make a deep copy

  String toString(); // matrix representation
}
```

- Die Implementierung des Spielbretts hat das vorgegebene **Interface Board** zu erfüllen.
  - die Shell kommuniziert ausschließlich über dieses Interface mit dem Spielbrett
  - Ermöglicht Erweiterbarkeit z.B. für die GUI-Aufgabe benötigt
  - move(int col) <=> null wenn Spalte col voll und Kopie mit ausgeführtem Zug sonst 2.
  - level bestimmt Look-Ahead, d.h. Tiefe des Spielbaums
  - getWitness(): kann mehrdeutig sein - welches "O" wurde zuletzt gesetzt?

```
. . . . . . .
. . . . . . .
. . . . . . .
. . . . . . .
O O O O O X .
X X X O X X .
```

- die Größe des Spielbretts bzw. der maximalen Gruppengrößen werden nur durch die Konstanten ROWS, COLS bzw. CONNECT definiert
  - einzige Ausnahme ist die Bewertungsformel, die mit diesen fixen Größen arbeiten darf

- Selbstdefinierte Exception "IllegalMoveException" wird geworfen, wenn gewählter Zug nicht erlaubt ist
  - der Spieler der gezogen ist, ist gar nicht am Zug
  - das Spiel ist bereits zu Ende

- inkrementelles Vorgehen:
  - erst alle Züge manuell durchführen
  - Gruppenidentifikation zuerst nur orthogonal
  - menschlicher Spieler eröffnet immer das Spiel
  - Tiefe des Spielbaums auf "1" beschränken und mit Stift, Zettel und Debugger abprüfen

Implementierungshinweise

- Die Hauptdatei des Programms muss als **Shell.java** benannt sein.

- Fehlermeldungen müssen von der Form `Error! ....` sein

- Es muss eine Datei **Tests.txt** mit entsprechenden Testfällen des Programms abgegeben werden - eine fehlende oder stark ungenügende Testfallbeschreibung wird mit Funktionalität 'C' bewertet.

- Gründliches Testen inkl. Spezialfälle dringend notwendig

- **Achtung! Der Praktomat erwartet KEIN LEERZEICHEN nach jeder Zeile die ausgegeben wurde**

- **Beachten Sie alle weiteren Hinweise in der Aufgabenstellung im Vorlesungsskript und besuchen Sie vor allem die Vorlesung!**

- Voraussetzung für das Bestehen sind ausschließlich die Testergebnisse des Praktomaten - die hier dargestellten Spielbäume sind nur zur Visualisierung gedacht, ohne Anspruch auf Korrektheit und Vollständigkeit.

Termine

Freitag, 07. Januar 2022, 10:00 Uhr:

Erste Lösung einreichen (empfohlen)
Freitag, 14. Januar 2022, 10:00 Uhr:
Erste Lösung einreichen (fest)

Ihre Lösung liegt noch nicht vor.
Sie *sollten* Ihre Lösung bis **Freitag, 07. Januar 2022, 10:00 Uhr** einreichen.
Sie *müssen* Ihre Lösung bis **Freitag, 14. Januar 2022, 10:00 Uhr** einreichen.

Schließen

Fragen? Kommentare? Wenden Sie sich an Praktomat-Support <chris@infosun.fim.uni-passau.de>.
Bei Problemen wenden Sie sich bitte an die Programmierberatung oder das Forum.
Praktomat 4.4.1 (Python 3.8.10), Praktomat-Zeit: 2021-11-29 09:59