Aufgabenstellung Programmierpraktikum WS 2003/2004

1 Lernziel

Im Rahmen dieses Praktikums sollen Sie eine größere Programmieraufgabe mit einer vorgegebenen Programmiersprache (hier Java) lösen und dabei einen Eindruck davon erhalten, welche Anforderungen bzw. Probleme typischerweise im Umfeld der Softwareerstellung entstehen. Es sollen die folgenden Lernziele erreicht werden:

1. Umsetzung einer nicht-trivialen Aufgabenstellung.
2. Selbständige Planung und Durchführung der Programmieraufgabe über einen längeren Zeitraum.
3. Entwicklung einer Lösung, die einer vorgegebenen Spezifikation genügt.
4. Erzeugung einer Lösung, die ausreichend getestet und dokumentiert ist.

Das Praktikum ist so angelegt, dass solide Kenntnisse in Java bereits vorausgesetzt werden und nicht erst während des Praktikums erworben werden können.

2 Durchführung des Praktikums

Der Bearbeitungszeitraum für die Programmieraufgabe beginnt nach dem Versand der Aufgabenstellung am 06.10.2003; Ihnen steht also ein Zeitraum von mehr als 3 Monaten bis zum Abgabetermin am 16.01.2004 zur Verfügung. An diesem Termin erwarten wir von Ihnen das komplett funktionsfähige, getestete und dokumentierte Programm; diese Programmversion wird auch die Grundlage für die Vorführung und Erweiterung des Programmes in der Präsenzphase am 28.02.2004 oder 29.02.2004 sein. An welchem der beiden Termine Sie an der Präsenzphase teilnehmen können, teilen wir Ihnen per Email nach der Begutachtung Ihrer Einsendungen mit. Sollten alle Teilnehmer die Aufgabe erfolgreich lösen, müssen wir gegebenenfalls noch ein weiteres Wochenende für die Präsenzphase hinzunehmen.

Erfahrungsgemäß wird der Aufwand zum Erstellen und Testen der Programmlösung unterschätzt. Beginnen Sie daher möglichst gleich mit der Lösung der Aufgabe und planen Sie mindestens die gleiche Zeit zum Testen ein, die Sie für die Erstellung des Programms benötigt haben.

Später eintreffende Lösungen bzw. zwischen dem Abgabetermin und der Präsenzphase durchgeführte Programmänderungen werden bei der Bewertung nicht mehr berücksichtigt.

3 Wann erhalten Sie den Leistungsnachweis?

Dazu erwarten wir von Ihnen:

1. Eigenständige Implementierung eines fehlerfreien Programms gemäß der nachfolgenden Spezifikation. **Wichtig: Gruppenlösungen sind nicht zulässig.**

2. Einsendung des Programms per Email bis zum 16.01.2004 unter Berücksichtigung der von uns angegebenen Programmier- und Dokumentationsrichtlinien.
   Email-Adresse: `dominic.heutelbeck@fernuni-hagen.de`
   Zu Ihrer Abgabe müssen die folgenden Komponenten gehören:

   - das vorcompilierte Programm als .jar-Datei, welches sich durch ein einfaches „`java -jar IHR_NAME.jar`“ ausführen und testen lassen muss,
   - der Quelltext mit erstelltem javadoc als .zip-Datei mit dem Namen „IHR_NAME.zip“,
   - die Dokumentation als ASCII-, RTF- oder PDF-Datei. Die javadoc Dateien müssen sich in einem gesonderten Verzeichnis „docs“ befinden.

4 Teilnahme an der Präsenzphase

In der Präsenzphase stellen Sie Ihr Programm vor, präsentieren sein Verhalten anhand ausgesuchter Beispiele und erweitern ggf. Ihre Lösung um eine zusätzliche Funktionalität.

Manipulationsversuche mit den eingesandten Programmen, z.B. durch Aufspielen von Computerviren oder anderen Programmkomponenten, die nicht der Lösung der Programmieraufgabe dienen, führen - unabhängig von Schadensersatzansprüchen seitens der FernUniversität - zu einer Nichterteilung des Leistungsnachweises.

Der Leistungsnachweis für das Programmierpraktikum ist unbenotet.

5 Die Programmieraufgabe

In diesem Praktikum werden Sie sich mit dem Brettspiel Gobang beschäftigen.

5.1 Spielregeln

Gobang ist ein Spiel für zwei Personen (Spieler A und Spieler B). Gobang wird auf einem Spielbrett gespielt, das sich aus n*m einzelnen Feldern zusammensetzt. Es ist zweckmäßig, das Brett wie beim Schach mit Koordinaten zu versehen, um Spielzüge eindeutig angeben zu können.

Als Spielfiguren werden farbige Spielsteine verwendet. Es gibt $\lceil (n*m)/2 \rceil$ weiße und $\lceil (n*m)/2 \rceil$ schwarze Steine, also insgesamt mindestens so viele, wie das Spielbrett Felder hat. Spieler A gehören die weißen Steine, Spieler B die schwarzen.

In jedem Spielzug platziert ein Spieler einen Stein seiner Farbe auf ein Feld des Spielbrettes. Das Spiel verläuft nach folgenden Regeln:

- Das Spiel beginnt mit einem leeren Spielbrett.
- Die Spieler müssen abwechselnd Spielzüge durchführen. Der Spieler mit den weißen Steinen (Spieler A) beginnt.
- Steine dürfen nur auf leere Felder des Spielbrettes gelegt werden.
- Spieler A darf nur weiße Steine, Spieler B nur schwarze Steine auf dem Spielbrett platzieren.
- Beim ersten Spielzug darf ein Stein auf ein beliebiges Feld des Spielbrettes gelegt werden.
- Ab dem zweiten Spielzug dürfen Steine nur auf solche Felder gelegt werden, von denen mindestens eines der acht angrenzenden Nachbarfelder bereits durch einen Stein belegt ist.
- Steine dürfen nicht auf dem Spielbrett verschoben werden.
- Einmal gelegte Steine dürfen nicht wieder vom Spielbrett entfernt werden.

Sobald in einer Vertikalen, Horizontalen oder Diagonalen fünf Steine einer Farbe eine durchgehende Reihe bilden, ist das Spiel beendet, und der Spieler mit der entsprechenden Farbe hat gewonnen. Sind alle Felder des Spielbrettes belegt und existieren keine 5er-Reihen, dann endet das Spiel mit einem Unentschieden.

5.2 Anforderungskatalog

Die Aufgabe ist es, ein interaktives Gobang-Spiel zu implementieren. Die Aufgabe besteht aus zwei Hauptbestandteilen, den Zuggeneratoren für die

Computergegner und einer graphischen Oberfläche. Das Spiel soll mindestens zwei Computergegner mit unterschiedlichen Strategien implementieren. Wie sie später sehen werden, bedeutet dies hauptsächlich die Implementierung von unterschiedlichen Methoden zur Stellungsbewertung.

Um die eigenen Strategien direkt mit denen der anderen Teilnehmer vergleichen zu können, ist zusätzlich ein „Zuggeneratorserver“ zu implementieren.

Beim Starten eines neuen Spiels soll der Anwender die Möglichkeit haben, die Spieler festzulegen. Für Spieler A und B soll es jeweils möglich sein, zwischen einem menschlichen Spieler, den verschiedenen Computergegnern oder Zuggeneratorservern auszuwählen. Folgende Paarungen müssen möglich sein:

- Mensch vs. Computer X,
- Mensch vs. Mensch,
- Computer X vs. Computer Y,
- Mensch vs. Zuggeneratorserver,
- Computer vs. Zuggeneratorserver,
- Zuggeneratorserver vs. Zuggeneratorserver.

Bei einem Spiel ohne menschliche Spieler soll das Spiel graphisch dargestellt werden und in einer angenehm zu verfolgenden Geschwindigkeit ablaufen. Wenn es sich um das Duell zweier Zuggeneratorserver handelt, läuft nur die Steuerung und Darstellung des Spielablaufes auf dem lokalen Rechner. Die Züge werden aber auf den entfernten Rechnern generiert.

Zusätzlich soll die Möglichkeit bestehen die Größe des Spielbretts auszuwählen. Bitte geben Sie dem Benutzer die Wahl zwischen folgenden Spielbrettgrössen: 8x8 10x10 15x15 20x20. Zusätzlich soll eine frei wählbare Spielbrettgrösse möglich sein, bei der die beiden Seitenlängen nicht gleich sein müssen.

Es müssen Eingabefelder zur Verfügung gestellt werden, um die Netzwerkadresse, Portnummer, Rechentiefe und Nummer (0 oder 1, siehe Abschnitt 5.2.3) des jeweiligen Zuggeneratorservers eingeben zu können.

Stellen Sie sicher, dass sowohl Mensch als auch Computergegner nur legale Züge machen.

5.2.1 Die graphische Oberfläche

Die Oberfläche ist mit der Swing-Klassenbibliothek zu implementieren. Die Oberfläche muss folgende Elemente beinhalten:

- Eine graphische Repräsentation des Spielbretts, über welche ein menschlicher Spieler seine Züge durch Klicken durchführen kann.
- Ein Menü mit folgenden Möglichkeiten:
  - **a)** ein neues Spiel zu starten,
  - **b)** das Spiel zu beenden,
  - **c)** einen Zugvorschlag abzurufen,
  - **d)** einen Konfigurationsdialog zu starten für die Rechentiefe der Zuggeneratoren.
- Einen Dialog, der beim Starten eines neuen Spieles geöffnet wird und zur Konfiguration des Spiels dient.
- Eine Anzeige, die beim Spielende das Spielergebnis mitteilt. Dieses kann ein Dialogfenster sein, kann aber auch graphisch auf dem Spielfeld dargestellt werden.

Wichtiger Hinweis: Die Spielfläche sollte optisch ansprechend gestaltet sein. Eine Implementierung der Spielfläche aus einem Raster von Schaltflächen (Buttons) wird nicht akzeptiert. Der unter c) genannte Menüpunkt sollte einen Computergegner einen möglichst guten Zug für den Spieler generieren lassen. Diese Zugmöglichkeit soll dem Spieler graphisch (z.B. durch Blinken des betreffenden Feldes) angezeigt werden.

5.2.2 Die Computergegner

Spiele wie Gobang lassen sich leicht als Spielbaum darstellen. Die Knoten des Baumes entsprechen den möglichen Spielsituationen. Die Wurzel des Baumes ist in unserem Fall das leere Spielbrett. Die Kinder jedes Knotens sind die Spielsituationen, die durch legale Züge aus der gegebenen Situation entstehen können. Die Blätter sind Spielsituationen, in denen kein weiterer Zug mehr möglich ist. Diese Situationen entsprechen einer Gewinnstellung für einen der Spieler oder einem unentschieden. Es ist leicht einzusehen, dass ein komplettes Durchrechnen dieses Baumes keine praktikable Lösung ist. Die exponentielle Laufzeit eines solchen Algorithmus macht ein interaktives Spiel unmöglich. Wir werden den Spielbaum trotzdem als Grundlage für die Strategie unseres Computergegners verwenden. Eine solche Strategie besteht aus drei wesentlichen Elementen:

- Berechnung aller möglichen Züge,
- Traversierung des Spielbaums,
- Bewertung der Stellungen.

In unserer Spielbaumanalyse werden wir den Baum nur bis zu einer bestimmten Tiefe, hier die Denktiefe genannt, betrachten. In den Knoten unseres Spielbaumes werden wir einen numerischen Wert für die Güte der Stellung verwalten. Für Sie wird in der Implementierung das Hauptgewicht auf die Berechnung dieser Werte, die Stellungsbewertung, fallen. Dazu müssen Sie eine Methode implementieren, die für eine bestimmte Stellung bewertet, welcher Spieler in der gegebenen Stellung im Vorteil ist. Der Rückgabewert ist eine Zahl und soll positiv sein, wenn Spieler A besser steht (je höher desto besser ist die Situation für Spieler A), und negativ, wenn Spieler B besser steht (je niedriger desto besser ist die Situation für Spieler B), und 0 für eine ausgeglichene Stellung. Diese Methode darf nur die gegebene Stellung bewerten. Der Methode ist es ausdrücklich verboten, eine Abfolge von Zügen zu analysieren. Bei der Bewertung ist Ihre Kreativität gefordert. Die Qualität dieser Methode wird maßgeblichen Einfluss auf die Spielstärke Ihres Programms haben.

Die meisten Spielprogramme arbeiten in etwa folgendermaßen: Für eine vorgegebene Denktiefe (z.B. 4) werden alle möglichen Zugfolgen berechnet. Die danach entstandenen Stellungen werden bewertet und dieser Wert als Wert der jeweiligen Zugfolge zurückgegeben. Der Zug, der die „beste“ Bewertung der Zugfolge bekommt, wird am Ende tatsächlich ausgeführt. Schreiben Sie eine rekursive Methode, die zu einem Spieler, der am Zug ist, zu einem gegebenen Brett, und einer Denktiefe den „besten“ Zug ermittelt und ihn zusammen mit seiner Bewertung zurückliefert. Dabei geht man wie folgt vor:

- Ermitteln Sie zuerst alle legalen Züge.
- Falls legale Züge existieren, wird für jeden dieser Züge die passende resultierende Stellung generiert. Dann ruft die Methode sich selbst rekursiv auf, aber mit der neuen Stellung und aus der Sicht des anderen Spielers.
- Ist die maximale Denktiefe erreicht, wird die Stellung bewertet.
- Ist die Stellung besser als alle bisherigen, merkt man sich den Zug.
- Wenn alle Züge so durchprobiert wurden, ist der „beste“ Zug gefunden.
- Falls keine legalen Züge existieren, ist das Spiel beendet, und als Bewertung wird Null zurück gegeben.

Die Bewertung, wann ein Zug besser ist als ein anderer, folgt dem so genannten Minimax-Prinzip. Der Name kommt von der Tatsache, dass einer der Spieler versucht den Wert der Stellung zu maximieren, während der andere Spieler versucht ihn zu minimieren. Betrachten wir ein Beispiel: In der Ausgangsstellung (siehe Abb. 1) sei Spieler A am Zug. Es soll gezeigt werden, wie die Stellungswerte von unten nach oben ermittelt werden. Das

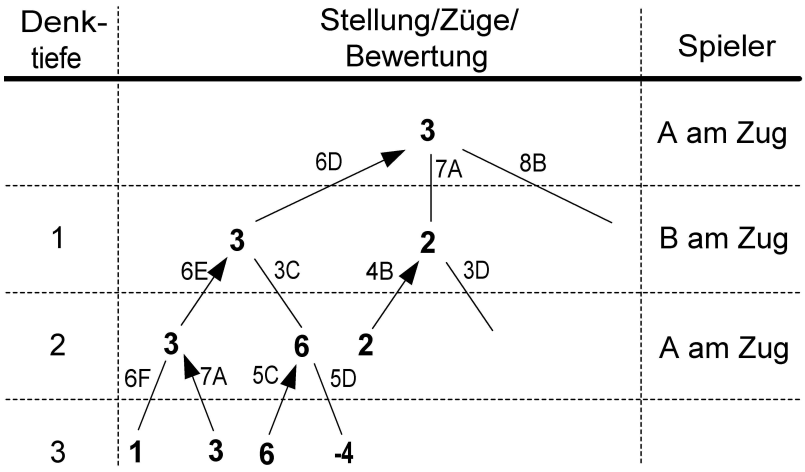

Abbildung 1: Ein Spielbaum

Prinzip ist, dass sich der Wert eines Zuges aus den Werten aller möglichen Antwortzüge ergibt, oder, falls die vorgegebene Denktiefe erreicht wurde, aus dem Wert des Stellungsbewerters.

Spieler A hat in der Ausgangsstellung drei legale Züge: 6D, 7A und 8B. Wenn er 6D ziehen würde, hätte Spieler B darauf zwei legale Antwortzüge: 6E oder 3C. Auf den ersten Zug davon hat Spieler A wieder zwei Zugmöglichkeiten: 6F oder 7A. Diese werden nun mit dem Stellungsbewerter bewertet (die fest vorgegebene Denktiefe ist hier 3).

Der Stellungsbewerter ermittelte für die aus der Zugfolge 6D-6E-6F resultierenden Stellung den Wert 1, und für die aus der Zugfolge 6D-6E-7A resultierende Stellung den Wert 3. Da Spieler A immer den besten Zug machen möchte und eine größere Zahl eine bessere Stellung für Spieler A verspricht, würde Spieler A, falls vorher 6D-6E gezogen würde, Zug 7A wählen, also den Zug, der die meisten Punkte ergibt.

Aus der nach 6D-6E entstandenen Stellung heraus kann Spieler A nun im besten Fall einen Zug machen, der eine Stellung mit Wert 3 erlaubt. Dies ist damit auch der Wert des Zuges 6E für Spieler B in der darüberliegenden Ebene! Dort versucht Spieler B natürlich eine Stellung mit möglichst kleiner Zahl zu erreichen, d.h. er wählt den Zug, gegen den Spieler A den schlechtesten Antwortzug hat! Nun wird also zunächst der Wert für den Zug 3C ermittelt. Der beste Zug, den Spieler A nach der Ausführung von 6D-3C machen kann, ergibt eine Stellung mit dem Wert 6 und ist deshalb schlechter für Spieler B. Also wählt Spieler B den Zug, dessen Antwort nur maximal 3 Punkte zulässt: 6E.

Dies ist dann wiederum auch der Wert des allerersten Zuges (6D) für Spieler A. Spieler A versucht nun wiederum, den Stellungswert zu maximie-

ren. Die Werte für die anderen Züge mögen mit dem gleichen Verfahren 2 und 0 ergeben. Diese sind alle kleiner, also schlechter für Spieler A. Deshalb wird Spieler A letztendlich den ersten Zug, nämlich 6D wählen. Dies ist das Ergebnis der Berechnung und wird zusammen mit dem Wert 3 zurückgeliefert.

Jetzt zeigt sich, wie gut Ihr Stellungsbewerter arbeitet. Ergeben zwei Züge den gleichen Wert, so ist es egal, für welchen man sich entscheidet. Hier kann auch ein Zufallszahlengenerator entscheiden, so dass Ihr Programm nicht immer gleich spielt.

Machen Sie sich klar, dass jeder Teilbaum einen rekursiven Aufruf der Methode zum Finden des besten Spielzuges mit verändertem Brett und veränderter Denktiefe darstellt und nur die Blätter einen Aufruf des Stellungsbewerters darstellen. Lassen Sie sich durch die Rekursion nicht verwirren. Wählen Sie zum Testen kleine Denktiefen (z.B. 4). Falls die Rechenleistung es zulässt, können Sie die Denktiefe erhöhen.

Diese Suchmethode, die ja alle Zugmöglichkeiten beachtet, wird in der Praxis brute-force Methode genannt. Durch die Technik des „α-β Schnittes“ lässt sich die Geschwindigkeit drastisch erhöhen. Der Schnitt reduziert die Anzahl der betrachteten Knoten im Baum. Die Idee ist, dass wir in großen Teilen des Baumes nicht an den exakten Werten der Stellungen interessiert sind, sondern nur daran, ob sie besser oder schlechter sind als die Positionen, die wir bereits gefunden haben. Der Algorithmus verwaltet zwei Werte α und β, die garantierte minimale Bewertung für den maximierenden Spieler, bzw. die garantierte maximale Bewertung für den minimierenden Spieler darstellen. Zu Beginn werden diese Werte auf $alpha = -\infty$ bzw. $\beta = +\infty$ gesetzt (oder einen passenden praktischen Wert). Im Verlauf der Rekursion wird dieses Intervall immer schmaler. Wenn β kleiner als α wird, kann die aktuelle Stellung nicht das Ergebnis des besten Spiels beider Spieler sein. Das wiederum bedeutet, dass dieser Teilbaum nicht weiter untersucht werden muss.

Dieser Algorithmus liefert dieselben Ergebnisse wie der Minimax-Algorithmus. Die Ersparnis durch die Schnitte ist allerdings so groß, dass man im Durchschnitt in derselben Zeit doppelt so tief rechnen kann. Der Algorithmus kann weiter verbessert werden, ohne seine Genauigkeit zu beeinflussen. Durch geschickte Heuristiken für die Reihenfolge der Knotentraversierung kann man frühe Schnitte erzwingen. Im Schach könnten z.B. schlagende Züge vor allen anderen untersucht werden, oder Züge die in vorausgegangenen Berechnungen hohe Bewertungen erhalten haben, könnten früher untersucht werden als andere. Es gibt noch zahlreiche weitere Verbesserungsmöglichkeiten. Es steht Ihnen frei auch hier weitere Optimierungen durchzuführen. Hier ein kurzer Pseudocode zur Illustration:

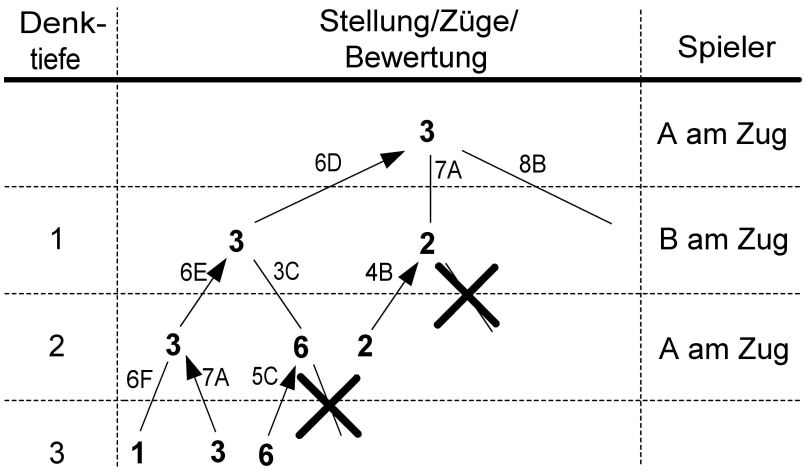

Abbildung 2: Der Spielbaum mit eingezeichneten Schnitten.

```
alphabeta(Stellung, Alpha, Beta, Maximierend) {
  if (Stellung ist Blatt) {
         return (bewertung(Stellung));
  } else if (not Maximierend) {
    for (alle Folgestellungen S') {
      Wert = alphabeta(S', Alpha, Beta, true);
      Beta = Min{Beta, Wert};
      if (Beta <= Alpha)
        break;
     }
     return (Beta);
  } else // aktueller Spieler maximiert
    for (alle Folgestellungen S') {
      Wert  = alphabeta(S', Alpha, Beta, false);
      Alpha = Max{Alpha, Wert};
      if (Beta <= Alpha)
        break;
     }
     return (Alpha);
  }
}
```

Der Algorithmus wird mit `alphabeta(Stellung,` $-\infty$`,` ∞`, true)` aufgerufen. Bitte dokumentieren Sie Ihre Version des Algorithmus im Detail und begründen Sie Ihre Designentscheidungen.

5.2.3 Die Zuggeneratorserver

Beim Start soll das Programm einen Zuggeneratorserver starten, um seine Computergegner über das Netzwerk zur Verfügung zu stellen. Das Prinzip des Zuggeneratorservers ist einfach. Ein Client sendet dem Server die aktuelle Spielsituation und teilt ihm mit, ob er Spieler A oder B ist. Daraufhin antwortet der Server mit einem Zug.

Implementieren Sie den Server über ServerSockets. Beim Aufbau einer Verbindung wird ein neuer Thread gestartet, der die Kommunikation mit dem Client und die Zugberechnung durchführt.

Nehmen Sie als Port für die Verbindungen **7888** an. Bitte sehen Sie aber eine Möglichkeit vor um diesen Port zur Laufzeit ändern zu können. Wenn Sie ihre Strategie mit der eines Kommilitonen über das Netz vergleichen möchten, kann es wegen Firewalls notwendig sein diesen Port zu ändern (z.B. auf Port 80, über den üblicherweise Webserver laufen).

Die Anzahl der gleichzeitig mit dem Server verbundenen Clients ist nicht begrenzt.

Zur Kommunikation nutzen wir SGP, das „Simple Gobang Protocol“. SGP besteht aus zwei Nachrichtentypen, einer **ZugAnfrageNachricht** und einer **ZugNachricht**. Die Nachrichten sind zeilenorientierte ASCII-Texte. Alle Zeilen werden Java-typisch mit `\n` terminiert. Für alle Zahlenwerte kann angenommen werden, dass sie sich mit den einfachen Integer Datentyp darstellen lassen. Die ZugAnfrageNachricht ist wie folgt aufgebaut:

- Die erste Zeile gibt an, mit welcher Strategie der Zug generiert werden soll. Der Wert ist entweder 0 oder 1.
- Die zweite Zeile gibt an, für welchen Spieler ein Zug generiert werden soll. Eine „0“ bedeutet „Generiere einen Zug für Spieler A“, und eine „1“ bedeutet „Generiere einen Zug für Spieler B“.
- Die dritte Zeile gibt die Breite B des Spielfeldes an.
- Die vierte Zeile gibt die Höhe H des Spielfeldes an.
- Die fünfte Zeile gibt an, mit welcher Rechentiefe die Antwort berechnet werden soll.
- Die Werte B und H bestimmen nun das Format der folgenden Zeilen. Jede der nachfolgenden Zeilen beschreibt eine Zeile des Spielbretts.
  - Ein Punkt „.“ beschreibt ein leeres Feld.
  - Ein kleines „o“ beschreibt ein Feld, auf dem ein weißer Stein liegt.
  - Ein kleines „x“ beschreibt ein Feld, auf dem ein schwarzer Stein liegt.

```
0
1
8
8
4
........
........
...o....
..ox....
........
........
........
........
```

Abbildung 3: Eine ZugAnfrageNachricht.

Abbildung 3 zeigt eine Beispielnachricht, die Computergegner 0 mit Rechentiefe 4 als Spieler B auffordert, einen Gegenzug zu der gegebenen Situation auf einem 8x8 Spielfeld zu generieren.

Nachdem die H-te Zeile der Brettbeschreibung eingelesen wurde, berechnet der Server einen passenden Antwortzug und sendet eine ZugNachricht.

Diese besteht aus vier Zeilen. Die erste Zeile enthält den String „OK“ oder den String „FEHLER: “, gefolgt von einer Beschreibung des aufgetretenen Fehlers. Die zweite Zeile enthält einen String, der den Autor des Servers angibt, also Ihren Namen, und den Namen der Strategie. Die dritte Zeile gibt die Zeile auf dem Spielbrett an, in der der Stein gesetzt werden soll. Die vierte Zeile beschreibt entsprechend die Spalte.

Im Falle einer Fehlermeldung sind die Zeilen drei und vier Leerzeilen, d.h. sie bestehen nur aus dem Zeichen für das Zeilenende.

Bei der Nummerierung fangen wir an mit 1 zu zählen, d.h. das obere linke Feld hat die Koordinaten $(1, 1)$.

Abbildung 4 zeigt eine mögliche Antwort auf die oben gegebene Situation.

```
OK
Karl Muster - [Strategie 0 - Defensives Spiel]
3
3
```

Abbildung 4: Eine ZugNachricht

Die Daten der zweiten Zeile der Antwortnachricht solle in der Oberfläche beim Spielen angezeigt werden.

Ein Fehler kann z.B. auftreten, wenn die Nachricht des Clients fehlerhaft war (z.B. syntaktisch oder die Spielsituation war nicht legal). Der Server beendet sofort nach Versenden der Fehlernachricht die Verbindung. Der Client beendet auch die Verbindung, nachdem er eine solche Nachricht erhalten hat. Der Client zeigt die Fehlermeldung an und beendet das laufende Spiel.

Am Ende der Zeilen in beiden Nachrichtentypen sind nach dem eigentlichen Inhalt keine weiteren Leerzeichen erlaubt.

Nach dem Erhalt der ZugNachricht kann der Client nun seinerseits einen Gegenzug bestimmen und erneut eine ZugAnfrageNachricht mit der neuen Spielsituation an den Server schicken.

Die Verbindung zwischen Client und Server bleibt nur für die Dauer einer Anfrage bestehen. Der Client baut bei jeder Anfrage eine neue Verbindung auf. Client und Server schließen die Verbindung nach dem Versand bzw. Empfang der Antwortnachricht.

6 Bonusaufgabe

Wenn Sie Spaß an der Aufgabe gefunden haben, können Sie noch folgende Bonusaufgabe bearbeiten. Die Bearbeitung dieser Aufgabe hat keine Auswirkungen auf die Erteilung des Leistungsnachweises.

Wir beabsichtigen auf der Präsenzveranstaltung, die verschiedenen Programme in einem Turnier gegeneinander antreten zu lassen. Der Sieger des Turniers wird am Ende einen Sachpreis erhalten.

Die Bonusaufgabe besteht darin, ein Werkzeug zur Steuerung eines Turniers zu implementieren. Es gibt keinerlei Einschränkungen bezüglich der Bedienung. Es kann also sowohl ein Kommandozeilenprogramm sein, ein eigenständiges Programm mit graphischer Oberfläche, oder eine Zusatzfunktion Ihres Spielprogramms.

Mit dem Programm soll es möglich sein, automatisch ein Turnier unter einer Menge von Zuggeneratorserver austragen zu können. Man sollte eine Liste von Servern in einer Datei angeben können. Dabei sind IP, Port-, Strategienummern und eine für alle geltende Rechentiefe anzugeben. Das Werkzeug sollte dann diese Server in einem Turnier nach dem Knock-Out-Verfahren gegeneinander antreten lassen. Der Verlauf soll protokolliert werden. Verwenden Sie dabei die Strings aus der ZugNachricht, welche den Autor und die Strategie im Klartext angeben. Berücksichtigen Sie auch Teilnehmerzahlen, die zu Freilosen führen.

Viel Spaß bei der Bearbeitung.

7 Weitere Anforderungen

Vergewissern Sie sich, dass Ihre Lösung unter folgenden Rahmenbedingungen lauffähig ist:

- Benutzen Sie bitte nur Klassen aus der Standard-Bibliothek von Java 1.4.1 SE.
- Die Entwicklung von graphisch ansprechenden **Benutzungsoberflächen** ist ausdrücklich erwünscht, sollte aber nicht auf zusätzliche Bibliotheken zurückgreifen (z.B. OpenGL für Java etc.). Wir sind nicht bereit, entsprechende Bibliotheken bei uns zu installieren und zu testen.
- Ihr Programm darf keine speziellen betriebssystemspezifischen Anforderungen machen (z.B. Makefiles, .bat Dateien, Bash Scripte) unsere Testumgebung besteht aus heterogenen Arbeitsplätze, welche die unterschiedlichsten Betriebssysteme verwenden.
- Wir erwarten bei Rechentiefe 4, auf einem PC mit 1 GHz CPU Takt, bei folgender Spielsituation eine Rechenzeit von weniger als 20 Minuten zur Berechnung des Zuges für den Spieler mit den „x“-Steinen.

```
..............
..............
....o.........
.....x........
......xo..oo...
.....xoxo.x....
....oooxxxxo...
....x.ooxox....
......xxo......
......o.......
..............
..............
..............
..............
..............
```

8 Dokumentationsrichtlinien

Neben dem eigentlichen Programm werden wir uns auch ausführlich mit Ihrer Dokumentation beschäftigen. Sie sollten daher bei der Gestaltung Ihrer Programmdokumentation die gleichen Maßstäbe ansetzen, nach denen Sie beispielsweise eine schriftliche Seminarausarbeitung erstellen würden.

Halten Sie sich bei der Erstellung Ihrer Dokumentation bitte an die folgenden Richtlinien:

Ihre Dokumentation besteht aus einer **kurzen Einführung** und dem **kommentierten Programmcode.**

- In der **Einführung** (ca. 1-5 DIN-A4-Seiten) beschreiben Sie das generelle Vorgehen Ihres Programmes. Skizzieren Sie seine Grundidee, die wichtigsten benutzten Datenstrukturen und den grundlegenden Aufbau Ihres Programmes (z.B. welche Klassen erledigen welche Aufgaben). Gehen Sie aber an dieser Stelle noch nicht auf Implementierungsdetails ein; Ihre Einführung sollte beispielsweise auch von einem Leser nachvollziehbar sein, der zwar grundlegende Programmiererfahrungen besitzt, aber die Programmiersprache Java nicht beherrscht. Vermeiden Sie es, diesen Teil als großen Java-Kommentar vor Ihr Programm zu "quetschen". Vergessen Sie auch nicht, Ihren Namen und Ihre Matrikelnummer auf dem Deckblatt anzugeben.

- Überlegen Sie sich sich einen **einheitlichen Kommentarkopf mit javadoc Elementen**, den Sie vor jeder Klasse und Methode einfügen. In diesem Kommentarkopf beschreiben Sie den Zweck der verwendeten Parameter, welche Aufgabe von der betreffenden Klasse bzw. Methode erfüllt wird, sowie die Einordnung der Klasse/Methode in den Gesamtkontext des Programmes. Vermeiden Sie es, in dem Kommentarkopf ganze "Romane" zu schreiben; je zwei bis drei prägnante Sätze zur Aufgabe und Einordnung der Methode bzw. Klasse sagen mehr als eine halbe Seite Erläuterungstexte. Benutzen Sie die das javadoc-Werkzeug. Beschreiben Sie damit die Parameter und Rückgabewerte.

- **Kommentieren Sie Ihren Programmtext.** Das soll nicht heißen, dass Sie zu jeder Anweisung einen Kommentar schreiben müssen, aber Ihr Programm muss mit Hilfe der Kommentare soweit verständlich sein, dass der Leser Ihre Lösung ohne ein langwieriges Hineindenken in Ihre Java-Konstrukte nachvollziehen kann. Bedenken Sie dabei auch, dass der Leser im Gegensatz zu Ihnen nicht mit den von Ihnen eingeführten Datenstrukturen und Funktionen vertraut ist. Ersparen Sie ihm daher ein Nachblättern der betreffenden Datentypen oder Funktionen, indem Sie bei Wertzuweisungen stichwortartig erklären, was dort bezweckt/ausgeführt wird, wenn dies nicht offensichtlich ist.

  Schließlich noch ein Hinweis: Wenn Sie bemerken, dass innerhalb einer Methode die Notwendigkeit auftritt, mehrere Sätze zur Erläuterung eines Programmabschnittes zu schreiben, dann ist dies ein Anzeichen dafür, dass Ihr Programm noch nicht genügend modularisiert ist. In diesem Fall sollten Sie erwägen, die betreffenden Programmteile als eigenständige Methode auszugliedern. Auf keinen Fall dürfen Sie aber das Problem so "lösen", dass Sie den betreffenden Teil nicht oder nur unzureichend kommentieren!

- Verwenden Sie **aussagekräftige Bezeichner** für Ihre Variablen, Klassen und Methoden: Ein gut gewählter Bezeichner ist so kurz wie möglich, aber lang genug, um seine Funktion verständlich zu beschreiben.

Abkürzungen sind erlaubt, sollten aber "entschlüsselbar" sein. Geben Sie Abkürzungen aus dem normalen Sprachgebrauch den Vorzug vor Eigenkonstrukten (also z.B. "nachf" für "Nachfolger" und nicht "nfolg". Achten Sie auch darauf, dass Abkürzungen aussprechbar bleiben (z.B. "elem" für "Element" und nicht "elmt").

- Benutzen Sie ein **einheitliches Vorgehen bei der Gliederung von Deklarationen und der Einrückung von Programmteilen**. Anweisungsfolgen zwischen geschweiften Klammern werden mit 2 bis 4 Leerzeichen eingerückt; die schließenden geschweiften Klammern stehen auf der gleichen Ebene wie die übergeordneten Konstrukte, zu denen sie gehören. Also zum Beispiel:

```
if (<Bedingung>) {
     <Anweisungsfolge>
} else {
     <Anweisungsfolge>
}
```

  Aber **nicht**:

```
if (<Bedingung>) {
       <Anweisungsfolge>
  } else {
       <Anweisungsfolge>
  }
```

- Vermeiden Sie es, mehr als eine Anweisung in eine Zeile zu schreiben.

9 Hinweise zum Testen des Programms

Das Testen von Programmen ist ein sehr umfangreiches Fachgebiet innerhalb der Informatik, das Stoff genug enthält, um eine ganze Serie von Vorlesungen oder Kurseinheiten zu füllen.

Wir wollen Ihnen hier nur einige Denkanstöße aus diesem Gebiet liefern, um Ihnen das Testen und damit das Abliefern einer (fast, s.u.) korrekten Lösung zu erleichtern.

1. Untersuchungen haben ergeben, dass ein Programm in der Realität niemals fehlerfrei ist. Für ein "frisch programmiertes" Programm (also vor der Testphase) ist es realistisch anzunehmen, dass auf 100 Zeilen Code ca. 4 bis 8 Fehler kommen!

2. Für nicht triviale Programme ist es aus Komplexitätsgründen nicht möglich, einen lückenlosen Korrektheitsbeweis zu führen, d.h. es ist für jedes reale Programm effektiv nicht beweisbar, dass es korrekt ist.

Daraus folgt, dass der Sinn des Testens eines Programms nicht darin bestehen kann, die völlige Fehlerfreiheit eines Programms nachzuweisen, da dies nach 1. extrem unwahrscheinlich und nach 2. objektiv ohnehin nicht beweisbar ist. Daher definiert man das Testen häufig wie folgt:

Testen bedeutet, ein Programm mit der Absicht auszuführen, Fehler zu finden.

Eine Testeingabe wird als erfolgreich bezeichnet, wenn sie das Programm zu falschem Verhalten verleitet (fehlerhafte Ausgabe, Programmabbruch etc.). Hingegen betrachtet man eine Testeingabe als nicht erfolgreich, wenn sich das Programm korrekt verhält (korrekte Ausgabe oder Zurückweisung einer Eingabe, die außerhalb des gültigen Bereiches liegt).

Die Teststrategie besteht also darin, gezielt möglichst viele Fehler in dem Programm zu finden. Damit kann man zwar nicht beweisen, dass ein Programm überhaupt keine Fehler mehr enthält (s.o.), das Vertrauen in die Zuverlässigkeit des Programmes wird jedoch mit der steigenden Anzahl nicht erfolgreicher Testfälle (= korrekter Reaktionen des Programmes) und daraufhin eliminierter Fehler erhöht.

Nachfolgend wollen wir Ihnen einige Anregungen geben, wie Sie Ihr Programm effektiv testen können:

- Zunächst einmal: Lassen Sie sich nicht dazu verleiten, Programmfehler als persönliche Fehlleistung aufzufassen (nach dem oben Gesagten sollte klar sein, dass sich Fehler zwangsläufig und unvermeidbar in Programme einschleichen)! Im Testen unerfahrene Programmierer empfinden das Testen häufig als unangenehm, da jeder Fehler als Rückschlag bzw. "Versagen" aufgefasst wird, und die Konsequenz ist häufig, dass entweder überhaupt nicht oder nur sehr halbherzig (mit korrekten, für das Programm harmlosen Testfällen) getestet wird. Sehen Sie die ganze Sache positiv: Jeder Fehler, den Sie finden und beheben, macht Ihr Programm ein Stück perfekter!

- Wählen Sie Ihre Testeingaben effizient aus. Ein geeignetes Verfahren ist z.B. die **Grenzwertanalyse**. Dabei ermitteln Sie zunächst für einen Eingabewert alle gültigen und ungültigen Werte, und wählen dann jeweils einen Repräsentanten aus, der gerade noch gültig ist ("auf der Grenze liegt") und je einen Repräsentanten, der knapp außerhalb der Grenze liegt und damit ungültig ist. Wenn Sie z.B. eine Funktion testen, die einen Text mit einer Länge von 1...40 Zeichen als Eingabe bekommt, würden Sie nach der Grenzwertmethode jeweils einen Text der Länge 1 und einen Text der Länge 40 als gültige Eingabe sowie Texte der Längen 0 bzw. 41 als ungültige Werte auswählen. Die

ungültigen Werte nimmt man in die Testmenge auf, um zu sehen, ob das Programm auch auf fehlerhafte Eingaben sinnvoll reagiert (indem es z.B. eine Warnung ausgibt o.ä). Falls solche Testfälle nicht vorgesehen werden, kann es vorkommen, dass zum Beispiel ein Programm in inneren Modulen später aus "unerklärlichen" Gründen abstürzt (weil im Verlauf der Berechnung intern ein ungültiger Wert erzeugt wurde), oder bei bestimmten Eingaben nur unsinnige (weil undefinierte) Ausgaben erscheinen.

- Eine ähnliche Strategie besteht darin, nach **Spezialfällen** (neutrale Elemente, Definitionslücken wie die berühmte Division durch Null) Ausschau zu halten. Arbeitet ein Sortieralgorithmus z.B. auch korrekt, wenn die Liste der zu sortierenden Worte leer, einelementig oder bereits sortiert ist?

- Versuchen Sie, eine Menge von Testeingaben so zu wählen, dass insgesamt alle Programmteile in möglichst **allen Kombinationen** durchlaufen werden.

  Beispiel:

  ```
  Zum Testen der Abfrage
  if (a=3) {
     if (b=4) {
      <Anweisungsblock>
     } else {
         <Anweisungsblock>
     }
   } else {
     <Anweisungsblock>
   }
  ```

  Zum Testen sind zumindest die Wertekombinationen $(a = 3, b = 4)$, $(a = 3, b <> 4)$, $(a <> 3, b$ beliebig) in die Testmenge aufzunehmen.

- Testen Sie Ihr Programm **klassenweise**. Dies bedeutet, dass Sie die zu testende Klasse direkt mit passenden Testwerten aufrufen und die zurückgegebenen Werte überprüfen. Eine in das Gesamtprogramm integrierte Klasse lässt sich nicht mehr zielgerichtet testen, da die Eingabe des Hauptprogrammes auf dem "Aufrufweg" zur Zielklasse oft so stark transformiert wird, dass man für die Klasse keine individuellen Testwerte mehr erzeugen kann. Gleiches gilt natürlich für die Ausgabe des Moduls, die bis zur endgültigen (Bildschirm-)ausgabe im kompletten Programm so weit umtransformiert werden kann, dass sie zu dem betreffenden Modul nicht mehr eindeutig in Beziehung gesetzt werden kann.

Die von uns in der Präsenzphase zur Vorführung Ihres Programmes ausgewählte Testeingabe wird unter anderem auch einige Grenzfälle enthalten, die nach den oben beschriebenen Methoden aufgebaut sind. Wir behalten uns vor, ein unter diesen Voraussetzungen extrem instabiles oder fehlerhaftes Programm als nicht ausreichend zurückzuweisen.

10 Sonstiges

Schauen Sie bitte regelmäßig (spätestens jede Woche) in die Newsgroup zum Praktikum. Zum einen haben Ihre Kommilitoninnen und Kommilitonen eventuell auch Anregung für Ihre Arbeit.

Wir wünschen Ihnen viel Erfolg bei diesem Praktikum.

Ihre Praktikumsbetreuer

Dominic Heutelbeck

Daniela Keller

Jörg Roth