

Aufgabenstellung des Programmierpraktikums im Wintersemester 2004/2005

1 Willkommen in der Firma Propra05

Mit Ihrer Anmeldung zum Programmierpraktikum sind Sie für dieses Semester Mitarbeiter in der Firma Propra05 geworden. Ihren Unterlagen haben wir entnommen, dass Sie sich intensiv mit der Programmiersprache Java beschäftigt haben und sich weiter beschäftigen möchten. Und dazu werden Sie jetzt bei uns ausreichend Gelegenheit haben.

Der Geschäftsführer, Herr Dr. Roth, ist Ihr Ansprechpartner für alles, was mit Ihrer Arbeit zu tun hat. Der sehr fleißige Außendienstmitarbeiter Herr Heutelbeck hat schon Kontakte mit der Spielefirma CreativeGames im Sauerland aufgenommen. Die Verhandlungen sind so weit gediehen, dass bereits eine Art Pflichtenheft erstellt werden konnte. Der Auftrag lautet, ein Gobang-artiges Computerspiel zu entwickeln, das auf einem Spielfeld aus Sechsecken gespielt wird. Der vorläufige Produktname ist Hexabang, aber vielleicht fällt uns zusammen noch ein besserer Namen ein.

Wie nicht anders zu erwarten, hat der Kunde besondere Wünsche. Damit Sie sich darauf einstellen können, beschreibe ich Ihnen Ihren Kunden etwas näher.

Die Firma CreativeGames ist ein alteingesessenes Familienunternehmen, das früher Holzspielzeug und Brettspiele produzierte. Da hierfür der Absatz rückläufig ist, beschlossen die Inhaber, zu den vorhandenen Brettspielen auch Computerspiele anzubieten. Dabei ist dem Seniorchef besonders wichtig, dass die Spiele eine ansprechende Darstellung auf dem Bildschirm haben, man das ehemalige Brettspiel wieder erkennt und die Spieler mit einer sinnvollen Benutzerführung durch das Spiel geleitet werden. Wie das Programm programmiert wird, ist ihm gleichgültig, Hauptsache, er kann gut und schnell damit spielen.

Der Juniorchef, Herr Müller-Lüdenscheid, hat ein paar Semester Informatik an der FernUniversität studiert. Er wird sich daher den Quellcode genau anschauen und erwartet, dass er ihn auch verstehen kann. Daher ist es sehr wichtig, dass Sie den Quellcode perfekt dokumentieren. Eventuell möchte Herr Müller-Lüdenscheid auch nachträglich Änderungen durchführen und das Spiel um zusätzliche Eigenschaften ergänzen. Da er gerade einen Javakurs gemacht hat, hat er genaue Vorstellungen davon, was für das Programm verwendet werden soll: Aus Zeitgründen hat er keine Lust, irgendwelche zusätzlichen Bibliotheken einzubinden - das Programm muss mit seinem Standard-Java (Java Version 1.4) laufen. Außerdem möchte er, dass für die Oberflächengestaltung des Spiels Swing verwendet wird – er weiß zwar nicht, wie das genau funktioniert, glaubt aber, dass es toll ist. Herr Heutelbeck hat lange mit ihm diskutiert, aber Herr Müller-Lüdenscheid lässt sich von seinen Vorstellungen nicht abbringen.

Ihre Aufgabe ist, dass Hexabang gemäß dem unten beschriebenen Pflichtenheft zu implementieren. Das ist eine spannende, aber keineswegs einfache Aufgabe. Sie müssen die Arbeit alleine durchführen, denn wir wollen am Ende eine größere Anzahl ver-

schiedener Implementierungen haben, um daraus diejenige aussuchen zu können, die unserem Kunden am besten gefällt. Allerdings dürfen Sie Ihre Probleme bzgl. Ihrer Arbeit mit uns, den festen Mitarbeitern der Firma Propra05, und den anderen Programmierern in unserer Newsgroup diskutierenden. Nutzen Sie diese Möglichkeit gleich von Anfang an, damit wir alle ein gutes Team bilden und uns schnell kennen lernen! Sie haben über drei Monate für die Erstellung des Programms, aber nie verrinnt die Zeit so schnell wie beim Programmieren und Testen. Also verabschieden Sie sich von Familie, Freundeskreis und Hobbies und widmen sich ganz dieser wichtigen Aufgabe, dem Hexabang.

Die festen Mitarbeiter unserer Firma sind (in der Reihenfolge ihrer Gehälter)

- Geschäftsführer Dr. Jörg Roth
- Stellvertretender Geschäftsführer Dr. Wolfgang Wilkes
- Assistentin des Geschäftsführers Dr. Daniela Keller
- Chefprogrammiererin Inga Saatz
- Leitender Programmierer (als studentischer Mitarbeiter) Jan Quadflieg
- Chefberater (als studentischer Mitarbeiter) Michael Paap
- Sekretariat Sibylle Schick (kann oft helfen, da sie auch im Prüfungsamt arbeitet)
- (studentische) Mitarbeiter in der Firma Praktische Informatik II:
 - Nils Kunstmann
 - Daniel Weinand

Um Ihre Java-Probleme und eventuelle Schwierigkeiten mit der Aufgabenstellung wird sich vor allem Herr Paap kümmern. Für alles Organisatorische und als "Kummerkasten" bei sonstigen Problemen bin ich (Daniela Keller) zuständig. Das letzte Wort hat immer Herr Roth (betrachten Sie das als Regel).

Wir wünschen Ihnen viel Erfolg und Freude bei der Arbeit!

Für das ganze Team

Ihre Daniela Keller

2 Terminplan, Ansprechpartner und Sonstiges

- Offizieller Arbeitsbeginn: Oktober 2004
- Abgabe des fertigen und getesteten Programms: Die Programme müssen bis Donnerstag, 13.01.2005, 23.59 Uhr, bei uns eingegangen sein, da unser Kunde am Freitag gegen 10 Uhr zur Besprechung des weiteren Vorgehens vorbeischaud. Planen Sie daher die typischen Pannen wie ein zusammengebrochenes Internet, einen defekten Computer, einen missgelaunten FernUni-Mailserver und sonstige Katastrophen mit ein!
- Sichtung der Ergebnisse durch unser Propra05Team: ab dem 14.01., Dauer: circa drei bis vier Wochen.

- Korrekturphase: Sie beginnt, nachdem Sie die vorläufige Bewertung Ihrer Arbeit erhalten haben – gegebenenfalls müssen Sie noch nachbessern, also halten Sie sich ab Mitte Februar noch Zeit frei.
- Treffen aller Programmierer am 05./06.03.2005 in Hagen, um die fertigen Produkte vorzustellen und die besten Programme zu finden.
 - Das Ganze hat eher informellen Charakter, d.h. Sie brauchen keinen Vortrag mit Powerpoint oder Ähnlichem vorzubereiten, es genügt, wenn Sie Ihr Programm perfekt kennen und uns auch noch die versteckteste Variable erklären können.
 - Gesucht sind zwei Siegerprogramme: Die Hexabang-Programme werden gegeneinander antreten und damit das Programm ermitteln, das am besten spielt. Des Weiteren vergibt eine von uns benannte Jury einen Design-Preis an das optisch ansprechendste Programm.
 - Es ist recht wahrscheinlich, dass zu diesem Zeitpunkt noch kleinere Sonderwünsche des Kunden anfallen. Diese Erweiterungen Ihres Programms nehmen Sie dann direkt bei unserem Treffen vor.
 - Sie müssen nur an einem der Termine anwesend sein. Die Terminplanung für diese "Präsenzphase" erfolgt nach der Sichtung der Hexabang-Programme. Bitte Terminprobleme wie Klausuren, Hochzeiten und Arbeitseinsätze für andere Firmen rechtzeitig bekannt geben, damit wir eine für alle positive Lösung finden können.

3 Wann erhalten Sie Ihr Gehalt?

Um Ihr Gehalt in Form eines (unbenoteten) Leistungsnachweises zu erhalten, erwarten wir von Ihnen:

1. Eigenständige Implementierung eines fehlerfreien Programms gemäß dem nachfolgenden Pflichtenheft

Wichtig: Gruppenlösungen sind nicht zulässig.

2. Einsendung des Programms per Email bis zum 13.01.2005 unter Berücksichtigung der von uns angegebenen Programmier- und Dokumentationsrichtlinien.

Email-Adresse: daniela.keller@fernuni-hagen.de

3. Unser Kunde erwartet, dass zum Hexabang-Programm 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.

Es ist ganz wichtig für uns, dass Sie hier keine Fehler machen, denn oft wird die Arbeit einer Firma auch schon danach bewertet, ob alles so verpackt ist, wie es der Kunde erwartet. Wenn wir hier gute Arbeit leisten, schaut sich der Kunde unser Programm mit viel mehr Freude und Zufriedenheit an.

4. Diesen Punkt schreiben wir nur hinzu, weil unsere Rechtsabteilung darauf besteht. Wir selbst sind sicher, nur verantwortungsvolle Programmierer zu haben.

"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."

4 Hexabang: Das Pflichtenheft

4.1 Spielregeln

Hexabang ist ein Brettspiel für zwei Personen (Spieler A und Spieler B). Es wird auf einem Spielbrett gespielt, das sich aus sechseckigen einzelnen Feldern zusammensetzt, die ein flächendeckendes Parkett bilden.

Als Spielsteine werden schwarze und weiße Chips verwendet. Es gibt hinreichend viele Spielsteine um das gesamte Spielfeld zu füllen. 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 setzen. Der Spieler A beginnt. Beim ersten Spielzug darf ein Stein auf ein beliebiges Feld des Spielbrettes gelegt werden.
- Spieler A setzt weiße Steine, Spieler B die schwarzen. Steine dürfen nur auf leere Felder des Spielbrettes gelegt werden. Gesetzt wird *in* ein Sechseck-Feld, also nicht auf eine Kante oder einen Eckpunkt.
- Ab dem zweiten Spielzug dürfen Steine nur auf solche Felder gelegt werden, von denen mindestens eines der sechs angrenzenden Nachbarfelder bereits durch einen Stein beliebiger Farbe belegt ist (siehe Abb. 4-1).

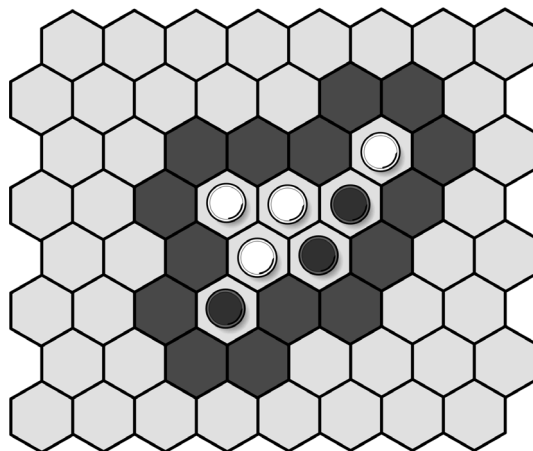


Abb. 4-1. Mögliche Züge

- Steine dürfen nicht auf dem Spielbrett verschoben werden. Einmal gelegte Steine dürfen nicht wieder vom Spielbrett entfernt werden.
- Sobald in einer der drei Hauptrichtungen fünf Steine einer Farbe eine durchgehende Reihe bilden, ist das Spiel beendet, und der Spieler mit der entsprechenden Farbe hat gewonnen (Abb. 4-2). Sind alle Felder des Spielbrettes belegt und existieren keine Fünferreihen, dann endet das Spiel mit einem Unentschieden.

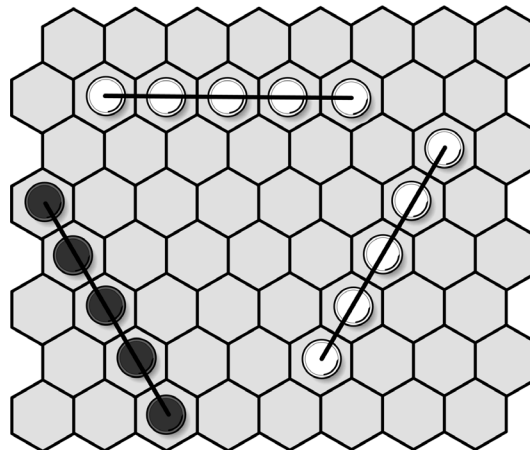


Abb. 4-2. Mögliche Fünferreihen

4.2 Spielfeldkoordinaten

Das Programm muss die Felder des Spielfeldes adressieren können. Dies wird benötigt, damit ein Spielfeld effizient gespeichert werden kann. Wir benötigen aber auch eine eindeutige Adressierung, wenn wir Spielsituationen später über ein Netzwerk transportieren wollen. Folgendes Adressierungsschema ist für die externe Kommunikation verpflichtend.

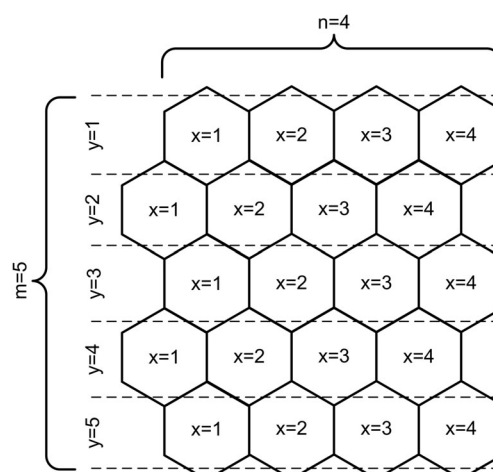


Abb. 4-3. Abbildung von x-y-Koordinaten auf das Spielfeld

Abb. 4-3 zeigt, wie die Koordinaten (x, y) auf das sechseckige Parkett des Spielfeldes abgebildet werden. Die y -Koordinate spezifiziert, wie in einem rechteckigen Parkett, die Zeile. Die x -Koordinate ist etwas problematischer, da sich die Sechsecke von Zeile zu Zeile verschieben: die Zeilen mit gerader y -Koordinate sind gegenüber den anderen

Zeilen um eine halbe Zelle nach links verschoben. Die gemäß diesem Schema angeordneten ersten Sechsecke jeder Zeile tragen die Nummer 1. Alle Zeilen (auf ungeraden und auf geraden y-Koordinaten) sind gleich lang.

Damit die im Kommunikationsprotokoll spezifizierten Spielfelder (siehe Abschnitt 4.3.3) von allen Programmen in derselben Weise interpretiert werden, ist eine einheitliche Abbildung der x-y-Koordinaten auf das Sechseckparkett unerlässlich.

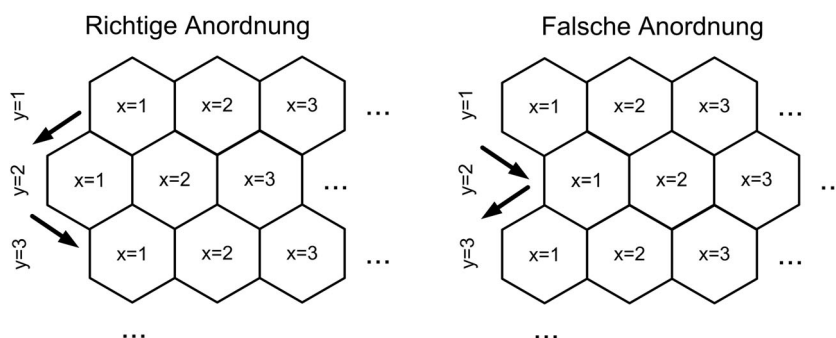


Abb. 4-4. Anordnung der Zeilen

Abb. 4-4 soll einem möglichen Fehler vorbeugen. Die linke Abbildung zeigt eine korrekte Zuordnung, während die rechte eine fehlerhafte Zuordnung darstellt. Das von Ihnen zu implementierende Kommunikationsprotokoll muss unbedingt die in der linken Darstellung gewählte Zuordnung verwenden. Sinnvollerweise speichern Sie auch die Spielsteine in einer analogen zweidimensionalen Struktur ab – im Gegensatz zum Kommunikationsprotokoll können Sie bezüglich der internen Speicherung des Spielfeldes aber auch andere Strukturen verwenden. Wichtig ist jedoch, dass die externe Kommunikation sich an den von uns dargestellten Standard hält.

4.3 Anforderungskatalog

Die Hauptbestandteile der Implementation unseres interaktiven Hexabang-Spiel sind

1. die Zuggeneratoren für die Computergegner und
2. die graphische Oberfläche.

Sie sollen 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.

Mit "Computer" ist der Computergegner der lokal gestarteten Anwendung gemeint, mit "Zuggeneratorserver" ein über das Netzwerk erreichbarer Computergegner (Erläuterungen folgen weiter unten in Abschnitt 4.3.3).

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.

Diese Brettgrößen beziehen sich auf die in Abschnitt 4.2 eingeführte Abbildung der Koordinaten auf das Spielfeld (siehe insbesondere Abb. 4-3).

Es müssen Eingabefelder zur Verfügung gestellt werden, um

- die Netzwerkadresse,
- Portnummer,
- Rechentiefe und
- Nummer (0 oder 1, siehe Abschnitt Zuggeneratorserver) des jeweiligen Zuggeneratorservers

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

4.3.1 Die graphische Oberfläche

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

1. Eine graphische Repräsentation des Spielbretts, über welche ein menschlicher Spieler seine Züge durch Klicken durchführen kann.
2. Ein Menü mit folgenden Möglichkeiten:
 - a.) ein neues Spiel zu starten,
 - b.) das Spiel zu beenden,
 - c.) einen Zugvorschlag abzurufen,
3. Einen Dialog, der beim Starten eines neuen Spieles geöffnet wird und zur Konfiguration des Spiels dient. Hier wird beispielsweise die Rechentiefe des Computergegners oder des Zuggeneratorservers eingestellt.

4. Eine Anzeige, die beim Spielende das Spielergebnis mitteilt.

Hier sind Ihrer Kreativität keine Grenzen gesetzt. Sie können z.B. ein Dialogfenster passend gestalten oder durch Farbwechsel und blinkende Steine den Gewinn graphisch auf dem Spielfeld darstellen.

Wichtiger Hinweis:

Die Spielfläche sollte optisch ansprechend gestaltet sein. Eine Implementierung der Spielfläche aus einem Raster von Schaltflächen (Buttons) wird nicht akzeptiert.

Der unter 2c) 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.

4.3.2 Die Computergegner

Spiele wie Hexabang 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:

1. Berechnung aller möglichen Züge,
2. Traversierung des Spielbaums,
3. 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.

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 gehen Sie 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 Mini-max-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. 4-5) sei Spieler A am Zug. Es soll gezeigt werden, wie die Stellungswerte von unten nach oben ermittelt werden. Das 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 Stellungsbewerter.

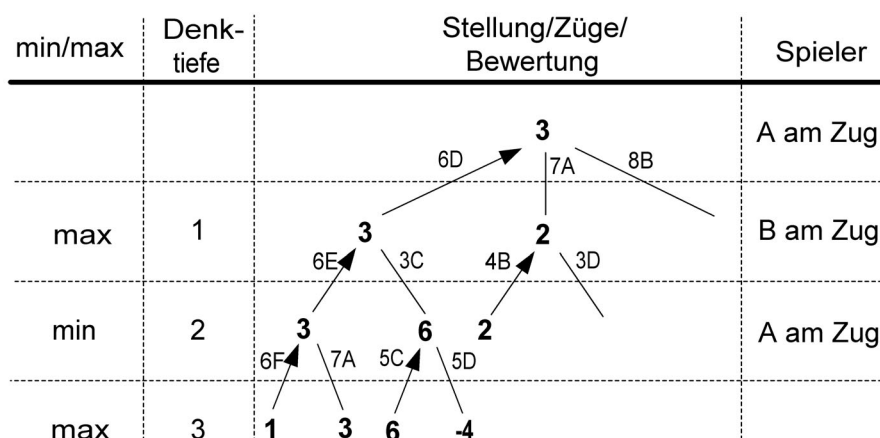


Abb. 4-5. Ein Spielbaum

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 resultierende 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 maximieren. 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 Stellungsbewerter 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 "Alpha-beta-Schnittes" lässt sich die Geschwindigkeit drastisch erhöhen. Der Schnitt reduziert die Anzahl der betrachteten Knoten im Baum.

Die Idee des Alpha-beta-Schnittes 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 alpha und beta, die 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 beta kleiner als alpha 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 vorausgegangen 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:

```

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,  $-\infty$ ,  $\infty$ , true)
```

aufgerufen. Bitte dokumentieren Sie Ihre Version des Algorithmus im Detail und begründen Sie Ihre Designentscheidungen.

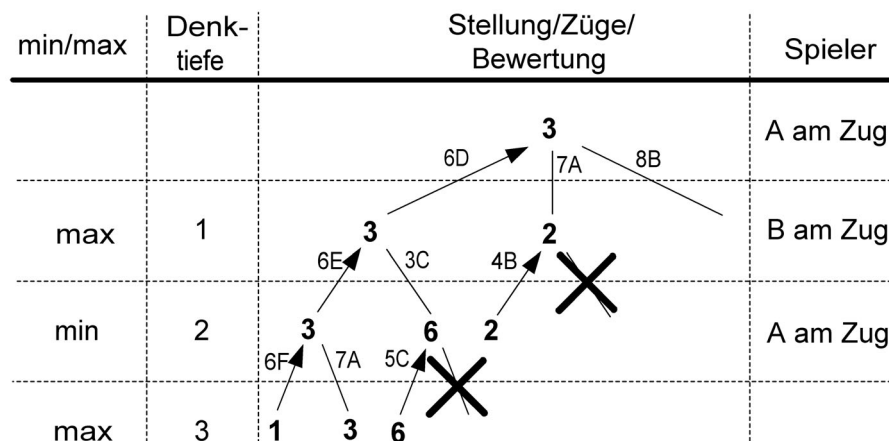


Abb. 4-6. Der Spielbaum mit eingezeichneten Schnitten

4.3.3 Die Zuggeneratorserver (ZGS)

Beim Start soll das Programm einen Zuggeneratorserver starten, um als Spielgegner über das Netzwerk zur Verfügung zu stehen. 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 Zugsberechnung 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 SHP, das "Simple Hexabang Protocol". SHP besteht aus zwei Nachrichtentypen, einer **ZugAnfrageNachricht** und einer **ZugNachricht**. Die Nachrichten sind zeilenorientierte ASCII-Texte. Alle Zeilen werden Java-typisch mit `\n` terminiert.

Achtung: An dieser Stelle soll vor einer Falle gewarnt werden. Benutzen Sie auf keinen Fall "println"-Methoden, um den Zeilenvorschub zu generieren. Diese Methoden generieren auf unterschiedlichen Plattformen (Linux, Windows) unterschiedliche Zeichenfolgen. Möglicherweise funktioniert Ihr Programm daher bei Ihnen aber nicht bei uns. Bitte senden Sie auf jeden Fall das `\n` als einzelnes Byte. In einem Praktikum mit vergleichbarer Aufgabenstellung war ein falscher Zeilenvorschub für 90% aller Kommunikationsprobleme verantwortlich.

Für alle Zahlenwerte kann angenommen werden, dass sie sich mit dem 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.

Abb. 4-7 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. Die Zeilen werden gemäß dem Zeilenschema aus Abschnitt 4.2 ausgewertet. Das Feld entspricht somit der in Abb. 4-8 dargestellten Situation.

Im Falle einer Fehlermeldung sind die Zeilen drei und vier Leerzeilen, d.h. sie bestehen nur aus dem Zeichen für das Zeilenende. Abb. 4-9 zeigt eine mögliche Antwort auf die oben gegebene Situation.

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

Abb. 4-9. Eine Zugnachricht

Die Daten der zweiten Zeile der Antwortnachricht sollen 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 Fehlermeldung 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.

An dieser Stelle soll noch einmal ausdrücklich darauf hingewiesen werden, dass das Kommunikationsprotokoll quasi bis auf das Byte genau implementiert werden muss. Dies ist wichtig, damit Programme verschiedener Teilnehmer über das Netzwerk miteinander spielen können. Der Test der Netzwerkfunktionalität ist einer der wesentlichen Tests, dem das eingesendete Programm unterworfen wird.

4.4 Tipps für eine gute Hexabang-Strategie

Teil der Aufgabe ist, einen *ernstzunehmenden* Computergegner zu entwickeln. Der Alpha-Beta-Algorithmus arbeitet nur so gut wie die Stellungsbewertung. Damit Sie eine sinnvolle Bewertung entwickeln können, müssen Sie selbst das Spiel hinreichend gut beherrschen. Bitte führen Sie deshalb Testspiele durch, damit Sie ein Gefühl für den Spielverlauf erhalten und bewerten können, was gute und was schlechte Spielsituationen sind. An dieser Stelle wollen wir schon einige Tipps zum Spielverlauf geben um Ihnen die Einarbeitung zu erleichtern:

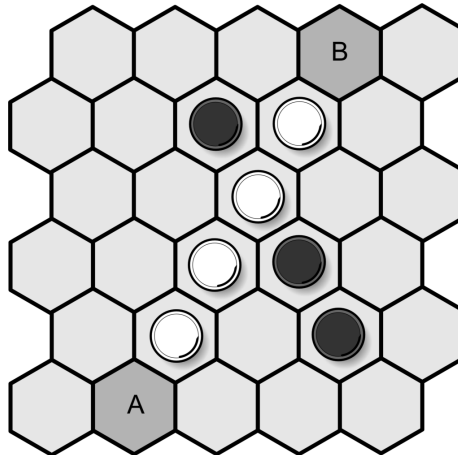


Abb. 4-10. Tipp 1: eine typische Gewinnsituation

Abb. 4-10 zeigt eine typische Gewinnsituation für weiß. Weiß hat vier Steine in eine Reihe gebracht, die von keiner Seite blockiert werden. Schwarz kann zwar jetzt noch auf das Feld A oder B setzen, jedoch kann Weiß die Fünferreihe in jedem Fall kompletieren. Es gilt also, solche offenen Viererreihen beim Gegner zu vermeiden oder selbst aufzubauen.

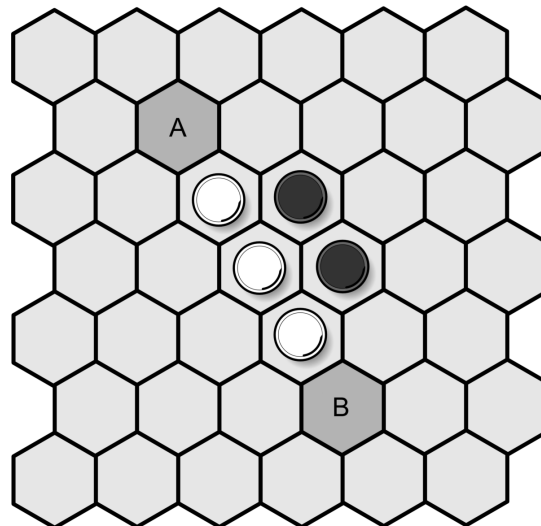


Abb. 4-11. Tipp 2: offene Dreierreihe

Abb. 4-11 zeigt eine offene Dreierreihe für weiß. Schwarz ist am Zug und muss jetzt entweder auf A oder B setzen, ansonsten ist der Verlust unvermeidbar. Setzt er nämlich weder auf A noch auf B, so kann Weiß eine offene Viererreihe produzieren und gewinnt damit sicher. Damit kann man folgende Regel ableiten: offene Dreierreihen müssen vom Gegner immer auf einer Seite blockiert werden.

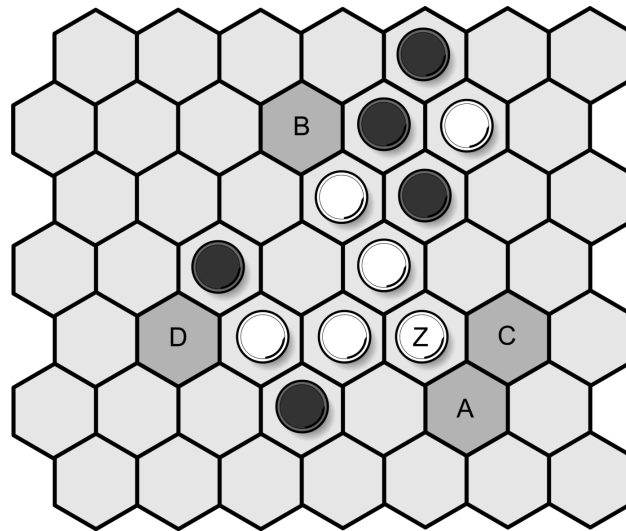


Abb. 4-12. Tipp 3: eine Falle

In Abb. 4-12 hat Weiß eine raffinierte Falle aufgebaut. Durch seinen letzten Zug Z hat Weiß zwei offene Dreierreihen aufgebaut. Schwarz ist am Zug, kann aber nur entweder die Dreierreihe zwischen A und B oder zwischen C und D blockieren. Weiß gewinnt damit sicher.

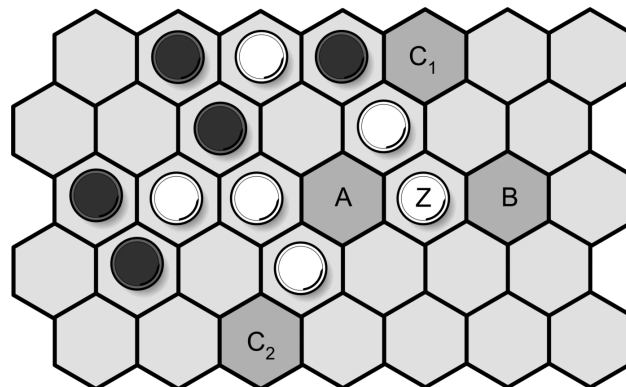


Abb. 4-13. Tipp 4: eine weitere Falle

Eine kompliziertere Situation zeigt Abb. 4-13. Hier versucht Weiß wieder eine Falle aufzubauen, Schwarz kann die Gefahr jedoch noch abwenden. Würde Schwarz nicht sinnvoll reagieren, könnte Weiß durch Setzen von A eine einseitig begrenzte Viererreihe aufbauen. Diese kann zwar von Schwarz durch Setzen von B begrenzt werden, jedoch würde dann noch eine offene Dreierreihe existieren. Die einfachste Art, dies zu verhindern ist, wenn Schwarz auf A setzt, damit verhindert er gleichzeitig die Vierer- und offene Dreierreihe. Auch durch Setzen von B, C1 oder C2 würde er den sicheren Sieg von Weiß verhindern. Hier könnte Weiß aber durch Vervollständigen der noch möglichen Reihe den nächsten Zug von Schwarz erzwingen und damit weiterhin den Druck auf Schwarz aufbauen.

Wichtig an dieser Situation ist, dass auch unterbrochene Reihen schon einen "Wert" haben können, wenn man durch einen gezielten Zug diese in mehrere vollständige Reihen verwandeln kann.

5 Weitere Anforderungen

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

- Benutzen Sie bitte nur Klassen aus der Standard-Bibliothek von Java 1.4.2 (Standard Edition).
- 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 Skripte). Unsere Testumgebung besteht aus heterogenen Arbeitsplätzen, welche die unterschiedlichsten Betriebssysteme verwenden.
- Wir erwarten bei Rechartiefe 4, auf einem PC mit 1 GHz CPU Takt, bei folgender Spielsituation eine Rechenzeit von weniger als 3 Minuten zur Berechnung des Zuges für den Spieler mit den "x"-Steinen. Diese Grenze ist sehr großzügig bemessen – bei einer vergleichbaren Aufgabenstellung haben Programme von Studenten Antwortzeiten im Sekundenbereich produziert.

```
.....  
.....  
.....  
.....  
.....  
.....OXO.....  
...O.O.X.....  
...XOOOX.....  
...XOOX.....  
..XOXXO.....  
.....O.X.....  
.....X.....  
.....  
.....
```

Diese Situation entspricht dem Spielfeld in Abb. 5-1.

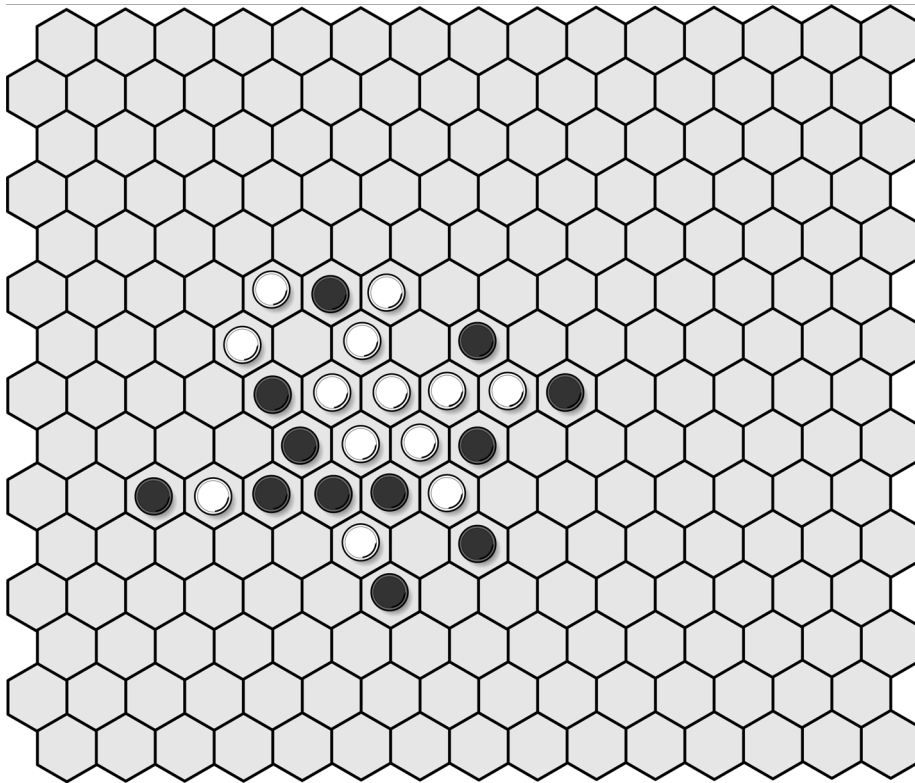


Abb. 5-1. Die Testsituation

6 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 Seminaarausarbeitung 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 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 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.

7 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 eliminierten 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:

```
if (a=3) {
    if (b=4) {
        <Anweisungsblock>
    } else {
        <Anweisungsblock>
    }
} else {
    <Anweisungsblock>
}
```

Zum Testen dieser Abfrage 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 Hexabang-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.