Aufgabenstellung
des Programmierpraktikums
im Wintersemester 2006/2007

1 Willkommen in der Firma Propra06/07

Mit Ihrer Anmeldung zum Programmierpraktikum sind Sie für dieses Semester Mitarbeiter in der Firma Propra06/07 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 Betermieux, ist Ihr Ansprechpartner für alles, was mit Ihrer Arbeit zu tun hat. Der sehr fleißige Außendienstmitarbeiter Herr Marty 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 Computerspiel zu entwickeln, das auf einem quadratischen Spielfeld gespielt wird. Der vorläufige Produktname ist Kaskade, 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 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 1.5) 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 Marty hat lange mit ihm diskutiert, aber Herr Müller-Lüdenscheid lässt sich von seinen Vorstellungen nicht abbringen.

Ihre Aufgabe ist, Kaskade 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 verschiedener 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 Propra06/07, und den anderen Programmierern in unserer Newsgroup diskutierenen. Nutzen Sie diese Möglichkeit gleich von Anfang an, damit wir alle ein gutes Team bilden und uns schnell kennen lernen! Sie haben 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, der Implementierung des Kaskade-Spiels.

Die festen Mitarbeiter unserer Firma sind:

Geschäftsführer	Stefan Betermieux
Stellvertretender Geschäftsführer	Ulrich Marty

Leitender Programmierer (als studentischer Mitarbeiter) Michael Paap

Um Ihre Java-Probleme und eventuelle Schwierigkeiten mit der Aufgabenstellung wird sich vor allem Herr Paap kümmern. Für alles Organisatorische und bei sonstigen Problemen bin ich (Stefan Betermieux) zuständig.

Wir wünschen Ihnen viel Erfolg und Freude bei der Arbeit!

Für das ganze Team

Ihr Stefan Betermieux

2 Terminplan, Ansprechpartner und Sonstiges

- Offizieller Arbeitsbeginn: Oktober 2006
- Abgabe des fertigen und getesteten Programms: Die Programme müssen bis Sonntag, 31.01.2007, 23.59 Uhr, bei uns eingegangen sein, da unser Kunde am Donnerstag gegen 10 Uhr zur Besprechung des weiteren Vorgehens vorbei schaut. 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: ab dem 01.02.2007,

  Dauer: circa zwei bis drei Wochen
- Korrekturphase: Sie beginnt, nachdem Sie die vorläufige Bewertung Ihrer Arbeit erhalten haben – gegebenenfalls müssen Sie noch nachbessern, also halten Sie sich ab Februar noch Zeit frei.
- Treffen aller Programmierer am Sa/So 10. und 11.03.2007 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 Kaskade-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 beiden Präsenztage anwesend sein. Die Terminplanung für diese "Präsenzphase" erfolgt nach der Sichtung der Kaskade-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. Hochladen des Programms mit dem bereitgestellten Webformular http://sirius.fernuni-hagen.de
   bis zum 31.01.2007 unter Berücksichtigung der von uns angegebenen Programmier- und Dokumentationsrichtlinien.
3. Sie müssen gegen die schwächste von uns bereitgestellte Strategie gewinnen können. Die Adresse des Rechners, auf dem diese Strategie läuft, entnehmen Sie bitte der Propra-Webseite.
4. Sie zeigen uns, dass Sie Ihr Programm verstehen und damit umgehen können, indem Sie in der Präsenzphase eine Präsenzaufgabe umsetzen und ein kleines Zusatzfeature in Ihr Programm einbauen.
5. Es wurde eine Newsgruppe für das Praktikum eingerichtet. Dort können Sie Fragen stellen, es wird allerdings auch von Ihnen erwartet in regelmäßigen Abständen dort hineinzuschauen und auf Ankündigungen zu achten. Wenn Sie Ihren Mitstudierenden helfen, finden wir das natürlich auch super:

   ```
   feu.informatik.kurs.1580+82+84.diskussion.ws
   ```

6. Unser Kunde erwartet, dass zum Kaskade-Programm die folgenden Komponenten gehören:

   a. das vorcompilierte Programm als .jar-Datei, welches sich durch ein einfaches java -jar IHR_NAME.jar ausführen und testen lassen muss.
   Nachdem Sie eine Aufgabe gelöst haben und Ihren Java-Code übersetzt und getestet haben, erstellen Sie ein ausführbares Archiv mit allen .class-Dateien. Ausführbare jar-Dateien gibt es ab der Java-Version 1.2. Dazu muss in der jar-Datei vermerkt werden, welche Klasse diejenige ist, deren main-Methode ausgeführt werden soll. Für diesen Zweck ist eine so genannte Manifest-Datei zu erstellen, in welcher der Name dieser Klasse als main-Klasse vermerkt ist. Wenn z.B. diese Klasse Kaskade heißt und im Paket propra.main liegt, dann lautet der Inhalt der Manifest-Datei wie folgt:

   ```
   Main-Class:  propra.main.Kaskade
   ```

   Ansonsten muss nichts zusätzlich in der Manifest-Datei stehen. Die Manifest-Datei muss beim Erstellen des jar-Datei angegeben werden. Angenommen, die Manifest-Datei wurde manifest.mf (der Standardname für Manifestdateien) genannt, dann würde folgender Aufruf eine ausführbares Archiv erzeugen (die Reihenfolge ist zu beachten):

   ```
   jar cmf manifest.mf IHR_NAME.jar *.class
   ```

   b. der Quelltext mit erstelltem javadoc als .zip-Datei mit dem Namen IHR_NAME.zip

   c. Die Javadoc-Dateien müssen sich in einem gesonderten Verzeichnis docs befinden.

d. Die Dokumentation als ASCII-, RTF- oder PDF-Datei.

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. Das Webformular zum Hochladen der Abgaben wird auch schon einige der gängigsten Fehler abfangen.

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."*
>
> *„Aus gegebenem Anlaß müssen wir darauf hinweisen, dass das Kopieren von Sourcecode aus ähnlichen Projekten aus dem Internet unter keinen Umständen gestattet ist und mit einem sofortigen Ausschluß geahndet wird. Wir besitzen sowohl die entsprechenden Werkzeuge, als auch die notwendige Sachkenntnis um solche 'Refactorings' zu erkennen.“*

4. Kaskade: Das Pflichtenheft

4.1. Die Spielregeln

Kaskade ist ein Spiel für 2 Personen (Spieler A und Spieler B), das auf einem rechteckigen Spielfeld gespielt wird. Die Anzahl der Felder kann in beide Richtungen variieren. Als Spielsteine werden farbige Kugeln gesetzt, pro Zug setzt der Spieler eine neue Kugel seiner Farbe auf das Spielfeld. Die Anzahl der verfügbaren Kugeln ist immer ausreichend, um das Spiel fortsetzen zu können.

Das Spiel verläuft nach folgenden Regeln:

Spieler A setzt die weißen Kugeln, Spieler B die schwarzen. Begonnen wird mit einem leeren Spielbrett. Spieler A beginnt und setzt eine weiße Kugel auf ein beliebiges Feld.

Der Spieler, der am Zug ist, hat prinzipiell zwei Möglichkeiten zum Setzen:

- Er kann seine nächste Kugel auf ein leeres Feld setzen.
- Er kann seine nächste Kugel auf ein Feld setzen, auf dem sich bereits Kugeln seiner Farbe befinden.

In Bild 1 ist Spieler B am Zug, und zwei mögliche Züge sind dargestellt.

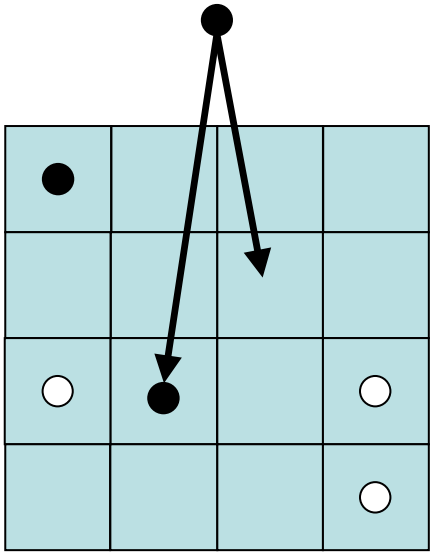


Bild 1

Die Felder können nur eine bestimmte Anzahl von Kugeln aufnehmen. Wenn ein Feld voll ist, bringt die nächste Kugel es zum Überlaufen. Ein Feld läuft über, wenn die Anzahl der Kugeln auf dem Feld der Anzahl der Nachbarfelder entspricht; dabei zählen diagonal benachbarte Felder nicht mit. Dies wird in Bild 2 verdeutlicht: Das Feld (3,3) ist voll, es enthält bereits drei schwarze Kugeln. Spieler B setzt nun eine weitere Kugel auf dieses Feld. Damit erhöht sich die Zahl der Kugeln auf diesem Feld auf 4 und es läuft über, da nun die Anzahl der

Kugeln der Anzahl der Nachbarfelder entspricht. Durch das Überlaufen werden die Kugeln auf die Nachbarfelder verteilt, so dass das Ausgangsfeld anschließend leer ist. Dafür haben alle Nachbarfelder dann eine Kugel mehr.

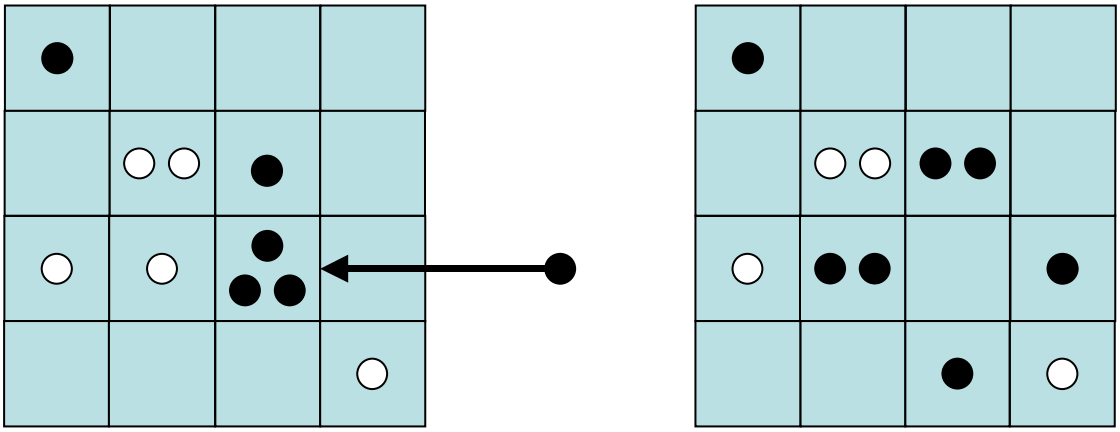


Bild 2

Durch das Überlaufen eines Feldes werden benachbarte Felder übernommen: Auf dem Feld (3,2) befand sich vor dem Zug eine weiße Kugel. Da Spieler B das Feld (3,3) zum Überlaufen gebracht hat, werden alle weißen Kugeln auf Nachbarfeldern schwarz und gehen damit in den Besitz von Spieler B über.

Als Nachbarfelder eines Feldes F gelten nur die Felder, die eine gemeinsame Seite mit F teilen. Damit hat ein Feld im Inneren des Spielfeldes vier Nachbarfelder, ein Feld am Rand hat drei Nachbarfelder, und die Ecken besitzen nur zwei Nachbarfelder. Ein Feld im Inneren ist also voll, wenn es mit 3 Kugeln besetzt ist, und es läuft über, wenn in einem Zug eine vierte Kugel auf dieses Feld gesetzt wird (Bild 3.a, blaue Kugeln). Ein Feld am Rande des Spielfeldes ist bereits bei 2 Kugeln voll (Bild 3.b, weiß), und ein Eckfeld bereits mit einer Kugel (Bild 3.c, gelb).

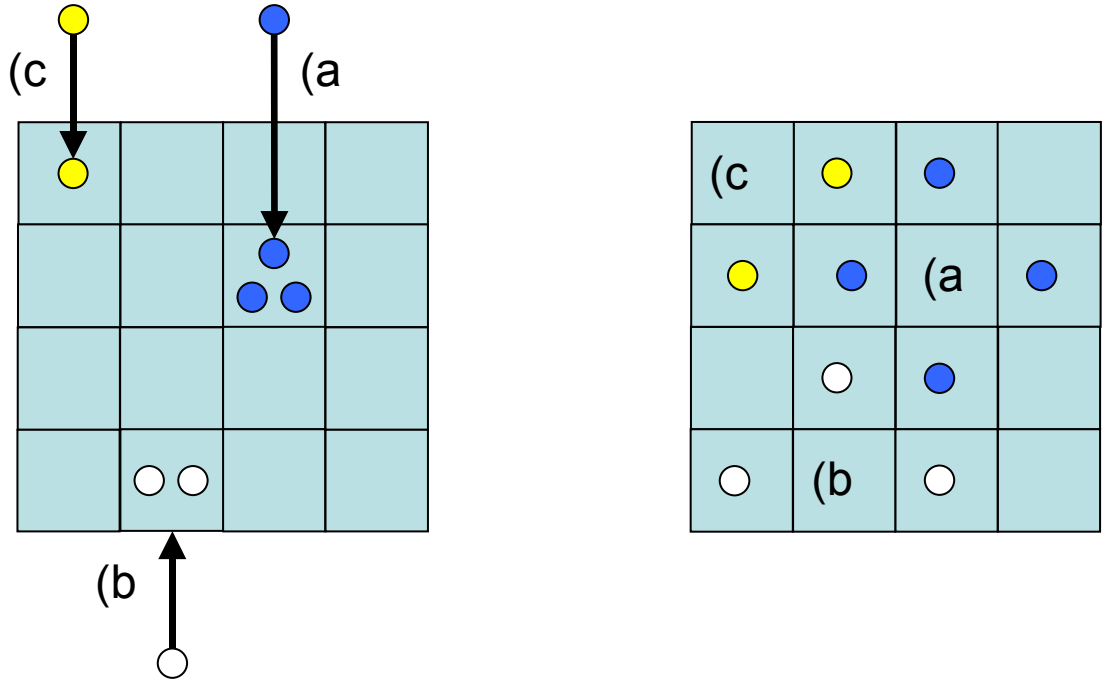


Bild 3

Falls auch eines der Nachbarfelder voll ist, wird es beim Verteilen der Kugeln ebenfalls zum Überlaufen gebracht. Ein Beispiel findet sich in Bild 4: Durch das Überlaufen des Feldes (2,3) wird auch eine Kugel in das volle Feld (3,3) übertragen, so dass dieses seinerseits überläuft und seine Kugeln an die Nachbarfelder abgibt.

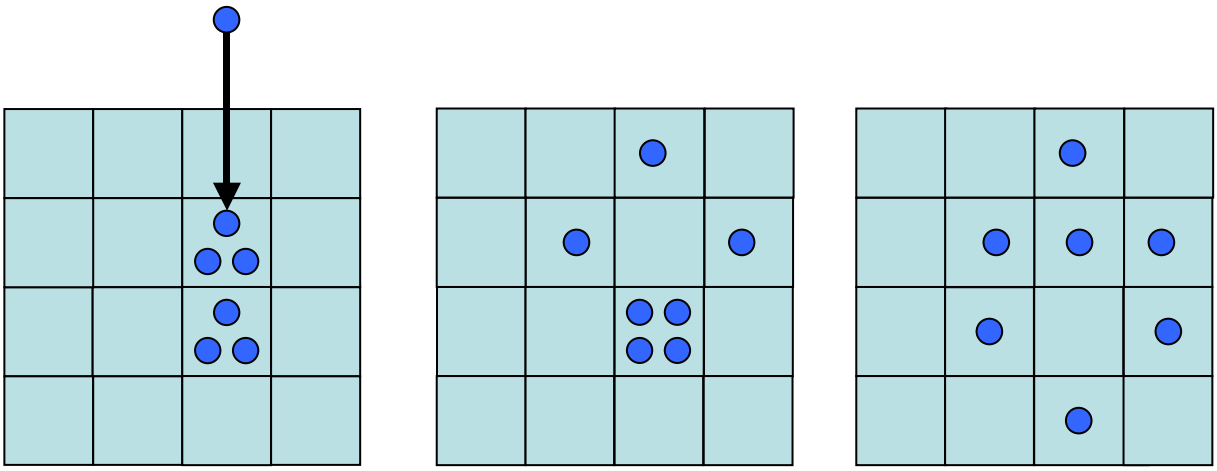


Bild 4

Derartige Ketten von Überläufen können sich kaskadierend über große Teile des Spielfeldes hinziehen. Dabei werden dann in der Regel auch viele gegnerische Felder übernommen.

Sieger des Spieles ist derjenige, der es schafft, alle gegnerischen Felder zu übernehmen. Das Spiel endet somit, wenn alle besetzten Felder einem Spieler gehören. Dies gilt natürlich nicht für den Spielbeginn: Spieler 1 hat durch Setzen seiner ersten Kugel das Spiel noch nicht gewonnen :-).

4.2 Anforderungskatalog

Die Hauptbestandteile der Implementierung unseres interaktiven Kaskade-Spiel sind

1. die Zuggeneratoren für die Computergegner
2. die graphische Oberfläche
3. die Netzwerkkomponente (siehe den eigenen Abschnitt „Netzwerkkommunikation“).

Sie sollen mindestens eine Strategie implementieren. Um die eigene Strategie direkt mit denen der anderen Teilnehmer vergleichen zu können, ist zusätzlich ein "Spielserver" zu implementieren.

4.3 Die graphische Oberfläche

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

1. Ein Menü mit folgenden Möglichkeiten:
    a. ein neues Spiel zu starten,
    b. das Spiel zu beenden,
2. Einen Dialog, mit dem der im Hintergrund laufende Server konfiguriert werden kann (für Details siehe den eigenen Abschnitt „Netzwerkkommunikation“):
    a. Maximale Zugzeit der Server-Strategie
    b. Portnummer des Servers
3. Einen Dialog, der beim Starten eines neuen Spieles geöffnet wird und zur Konfiguration des Spiels dient. Eingetragene Werte sollten beim nächsten Spiel des gleichen Programmlaufs wieder voreingestellt sein.

    Beim Starten eines neuen Spiels soll der Anwender die Möglichkeit haben, die Spieler festzulegen und zu entscheiden, wer anfängt.

    Folgende Paarungen müssen möglich sein:

    - Mensch gegen lokalen Computer X,
    - Mensch gegen Mensch,
    - lokaler Computer X gegen lokalen Computer X’, d.h. der Computer spielt gegen sich selbst,
    - lokaler Computer X gegen entfernten Computer Y, d.h. beide Computer spielen über eine zu programmierende Netzwerkkomponente gegeneinander
    - Mensch gegen entfernten Computer Y.

    Bei einem Spiel ohne menschliche Spieler soll das Spiel graphisch dargestellt werden und in einer angenehm zu verfolgenden Geschwindigkeit ablaufen.

    Zusätzlich soll die Möglichkeit bestehen, die Größe des Spielbretts auszuwählen. Bitte geben Sie dem Benutzer die Wahl zwischen folgenden Spielbrettgrößen:

    - 6x6
    - 8x8

- 10x10
- Zusätzlich soll eine frei wählbare Spielbrettgröße möglich sein, bei der die beiden Seitenlängen nicht gleich sein müssen. Minimum ist 3x3, Maximum ist 20x20. Ihre GUI muss diesen Bereich auch visualisieren können.

Falls einer der Spielgegner ein entfernter Computer ist, müssen zu seiner Konfiguration Eingabefelder für die folgenden Einstellungen zur Verfügung gestellt werden:

- Netzwerkadresse des entfernten Rechners,
- Portnummer des entfernten Rechners,
- Maximale Zugzeit der lokalen Strategie

4. Eine graphische Repräsentation des Spielbretts, über welche ein menschlicher Spieler seine Züge durch Klicken durchführen kann.
5. Darstellung der Überläufe in mindestens zwei Modi: Im Modus 1 soll das Ergebnis eines Zuges ohne Verzögerung sofort dargestellt werden, im Modus 2 sollen die einzelnen Kaskadeschritte sichtbar sein. Alle weiteren Animationsmöglichkeiten überlassen wir Ihnen, wir möchten an dieser Stelle nur darauf hinweisen, dass das Projekt sehr umfangreich ist und sie für den Anfang auch nur diese beiden Anzeigemöglichkeiten einbauen sollten.
6. Eine Anzeige, die die Namen der beiden Spieler anzeigt (auch bei Computergegnern und Netzwerkspielern sind deren Namen anzuzeigen).
7. Eine Anzeige, die bei 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.

4.4 Weitere Anforderungen

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

- Benutzen Sie bitte nur Klassen aus der Standard-Bibliothek von Java 1.5.0 (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.
- An die Repräsentation des Spielbretts stellen wir folgende beiden Anforderungen, die unbedingt einzuhalten sind:
  - Das Spielfeld darf nicht aus einem Gitternetz von Komponenten bestehen, also bei einem 10x10 Spielfeld dürfen nicht 100 Buttons verwendet werden. Das Spielfeld sollte eine Komponente sein, in die Sie selber die Steine zeichnen.
  - Die Steine dürfen nicht aus Bitmaps (also aus java.awt.Image) bestehen, sondern sollen aus Java 2D Operationen erzeugt werden, die sich im Paket java.awt.geom befinden.
- Ihr Programm darf keine speziellen betriebssystemspezifischen Anforderungen machen (z.B. Makefiles, .bat Dateien, Bash Scripte). Unsere Testumgebung besteht aus heterogenen Arbeitsplätzen, welche die unterschiedlichsten Betriebssysteme verwenden.

Wir erwarten, dass mit einer maximalen Zugdauer von fünf Sekunden auf einem durchschnittlichen PC (mit 1 GHz CPU Takt) bei einer Spielfeldgröße von 10x10 ein für einen menschlichen Gegner herausforderndes Spiel durchgeführt werden kann.

4.5 Empfohlene Vorgehensweise

Die folgende Liste zeigt Ihnen exemplarisch, wie Sie bei der Umsetzung der Anforderung vorgehen sollten. Dadurch wird verhindert, dass Sie sich gleich mit der ganzen Komplexität der Aufgabe befassen müssen. Stattdessen können Sie sich der Aufgabe schrittweise nähern.

1. Erstellen Sie ein Grobkonzept. Überlegen Sie, wie Sie vorgehen wollen. Schauen Sie in Ihren Terminkalender und erstellen Sie eine Zeitplanung.
2. Implementieren Sie eine GUI, mit der Sie die Steine auf dem Spielfeld platzieren können. Die Befolgung der Regeln und die Berechnung der Überläufe lassen Sie erst mal weg.
3. Implementieren Sie die Clientkomponente des Netzwerkprotokolls. Spielen Sie ruhig mal mit Telnet gegen unseren Testserver um ein Gefühl für die Abläufe der Kommandos zu bekommen.
4. Spielen Sie gegen unseren Testserver. Lassen Sie die Aktionen, die Sie im Punkt 1. auswählen können, mit der Netzwerkkomponente aus Punkt 2. an unseren Netzwerkserver übermitteln. Werten Sie die Antwort des Servers aus und stellen die übermittelte Spielfeldsituation bei sich dar. Sie können jetzt schon ein Spiel Mensch gegen entfernten Computer spielen.
5. Bauen Sie in Ihr Programm die Regeln ein. Lassen Sie keine illegalen Züge mehr zu. Berechnen Sie selbst die Überläufe und vergleichen Ihre Spielfeldsituation mit der des Servers.
6. Entwickeln Sie eine eigene (rudimentäre) lokale Strategie.
7. Erweitern Sie die eigene Netzwerkkomponente um die geforderte Serverfunktionalität.
8. Bauen Sie weitere GUI Funktionalitäten ein
9. Verbessern Sie Ihre Computerstrategie.

5 Kaskade-Strategie

Teil der Aufgabe ist, einen ernstzunehmenden Computergegner zu entwickeln. 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. Da das Spiel recht schwer auf einem Brett zu spielen ist (man muss recht viele Steine als Folge von Überläufen umsetzen), bietet es sich an, gemäß der oben beschriebenen Implementierungs-Strategie vorzugehen und zunächst im Spiel mit dem System und dem Testsystem auf unserem Server Erfahrungen zu sammeln. Dann können Sie verfolgen, wie die einzelnen Züge und Strategien wirken. Sie können natürlich durch einige Hilfsmittel auch den „menschlichen Lerner“ unterstützen, wie etwa die Rücknahme von Zügen oder die Möglichkeit, Züge versuchsweise zu setzen, die Ihnen ermöglichen, das Resultat eines Zuges mit allen Überläufen zu sehen, bevor dieser Zug tatsächlich bestätigt wird und damit der Gegner an der Reihe ist. Auf dieser Basis sollte es ihnen dann gelingen, die Strategie des Computerspielers zu implementieren. Im Folgenden geben wir einige Tipps zum Spielverlauf, um Ihnen die Einarbeitung zu erleichtern:

Das Ziel besteht darin, die Überläufe so zu steuern, dass die gegnerischen Felder übernommen werden. Daher kommt den vollen Feldern eine besondere Bedeutung zu: Durch Ablage einer zusätzlichen Kugel wird das Feld zum Überlaufen gebracht und kann damit die Nachbarfelder übernehmen. Im Prinzip stärkt ein volles Feld also die eigene Position (bzw., falls ein gegnerisches Feld voll ist, die Position des Gegners). Aber Vorsicht: Falls der Gegner die Möglichkeit erhält, ein eigenes volles Feld zu übernehmen, bringt er es mit der Übernahme auch gleich zum Überlaufen. Damit gehen dann möglicherweise auch weitere eigene Felder verloren.

Schauen wir uns einige Spielsituationen an:

Beim Start versuchen die Gegner üblicherweise, die Ecken zu besetzen. Eine Ecke hat den Vorteil, dass sie bereits voll ist, wenn sie nur mit einer Kugel belegt ist. Auch die Randfelder sind attraktiv, da sie bereits bei zwei Kugeln voll sind und bei der dritten Kugel überlaufen. Häufig sieht man eine ganze Reihe von vollen Feldern, die durch eine einzige weitere Kugel „gezündet“ werden können (siehe Bild 5).

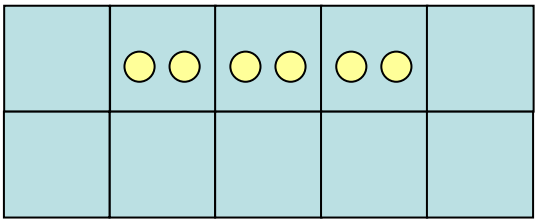


Bild 5

Gefährlich wird es dann, wenn man seine eigenen Spielsteine neben einem vollen Feld des Gegners ablegt. Der Gegner kann im nächsten Zug sein Feld zum Überlaufen bringen und damit die Steine des eigenen Feldes übernehmen. Dies wird sicherlich öfter passieren, schlimm kann es jedoch sein, wenn ein eigenes volles Feld neben einem vollen Feld des Gegners liegt: Der Gegner kann dann im nächsten Zug sein Feld überlaufen lassen, damit unser Feld übernehmen und zum Überlaufen bringen und damit möglicherweise eine ganze von uns aufgebaute Kette von vollen Feldern übernehmen. Ein solches Beispiel sehen wir im Bild 6: Falls Gelb auf das Feld (2,4) eine weitere Kugel legt und es damit auffüllt, setzt er sich der Gefahr aus, dass sein Gegner beim nächsten Zug durch das Überlaufen des Feldes (3,4) die ganze Kette übernimmt. Daher muss genau überlegt werden, wie man sich den Feldern des Gegners nähert und sich selbst Chancen aufbaut, um gegnerische Felder zu übernehmen.

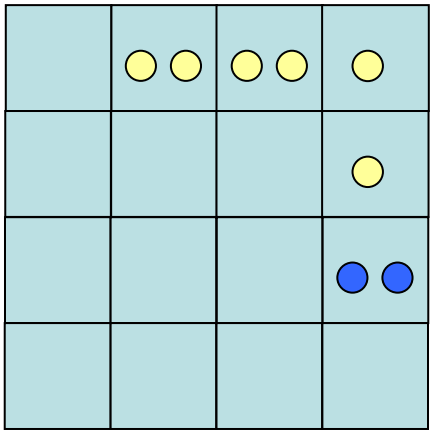


Bild 6

Zu diesem Zweck kann man Fallen aufbauen, über die man gefährliche gegnerische Kugeln übernehmen kann, ohne sich selbst dem Risiko auszusetzen, vom Gegner übernommen zu werden.

Bild 7 zeigt eine solche Falle, die Schwarz für Weiß gelegt hat. Eigentlich sieht die Situation für Weiß auf dem Feld (1,3) nicht gefährlich aus: Das Nachbarfeld ist nur mit einer schwarzen Kugel belegt, so dass sie scheinbar nicht überlaufen kann. Auf der anderen Seite ist schwarz durch weiß auch nicht gefährdet, da sich in der Nachbarschaft von (1,3) kein schwarzes volles Feld befindet, was eine schwarze Kette an den Gegner ausliefern könnte.

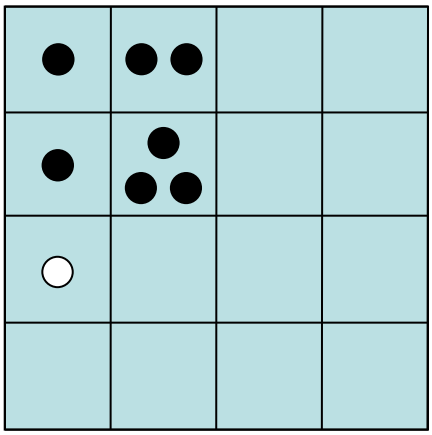
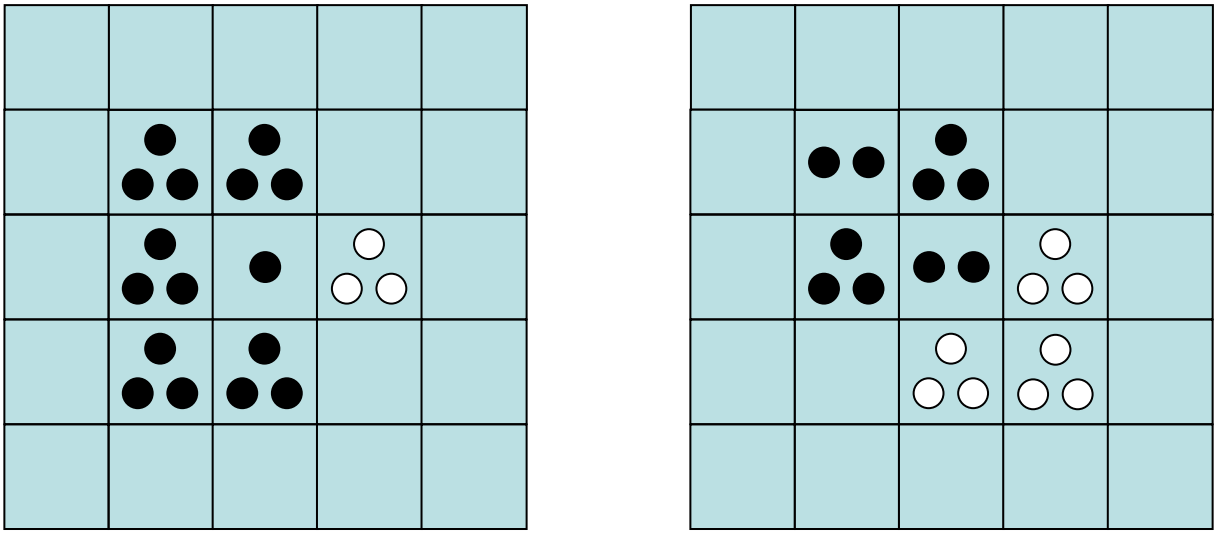


Bild 7

Bei genauerem Hinsehen erkennt man jedoch, dass die Situation für Weiß ziemlich gefährlich ist: Die drei schwarzen Felder (1,1), (1,2) und (2,2) sind alle voll und können damit im nächsten Zug zum Überlauf gebracht werden. Das bewirkt, dass Feld (2,1) eine Kugel von (1,1) und eine Kugel von (2,2) bekommt. Damit läuft es über, also wird das Feld (3,2) von Schwarz übernommen. Und Schwarz ist in der komfortablen Situation, dass Weiß diese Kette nicht zünden kann, da ein Überlaufen von (3,1) nur zur Übernahme des Feldes (2,1) führt.

Weitere solche Fallen sind in Bild 8 dargestellt. Die Idee ist immer die gleiche: Das Angriffsfeld ist selbst nicht voll, aber die Nachbarfelder sind so „zusammengeschaltet“, dass durch einen Überlaufvorgang die fehlenden Kugeln in das Angriffsfeld gebracht werden. Dabei kann es vorkommen, dass beide Gegner sich um die Eroberung eines Feldes bemühen, wie in Bild 8 b.



(a) (b)

Bild 8

Bei der Analyse einer Situation ist also der Füllungsgrad eines Feldes zu berücksichtigen. Es kann aber auch vorteilhaft sein, derartige Fallen mit in die Analyse einzubeziehen, um einen genaueren Überblick über tatsächliche Spielsituation zu erlangen. Und vielleicht fallen Ihnen weitere Elemente einer Analyse ein.

Implementierung der Spielstrategie

Spiele wie Kaskade lassen sich 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 (eigene und folgender gegnerische Züge),
- Traversierung des Spielbaums,
- Bewertung der Stellungen.

In der Spielbaumanalyse wird der Baum üblicherweise nur bis zu einer bestimmten Tiefe, hier die Denktiefe genannt, betrachtet. In den Knoten des Spielbaumes wird ein numerischer Wert für die Güte der Stellung verwaltet. Für Sie wird in der Implementierung das Hauptgewicht auf die Berechnung dieser Werte, also auf die Stellungsbewertung, fallen. Dazu müssen Sie eine Methode entwickeln, 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 einen Spielbaum 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.

- 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.

Im folgenden skizzieren wir eine einfache Strategie. **Wir erwarten, dass Sie zumindest diese Strategie implementieren.** Darüber hinaus überlassen wir es Ihrer Phantasie, Verbesserungen oder ganz andere Strategien zu entwickeln und zu realisieren. Wir gehen davon aus, dass Sie eine weitere Strategie in Ihrem System implementieren.

Bewertung einer Spielsituation:

- Jedes Spielfeld des Spielbretts bekommt eine Bewertung - die Gesamtbewertung des Bretts ergibt sich aus der Summe der Feldbewertungen.
- Positive Punkte werden vergeben für Felder, auf denen sich eigene Steine befinden und die nicht vom Gegner bedroht werden, d.h., keines der vom Gegner besetzten Nachbarfelder ist voll und droht mit Überlauf. Pro Stein auf einem solchen Feld werden n Punkte gegeben.
- Positive Punkte werden auch für die Felder vergeben, die dem Gegner gehören, die jedoch von einem eigenen Feld bedroht werden. Jeder Stein auf einem solchen Feld erhält ebenfalls n Punkte, allerdings mit einem Faktor x gewichtet. Damit erhält ein solcher Stein x*n Punkte.
- Pro Stein auf einem gegnerischen Feld, das nicht von einem eigenen Feld bedroht wird, werden n negative Punkte vergeben.
- Jeder durch ein gegnerisches volles Feld bedrohte eigene Stein wird ebenso negativ gezählt, jedoch mit dem Faktor y gewichtet. Ein solcher Stein erhält also -y*n Punkte.

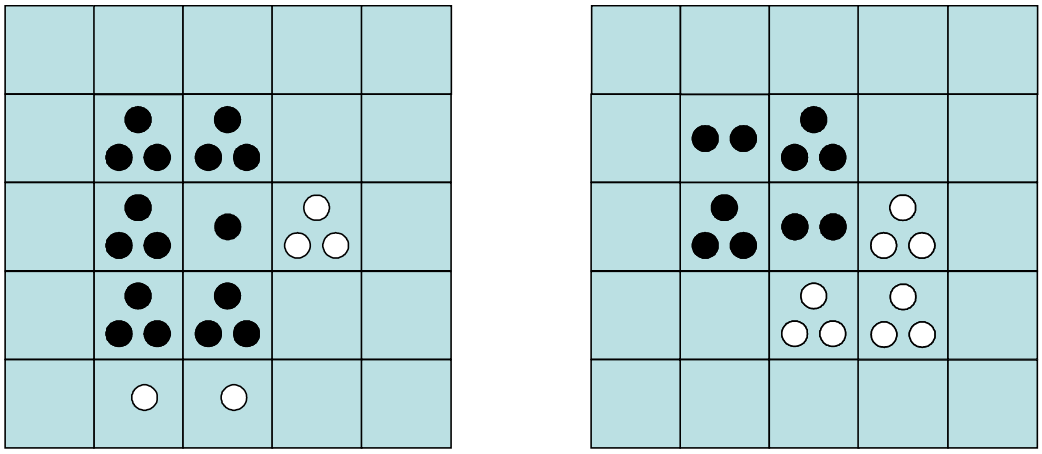


Bild 9

Beispiel: Betrachten wir die Situationen aus Bild 9 und bewerten die Stellungen aus der Sicht von Weiß. Wir nehmen an, dass n = 1, x = 0,5 und y = 0,2 ist. In der Situation a werden die folgenden Punkte vergeben:

(2,2)	-3,0
(2,3)	-3,0
(3,2)	-3,0
(3,3)	0,5
(3,4)	3,0
(4,2)	-3,0
(4,3)	-3,0
(5,2)	-0,2
(5,3)	-0,2

	-8,9

In der Situation (b) erhalten wir die folgende Punktevergabe:

(2,2)	-2,0
(2,3)	-3,0
(3,2)	-3,0
(3,3)	1,0
(3,4)	3,0
(4,3)	3,0
(4,4)	3,0

	2,0

Ausgehend von der aktuellen Spielsituation werden nun alle möglichen Züge simuliert und schließlich derjenige ausgewählt, der die Situation mit den meisten Punkten zum Ergebnis hat (Denktiefe 1). Man kann nun einen Schritt weiter gehen und auch die möglichen Reaktionen des Gegners bewerten (Denktiefe 2): Von jeder möglichen Situation, die durch einen eigenen Zug entstehen kann, wird ermittelt, was der bestmögliche Zug des Gegners ist. Schließlich wird derjenige Zug ausgewählt, dessen bester Zug des Gegners die geringste Punktzahl aufweist. Dies kann man natürlich fortsetzen und so zu tieferen Denktiefen kommen.

Wenn Sie diese Strategie realisiert haben, können Sie zum einen mit den verschiedenen Parametern experimentieren, um zu sehen, ob dadurch eine Steigerung der Spielstärke

erreicht wird. Darüber hinaus können Sie Optimierungen einbringen, die zu einer Beschleunigung des Spiels führen. Dies ist vor allem dann angebracht, wenn die Laufzeit Ihres Programms kein angenehmes Spiel Mensch gegen Computer zulässt. Hier einige Beispiele:

- Sie sollten Gewinn-Situationen erkennen können – in dem Fall braucht keine weitere Suche stattzufinden, Sie müssen diesen Zug durchführen.
- Zur Optimierung der Performance können Sie ein Caching der Feldbewertungen einbauen. Dann brauchen Sie bei der Simulation der Züge nur diejenigen Felder neu berechnen, die sich durch den Zug und die daraus folgenden Kaskaden verändert haben, einschließlich der Nachbarn (da diese in die Bewertungsfunktion mit einfließen). Dies kann vor allem zu Beginn des Spiels, wo sich wenige Felder durch einen Zug ändern, einen Performanceschub bringen.
- Sie können versuchen, die Traversierung von Zweigen zu vermeiden, die mit Sicherheit nicht zu einem besseren Zug führen als der bisher beste Zug. Beispiel: Sie analysieren die Züge mit einer Denktiefe 2. Angenommen, für Ihren ersten möglichen Zug ermitteln Sie bei der Betrachtung aller möglichen Antwortzüge des Gegners eine Bewertung von 10. Dies ist der minimale Wert aller Stellenbewertungen nach den möglichen gegnerischen Zügen, aus Sicht Ihres Gegners also der beste Antwortzug. Bei der Betrachtung Ihres 2. Zuges können Sie nun aufhören, sobald ein Antwortzug des Gegners einen geringeren Wert ergibt: Damit ist klar, dass dieser Zug Ihren Gegner zu einer besseren Spielposition führt, und die weiteren möglichen Gegner-Züge brauchen nicht mehr betrachtet werden. Sie werden den 1. Zug auf jeden Fall diesem Zug vorziehen.
- Sie können auch bereits auf den Zwischenstufen Bewertungen durchführen, um die Erfolg versprechenden Zweige zu ermitteln. Im weiteren Verfahren werden dann nur diese Zweige verfolgt.

Voraussetzung für die Stellungsbewertung ist natürlich, dass Sie die Situation nach einem Zug korrekt darstellen, d.h. dass alle Überläufe korrekt durchgeführt werden. Darüber hinaus ist es wichtig, dass Sie in der Lage sind, Züge in einem bestimmten Zeitlimit durchzuführen. Sie müssen also evtl. abhängig von der Zeit die Bewertung abbrechen können, um den bis dahin besten Zug durchzuführen. Somit kann es von entscheidender Bedeutung sein, zuerst die Zweige im Spielbaum zu untersuchen, die ein gutes Ergebnis erwarten lassen, während andere Züge bei genügender Zeit später noch untersucht werden können.

Noch ein Hinweis: In Gewinnsituation kann es vorkommen, dass die Überläufe sich unendlich über das Spielfeld hinziehen. Daher ist es wichtig, sowohl beim Setzen und ermitteln der aktuellen Situation nach dem Zug als auch bei der virtuellen Zugumsetzung für die Stellungsbewertung, dass Sie Gewinnsituationen erkennen (d.h. die Situation, das ein Spieler alle Steine verloren hat) und die Überlaufbehandlung abbrechen.

6 Der Spielserver

6.1 Netzwerkkommunikation

Eine weitere Anforderung an "Kaskade" ist, das Spiel über das Netzwerk gegen einen anderen (Computer-) Gegner spielen zu können. Zu diesem Zweck soll das Programm, wenn es gestartet wird, automatisch einen Spiel-Server auf dem Default-Port 4711 bereitstellen, die maximale Dauer für die Berechnung eines Spielzuges ist auf fünf Sekunden zu beschränken. Weitere Optionen (alternative Strategie für den Spiel-Server, Parameter/Zugzeitlimit für die Strategie, alternativer Port) sollen einstellbar sein, doch es muss auch ohne Parameterangabe ein Spiel-Server mit Default-Einstellungen laufen - dies wird dann auch die Bewertungsgrundlage für die Abgabe sein, daher sollte der Server in den Defaulteinstellung möglichst stark sein (z.B. die stärkste Strategie ausgewählt, falls es mehrere zur Auswahl gibt). Die Kommunikation soll über TCP-Sockets implementiert werden. Das im Folgenden vorgegebene Netzwerk-Protokoll basiert auf einer einfachen Klartext-Kommunikation, um das Spiel im Extremfall auch über Telnet spielen zu können.

Bitte sehen Sie eine Möglichkeit vor, um die Portadresse zur Laufzeit ändern zu können (siehe Anforderungen an die graphische Oberfläche). 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).

Es ist erlaubt, für den lokalen Computergegner den eigenen Netzwerkserver zu benutzen (und sich damit redundante Arbeit zu ersparen). Dazu muss der Server dann allerdings in der Lage sein, mindestens zwei Verbindungen gleichzeitig zu verarbeiten (Situation Computer lokal gegen Computer lokal).

An der Fernuniversität werden wir zwei Kaskadeserver mit unterschiedlichen Spielstärken installieren, die rund um die Uhr erreichbar sind. Nähere aktuelle Informationen entnehmen Sie bitte der Praktikumsseite im Netz.

Grundlegende Konventionen:

1. Nicht vom Client bzw. Server auszuwertende Zeilen sind mit dem Präfix ">" auszuzeichnen - dies wird z.B. für die Antwort auf das `help`-Kommando benutzt.
2. Jede Zeile muss mit CRLF beendet werden - in Java wird dies mit dem String "`\r\n`" realisiert.
3. Die allgemeine Kommandostruktur ist "`Kommando(Parametername1=Wert1;Parametername2=Wert2;...)`", wobei bei Kommandos ohne Parameter die Klammer weggelassen wird. Die Reihenfolge der Parameter ist irrelevant, diese werden über ihren Namen eindeutig identifiziert. Der Kommando-Interpreter muss bezüglich überflüssiger Leerzeichen vor oder nach den Schlüsselzeichen (Klammern, Semikolon, Gleichheitszeichen) robust sein, also z.B. auch "`move (x =3; y= 3 )`" akzeptieren. Optionale Parameter sollen, wenn Sie nicht verwendet werden, komplett weggelassen werden.
4. Alle Antworten des Servers inkl. Fehler- und Statusmeldungen folgen der gleichen Syntax wie die Kommandos.

6.2 Kommandos

Fehlermeldungen

Fehlermeldungen werden auch in Kommando-Syntax zurückgeliefert:

Kommando	*Parameter*	*Bemerkung*
`Error`		Fehlermeldung
	`message`	Text der Fehlermeldung

Beispiel:

Client-Anfrage	`test`
Server-Antwort	`error(message=unknown command 'test')`

Die Fehlertexte haben keine Semantik - sie sollten jedoch "sprechend" sein, damit sie z.B. in einer Dialogbox angezeigt werden können. Falls Client und Server das Protokoll richtig implementieren und die Kaskade-Regeln korrekt beachten, dürfen keine Fehler auftreten! Daher sollte es eigentlich nur zu Fehlersituationen kommen, falls sich ein Mensch in einer Telnet-Sitzung mit dem Spiel-Server befindet - dementsprechend sollte der Server nach einem Fehler so reagieren, dass er erwartet, dass das Kommando vom Client (dann hoffentlich ohne Fehler) wiederholt wird.

`help`

Anfrage:

Kommando	*Parameter*	*Bemerkung*
`Help`		Hilfefunktion (für die Telnet-Bedienung)

Antwort:

Hilfetexte zu den möglichen Kommandos. Da diese nur für den Menschen interessant sind und nicht vom Programm interpretiert werden müssen, sollte es sich hierbei um einfache mit dem Kommentarzeichen ">" vorangestellte Freitexte handeln.

Beispiel:

Client-Anfrage	`help`
Server-Antwort	`> Diese Hilfe ist` `> verbesserungswuerdig!`

`init`

Anfrage:

Kommando	*Parameter*	*Bemerkung*
`Init`		Initialisierung eines neuen Spielfeldes
	`xDim`	Ausdehnung des Spielfelds in horizontaler Richtung, mindestens 3, maximal 20.
	`yDim`	Ausdehnung des Spielfelds in vertikaler Richtung, mindestens 3, maximal 20.
	`begin`	bestimmt, wer den ersten Zug hat: möglich ist "`c`" (Client) oder "`s`" (Server)
	`[setup]`	Anfänglicher Spielfeldaufbau als eine Kette von Ziffer/Buchstaben-Paaren (jeweils ein Paar pro Feld), beginnend oben links (x=0;y=0), dann zeilenweise fortsetzend bis unten rechts. Die Ziffern bestimmen dabei die Anzahl der Steine auf dem Spielfeld und das Zeichen legt den Besitzer fest - möglich ist hier "`c`" (Client), "`s`" (Server) oder "`n`" (neutral, kein Besitzer). Dieser Parameter ist optional, die Vorgabe ist ein leeres Spielfeld. Siehe auch „illegales Spielfeld-Setup“ bei den möglichen Fehlermeldungen.
	`User`	Identifizierung des Clients durch einen Namen.

Antwort:

Kommando	*Parameter*	*Bemerkung*
`initialized`		Quittung, dass das Spielfeld initialisiert wurde.
	`user`	Identifizierung des Servers durch einen Namen.

Darauf folgt jeweils für jede Spielfeld-Zeile, beginnend von oben (y=0):

Kommando	*Parameter*	*Bemerkung*
`board`		Es wird eine Zeile der aktuellen Spielfeldsituation zurückgegeben. Die `board`-Rückmeldungen sollten vom Client dazu benutzt werden, um eventuelle Inkonsistenzen (Schummeleien?) in der Spielfeldsicht zwischen Client und Server aufzudecken und dann das Spiel abzubrechen
	`row`	Kette von Ziffern/Buchstaben-Paaren, die eine horizontale Zeile des Spielfelds beschreibt (analog dem

		`setup`-Parameter)

Falls der Server den ersten Zug hat ("`begin=s`"), folgen noch zwei weitere Rückmeldungen – der Zug des Servers sowie die Situation nach dem Zug:

Kommando	*Parameter*	*Bemerkung*
`move`		bezeichnet einen Spielzug (in diesem Zusammenhang vom Server)
	`x`	horizontale(X)-Koordinate des Spielfeldes, welches für den Zug ausgewählt wurde (`x=0` bedeutet linker Rand bzw. `x=xDim-1` rechter Rand)
	`y`	vertikale(Y)-Koordinate (`y=0` bedeutet oberer Rand bzw. `y=yDim-1` unterer Rand)

Darauf folgt, sofern dass Spiel nach dem Zug noch nicht beendet ist[1], jeweils für jede Spielfeld-Zeile wieder jeweils eine `board`-Zeile.

Falls der Server einen Zug gemacht hat, und das Spiel nach diesem Zug beendet ist, antwortet der Server mit folgendem Kommando:

Kommando	*Parameter*	*Bemerkung*
`bye`		beendet das laufende Spiel und die Netzwerkverbindung
	`winner`	bestimmt, wer gewonnen hat: möglich ist hier nur "`s`" (Server)

Beispiel:

Client-Anfrage	`init(xDim=3;yDim=3;begin=s;setup=0n0n0n2s1c0n0n0n1c; user=Wilfried)`

Server-Antwort	`initialized(user=Mr. Unschlagbar)`
	`board(row=0n0n0n)`
	`board(row=2s1c0n)`
	`board(row=0n0n1c)`
	`move(x=0;y=1)`
	`board(row=1s0n0n)`
	`board(row=0n2s0n)`
	`board(row=1s0n1c)`

[1] Da eine Gewinnsituation nicht immer in einem statischen Endzustand endet, sondern sich durchaus eine unendliche Überlaufreaktion einstellen kann, ist der Spielfeldzustand in solch einem Fall nicht definiert und wird daher nicht zurückgeliefert – auch nicht, falls die Überlaufreaktion enden würde.

Mögliche Fehlermeldungen:

- Spielfeld wurde bereits zuvor initialisiert:
  Ein `init` ist während einer Sitzung nur einmal erlaubt.
- Spielfeld wurde mit ungültigen Dimensionen initialisiert.
  xDim oder yDim liegen außerhalb des erlaubten Bereichs.
- Illegales Spielfeld-Setup:
  Das übergebene Spielfeld stellt bereits eine Gewinnsituation dar.
  Um dies zu vermeiden muss, sofern sich mindestens zwei Spielsteine auf dem Spielfeld befinden, jede Partei mindestens einen Spielstein besitzen. D.h. „einfarbige" Setups mit mehr als einem Spielstein sind illegal.
- Sonstige Illegale Parameter

move

Anfrage:

Kommando	*Parameter*	*Bemerkung*
`move`		bezeichnet einen Spielzug (in diesem Zusammenhang vom Client)
	`x`	horizontale(X)-Koordinate des Spielfeldes, welches für den Zug ausgewählt wurde (`x=0` bedeutet linker Rand bzw. `x=xDim-1` rechter Rand)
	`y`	vertikale(Y)-Koordinate (`y=0` bedeutet oberer Rand bzw. `y=yDim-1` unterer Rand)

Antwort:

Kommando	*Parameter*	*Bemerkung*
`move`		bezeichnet einen Spielzug (in diesem Zusammenhang vom Server)
	`x`	horizontale(X)-Koordinate des Spielfeldes, welches für den Zug ausgewählt wurde (`x=0` bedeutet linker Rand bzw. `x=xDim-1` rechter Rand)
	`y`	vertikale(Y)-Koordinate (`y=0` bedeutet oberer Rand bzw. `y=yDim-1` unterer Rand)

Darauf folgt, sofern dass Spiel nach dem Zug noch nicht beendet ist, jeweils für jede Spielfeld-Zeile wieder jeweils eine `board`-Zeile.

Falls das Spiel nach dem Zug des Client oder des Servers beendet wird, antwortet der Server mit folgendem Kommando[2]:

Kommando	*Parameter*	*Bemerkung*
`bye`		beendet das laufende Spiel und die Netzwerkverbindung
	`winner`	bestimmt, wer gewonnen hat: möglich ist "`c`" (Client) oder "`s`" (Server)

Beispiel:

Dargestellt ist die letzte Board-Rückmeldung vom Server, der Client ist am Zug, der Server antwortet mit seinem Zug und gewinnt:

(...)	`(...)`

Server-Antwort	`board(row=1s0n0n)`
	`board(row=2s2s0n)`
	`board(row=1s2c0n)`

Client-Anfrage	`move(x=2;y=2)`
Server-Antwort	`move(x=0;y=2)` `bye(winner=s)`

Gleiche Situation, diesmal gewinnt der Client mit seinem Zug:

(...)	`(...)`

Server-Antwort	`board(row=1s0n0n)`
	`board(row=2s2s0n)`
	`board(row=1s2c0n)`

Client-Anfrage	`move(x=1;y=2)`
Server-Antwort	`bye(winner=c)`

Mögliche Fehlermeldungen:

- falls das Spielfeld noch nicht initialisiert wurde
- falls der Client einen illegalen Zug tätigt
- illegale Parameter
- …

[2] Also auch wenn der Client weiß, dass er gewinnt, muss er den Gewinnzug zum Server schicken und seine Antwort abwarten. In dem Fall eines Gewinnszugs des Clients entfällt natürlich auch die `move`-Antwort des Servers und dieser antwortet direkt mit „`bye(winner=c)`“.

`exit`

Anfrage:

Kommando	*Parameter*	*Bemerkung*
`exit`		Anfrage zum Verlassen des aktuellen Spieles

Antwort:

Kommando	*Parameter*	*Bemerkung*
`bye`		beendet das Spiel und bricht die Netzwerkverbindung ab

6.3 Zusammenfassung

Alle Anfragen vom Client:

- `init(xDim;yDim;begin[;setup];user)`
- `move(x;y)`
- `exit`

Alle Antworten vom Server:

- `initialized(user)`
- `board(row)`
- `move(x;y)`
- `bye([winner])`
- `error(message)`

7 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.

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

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

Einführung in das Programm:

In der Einführung (max. 10 DIN-A4-Seiten) beschreiben Sie das generelle Vorgehen Ihres Programms. Skizzieren Sie seine Grundidee, die wichtigsten benutzten Datenstrukturen (inkl. UML-Klassendiagramm) und den grundlegenden Aufbau Ihres Programms (z.B. welche Klassen erledigen welche Aufgaben) sowie den Strategie-Algorithmus im Detail und begründen Sie Ihre Designentscheidungen. 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.

Folgende Gliederungspunkte erwarten wir:

1. Einleitung
2. Grundidee und Konzepte
   (dies beinhaltet auch die Klassen- und Packageaufteilung)
3. Designentscheidungen und Dialogstrukturierung
4. Bedienungsanleitung
5. Implementierte Strategie
6. UML-Klassendiagramm

Freie Tools zur Erstellung von UML-Klassendiagramme sind z.B.:

http://argouml.tigris.org/

weitere finden Sie beispielsweise bei http://sourceforge.net .

Kommentierung des Programmcodes:

Beachten Sie bei der Erstellung Ihres Programmcodes bitte die unter http://java.sun.com/docs/codeconv publizierten Code Conventions for the Java Programming Language.

Insbesondere achten Sie noch auf folgende Punkte:

1. 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.

2. Vermeiden Sie es, die Dokumentation als großen Java-Kommentar vor Ihr Programm zu "quetschen".
3. Überlegen Sie sich einen einheitlichen Kommentarkopf mit Javadoc-Elementen, den Sie vor allen Klassen, Attributen und Methoden 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 Programms. 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, erzeugen Sie auch Javadoc für die Elemente mit der Sichtbarkeit „package“ und „protected“, private Elemente müssen nicht mit Javadoc dokumentiert werden. Beschreiben Sie damit die Parameter und Rückgabewerte.
4. 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!
5. 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").
6. Vermeiden Sie es, mehr als eine Anweisung in eine Zeile zu schreiben.

8 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 Programms wird jedoch mit der steigenden Anzahl nicht erfolgreicher Testfälle (= korrekter Reaktionen des Programms) 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:

```
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 Hauptprogramms 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 Kaskade-Programms 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.