# Computer spielen "Fünf gewinnt"

Jugend forscht 2002
Mathematik/Informatik

Johannes Hauser  Alexander Heß  Martin Kleppmann

**Kurzfassung**

"Fünf gewinnt" ist ein einfaches Brettspiel, das "Tic-Tac-Toe" und "Vier gewinnt" ähnlich ist. Zwei Spieler setzen auf einem karierten Spielbrett abwechselnd Steine mit dem Ziel, fünf davon in eine Reihe zu bekommen. Gleichzeitig müssen sie verhindern, dass der Gegenspieler dieses Ziel erreicht.

Es gibt viele verschiedene Wege, um bei "Fünf gewinnt" zum Sieg zu kommen: Die Spieler können z.B. versuchen, sofort mit Druck Gewinnsituationen zu erreichen, oder zuerst ihre Steine in günstige Positionen bringen, um später zuzuschlagen.

Ziel unseres Projektes ist, eine Reihe von Algorithmen zu erstellen, die auf unterschiedliche Weise "Fünf gewinnt" spielen. Im direkten Vergleich wollen wir daraus Schlüsse über ihr Verhalten, ihre Möglichkeiten und Grenzen, aber auch über das Spiel an sich ziehen. Dazu haben wir die Software "Morphion" entwickelt, die es erlaubt, beliebige Algorithmen in Form von "Engines" gegeneinander (auch in Teams) spielen zu lassen. Außerdem kann der Benutzer selbst gegen den Computer spielen und sich dabei nach Wunsch von anderen Engines beraten lassen.

Abbildung: Der japanische Meister Takagi Rakuzan beim Renju-Spiel (Quelle: http://www.renju.nu)
Im Hintergrund: Ausschnitte des Quelltextes unserer Software

# Inhaltsverzeichnis

# 1 Einleitung

## 1.1 Über "Fünf gewinnt"

Das Spiel "Vier gewinnt" ist bekannt; "Fünf gewinnt“ hingegen hat im europäischen Raum nur wenige Anhänger. "Fünf gewinnt", ein sog. taktisch-topologisches Denkspiel (wie z.B. auch Tic Tac Toe und Mühle), wird auf einem karierten Brett gespielt. Die Spieler setzen abwechselnd Steine mit dem Ziel, fünf davon in eine nicht unterbrochene Reihe zu bekommen (waagerecht, senkrecht oder diagonal), was der Gegenspieler verhindern muss.

Über die Geschichte von "Fünf gewinnt" gibt es widersprüchliche Angaben. Die Ursprünge gehen vermutlich bis ins alte China zurück: Dort wurde vor etwa 4000 bis 5000 Jahren das heute unter dem japanischen Namen "Go" bekannte Spiel "Wei-qi" erfunden. "Go" wird mit schwarzen und weißen Steinen auf einem 19x19 Feldern großen Spielbrett gespielt. Später wurde begonnen, auf dem "Go"-Feld andere Spiele zu spielen, darunter "Gomoku narabe" (Fünf Steine in einer Reihe auf einem "Go"-Brett), das heute hauptsächlich unter dem Namen "Gobang" oder einfach "Gomoku" bekannt ist. Weitere Namen sind "Kakugo", "Itsutsu-Ishi" und "Morphion" (daher auch der Name unserer Software).

## 1.2 Theoretische Eigenschaften des Spiels

Der wesentliche mathematische Unterschied zu "Vier gewinnt" besteht darin, dass "Fünf gewinnt" echt zweidimensional ist. In "Vier gewinnt" ist keine wirkliche Zweidimensionalität gegeben, da Steine immer in einer Spalte nach unten fallen; zur Auswahl für den nächsten Zug stehen maximal sieben Spalten. "Fünf gewinnt" verlangt vom Spieler ein höheres Maß an Wachsamkeit und Kreativität, da er seine eigenen und die gegnerischen Zugmöglichkeiten nach vielen verschiedenen Gesichtspunkten bewerten kann.

Es gibt die Hypothese, dass der Spieler, der den ersten Zug macht, bei optimalen Zügen beider Spieler garantiert gewinnt. Sakata und Ikawa veröffentlichten 1981 einen 32 Seiten langen (angeblichen) Beweis hierfür, der anderen Quellen zufolge jedoch nicht vollständig ist: Er berücksichtigt im 2. Zug nur die acht umliegenden Felder des ersten Steins. Weitere gesicherte Aussagen zu dieser Hypothese gibt es unseres Wissens nach nicht.

Im Laufe der Zeit zeigte sich jedoch deutlich, dass der zuerst Setzende einen klaren Vorteil hatte. Aus diesem Grund beinhalten heutige Varianten des Spiels meist zusätzliche Regeln: Bei dem 1899 erfundenen "Renju", das heute als japanische Nationalsportart betrachtet wird, wird dem Anzugsvorteil entgegengewirkt, indem für den Spieler, der den ersten Zug ausführt, bestimmte Konstellationen verboten sind.

## 1.3 Fragestellung unserer Arbeit

Unsere Problemstellung war es, einige möglichst unterschiedliche Algorithmen zum Spielen von "Fünf gewinnt" zu entwickeln und zu untersuchen. Um unsere Arbeit überschaubar zu halten – und weil wir viele diese Regeln erst durch spätere Recherchen erfuhren – beschränken wir uns auf die klassische Form: Mindestens fünf Steine in einer Reihe führen zum Gewinn, weitere Regeln gibt es nicht. (Selbstverständlich dürfen keine Steine verschoben oder weggenommen werden.) Desweiteren benutzen wir ein praktisch unbeschränktes Spielbrett von 256x256 Feldern und beziehen Sonderstellungen am Spielfeldrand nicht mit ein.

Wie reagieren unterschiedlich angelegte Algorithmen, wenn sie mit verschiedenen Stellungen konfrontiert werden? Lassen sich bestimmte Verhaltensmuster ausmachen? Sind die Reaktionsmuster auf die Struktur zurückzuführen, und wenn ja, wie? Können Engines sinnvoll kooperieren, auch wenn sie gegensätzliche Ziele verfolgen, und sind mehrere kooperierende Engines besser als eine einzelne? Und welche Engines "vertragen" sich besser als andere?

Der Computer kann sehr viele Lösungsmöglichkeiten in relativ kurzer Zeit durchsuchen. Hat der Mensch noch eine Chance dagegen (vgl. das legendäre Schachspiel von Kasparov gegen Deep Blue) oder sind ihm

bei derartigen Aufgaben die Maschinen komplett überlegen? Worin liegen die Unterschiede in der Spielweise von Mensch und Rechner?

Weiterhin wollen wir versuchen, den oben erwähnten Beweis auf empirische Art und Weise zu bestätigen oder zu widerlegen. Ausgangspunkt hierfür wären Engines, die möglichst gute oder sogar optimale Züge finden, zum Beispiel durch eine hohe Suchtiefe.

# 2 Entwicklung

## 2.1 Software: Aufbau

Um verschiedene Algorithmen untersuchen und vergleichen zu können, benötigen wir einerseits die Implementierung einer Auswahl von Algorithmen, und andererseits eine automatisierte Möglichkeit, verschiedene Algorithmen gegeneinander oder gegen einen Menschen spielen zu lassen.

Wir haben uns entschieden, unsere Software namens "Morphion" nach dem Client-Server-Prinzip zu entwickeln: Die Algorithmen werden als Clients, genannt "Engines", implementiert. Ein Server verbindet die Engines, visualisiert die Spiele, überwacht die Einhaltung der Regeln und ermöglicht das Zusammenfassen mehrerer Engines zu einem Team.

Vergleichbare Software für diesen Zweck konnten wir nicht finden – zwar gibt es im Internet Engines mit grafischem Interface (beispielsweise von S.Gerlach, siehe Quellenverzeichnis); es gibt auch Programme zur Verwaltung von Renju-Partien, die jedoch keine Möglichkeit zum Einbinden von Engines besitzen. Für unsere Zwecke bieten diese Programme eine zu eingeschränkte Funktionalität.

## 2.2 Der Server

Um die gewünschten Funktionen in einem Programm zu vereinen, haben wir den Server in folgende Funktionsgruppen gegliedert:

- Modus "Mensch gegen Computer": Der User kann interaktiv gegen eine oder mehrere zusammengeschlossene Engines spielen, und sich dabei optional von beliebigen Engines beraten lassen (diese schlagen dann Züge und deren Bewertungen vor, der User entscheidet letztlich).

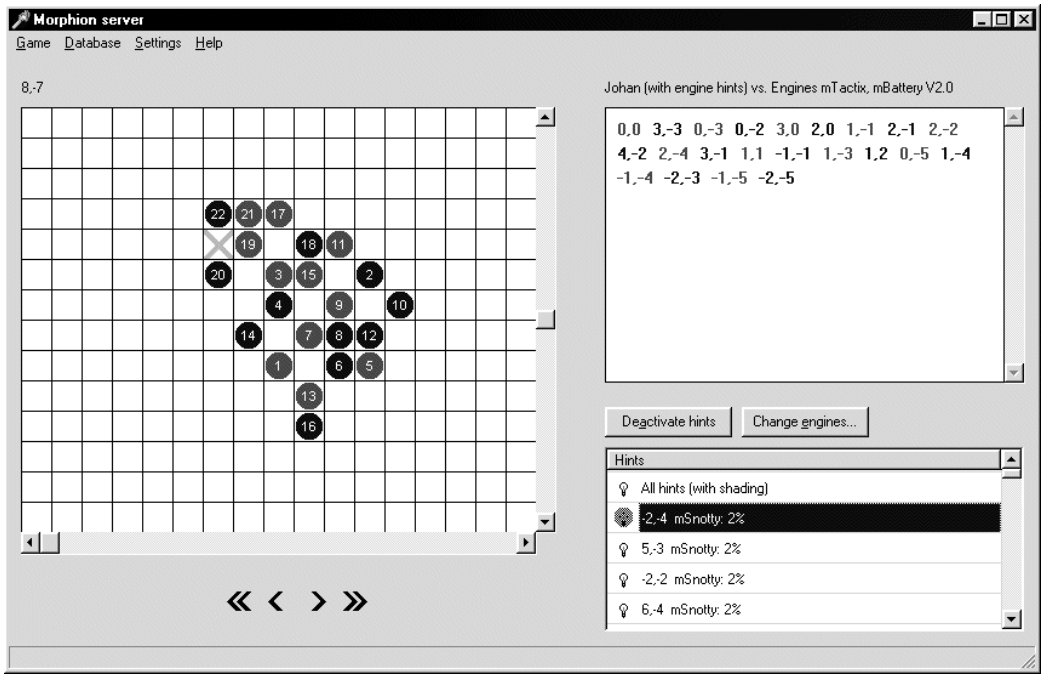
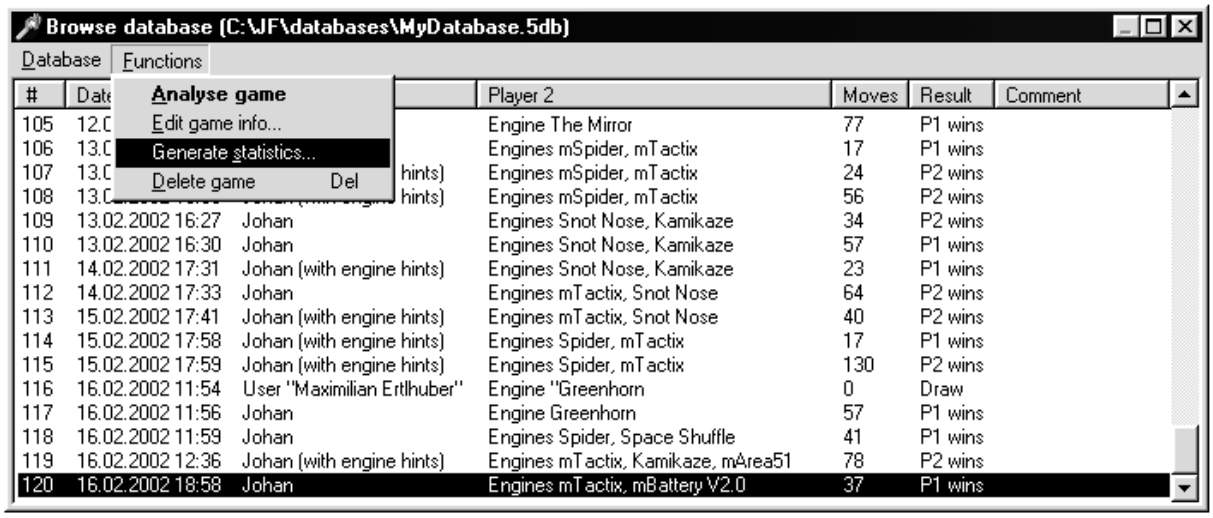



Abbildung 1: Screenshot des Servers im Modus "Mensch gegen Computer"

- Analysemodus: Beendete Partien können schrittweise betrachtet werden; zu jedem Zug können Kommentare eingefügt werden; Varianten können zusätzlich untersucht werden.

- "Engine Wars": In diesem Modus werden zunächst Engines bzw. Teams von Engines ausgewählt, die dann eine festgelegte Zahl von Partien unter einstellbaren Bedingungen automatisch austragen. Die Partien werden gespeichert und können nachträglich im Analysemodus untersucht werden. Zudem wird eine Statistik über das Verhalten der Engines errechnet.

- Datenbankfunktionalität: Eine eingebaute Datenbank ermöglicht die einfache Verwaltung von Partien, indem sie zusammenfassend in einer Tabelle dargestellt werden. Ein "Engine War" wird ebenfalls zur nachträglichen Analyse in eine solche Datenbank geschrieben.




| # | Date | | Player 2 | Moves | Result | Comment |
|---|---|---|---|---|---|---|
| 105 | 12.0 | | Engine The Mirror | 77 | P1 wins | |
| 106 | 13.0 | | Engines mSpider, mTactix | 17 | P1 wins | |
| 107 | 13.0 | hints) | Engines mSpider, mTactix | 24 | P2 wins | |
| 108 | 13.0 | hints) | Engines mSpider, mTactix | 56 | P2 wins | |
| 109 | 13.02.2002 16:27 | Johan | Engines Snot Nose, Kamikaze | 34 | P2 wins | |
| 110 | 13.02.2002 16:30 | Johan | Engines Snot Nose, Kamikaze | 57 | P1 wins | |
| 111 | 14.02.2002 17:31 | Johan (with engine hints) | Engines Snot Nose, Kamikaze | 23 | P1 wins | |
| 112 | 14.02.2002 17:33 | Johan | Engines mTactix, Snot Nose | 64 | P2 wins | |
| 113 | 15.02.2002 17:41 | Johan (with engine hints) | Engines mTactix, Snot Nose | 40 | P2 wins | |
| 114 | 15.02.2002 17:58 | Johan (with engine hints) | Engines Spider, mTactix | 17 | P1 wins | |
| 115 | 15.02.2002 17:59 | Johan (with engine hints) | Engines Spider, mTactix | 130 | P2 wins | |
| 116 | 16.02.2002 11:54 | User "Maximilian Ertlhuber" | Engine "Greenhorn | 0 | Draw | |
| 117 | 16.02.2002 11:56 | Johan | Engine Greenhorn | 57 | P1 wins | |
| 118 | 16.02.2002 11:59 | Johan | Engines Spider, Space Shuffle | 41 | P1 wins | |
| 119 | 16.02.2002 12:36 | Johan (with engine hints) | Engines mTactix, Kamikaze, mArea51 | 78 | P2 wins | |
| 120 | 16.02.2002 18:58 | Johan | Engines mTactix, mBattery V2.0 | 37 | P1 wins | |

Abbildung 2: Datenbankfunktionalität des Servers.

Weitere Funktionen, die nicht unmittelbar für die Erforschung der Engines notwendig sind, sind in Planung (z.B. Turniermodus). Siehe dazu auch: 5.3 Ausblick.

Der Server soll eine leicht bedienbare grafische Oberfläche (GUI; siehe Screenshot) besitzen, damit unser Projekt auch für Dritte leichter zugänglich wird. Die Handhabung haben wir an das weit verbreitete Schachprogramm "Fritz" angelehnt, um den Einstieg für Benutzer von "Fritz" zu erleichtern. Die Einteilung ist auch für "Morphion" sinnvoll: Auf der linken Seite ist das Spielbrett zu sehen; die Farben für die beiden Spieler, eine Nummerierung der Züge und weitere Optionen sind einstellbar. Wenn sich ein Spiel über den angezeigten Bereich des Spielbretts hinaus ausbreitet, kann man weiterhin durch Scrollbalken alles im Blick behalten.

Auf der rechten Seite wird eine Liste der Züge angezeigt. Im Analysemodus können hier Nebenvarianten zum Spiel und Kommentare eingefügt werden; ein Klick auf einen Zug zeigt das Spielbrett zu diesem Zeitpunkt im Spiel an. Rechts unten ist im Beratungsmodus von "Mensch gegen Computer" ein zusätzliches Fenster, in dem die Engines ihre Zugvorschläge abgeben.

Als Programmiersprache haben wir Borland Delphi gewählt. Ursprünglich hatten wir eine Entwicklung der Software unter Linux ins Auge gefasst; diese Idee haben wir jedoch aufgrund des größeren Aufwands verworfen.

## 2.3 Server-Engine Interface

Die Engines realisieren wir als Win32-Laufzeitbibliotheken (DLLs), die flexibel durch den Server geladen werden können. Diese DLLs können mit jeder beliebigen Windows-Programmiersprache (in unserem Fall ebenfalls Delphi) geschrieben werden. Für die Kommunikation zwischen Server und Engine haben wir ein Interface definiert, das den Austausch folgender Informationen ermöglicht:

- Name, Copyright und Beschreibung der Engine übermitteln,
- Züge vom Server entgegennehmen und in das eigene Spielbrett übertragen,
- eigene Zugvorschläge an den Server übermitteln,
- die Remis-Bereitschaft zurückgeben,
- das Spielende und den Ausgang des Spiels wahrnehmen.

Jede Engine bildet das Spielbrett intern ab und bekommt als Eingabe lediglich bei jedem Zug die Position des neu belegten Feldes und die Nummer des Spielers, der gezogen hat (1 oder 2). Diese Struktur ist weniger anfällig für Programmierfehler in den Engines und reduziert den Datenaustausch zwischen Server und Clients auf ein Minimum.

### 2.4 Die Engines und ihr Teamspiel

Engines geben in der Regel als Ausgabe nicht nur ein Feld zurück, sondern eine Liste von Zugvorschlägen mit jeweiliger Bewertung. Dies ist wichtig für die Kooperation von Engines: Zuerst normalisiert der Server die Bewertungszahlen in der Zugliste, die die Engine zurückliefert, so dass entweder die Summe der Bewertungen oder die höchste Bewertung gleich einer bestimmten Konstante ist (welche Normalisierung angewendet wird, hat merklichen Einfluss auf das "Teamverhalten" der Engines; siehe hierzu das Kapitel "Erforschung").

Wenn ein neues Spiel mit einem Team von Engines begonnen wird, kann man mit der Einstellung "Tune teamplay" jeder Engine einen Gewichtungsfaktor zuweisen (standardmäßig 1.0). Der Server multipliziert dann die Bewertung jedes Zugs, den eine Engine liefert, mit ihrem Gewichtungsfaktor. Wenn eine andere Engine den gleichen Zug ausgegeben hat, werden die gewichteten Bewertungszahlen addiert. Nach abschließendem Sortieren erhält man eine neue Liste von Zügen und Bewertungen, in der die "Meinung" aller beteiligten Engines zusammengefasst ist.

Aus den Bewertungszahlen der Züge berechnet der Server mit der Normalverteilungsfunktion nun die Wahrscheinlichkeit, mit der jeder Zug gewählt wird; anhand der Wahrscheinlichkeit wird dann ein Zug zufällig ausgewählt. Die Wahrscheinlichkeit, mit der auch schlechter bewertete Züge ausgeführt werden, ist durch den Parameter $\sigma$ (sigma) der Normalverteilungsfunktion einstellbar. Dies hat den Vorteil, dass nicht immer der gleiche Zug gewählt wird, und so stets unterschiedliche Spielabläufe auch bei den gleichen Engines zustande kommen.

## 3 Engines

### 3.1 Vorgaben

**Lösungsansätze**

Prinzipiell sind verschiedene Ansätze zum Finden der besten Züge möglich:

- Algorithmen, die die aktuelle Stellung strukturell untersuchen und nach bestimmten Strukturen Ausschau halten, um Züge zu bewerten; hierbei sind unterschiedliche taktische und strategische Ansätze möglich (s.u.);
- Ein "Brute Force"-Vorgehen, bei dem mehrere Züge in die Zukunft gedacht und ein Gewinn bzw. ein Verlust gesucht wird;
- Intelligente, vor allem lern- und anpassungsfähige Algorithmen.

**Terminologie**

- *Strategie/Taktik*
  In den folgenden Erläuterungen wird oft auf den Unterschied zwischen Taktik und Strategie verwiesen.

Wie im Schachspiel definieren wir die Begriffe folgendermaßen: Strategisches Spiel besteht darin, eine möglichst gute Gesamtstellung zu schaffen, ohne Berücksichtigung von unmittelbaren Gewinnmöglichkeiten. Taktik bedeutet die "kurzsichtige" Suche nach Spielzügen, die einen bestimmten Antwortzug des Gegners erzwingen.

- *Anzug/Nachzug*
  Spieler mit Anzug ist der, der den ersten Stein setzt, die Partie also beginnt. Spieler im Nachzug ist folglich der, der den Zweiten setzt.

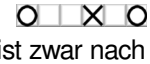
**Konstellationen im Spiel**

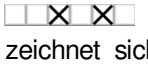
Die Benennungen dieser Konstellationen stammen von uns, da es hierfür keinen Standard gibt. Die Abbildungen zeigen jeweils eine Stellung aus der Sicht des Spielers X; die Steinfolge kann beliebig gedreht im Spielfeld liegen.

- *Offener Vierer* 
  Ein offener Vierer ist eine Anordnung von vier in einer ununterbrochenen Reihe liegenden Steinen, die durch Anlegen eines weiteren Steins *auf beiden Seiten* zu einer Fünferreihe ergänzt werden können.

- *Halboffener Dreier* 
  Ein Dreier, der nur auf einer Seite vervollständigt werden kann, wird als halboffener Dreier bezeichnet.

- *Halboffener Zweier* 
  Zunächst liegt die Vermutung nahe, dass es sich hier um einen offenen Zweier handelt. Dem Spieler X ist es aber nicht möglich, einen offenen Vierer zwischen den beiden gegnerischen Steinen zu errichten.

- *Falscher Einser* 
  Der einzelne Stein ist zwar nach rechts und links frei, beim Abzählen der freien Felder zu beiden Seiten sieht man hingegen, dass hier maximal vier Steine zwischen den beiden gegnerischen Platz haben. Somit ist das Erreichen des Spielziels nicht möglich; diese Konstellation ist völlig wertlos.

- *Pseudo-Zweier* 
  Ein Pseudo-Zweier zeichnet sich durch zwei Eigenschaften aus: Erstens kann er durch Anlegen weiterer Steine zu einem offenen Dreier/Vierer aufgewertet werden. Zweitens liegen seine Steine in einer nicht ununterbrochenen Reihe. Der Gegner kann also auch dazwischen blocken. Ansonsten haben Pseudo-Stellungen im taktischen Bereich den gleichen Wert wie Offene.

## 3.2 Unsere Engines

**Referenz: "Space Shuffle", "The Mirror"**

"Space Shuffle" ist ein Zufallsgenerator, der nicht auf die Reaktion des Gegners eingeht. Die Wahrscheinlichkeit für Felder ist im mittleren Bereich höher gelegt als in Randbereichen, um ein zu weites Ausbreiten zu vermeiden.

"The Mirror" spiegelt den jeweils vorausgegangenen Zug des Gegners am Punkt (0|0).

**Strategie durch Vernetzung von Feldern – "Spider"**

Spider erzeugt zunächst eine Matrix mit "Werten" für jedes Feld: Von jedem eigenen Stein aus wird der Wert aller Felder, die senkrecht, waagerecht oder diagonal in einem maximalen Abstand von vier liegen, um 1 erhöht. Daraufhin untersucht Spider, wie sich die Feldwerte verändern würden, wenn sie jeweils auf ein bestimmtes Feld setzt, und wählt eines derjenigen Felder, die den größten Zuwachs von Feldwerten mit sich bringen.

Durch diese Vorgehensweise kann Spider Stellungen auf strategische Weise erfassen, da die Matrix der Feldwerte die eigenen Steine untereinander "vernetzt" (vom Netz rührt auch der Name Spider). Die Strategie ist also Spiders große Stärke. Da sie jedoch aufgrund taktischer Mängel früher des öfteren Schiffbruch erlitt, wurde sie erweitert und erkennt jetzt wesentliche taktische Merkmale.

**Rücksichtslose Taktik – "Kamikaze"**

Kamikaze untersucht eine Stellung lediglich in Bezug auf die oben beschriebenen Konstellationen und deren Varianten; für jede Konstellation hat sie eine konstante Punktzahl einprogrammiert. Eigene Steine werden dabei höher bewertet als Gegnerische, und offene Reihen werden Halboffenen deutlich bevorzugt.

Für jedes freie Feld wird nun geprüft, zu welchen Konstellationen es gehört, und auf diese Weise sein Wert berechnet. Der Engine sind bestimmte Stellungen bekannt, die zwangsläufig zum Sieg führen (zum Beispiel zwei offene Dreier); das Ziel ist stets, eine solche Stellung aufzubauen. Falls entsprechende Strukturen beim Gegner entdeckt werden, werden diese geblockt. Es werden keine Zugfolgen in die Zukunft gedacht.

**Überlegtes Handeln – "Snotnose"**

Snotnose beherrscht eine Kombination aus Taktik und Strategie; wichtig hierfür ist, dass sie zwischen zwei verschiedenen Arten von eigenen Zügen unterscheidet: Einerseits sogenannte "Tempozüge" (vgl. Schach), die ein sofortiges Reagieren (Blocken) des Gegners erfordern, und andererseits "Planzüge", deren Ziel die Verbesserung der eigenen Stellung bzw. das Hindern des Gegners am Aufbau einer starken Stellung ist.

Wie Kamikaze untersucht Snotnose die Stellung auf Konstellationen, allerdings ist die Punktgebung unterteilt in Taktik-(Tempo-) und Strategiepunkte: Für Tempozüge werden nur taktische, jedoch keine Strategiepunkte vergeben. Wenn die Taktikpunkte unter einem bestimmten Schwellenwert (dem Wert einer garantiert gewonnenen Stellung) liegen, werden sie nicht beachtet. Das führt dazu, dass Snotnose anfangs im Spiel nicht blind Angriffszüge ausspielt, was sie in eine strategisch schlechtere Position bringen würde, sondern aufgrund der Strategiepunkte zuerst eine gute Stellung aufbaut – um dann im geeigneten Moment loszuschlagen. Parallel prüft sie, ob der Gegner geblockt werden muss.

Im Gegensatz zu Kamikaze sind Snotnose nicht alle bekannten Konstellationen vorgegeben; vielmehr vergibt sie Taktikpunkte nur für offene Dreier sowie offene und halboffene Vierer, die durch den nächsten Zug erreicht würden. Danach zählt sie die Anzahl der Züge, die sie mit dieser Stellung bis zum Sieg benötigen würde.

## 3.3 Modularisierung

Um die Eigenschaften und Auswirkungen von Taktik- und Strategiespiel besser beobachten zu können, haben wir zusätzlich zu den bisher erwähnten "vollständigen" Engines einige Enginemodule geschrieben. Ein Modul verfolgt i. A. nur ein einziges Ziel; durch die Zusammenschaltung mehrerer Algorithmen mit teilweise sehr unterschiedlichen Zielen erhofften wir uns eine herausragende Spielstärke im Team, worin wir später bestätigt wurden. (Die vollständigen Engines haben sich im Team oft gegenseitig gestört, wie sich in den "Engine Wars" zeigte – ein Problem, das bei Modulen kaum auftrat.) Die einzelnen Module sind besser auf eine Aufgabe spezialisiert, daher können sich ihre Leistungen ergänzen, wenn das Zusammenspiel funktioniert – im Gegensatz zu den "Allround"-Engines, die mehrere Ziele zugleich verfolgen und deshalb weniger stark auf den einzelnen Gebieten (unterschiedliche Strategien) sind.

**Noch mehr Strategie – mSpider**

Hier wurde die Idee, die hinter Spider steckt, weitergeführt und stark verbessert. Vor allem als Berater zum Verbessern der eigenen Stellung kann sie äußerst nützlich sein.

Wie alle Module sollte mSpider nicht alleine spielen, da sie durch ihre rein strategische Ausrichtung keine Gewinn-/Verlustmöglichkeiten erkennt. Folglich benötigt sie taktische Unterstützung, am besten durch mTactix.

**Reine Taktik – mTactix**

mTactix prüft, ob die Stellung in einer bestimmten Anzahl von Zügen (Suchtiefe derzeit vier Züge) zu gewinnen ist. Falls ja, bewertet sie diese Züge mit bis zu 100%, sonst liefert sie keine eindeutigen Züge.

Wer nur gegen mTactix spielt, wird schon bald festellen, dass ihre Züge – besonders am Anfang einer Partie – wertlos sind, weil sie überhaupt keine strategischen Kenntnisse hat. In Kombination mit einem Strategen (z.B. mSpider, mBattery) kann ein sehr gutes Team entstehen.

#### Strategischer Helfershelfer – mBattery

Dieses Modul wurde als Ergänzung für mTactix geschrieben: Ziel ist es, eine Stellung aufzubauen, die möglichst viele Drohungen in einem Zug zulässt. mBattery bereitet die Stellung für mTactix mundgerecht vor, hat dabei aber selbst keine taktischen Kenntnisse und zieht sich in dem Moment aus der Berechnung zurück, wenn mTactix einsetzt.

#### Fast "intuitiv" – mSnotty

Das Modul zu Snotnose, das in der Berechnung sehr ähnlich blieb, aber dennoch einige Verbesserungen beinhaltet. Anstatt zu prüfen, wieviele Felder bis zum Sieg benötigt werden, wird gezählt, wieviele gebraucht werden, um eine Drohung zu erschaffen. Die resultierende Spielweise ist der menschlichen Intuition ähnlich.

Der Stratege, der zudem die wichtigsten Siegstellungen kennt, bringt das beste Resultat erstaunlicherweise mit einem weiteren Strategen: mArea51.

#### Neue Welt – mArea51

Die Bewertung jedes Feldes hat Einfluss auf die Wertigkeit der umliegenden Felder. Außerdem werden offene Konstellationen deutlich höher bewertet als Pseudo-Stellungen. Dank dieser neuen Ansätze überspielt mArea51 ihre Gegner häufig.

Die beste Kombination ergibt sich hier mit mSnotty, beide zusammen entwickeln ein hervorragendes Teamspiel.

### 3.4 Weitere Pläne

#### Pseudo-lernfähige Engine - TeachMe

Diese Engine besteht eigentlich aus zwei Engines: einer Lern-Engine (LE) und einer Spiel-Engine (SE). TeachMe ist nicht wirklich lernfähig, d.h. sie lernt nicht aus eigenen Erfahrungen, sondern vom Spiel anderer Engines. Dabei wird sie maximal so gut wie ihre Lehrer.

Die LE lernt, indem sie bei Engine Wars mitspielt. Die Reaktionen ihrer Mitspieler auf bestimmte Stellungen speichert sie in einer Wissensbasis. Die SE spielt anhand dessen, was die LE gelernt hat und in die Wissensbasis geschrieben hat. Jedesmal, wenn sie am Zug ist, überprüft sie, ob die auf dem Brett vorhandenen Stellungen in ihrer Wissensbasis vorkommen und setzt entsprechend.

#### Eröffnungsbibliothek

In der Eröffnungsphase liefern die Engines des öfteren Züge, die wenig Sinn ergeben. Um diesen Mangel zu beheben, planen wir, Eröffnungsrepertoires in Form von Modulen einzubinden. Diese wären nur während der ersten paar Züge aktiv. Die Eröffnungsmodule sollen auf eine eigene Datenbank zugreifen, die ähnlich der lernfähigen Engine ständig erweitert werden kann. Um die Anzahl möglicher Eröffnungen zu reduzieren, lassen sich Symmetrien unter den Zügen ausnutzen.

#### Neuronale Netze

In der Planung einer wirklich intelligenten Engine, evtl. basierend auf neuronalen Netzen, sind wir erst am Anfang. Derartige Entwicklungen gingen über den gesetzen Rahmen unseres Projektes vorläufig hinaus.

### 3.5 Gescheiterte Ansätze

#### Brute Force

In Planung war eine Brute-Force-Engine, die nur Zugfolgen in die Zukunft berechnet und dabei auf Gewinn oder Verlust hinarbeitet. Sie sollte keine vorgegebenen Konstellationen erkennen oder behandeln.

Nachdem wir den Brute-Force-Algorithmus theoretisch geplant hatten, war klar, dass bei einer sinnvollen Suchtiefe (>7 Züge) selbst mit bereits entwickelten Optimierungen die Berechnung nicht zu bewältigen wäre.

**Pure Strategie – "Greenhorn", "mGreenhorn"**

Greenhorn war eine der ersten Engines. Sie macht bei ihren Berechnungen keinen Unterschied zwischen offenen, halboffenen oder Pseudo-Konstellationen. Sie zählt lediglich, zu wie vielen eigenen Steinen sie durch Setzen auf das jeweilige Feld "Kontakte knüpfen" kann. Nachdem sie sich als strategisch unbrauchbar erwiesen hat und größere Änderungen praktisch zu einer Neuprogrammierung geführt hätten, haben wir sowohl die Engine "Greenhorn" als auch das Modul "mGreenhorn" verworfen.

# 4 Erforschung

## 4.1 Die große "Battle of Engines"

Für die Bewertung der Engines und die Beobachtung ihrer Eigenschaften haben wir im Server eine Funktion eingebaut, um einzelne Engines (oder zusammengesetzte Teams) automatisch gegeneinander spielen zu lassen. Dabei hat der Benutzer zunächst die Möglichkeit, die antretenden Engines und die Zahl der Spiele, sowie einige weitere Optionen festzulegen. Nachdem die Spiele durchgeführt wurden, werden alle Partien in eine Datenbank eingetragen und eine Statistik angezeigt. Die Statistik gibt Aufschluss über Gewinne, Remis und Verluste; Partielängen; Auswirkungen der Tatsache, welcher Spieler begonnen hat; Beitrag einzelner Engines zum Gesamtergebnis etc.

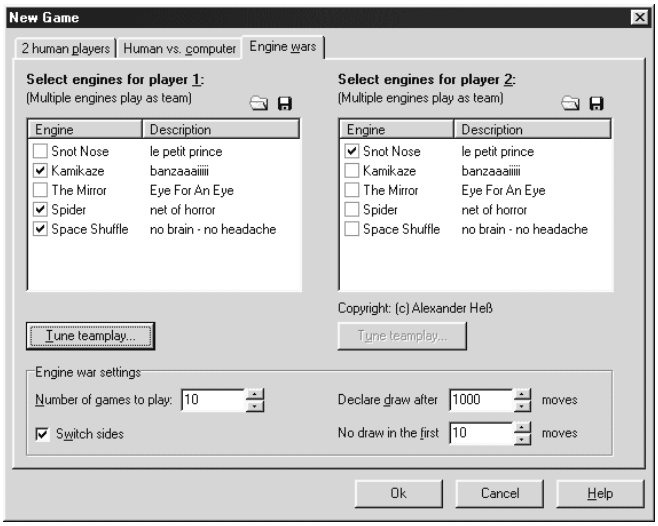



Abbildung 3: Auswahlmaske für "Engine Wars" im Server

Um die Möglichkeiten, die das Zusammenschließen mehrerer Engines eröffnen, zu verdeutlichen, sei folgendes angemerkt: Bei $n$ Engines sind bereits $k = (2^n\text{-}1)^2$ Paarungen von gegnerischen Engine-Teams möglich (bei fünf Engines wären das schon 961 denkbare Kombinationen). Außerdem bietet unser Programm die Option "Tune teamplay", die es ermöglicht, den Einfluss der verschiedenen Engines auf den Spielablauf einzustellen (in %). Wenn man die verschiedenen Prozentwerte dazurechnet, ergeben sich für n Engines $M$ Kombinationsmöglichkeiten:

$$M(n) = \left[ \sum_{i=1}^{n} \binom{n}{i} \cdot 100^{(i-1)} \right]^2$$

Näherungsweise entspricht das $M(n) = 10^{4(n-1)}$. Bei 4 Engines gibt es etwa $10^{12}$ mögliche Paarungen und Einstellungen, bei 5 Engines etwa $10^{16}$. Daher gehen wir hier auf eine repräsentative Auswahl von Spielen unterschiedlicher Engines ein.

## 4.2 Spiele von Engines alleine und im Team

Für die folgenden Engine-Spiele galten folgende Regeln: 100 Spiele nacheinander mit wechselndem Anzug, Remis bei mehr als 250 Zügen, Remisangebote der Engines verboten. Es sei auch angemerkt, dass die Ergebnisse aufgrund der teilweise zufälligen Auswahl von Zügen ($\sigma = 10$) schwanken können.

**Spielweise der Engines gegen sich selbst**

Als repräsentatives Beispiel beschreiben wir hier die Spiele zweier Instanzen von Kamikaze gegeneinander. Seinen Anzugsvorteil konnte Kamikaze nicht allzu oft ausnutzen. Durch seinen rücksichtslosen Angriffstil bringt er sich nicht selten selbst in Bedrängnis, d.h., dass er sich nach dem Ausspielen aller taktischen Möglichkeiten manchmal in (strategisch) verlorener Position wiederfindet: Dadurch, dass er den Gegner ständig zum Blocken zwingt, schließt er sich quasi selbst ein.

Es zeigte sich, dass Engines allgemein im Spiel gegen sich selbst zwar häufiger im Anzug als im Nachzug gewinnen, das Verhältnis beträgt jedoch höchstens 60:40. Es gibt also den oben erwähnten Anzugsvorteil, der jedoch i.A. nur wenig ausschlaggebend ist.

**Kamikaze – Spider**

Trotz seiner strategischen Überlegenheit war Spider dem Taktiker Kamikaze hoffnungslos ausgeliefert: Von 100 Spielen errang Spider lediglich vier Siege, je zweimal im An- und Nachzug. Dementsprechend kurz waren die Partien. Daraus wird ersichtlich, dass ein Spiel ohne solide Taktikkenntnisse der Engine wenig aussichtsreich ist.

**Snotnose – Kamikaze**

Dieses Duell konnte Snotnose, wenn auch nur knapp, für sich entscheiden. Aufgrund der zusätzlichen strategischen Fähigkeiten von Snotnose hatten wir ein deutlicheres Ergebnis für sie erwartet. Wir erklären uns das so, dass Snotnose von seiner Strategie nicht viel Gebrauch machen konnte, da sie im Spiel hauptsächlich taktisch beschäftigt war. Das Ergebnis war 52:48 mit gleicher Verteilung im An- und Nachzug.

**Snotnose, Spider – Kamikaze**

Das deutliche Ergebnis von 26:74 lässt vermuten, dass sich die beiden Strategien von Snotnose und Spider durch ihre unterschiedliche Stellungsbehandlung gegenseitig behindern. Wegen der zufälligen Auswahl kann sich keine der beiden Strategien durchsetzen, was meist zu einer zusammenhangslosen Stellung führt.

## 4.3 Spiele gegen zusammengeschlossene Module

**mSpider, mTactix – Snotnose**

Nach den spielerischen Desastern in allen Kombinationen von Spider legten wir den Schwerpunkt unserer Entwicklung auf die Enginemodule. Die erste und sogleich sehr erfolgreiche Kombination waren mSpider und mTactix.

Das Resultat der Spiele gegen Snotnose hat mit 85:15 unsere kühnsten Erwartungen übertroffen. Das Zusammenspiel zwischen den zwei Modulen klappte hervorragend: Solange mTactix noch keine Gewinnmöglichkeit sieht, hat mSpider größtenteils die Kontrolle. Ergibt sich jedoch eine Gewinnstellung, greift mTactix mit hohen Zugbewertungen durch und übernimmt das Spiel. An dieser Stelle sei noch einmal erwähnt, dass es zwischen den Modulen keine direkte Kommunikation gibt – beide teilen ihre Zugvorschläge dem Server durch die Bewertungszahlen mit.

**mTactix, Snotnose – Snotnose**

Nach den Beobachtungen im letzten Modulspiel war das Ergebnis dieser Begegnung abzusehen. Wie im Zusammenspiel mit mSpider zeigt mTactix auch hier ein sehr "kollegiales" Zusammenspiel: Sie lässt ihren Teamkollegen den Vortritt, solange sie noch keine unmittelbare Gewinnmöglichkeit sieht. Somit spielte Snotnose in der Anfangsphase der Spiele praktisch gegen sich selbst, bis mTactix an einem Punkt durchgriff. Das Ergebnis: 88:12 für das Team.

**Notwendigkeit von Strategie**

mTactix unterscheidet sich von den anderen Taktikengines insbesondere durch eine höhere Suchtiefe der Züge. Da wir jedoch der Überzeugung waren, dass reine taktische Vorausüberlegung ohne strategische Ziele nicht das Patentrezept sein kann, haben wir im letzten Spiel die vereinigte taktische und strategische Macht der drei vollständigen Engines Kamikaze, Snotnose und Spider gegen mTactix allein antreten lassen.

Wir wurden bestätigt: Mit 42:58 unterlag die reine Taktik von mTactix den drei kooperierenden Engines. Eine Untersuchung der einzelnen Partien zeigte die erwartete Ursache: Obwohl die drei Engines weniger weit in die Zukunft rechnen als mTactix, bauen sie eine strategisch günstige Stellung auf. Da mTactix meist nicht sofort gewinnen kann, wird sie von ihren Gegnern eingeschlossen oder erreicht andere, für sie ungünstige Stellungen und verliert schließlich, da ihre Stellung unhaltbar wird.

In späteren Beobachtungen hat sich ergeben, dass das Ergebnis sogar noch deutlicher hätte sein können, da sich die drei Engines teilweise störten. Daher konzentrierten wir unsere Neuentwicklung von Engines auf eine Reihe von Strategiemodulen.

**mTactix, mSpider – mTactix, mBattery**

Das Strategiemodul mBattery, welches speziell als Erweiterung auf mTactix zugeschnitten wurde, erfüllte seine Aufgabe wie erwartet: Es bereitete die Stellung stets so vor, dass mTactix zuschlagen konnte. Dadurch war das neue Team der Kombination mit mSpider überlegen: Es gewann mit 59:41.

**mTactix, mGreenhorn – mTactix, mSpider**

Mit diesen Spielen bestätigen wir den verfehlten Ansatz von Greenhorn beziehungsweise mGreenhorn: Die Strategie erweist sich für "Fünf gewinnt" als nicht sinnvoll, wie das Ergebnis von 28:72 bestätigt. Wie sich herausstellte, zielt Greenhorn's Strategie darauf ab, möglichst viele (also mehr als fünf) Steine aufzureihen; dies führt dazu, dass die Steine häufig zu weit auseinander gesetzt werden.

**mTactix, mArea51 – mSnotty, mArea51**

Die geballte Strategie, die hinter mSnotty und mArea51 steckt, reichte aus, um die Kombination aus Taktiker und Stratege zu bezwingen (54:46). Die Strategen erspielen sich ihre Gewinnstellung meist "unbewusst" und setzten dank der richtigen Stellungsbeurteilung die Partie richtig fort. mTactix konnte seinen Vorteil, zwei Züge weiter zu rechnen als mSnotty, nur selten ausnutzen.

**Allgemein: Vergleich Module – Engines**

Um den Unterschied zwischen einer Engine und dem daraus abgeleiteten Modul zu verdeutlichen, haben wir stellvertretend mSpider / Spider mit mTactix gegen mSnotty spielen lassen. Mit Spider als Teamkollege war das Ergebnis genau 50:50, mit mSpider 67:33. Daraus wird abermals ersichtlich, dass Module den Engines im Teamspiel weit überlegen sind.

# 5 Entdeckungen, Enttäuschungen, Erweiterungen

## 5.1 Zusammenfassung der Beobachtungen

Für das Spielen im Team hat noch eine weitere Einstellmöglichkeit im Server erhebliche Auswirkungen: Nachdem jede Engine ihre Zugwünsche zurückgeliefert hat, muss der Server die Bewertungszahlen normalisieren, so dass die Werte der Engines untereinander vergleichbar werden. Ursprünglich setzte die Normalisierungsfunktion den besten Zug, den eine Engine zurücklieferte, auf eine bestimmte Konstante und passte alle anderen Bewertungszahlen im Verhältnis dazu an.

Da wir mit diesem Vorgehen jedoch ein oft schlechtes Teamspiel beobachteten, änderten wir diese Funktion: Nun wird die Summe der Bewertungszahlen aller Züge einer Engine auf eine bestimmte Konstante gebracht. Hierdurch konnten wir die Spielstärke der Teams in allen Kombinationen deutlich verbessern: Wenn eine Engine keinen besonders guten Zug sieht, liefert sie viele ähnliche Bewertungszahlen zurück. Durch die aufsummierte Normalisierung fallen diese weit weniger ins Gewicht als ein einziger herausragender Zug, den eine andere Engine erkennt. Um dieses Verhalten beobachten zu können, haben wir die Art der Normalisierung im Server einstellbar gelassen.

Anders als erwartet, spielen Hochleistungsengines auch bei Kooperation mit der Zufalls-Engine "Space Shuffle" relativ gut. Das liegt daran, dass "Space Shuffle" seine Zugbewertung für die einzelnen Felder auf nur minimal unterschiedliche Werte setzt. Die kooperierenden Engines setzen für ihre Favoriten-Felder jedoch relativ hohe Werte an, die durch die minimalen Unterschiede durch "Space Shuffle" kaum verfälscht werden, insbesondere bei der verbesserten Normalisierung.

## 5.2 Schlussfolgerungen

In unserer Forschung über das Verhalten der Engines haben sich unsere Vermutungen, die bei der Konzeption der Algorithmen bereits vorhanden waren, weitgehend bestätigt. Eine interessante Erkenntnis

ist, dass sich ähnliche Funktionen in zusammengeschalteten Engines weitgehend stören, während sich gänzlich unterschiedliche Funktionen ergänzen können. Das ist der Grund, weshalb die "Allround"-Engines schlecht, die Module dagegen hervorragend kooperieren.

"Fünf gewinnt"-Spiele werden unseren Erkenntnissen nach hauptsächlich durch Taktik entschieden – der Strategieteil darf jedoch, wie sich wiederholt gezeigt hat, nicht vernachlässigt werden. Die Modularisierung dieser Spielweisen erwies sich als sehr fruchtbar, da hier einerseits die einzelnen Module besser weiterentwickelt werden können, und andererseits der Einfluss der unterschiedlichen Spielstile besser kontrolliert und untersucht werden kann.

Es ist uns gelungen, verschiedene Algorithmen korrekt umzusetzen, sowie eine funktionsreiche Serversoftware zu entwickeln.

Den Vorteil des Spielers, der den ersten Stein setzt, konnten wir bei den Engines deutlich beobachten: So gut wie immer gab es auf beiden Seiten im Anzug mehr Siege als im Nachzug. Allerdings konnten wir noch keinen Hinweis auf einen zwangsläufigen Gewinn des ersten Spielers finden, und stellen daher die anfangs genannte Hypothese in Frage.

Bei Spielen gegen einen menschlichen Gegner zeigte sich deutlich, dass der Mensch fast immer durch Leichtsinnsfehler oder Unaufmerksamkeit verlor. Diese Faktoren sind natürlich beim Computer nicht gegeben. Gegen einige Enginekombinationen hat der Mensch praktisch keine Chance; selbst trainierte Spieler halten sich kaum länger als ca. 30 Züge. Wir haben aber bereits von einigen Spielern gehört, die sich das Spiel von unserer Internetseite geladen haben und die behaupten, selbst gute Kombinationen regelmäßig zu besiegen: Während der Mensch seine Spielstrategie üblicherweise ständig ändert (durch Gewöhnung und Lernen aus Fehlern), spielen unsere bisherigen Engines immer nach dem gleichen Muster, welches ein Mensch mit etwas Übung durchschauen kann.

## 5.3 Ausblick

Wir beabsichtigen, unsere Entwicklung und Forschung in "Fünf gewinnt" auch über "Jugend forscht" hinaus fortzuführen. Die Entwicklung eines benutzerfreundlichen Servers und einem exakt spezifizierten Interface zu den Engines war uns wichtig, da wir das Projekt veröffentlichen und dabei Dritten die Möglichkeit geben wollen, selbst Algorithmen und Engines zu entwickeln.

Auf unserer Internetseite (http://www.morphion.de) gibt es unsere eigenen Engines und Server inklusive Quelltext zum Herunterladen, darunter auch Schablonen (Templates) für Engines, in die nur noch ein Algorithmus eingefügt werden muss. Zudem soll es Möglichkeiten zum Informationsaustausch geben: Wir planen ein Forum, die Publizierung von Engine Wars-Ergebnissen, Partien zum Download, etc.

Unsererseits wollen wir die Software weiterentwickeln: Vorgesehen ist neben weiterer Verbesserung des Servers (Netzwerkfähigkeit, Gestaltung, Geschwindigkeit) natürlich die Entwicklung weiterer Engines und vor allem Module, um damit neue, verbesserte Ansätze zu finden. Vermutlich gibt es keinen Ansatz, der in allen Situationen optimal reagiert; wir wissen auch nicht, ob die "optimale" Behandlung einer jeden Stellung überhaupt existiert.

## 5.4 Nutzbarkeit in anderen Fachgebieten

Auf dem Regionalwettbewerb wurde unter anderem der Vorschlag an uns herangetragen, über eine militärische Nutzung unserer Erkenntnisse nachzudenken. In der Tat gibt es einige Ähnlichkeiten zwischen dem Ablauf eines Spiels und militärischen Operationen; diese reichen unserer Meinung nach jedoch nicht soweit, dass wir diesen Vorschlag ernsthaft in Erwägung gezogen hätten.

Ein weiterer Vorschlag war, das Spiel in die Dreidimensionalität zu erweitern und Entsprechungen im Bereich der Chemie, beispielsweise bei der Bildung von Molekülketten, zu suchen.

Derartige Weiterentwicklungen gingen jedoch sehr weit über unser Themen- und Fachgebiet und über unsere Kenntnisse hinaus.

# 6 Ende

## 6.1 Änderungen und Erweiterungen seit dem Regionalwettbewerb

Seit der Abgabe der Arbeit für den Regionalwettbewerb haben wir einige Module implementiert und ihr Verhalten untersucht. Die Module sind oben vorgestellt.

Die schriftliche Arbeit mussten wir durch die neuen Inhalte stellenweise etwas kürzen; auch haben wir einige Abschnitte innerhalb des Dokuments verschoben und sprachliche Anpassungen vorgenommen. Die frühere Version der schriftlichen Arbeit kann auf unserer Internetseite (http://www.morphion.de) eingesehen werden.

## 6.2 Danksagungen

Zuerst danken wir *Borland Inprise*, die durch die kostenlose Stellung von *Delphi* die Arbeit erst ermöglicht haben.

Ein ganz großer Dank geht an Familie Kleppmann, bei der wir uns für zwei Wochen Intensivvorbereitung eingenistet haben; außerdem an Herrn W. Kleppmann für seine Hilfe in mathematischen Fragen (Berechnung der Normalverteilung).

Außerdem danken wir unserem stellvertretenden Schulleiter Herrn Seyfert für die Möglichkeit der Nutzung schulischer Geräte.

## 6.3 Quellenverzeichnis

- http://www.renju.nu – Die offizielle Seite des Renju-Spiels.
- http://www.steffengerlach.de/xo5k/ – Der Programmierer Steffen Gerlach hat hier ein online spielbares "Fünf gewinnt"-Spiel ins Netz gestellt.
- http://ivs.cs.uni-magdeburg.de/~mahrenho/seminar/vg-strategie.html – Daniel Mahrenholz veröffentlicht hier detaillierte Algorithmen für "Vier gewinnt", die sich in begrenztem Umfang auf "Fünf gewinnt" übertragen lassen.
- http://www.dgob.de – Die offizielle Seite des deutschen Go-Bundes e.V.