

Rechnerpraktikum
aus Programmierung (Java)
WS2006:



Stefan Beranek

1. Januar 2007



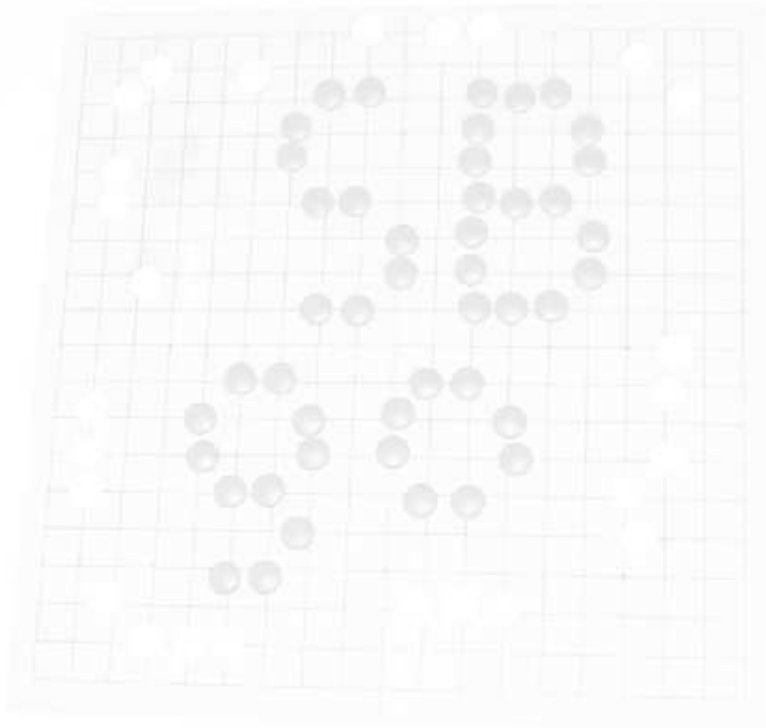
Inhaltsverzeichnis

1 Problemstellung	7
2 Problemlösung	9
2.1 Analyse - Das Programm und seine Use-Cases	9
2.2 Modellierung - Sequence-Diagramm	10
3 Implementierung	11
3.1 Beschreibung der Programmfähigkeiten	11
3.2 Die verwendeten Klassen	12
3.2.1 Klasse sbGUI	12
3.2.2 Klasse sbGO	13
3.2.3 Klasse Feld	14
4 Regelwerk von GO	15
4.1 Überblick	15
4.2 Regelwerk im Detail	15
4.2.1 Erlaubtes Feld	15
4.2.2 Gruppe	15
4.2.3 Wertung	15
5 Organisatorisches	17
5.1 Projekttagbuch - Zeitangaben in UTC (2006)	17
5.2 Installation und Wartung	17



Vorwort

Dieses Projekt wurde von mir im Rahmen des Faches „Priv. Doz. Dr. Michael Hahsler: Rechnerpraktikum aus Programmierung (Java)“ im Wintersemester 2006 durchgeführt. Ich habe mich hierbei für ein kleines GO-Spiel entschieden, da ich privat immer schon von GO fasziniert bin.





Kapitel 1

Problemstellung

Im Rahmen dieses Projekt wird ein GO-Spiel mit GUI, basierend auf der Programmiersprache JAVA, erzeugt. Dabei setzen 2 Spieler abwechselnd einen Stein und versuchen sich gegenseitig zu umschliessen (zu den Regeln sei hier nur auf das Kapitel Regelwerk verwiesen); Umschlossene gegnerische Steine darf man vom Feld nehmen und am Ende werden umschlossene freie Felder und „gefressene“ gegnerische Steine zusammengezählt und so der Sieger entschieden. Dieses Projekt sollte mit mäßigen Vorgaben in 20-40 Arbeitsstunden machbar sein.



Kapitel 2

Problemlösung

2.1 Analyse - Das Programm und seine Use-Cases

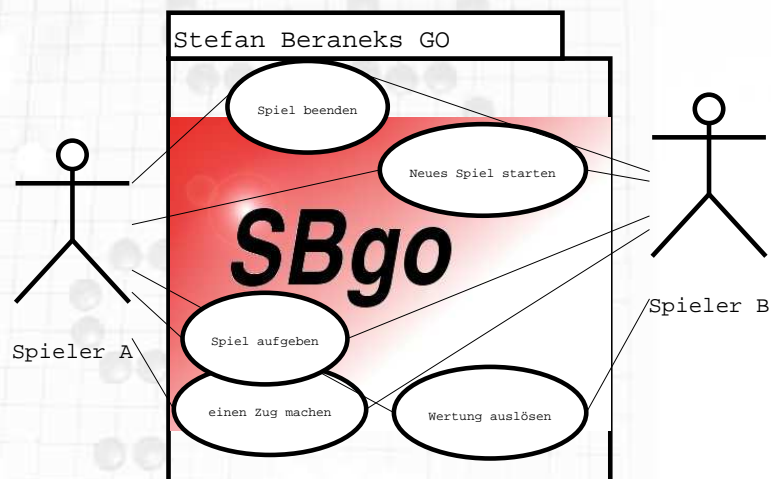


Abbildung 2.1: Use-Case aus der Benutzersicht

Da es sich bei diesem Programm um eine sehr einfache Anwendung handelt, werden nur eine geringe Anzahl von Use-Cases benötigt und realisiert.

2.2 Modellierung - Sequence-Diagramm

Der Ablauf des Programmes teilt sich dabei in die *Spielphase* und in die *Wertungsphase* auf und ist hier in zwei einfachen Sequence-Diagrammen dargestellt.

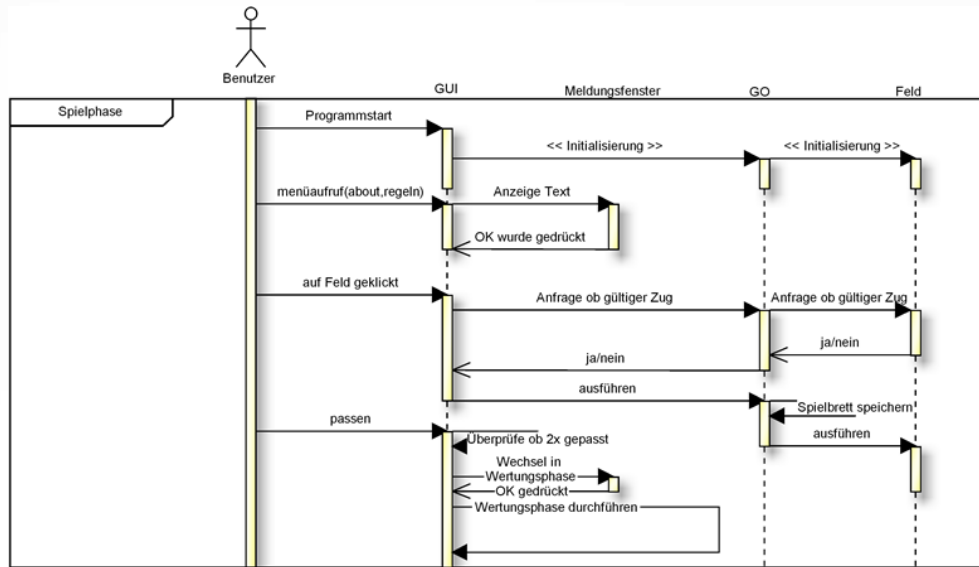


Abbildung 2.2: Sequence-Diagramm für die Spielphase

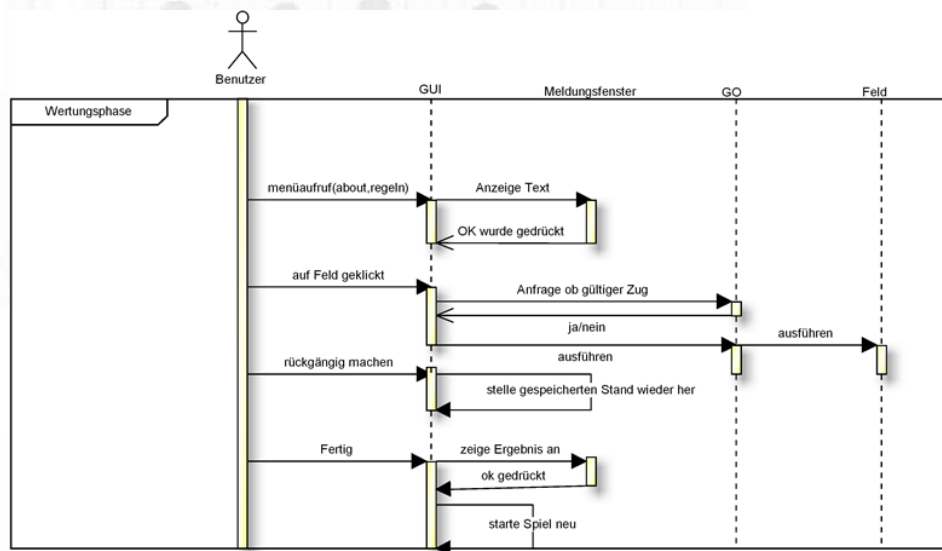


Abbildung 2.3: Sequence-Diagramm für die Wertungsphase

Kapitel 3

Implementierung

Das Programm bietet dem Benutzer ein sehr schlichtes Interface, als Benutzer kann man nur einige wenige Einträge in einem Drop-Down-Menü anwählen, einen der Knöpfe neben dem Spielfeld betätigen oder eine Position am Spielfeld anklicken, um einen gültigen Zug durchzuführen bzw. einen Stein in der Wertungsphase zu entfernen.

Obwohl dieses Spiel als zwei-Personen Spiel ausgelegt ist, basiert dies ausschliesslich darauf, dass die Spieler Schwarz und Weiß abwechselnd ihren Zug wählen und diesen eingeben. In der Wertungsphase wird dann überhaupt nicht mehr zwischen den zwei Spielern unterschieden.

Als Grafiken wurden nachbearbeitete Fotos meiner Spielsteine und als Hintergrund des Brettes eine Aufnahme der Holzmaserung meines Vorzimmerkastens benutzt, sowie ein eine nicht computergenerierte Spielsituation als Hintergrundbild der Dokumentation.

Bei der Entwicklung kamen *SUN-Java*, *eclipse* als Entwicklungsumgebung sowie *dia* für die Use-Cases und der *UML-Diagrammer* für das Sequence-Diagramm zur Anwendung. Zur Bildbearbeitung und Textbearbeitung wurden *pdfTEX* und *gimp* benutzt.

Als einzige nicht absolut notwendige Funktionen besitzt das Programm eine Ausgabe der Züge auf Standard-Output, sowie in der normalen Spielphase einen „Zeige alle gültigen Spielzüge“-Knopf, der dafür sorgt, daß immer die gültigen Züge angezeigt werden, was vielleicht gerade für einen Anfänger von Vorzug ist.

3.1 Beschreibung der Programmfähigkeiten

Das Programm wird gestartet, automatisch beginnt ein neues Spiel, das Programm befindet sich im SPIELMODUS. Schwarz ist am Zug, was durch eine schwarzen Hintergrund hinter dem SPIelbrett angezeigt wird. Außerdem ist in der Titelleiste der aktuelle Punktestand zu sehen.

Der Benutzer kann jetzt ein gültiges Feld anklicken, alle gültigen Züge sind optisch hervorgehoben, falls der „Alle gültigen Züge anzeigen“-Knopf gedrückt ist, dann wird der entsprechende Zug durchgeführt. Anschliessend ist die andere Farbe am Zug usw. Bei jedem Zug kann der Spieler statt eines Feldes auch auf „Passen“ (kein Stein setzen, andere Farbe ist am Zug) oder auf „Aufgeben“ (übergang zur → EREIGNISSANZEIGE) klicken. Über das Menü kann jederzeit das Programm beendet werden („Quit“) oder das aktuelle Spiel verworfen und ein neues Spiel begonnen werden („Neues Spiel starten“) sowie eine kurze Information zum Spiel („Über Stefan Beranek's GO“) angezeigt werden. Auch ist ein kurzer Text zu den Regeln anzuwählen. Wenn 2 Züge hintereinander auf „Passen“ geklickt wurde, geht das Spiel in den → WERTUNGSMODUS über.

WERTUNGSMODUS: Hier werden beliebig viele Steine durch anklicken vom Brett entfernt, bis alle toten Gruppen (siehe Regelwerk) entfernt sind, dann klickt man rechts auf „Fertig“, das Programm wechselt zur → EREIGNISSANZEIGE. Alternativ kann man auch auf „Ausgangszustand wiederherstellen“ klicken, wenn man einen falschen Stein entfernt hat.

Danach kommt eine abschliessende ERGEBNISSANZEIGE in der der Gewinner angegeben wird und der

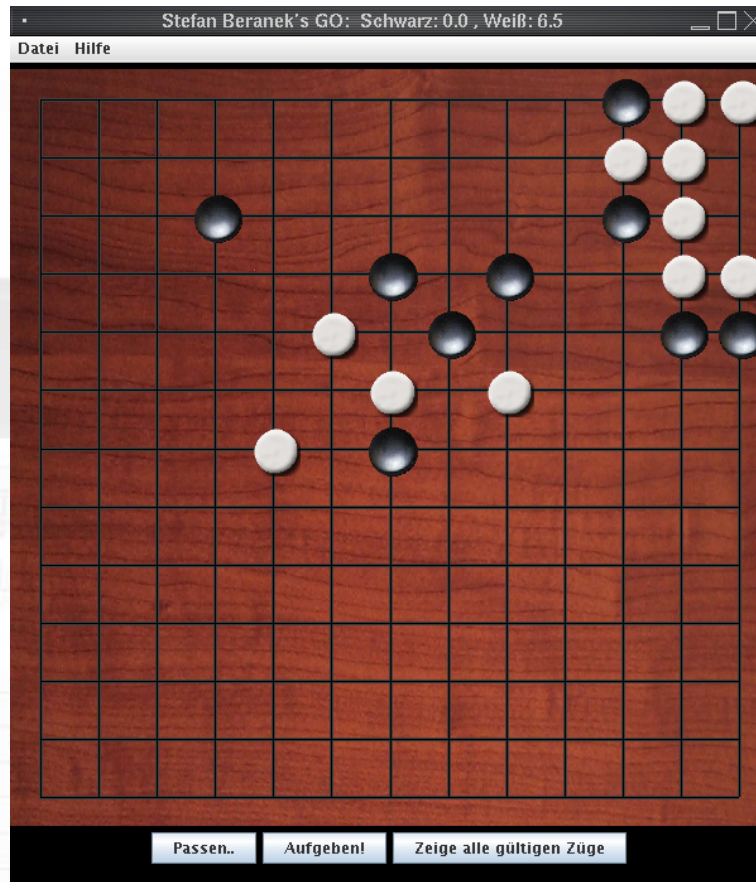


Abbildung 3.1: Ein Screenshot zur Veranschaulichung

Punktestand, falls vorhanden. Wenn man in dieses Fenster klickt, beginnt das Spiel wieder von neuem.

3.2 Die verwendeten Klassen

Das Spiel besitzt die Klasse `sbGUI` die sämtliche Interaktion mit dem Spieler erledigt, sowie die Klasse `sbGO` die die regeltechnischen Fragen (Was ist ein gültiger Zug, wer hat wieviele Punkte, etc..) klärt. Dazu nutzt `sbGO` ausgiebig die Funktionen der Klasse `Feld`; Diese Klasse wiederum soll alle Aufgaben (und nur diese) bewältigen, die durchführbar sind, ohne Wissen über den zukünftigen bzw. vergangenen Spielablauf und regeltechnisch von Bedeutung sind.

3.2.1 Klasse `sbGUI`

Als Einstiegspunkt in das Programm dient eine `main`-Methode die sich in dieser Klasse befindet. Wie bereits beschrieben kümmert sich diese Klasse fast nur um die direkte Interaktion mit dem Benutzer, mithin wird hier auch das GUI aufgebaut, sowie die spieltechnischen Handlungen wie vom Benutzer gewünscht angestossen.

Zur Darstellung des Guis hält sich diese Klasse das Hauptfenster in Form des `mainframe` mit seiner Contentebene `content` vor. Die Knöpfe neben dem Spielbrett werden auch einzeln aufgehoben, um sie bei Bedarf gezielt entfernen zu können (`jbaufgeben`, `jbpassen`, `jvalid`, `jbfertig`). Weiters werden die Bilddaten für die Steinbilder und das Spielbrett sowie das gerade dargestellte Brett mit allen eingetragenen Steinen (`whitestone`, `blackstone`, `background`, `board`) aufbewahrt.

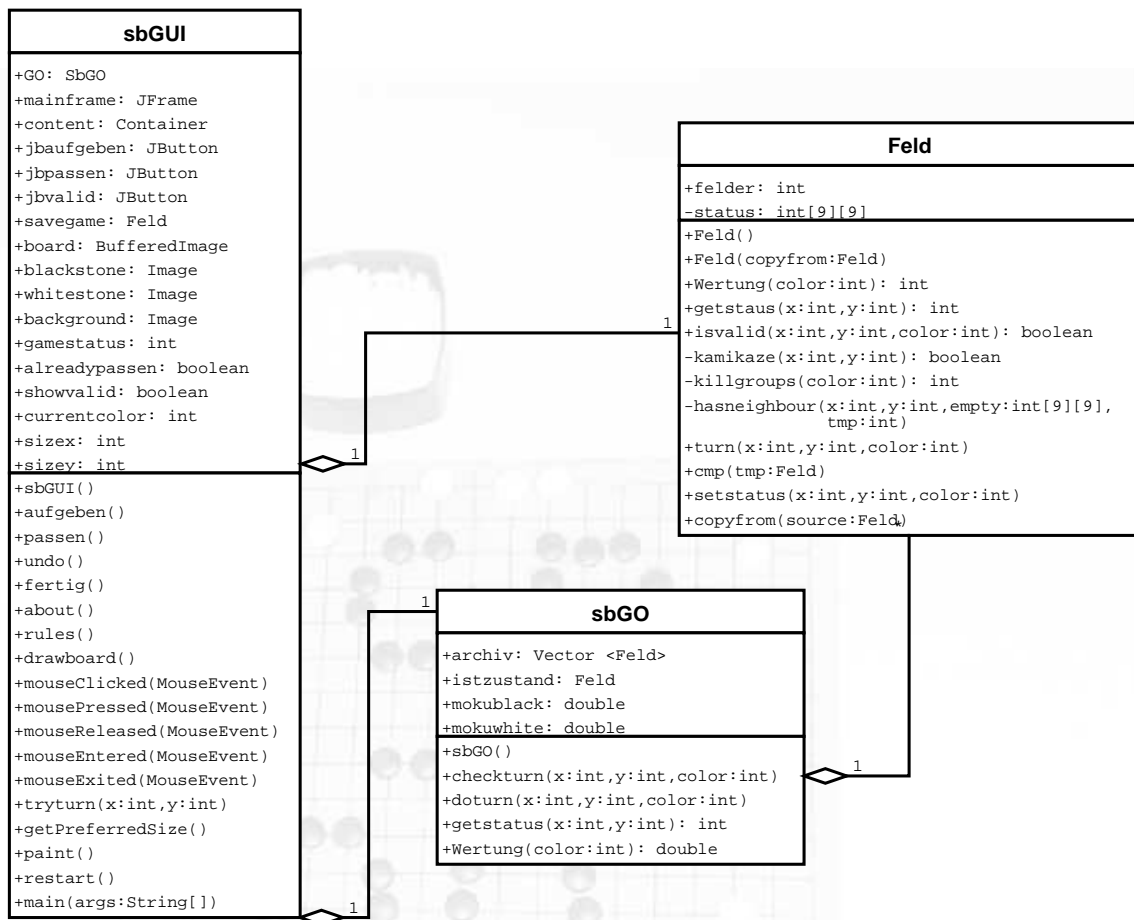


Abbildung 3.2: Die Klassen dieses Projekts

Spieltechnisch wird hier noch die Größe des Spielfeldes als Konstante (*size*), die aktuelle Spielphase (*gamestatus*) und der Spieler, der am Zug ist (*currentcolor*) vermerkt.

An Methoden sind die von der Java-Swing Bibliotheken abgeleiteten Methoden (*paint()*, *mouseClicked()*, *mousePressed()*, *mouseReleased()*, *mouseEntered()*, *mouseExited()*, *getPreferredSize()*) zu nennen, die ausschliesslich dem Empfangen und Verarbeiten der Anweisungen bzw. Events der Oberfläche dienen. Für die Betätigung der Knöpfe ist noch für jeden Knopf und Menüeintrag eine Callback-Routine erstellt worden (*aufgeben()*, *passen()*, *undo()*, *fertig()*, *about()*, *rules()*, *restart()*). An verbleibenden Methoden ist nur mehr *drawboard()*, das den jetzigen Spielzustand zeichnet sowie *tryturn()* zu erwähnen, das einen Zug auf Gültigkeit im aktuellen Kontext (Wertungs- oder Spielphase) prüft.

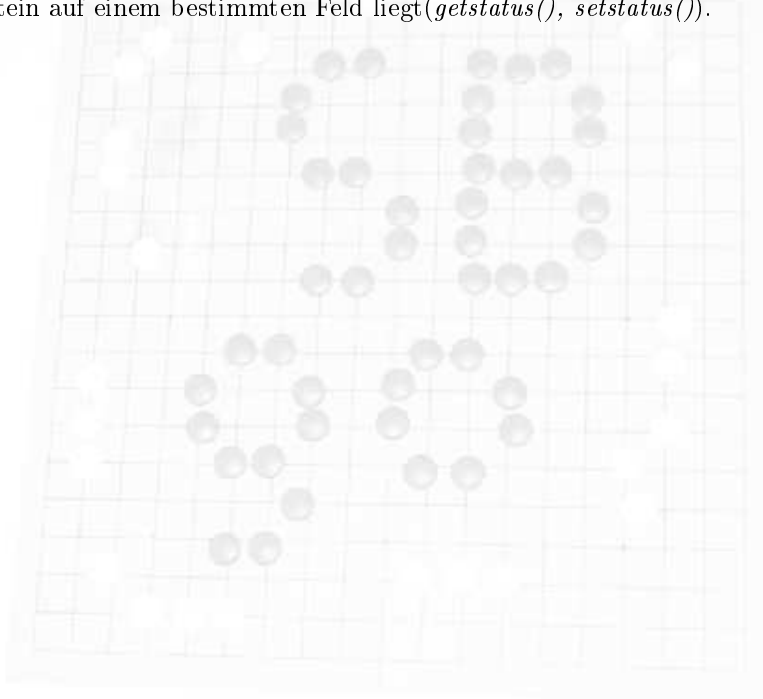
3.2.2 Klasse sbGO

Diese Klasse implementiert alles regeltechnische für dieses Spiel, was nicht bereits durch die Klasse *Feld* erfüllt wird. Dazu hält sie sich sowohl den aktuellen Zustand des Spielbrettes unter *istzustand* als auch alle bisherigen Zustände des Brettes im *archiv* in Evidenz. Des weiteren kann ein Zug auf regeltechnische Gültigkeit überprüft werden mit *checkturn()* und anschliessend mit *doturn()* durchgeführt werden. In den Variablen *mokublack* + *mokuwhite* merkt sich diese Klasse die gefressenen gegnerischen Steine. Der Aufruf der Methode *Wertung()* retourniert den aktuellen Punktestand unter Berücksichtigung der bereits gefressenen Steine.

3.2.3 Klasse Feld

Die Instanzen der Klasse Feld repräsentieren jeweils eine Brettaufstellung, die physikalisch möglich (ein Stein pro Kreuzung), aber nicht unbedingt regeltechnisch gültig sein muss. Ein Feld kann leer erstellt werden oder als Kopie eines bereits bestehenden Feldes (Anlage des Objektes mit Übergabe einer zu kopierenden Vorlage). Es kann die Punkte einer Farbe mit der Funktion *Wertung()* berechnen, klären ob ein Stein eine gültige Position inne hat wenn er an eine bestimmte Position gesetzt werden soll (*isvalid()*). (Achtung! keine Überprüfung der KO-Regel, da hierbei auch die vergangene Spielentwicklung wichtig wäre).

Diese Klasse kann natürlich auch einen Stein auf einem Feld platzieren mit der Methode *turn()*, und zu guter Letzt kann ein Objekt mit der Methode *cmp()* überprüfen ob sein Argument, ein anderes Objekt der Klasse Feld von der Steinplatzierung identisch ist. Intern besitzt es die Funktion *kamikaze()* die prüft ob ein gesetzter Stein durch sein Setzen eine Gruppe der eigenen Farbe tötet (nicht regelkonformer Zug), sowie *hasneighbour()*, das prüft ob eine Gruppe von Feldern vollkommen umschlossen ist bzw. Augen besitzt. Ansonsten ist nur eine get und set Methode für den status vorhanden, also dafür, welcher bzw. ob ein Stein auf einem bestimmten Feld liegt (*getstatus()*, *setstatus()*).



Kapitel 4

Regelwerk von GO

4.1 Überblick

Dieses Spiel lehnt sich stark am japanischen Reglement an, mit einigen notwendigen bzw. sinnvollen Vereinfachungen. Für Kenner des Spieles die Kurzfassung: KOMI ist fix 2.5 Moku, Keine Handicap-Spiele, 13x13 Spielfeld, KO-Regelung siehe unten (nicht ING-kompatibel!).

Allgemeine Vereinfachungen: Kein Computergegner, eventuell in einer Ausbaustufe ein sehr einfacher (gültige Zufallszüge durch KI bzw. Züge mit maximalen Punktegewinn falls „bessere“ Züge Möglich sind(eventuell durch einen einfachen, rekursiven Algorithmus geringer Verzweigungstiefe).

4.2 Regelwerk im Detail

Auf einem Spielfeld mit 13x13 Kreuzungspunkten (Schachbrettmuster), setzen 2 Spieler abwechselnd einen Stein ihrer Farbe (schwarz und weiss, schwarz beginnt) auf ein ERLAUBTES FELD(s.u.). Wenn eine GRUPPE (s.u.) von gegnerischen Steinen durch Setzen dieses Steines eingeschlossen wurde wird diese vom Feld genommen. Ein Spieler kann auch auf seinen Zug verzichten oder keine gültige Zugmöglichkeit haben. Wenn beide Spieler hintereinander keinen Stein setzen (passen), endet das Spiel oder wenn ein Spieler aufgibt.

4.2.1 Erlaubtes Feld

Nur freie Kreuzungspunkte, durch diesen Zug darf es weder zur Wiederholung einer bereits im Laufe des Spieles vorgekommenen Stellung kommen („KO-REGEL“) noch darf der Stein so gelegt werden, dass seine \rightarrow Gruppe vom Gegner entfernt werden muss (Selbstmordverbot).

4.2.2 Gruppe

Jedes Feld im Zustand X (Farbe A/Leeres Feld) bildet mit den horizontal und vertikal anliegenden Feldern gleichen Zustands und deren anliegenden Feldern (u.s.w) eine GRUPPE. Diese Gruppe gilt als von Farbe B umschlossen wenn sie nur den Spielrand oder Steine der Farbe B als direkte Nachbarn vertikal oder horizontal besitzt. Dann spricht man auch davon das diese Gruppe keine „Augen“ besitzt. Wenn Eine Gruppe 2 Augen (2 freie Felder, die nicht zusammenhängen) besitzt, kann sie nicht mehr vom Gegner angegriffen werden, da er sie nie mehr vollständig umschliessen kann, ohne zuerst einen Stein auf ein Feld zu setzen, das ihm durch die Selbstmordsverbotregel verboten ist.

4.2.3 Wertung

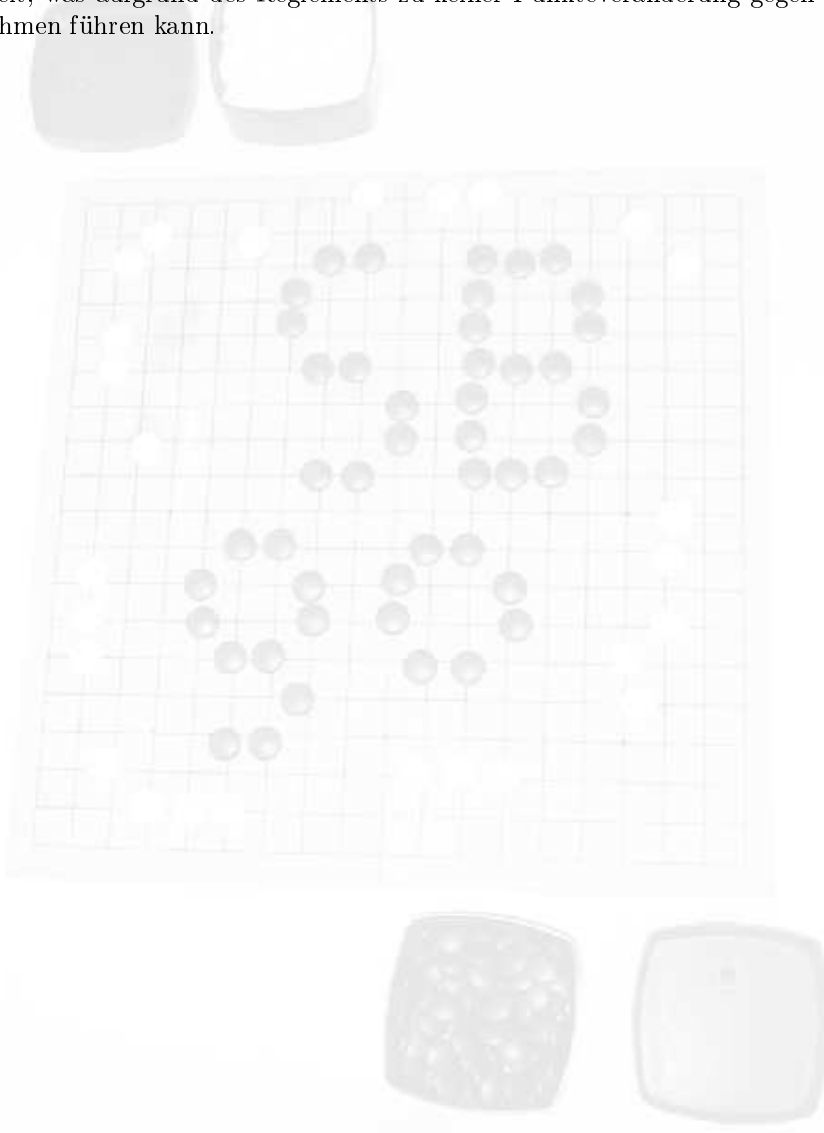
Für jeden Spieler wird die Anzahl der vom Feld genommenen gegnerischen Steine mit der Felderanzahl der umschlossenen Felder (Leere Felder die eine \rightarrow Gruppe bilden) zusammengezählt, weiss erhält noch

2.5 Punkte dazugezählt, um auszugleichen, dass Schwarz begonnen hat.

Wer mehr Punkte(Moku genannt) hat, gewinnt mit der Differenz.

Wenn ein Spieler aufgibt, gewinnt der Gegner ohne definierten Punktevorsprung.

Da für einen geübten Spieler bereits klar ist, wie gewisse Bereiche sich weiter entwickeln (ob der Gegner diese Gruppe umschliessen kann oder diese Gruppe 2 Augen bildet), wird das Spiel nicht bis zum bitteren Ende ausgespielt, sondern bei klaren Situationen wird in der Bewertungsphase jede Gruppe, die keine Überlebenschance hat, vom Brett genommen und als gefressen gezählt. Im realen Spiel einigen sich die beiden Spieler, welche Gruppen nicht überlebensfähig sind und bei Uneinigkeit wird die Situation ausgespielt, was aufgrund des Reglements zu keiner Punkteveränderung gegen über dem einfachen vom Brett nehmen führen kann.



Kapitel 5

Organisatorisches

5.1 Projekttagbuch - Zeitangaben in UTC (2006)

24.10. = Problemstellung wird von Dr. Hahsler genehmigt (gesehen als ASCII-Text.)

01.11. 21:00-23:15 = Arbeitsbeginn: JDK Version 1.5 zum Laufen gebracht und mit der Dokumentation begonnen nach dem anschauen bisheriger Projekte dieses Faches. Watermark (Dokumenthintergrund) erstellt und fotografiert.

06.11. 10:00-11:15 = Einarbeitung Problemstellung in TEX-Dokument, Einbindung von UML-Grafiken aus "dia"., Grobdesign der Klassen und ihrer Interaktion

06.11. 12:45-13:45 = Arbeit an GO-Klasse. Design von FELD und GO-Klasse. Implementierung Basisstrukture.

09.11. 00:00-01:30 = Arbeit an GO-Klasse; kann Zug auf Gültigkeit überprüfen und ihn durchführen. Kann Punkte für eine Farbe berechnen.

10.11. 18:30-22:00 = Arbeit am GUI-Quellcode. USE-Case Diagramm, Klassendiagramm tw.

11.11. 14:45-16:15 = Regeln/About-Menüeintrag; GO-Klasse fertig. Entfernt tote Steine, berechnet deren Anzahl. Erstellung der Symbole für die Steine und Grafik für den Bretthintergrund

13.11. 18:00-20:00 = Einarbeitung Java Swing-grafik

15.11. 22:30-00:45 = weitere Arbeit an GUI, ausbessern einiger sbGO-Fehler aus Testfeedback durch einen Freund

16.11. 07:45-08:15 = weiteres Fehlerausmerzen in sbGO

09.12. 16:50-22:57 = Wertungsphase, Programm bis auf Dokumentation fertig

11.12. 20:05-22:30 = Arbeit an Dokumentation

12.12. 07:03-10:21 = Fertigstellung Dokumentation(Sequence-Diagramme und Überarbeitung)

12.12. Vorgehenmigung des Prototypen und der Dokumentation durch Dr. Hahsler (Per Mail einsenden: JAR-File, Dokumentation als PDF-Datei und Quellcode)

01.01. 18:10-20:10 = Laden der Bilder aus JAR-Archiv, Punkteanzeige in Titelleiste, kleinen Bug bei Wertung behoben, Spielfarbe am Zug wird angezeigt

Gesamtaufwand: ca. 29 Arbeitsstunden exklusive Erarbeitung der Problemstellung.

5.2 Installation und Wartung

Als Java Anwendung benötigt dieses kleine Spiel ein JRE 1.5 oder kompatibles. Eventuell werde ich dieses Programm noch in meinem weiteren Leben verbessern, etwaige Updates werden dann via www.maze.at erreichbar sein, Anregungen, bemerkte Fehler etc. bitte per Mail an <Stefan.Beranek@maze.at>.



Abbildungsverzeichnis

2.1 Use-Case aus der Benutzersicht	9
2.2 Sequence-Diagramm für die Spielphase	10
2.3 Sequence-Diagramm für die Wertungsphase	10
3.1 Ein Screenshot zur Veranschaulichung	12
3.2 Die Klassen dieses Projekts	13