

Programmierpraktikum

Kalaha SS15

Paul-Louis Pröve
MInf 100941
4. Semester
B_MInf14.0

Table of Contents

PROGRAMMIERPRAKTIKUM.....	1
KALAHHA SS15.....	1
1 ALLGEMEINE PROBLEMSTELLUNG	4
1.1 ZIEL DES SPIELES	4
1.2 SPIELABLAUF	4
1.3 REGELN	4
1.4 AUFGABENSTELLUNG	5
1.5 WAS IST EIN SPIELBAUM?	5
1.6 WAS SOLL MEIN PROGRAMM TUN	7
1.7 GEWINNSTRATEGIE	7
1.8 BEDIENUNGSMÖGLICHKEITEN	8
1.9 LOG.....	9
1.10 DARSTELLUNG	10
1.11 PROGRAMMAUFTEILUNG	10
1.12 KLASSENVARIABLEN.....	10
1.13 DOKUMENTATION	10
1.14 ABGABETERMINE	10
2 BENUTZERHANDBUCH	11
2.1 ABLAUFBEDINGUNGEN	11
2.1.1 Windows	11
2.1.2 Linux.....	11
2.1.3 Mac OS X.....	11
2.2 PROGRAMMINSTALLATION.....	12
2.3 BEDIENUNGSANLEITUNG	12
2.3.1 Menü.....	12
2.3.2 Spielfeld	13
2.4 FEHLERMELDUNGEN.....	14
3 PROGRAMMIERHANDBUCH.....	15
3.1 ENTWICKLUNGSKONFIGURATION	15
3.2 PROBLEMANALYSE	15
3.2.1 Spielelogik.....	15
3.2.2 Künstliche Intelligenz	16
3.2.3 Steuerung.....	17
3.2.4 Benutzeroberfläche.....	17
3.3 REALISATION	18
3.3.1 Spielelogik.....	18
3.3.2 Künstliche Intelligenz	19
3.3.3 Steuerung.....	20
3.3.4 Oberfläche	20
3.4 BESCHREIBUNG GRUNDLEGENDER KLASSEN	22
3.4.1 Board	22
3.4.2 Tree	23
3.4.3 Game	23
3.4.4 Main.....	24
3.4.5 MenuUIController	24
3.4.6 GameUIController	25
3.4.7 Timer und TimerListener	25
3.5 CHECKLISTE.....	26

3.5.1 Spielelogik.....	26
3.5.2 Künstliche Intelligenz	27
3.5.3 Steuerung.....	27
3.5.4 Oberfläche	28
3.5 PROGRAMMIERORGANISATIONSPLAN	28
3.6 PROGRAMMTTESTS.....	29
3.6.1 User Input	29
3.6.2 Konsolenmodus.....	30
3.6.3 Log	30
3.6.4 Unit Tests	30
3.7 AUFWANDSAUSWERTUNG	32

1 Allgemeine Problemstellung

Kalaha ist ein Taktik- und Konzentrationsspiel für 2 Spieler.

Es wird mit 72 Bohnen auf einem Spielbrett mit 12 + 2 Mulden gespielt.

Das Alter dieses Spieles, auch Bohnenspiel genannt, ist nicht bekannt. Es steht jedoch fest, dass es auf verschiedene Weise gespielt wurde und außer in Europa auch in Afrika und Asien weite Verbreitung fand.

1.1 Ziel des Spieles

Ziel des Spieles ist es, durch vorausschauendes, kluges Verteilen der Bohnen auf die Mulden möglichst viele eigene und gegnerische Bohnen zu gewinnen und in seiner Gewinnmulde zu sammeln.

1.2 Spielablauf

Das Brett wird quer zwischen die beiden Spieler gelegt. In jede der 12 Mulden werden 6 Bohnen gefüllt. Die beiden Gewinnmulden an den Außenseiten bleiben zunächst leer.

Ein durch das Los bestimmter Spieler entnimmt aus einer beliebigen Mulde seiner Seite alle Bohnen und legt entgegen dem Uhrzeigersinn je eine Bohne in die folgenden Mulden, auch in die eigene Gewinnmulde. Hat der Spieler danach noch Bohnen in der Hand, verteilt er diese über seine Gewinnmulde hinaus auch in die Mulden des Gegners. In die Gewinnmulde des Gegners wird keine Bohne gelegt. Diese wird übersprungen.

1.3 Regeln

Fällt die letzte Bohne eines Spielers in die eigene Gewinnmulde, darf er noch einmal spielen, auch mehrmals, falls sich dies wiederholt.

Wird die letzte der zu verteilenden Bohnen in eine leere Mulde der eigenen Reihe gelegt, darf der Spieler diese Bohne und auch die aus der gegenüber liegenden Mulde des Gegners nehmen und sie direkt in seine Gewinnmulde legen. Ein solcher Fang beendet den Zug, der Gegenspieler ist an der Reihe.

Beim Fang spielt es keine Rolle, wie viele Bohnen in einem Zug gelegt wurden. Man kann z.B. eine Bohne, die einzeln in einer Mulde liegt in die angrenzende leere Mulde nach rechts legen. Oder mit vielen Bohnen einer prall gefüllten Mulde die eigenen und die ganze gegnerische Seite bestücken, um schließlich mit der letzten Bohne in der eigenen, leeren Mulde zu landen.

Das Spiel ist beendet, sobald ein Spieler alle 6 Mulden (außer der Gewinnmulde) vollständig geleert hat. Der Gegenspieler kann jetzt alle Bohnen, die sich noch auf seiner Seite befinden, in seine Gewinnmulde legen.

Sieger ist, wer die meisten Kalaha-Bohnen gesammelt hat.

1.4 Aufgabenstellung

Implementiert das Bohnenspiel Kalaha.

Ein Spieler soll gegen den Computer antreten können.

Der Spieler besitzt immer die unteren Mulden und der Computer die oberen.

Der Computer soll seine Zugentscheidung mit Hilfe eines Spielbaums fällen.

1.5 Was ist ein Spielbaum?

Zu Spielbeginn sind alle Mulden gleichmäßig gefüllt. Derjenige, der das Spiel eröffnet, kann eine beliebige seiner sechs Mulden leeren. Er hat also sechs verschiedene Zugmöglichkeiten. Wenn wir zunächst einmal die Möglichkeit außer Acht lassen, dass ein Spieler noch einmal dran sein könnte, hat danach auch sein Gegner bei seinem ersten Zug noch sechs Varianten offen. Im weiteren Spielverlauf kann es jedoch vorkommen, dass eine Mulde nicht geleert werden kann, weil sie schon leer ist. Darum hat der Spieler dann eine Möglichkeit weniger zur Auswahl.

Jede beliebige Spielsituation ist nur ein momentaner Zustand. Durch irgendeine Zug-Reihenfolge ist der aktuelle Spielstand entstanden und mit einem weiteren Zug wird dann der Zustand wieder verändert. Ein solcher Zustand wird als Knoten bezeichnet. Von einem Zustand in den nächsten kommt man, indem man einen Spielzug ausführt. Ein Zug verbindet also zwei Zustände. Deshalb wird ein Zug als Kante dargestellt.

Man kann alle möglichen Spielzüge eines Kalaha-Spiels aufzeichnen:

Der Anfangsknoten ist der Zustand, bei dem alle Mulden gleichmäßig gefüllt sind. Dieser Knoten heißt in der Fachsprache Wurzel. Wie wir oben gesehen haben, gibt es nun sechs Möglichkeiten, welche Mulde der beginnende Spieler leeren kann. Diese Möglichkeiten werden durch die sechs Kanten dargestellt, die von der Wurzel weggehen. Jede dieser Kanten führt zu einem Knoten in der zweiten Ebene. Diese Knoten sind also alle möglichen Zustände nach dem ersten Zug. Von jedem dieser Knoten aus geht das Spiel natürlich wieder weiter und für jeden möglichen Zug des zweiten Spielers ist wieder eine Kante eingezeichnet.

Das ganze Gebilde sieht jetzt wie ein auf den Kopf gestellter Baum aus. Und weil dieser Baum alle Spielmöglichkeiten darstellt, wird er Spielbaum genannt. (Zusatzinfo: Einen solchen Spielbaum gibt es aber nicht nur für Kalaha, sondern für jedes beliebige Zwei-Personen-Spiel mit perfekter Information, d.h. in jeder Spielsituation ist der Zustand für beide Spieler bekannt. Im Gegensatz dazu steht ein Kartenspiel, bei dem nicht bekannt ist, welche Karten der Gegner auf der Hand hat und welche noch im Stapel liegen.)

Wenn man nun einen frei wählbaren Weg entlang der Kanten nach unten geht, kommt man irgendwann zu einem Knoten, von dem keine Kanten mehr weitergehen. Einen solchen Endknoten bezeichnet man als Blatt. Ein Blatt im Baum entspricht also einem Zustand, nach dem nicht mehr weitergespielt werden kann und demzufolge das Spiel zu Ende ist.

Spieler B kann jetzt aus jeder seiner Mulden ziehen, mit keinem Zug erreicht er, ein zweites Mal dran zu sein. Der Inhalt seiner Gewinnmulde würde sich bei jedem Zug nur um eine Bohne erhöhen. Verfolgen wir den Zug aus seiner ersten Mulde weiter, so bleiben für A folgende Zugmöglichkeiten:

A setzt die Bohne aus der ersten Mulde in die zweite und fängt somit die eigene und die Bohnen aus der gegenüberliegenden Mulde. GewinnmuldeA - GewinnmuldeB ist jetzt 7. A kann nicht aus der zweiten Mulde ziehen, da diese leer ist.

Zieht A aus der dritten Mulde, so gewinnt er eine Bohne. GewinnmuldeA - Gewinnmulde B ist somit 2.

Sind wir jetzt an der Suchgrenze (hier Tiefe 3, d.h. der Spieler am Zug hat 3 Mal gewechselt,) angekommen, so kann die Bewertung nach oben gereicht werden:

A würde im letzten Zug die maximale Bewertung wählen, also die 7.

B würde bei seinem Zug eine minimale Bewertung wählen, also sicher nicht die 7, sondern einen der beiden anderen Züge mit der Bewertung 2.

A würde sich aus seinen beiden Zugmöglichkeiten denjenigen mit der größten Bewertung aussuchen.

Auch bei seinem ersten Zug würde A den Zug, der zur größten Bewertung führt, wählen.

1.6 Was soll mein Programm tun

Für die eindeutige Bezeichnung der Mulden (bei den folgenden Beschreibungen und auch bei Nachfragen in der Newsgroup) werden diese gegen den Uhrzeigersinn beginnend mit 0 bei der ersten Spielermulde links unten durchnummeriert. Die Gewinnmulde des Spielers ist also Mulde 6, die des Computers Mulde 13.

1.7 Gewinnstrategie

Zur Ermittlung der Gewinnstrategie soll ein Spielbaum erstellt werden. Der Baum soll nur bis zu einer vorgegebenen Tiefe erstellt werden (Anfänger: 1, Fortgeschrittener: 3, Profi: 5) [geändert am 3.9.15] (Anfänger: 1, Fortgeschrittener: 5, Profi: 10).

Die Rekursionstiefe soll dabei nur dekrementiert werden, wenn sich der Spieler ändert, d.h. bei einer Tiefe von eins wird beim Spielstart berücksichtigt, dass der erste Spieler beim Zug aus der ersten Mulde noch einmal dran ist und somit zwei Bohnen in seiner Gewinnmulde ergattern kann.

Führen mehrere Züge zu einer gleichen Bewertung, so soll die der Gewinnmulde am entferntesten liegende Mulde gewählt/empfohlen werden.

Der Spielbaum wird nur vor dem ersten Zug bis zur gewünschten Tiefe komplett aufgebaut. Bei den folgenden Zügen wird jeweils der nicht mehr benötigte Teil des Baumes verworfen und beim restlichen Baum eine Zugtiefe weiter erstellt. So muss zwar die Bewertung neu ermittelt, aber nur relativ wenige Knoten neu erstellt werden.

1.8 Bedienungsmöglichkeiten

Bedienung über grafische Oberfläche: Werden dem Programm keine Aufrufparameter übergeben, so startet eine grafische Oberfläche, die folgendermaßen bedient werden kann:

Mulde anklicken:

Während des Mausdrucks mit der linken Maustaste wird die Mulde hervorgehoben, in der die letzte Bohne landen wird.

Mulde doppelklicken:

Die Mulde wird geleert, bei ungültigen Mulden (Gewinnmulde/Gegnermulde/leere Mulde) erfolgt keinerlei Programmreaktion.

Optionen:

Schwierigkeitsgrad:

Der Schwierigkeitsgrad kann in einem Menü auf Anfänger/Fortgeschrittener/Profi eingestellt werden.

Darstellungsgeschwindigkeit:

Das Verteilen der Bohnen aus der geleerten Mulde soll nachvollziehbar dargestellt werden, d.h. die Werte der nacheinander aufgefüllten Mulden sollen nacheinander erhöht werden.

Die Verzögerung soll in drei Stufen einstellbar sein.

Hilfe:

Wenn die Hilfestellung an ist, so wird bei jedem Zug des Benutzers die Bewertung jeder seiner Mulden angezeigt und der daraus resultierende optimale Zug hervorgehoben.

Bedienung per Aufrufparameter: Werden dem Programm Aufrufparameter übergeben, so wird keine grafische Oberfläche angezeigt, sondern die Eingaben interpretiert und eine Auswertung auf der Konsole ausgegeben. [Präzisierung vom 11.9.15:] Der Computer soll hierbei im fortgeschrittenen Modus agieren, also soll der Baum mit der Rekursionstiefe 3 aufgebaut werden.

Eingabeparameter:

Die Nummern der durch den Benutzer geleerten Mulden werden kommasepariert angegeben.

Beispiel: "0,1,5" Der Benutzer leert die Mulde 0, da die letzte Bohne in der Gewinnmulde landet, leert er unmittelbar danach Mulde 1. Dann würde der Computer ziehen und anschließend der Benutzer Mulde 5 leeren.

Ausgabe:

Es wird der erreichte Spielzustand in Form der Muldenfüllungen kommasepariert ausgegeben, der nach Durchführen der angegebenen Benutzerspielzüge und des nachfolgenden Computerzuges herrscht. Erfolgte als Eingabeparameter eine ungültige Eingabe, z.B. eine nicht vorhandene Muldennummer oder eine (inzwischen) leere Mulde, so wird eine einfache Fehlermeldung ausgegeben.

Beispiel 1: Wurde als Eingabeparameter "5" eingegeben, so leert der Computer in Folge Mulde 7 und der ausgegebene Spielstand wäre "7,6,6,6,6,0,1,0,8,8,8,7,1".

Beispiel 2: Wurde als Eingabeparameter "0" eingegeben, so erfolgt die Ausgabe "0,7,7,7,7,7,1,6,6,6,6,6,0". Da der Benutzer noch einmal am Zug wäre, erfolgt hier kein anschließender Computerzug.

Beispiel 3: Der Eingabeparameter "0,0" oder "0,14" führt zu einer Fehlermeldung.

1.9 Log

Mit jedem Spiel wird eine Logdatei mit festem Namen im selben Verzeichnis wie das ausführbare Programm erstellt bzw. überschrieben. Kann die Datei in diesem Verzeichnis weder überschrieben noch erstellt werden, ist dies mit einer Fehlermeldung anzuzeigen und das Spiel erfolgt ohne Loggen. Jeder Spielzug ist in dieser Datei mit dem aktuellen Spielbaum zu protokollieren.

Dafür werden nacheinander die Zustände protokolliert. Ein Zustand beginnt mit einer Zeile mit der Angabe, welche Mulde geleert wurde. hier steht also eine Zahl zwischen 0 und 5 oder zwischen 7 und 12. Zu Spielbeginn ist diese Zeile leer. Es folgt eine Abbildung des aktuellen Spielbaums. Ein Zustand endet mit einer Zeile mit 10 Bindestrichen.

Der Spielbaum wird folgendermaßen dargestellt: jeder Knoten wird in einer Zeile abgebildet durch eine Nummerierung, den Spieler, der am Zug ist, die Bewertung des Knotens und die Inhalte der Mulden in Reihenfolge der Nummerierung.

Ist ein Zug nicht möglich, da die Mulde leer ist, ist an dieser Stelle kein Knoten entstanden und es wird eine nummerierte Leerzeile dargestellt.

Ausschließlich der Wurzelknoten besitzt keine Nummerierung. Es folgen sechs Knoten, die von 0 bis 5 durchnummeriert und mit einem folgenden Punkt dargestellt werden. Unterhalb jedes Knotens erhält die Nummerierung mit jeder abgelaufenen Kante eine weitere Stufe.

Passend zum obigen Beispielspielbaum entstünde also folgendes Log, wobei ein Unterstrich eine im Beispiel nicht angezeigte Bewertungszahl angibt und die drei Punkte auf nicht weiter ausgeführte Knoten hindeuten:

```
A _ 3 3 3 0 3 3 3 0
0. A _ 0 4 4 1 3 3 3 0
0.0.
0.1. B 2 0 0 5 2 4 4 3 0
0.1.0. A 7 1 0 5 2 0 5 4 1
0.1.0.0. B 7 0 0 5 8 0 0 4 1
0.1.0.1.
0.1.0.2. B 2 2 0 0 3 1 6 5 1
0.1.1. A 2 1 1 5 2 4 0 4 1
0.1.1.0. B 1 ...
0.1.1.1. B 1 ...
0.1.1.2. B 2 ...
0.1.2. A 2 1 1 5 2 4 4 0 1
0.1.2.0. B 1 ...
0.1.2.1. B 1 ...
0.1.2.2. B 2 ...
0.2. ...
...
```

1.10 Darstellung

Zur Darstellung des Spielbrettes ist mindestens ein GridPane mit 14 Buttons notwendig, die die Mulden repräsentieren. Die Anzahl der enthaltenen Bohnen kann einfach über den ButtonText angezeigt werden. Schöner ist die [grafische Anzeige](#) der enthaltenen Bohnen bis zu einer bestimmten Anzahl Bohnen und erst bei Übersteigen einer übersichtlichen Anzahl die Verwendung von Zahlen.

1.11 Programmaufteilung

Es ist darauf zu achten, dass zwischen Oberfläche und Programmlogik sauber getrennt wird und dass im FXML-Controller nur Routinen zu finden sind, die etwas mit der direkten Oberflächensteuerung zu tun haben. Alle anderen Routinen sind in entsprechende andere Klassen auszulagern.

1.12 Klassenvariablen

Versucht die Verwendung von Klassenvariablen so weit wie möglich einzuschränken. Und versucht, wenn Ihr Klassenvariablen deklariert habt, so wenig wie möglich aus anderen Klassen auf diese Variablen zuzugreifen (auch nicht über Getter), sondern sie stattdessen an die entsprechenden Prozeduren und Funktionen als Parameter zu übergeben.

1.13 Dokumentation

Die Dokumentation ist ein Grundbestandteil der Lösung. Bei der Beschreibung von Problemanalyse und -Realisation orientiert Euch an dem Gedanken, was ein nachfolgender Entwickler von Eurem Programm kennen muss, um z.B. das Programm um die Funktionalität zu erweitern, dass ein Spielstand gespeichert und wieder aufgenommen werden kann.

1.14 Abgabetermine

Termine für die Zwischenpräsentation werden zwischen dem 12.10. und 23.10.2015 vergeben, Sondervereinbarungen für davor liegende Termine sind möglich. Die Repositories werden am 27.2.2016 um 12.00 Uhr geschlossen.

2 Benutzerhandbuch

2.1 Ablaufbedingungen

Um KalahaGo ausführen zu können, muss das Java 8 Runtime Environment installiert sein. Die Systemanforderungen für Java 8 und JavaFX müssen erfüllt werden und das auszuführende System benötigt eine Displayauflösung von mindestens 700x300 Pixel. Die genauen Anforderungen können den folgenden Listen entnommen werden.

2.1.1 Windows

Windows 10 (8u51 und höher)

Windows 8.x (Desktop)

Windows 7 SP1

Windows Vista SP2

Windows Server 2008 R2 SP1 (64-Bit)

Windows Server 2012 und 2012 R2 (64-Bit)

RAM: 128 MB

Datenträgerkapazität: 124 MB für JRE; 2 MB für Java Update

Prozessor: Mindestens Pentium 2 266 MHz-Prozessor

Ausflösung: 700x300 Pixel

2.1.2 Linux

Oracle Linux 6.x (32-Bit), 6.x (64-Bit)

Oracle Linux 7.x (64-Bit) (8u20 und höher)

Red Hat Enterprise Linux 6.x (32-Bit), 6.x (64-Bit)

Red Hat Enterprise Linux 7.x (64-Bit) (8u20 und höher)

Suse Linux Enterprise Server 10 SP2+, 11.x

Suse Linux Enterprise Server 12.x (64-Bit) (8u31 und höher)

Ubuntu Linux 12.04 LTS, 13.x

Ubuntu Linux 14.x (8u25 und höher)

Ausflösung: 700x300 Pixel

2.1.3 Mac OS X

Intel-basierter Mac unter Mac OS X 10.8.3+, 10.9+

Administratorberechtigungen für die Installation

Ein 64-Bit-Browser (Beispiele: Safari, Firefox) ist zur Ausführung von Oracle Java auf Mac OS X erforderlich.

Ausflösung: 700x300 Pixel

2.2 Programminstallation

Für KalahaGo ist keine Installation notwendig. Man muss lediglich die KalahaGo.jar Datei über die Kommandozeile starten. Dies funktioniert wie folgt:

1. Öffnen Sie cmd.exe unter Windows bzw. den Terminal unter OS X oder Linux
2. Navigieren Sie zum Ordner, welcher die KalahaGo.jar Datei enthält über den Befehl 'cd ORDNERPFAD'
3. Führen Sie folgenden Befehl aus: 'java -jar KalahaGo.jar'
4. Das Spiel startet in einem neuen Fenster

Sollte das Spiel nicht starten, stellen Sie sicher, dass die neuste Version von Java auf dem System installiert ist und nicht noch ältere Versionen von Java Verwendung finden.

2.3 Bedienungsanleitung

Im folgenden werden Sie durch Bedienelemente von KalahaGo geführt.

2.3.1 Menü

Nach dem Start des Spiels öffnet sich das Spielmenü, in welchem verschiedene Einstellungen zum Spiel getroffen werden können. Grün eingefärbte Knöpfe zeigen an, welche der Einstellungen zur Zeit ausgewählt sind.



- **Difficulty:** Legt den Schwierigkeitsgrad fest, den der computergesteuerte Gegenspieler übernimmt. Standardwert ist "Medium". Die Schwierigkeitsstufe "Hard" kann auf langsamen Computersystemen zu Verzögerungen und Wartezeiten führen.
- **Number of Stones:** Legt die Anzahl der Steine pro Spielmulde fest. Standardwert ist "6" gemäß der regulären Spielregeln von Kalaha.

- **Who starts the Game?:** Legt fest, wer den ersten Zug ausführen darf. "Human" repräsentiert den Benutzer, "Robot" repräsentiert den Computergegner und bei "???", wird per Zufall entschieden, wer beginnt. Standardwert ist "???".
- **Animation Speed:** Legt die Geschwindigkeit der Zuanimationen fest. Standardwert ist "Medium".

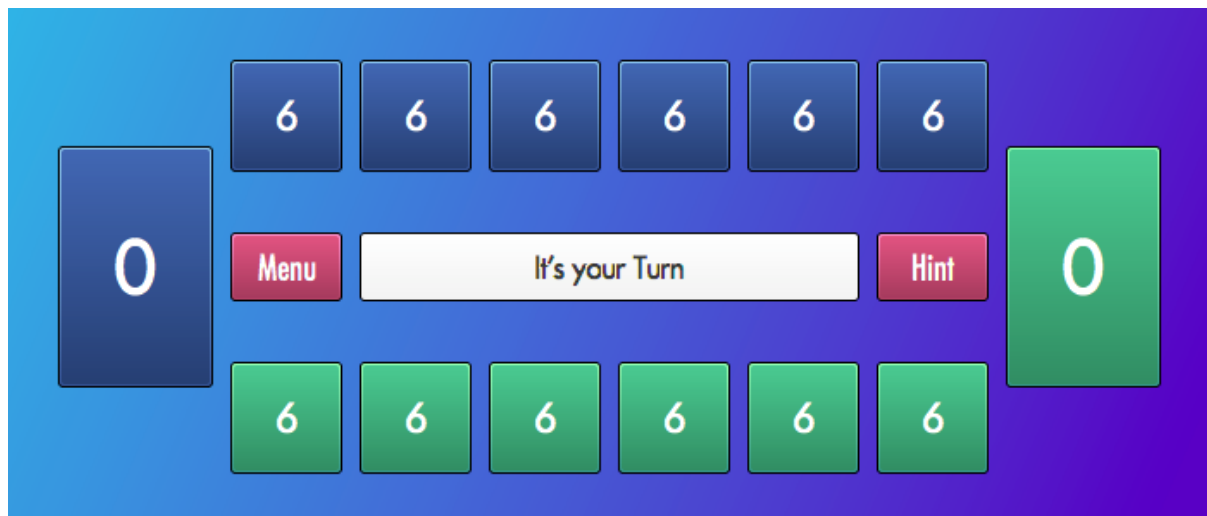
Der **Start**-Knopf startet das Spiel unter den definierten Einstellungen.

Bitte beachten Sie, dass während des Spielens im Verzeichnis von KalahaGo.jar eine Datei mit dem Namen "KalahaGo_Log.txt" angelegt wird. Diese Datei ist nur relevant, wenn Sie sich wegen Problemen zum Spielgeschehen an unseren Support wenden möchten. Die Datei kann je nach Schwierigkeitsstufe mehrere hundert MB groß werden.

2.3.2 Spielfeld

Über den Start-Knopf gelangt man zum Spielfeld von KalahaGo.

- Die aus Kalaha bekannten Mulden werden in KalahaGo als Knöpfe repräsentiert.
- Die aus Kalaha bekannten Bohnen oder Steine werden in KalahaGo als Zahlen repräsentiert, um eine schnelle Analyse für Spielzüge treffen zu können.



- Die untere **grüne Reihe** zeigt die eigenen Spielmulden
- Die obere **blaue Reihe** zeigt die gegnerischen Spielmulden
- In der Mitte des Fensters ist ein **Statusfeld** sichtbar, welches den Spieler über relevante Spielzustände informiert.
- Der **Menu-Knopf** bricht das aktuelle Spiel ab und bringt den Benutzer zurück in das Hauptmenü.
- Der **Hint-Knopf** aktiviert den Hilfestellungsmodus und ist standardmäßig ausgeschaltet. Ist der Knopf grün, so ist der Modus aktiviert. Bewegt der Spieler die Maus über seine eigenen Spielmulden bekommt er im Hilfestellungsmodus eine Bewertung der entsprechenden Mulden im Statusfeld angezeigt. Der bestmögliche Zug wird in rot hervorgehoben.

Klickt man eine Spielmulde an und hält die Maustaste gedrückt, wird durch eine rote Markierung sichtbar, in welcher Mulde die letzte Bohne landen würde.

Ein Spielzug kann per Doppelklick auf die entsprechende Mulde ausgeführt werden. Die benötigte Doppelklick-Geschwindigkeit orientiert sich dabei an den Systemeinstellungen des Betriebssystems.

Bitte beachten Sie, dass die Nutzung der Spiel-Knöpfe während der Animation von Spielzügen nicht möglich ist. Ausschließlich der Menü-Knopf ist jederzeit ansprechbar und bricht das aktuelle Spiel ab.

2.4 Fehlermeldungen

Es gibt in KalahaGo keine Fehlermeldungen über die der Nutzer in der Oberfläche informiert wird. Relevante Informationen zum aktuellen Spielgeschehen werden im Statusfeld des Spielbretts angezeigt.

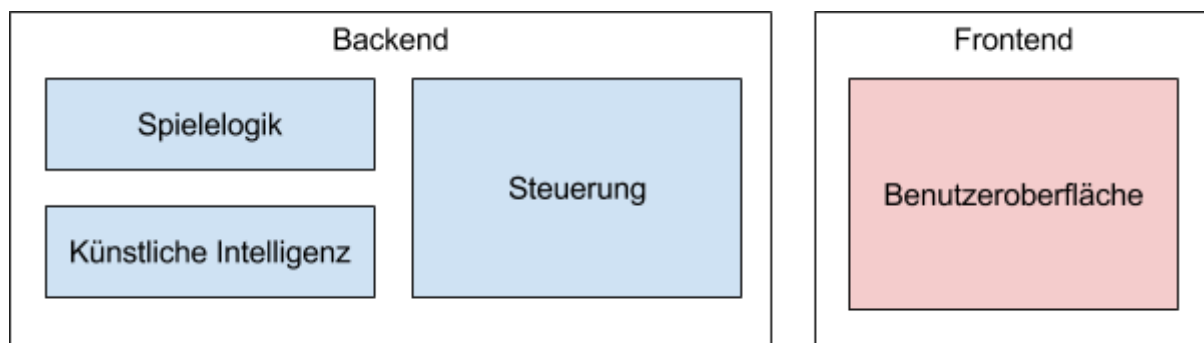
3 Programmierhandbuch

3.1 Entwicklungskonfiguration

Prozessor RAM Grafik Festplatte	3,5GHz Intel Core i7 16GB 1333MHz DDR3 NVIDIA GeForce GTX 650 Ti Samsung SSD 850
Betriebssystem Javaversion IDE	OS X El Capitan (10.11.3) Java SE 8 Update 73 IntelliJ IDEA 14.x, 15.0.x / Netbeans 8.0.1, 8.1

3.2 Problemanalyse

Zur Problemanalyse gehören 4 verschiedene Komponenten, die das Spiel Kalaha in der formulierten Aufgabenstellung ausmachen. Hier werden die größten Herausforderungen vor der Implementationsphase beschrieben.



3.2.1 Spiellogik

Die Spiellogik beinhaltet die Regeln, Umstände und Aktionen, welche in Kalaha zwischen zwei Spielern möglich sind. Herausgetrennt aus den anderen Bereichen könnte man sagen, dass die Spiellogik allein das Spielen zweier Spieler, welche aktive Entscheidungen treffen, ohne visuelle Elemente ermöglicht. Folgende Objekte aus der Realität spielen dabei eine tragende Rolle:

- **Steine:** Die Steine oder auch Bohnen, welche in Zügen verteilt werden.
- **Mulden:** Die Mulden in denen die Steine abgelegt werden. Differenzierung: Eigene und gegnerische Spielmulden, sowie die eigene und gegnerische Gewinnmulde.
- **Spielbrett:** Das Spielbrett fasst die Mulden zusammen in welchem dann die Steine abgelegt werden können.

- **Aktiver Spieler:** Der aktive Spieler ist für viele der Regeln in Kalaha relevant und die Information wer gerade mit dem Zug dran ist, muss gespeichert werden.

Daraus ergaben sich verschiedene zu erledigende Dinge:

- Die Hauptproblematik bei einem Spielzug ist, dass hier viele Aktionen, Zustände und Nebenregeln von Kalaha beachtet werden müssen, so dass der Code schnell unübersichtlich erscheinen kann.
- Es soll möglich sein, die erwähnten Eigenschaften flexibel verändern zu können, um verschiedene Spielbrettgrößen oder Steinanzahlen umzusetzen.
- Ein Spielbrett muss kopierbar sein, um mögliche Änderungen ausschließlich auf andere Instanzen des Spielbretts umsetzen zu können.
- Nebenregeln, wie das Capture müssen mit allen Feinheiten beachtet werden.
- Der Verteilung der Steine auf die Mulden variiert je nachdem welcher Spieler am Zug ist

3.2.2 Künstliche Intelligenz

Die künstliche Intelligenz oder KI, soll ermöglichen, dass auf Basis von zukünftigen Spielzuständen automatisch der beste Zug gewählt wird. Herausgetrennt aus den anderen Teilen könnte man sagen, dass die KI die intelligente Entscheidungsfindung des Gegenspielers darstellt. Zusammen mit der Spielelogik könnte man so ohne eine visuelle Darstellung Kalaha gegen einen computergesteuerten Gegenspieler spielen.

- Die Entscheidungsfindung der KI basiert auf einem Spielbaum, der jeden möglichen Zug über eine spezifizierte Tiefe berechnet. Es muss der Minimax-Algorithmus implementiert werden, der darauf basiert, dass jeder Spieler immer die bestmögliche Entscheidung fällt.
- Der Spielbaum wächst mit jeder Ebene um den Faktor 6, weshalb eine effiziente Implementierung eine extrem hohe Priorität hat und sich durch alle Klassen ziehen muss. Besonders, weil Ebenen laut Aufgabenstellung nicht unbedingt mit den tatsächlichen Ebenen des Baums zu vergleichen sind. Eine Spielebene gilt dann als abgeschlossen, wenn ein Spielerwechsel stattgefunden hat. Da es bei Kalaha nicht selten vorkommt, dass ein Spieler mehrmals hintereinander am Zug ist, kann eine maximale Tiefe von 5 schnell zu einer fünfstelligen Anzahl an Baumknoten führen.
- Die Tiefe des Spielbaums muss variabel implementiert werden, um die drei vorgegebenen Schwierigkeitsstufen (1, 3 und 5) realisieren zu können.
- Es stellt sich die Frage ob es empfehlenswert ist, den Spielbaum mit einer Empty/Node-Struktur aufzubauen.

3.2.3 Steuerung

Die Steuerung ist für die Programminitialisierung verantwortlich und adaptiert weiter unten liegende Logiken im Kontext auf die ursprüngliche Problemstellung. Im Bezug auf Kalaha sind hier folgende Probleme zu lösen:

- Starten der Applikation
- Wrappen von Methoden dessen Funktionalität abstrakt zum Nutzungskontext erscheinen
- Verwaltung von Exceptions
- Schnittstelle für den Spielstart bieten, um Einstellungen entgegenzunehmen
- Loggen eines Spiels
- Spielablauf durch Eingabeparameter

Außerdem muss auch hier die Logik für die Animation der Spielzüge stattfinden, welche dann auf der Oberfläche nur noch aktualisiert wird. Es ist wichtig, dass die Zeit während der Animation genutzt werden soll, um den nächsten Zug bereits zu berechnen. So kann man Ladezeiten vermeiden, auf die der Nutzer warten müsste.

3.2.4 Benutzeroberfläche

Die Benutzeroberfläche stellt die Interaktionsschnittstelle des Nutzers dar und visualisiert das Spielgeschehens des Backends. Folgende Interaktionen sollen möglich sein:

- Zugausführung durch Doppelklick
- Anzeige der Mulde in welche der letzte Stein eines Zuges fällt (durch Klick und halten)
- Anzeige der Bewertung eines Zuges auf Basis des Spielbaums
- Anzeige des bestmöglichen Zugs
- Aktualisierung der Animationsschritte
- Anzeige korrekter und aktueller Steinanzahl pro Mulde

Neben diesen Aktionen gab es ebenfalls vorgegebene Einstellungsmöglichkeiten:

- Schwierigkeitsgrad des Computergegners
- Animationsgeschwindigkeit mit drei Einstellungen
- Option zum Aktivieren/Deaktivieren der Bewertungsanzeige

Die Benutzeroberfläche sollte außerdem spielend und modern aussehen, um den Charakter eines Computerspiels zu unterstreichen. Ein auftretendes Problem war die Benutzung von Schriftarten, welche nicht auf jedem Computer vorinstalliert sind und trotzdem überall funktionieren sollten.

Neben der Interaktion über ein User Interface, soll ebenfalls die Eingabe von Zugkombinationen über die Java-Eingabeparameter möglich sein. Dies ist zwar auch eine Interaktion mit dem Nutzer, aber stellt eine direkte Anbindung an das Backend dar und hat wenig mit der Benutzeroberfläche zu tun.

3.3 Realisation

In der Realisation wird beschrieben, wie die größten Herausforderungen und Probleme aus der Problemanalyse zu beheben waren. Die Unterkapitel orientieren sich an der gleichen 4-Komponenten Struktur wie in der Problemanalyse.

3.3.1 Spielelogik

Auch bei der Spielelogik musste auf Effizienz Wert gelegt werden, damit die hohe Anzahl von berechneten Spielfelder tragbar ist. Die Änderung von einem Int- zu einem Byte-Array viertelte die Speicherauslastung und halbierte die Laufzeit in den Tests. Durch dieses Vorhaben ist die Gesamtanzahl der Steine im Spiel auf insgesamt 127 beschränkt, weshalb größere Werte mit entsprechenden asserts abgefangen werden. Hierzu erfolgen keine weiteren Fehlerausgaben.

Die einzelnen Aktionselemente von Kalaha ließen sich in heruntergebrochene Funktionen sauber strukturieren und mehrmals verwenden. Die Zugausführungsmethode kann so, trotz ihrer Komplexität, gut gelesen werden.

Die hier implementierte Logik orientiert sich an der "Empty Capture" Variante, bei der es keine Rolle spielt, wie viele Steine in der gegenüberliegenden Mulde liegen. Eine Erweiterung der entsprechenden Bedingung liegt dem Quelltext als Kommentar bei, um die Funktionalität schnell ändern zu können.

Durch die Verwendung von Mulden-Indices auf Basis von verschiedenen Arraylängen konnte das Spielfeld sehr flexibel umgesetzt werden. Die entsprechenden Einstellungen werden im Konstruktor übergeben.

Die korrekte Angabe der nächsten zu verteilenden Mulde ließ sich über die Hilfsmethode `nextPit(int i)` lösen, welche auf Basis des aktiven Spielers die nächste Mulde ausgibt. Nach jedem Zug wird geprüft ob das Spiel vorbei ist und die letzten Steine der anderen Spielers eingesammelt werden müssen.

3.3.2 Künstliche Intelligenz

Durch das exponentielle Wachstum an Baumknoten war eine Empty/Node Struktur keine empfehlenswerte Wahl. Bei jedem Zug hätte man für jedes Blatt im Baum 6 weitere Konstruktoraufrufe ausführen müssen. Dafür hätte man im Gegenzug keine Abfragen auf null gebraucht. Die Anzahl der Abfragen auf null sind allerdings 6 mal geringer als die Konstruktor Aufrufe, die man benötigt hätte.

Nimmt man zum Beispiel einen Baum mit 4 Ebenen, der auf eine 5. Ebene ausgebaut werden soll, so würden bei der null-Variante 7776 Abfragen auf null ausgeführt werden. Bei der Empty/Node-Variante wäre diese Abfrage überflüssig, aber dafür hätte man 46656 Empty-Konstruktor Aufrufe für das gleiche Ergebnis. Auch beim Logging hat man so zwar 7776 Abfragen mehr, aber dafür 46656 Methodenaufrufe weniger. Es war deshalb eine leichte Entscheidung mit der null-Variante fortzufahren.

Insgesamt gibt es drei Funktionen im Baum, die eine rekursive Umsetzung verlangten und mindestens mit einer $O(n)$ Laufzeit arbeiten würden.

- Das Aufbauen des Baums
- Das Bestimmen der Zugbewertungen
- Das Schreiben des Logs

Die Reihenfolge der drei Funktionen musste auch genau in dieser Reihenfolge passieren, weil eine korrekte Zugbewertung einen vollständig aufgebauten Baum voraussetzte und auch das Loggen wiederum die Zugbewertungen benötigte. Daraus resultierte eine Komplexität von $O(3n)$, wobei n die Anzahl der Elemente im Baum darstellt.

Durch etwas Aufwand war es möglich diesen Wert auf $O(2n)$ zu reduzieren, weil man das Aufbauen des Baums und das Auswerten der Bewertungen kombinieren konnte. Auf dem rekursiven Weg nach unten wird der Baum aufgebaut und auf dem Weg zurück nach oben werden durch Rückgabewerte die Bewertungen eingetragen. Das Zusammenlegen resultierte in einer weniger klaren Unterteilung an Funktionen, aber die Laufzeit verbesserte sich um 40%, was in diesem Ausmaß höhere Priorität hatte.

Die daraus resultierende buildTree()-Methode bekommt als Parameter die gewünschte Tiefe des Baums mitgegeben, um flexibel verschiedene Tiefen zu realisieren. Sie bestimmt pro Baumebene das vorhandene Minimum oder Maximum, je nachdem welcher Spieler am Zug ist und leitet die Bewertung nach oben weiter.

Es war ohne erheblichen Aufwand nicht möglich ebenfalls das Loggen in diese Laufzeit zu integrieren, da dieses Vorhaben rekursiv von oben nach unten arbeitet. Zu diesem Zeitpunkt stehen aber die Bewertungen noch nicht und es wäre nur durch komplexe Stringmodifikationen möglich dieses Problem zu umgehen.

3.3.3 Steuerung

JavaFX stellt die Start-Methode zur Verfügung die bei Programmausführung ein entsprechendes GUI startet. Es muss deshalb vorher in der Main-Methode eine Unterscheidung passieren, falls Kalaha nur über die Konsole gespielt werden soll. Dieses Vorhaben lässt sich über die Differenzierung der Anzahl der Eingabeparameter einfach umsetzen.

Einstellungen müssen über einen Konstruktor gesammelt entgegengenommen werden und auf die einzelnen Unterbereiche des Backend verteilt werden, wofür der Konstruktor der Game-Klasse zuständig ist.

Die rekursive Funktion des Logs passiert zwar im Spielbaum, aber das Erstellen der Datei und das damit verbundene Exception-Handling passiert auf der Steuerungsebene. Hier war es wichtig den Log zugweise zu schreiben und nicht einmal am Ende, damit bei Problemen oder Abstürzen der Spielverlauf bis hier hin nachvollziehbar ist. Die Syntax des Logs lässt sich durch folgende Backus-Naur-Form nachvollziehen:

```
<Logdatei> ::= <Ausgeführter Zug> <Spielbaum> <Trennzeichen>
<Ausgeführter Zug> ::= <Muldenindex> <EOL>
<Muldenindex> ::= <EOL> | 0 | 1 | 2 | 3 | 4 | 5 | 7 | 8 | 9 | 10 | 11 | 12
<Spielbaum> ::= <Spielbrett> {<Spielbrett>}
<Spielbrett> ::= <Knoten> <Spieler> <Differenz> <Muldenbelegung> <EOL>
<Knoten> ::= {( 0 | 1 | 2 | 3 | 4 | 5 ) '}'
<Spieler> ::= 'A' | 'B'
<Differenz> ::= <Ganze Zahl>
<Muldenbelegung> ::= { <Ganze positive Zahl> }14
<Trennzeichen> ::= '-----' <EOL>
```

Das Ausführen eines Zuges wird aus der Spielelogik gewrappt und um Funktionalitäten ergänzt, die außerhalb dieser Ebene liegen. Dazu gehören die Log-Ergänzung für den Zug, die Zuweisung im Spielbaum und das weitere Ausbauen des Baums.

Die Steuerung kümmert sich ebenfalls um die Verarbeitung des Konsolenmodus von Kalaha. Hier war die akzeptierte Syntax fest vorgegeben:

```
<Konsoleneingabe> ::= <Mulden-Index> { ' ' <Mulden-Index> }
<Mulden-Index> ::= 0 | 1 | 2 | 3 | 4 | 5
```

3.3.4 Oberfläche

Bei der Oberfläche war es sinnvoll die klare Unterscheidung in Interaktionen und Einstellungen auch im UI widerzuspiegeln. Der Benutzer sollte nach dem Start von einem Menü begrüßt werden, bevor das richtige Spiel mit dem entsprechenden Spielfeld startet.

Einzig beim Hilfemodus für den menschlichen Spieler erschien es sinnvoll, diese bei der Laufzeit ein- und ausschalten zu können.



Die zwei vorgegebenen Einstellungen für die Animationsgeschwindigkeit und den Schwierigkeitsgrad wurden außerdem um eine variable Steinanzahl und die Angabe wer das Spiel beginnt, erweitert.

Mit Hilfe des JavaFX Scene Builders wurden als erstes grobe Entwürfe für das Menü und das Spielfeld erstellt. Durch die Einbindung von CSS Stylesheets konnte das generische Aussehen, um bunte UI Elemente erweitert werden.



Die 13 Schriftarten, welche man auf jedem Betriebssystem erwarten kann, wirkten etwas zu klassisch für ein Computerspiel, welches modern und visuell ansprechend aussehen soll. Im Scene Builder ließen sich zwar ganz einfach auch andere Schriften auswählen, aber diese funktionierten nicht auf Systemen, welche diese Schriftart nicht installiert hatten. Die beiden benutzten Schriften "Futura Medium" und "Futura Condensed Medium" wurden deshalb mit in die ausführbare Spieldatei gebunden und mussten von der main-Methode geladen werden. Auch das KalahaGo-Logo und -Icon findet man eingebunden in die jar-Datei.

Neben den FXML Dateien für das Menü und das Spielfeld generierte der Scene Builder die entsprechenden Controller für beide GUIs. In diesen werden die Interaktionen des Nutzers in Funktionen der Programmlogik umgewandelt.

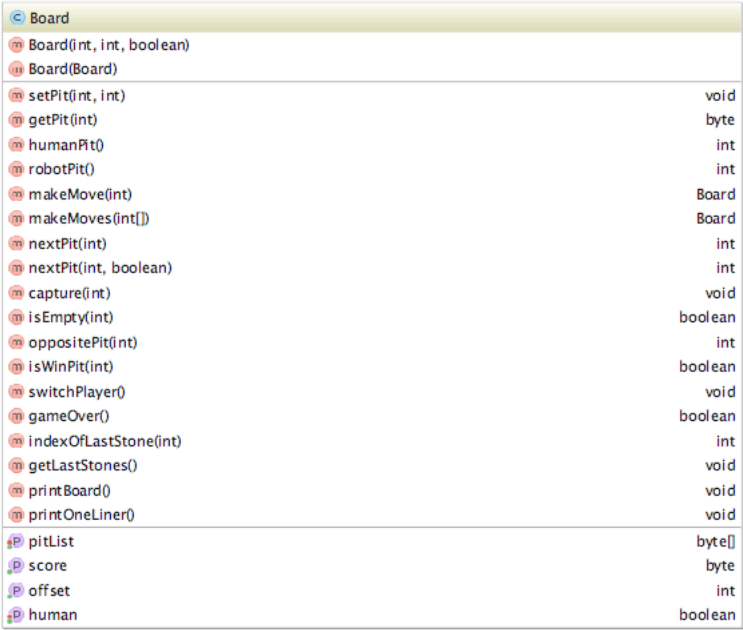
Da es bei Kalaha verschiedene Aktionen für bestimmte Maus-Verhalten geben sollte, mussten die ActionEvents gegen verschiedene MouseEvents ausgetauscht werden. Click, Press, Release, Enter und Exist boten die nötigen Eigenschaften, um alle gewünschten Interaktionen umsetzen zu können.

3.4 Beschreibung grundlegender Klassen

Im folgenden werden die elementaren Klassen von KalahaGo beschrieben und einzelne Attribute erklärt. Eine detaillierte Beschreibung der einzelnen Methoden ist über die JavaDoc Kommentare im Quellcode möglich.

3.4.1 Board

Die Klasse Board repräsentiert das Spielbrett. Sie speichert Variablen für die damit verbundenen Elemente und beinhaltet die gesamte Spielelogik von Kalaha.



Board	
Board(int, int, boolean)	
Board(Board)	
setPit(int, int)	void
getPit(int)	byte
humanPit()	int
robotPit()	int
makeMove(int)	Board
makeMoves(int[])	Board
nextPit(int)	int
nextPit(int, boolean)	int
capture(int)	void
isEmpty(int)	boolean
oppositePit(int)	int
isWinPit(int)	boolean
switchPlayer()	void
gameOver()	boolean
indexOfLastStone(int)	int
getLastStones()	void
printBoard()	void
printOneLiner()	void
pitList	byte[]
score	byte
offset	int
human	boolean

- **byte pitList[]** - Jeder Index repräsentiert eine Mulde. Jeder Inhalt die dort hinterlegten Steine.
- **boolean human** - Definiert wer gerade am Zug ist: true = Mensch, false = Roboter

Die Klasse beinhaltet einen Copy-Konstruktor der ein übergebenes Board kopiert und zurückgibt. Damit kann man Javas "Call-by-Value where the value is a reference"-Verhalten bei Objekten umgehen, um Board-Änderungen ausschließlich auf andere Instanzen zu ermöglichen.

Die Implementierung ist komplett flexibel zur Stein- und Mulden-Anzahl mit der bereits erwähnten Byte-Umsetzung.

3.4.2 Tree

Die Klasse Tree repräsentiert einen einzelnen Baumknoten.

C Tree	
(m) Tree(int, int, int, boolean)	
(m) Tree(Board, int)	
(m) setNextElement(int, Tree)	void
(m) buildTree(int)	byte
(m) betterScore(byte, byte)	byte
(m) playersSwitched(int)	boolean
(m) printNode(Writer, String)	void
(p) next	Tree[]
(p) rating	byte
(p) indexOfBestScore	int
(p) board	Board
(p) bottom	boolean

- **Board board** - Ein einzelnes Spielbrett das einen momentanen Spielaufbau speichert
- **Tree[] next** - Ein Array aus Tree-Elementen, das die nächsten möglichen Züge speichert
- **byte Rating** - Die Bewertung dieses Spielzugs, welches sich aus der untersten Ebene und den dadurch hochgereichten Wert errechnet.

Der Spieler soll während seines Spielzugs schnellen Zugriff auf die Bewertung der einzelnen Züge haben, wenn der entsprechende Modus aktiviert ist. Es war deshalb wichtig diese Bewertung fest zu speichern, anstatt sie jedes mal berechnen zu müssen.

3.4.3 Game

Die Klasse Game repräsentiert eine aktive Spielinstanz zwischen zwei Spielern. Sie steht nicht für das Spiel Kalaha und die damit verbunden Regeln (siehe Board 3.4.1).

C Game	
m Game(int, int, int, int)	
m randomStart()	boolean
m takeTurn(int)	void
m takeTurn()	void
m bestTurn()	int
m writeLog(int)	void
p difficulty	int
p gameOver	boolean
p board	Board
p humansTurn	boolean
p tree	Tree

Game baut alle Daten auf, um ein Spiel zwischen Mensch und Roboter zu gewährleisten. Es müssen deshalb die entsprechenden Einstellungen über den Konstruktor übergeben werden.

- **Tree tree** - Der Spielbaum inklusive des aktuellen Spielbretts
- **int difficulty** - Die ausgewählte Schwierigkeit des Nutzers bzw. die Tiefe des Baums.
- **boolean logCreated** - Gibt an ob ein Log für dieses Spiel bereits angelegt wurde
- **boolean io_error** - Gibt an, ob ein IO-Error bereits ausgegeben wurde, damit dies nicht mehrmals hintereinander passiert

3.4.4 Main

In der Main-Klasse wird bei Programmstart die main()-Methode ausgeführt.

Main	
m start(Stage)	void
m main(String[])	void
m playCommandLine(String[])	

In dieser findet die Unterscheidung in GUI- und Konsolen-Spiel statt. Für letzteres existiert eine weitere Methode, die sich um genau diesen Ablauf kümmert. Für den Fall eines Spiels in grafischer Umgebung wird die JavaFX start()-Methode mit entsprechenden Einstellungen aufgerufen.

3.4.5 MenuUIController

Der MenuUIController kümmert sich um die Zuweisung und Speicherung von benutzerdefinierten Einstellungen. Da hier nichts instanziiert wird oder eine weiterreichende Programmlogik stattfindet, werden die Einstellungen als Klassenvariablen gespeichert und können von anderen Punkten des Programms abgerufen werden.

- **static int difficulty** - Der Schwierigkeitsgrad
- **static int stones** - Die Anzahl an Steinen pro Mulde
- **static int start** - Angabe wer das Spiel beginnt. Durch eine mögliche Zufallsentscheidung hat ein boolean nicht ausgereicht
- **static int speed** - Angabe der Animationsgeschwindigkeit

Die Aktionsmethode des Startknopfes startet eine Stage des Spielfeld-UI.

3.4.6 GameController

Der GameController instantiiert ein Objekt der Klasse Game, welches das laufende Spiel repräsentiert. Dies passiert in der initialize()-Methode, welche beim Start ausgeführt wird. In dieser Methode werden ebenfalls alle Spielmulden in einem Array aus Buttons gespeichert, um indiziert auf diese zugreifen zu können.

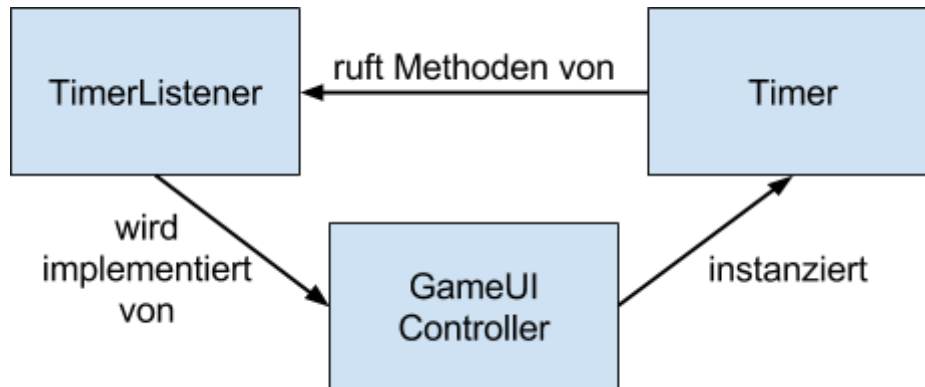
- **Game game** - Das aktive Spiel mit allen enthaltenen Daten und Methoden
- **Button[] btnPit** - Das Array aus Spielmulden-Knöpfen
- **Timer timer** - Eine Timer-Objekt welches für die Animation benötigt wird
- **int chosenIndex** - Index der Mulde auf die ein Spielzug ausgeführt wurde
- **int currentIndex** - Index der Mulde die gerade animiert wird
- **boolean hint** - Angabe ob der Hilfemodus aktiviert ist
- **boolean animationActive** - Angabe ob zur Zeit eine Animation läuft, dann werden alle Knöpfe, bis auf der Menü-Knopf für Userinteraktionen gesperrt

Die hier benutzten Methoden aktualisieren Werte im UI und lösen Methoden in der Spielelogik aus. Die überladene Methode play() wird bei der tatsächlichen Spielausführung aufgerufen.

3.4.7 Timer und TimerListener

Die Klasse Timer ist für die zeitbasierte Animation in der UI zuständig. Sie erbt von der Klasse AnimationTimer, um die darin enthaltenen Methoden zur Zeitverwaltung zu überschreiben. Ihre Zusammenarbeit mit dem Interface TimerListener kann ich folgenden Schritten erklärt werden.

1. Das TimerListener Interface gibt drei Methoden vor
2. Der GameController implementiert dieses Interface und muss dementsprechend ebenfalls die drei Methoden enthalten
3. Der Timer bekommt bei der Instanzierung im GameController eine Referenz auf den Controller und speichert sie als Instanzvariable
4. Diese Referenz ist vom Typ TimerListener, welche vom Controller implementiert wird
5. Im Timer werden an verschiedenen Stellen die drei TimerListener Methoden aufgerufen, welche den Programmverlauf wieder in den Controller holen.



Dadurch kann die gesamte Logik zum Aktualisieren von Knöpfen direkt im Controller bleiben und eine entsprechende Methode auslösen, wenn die Animation vorbei ist. Außerdem kann die Logik aus dem Controller während der Animation weiterlaufen und die benötigten Werte für den nächsten Zug berechnen.

So ist es sogar möglich Kalaha auf Baumtiefe 6 ohne jegliche Verzögerungen zu spielen. Schaltet man die Funktionalität des Logs aus, ist selbst Stufe 8 mit nur kurzen Ladezeiten möglich. Einem Ausbau weiterer Schwierigkeitsstufen steht nichts im Wege.

3.5 Checkliste

Die Checkliste dient als Pflichtenheft, um die geforderten Funktionen aus der allgemeinen Problemstellung final verifizieren zu können.

3.5.1 Spielelogik

- Fällt die letzte Bohne eines Zuges in die jeweils eigene Gewinnmulde, darf der aktuelle Spieler einen weiteren Zug ausführen
- Fällt die letzte Bohne eines Zuges in eine leere Mulde der eigenen Spielfelder, wird diese Bohne und die gegenüberliegenden Bohnen in die eigene Gewinnmulde hinzugefügt
- Das Spiel ist beendet, sobald ein Spieler keine Bohnen mehr in seinen Spielfeldern hat. Der jeweils andere Spieler darf die letzten Bohnen seiner Spielfelder in die eigene Gewinnmulde hinzufügen
- Sieger ist, wer die meisten Bohnen in seiner Gewinnmulde hat

Alle 4 Stichpunkte wurden getestet und verifiziert.

3.5.2 Künstliche Intelligenz

- Die KI des Gegenspielers fällt Entscheidungen auf Basis eines Spielbaums
- Die Tiefe des Baums ist auf drei verschiedenen Tiefen (1, 3 und 5) spielbar
- Eine Tiefenebene wird dann inkrementiert, wenn ein Spielerwechsel stattgefunden hat
- Mulden in denen 0 Steine liegen sind nicht ausführbar und der Baum wird hier nicht weiter berechnet
- Die Tiefe des Baums wird am Anfang des Spiels vollständig aufgebaut
- Die Tiefe des Baums wird nach jedem Spielzug wieder hergestellt
- Die Bewertung im Baum ergibt sich aus möglichen Zwischenständen, die sich auf der untersten Ebene im Baum befinden
- Ist der Nutzer am Zug wird die maximale Bewertung der nächsten sichtbaren Ebene gewählt
- Ist der Roboter am Zug wird die minimale Bewertung der nächsten sichtbaren Ebene gewählt
- Erscheint die bestmögliche Bewertung mehrmals, wird jene gewählt, die weiter von der eigenen Gewinnmulde entfernt liegt
- Der Zustand des Baums wird in beschriebener Syntax bei jedem Zug in eine Log Datei geschrieben

Alle 11 Stichpunkte wurden getestet und verifiziert.

3.5.3 Steuerung

- Mit dem Start der Applikation öffnet sich die Benutzeroberfläche
- Bei einem Eingabeparameter wird der Konsolenmodus anstatt des GUI ausgeführt und der Punktestand zu diesem Zeitpunkt ausgegeben. Damit verbundene Fehlerfälle werden entsprechend abgefangen
- Eine Logdatei wird bei Spielstart angelegt. Ist bereits eine Datei vorhanden wird sie überschrieben. Kann sie nicht überschrieben werden, wird ein Fehler ausgegeben und ohne Logging fortgefahren
- Ein Spielaufruf mit variabler Steinanzahl, Muldenanzahl und Baumtiefe ist möglich
- Die Verwendung von Klassenvariablen und deren Gettern und Settern werden ausschließlich an einer Stelle verwendet (MenuUIController)
- Die Programmaufteilung zwischen Backend und Frontend ist klar definiert

Alle 6 Stichpunkte wurden getestet und verifiziert.

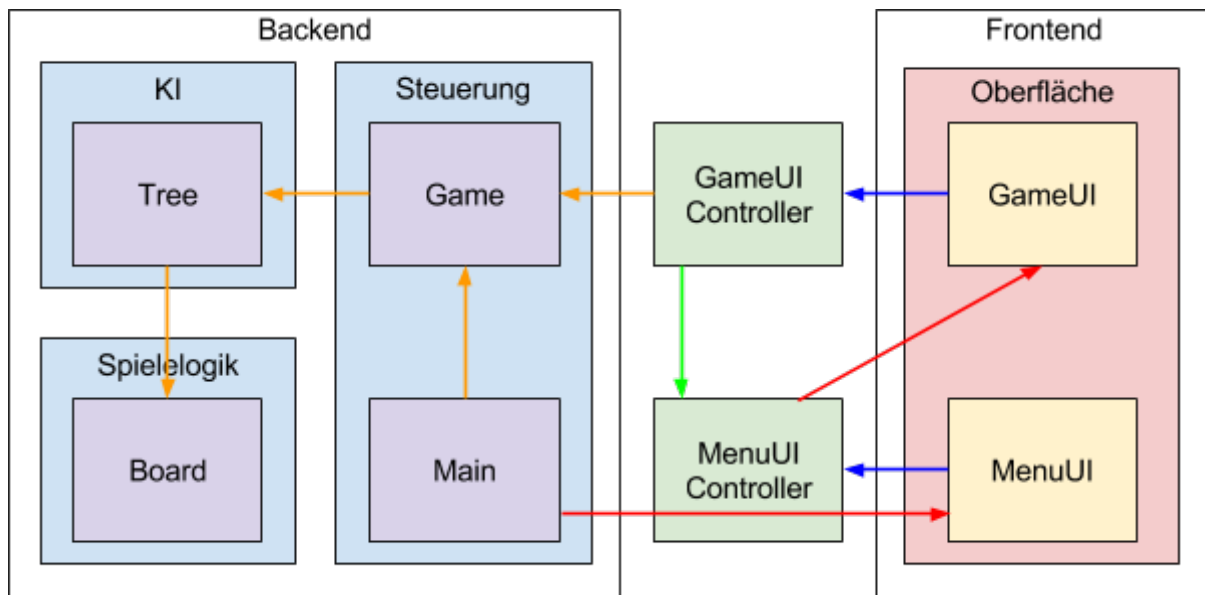
3.5.4 Oberfläche

- Einstellung für Animationsgeschwindigkeit ist in drei Stufen vorhanden
- Einstellung für Animationgeschwindigkeit funktioniert wie beschrieben
- Spielknöpfe lassen sich nicht bedienen während die Animation läuft
- Einstellung für Schwierigkeitsgrad ist in drei Stufen vorhanden
- Einstellung für Schwierigkeitsgrad funktioniert wie beschrieben
- Einstellung für das Aktivieren des Hilfemodus ist vorhanden
- Bei aktiviertem Hilfemodus wird der beste Zug angezeigt
- Bei aktiviertem Hilfemodus ist die Bewertung jedes Spielfelds ersichtlich
- Doppelklick auf eine Spielmulde führt zu einer Zugausführung
- Drücken auf eine Spielmulde führt zur Anzeige wo der letzte Stein landen würde

Alle 10 Stichpunkte wurden getestet und verifiziert.

3.5 Programmierorganisationsplan

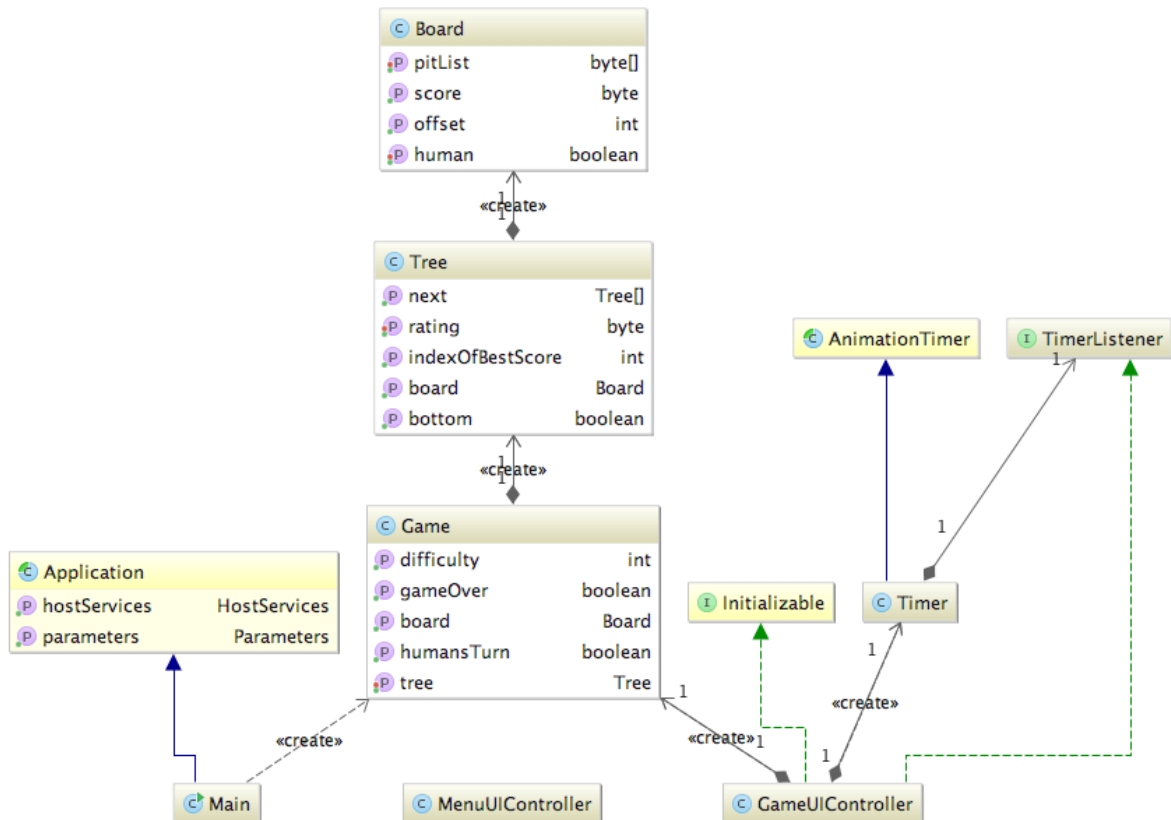
Die vorhergehende Grafik aus der Problemanalyse visualisiert den Aufbau ausschließlich auf der Design-Ebene. In folgender Grafik wurde die Betrachtung auf Klassen-Ebene erweitert und Beziehungen mit angezeigt. Timer und TimerListener können als Teil des GameControllers gesehen werden und werden hier aus Gründen der Übersichtlichkeit nicht weiter aufgeführt.



- **Violette** Elemente sind Java-Klassen im Backend von Kalaha (Model)
- **Gelbe** Elemente sind die FXML-Dateien des Frontends (Views)
- **Grüne** Elemente sind die JavaFX Controller, welche die Verbindung zwischen dem Frontend und Backend schaffen (Controller)

- **Orange** Pfeile stehen für die Instanziierung von Objekten und dem Zugriff auf dort enthaltene Methoden.
- **Rote** Pfeile stehen für das Initialisieren von Oberflächen
- **Blaue** Pfeile stehen für die Eingabeverarbeitung in einer Oberfläche
- Der **grüne** Pfeil steht für den Zugriff auf Methoden von nicht instanziierten Klassen

Folgendes UML-Diagramm wurde mit IntelliJ IDEA 15 erstellt:



3.6 Programmtests

Das Programm wurde auf verschiedenen Ebenen durch unterschiedliche Vorgehensweisen getestet.

3.6.1 User Input

Die Interaktion mit dem Endnutzer findet bei Kalaha ausschließlich über die Maus und damit einhergehenden Klickaktionen statt. Dadurch konnten die möglichen Kombinationen aus Fehleingaben, um ein immenses Maß reduziert werden.

Bei den Spielzügen war darauf zu achten, dass während der Animationen kein weiterer Zug ausgeführt werden kann. Die Spielmulden werden solange blockiert, bis die Animation vorbei ist. Ausschließlich Der Menu-Knopf bleibt jederzeit ansprechbar, um zurück zu den Einstellungen zu kommen.

Mit strukturiert chaotischer Vorhergehensweise ließen sich unzählige Klickaktionen auf verschiedenen Schaltflächen anwenden, um sicher zu gehen, dass nichts unerwartetes passiert. Diese Art von Tests ließen sich schlecht in schriftlicher Form festhalten, weil sie ausschließlich aus Maus-Interaktionen bestanden.

3.6.2 Konsolenmodus

Folgende Fehler werden im Konsolenmodus abgefangen:

- Mehr als ein Eingabeparameter
- Eingabeparameter, die außerhalb der eigenen Spielmulden-Indices liegen
- Eingabeparameter, die keine ganzen Zahlen sind
- Eingabeparameter die eine unzulässige Kombination an Spielzügen beinhalten

3.6.3 Log

Die Logausgabe wurde manuell per Programmstart getestet, um ein korrektes Verhalten bei schreibgeschützten und vorhandenen Dateien sicherzustellen. Zur Kontrolle der Syntax dienten die Aufgabenstellung und daraus hervorgehende Beispielausschnitte, sowie Vergleiche mit der tatsächlichen Datenstruktur des Baums.

3.6.4 Unit Tests

Die drei großen Logik-Klassen Board, Tree und Game wurden mit Hilfe von JUnit getestet. Jede Methode deren Logik über die Komplexität von Gettern oder Settern hinausgeht, wurde mit passenden Fehlerfällen beladen. Durch eine sehr feine Aufteilung von Aktionen in Hilfsmethoden ließen sich Randfälle häufig durch einzelne Aufrufe testen. Testmethoden wie `makeMoves()` ließen spezielle Zustände schnell herbeiführen, um sie auf Richtigkeit zu prüfen. Jede der drei Klassen hat mindestens einen aufwendigen Test, der die Kombination der Kernmethoden überprüft. Diese Tests greifen über hohe Baumtiefen oder viele Züge auf einen großen Teil der möglichen Datenkombinationen zu.

Die `simulateGame()`-Tests aus der `TestGame` Klasse eigneten sich wunderbar, um nach Änderungen im Code die weitere Funktionalität des gesamten Backends zu testen, da hier fast alle Methoden mit einfließen.

Einige Beispiele für einprägsame Testfälle waren:

Testfall	Erwartetes Ergebnis	Erzieltes Ergebnis
Der letzte Stein eines Zugs fällt in eine leere Spielmulde des Nutzers	Dieser Stein und die gegenüberliegenden Steine werden in die eigene Gewinnmulde gelegt	Korrekt ausgeführt
Der Computer verteilt Steine über die Benutzer-Gewinnmulde	Benutzer-Gewinnmulde wird übersprungen	Korrekt ausgeführt
Der Benutzer hat keine Steine mehr in seinen Spielmulden	Spiel ist vorbei und die verbleibenden Steine des Gegners werden in seine Gewinnmulde gelegt	Korrekt ausgeführt
Der Spielbaum wird zu Beginn auf Tiefe 1 ausgebaut	Der Baum hat bei den Indices 1 bis 5 tatsächlich nur eine Ebene. Bei Index 0 kommt eine weitere Ebene dazu, weil der Spieler es hier erreicht 2mal hintereinander am Zug zu sein.	Korrekt ausgeführt
Der Spieler kann mit einem Zug das Spiel beenden während er auf Tiefe 5 spielt.	Der Baum hat trotz Tiefe 5 nur noch eine weitere Ebene auf dem entsprechenden Index, weil danach keine weiteren Züge mehr möglich sind.	Korrekt ausgeführt
Die Klasse Board bekommt auf dem Konstruktor den start-Parameter -1 übergeben.	Der Computer beginnt das Spiel	Korrekt ausgeführt
Der Computer führt einen Zug auf Index 7 aus.	Die Bewertung des noch aktuellen Boards und die Bewertung des Boards unter diesem Index sind identisch	Korrekt ausgeführt
Der Computer führt bei Spielbeginn einen Zug auf Index 0 aus	Der Computer darf einen weiteren Zug ausführen	Korrekt ausgeführt

3.7 Aufwandsauswertung

Die zeitliche Auswertung der Arbeitsstunden ist wie folgt ausgefallen:

Einlesen & Informieren:	07:02
Klassenkonzept:	03:48
UI Erstellung & Verknüpfung:	20:33
Implementation:	29:03
Tests erstellen & Testen:	14:12
Dokumentation:	12:58
 Gesamt:	 87:36