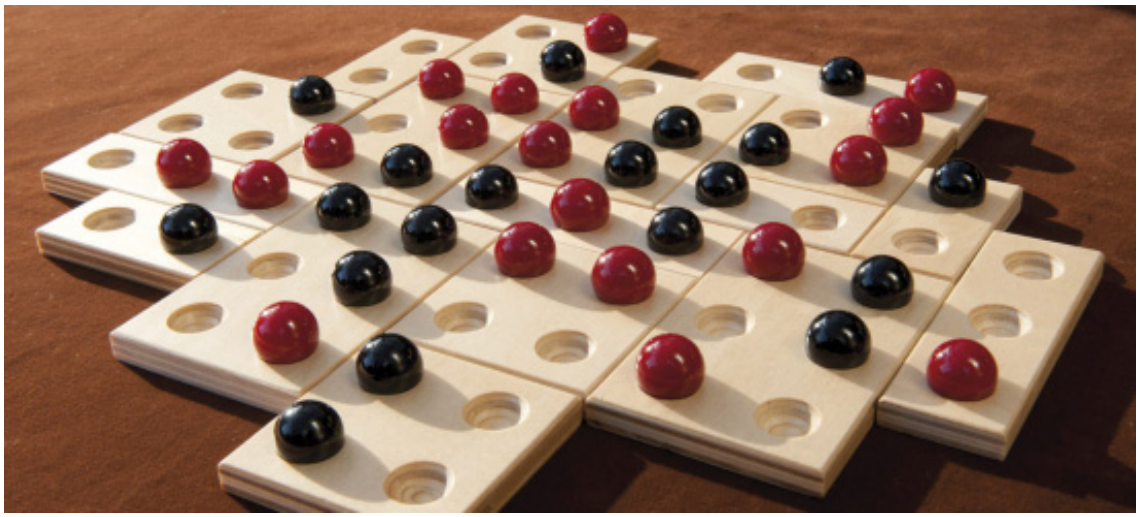
1 Kulami

1.1 Einführung

Kulami ist ein strategisches Legespiel für zwei Spieler, das auf einem variablen Feld ausgetragen wird. Die Spieler verteilen abwechselnd ihre Murmeln auf dem Spielfeld, welches sich aus verschieden großen Platten zusammensetzt. Durch geschicktes Platzieren der eigenen Murmeln versucht jeder Spieler möglichst viele Platten zu erobern. Dabei gilt eine Platte als erobert, wenn ein Spieler auf ihr mehr Murmeln positioniert hat als sein Gegner.

Das für diese Aufgabe maßgebliche Regelwerk finden Sie im Internet[1]. Laden Sie die Regeln herunter und studieren Sie sie gründlich. (Es kann hilfreich sein, sich dieses Spiel zu kaufen und es tatsächlich zu spielen.) Das Spiel wird durch Steffen-Spiele vertrieben, nachdem es von Andreas Kuhnekath erfunden wurde.



1. http://www.steffen-spiele.de/fileadmin/Spiele/PDFs/Kulami-DE.pdf

1.2 Problemstellung

Das Spiel soll als Client-Server-Anwendung realisiert werden. Der Server-Prozess wartet darauf, dass sich ein Client-Prozess (Spieler) bei ihm anmeldet. Der Server erzeugt dann aus der Anmeldung des ersten Clients (C_1) eine neue Spielrunde und erfragt von ihm die Spielfeldkonfiguration sowie den gewünschten Spiellevel. Anschließend weist er C_1, sofern die Angaben verifiziert wurden, eine Farbe zu. Meldet sich ein zweiter Client (C_2) beim Server an, werden C_2 die Spielfeldkonfiguration, der Spiellevel und seine Farbe mitgeteilt. Der Server lost aus, wer die erste Murmel aufs Spielfeld legen darf und informiert die Clients darüber, bevor das Spiel beginnt.

Während des Spiels dient der Server als unparteiischer Schiedsrichter. Er fordert die Clients abwechselnd dazu auf, ihre Züge durchzuführen, und überprüft dann, ob diese gültig sind. Nach jedem korrekten Zug wird das aktualisierte Spielfeld an beide Clients übertragen. Der Server beendet das Spiel, wenn kein Zug mehr möglich ist oder falls das Spiel oder die Verbindung von einem der Clients abgebrochen wird. Im Fall eines regulären Spielendes gibt der Server das Spielergebnis an die Clients weiter.

Ein Kulami-Server wird von der Kursbetreuung zur Verfügung gestellt. **Ihre Aufgabe besteht darin, ein Client-Programm zu entwickeln**, welches sowohl von einem menschlichen Spieler genutzt als auch mit künstlicher Intelligenz betrieben werden kann. Das Programm soll sich am Server anmelden und mit einem anderen Client-Programm (auch einem fremden) Kulami spielen können. Es muss dabei die Spielregeln von Kulami und das Kommunikationsprotokoll des Servers (dessen Spezifikation Sie in Abschnitt 1.5 finden) beachten.

Folgende Anforderungen gelten für beide Spielmodi:

- **Beachten des Protokolls**: vollständige Implementierung des in Abschnitt 1.5 beschriebenen Protokolls
- **Schließen-Knopf**, um das Programmfenster zu schließen und den Client so jederzeit geordnet beenden zu können
- **Auswahl der Art des Spielers** (Mensch oder Computer) und des **Spielernamens**
- **Interaktive Einstellung der Kommunikationsparameter**, etwa: IP-Adresse und Port des Servers (*optional* zusätzlich über eine Parameterdatei)
- **Graphischer Spielfeld-Editor**, der aus dem Hauptmenü aufrufbar ist und die folgenden Optionen bietet:
  - **Neues, leeres Spielfeld erstellen**
  - **Bearbeiten des Spielfelds**: Verteilung der verfügbaren Platten auf dem 10x10 Felder großen Spielfeld

  - **Neu erstelltes Spielfeld dauerhaft speichern**, wobei das neue Spielfeld zunächst auf Korrektheit geprüft wird
  - **Laden gespeicherter Spielfelder** mit Prüfung der Gültigkeit des Spielfelds
  - **Übernahme** des aktuellen Spielfelds ins Spiel
- **Anmeldung am Server, ggf. Konfiguration von Spielfeld und Level der Punktezählung** (für den ersten Spieler); es muss entweder ein gespeichertes Spielfeld verwendet oder ein neues erzeugt werden
- **Übermittlung der Spielzüge an den Server**
- **Laufende graphische Darstellung des Spielgeschehens**, d.h. übersichtliche Anzeige des Spielfelds und des am Zug befindlichen Spielers
- **Ein- und ausblendbare Informationen**, d.h. übersichtliche Anzeige der aktuell möglichen Züge, der zuletzt belegten (und daher nicht verfügbaren) Platten, des aktuellen Punktestands, des aktuellen Plattenbesitzes sowie der Anzahl noch zu legender Murmeln
- **Spielergebnisanzeige** bei Spielende
- **Anzeige von Nachrichten** vom Gegner und vom Client
- **Senden von Nachrichten** an den anderen Spieler
- *Ferner sind nützliche optionale (freiwillige) Erweiterungen denkbar:*
  - **Protokollfenster** zur Anzeige empfangener und gesendeter Client-Server-Nachrichten (hilfreich beim Debuggen des Protokolls)

1.3 HIC-Modus

Der erste Programmmodus soll einen Human Interface Client (HIC) realisieren, mit dessen Unterstützung menschliche Spieler über eine graphische Benutzeroberfläche (GUI) an einem servergestützten Spiel teilnehmen können. Der Human Interface Client soll alle Möglichkeiten des in Abschnitt 1.5 beschriebenen Protokolls nutzen, um mit dem Server und indirekt einem anderen Client zu kommunizieren.

Anforderungen an das HIC-Programm (zusätzlich zu denen aus Abschnitt 1.2):

- **Durchführen von Spielzügen** durch Klick- oder Drag-and-Drop-Mechanismen
- **Möglichkeit zur Spielaufgabe** (z.B. über eine Schaltfläche oder ein Menü)

1.4 KI-Modus

Der zweite Programmmodus soll eine künstliche Intelligenz (KI) implementieren, die zwischen Spielstart und Spielende völlig selbstständig agiert.

Anforderungen an das KI-Programm (zusätzlich zu denen aus Abschnitt 1.2):

- **Es sollen verschiedene Spielstärken realisiert werden:**
  - **Stufe 1:** Die KI führt einen zufälligen gültigen Zug aus.
  - **Stufe *n*** (mit $n \in \{2, ..., 10\}$): Die KI führt denjenigen Zug aus, der ihr gemäß einem (zu implementierenden) Spielbaum der Höhe $n - 1$ den größten Erfolg verschafft. Dabei soll n vor Beginn des Spiels gewählt werden. Für $n = 2$ wird beispielsweise immer derjenige Zug ausgeführt, der die bestmögliche Auswirkung auf den aktuellen Punktestand der KI hat (der aber nicht unbedingt die besten Siegchancen garantiert sondern möglicherweise zu einer raschen Niederlage führt)

1.5 Das Kommunikationsprotokoll

Die Kommunikation zwischen Serverprogramm und Clientprogrammen folgt einem vorgegebenen Protokoll. Es findet keine direkte Kommunikation zwischen den Clientprogrammen statt.

Die Kommunikation erfolgt mittels eines festgelegten Nachrichtensatzes über eine TCP/IP-Verbindung. In jeder Situation sind nur bestimmte Nachrichten erlaubt. Trotzdem müssen die Clients fehlertolerant implementiert werden, d.h. sie müssen

1. im Falle eines plötzlichen unerwarteten Verbindungsabbaus eine Fehlermeldung ausgeben und sich ggf. ordentlich herunterfahren,
2. unbekannte und unerwartete Nachrichten entgegennehmen und verwerfen ohne abzustürzen und angemessen darauf reagieren (z.B. durch eine Fehlermeldung, Beenden des Spiels etc.).

Ein Client baut eine dauerhafte Verbindung zum Server auf, die während des gesamten Spiels oder sogar über mehrere Spiele hinweg bestehen bleibt. Er empfängt Nachrichten vom Server, auf die entsprechend zu reagieren ist. Er sendet Nachrichten an den Server, auf die dieser wiederum reagiert. Jede Nachricht besteht aus einer einzigen Zeile, die mittels „\n“ abgeschlossen wird.

Nachrichten, die vom Server akzeptiert werden, sind:

neuerClient(<Name>).

Anmeldung eines neuen Mitspielers (der abschließende Punkt gehört mit zur Nachricht).

spielparameter(<Board>, <Level>).

Übermittelt das initiale Spielfeld sowie den Level für die Punktezählung an den Server. Diese Nachricht kann einmalig vom ersten angemeldeten Spieler versendet werden.

Der Level ist eine ganze Zahl zwischen 0 und 2 mit folgender Bedeutung:

- 0 : nur Platten werden gezählt
- 1 : Platten und Gebiete werden gezählt
- 2 : Platten, Gebiete und Ketten werden gezählt

Das Spielfeld wird durch genau 200 Zeichen repräsentiert, d.h. jedes Feld des 10x10 Felder großen Spielfelds ist durch zwei Zeichen kodiert. Das erste Zeichen ist ein Buchstabe $\in \{a, ..., r\}$, der die zum Feld gehörige Platte beschreibt. Dabei bedeuten:

- a : Feld nicht durch Platte belegt
- b-e : 6er Platten
- f-j : 4er Platten
- k-n : 3er Platten
- o-r : 2er Platten

Das zweite Zeichen $\in \{0, 1, 2\}$ zeigt die aktuelle Belegung des Felds an:

- 0 : nicht belegt
- 1 : schwarz
- 2 : rot

"a"-Felder dürfen nicht belegt sein. Das Spielfeld wird von links nach rechts und oben nach unten aufgebaut. Einen Beispielstring finden Sie am Endes dieses Abschnitts.

Natürlich erscheint es zunächst sinnlos, die aktuelle Belegung beim Spielstart ebenfalls zu übermitteln. Dies dient jedoch der einheitlichen Beschreibung des Spielfelds zu Beginn und während des Spiels. Eine mitgelieferte Belegung wird beim Spielstart vom Server gelöscht, so dass stets mit einem leeren Spielfeld begonnen wird.

zug(x, y).

Führt einen Zug im laufenden Spiel aus, mit $x, y \in \{0, ..., 9\}$.

message(<Nachricht>).

<Nachricht> wird an den anderen Spieler weitergeschickt, sofern dieser bereits verbunden ist.

spielaufgabe.

Führt zum Beenden des laufenden Spiels und zur Trennung der Verbindung zum Server. Der zweite Spieler wird über den Abbruch unterrichtet und ebenfalls getrennt.

neuesspiel.

Nach einem regulären Spielende kann hiermit ein neues Spiel mit der gleichen Ausgangskonfiguration (Board, Farbzuteilung an Spieler, Punktelevel) gestartet werden.

Andere Nachrichten an den Server gelten als Protokollverstoß und führen zur Trennung der Verbindung zum Spieler.

Die Nachrichten, die der Server verschickt, sind die folgenden:

Kulami?

Identifikation des Servers. Wird direkt nach erfolgreicher Verbindung gesendet.

message(<Nachricht>).

Nachricht vom Server an den Spieler. Kann jederzeit nach Verbindungsaufbau auftreten.

spielparameter?

Aufforderung zum Senden der Spielparameter. Wird einmalig an Spieler 1 nach dessen Anmeldung (mittels **neuerClient(<Name>).**) geschickt.

spielparameter(<Board>, <Level>, <Farbe>, <Name>).

Übertragung bereits feststehender Spielparameter

- <Board> aktuelles Spielfeld
- <Level> Level für die Punktezählung
- <Farbe> zugewiesene Farbe
- <Name> Name des gegnerischen Spielers

Wird an Spieler 2 geschickt, nachdem sich dieser angemeldet hat und <Board> sowie <Level> von Spieler 1 bekannt gegeben sind. Nach Neustart des Spiels wird diese Nachricht an beide Spieler verschickt.

name(<Name>).

Nachricht an Spieler 1, dass sich ein zweiter Spieler mit Namen <Name> angemeldet hat.

farbe(<Farbe>).

Zuteilung einer Farbe durch den Server an Spieler 1. <Farbe> ist hier entweder r oder b.

spielstart(<Farbe>).

Spiel hat begonnen, Spieler <Farbe> ist an der Reihe. Wird an beide Spieler geschickt.

ungültig(<Nachricht>).

Reaktion des Servers auf einen ungültigen Zug. <Nachricht> enthält den Grund der Ungültigkeit.

gültig(<Board>).

Reaktion des Servers auf einen gültigen Zug (geändertes Spielfeld wird übertragen)

zug(<Board>).

Mitteilung an einen Spieler, dass der Gegner einen gültigen Zug durchgeführt hat.

spielende(<punkteRot>, <punkteSchwarz>).

Spiel wurde normal beendet (keine Kugeln mehr vorhanden oder kein Zug mehr möglich). Die Punkte beider Spieler für den ausgewählten Punktelevel werden übertragen.

playerMessage(<Nachricht>).

Nachricht vom anderen Spieler kommend. Kann jederzeit auftreten.

Das folgende Szenario beschreibt eine mögliche Kommunikation zwischen dem Server S und den Spielern P_1 und P_2.

P_1 verbindet sich mit S

S sendet **Kulami?** an P_1

P_1 sendet **neuerClient(Fabio).** an S

S sendet **message(Willkommen Fabio).** an P_1

S sendet **spielparameter?** an P_1

P_1 sendet **spielparameter(a0a0...r0, 1).** an S

P_2 verbindet sich mit S

S sendet **Kulami?** an P_2

P_2 sendet **neuerClient(Thomas).** an S

S sendet **message(Willkommen Thomas).** an P_2

S sendet **name(Thomas).** an P_1

S sendet **Farbe(b).** an P_1

S sendet **spielparameter(a0a0...r0, 1, r, Fabio).** an P_2

S sendet **spielstart(r).** an P_1

S sendet **spielstart(r).** an P_2

P_1 sendet **message(Hallo Thomas).** an *S*

S sendet **playerMessage(Hallo Thomas).** an P_2

P_2 sendet **message(Hallo Fabio).** an *S*

S sendet **playerMessage(Hallo Fabio).** an P_1

P_1 sendet **zug(4, 4).** an *S*

S sendet **ungültig(Rot ist am Zug).** an P_1

P_2 sendet **zug(6, 6).** an *S*

S sendet **gültig(a0a0...r0).** an P_2

S sendet **zug(a0a0...r0).** an P_1

P_1 sendet **zug(8, 3).** an *S*

.....

.....

S sendet **spielende(24, 24).** an P_1

S sendet **spielende(24, 24).** an *P2*

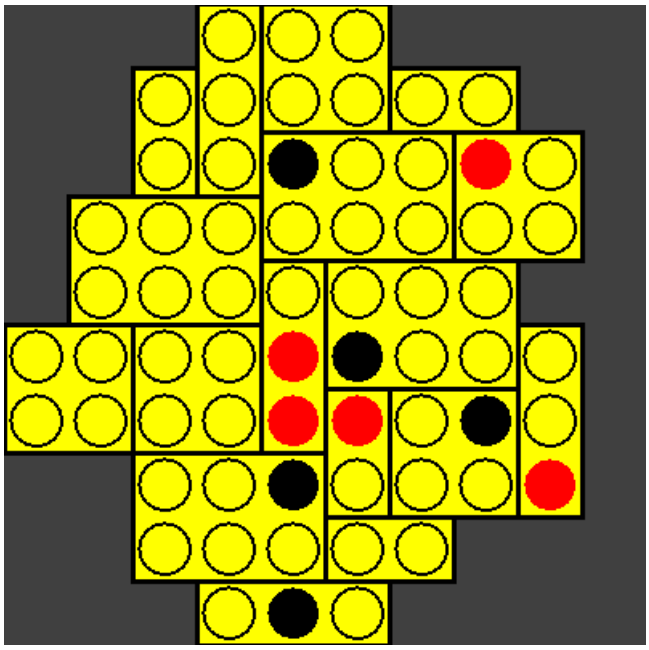
P_1 beendet die Verbindung zu *S*

S sendet **message(Fabio hat sich abgemeldet).** an P_2

S beendet die Verbindung zu P_2.

Die Clients müssen auf Nachrichten vom Server stets angemessen reagieren. Dies umfasst z.B. die Aktualisierung des angezeigten Spielfelds und der Punktestände sowie die Anzeige des Gegnernamens. Das Warten auf Servernachrichten darf den Client nicht blockieren. Beispielsweise ist es immer möglich, dem Gegner Nachrichten zu schicken.

1.6 Spielfeldkodierung



Dieses Spielfeld wird mittels des folgenden Strings kodiert übertragen:

```
a0a0a0k0f0f0a0a0a0a0
a0a0o0k0f0f0p0p0a0a0
a0a0o0k0b1b0b0g2g0a0
a0c0c0c0b0b0b0g0g0a0
a0c0c0c0l0d0d0d0a0a0
h0h0i0i0l2d1d0d0m0a0
h0h0i0i0l2q2j0j1m0a0
a0a0e0e0e1q0j0j0m2a0
a0a0e0e0e0r0r0a0a0a0
a0a0a0n0n1n0a0a0a0a0
```

Die Zeilenumbrüche dienen nur der besseren Orientierung und sind nicht Bestandteil der Kodierung.

10