MNM
TEAM
Munich Network Management Team

Ludwig-Maximilians-Universität München

Prof. Dr. D. Kranzlmüller
Dr. N. gentschen Felde

Systempraktikum — Projektaufgabe (Teil 1 von 4)

Willkommen in der Gruppenphase des Systempraktikums. Ihre Aufgabe in der Projektphase ist es, einen Client für das Brettspiel *Quarto* in der Programmiersprache `C` zu entwickeln. Die Übungsblätter werden Sie schrittweise zu diesem Ziel führen.

Der Lehrstuhl stellt im Rahmen des Systempraktikums einen Gameserver und ein Webinterface bereit. Der Gameserver implementiert den Spielablauf, Ihr Client die Spielelogik. Der Gameerver ist gewissermaßen der Spielleiter und somit insbesondere für die Regeleinhaltung verantwortlich, wohingegen Ihr Client in die Rolle eines Spielsers schlüpft. Über das Webinterface unter `http://sysprak.priv.lab.nm.ifi.lmu.de`[1] können Spiele verwaltet werden.

Um das Testen Ihres Clients zu erleichtern, bietet das Webinterface zudem die Möglichkeit als menschlicher Spieler an einem Spiel teilzunehmen. Sie können jetzt testweise ein neues Spiel erstellen und zum Einstieg gegen ihre Kommilitonen oder sich selbst spielen. Sie werden einen Einblick gewinnen, wie das Spiel abläuft. Nähere Hinweise zu den Regeln finden Sie im weiteren Verlauf dieses Dokuments ab Seite 2.

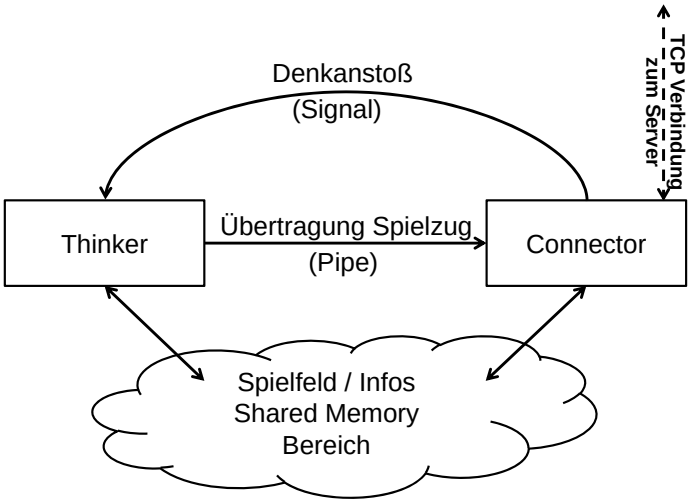

Abbildung 1: Übersicht über den Client

Der von Ihnen zu entwickelnde Client wird im Verlaufe des Systempraktikums schrittweise entwickelt. Maßgeblich besteht er aus zwei Prozessen, dem *Thinker* und dem *Connector*. Der *Connector* übernimmt die Kommunikation mit dem Gameserver und der *Thinker* berechnet den nächsten Spielzug.

Wird der Connector vom Gameserver zum nächsten Spielzug aufgefordert, so legt er alle Informationen, die er vom Gameserver bekommen hat um den nächsten Spielzug zu berechnen, in einen *geteilten Speicherbereich*. Danach sendet der Connector ein *Signal* an den Thinker. Der dadurch „aufgeweckte“ Thinker fängt nun an, mithilfe der aus dem *geteilten Speicherbereich* gelesenen Informationen, den nächsten Spielzug zu berechnen. Das Ergebnis der Berechnungen, den Spielzug, sendet der Thinker über eine *Pipe* an den Connector zurück, welcher den Spielzug an den Gameserver sendet. Abbildung 1 gibt Ihnen einen groben Überblick über den Aufbau des Clients.

[1] Der Gameserver und das Webinterface sind nur aus dem MWN erreichbar. (Informationen zum MWN: `http://www.lrz.de/services/netz/`)

MNM
TEAM
Munich Network Management Team

Ludwig-Maximilians-Universität München

Prof. Dr. D. Kranzlmüller
Dr. N. gentschen Felde

Spielregeln zu „Quarto“

Es gelten folgende Regeln bei Quarto:

- Das Spielfeld ist Quadratisch und ist $n \times n$ groß. In unserem Fall wird später $n = 4$ gelten, Ihre Implementierung soll jedoch mit einem beliebig großen Spielfeld umgehen können.
- Es gibt n binäre Eigenschaften. Es gibt n^2 viele Steine. Jeder dieser Steine besitzt alle n Eigenschaften, jedoch keiner die selben Ausprägungen. Ein Spielstein kann somit als n-Tupel binärer Werte aufgefasst werden.
  Beispiel: Gehen wir von $n = 2$ und den Eigenschaften $Größe \in \{groß, klein\}$ und $Form \in \{rund, eckig\}$ aus. Es gibt also $2^2 = 4$ Spielsteine, nämlich...
  1. einen Stein, der groß und eckig ist,
  2. einen Stein, der groß und rund ist,
  3. einen Stein, der klein und eckig ist und
  4. einen Stein, der klein und rund ist.
- Zu Beginn ist das Spielfeld leer und der Gameserver wählt den ersten Spieler sowie einen Stein aus.
- Ist ein Spieler am Zug, so muss er den ihm übergebenen Stein auf ein freies Feld setzen und einen Stein auswählen, welchen sein Gegenspieler im nächsten Zug setzen muss. Letzteres entfällt, sollten mit diesem Zug alle Felder besetzt sein. Es darf kein Stein für den nächsten Zug ausgewählt werden, welcher schon auf dem Feld ist oder den der Spieler selbst gerade setzen muss.
- Gesetzte Steine können nicht entfernt oder verschoben werden.
- Gewonnen hat, wer den Stein gelegt hat, mit dem eine horizontale, vertikale oder diagonale Linie von n Steinen vervollständigt wurde, die folgender Anforderung genügt: Die Ausprägungen mindestens einer Eigenschaft ist bei allen Steinen auf der Linie identisch (z. B. alle Steine sind rund).
- Gibt es nach einem Zug keine freien Felder mehr und ist das Spiel nicht gewonnen worden, so gilt das Spiel als Unentschieden.

Auf dem Webinterface können Sie versuchshalber auch mit zwei menschlichen Spielern spielen, um etwas zu üben und die Regeln zu verinnerlichen.

MNM
TEAM
Munich Network Management Team

Ludwig-Maximilians-Universität München

Prof. Dr. D. Kranzlmüller
Dr. N. gentschen Felde

Übungsaufgaben

Ihre Aufgabe für dieses Übungsblatt ist es, die erste Protokollphase („Prolog“) der Kommunikation mit dem Gameserver zu implementieren. In Abbildung 1 ist dies durch die TCP-Verbindung zum Gameserver des rechten Prozesses (*Connector*) dargestellt. Die genaue Protokolldefintion finden Sie in diesem Dokument ab Seite 4. Folgende Teilaufgaben sind dazu unteranderem zu erledigen.

- Ihr Programm muss eine 11-stellige Game-ID als Kommandozeilenparameter (Hinweis: `argv` und `argc`) auslesen können.
- Definieren sie die drei Konstanten `GAMEKINDNAME` mit dem Wert `"Quarto"`, `PORTNUMBER` mit dem Wert `1357`, und `HOSTNAME`, welche den Wert `"sysprak.priv.lab.nm.ifi.lmu.de"` erhält (Hinweis: `#define`). Die Werte ergeben sich aus der Protokollbeschreibung.
- Anschließend verbinden Sie sich mit dem Gameserver (Hinweis: `socket`, `gethostbyname()` und `connect()`) und rufen die später von Ihnen zu implementierende Methode `performConnection()` auf, welche als Argument den File-Descriptor Ihres Sockets übergeben bekommt.
- Implementieren Sie nun die „Prolog“-Phase der Kommunikation in der Methode `performConnection()`. Geben Sie beim `PLAYER`-Kommando keine Werte mit und lassen Sie sich vom Gameserver einen freien Spieler zuweisen. Diese Methode sollte sich der besseren Übersichtlichkeit halber in der separaten Datei `performConnection.c` befinden. Sie können nach Belieben zusätzliche Methoden und Dateien erstellen, wenn diese Ihnen helfen.
- Geben Sie alle vom Gameserver erhaltenen Informationen wohl formatiert aus, d. h. nicht die Protokollzeile vom Gameserver, sondern z. B.: „Spieler 1 (Uli) ist noch nicht bereit“. Achten Sie hierbei darauf, dass Integer-Werte, wie z. B. die 1, auch als solche interpretiert werden.
- Achten Sie bei all Ihren Aufrufen auf eine ordentliche Fehlerbehandlung (Hinweis: `perror`), da Ihr Programm Fehler, wie z. B. ein nicht vorhandener Host oder ein nicht laufender Gameserver, erkennen und melden sollte.
- Testen Sie Ihren Client ausführlich mit dem Gameserver! Versuchen Sie einem (nicht) existierenden Spiel mit freien/besetzten Computergegnern beizutreten. Achten sie dabei darauf, dass Ihr Client die Fehlermeldungen des Gameservers richtig interpretiert und sich entsprechend verhält.
- Zum leichteren Übersetzen Ihres Programms erstellen Sie ein Makefile für Ihr Projekt. Es soll die einzelnen Quelldateien zu Objektdateien kompiliert und diese zu der ausführbaren Datei `client` linken.

Achtung: Verwenden Sie für all Ihre Übersetzungen die gcc-Schalter `-Wall -Werror`. Dies führt dazu, dass auch Kleinigkeiten als Warnung ausgegeben werden und der Compiler eine Warnung als einen Fehler ansieht und abbricht. Dies dient dem Zweck Ihnen eine spätere, lästige Fehlersuche zu ersparen, die wesentlich aufwändiger ist als die Warnungen frühzeitig zu beseitigen bzw. zu vermeiden.

MNM
TEAM
Munich Network Management Team

Ludwig-Maximilians-Universität München

Prof. Dr. D. Kranzlmüller
Dr. N. gentschen Felde

Protokolldefinition des Gameservers

Der MNM-Gameserver ist wie folgt zu erreichen:

- *Hostname:* `sysprak.priv.lab.nm.ifi.lmu.de`
- *Port:* `1357` (TCP)

Die folgende Protokolldefinition kürzt eine Zeile, welche vom Client an den Gameserver geschickt wird, mit **C:** für *Client* ab. Eine Zeile, welche vom Gameserver an den Client übermittelt wird, wird mit **S:** für *Server* abgekürzt.

Wenn der Gameserver eine Zeile mit einem + als ersten Buchstaben schickt, ist dies eine positive Antwort. Im Folgenden ist nur der positive Verlauf einer Kommunikation angegeben. An jedem Schritt kann eine Negativantwort auftreten, diese ist erkennbar an dem **-** als erstes Zeichen der Zeile. Ein „-“ ist stets gefolgt von einer aussagekräftigen Fehlermeldung. Im Anschluss an die Fehlermeldung wird die Verbindung getrennt.

In doppelten spitzen Klammern eingeschlossene Werte werden obligatorisch durch die ihnen entsprechenden Werte ersetzt, wie z. B. ⟨⟨ *Game-ID* ⟩⟩ durch die 11-stellige Game-ID. Werte, die in doppelten eckigen Klammern eingeschlossen sind, geben optionale Werte an, d. h. sie können auch weggelassen werden.

Es gibt drei Phasen in diesem Protokoll:

1. *Prolog* – hier wird dem Spiel beigetreten und Informationen über das Spiel ausgetauscht
2. *Spielverlauf* – hier wird gewartet bis man an der Reihe ist, bzw. das Spiel beendet wird
3. *Spielzug* – hier übermittelt der Gameserver ein Spielfeld und erwartet einen Spielzug

Wenn nicht innerhalb eines vom Gameserver festgelegten Zeitraums auf Befehle geantwortet wird oder eine zu lange „Denkzeit“ benötigt wird (s. u.), schickt der Gameserver:
S: - TIMEOUT ⟨⟨ *Begründung* ⟩⟩

Abbildung 2 gibt eine grob-granulare Übersicht über den Ablauf der drei Protokollphasen.

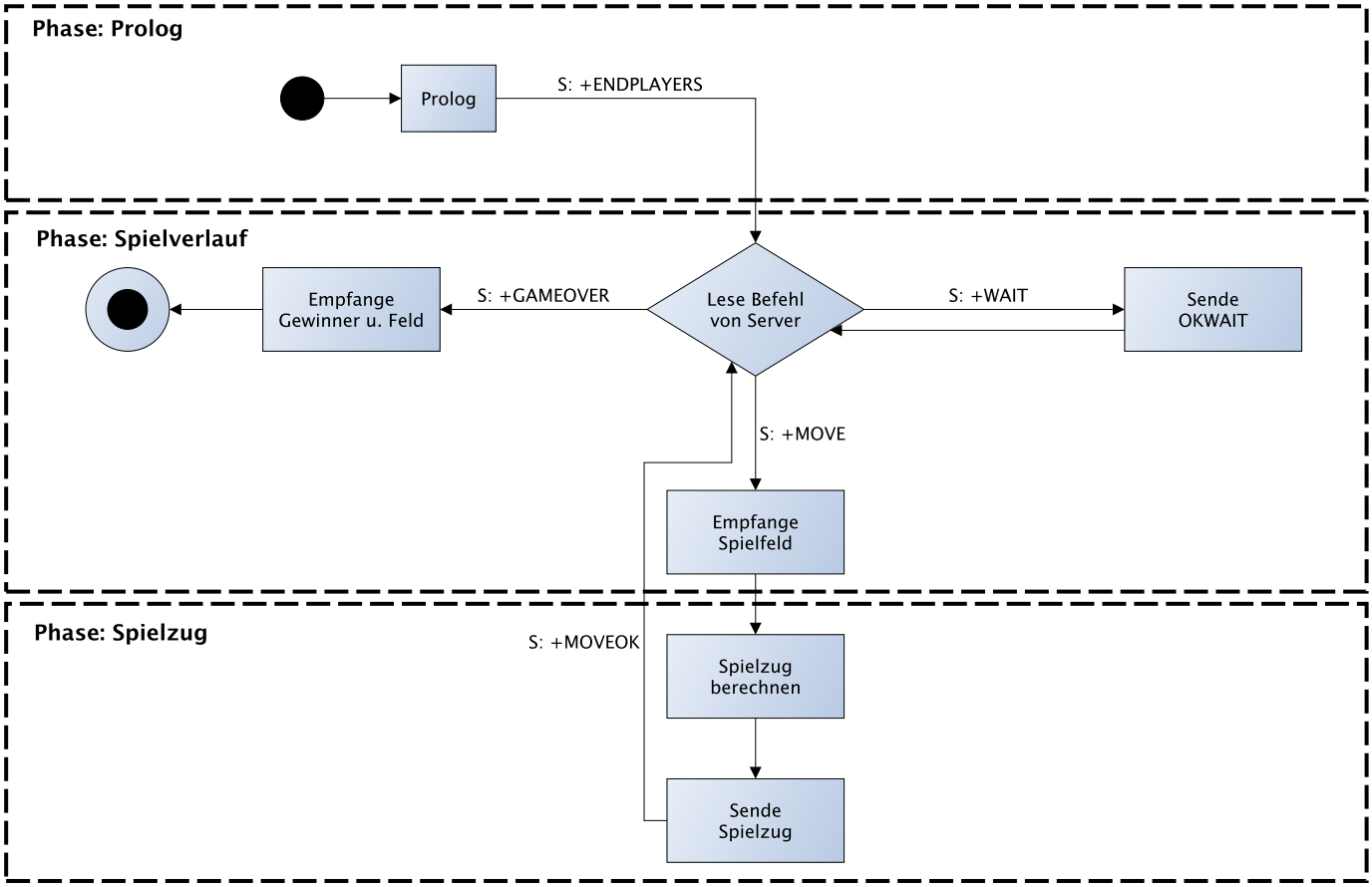

Abbildung 2: Protokollübersicht

1. Protokollphase „Prolog“

⟨⟨ *Aufbau der TCP-Verbindung durch Client* ⟩⟩
S: `+ MNM Gameserver` ⟨⟨ *Gameserver Version* ⟩⟩ `accepting connections`
C: `VERSION` ⟨⟨ *Client Version* ⟩⟩
S: `+ Client version accepted - please send Game-ID to join`
C: `ID` ⟨⟨ *Game-ID* ⟩⟩
S: `+ PLAYING` ⟨⟨ *Gamekind-Name* ⟩⟩
S: `+` ⟨⟨ *Game-Name* ⟩⟩
C: `PLAYER` *[[Gewünschte Spielernummer]]*
S: `+ YOU` ⟨⟨ *Spielernummer* ⟩⟩ ⟨⟨ *Spielername* ⟩⟩
S: `+ TOTAL` ⟨⟨ *Spieleranzahl* ⟩⟩

Nun kommt für jeden der anderen Spieler die Zeile:
S: `+` ⟨⟨ *Spielernummer* ⟩⟩ ⟨⟨ *Spielername* ⟩⟩ ⟨⟨ *Bereit* ⟩⟩

S: `+ ENDPLAYERS`

Der Gameserver akkzeptiert den Client, wenn die Major-Versionsnummern (links vom Punkt) von ⟨⟨ *Gameserver Version* ⟩⟩ und ⟨⟨ *Client Version* ⟩⟩ gleich sind. Der Gameserver setzt jeweils noch ein „v“ voran. Die Major-Versionsnummer des Gameserver ist immer 1, die Minor-Versionsnummer kann sich jedoch ändern. Kompatibel sind z.B. ein Gameserver mit der Version `v1.1` und ein Client mit der Version `1.42`. Nicht kompatibel wären hingegen `v1.1` und `2.1` oder `v1.1` und `v1.1` (Client hat ein „v“ Präfix).

Hinweise:

- Da der Gameserver nicht nur Quarto spielen kann, teilt ⟨⟨ *Gamekind-Name* ⟩⟩ die Spielart mit. Ist diese anders als `Quarto`, muss der Client sich mit einer Meldung, die auf dieses Problem hinweist, beenden.
- ⟨⟨ *Game-Name* ⟩⟩ und ⟨⟨ *Spielername* ⟩⟩ können im Webinterface beim erstellen eines Spiel angegeben werden.
- Wenn man einen Spieler mit einer bestimmten Nummer übernehmen möchte, kann dies mit *[[Gewünschte Spielernummer]]* als Parameter beim `PLAYER`-Kommando angegeben werden. Wenn die Angabe ausbleibt, wird eine freier Computerspieler vom Gameserver zugeteilt.
- Die ⟨⟨ *Spieleranzahl* ⟩⟩ wird bei einem Quarto-Spiel immer `2` sein.
- Ist ein Spieler bereits verbunden, so ist ⟨⟨ *Bereit* ⟩⟩ `1`, ansonsten `0`.

2. Protokollphase „Spielverlauf“

In dieser Phase können die folgenden drei Befehlssequenzen vorkommen.

2.1. „Idle“

S: `+ WAIT`
C: `OKWAIT`

Diese `WAIT`-Befehle kommen in regelmäßigen Abständen und müssen rechtzeitig mit einem `OKWAIT` quittiert werden, ansonsten beendet der Gameserver die Verbindung und das Spiel gilt im Turnier als verloren. Wie dem Diagramm aus Abbildung 2 zu entnehmen ist, bleiben die `WAIT`-Befehle z.B. in der Protokollphase „Spielzug“ oder währrend der „Move“ Befehlssequenzen aus.

2.2. „Move“

S: + MOVE ⟨⟨ *Maximale Zugzeit* ⟩⟩
S: + NEXT ⟨⟨ *Zu setzender Spielstein* ⟩⟩
S: + FIELD ⟨⟨ *Spielfeld Breite* ⟩⟩ , ⟨⟨ *Spielfeld Höhe* ⟩⟩

Die folgende Zeile wird nun für jede Zeile des Spielfeldes geschickt, beginnend bei der obersten Zeile des Spielfelds:
S: + ⟨⟨ Y ⟩⟩ ⟨⟨ $Stein_{1Y}$ ⟩⟩ ⟨⟨ $Stein_{2Y}$ ⟩⟩ ... ⟨⟨ $Stein_{X_{max}Y}$ ⟩⟩

S: + ENDFIELD
C: THINKING
S: + OKTHINK

Der MOVE-Befehl fordert zum Zug auf. Nach der in Millisekunden angegebenen ⟨⟨ *Maximale Zugzeit* ⟩⟩ muss die anschließende Protokollphase „Spielzug“ abgeschlossen sein. Bitte berücksichtigen Sie bei Ihrer Implementierung die Latenzzeiten der Verbindung. Der Client muss direkt nach der Übermittlung des Spielfeldes THINKING schicken und hat hierfür wenig Zeit. Nach der Gameserver-Antwort OKTHINK befinden wir uns in der Protokollphase „Spielzug“.

Die Spielfeldgröße wird für Quarto immer 4x4 sein. Ist ein Feld belegt, so entspricht ⟨⟨ $Stein_{XY}$ ⟩⟩ der Nummer des Spielsteins (0 – 15), ansonsten *. Zum Beispiel könnte die erste übermittelte Zeile + 4 * * 15 * lauten. Somit ist das Feld C4 mit Spielstein 15 belegt und A4, B4 sowie D4 sind frei.

2.3. „Game over“

S: + GAMEOVER *[[* ⟨⟨ Spielernummer des Gewinners ⟩⟩ ⟨⟨ Spielername des Gewinners ⟩⟩ *]]*
S: + FIELD ⟨⟨ *Spielfeld Breite* ⟩⟩ , ⟨⟨ *Spielfeld Höhe* ⟩⟩

Die folgende Zeile wird nun für jede Zeile des Spielfeldes geschickt, beginnend bei der obersten Zeile des Spielfelds:
S: + ⟨⟨ Y ⟩⟩ ⟨⟨ $Stein_{1Y}$ ⟩⟩ ⟨⟨ $Stein_{2Y}$ ⟩⟩ ... ⟨⟨ $Stein_{X_{max}Y}$ ⟩⟩

S: + ENDFIELD
S: + QUIT
⟨⟨ *Abbau der TCP-Verbindung durch Gameserver* ⟩⟩

Bei einem unentschieden, wird kein Gewinner angegeben.

3. Protokollphase „Spielzug“

C: PLAY ⟨⟨ *Spielzug* ⟩⟩
S: + MOVEOK

Ein ⟨⟨ *Spielzug* ⟩⟩ besteht aus der Koordinate, an der der mit NEXT übergebene Stein plaziert werden soll. Sofern mit diesem Zug das Spielfeld nicht voll belegt ist, follgt die Nummer des Steins, den der Mitspieler als nächstes Plazieren soll. Die beiden angaben werden mit Komma getrennt. Ein gültiger Spielzug ist z.B. B4,3. Sollte der übermittelte Zug nicht gültig sein, wird eine entsprechende Fehlermeldung übermittelt und die Verbindung getrennt.