MNM
TEAM
Munich Network Management Team

Ludwig-Maximilians-Universität München

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

Willkommen in der Gruppenphase des Systempraktikums. Ihre Aufgabe in der Projektphase ist es, einen Client für das Brettspiel *Reversi* 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 Web-Interface bereit. Der Gameserver implementiert den Spielablauf, Ihr Client die Spielelogik. Der Gameserver ist gewissermaßen der Spielleiter und somit insbesondere für die Regeleinhaltung verantwortlich, wohingegen Ihr Client in die Rolle eines Spielers schlüpft. Über das Web-Interface unter `http://sysprak.priv.lab.nm.ifi.lmu.de` können Spiele verwaltet werden. Der Gameserver und das Web-Interface sind nur aus dem MWN[1] erreichbar (Hinweis: `ssh -L` auf die CIP-Rechner).

Um das Testen Ihres Clients zu erleichtern, bietet das Web-Interface 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.

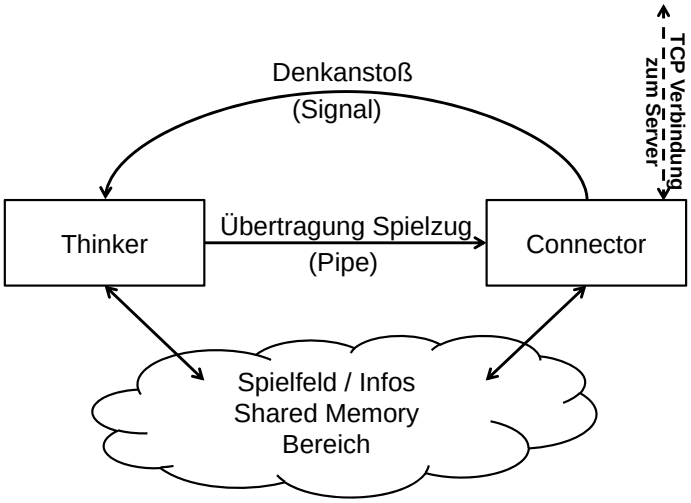

Abbildung 1: Übersicht über den Client

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 *unnamed 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] Informationen zum MWN: `http://www.lrz.de/services/netz/`

1 Reversi

In einer Zeit, in der der erste Sherlock Holmes Roman veröffentlicht wird, Bismarck ein Sozialversicherungssystem einführt und Herman Hollerith die ersten Daten auf Lochkarten speicherte, wurde auch das Spiel *Reversi* erfunden, bzw. es wurde gleich mehrfach erfunden. So stritten sich in den 1880er Jahren zwei Londoner um das Urheberrecht. 1971 wurde in Japan auch noch *Othello*, ein verblüffend ähnliches Spiel, angemeldet, das eines der ersten Arkade-Spiele von Nintendo wurde. Heutzutage versteht man unter dem Begriff "Reversi" in der Regel ein Spiel nach den Othello regeln, so auch hier im Systempraktikum. Für ein Spiel mit gerader Anzahl an Zeilen und Spalten wird davon ausgegangen, dass ein perfekter Spieler nicht verlieren kann.

1.1 Spielregeln

Ein normales Reversi Spielbrett besteht aus 8×8 Feldern. Hier, im Systempraktikum, ist die Spielbrettgröße jedoch flexibel. Es kann auf Feldern der Größe 6×6 bis 16×16 gespielt werden. Die Anzahl der Zeilen/Spalten kann im Web-Interface beim anlegen eines neuen Spiels angegeben werden. Die Anzahl muss dabei stets gerade sein.

Bei der Startaufstellung zu Beginn einer Partie liegen bereits zwei Steine eines jeden Spielers in einem 2×2-Block in der Mitte des Spielfeldes. Die Steine des weißen Spielers sind links oben und rechts unten, die Steine des schwarzen Spielers liegen jeweils auf den beiden verbleibenden Feldern des Blocks.

Der erste Spielzug obliegt dem weißen Spieler. Danach wird abwechselnd gezogen. Kann ein Spieler keinen gültigen Zug machen, muss er aussetzen. Kann er hingegen ziehen, muss er dies auch tun (Zugzwang).

Bei einem gültigen Zug wird ein weiterer Stein in der Farbe des ziehenden Spielers auf dem Feld platziert. Für einen gültigen Zug müssen gegnerische Steine "umgedreht" werden. Es gilt, dass alle gegnerischen Steine umgedreht werden, die den neu gelegten Stein mit einem anderen Stein des ziehenden Spielers auf einer ununterbrochenen horizontalen, vertikalen oder diagonalen Linie verbinden. "Ununterbrochenen" heißt dabei, es dürfen auf den Verbindungslinien keine freien Felder oder Steine des am Zug befindlichen Spielers liegen. Einen Stein "umzudrehen" bedeutet, dass aus einem weißen Stein ein schwarzer bzw. aus einem schwarzen Stein ein weißer wird. Es können keine Steine geschlagen oder anderweitig vom Feld entfernt werden.

Das Spiel endet, wenn kein Spieler mehr einen gültigen Zug machen kann. Dies kann bereits der Fall sein, bevor das Feld voll belegt ist. Gewonnen hat der Spieler, der die meisten Steine seiner Farbe auf dem Feld hat. Bei Gleichstand endet das Spiel unentschieden.

2 Protokolldefinition des Gameservers

Ihr zu implementierender Client kann mit dem Gameserver über das hier beschriebene zeilenorientierte Protokoll kommunizieren. Zeilen werden, wie unter Unix üblich, mit einem "newline character" (`'\n'`) abgeschlossen. Der MNM-Gameserver ist wie folgt zu erreichen:

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

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

Wenn nicht innerhalb von im Gameserver festgelegten Zeitgrenzen auf Befehle geantwortet wird oder eine zu lange "Denkzeit" benötigt wird (s. u.), schickt der Gameserver:
S: - TIMEOUT ⟨⟨ *Begründung* ⟩⟩

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.

In ⟨⟨ ⟩⟩ eingeschlossene Werte werden obligatorisch durch die ihnen entsprechenden Werte ersetzt. Werte, die in *[[ ]]* eingeschlossen sind, geben optionale Werte an, d. h. sie können auch weggelassen werden.

Das Protokoll gliedert sich in die folgenden drei Phasen:

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

Die Phasen werden in den folgenden Unterkapiteln ausführlich beschrieben. Das Zustandsdiagramm in Abbildung 2 gibt eine grobe Übersicht über den Ablauf der drei Protokollphasen.

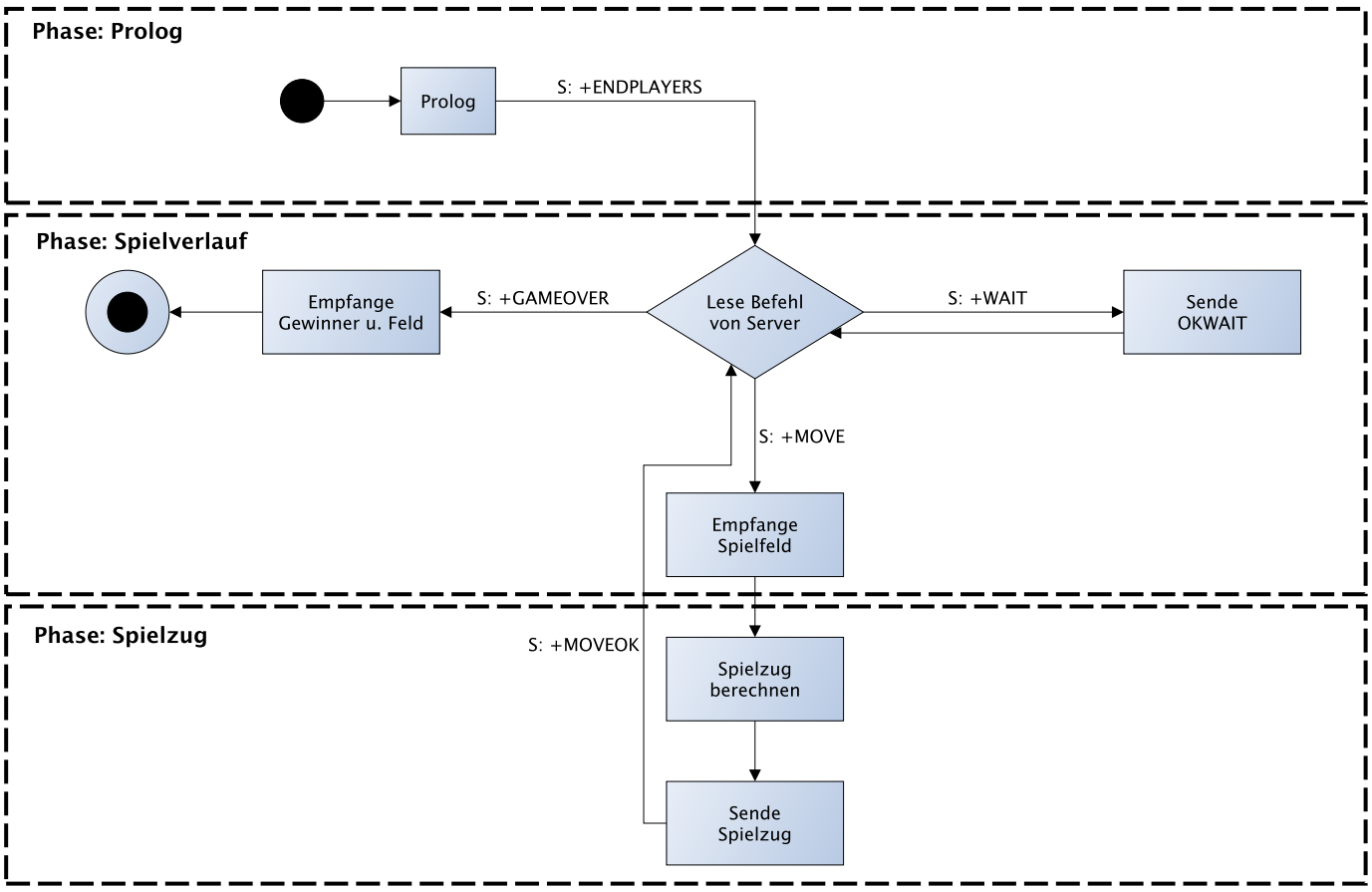

Abbildung 2: Protokollübersicht

2.1 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 akzeptiert einen Client, wenn die Major-Versionsnummern (links vom Punkt) von ⟨⟨ *Gameserver Version* ⟩⟩ und ⟨⟨ *Client Version* ⟩⟩ gleich sind. Der Gameserver setzt jeweils ein `v` voran. Die Major-Versionsnummer des Gameserver ist dieses Semester 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).

Da der Gameserver nicht nur Reversi spielen kann, teilt ⟨⟨ *Gamekind-Name* ⟩⟩ die Spielart mit. Ist diese anders als `Reversi`, muss der Client sich mit einer Meldung, die auf dieses Problem hinweist, beenden.

⟨⟨ *Game-Name* ⟩⟩ und ⟨⟨ *Spielername* ⟩⟩ können im Web-Interface beim Erstellen eines Spiels angegeben werden, bzw. enthalten Standartwerte, wenn die Angabe ausbleibt.

Wenn man einen Spieler mit einer bestimmten Nummer übernehmen möchte, kann dieser Wunsch mit *[[Gewünschte Spielernummer]]* angegeben werden. Lässt man die Spielernummer und das Leerzeichen nach `PLAYER` weg, so bekommt man einen freien Computerspieler vom Gameserver zugeteilt. Es ist zu beachten, dass der Spieler nicht sofort nach dem Abbruch der Verbindung (z. B. nach Absturz der Client-Software) wieder zur Verfügung steht. Es muss ein paar Sekunden gewartet werden, bevor der Gameserver den Computerspieler frei gibt.

Die ⟨⟨ *Spieleranzahl* ⟩⟩ wird bei einem Reversi-Spiel immer `2` sein.

Ist ein Spieler bereits verbunden, so ist ⟨⟨ *Bereit* ⟩⟩ `1`, ansonsten `0`. Auch wenn nicht alle Spieler bereit sein sollten, so wird mit der nächsten Protokollphase fortgefahren.

2.2 Protokollphase *Spielverlauf*

In dieser Phase können eine *Idle*, *Move* sowie eine *Game over* Befehlssequenz vorkommen. Wurde das erste Mal in die Spielverlaufsphase gewechselt oder eine der Befehlssequenzen beendet, so entscheidet die nächste Zeile (die im Diagramm an den Kanten annotiert ist) über die nächste abzuarbeitende Befehlssequenz.

2.2.1 *Idle* Befehlssequenz

S: `+ WAIT`
C: `OKWAIT`

`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 in Abbildung 2 zu entnehmen ist, bleiben die `WAIT`-Befehle z. B. in der Protokollphase *Spielzug* oder während der *Move* Befehlssequenzen aus.

2.2.2 *Move* Befehlssequenz

S: `+ MOVE` ⟨⟨ *Maximale Zugzeit* ⟩⟩
S: `+ FIELD` ⟨⟨ *Anzahl Spalten* ⟩⟩ `,` ⟨⟨ *Anzahl Zeilen* ⟩⟩

Die folgenden sechs Zeilen sind Beispielhaft für ein 6×6*-Feld:*
S: `+ 6 * * * * * *`
S: `+ 5 * * * * * *`
S: `+ 4 * * W B * *`
S: `+ 3 * * B W * *`
S: `+ 2 * * * * * *`
S: `+ 1 * * * * * *`
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. Beachten Sie, dass zwischen ⟨⟨ *Anzahl Spalten* ⟩⟩ und ⟨⟨ *Anzahl Zeilen* ⟩⟩ kein Leerzeichen ist. Zwischen `FIELD` und `ENDFIELD` wird der Spielbrettaufbau, für den ein Zug zu berechnen ist, in einer ASCII-Repräsentation ausgegeben. Das `+` steht dabei für eine positive Nachricht vom Server. Darauf folgt die Zeilennummer. Die restlichen ⟨⟨ *Anzahl Spalten* ⟩⟩ Elemente stehen für ein leeres Feld (`*`), einen weißen (`W`) oder schwarzen (`B`) Spielstein. Die oben genannten Elemente sind jeweils mit einem Leerzeichen getrennt.

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".

2.2.3 *Game over* Befehlssequenz

S: `+ GAMEOVER` *[[* ⟨⟨ `Spielernummer des Gewinners` ⟩⟩ ⟨⟨ `Spielername des Gewinners` ⟩⟩ *]]*
S: `+ FIELD` ⟨⟨ *Anzahl Spalten* ⟩⟩ `,` ⟨⟨ *Anzahl Zeilen* ⟩⟩

Die folgenden sechs Zeilen sind Beispielhaft für ein 6×6*-Feld:*
S: `+ 6 * * * * * *`
S: `+ 5 * * * * * *`
S: `+ 4 * * W B * *`
S: `+ 3 * * B W * *`
S: `+ 2 * * * * * *`
S: `+ 1 * * * * * *`
S: `+ ENDFIELD`
S: `+ QUIT`
⟨⟨ *Abbau der TCP-Verbindung durch Gameserver* ⟩⟩

Bei einem Unentschieden wird kein Gewinner angegeben. Die weiteren Zeilen geben den Spielzustand an, mit dem das Spielende erreicht wurde. Dieser wird an alle Spieler geschickt. Nach `QUIT` beendet der Server die Verbindung.

2.3 Protokollphase *Spielzug*

C: `PLAY` ⟨⟨ *Spielzug* ⟩⟩
S: `+ MOVEOK`

Ein ⟨⟨ *Spielzug* ⟩⟩ besteht schlicht aus der Koordinate, an der der neue Stein gesetzt werden soll. Die Koordinate wird in der Schachnotation angegeben, also z.B. `C3` oder `H11`. Sollte der übermittelte Zug nicht gültig sein, wird eine entsprechende Fehlermeldung übermittelt und die Verbindung getrennt. Ist absehbar, dass der Spieler erneut zum Zug aufgefordert wird (z. B. da erkannt wurde, dass der Gegner zugunfähig ist) so können mehrere Spielzüge mit `;` getrennt aneinandergehängt werden.