Netzwerkprotokoll V1.11

Änderungen (zum Beschreibung):
- *Nachrichtenbegrenzung*: bemerkt, dass jede Nachricht beendet bei NL (12. Mai).
- Beschreibung vom Test-Server hinzugefügt (12. Mai).
- *Noten zum Koordinatensystem* wurde hinzugefügt (1. Mai).
- *Einführung* und *Durch netcat testen* wurden hinzugefügt (28. April)

Einführung
Eure Spiel-Clients sollen zusammen spielen können. Um zusammen zu spielen, nehmen die Clients durchs Netz an einem Netzwerk-Spiel teil. Sie verbinden mit ein Spiel-Server der die Spielregeln implementiert, schicken zum Server Nachrichten, der z.B. Züge enthalten. Wenn ein Client eine unerlaubte Nachricht schickt, ist die Verbindung geschlossen und hat der Client verloren.

Der Server und der Client übertragen durch eine TCP/IP Verbindung. Ein Test-Spiel-Server läuft auf `vh12009.mathematik.uni-marburg.de` auf Port `34567`. Ein anderer Spiel-Server läuft auf Port `34568`. Die einige Unterschied ist, dass der Spiel-Server gibt keine Extra-Informationen mit der `YOUR_TURN` Nachricht (siehe die Beschreibung von `YOUR_TURN`). Ihr sollt am Ende beide Server unterstützen.

Die Nachrichten, die sie austauschen, enthalten Text (keine binär Daten). Deswegen könnt ihr das Protokoll durch *netcat ausprobieren, um das besser zu verstehen. Eure Aufgabe ist, dass* der Spiel-Client das Protokoll implementiert.

Durch *netcat* testen
Netcat ist einfach installierbar auf Linux durch eure „Package Manager“. Auf Windows kann man es herunterladen von:
http://www.rodneybeede.com/downloads/nc111nt_rodneybeede.zip
Nach der Installation kann man auf einen Kommandozeile-Prompt die folgenden Kommando durchführen:
```
nc vh12009.mathematik.uni-marburg.de 34567
```
Dann kann man Nachrichten zum Server schicken wie während der Unterricht oder wie in folgenden Beispielen.

Protokoll
Jede Nachricht beinhaltet genau einen Befehl.
Mit TCP/IP selbst gibt es keine Begrenzung der Nachrichtenlänge, deswegen hat dieses Protokoll eine eigene Definition der Begrenzung.

Nachrichtenbegrenzung
Jede Nachricht beginnt mit seiner Länge in Bytes. Zum Beispiel:
```
8
HELLO C
```

Hierbei ist 8 die Länge von `"HELLO C\n"`. Die ganze Nachricht ist deswegen in Java beschrieben als `"8\nHELLO C\n"`; `"8 \nHELLO C\n"` wäre schon falsch wegen des zusätzlichen Leerzeichens. Passt auf, dass ein Charakter (Java `char`) kann auf mehrere Bytes bestehen.
Note: Jede Nachricht muss mit `"\n"` enden, und dieses Charakter soll in der Länge der Nachricht gerechnet werden.

Richtung der Nachricht
CS = Client → Server
SC = Server → Client
Diese Bezeichnung steht in diesem Dokument zu Kommentarzwecken, um verständlich zu machen, in welcher Richtung die Kommunikation abläuft. Zwischen Client und Server werden keine Nachrichten „SC“ oder „CS“ gesendet.

Auflistung der Befehle (Beschreibung siehe unten)
Note: NL ist ein Zeilenumbruch (NewLine).
SC: WELCOME_VERSION String
CS: HELLO PlayerId
CS: LIST_GAMES
SC: GAMES Seq[GameId]
CS: CREATE N_Players GameId
CS: JOIN GameId

SC: OK boardSize nPlayers nPlayersAlreadyJoined

SC: GAME_STARTUP Seq[PlayerId]
SC: NEW_TILE Tile

SC: YOUR_TURN NL Option[Board] NL Option[Score] NL Seq[Tile]
CS: MOVE NL PlacedTile NL Renew
SC: WIN PlayerId
SC: OTHER_PLAYER_TURN NL PlacedTile NL PlayerId
SC: ERROR String

Beschreibung des Ablaufs
Nach Aufbau der Verbindung schickt der Server eine WELCOME Nachricht mit der Version des Netzwerkprotokolls:
SC: WELCOME_VERSION String
Aktuell ist die Version V1.11, deswegen schickt der Server:
WELCOME_VERSION V1.11

Diese Nachricht schickt der Server, damit der Client erkennt, ob er die richtige Version implementiert. So versteht der Client, ob er mit diesem Server kommunizieren kann, oder ob man den Quelltext ändern muss.

Mit dem ersten Nachricht meldet sich der Client beim Server an
CS: HELLO PlayerId
Der Client erhält vorm Server darauf keine Antwort.

Danach kann der Client optional die Liste der verfügbaren Spiele abfragen:
/* Optional: */
CS: LIST_GAMES
Antwort:
SC: GAMES Seq[GameId]

Der Client kann entweder ein neues Spiel erzeugen (CREATE), oder einem bestehendem Spiel beitreten (JOIN):
/* Either of: */

CS: CREATE N_Players GameId //Der Client tritt automatisch dem Spiel bei
CS: JOIN GameId //GameId muss sich auf ein bestehendes Spiel beziehen

Antwort: entweder
SC: OK boardSize nPlayers nPlayersAlreadyJoined
oder
SC: ERROR String

Wenn der Client eine falsche Nachricht an den Server schickt, antwortet der Server mit einer ERROR Nachricht.
Der String ist die Fehlernachricht. Nach dieser Nachricht hat der Spieler verloren und die Verbindung wird vom Server abgebrochen.

Wenn ein Spiel voll ist, also entsprechend viele Spieler beigetreten sind wie vorgegeben, dann startet der Server das Spiel und sendet an alle Clients eine Nachricht, mit der alle Spielernamen bekanntgegeben werden (z. B. für die Darstellung in der GUI):
SC: GAME_STARTUP Seq[PlayerId]

Danach teilt der Server dem Spieler 6 Steine aus. Für jeden Stein erhält der Client eine eigene Nachricht.
SC: NEW_TILE Tile

Der Server vergibt reihum das Zugrecht an die Spieler und informiert über gültige Züge anderer Spieler
Für jeden Spieler, der am Zug ist, wird die folgende Sequenz so lange wie notwendig wiederholt (im Fall von Bonus-Zügen)
{
1. SC: YOUR_TURN NL Option[Board] NL Option[Score] NL Seq[Tile]
//Alternativ 0 Steine oder alle verfügbaren Steine (6 bzw bei Ausführung von Bonus-Zügen auch weniger)
2. CS: MOVE NL PlacedTile NL Renew //Renew wird unter beschrieben
}

Wenn der Client dem Server einen nicht erlaubten Zug schickt, bekommt der Client eine ERROR Nachricht, wie oben angegeben.
Der Test-Spiel-Server schickt alle optionelle Informationen mit der YOUR_TURN Nachricht; die sind nur für Debugging gegeben. Der andere Spiel-Server schickt sie nicht. Deswegen schickt er nur Nachrichten wie:
YOUR_TURN
0
0
0

Sonst schickt der Server entweder eine neue YOUR_TURN Nachricht wie oben, oder der Server schickt genug NEW_TILE Nachrichten bis der Client wieder 6 Steine hat, oder er bekommt eine WIN Nachricht, falls das Spiel beendet ist:
SC: WIN PlayerId

Nachdem der Spieler genug NEW_TILE Nachrichten bekommen hat, ist er nicht mehr dran. Nur in diesem Zustand kann er OTHER_PLAYER_TURN Nachrichen bekommen.
Wenn ein Spieler einen gültigen Zug spielt, schickt der Server allen anderen Spielern eine OTHER_PLAYER_TURN Nachricht:

SC: OTHER_PLAYER_TURN NL PlacedTile NL PlayerId

Auch nach OTHER_PLAYER_TURN kann man die WIN Nachricht erhalten.

Neue Spielsteine nachziehen
Um Steinaustausch zu bitten benutzt man Renew im der MOVE Nachricht (siehe dort):
Renew ::= Seq[Color]
Wenn diese Sequenz leer ist, bedeutet das, dass der Spieler keinen Austausch der Steine wünscht, das ist immer eine gültige Möglichkeit.
Wenn diese Sequenz nicht leer ist, drückt es die Bitte aus, alle Steine auszutauschen. Das ist nur erlaubt, wenn alle Bonus-Züge fertig sind; der Spieler muss alle Farben angeben, für die er den niedrigsten Punktestand hat. Er darf dann auch keine Steine mit den betreffenden Farben mehr besitzen.
Wenn die Bitte für Austausch nicht gültig ist, verliert der Spieler und die Verbindung ist abgebrochen.

Syntax-Beschreibung der Nachrichten
Definitionen werden eingeleitet mit ::= (vgl Backus-Naur Form).

NL ::= ein Zeilenumbruch (Newline), "\n". Normalerweise gibt es ein Leerzeichen zwischen zwei Elementen einer Nachricht; bei einem Newline gibt es keine Leerzeichen vorher oder nachher.
Die Unterschied zwischen Newline und Leerzeichen ist fast nie wichtig beim Lesen. Deswegen könnt ihr java.util.Scanner benutzen und seine Methoden nextInt(), next(String), findInLine(Pattern) und skip(String).

String ::= // Eine Folge aller Zeichen bis zum Zeilenende
N_Players ::= Int // Anzahl der Spieler: Von 1 bis 4
Color ::= Int // Zwischen 0 und 5. Farbe-Code: RedStar=0, GreenCircle=1,
  // BlueStar=2, OrangeHex=3, YellowSun=4, PurpleCircle=5
PlayerId ::= String //Name des Spielers, z. B. „Spieler 1“
GameId :: = String //Name des Spiels, z. B. „Stupid_Game“
Tile ::= Color Color //z. B. 5 0
Cell ::= (x:Int) (y:Int) Color //z. B. 5 0 2
PlacedTile ::= Cell NL Cell

Ein Beispiel für PlacedTile:
```
0 -1 0
1 -1 5
```

Es gibt parametrische Definitionen mit T als Parameter, also entweder 0 NL oder 1 T NL:
Option[T] ::= 0 | 1 T

ein Beispiel für Option[Cell] :
```
1 0 -1 4
```
ein weiteres Beispiel:
```
0
```

Sequenzen variabler Anzahl n mit T als Parameter:
Seq[T] ::= (n:Int) NL (T NL){n}
Ein Integer n für die Anzahl, danach wird n mal T wiederholt, jeweils getrennt durch NL

Ein Beispiel für Seq[Cell] ist:

```
0
```
Ein anderes Beispiel:
```
1
0 -1 4
```
Noch ein Beispiel:
```
2
0 -1 4
-1 1 3
```

Sequenzen fester Anzahl n mit T als Parameter
FixedSeq[T](n:Int) ::= (T){n} //T wird n mal wiederholt, getrennt durch Leerzeichen

Score ::= FixedSeq[Int](6) //z. B. 5 0 2 6 18 10
Board ::= (boardSize:Int) NL Seq[PlacedTile]

boardSize steht hier für die Spielfeldgröße (5, 6 oder 7)

Noten zum Koordinatensystem:
Wenn die Spielfeldgröße ist z.B. 5, sind z.B. gültig diese Positionen: (-5, 5), (0, 5), (5, 0), (5, -5), (0, -5), (-5, 0), die sind an den Grenze des Gitters. Dieselbe Zellen haben schon am Anfang ein aufgedrücktes Symbol (auch wenn die Spielfeldgröße ist 6 oder 7). Diese Symbolen haben jeweils als Farbe Rot, Grün usw., in derselben Reihenfolge als dem Farbe-Code.
Wenn die Spielfeldgröße ist *r*, ein Feld (*x*, *y*) ist im Spielfeld wenn alle diese Bedingungen gelten:

$|x| \leq r$
$|y| \leq r$
$|x+y| \leq r$

Beispiel Verbindungs-Transkript
"CS:" und "SC:" stehen hier als Kommentar vor der Nachricht, mit der selben Bedeutung wie oben. Leere Zeilen sind zur besseren Lesbarkeit hinzugefügt.

CS:
```
12
HELLO Paolo
11
LIST_GAMES
```

SC:
```
17
GAMES 1
THE_GAME
```

CS:
```
14
CREATE 1 GAME
```
[//Oder, für andere Spieler die mitspielen möchten:
CS:
```
14
JOIN THE_GAME
```
Aber dieses Spiel braucht 4 Spieler, deswegen bekommt man erst dann eine Antwort, wenn auch insgesamt 4 Spieler beigetreten sind.
]

SC: //Jetzt schickt der Server mehrere Nachrichten. Sie werden nur durch die Byte-Anzahlen begrenzt.

```
9
OK 5 1 0
21
GAME_STARTUP 1
Paolo
13
NEW_TILE 3 2
13
NEW_TILE 5 0
13
NEW_TILE 4 2
13
NEW_TILE 5 2
13
NEW_TILE 4 3
13
NEW_TILE 0 0
56
YOUR_TURN
1 5
0
1 0 0 0 0 0 0
6
3 2
5 0
4 2
5 2
4 3
0 0
```

CS:

```
19
MOVE
3 2 2
3 3 5
0
```

SC:

```
69
ERROR Illegal move Tile((Pos(3,2),BlueStar),(Pos(3,3),PurpleCircle))
```