Übung 4: Networking (40 Punkte)

Aufgabenstellung

Wir wollen in dieser Übung mit `ServerSocketChannels`, `SocketChannels` und `Selectors` eine Client/Server-Lösung für ein Spiel realisieren, bei dem sich (viele) Clients auf einer Szene bewegen können. Abbildung 1 zeigt die Ausgabe eines Clients. Seine Position ist mit einem roten Punkt und seinem Namen dargestellt. Er kann sich mittels der Buttons in die vier Himmelsrichtungen bewegen. Die weiteren Punkte stellen die anderen Spieler dar.

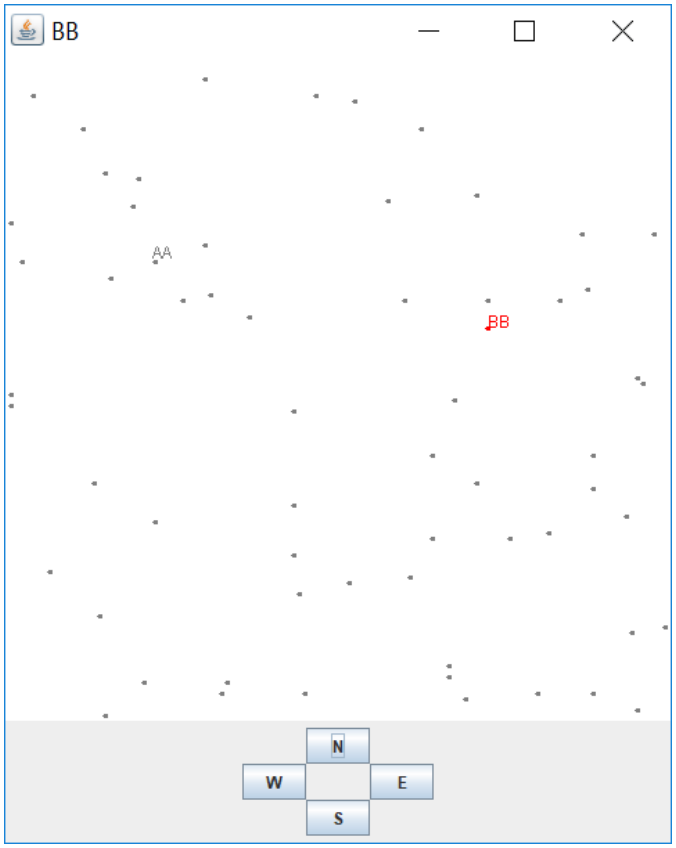

Abbildung 1

Der Programmablauf soll folgend gestaltet sein:

Anmelden eines Clients über einen eindeutigen Namen. Nachdem sich der Client mit dem Server verbunden hat, schickt er eine Login-Message zur Anmeldung zum Server. Der Server bestimmt eine Position für den neuen Spieler und schickt eine Bestätigung zum Client. Der Client wird von den Positionen aller Mitspieler informiert und alle Mitspieler erhalten eine Message mit der Position des neuen Spielers. Ist der Namen schon vergeben, wird das Anmelden abgelehnt und die Kommunikation sofort beendet.

Bewegung eines Spielers: Der Client schickt eine Bewegungsanforderung mit der gewünschten Richtung (Nord, Ost, Süd, West) an den Server. Der Server bestätigt die Bewegung. Die neue Position dieses Spielers wird an alle Mitspieler gesendet.

Abmelden eines Clients: Der Client will sich abmelden (Beendigung des Clientprogramms) und schickt eine Meldung an den Server. Der Server bestätigt die Abmeldung, entfernt ihn in aus der Menge der Spieler und schickt die Information, dass der Spieler sich abgemeldet hat, an alle Mitspieler.

Anmerkungen/Erweiterungen:

Realisieren Sie am Server ein einfaches Verfahren zur Repräsentation der Spieler: Jeder Spieler soll einen eindeutigen Namen und eine aktuelle Position (int-Koordinaten x/y) haben. Eine Bewegung soll immer nur um eine Koordinate erfolgen. Dabei ist ein zu lösendes Problem, was passieren soll, wenn sich ein Spieler über den Spielrand hinausbewegen würde. Eine Strategie ist, einen geschlossenen Torus anzunehmen, d.h., die Position *Spielrand* + 1 ist die Position 0. Entsprechende Umsetzungsdetails sind Ihnen überlassen.

Der obige Spielablauf stellt eine Minimalanforderung dar. Sie können/sollen das Programm erweitern, dass ein wirkliches Mehrpersonenspiel realisiert wird (**ist aber für die LVA nicht gefordert**). Hier einige Anregungen:

Mehrere Arten von Spielern: Mit mehreren Arten von Spielern kann man gegnerische Mannschaften bilden. Die Zugehörigkeit zu einer Mannschaft sollte man beim Login festlegen. Tatsächlich gibt es bei dem Spiel wie in Abbildung 1 dargestellt zwei unterschiedliche Spieler, die mit und die ohne Namen, wobei die große Zahl der Spieler ohne Namen durch autonome Agenten gesteuert werden, die zufällig die Richtung wechseln.

Ziel und Erfolg: Es kann natürlich auch ein Spielziel realisiert werden, z.B. Fangen der Spieler der einen Partei durch die andere Partei. Hier sind am Server entsprechende Spielzüge zu realisieren und entsprechende Messages auszutauschen.

Eingeschränktes Sichtfeld: Eine sinnvolle Variante wäre, dass Spieler nur ein eingeschränktes Sichtfeld haben. Damit muss man auch nur Positionsdaten von den Mitspielern austauschen, die sich innerhalb des Sichtfelds befinden. Abbildung 2 zeigt, wie das dann ausschauen könnte.

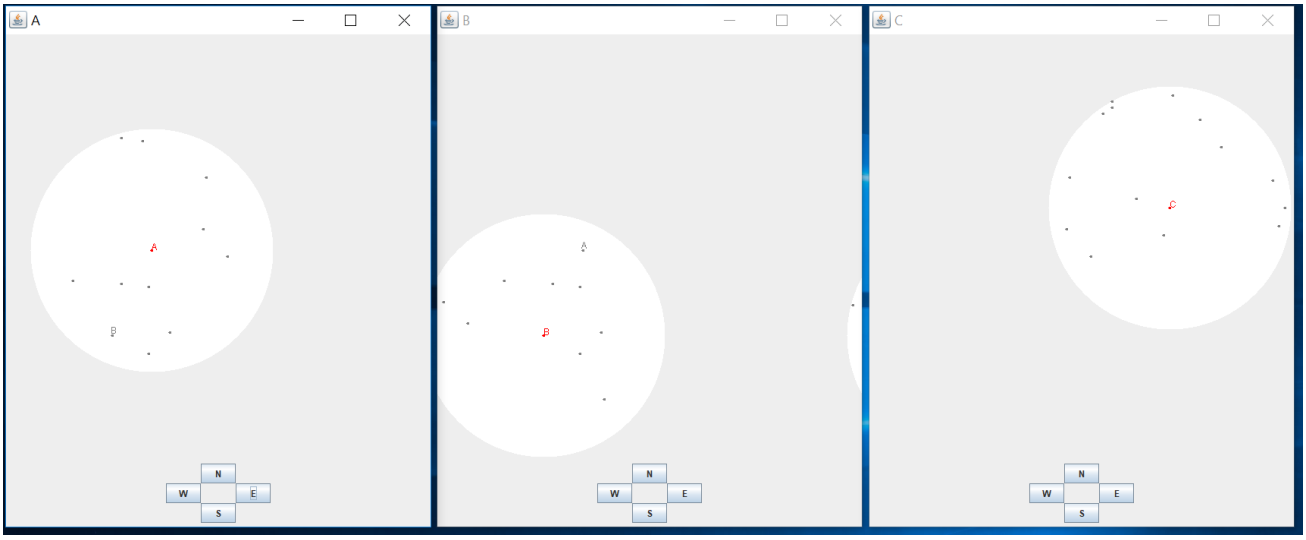

Abbildung 2: Drei Clients mit eingeschränktem Sichtfeld

Technische Anforderungen

Technischer Lösungsansatz für Server

Server: Das Programm ist mit nicht-blockierenden `SocketChannel`s und unter Verwendung eines `Selector`s zu lösen. Sowohl der Verbindungsaufbau (`accept`) als auch das Lesen der Messages soll durch einen `Selector` erkannt und dann direkt behandelt werden. Das heißt, dass sowohl der Verbindungsaufbau als auch die Kommunikation mit den Clients in einem einzigen Thread passieren soll. Damit ist auch keine Thread-Synchronisation notwendig.

Client: Am Client ist eine einfache Swing-Anwendung zu realisieren, die die Positionen der Spieler darstellen kann und Bewegungskommandos umsetzt. Zu Beginn soll der Name des Spielers eingelesen werden (am besten mit `JOptionPane.showInputDialog`). Das Schreiben der Kommandos erfolgt ausgehend von den Benutzeraktionen (Button-Clicks). Das Lesen der Antworten muss in einem eigenen Thread erfolgen. Das Zeichnen passiert im AWT-Thread. Damit sind am Client die Zugriffe auf gemeinsame Daten entsprechend thread-safe zu gestalten.

Architektur: Abbildung 3 zeigt die Architektur des Systems. Am Server gibt es einen Thread für die Reaktion des Selectors auf einkommende Connections und Inputs von Clients. Der ServerSocketChannel wird beim Selector registriert. Bei erkannten Verbindungen müssen die Channels zum Client registriert werden. Bei einem durch den Selector erkannten Input-Ereignis soll die Kommunkation mit dem Client erfolgen.

Am Client gibt es EventHandler für UI Ereignisse, die die Kommandos zum Server schreiben. Das Empfangen der Antworten vom Server soll in einem separaten Thread (*CommThread*) erfolgen.

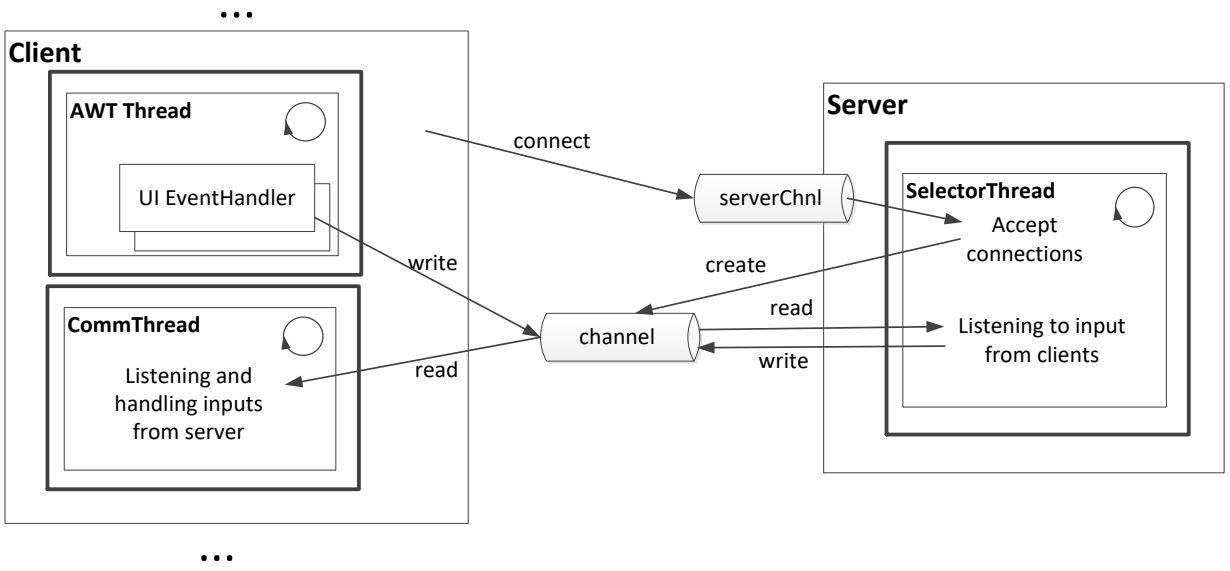

Abbildung 2: Architektur

Hinweise:

Achten Sie auf das korrekte Schließen der Channels.

Achten Sie auf das korrekte Beenden der Client- und Server-Programme.

Beachten Sie, dass in Übung 5 die gleiche Anwendung mit RMI realisiert werden soll.