P4: Client Server

Abgabetermin: 22. Mai 2025

Punktzahl: 23 + 5*

In diesem Projekt[1] wird das Thema Client Server behandelt. Sie sollen hierzu eine Anwendung erstellen, mit der Sie das Spiel Swap [2] über eine Client-Server Verbindung spielen können. Bei diesem Spiel tauscht der Spieler bei jedem Zug zwei Würfel um damit bestimmte Kombinationen von Würfeln zu bilden. Je mehr Kombinationen (Sequenzen) dabei gebildet werden und je länger diese sind, desto mehr Punkte gibt es. Die längste Sequenz, die auch am meisten Punkte bringt, besteht aus 6 Würfeln und wird als "Swap" bezeichnet. Das Spiel ist beendet, wenn der Spieler 40 Punkte erreicht.

Die genauen Spielregeln, insbesondere die Berechnung der Punkte und die Erklärung, was ein *Swop* ist, entnehmen Sie der beigefügten Spieleanleitung. Hierbei ist nur das Grundspiel von Bedeutung. Darüberhinaus gibt es in dieser Anwendung auch kein separates Brett für die erreichten Punkte.

Zu Beginn wird serverseitig ein Spielfeld von 66 farbigen Würfeln erstellt, deren Farbe und Nummer zufällig gewählt werden. Clientseitig tauscht der Spieler jeden Zug zwei Würfel miteinander, dessen Positionen dann (nach Betätigen des Submit Buttons) an den Server geschickt werden. Der Server überprüft, wie viele Punkte durch das Tauschen erreicht wurden und schickt das Ergebnis zurück an den Client. Sobald (mindestens) 40 Punkte erreicht sind, ist das Spiel beendet.

Importieren Sie zunächst die Archivdatei **p4-vorgaben.zip** in Eclipse (als existierendes Projekt aus einem Archivfile). Ändern Sie dann den Namen des Projekts in *<IhrNachname.IhrVorname.P4>*

Vorgaben

Das importierte Projekt definiert bereits eine Grundstruktur, die Sie für Ihre Implementierung verwenden sollen. In den Vorgaben sind 5 Packages vorgegeben, die nachfolgend kurz beschrieben werden.

[1] Das Projekt wurde in einer Seminararbeit entwickelt.

[2] siehe beigefügte Datei *swap.pdf*

main

In diesem Package befinden sich die beiden Klassen **ServerMain** und **ClientMain**, mit denen die Server- und die Client-Applikation gestartet werden können.

data

Dieses Package beinhaltet 4 Klassen:

- Der Aufzählungstyp **Coloring** definiert 11 Farben für die Würfel **(RED, GREEN, BLUE, CYAN, MAGENTA, YELLOW, BLACK, WHITE, GRAY, PINK, ORANGE)**.

  Für eine Ausgabe einer Konstante ist die **toString** Methode überladen. Darüberhinaus bietet die Methode **getColor** die Möglichkeit, zu einer Konstanten die zugehörige Instanz der Klasse **Color** zu erhalten. Umgekehrt generiert die Methode **getColoringByString** aus einem String die zugehörige `Coloring`-Konstante und die Methode **getColoringByColor** aus einer Instanz der Klasse **Color** die zugehörige `Coloring`-Konstante.

- Die Klasse **Dice** repräsentiert einen Würfel auf dem Spielbrett. Neben den Attributen (u.A. für die Farbe, die Würfelzahl und die Position) sowie einem Konstruktor und Getter- bzw. Settermethoden implementiert die Klasse hauptsächlich Methoden, die vom Client genutzt werden, um die Würfel zu zeichnen.

- Die Klasse **Pair** repräsentiert ein Paar aus der Farbe und der Nummer (Augenzahl) eines Würfels. Die Farbe ist eine Instanz der Klasse **String** und die Nummer ist eine `int`-Variable. Diese Klasse implementiert bereits das Interface `Serializable`, um Informationen eines Würfels an den Client zu schicken, der mit diesen Informationen die Würfel zeichnet bzw. das Spielfeld aktualisiert.

- Die Klasse **DicePanel** repräsentiert eine vereinfachte Version des Spielfelds. Es speichert die 66 Würfelinstanzen (auf der Serverseite) in einem zweidimensionalen Array mit der Dimension 6x11. Eine Instanz der Klasse **DicePanel** wird zu Beginn einer Verbindung mit dem Server im Konstruktor der Klasse **ServerProtocol** erstellt und kann später von Ihnen genutzt werden (um Punkte zu prüfen oder den Inhalt des Spielfelds an den Client zu schicken). Für die Übertragung des Spielfelds vom Server an den Client wird dieses zeilen- und spaltenweise in einer ArrayListe aus Paaren (Klasse **Pair**) gespeichert. Der Client nutzt diese Liste später, um die Würfel und somit das Spielfeld zu zeichnen. Die Liste mit den Informationen kann man durch die Methode **getList** erhalten. Die Methode **checkPoints** wird dazu verwendet, die Punkte auszurechnen, die durch das Tauschen zweier Würfel erreicht wurden. Sie nutzt dazu die x- und y-Koordinaten der zwei Würfel und gibt die Anzahl der Punkte zurück. Die Methode aktualisiert auch die Liste der Paare, die den Inhalt des Spielbretts repräsentieren und die vom Server verschickt wird.

message

Dieses Package beinhaltet neben dem Aufzählungstyp **MessageType** die abstrakte Klasse **Message** sowie alle davon abgeleiteten Nachrichtenklassen. Einen Teil der Nachrichten sollen Sie in Aufgabe 1 erstellen. Im Folgenden sind die bereits vorgegebenen Klassen des Packages aufgeführt.

- Der Aufzählungstyp **MessageType** definiert die hier benötigten Nachrichtentypen. Aktuell gibt es nur den Typ **DISCONNECT**, die anderen Typen müssen Sie später hier ergänzen.
- Die Klasse **Message** bildet die abstrakte Basisklasse für alle Nachrichten, die vom Client oder Server verschickt werden. Der einzige Konstruktor benötigt den Messagetype als Parameter.
- Die Klasse **DisconnectMessage** ist eine Beispiel-Nachrichtenklasse, die sowohl vom Server als auch vom Client verwendet wird, um dem Kommunikationspartner anzukündigen, das die Verbindung geschlossen wird.

server

Dieses Package beinhaltet die folgenden drei Klassen, die für die Implementierung der serverseitigen Kommunikation benötigt werden:

- Die Klasse **Server** stellt einen Multiserver dar, der quasi parallel mehrere Clients versorgen kann. In der Methode **listen** wird der Server Socket erstellt und der Server wartet in einer Schleife auf Anfragen von Clients. Alle Verbindungen zu Clients werden dabei in einer Liste gesammelt, die als Attribut (mit dem Namen **clientConnections**) in der Klasse vorgegeben ist.
- Die Klasse **ServerUI** stellt eine einfache GUI bereit, bei der der Server mit Hilfe von zwei Buttons gestartet bzw. gestoppt werden kann.
- Die Klasse **ServerProtocol** repräsentiert den serverseitigen Teil des Protokolls. Einige der Methoden sind dabei von Ihnen in Aufgabe 2 noch zu ergänzen.

client

Dieses Package umfasst die folgenden beiden Klassen, die für die Implementierung der clientseitigen Kommunikation benötigt werden:

- Die Klasse **ClientUI** stellt die Benutzeroberfläche zur Verfügung, die zu Beginn ein Panel mit einem einzelnen Button mit der Aufschrift *"Connect"* zeigt. Wenn man dann auf den Button klickt, wird die Verbindung zum Server hergestellt. Ist dieser Vorgang erfolgreich, wird ein Fenster angezeigt, das ein Spielbrett mit 66 Würfeln zeigt. Außerdem sind 2 Labels zu sehen,

die die aktuelle Punktzahl und die zu erreichende Punktzahl darstellen. Die Würfel sind als Buttons realisiert. Zum Auswählen muss man sie anklicken. Über den Button mit der Aufschrift "Submit", der erst anzuklicken ist, wenn genau 2 Würfel ausgewählt sind, werden die Positionen der zwei ausgewählten Würfel an den Server geschickt. Die Punktzahl, die der Server berechnet und an den Client zurück schickt, wird auf die bisherige Punktzahl aufsummiert und dann mit dem Label mit dem Titel *"Current Points"* angezeigt.

Am Ende des Spiels (wenn der Spieler 40 Punkte erreicht hat) wird eine Nachricht angezeigt, die den Erfolg des Spielers signalisiert. Die Verbindung zum Server wird abgebrochen und der Startbildschirm wieder angezeigt.

Die Klasse **ClientUI** besitzt ein Attribut vom Typ **ClientProtocol**, mit dessen Hilfe von der GUI aus die Verbindung erstellt bzw. beendet werden kann. Hierzu werden in den Methoden **connectToServer** und **disconnect** der Klasse **UI** der Konstruktor der Klasse **ClientProtocol** aufgerufen sowie die Methoden **isAlive, isOK, start** und **disconnect** dieser Klasse. Darüberhinaus wird von der GUI aus noch die Methode **sendDicesPosition** aus der Klasse **ClientProtocol** aufgerufen, mit der die Positionen der 2 ausgewählten Würfel an den Server geschickt werden sollen und die Sie in Aufgabe 3 implementieren müssen.

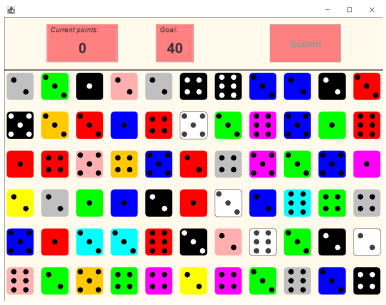
Folgende Methoden aus der Klasse **ClientUI** müssen Sie darüberhinaus später (von der Klasse **ClientProtocol** aus) aufrufen:

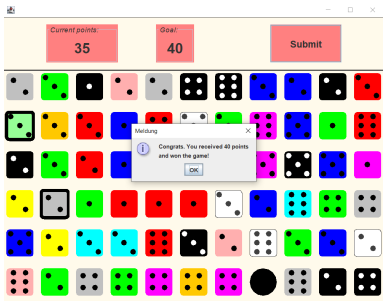
- Mit der Methode **setDicesList** wird die Liste der Paare mit den Informationen der Würfel des gesamten Spielfelds in der GUI eingetragen. Aus den Elementen dieser Liste werden Instanzen der Klasse **Dice** erstellt und ein Spielfeld mit 6x11 Würfeln gezeichnet.
- Mit der Methode **setPoints** werden die Punkte in der GUI eingetragen. Das sind dann die Punkte, die bei Aktualisierung der GUI auf dem Label mit dem Titel *"Current points"* dargestellt werden.
- Die Methode **createGameUI** dient dazu, die Benutzeroberfläche aktualisiert darzustellen.
- Die Methode **showWinningMessage** dient dazu, ein Dialogfenster mit einer Meldung anzuzeigen, die den Erfolg des Spielers anzeigt und vom Spieler quittiert werden muss. Anschließend wird das Spiel mit Hilfe der Methode **resetUI** reinitialisiert.
- Die Methode **resetUI** reinitialisiert die GUI und sorgt dafür, dass wieder der Startbildschirm angezeigt wird.

- Die Klasse **ClientProtocol** beinhaltet die clientseitige Kommunikation mit dem Server. Die Klasse ist als Thread angelegt und muss von Ihnen in Aufgabe 3 vervollständigt werden.

Um den Server zu starten, müssen sie die Klasse **ServerMain** ausführen und auf den Start-Button klicken.

Um den Client zu starten, müssen sie die Klasse **ClientMain** ausführen und dann auf den Connect-Button klicken. Die folgenden Bilder zeigen die GUI nach dem Verbinden mit dem Server und einer erfolgreichen Initialisierung der Nachrichtenkommunikation über den Start des Spiels sowie am Ende eines Spiels.





Aufgabenstellung:

Vervollständigen Sie nun die Anwendung wie im Folgenden beschrieben:

1 Nachrichtenklassen schreiben 7P

Legen Sie im Package **messages** drei neue Nachrichten-Klassen an, die von der Klasse **Message** abgeleitet sind. Definieren Sie für jeden der Klassen einen eigenen Typ in der Klasse **MessageType** und statten Sie diese für ihren jeweiligen Zweck (Transport bestimmter Daten) aus. Benennen Sie dabei die Nachrichtentypen- und Klassen sinnvoll. Schreiben Sie jeweils einen Konstruktor, der die transportierten Daten als Parameter besitzt und den Konstruktor der Oberklasse nutzt um den zugehörigen Typ zu initialisieren.

- Die erste Nachrichtenklasse soll `StartMessage` heißen und den Start signalisieren. Der Server schickt hierbei eine Liste von Paaren (Klasse **Pair**) an den Client, sodass dieser die Würfel darstellen kann.
- Die zweite Nachrichtenklasse soll `CheckMessage` heißen und die Koordinaten von zwei (vom User) ausgewählten Würfeln transportieren, sodass die erreichten Punkte geprüft werden können.
- Die dritte Nachrichtenklasse soll `ResultMessage` heißen und die Ergebnisse einer Auswertung, also die Anzahl der erreichten Punkte und dazu die (aktualisierte) Liste der Würfel transportieren.

2 Serverseitiges Protokoll implementieren 8P

Implementieren Sie nun den serverseitigen Anteil der Kommunikation in der Thread-Klasse **ServerProtocol**. Folgende Attribute müssen Sie hierbei verwenden:

- Die Ein- und Ausgabeströme (Attribute **fromClient** und **toClient**) werden durch den Konstruktor geöffnet.
- Das Attribut **running** signalisiert, ob das Protokoll noch läuft.
- Das Attribut **dicePanel** gibt die Instanz der Klasse **DicePanel** an, die das Spielfeld darstellt. Dieses kann dazu verwendet werden, um auf die Liste mit den Würfelpaaren zuzugreifen und, nach einem Würfeltausch, die Punkte zu berechnen.

(a) Implementieren Sie die Methode **sendMessageToClient**: Senden Sie das als Parameter übergebene Nachrichtenobjekt an den Client. Leeren Sie den Ausgabestrom und sorgen Sie dafür, dass keine Verbindung mehr zu dem Objekt besteht, welches in den Strom gestellt wurde.[3] Falls es zu einem Fehler kommt, sorgen Sie dafür, dass das Protokoll beendet wird und geben Sie eine Fehlermeldung aus.

(b) Implementieren Sie die Methode **sendListToClient**, die die bereits in der Klasse **DicePanel** verwaltete Liste von Paaren, welche den Inhalt des Spielfelds repräsentiert, an den Client senden soll. Nutzen sie hier die Methode **sendMessageToClient**, die sie in (a) implementiert haben. Erst dann, wenn diese Methode erfolgreich implementiert wurde, kann die GUI mit dem Spielfeld angezeigt werden.

(c) Implementieren Sie nun die **run**-Methode der Klasse **ServerProtocol**. Solange das Protokoll aktiv ist, soll der Server in einer Schleife eine Nachricht von dem zugeordneten Client einlesen und wie folgt darauf reagieren:

- Wenn der Client eine Nachricht mit den Koordinaten zweier Würfel schickt, soll die erreichte Punktzahl mit Hilfe der Methode **checkPoints** der Klasse **DicePanel** berechnet werden und das Ergebnis an den Client geschickt werden. Außerdem soll die aktualisierte Liste der Würfel an den Client geschickt werden.
- Wenn der Client eine Nachricht vom Typ **DISCONNECT** geschickt hat, soll die vorgegebene Methode **removeConnection** aufgerufen werden. Das gleiche soll passieren, wenn der Client irgendeine andere Nachricht als die bisher genannten schickt.

Falls ein Fehler bei der Kommunikation auftritt, soll der Fehler in der Konsole ausgegeben werden. Handelt es sich um ein Ein- oder Ausgabefehler,

[3] siehe Hinweis

soll die Methode **removeConnection** aufgerufen werden. In allen anderen Fällen soll der Methoden-Aufrufstapel ausgegeben werden.

Um Nachrichten an den Client zu versenden, verwenden Sie die in Teil (a) implementierte Methode.

3 Clientseitiges Protokoll implementieren 8P

Vervollständigen Sie nun noch den clientseitigen Teil der Kommunikation in der Klasse **ClientProtocol**. Folgende Attribute müssen Sie hierbei verwenden.

- Die Ein- und Ausgabeströme (Attribute **fromServer** und **toServer**) werden durch den Konstruktor geöffnet.
- Das Attribut **running** signalisiert, ob das Protokoll noch läuft und wird am Anfang mit `true` initialisiert.
- Das Attribut **points** gibt die Anzahl an Punkten an, die der Spieler bereits erreicht hat.
- Das Attribut **gui** ermöglicht Ihnen, die Methoden der GUI aufzurufen.

(a) Implementieren Sie die Methode **sendDicesPosition**, mit der der Client eine Nachricht mit den als Parameter übergebenen Positionen der zwei auszutauschenden Würfel an den Server sendet. Sorgen Sie dafür, dass Objekte, die in den Ausgabestrom gestellt werden, anschließend nicht mehr referenziert werden.[4] Überprüfen Sie mit Hilfe der Methode **isOK** vor dem Verschicken der Nachricht, ob die Verbindung existiert und nicht geschlossen ist. Im Fehlerfall soll die Verbindung zum Server jeweils mit Hilfe der Methode **closeConnection** geschlossen werden.

(b) Implementieren Sie nun die **run**-Methode der Klasse **ClientProtocol**. Diese Methode soll in einer Schleife (die solange läuft, bis das Protokoll beendet werden soll) zunächst eine Nachricht vom Server einlesen und dann, je nach Typ der Nachricht, wie folgt reagieren:

- Wenn der Client eine Nachricht mit der Liste der Würfel erhält, signalisiert dies, dass das Spiel startet. In diesem Fall soll die Liste in der GUI (Attribut **gui**) mit Hilfe der Methode **setDicesList** eingetragen werden. Verwenden Sie anschließend die Methode **createGameUI**, um das Spielbrett mit den Würfeln anzuzeigen.
- Wenn der Client eine Nachricht mit dem Ergebnis einer Punkteauswertung erhält, soll die Punktzahl auf die bisherigen Punkte aufsummiert werden. Dann soll geprüft werden, ob der Spieler bereits das Ziel von 40 Punkten erreicht hat. Ist dies der Fall, soll die Methode **showWinningMessage** auf der Instanzvariablen **gui** mit einer passenden Meldung aufgerufen werden. Wenn das Ziel nicht erreicht wurde, sollen

[4] siehe Hinweis am Ende dieses Dokuments

die Punkte in der GUI mit Hilfe der Methode **setPoints** neu gesetzt werden. Außerdem soll die Liste der Würfel mit Hilfe der Methode `setDicesList` neu gesetzt und das aktualisierte Spielbrett dann mit Hilfe der Methode `createGameGUI` angezeigt werden.

Zählen Sie die Anzahl der Versuche mit, die der Spieler benötigt, um die 40 Punkte zu erreichen. Geben Sie diese in der Meldung mit aus, die angezeigt wird, wenn der Spieler die 40 Punkte erreicht hat.

- Wenn der Client eine Nachricht vom Typ **DISCONNECT** vom Server erhält, muss das Spiel durch Aufruf der Methode **resetUI** auf der Instanzvariablen **gui** beendet werden.
- Alle anderen Nachrichten sollen ignoriert werden.

Kommt es zu einem Kommunikationsfehler (beispielsweise dadurch, dass der Server ein Objekt geschickt hat, welches nicht der Klasse **Message** angehört), soll der Fehler in der Konsole ausgegeben und das Protokoll kontrolliert beendet werden.

Hinweis

*Die **reset()**-Methode bei **ObjectOutputStream** wird verwendet, um die interne Objekt-Tabelle zu leeren, sodass nachfolgende Objekte als neue Instanzen behandelt und vollständig serialisiert werden. Dies verhindert, dass der Stream dieselben Objektreferenzen sendet, was besonders wichtig ist, wenn wiederholt identische Objekte übertragen werden sollen.*[5]

[5] Quelle: ChatGPT