Angabe für das Abschlussprojekt in WS2014 / MIT / SAD / Netzwerke

1 Einführung

Das vorliegende Dokument beschreibt die Aufgabenstellung[1] für das Abschlussprojekt. Es handelt sich um ein Programm, dass ein Spiel basierend auf mathematischen Kenntnissen repräsentieren soll. Nachfolgend wird hierfür der Name *MathGame* verwendet.

2 Anforderungen

Nachfolgend werden die **Mindestanforderungen** beschrieben. Die Nichteinhaltung dieser Mindestanforderungen resultiert automatisch in einer negativen Note.
Das Programm **muss** aus Enums, Klassen, Interfaces, Delegates und Events bestehen. Die Klassen und deren Methoden müssen sinngemäß gekapselt werden, so dürfen in Methoden die z.B.: der Berechnung von Werten dienen keine Eingaben von bzw. Ausgaben am Bildschirm vorgenommen werden. Entwickeln Sie das Programm in Hinblick auf Erweiterbarkeit.

2.1 Coding Guidelines

Um die Lesbarkeit und Wartbarkeit des Codes zu gewährleisten sind ALLE StyleCop-Regeln einzuhalten. Automatisch generierte Dateien, wie z.B.: Assembly.cs, sind von dieser Regel nicht betroffen, Änderungen an diesen Dateien müssen also nicht vorgenommen werden.

2.2 Bedienbarkeit

Das Programm muss für den Benutzer intuitiv bedienbar sein, Fehleingaben müssen so weit wie möglich toleriert werden. Abstürze sämtlicher Art sind zu vermieden. Das bedeutet, dass sämtliche Laufzeitfehler abgefangen, entsprechend behandelt und ggf. dem Benutzer in sinnvoller Art und Weise zum Quittieren übergeben werden müssen.

[1] Allgemeiner Hinweis zur Abgabe: Es wird eine Abgabe inklusive Abgabe-Prüfungsgespräch geführt in dem nachgewiesen werden muss, dass sämtliche Inhalte selbstständig erarbeitet und verstanden wurden. Dies wird unter Umständen auch durch implementieren von Funktionalität während des Prüfungsgesprächs überprüft.

3 Systemarchitektur

Es existieren Game-Server, Game-Clients und Clients, die in der Lage sind den aktuellen Zustand des Game-Servers zu ermitteln. In der nachfolgenden Abbildung wird die Architektur des Systems schematisch dargestellt:

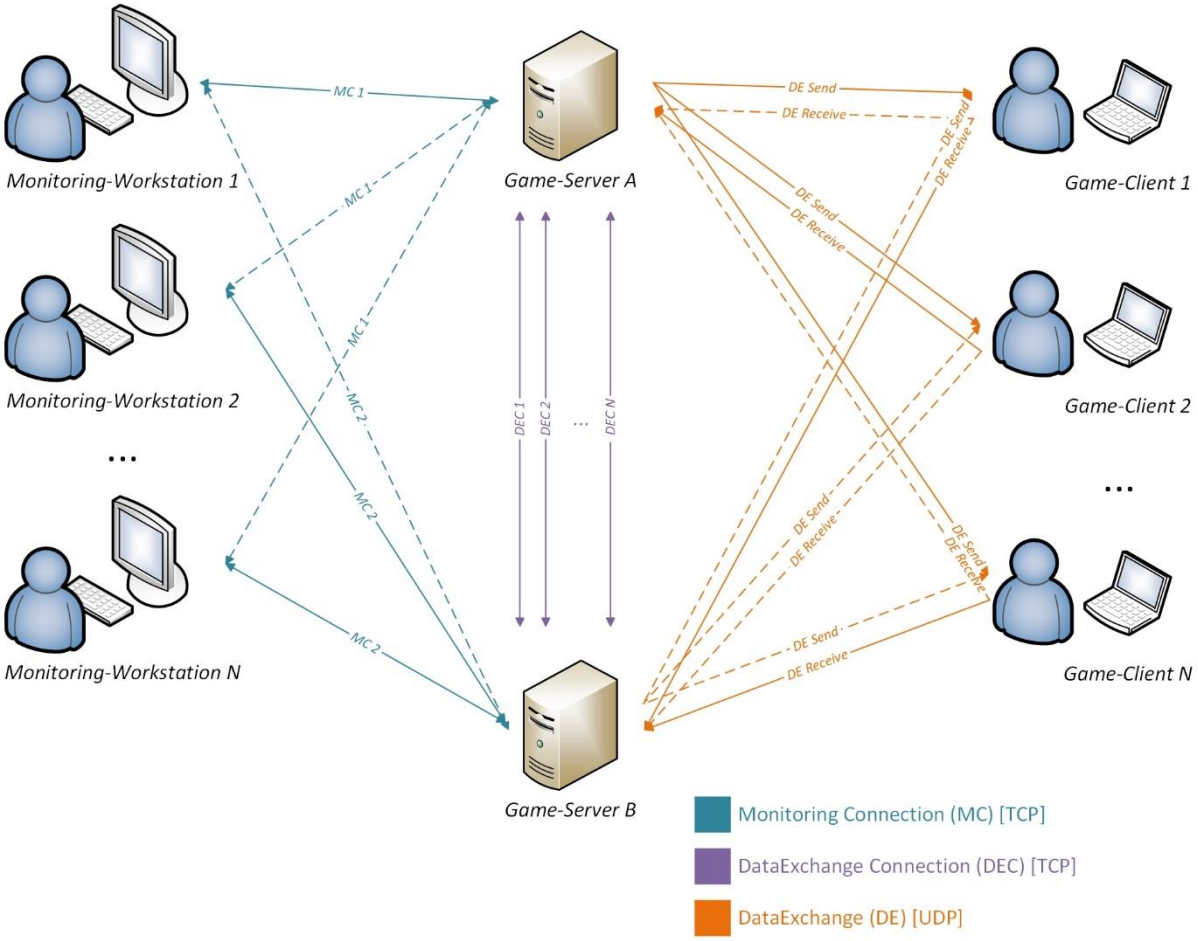

Abbildung 1 – Schematische Darstellung der Systemarchitektur

Wie in *Abbildung 1 – Schematische Darstellung der* Systemarchitektur ersichtlich, existiert ein redundantes Game-Server-Paar (*Game-Server A* und *Game-Server B*). Einer der beiden *Game-Server* ist der aktive *Game-Server* (kann Daten von *Game-Clients* entgegen nehmen und Daten an *Game-Clients* schicken), der andere der passive *Game-Server* (kann Daten von *Game-Clients* entgegen nehmen, darf aber keine Daten an *Game-Clients* schicken). Der Status dieser *Game-Server* kann mittels einer *Monitoring Applikation*, die auf einer *Monitoring-Workstation* läuft, überwacht werden. Hierbei wird von der *Monitoring-Applikation* eine TCP-Verbindung zu einem der beiden *Game-Server* aufgebaut, über die entsprechende Monitoring-Daten zur Verfügung gestellt werden.
Andererseits können sich *Game-Clients* über UDP bei einem der *Game-Server* melden, der jeweils aktive *Game-Server* antwortet ebenfalls immer mittels UDP.

4 Mindestanforderungen pro Notengrad und Überprüfung

Nachfolgend werden die jeweiligen Mindestanforderungen pro Notengrad beschrieben. Für eine positive Note müssen mindestens die Anforderungen der Note „Genügend" vollständig und fehlerfrei umzusetzen. In Abhängigkeit der Anzahl an zusätzlich umgesetzten Funktionalitäten (Erweiterungen A – C) ergeben sich die Notengrade „Befriedigend", „Gut" und „Sehr Gut". Für eine zusätzlich umgesetzte Funktionalität kann bestenfalls die Note „Befriedigend", für zwei zusätzlich umgesetzte Funktionalitäten die Note „Gut" und für drei zusätzlich umgesetzte Funktionalitäten die Note „Sehr Gut" erreicht werden. Die Reihenfolge der zusätzlich umgesetzten Funktionalitäten **ist hierbei von Relevanz, nach der Erfüllung der Anforderungen für den Notengrad Genügend folgt Erweiterung A, danach Erweiterung B und abschließend Erweiterung C**.

4.1 Genügend

Implementieren Sie den *Game-Server* sowie die *Monitoring-Applikation* als Consolen Applikation und mindestens einen *Game-Client* nach Wahl (Consolen Applikation, WPF-Applikation, Web-Applikation, XNA-Applikation, Mobile-Applikation, …) und erfüllen Sie alle Mindest- und in diesem Kapitel beschriebenen Anforderungen.

Abbildung 2 – Schematische Darstellung der Systemarchitektur für Notengrad Genügend stellt die geforderte Systemarchitektur für den *Notengrad Genügend* schematisch dar. Hierbei ist zu erkennen, dass der *Game-Server* in einer „*standalone*" Variante ausgeführt ist und lediglich eine *Monitoring-Applikation* sowie einen *Game-Client* gleichzeitig bedienen kann.

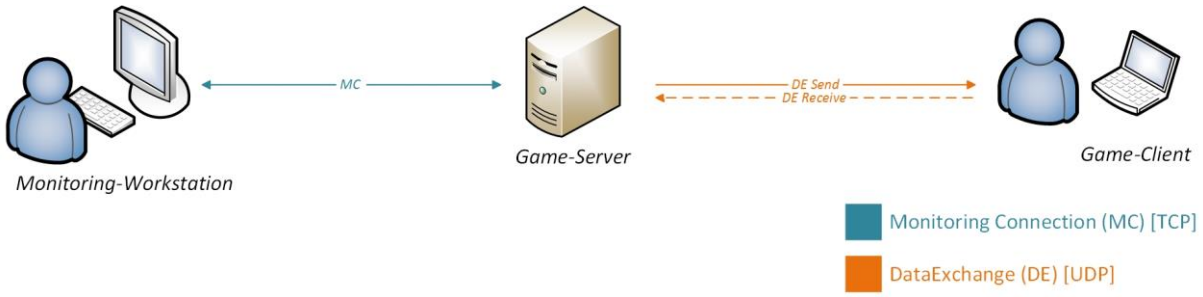

Abbildung 2 – Schematische Darstellung der Systemarchitektur für Notengrad Genügend

Datenaustauschprotokoll:

- Überlegen Sie sich entsprechende Datenaustauschprotokolle um Daten zwischen den *Game-Servern* (muss TCP sein), zwischen *Game-Server* und *Monitoring-Applikation* (muss TCP sein) und zwischen *Game-Server* und *Game-Client* (muss UDP sein) austauschen zu können.
  (**Hinweis:** Es ist nicht relevant, das alle Verbindungen das gleiche Datenaustauschprotokoll unterstützen, ein eigenes Datenaustauschprotokoll pro Verbindung ist erlaubt – also z.B.: proprietär, binär zwischen *Game-Server* und *Game-Client*, XML-basiert zwischen *Game-Server* und *Monitoring-Applikation.*)
- Erstellen Sie eine Protokollspezifikation für jedes vorhandene Datenaustausch-protokoll.
  (**Hinweis:** Beschreiben Sie nicht nur, wie die Daten transportiert und interpretiert werden sondern auch den Ablauf des Datenaustausches (z.B.: Wie wird der *Game-Client* beim *Game-Server* registriert, …).)

Beispiel für ein binäres Datenaustauschprotokoll:
Das nachfolgende Protokoll dient als Beispiel für ein binäres Datenaustauschprotokoll. Hierbei wird zuerst ein Basisprotokoll definiert, das in allen Verbindungen (MC, DE und DEC) gleich aufgebaut ist. In Abhängigkeit der Verbindung und der zu übertragenden Daten ändert sich dann der Payload (ähnlich wie aus dem ISO/OSI-Modell bekannt, werden hier also Pakete in Pakete gepackt). In *Tabelle 1 - Aufbau des Beispiel-Datenaustauschprotokolls* wird der Aufbau des Beispiel-Datenaustauschprotokolls dargestellt. *Tabelle 2 - Felddefinition des Beispiel-Datenaustauschprotokolls* stellt die Position und Länge sowie die Bedeutungen bzw. gültige Werte der einzelnen Felder dar:

Protocol Identification Header	Sub-Protocol class	Version	Payload length	Payload data	Checksum
5 Bytes	1 Byte	1 Byte	2 Bytes	N Bytes	2 Bytes

Tabelle 1 - Aufbau des Beispiel-Datenaustauschprotokolls

Offset	Länge	Bedeutung
0	5	Protocol Identification Header
		Always 0x4D, 0x47, 0x44, 0x50, 0x49
5	1	Sub-Protocol class
		0x01 for MC
		0x02 for DE
		0x03 for DEC
6	1	Version
		Currently always 1
7	2	Payload length (PL)
9	PL	Payload data
PL	2	Checksum

Tabelle 2 - Felddefinition des Beispiel-Datenaustauschprotokolls

Server:

- Der *Game-Server* ist **nicht** redundant ausgeführt.
- Der Game-Server unterstützt einen *Game-Client*.
- Der Game-Server unterstützt eine *Monitoring-Applikation*.
- Die Default-Konfiguration des Servers kann über eine Konfigurationsdatei, welche dem Server beim Start angegeben wird, modifiziert werden. Sollte keine Konfigurationsdatei angegeben werden so verwendet der Server entsprechend sinnvolle Default-Werte.
- Sobald ein *Game-Client* verbunden ist startet der *Game-Server* das Spiel (der detaillierte Spielablauf wird nachfolgend in *Spielablauf* beschrieben).
- Der Server loggt alle möglichen Zustandsänderungen (Neuer *Game-Client* hat sich gemeldet, Spiel beendet, etc.).

Client:

- Der *Game-Client* kann sich beim *Game-Server* registrieren.
- Der *Game-Client* stellt dem Benutzer ein entsprechendes Interface zur Verfügung um alle Anforderungen, die in *Spielablauf* beschrieben sind, zu erfüllen.

Monitoring Applikation:

- Die *Monitoring-Applikation* kann sich zu einem *Game-Server* verbinden.
- Nach erfolgreicher Verbindung stellt die *Monitoring-Applikation* sämtliche, vom *Game-Server* geloggten Daten optisch ansprechend aufbereitet zur Verfügung.

Spielablauf:

- Der *Game-Client* registriert sich beim *Game-Server*, hierbei wird der Spielername gemeldet.
- Nach erfolgreicher Registrierung startet der *Game-Server* nach eigenem Ermessen das Spiel.
- Der *Game-Server* schickt an den *Game-Client* eine mathematische Fragestellung, wie z.B.: 1 + 1 = ?
- Der *Game-Client* muss innerhalb einer am *Game-Server* einstellbaren Zeitspanne (Timeout) auf die gestellte Fragestellung antworten.
- Erhält der *Game-Server* innerhalb der *Game-Server* vor Ablauf des Timeouts eine Antwort so wird diese auf ihre Korrektheit ausgewertet.
- Ist die erhaltene Antwort korrekt, so erhält der *Game-Client* einen Punkt.
- Ist die erhaltene Antwort inkorrekt oder hat der *Game-Client* keine Antwort übermittelt, so erhält der *Game-Client* einen Minuspunkt.
- Sobald die Punkteanzahl des *Game-Clients* eine einstellbare Obergrenze überschreitet oder einstellbare Untergrenze unterschreitet endet das Spiel und der *Game-Server* übermittelt an den *Game-Client* das Resultat (Anzahl der Spiele, Punktestand).

4.2 Erweiterung A

Zur Umsetzung der *Erweiterung A* sind nachfolgende Anforderungen zu erfüllen:

- Der *Game-Server* ist redundant implementiert und unterstützt eine beliebige Anzahl an Verbindungen zu seinem redundanten Partner.
- Einer der *Game-Server* muss aktiv, der andere passiv sein. Der jeweils aktive *Game-Server* schickt alle Antworten zum *Game-Client*.
- Sowohl der aktive als auch der passive *Game-Server* können Daten vom *Game-Client* entgegen nehmen.
- Sobald die Verbindung (bei mehreren bestehenden Verbindungen alle Verbindungen) zwischen den *Game-Servern* getrennt wird, werden beide Game-Server aktiv und arbeiten für sich autonom.
- Bei Wiederherstellen der Verbindung wird jener *Game-Server* aktiv, der länger „am Leben“ ist. Hierbei werden die Teilergebnisse bestmöglich zusammengeführt. Sollten beide *Game-Server* gleich lange „am Leben“ sein, wird zufällig einer der *Game-Server* als aktiver *Game-Server* gewählt.

Abbildung 3 – Schematische Darstellung der Systemarchitektur für die Erweiterung A stellt die geforderte Systemarchitektur für die *Erweiterung A* schematisch dar. Hierbei ist zu erkennen, dass der *Game-Server* in einer redundanten Variante ausgeführt ist und lediglich eine *Monitoring-Applikation* sowie einen *Game-Client* gleichzeitig bedienen kann.

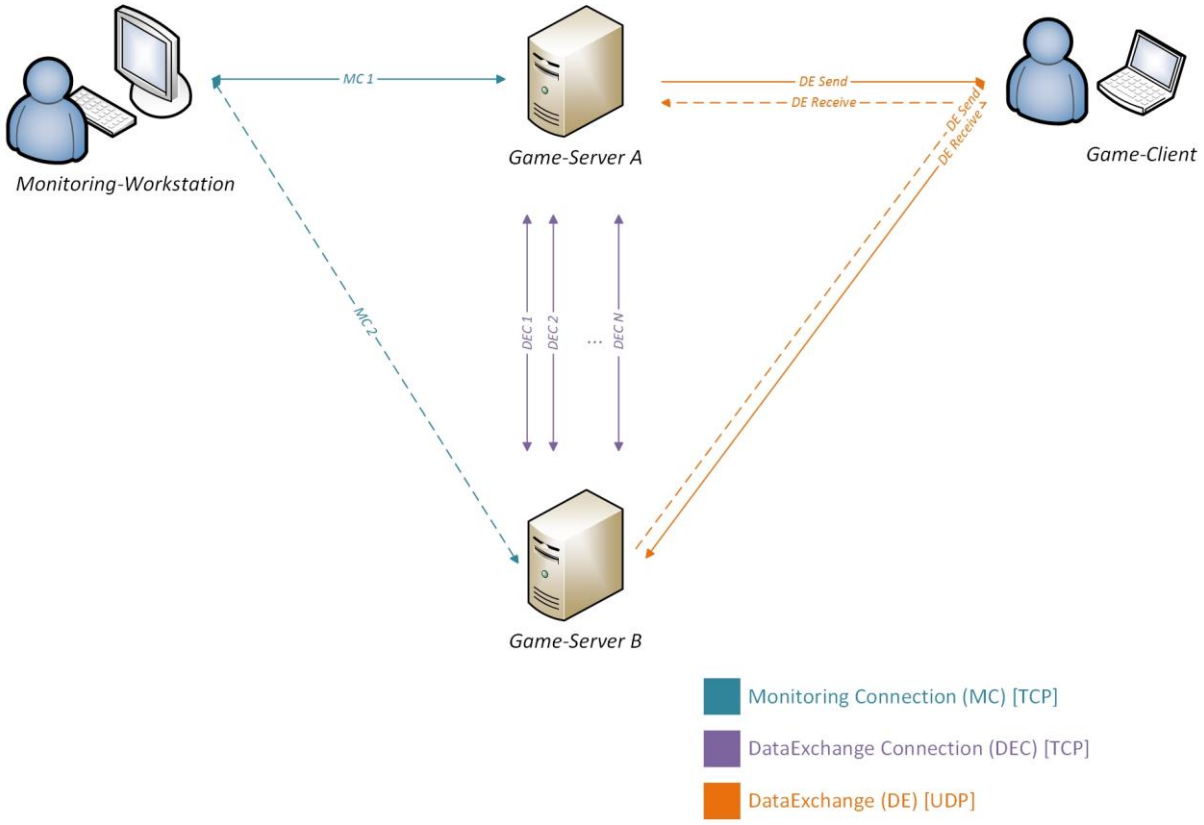

Abbildung 3 – Schematische Darstellung der Systemarchitektur für die Erweiterung A

4.3 Erweiterung B

Zur Umsetzung der *Erweiterung B* sind nachfolgende Anforderungen zu erfüllen:

- Die *Game-Server* unterstützen eine beliebige Anzahl an *Game-Clients*.
- Sobald ein *Game-Client* ein Spiel beendet hat wird das Resultat (Name des *Game-Clients* oder Spielers, Anzahl der Spiele, Punktestand) an alle sich noch im Spiel befindlichen *Game-Clients* übermittelt und entsprechend dargestellt.
- Der Game-Server führt ab dem Start eine Highscore-Liste aller teilnehmenden *Game-Clients*, diese kann jederzeit von jedem *Game-Client* abgefragt werden.
- Die *Game-Server* unterstützen eine beliebige Anzahl an *Monitoring-Applikationen*.

Abbildung 4 – Schematische Darstellung der Systemarchitektur für Erweiterung B stellt die geforderte Systemarchitektur für die *Erweiterung B* schematisch dar. Hierbei ist zu erkennen, dass der *Game-Server* in einer redundanten Variante ausgeführt ist und eine beliebige Anzahl an *Monitoring-Applikation* sowie *Game-Clients* gleichzeitig bedienen kann.

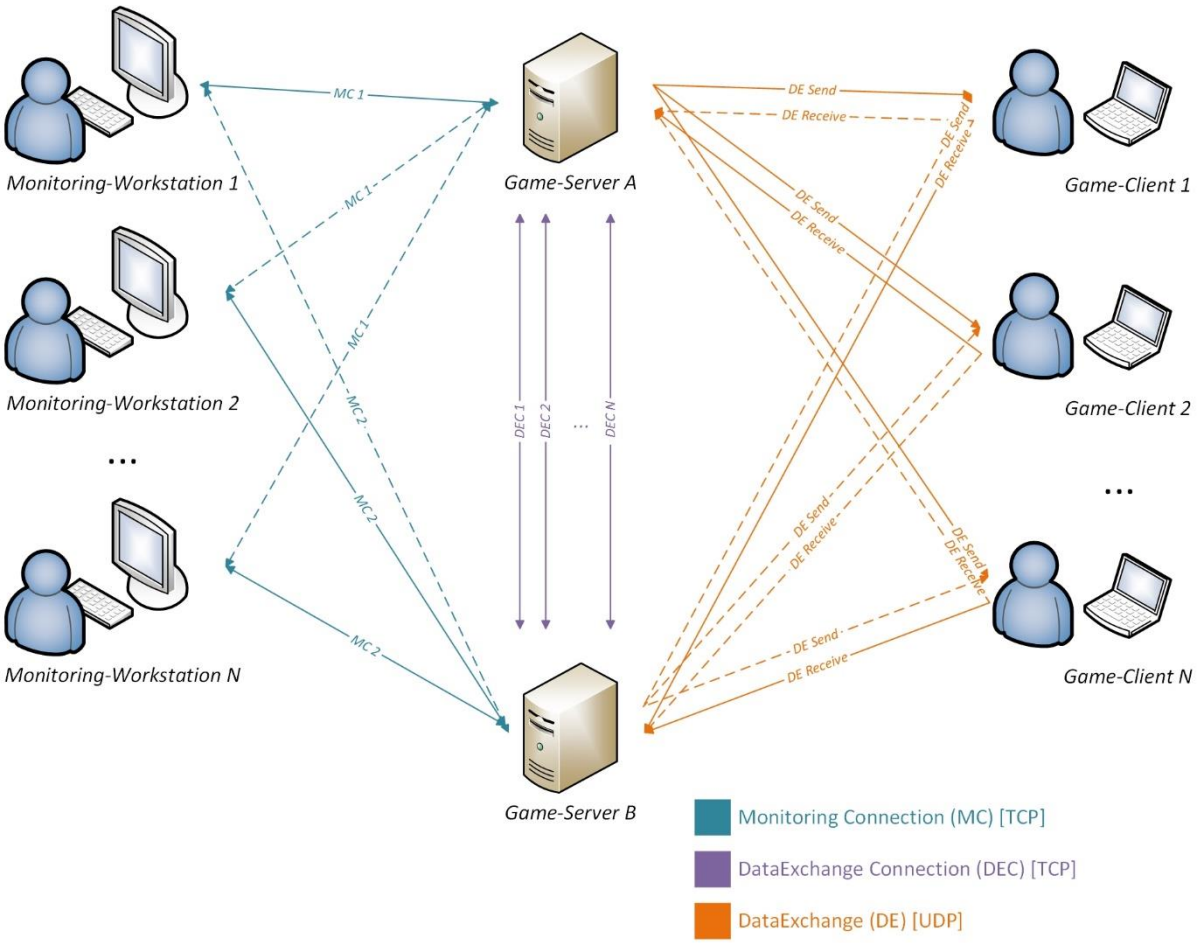

Abbildung 4 – Schematische Darstellung der Systemarchitektur für Erweiterung B

4.4 Erweiterung C

Zur Umsetzung der *Erweiterung C* sind nachfolgende Anforderungen zu erfüllen:

- Implementieren Sie die Netzwerkübertragungsschicht derart, dass durch Einhalten von Ihnen definierter Interfaces die Datenübertragung auf beliebigem Wege stattfinden kann (z.B.: über Dateien auf einem Fileserver).
- Implementieren Sie exemplarisch eine alternative Art der Datenübertragung für die Verbindung DE (zwischen *Game-Server* und *Game-Client*).

4.5 Überprüfung

Es wird eine Abgabe inklusive Abgabe-Prüfungsgespräch geführt in dem nachgewiesen werden muss, dass sämtliche Inhalte selbstständig erarbeitet und verstanden wurden. Dies wird unter Umständen auch durch implementieren von Funktionalität während des Prüfungsgesprächs überprüft.