Systementwurf

(Aufgabe 3 – Systemproggrammierung)

1. Projektteam

Teammitglieder:
Sergei Neigum (23758)
Patrick Fetscher (23950) [Teamleiter]
Filipe Soares Felisberto (24629)

Aufgabenaufteilung:
Client

RFC:	Sergei Neigum
Login-Phase:	
main-Thread:	Sergei Neigum
Spielvorbereitung:	
main-Thread:	Sergei Neigum
GUI-Thread:	Sergei Neigum
Listener-Thread:	Sergei Neigum
Spielphase:	
main-Thread:	Sergei Neigum
GUI-Thread:	Sergei Neigum
Listener-Thread:	Sergei Neigum
Fragewechsel-Thread:	Sergei Neigum
Spielende:	
main-Thread:	Sergei Neigum
GUI-Thread:	Sergei Neigum
Listener-Thread:	Sergei Neigum

Server

RFC:	Filipe Soares Felisberto, Patrick F.
Login-Phase:	
main-Thread:	Filipe Soares Felisberto, Patrick F.
Login-Thread:	Filipe Soares Felisberto, Patrick F.
Spielvorbereitung:	
main-Thread:	Filipe Soares Felisberto, Patrick F.
Client-Thread:	Filipe Soares Felisberto, Patrick F.
Score-Thread:	Filipe Soares Felisberto, Patrick F.
Spielphase:	
main-Thread:	Filipe Soares Felisberto, Patrick F.

Client-Thread: Filipe Soares Felisberto, Patrick F.
Score-Thread: Filipe Soares Felisberto, Patrick F.

Spielende:

main-Thread: Filipe Soares Felisberto, Patrick F.
Client-Thread: Filipe Soares Felisberto, Patrick F.
Score-Thread: Filipe Soares Felisberto, Patrick F.

2. Aufbau des Systems

2.1. Welche Prozesse und Threads existieren?

Prozesse:	Client (max. 4 zusammen aufgrund der Spieler)	Server	Loader
Threads:	GUI-Thread Listener-Thread Fragewechsel-Thread	Login-Thread Client-Thread Score-Thread	Loader-Thread

2.2. Welche Daten existieren zwischen den Prozessen / Threads?

Kommunikation zwischen Client und Server (Client ←→ Server)
Die Kommunikation zwischen Client und Server soll über Sockets erfolgen, d.h. über die Sockets können Daten sowohl empfangen als auch gesendet werden.

Kommunikation zwischen Server und Loader (Server ←→ Loader)
Der Datenaustausch zwischen Server und Loader findet über das Shared-Memory statt. Die Synchronisation zwischen den Prozessen, die Zugriff auf das Shared-Memory haben, soll über binäre Semaphore realisiert werden.
Ein Zugriff auf den Loader wird von einem Client-Thread über Pipes aktiviert/getätigt.
Der Client-Thread schickt eine Anfrage an den Loader. Daraufhin antwortet der Loader entweder mit der Katalogliste oder schreibt Daten ins Shared-Memory. Das versendete Ergebnis wird mit einer „Success-Message“ gesendet.

Login-Thread (Server) ←→ Client
Der Login-Thread überprüft, ob die Anmeldung erfolgreich war. Daraufhin werden die Anmeldedaten auf dem Server gespeichert und die Auswertung wird an den Client übermittelt.

GUI-Thread (Client) ←→ Client-Thread (Server)
Der GUI-Thread empfängt Eingaben des Spielers und sendet diese Daten über die Sockets direkt an den Client-Thread des Servers weiter.

Listener-Thread(Client) ←→ GUI-Thread (Client)
Der Listener-Thread stellt die Fragen aus dem Fragekatalog auf der GUI dar und ist auch zuständig für die Aktualisierung der Fragekataloge. Er benachrichtigt den GUI-Thread sobald eine neue Spielphase anfängt.

Listener-Thread (Client) ←→ Fragewechsel-Thread (Client)
Der Fragewechsel-Thread wartet auf eine Anweisung vom Listener-Thread, damit er die eingegangene Nachricht weiterleiten kann.

Fragewechsel-Thread (Client) ←→ Client-Thread (Server)
Der Fragewechsel-Thread ist dafür zuständig eine neue Frage zu generieren und kommuniziert mit dem Client-Thread des Servers sobald eine neue Frage aufgerufen werden soll.

Listener-Thread (Client) ← Login-Thread (Server)
Der Login-Thread sendet dem Listener-Thread Nachrichten, falls neue Spieler das Spiel gestartet oder die Spieler das Spiel verlassen haben.

Listener-Thread (Client) ←→ Client-Thread (Server)
Die Daten, welche vom Loader kommen, werden von dem Server über das Netzwerk an den Client weitergesendet. Für jeden einzelnen Spieler gibt es dafür einen Client-Thread. Sämtliche Daten, die vom Server kommen werden im Listener-Thread aufgefangen.

Login-Thread (Server) → Client-Thread (Server)
Der Login-Thread sendet eine Nachricht an den Client-Thread, mit dem Inhalt: “Anmeldung OK“.

Login-Thread (Server) → Score-Thread (Server)
Der Score-Thread greift auf die Liste der Spieler zu, damit dieser den einzelnen Spielern die jeweiligen Punkte während der Spielphase vergeben kann.

Client-Thread (Server) → Score-Thread (Server)
Der Score-Thread empfängt von dem Client-Thread die Änderung des Punktestands und aktualisiert anschließend den Punktestand der Spieler.

Client-Thread (Server) ←→ Loader
Der Client-Thread tauscht Daten zwischen dem Loader über das Shared-Memory aus.

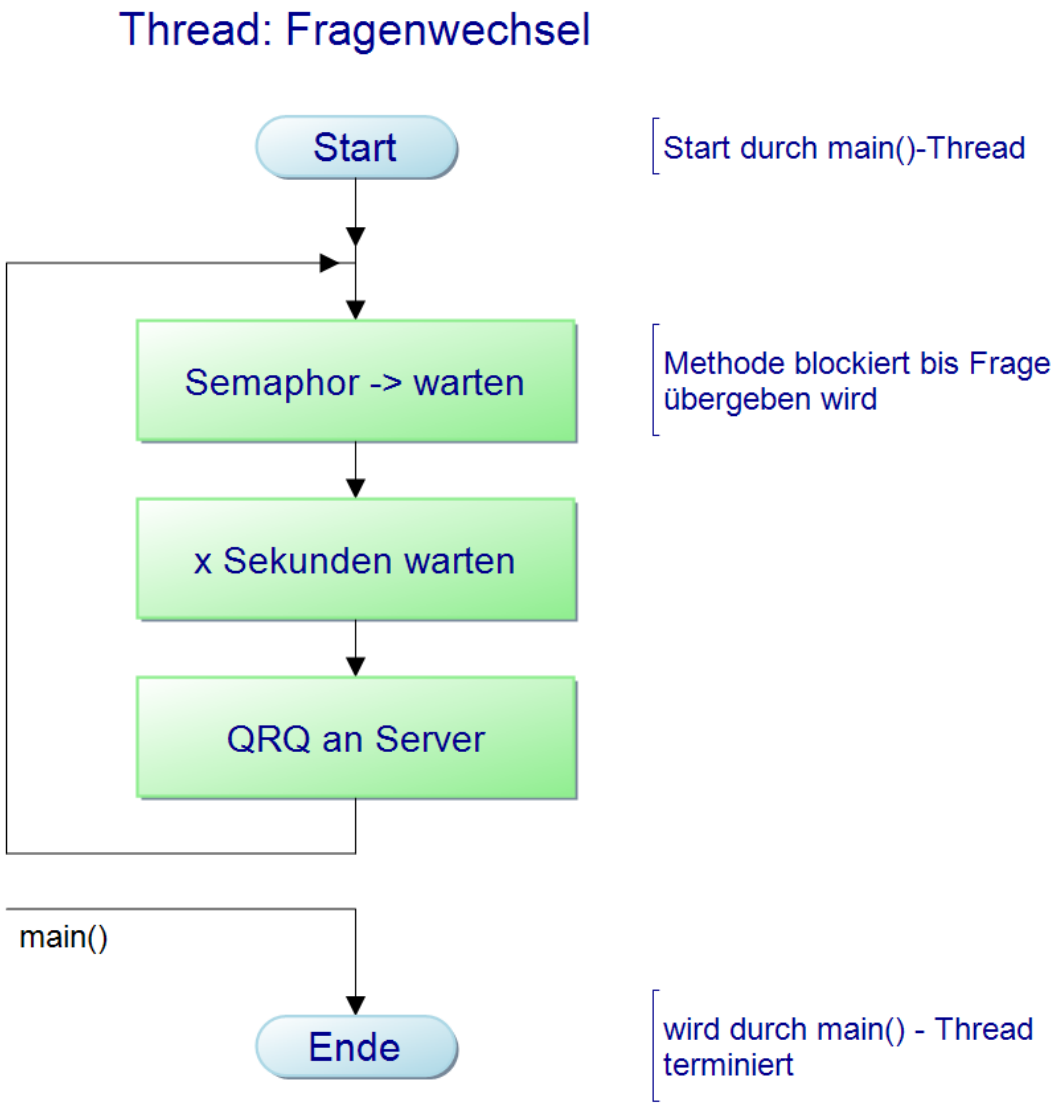
- Die Kommunikation zwischen den einzelnen Threads funktioniert über Mutexe.
- Die Daten, die zwischen dem Server und dem Client ausgetauscht werden, werden mit dem Protokoll TCP übermittelt.
- Die Kommunikation zwischen Client und Server erfolgt über Sockets. Die

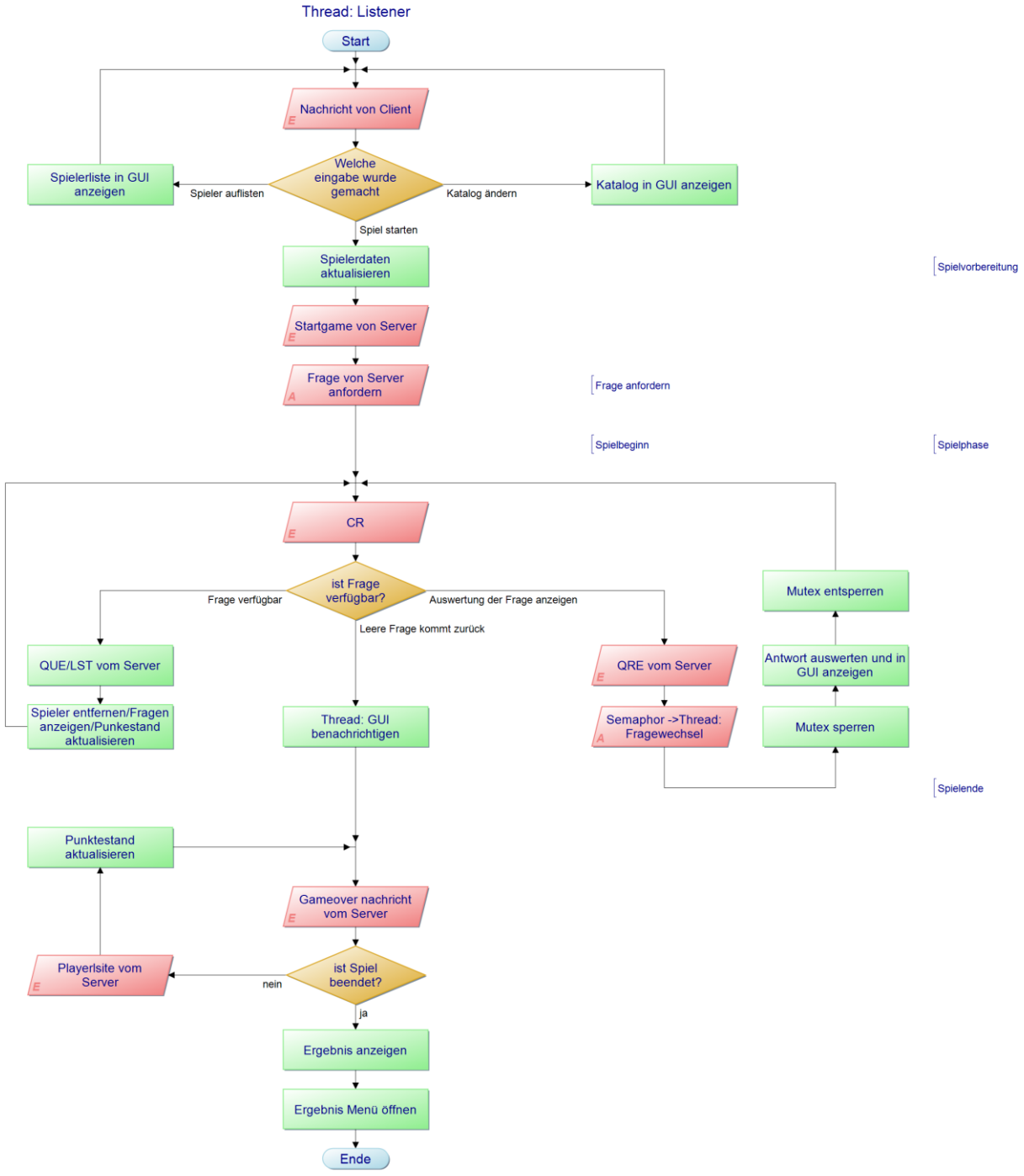
Kommunikation zwischen Server und Loader erfolgt über Pipes.

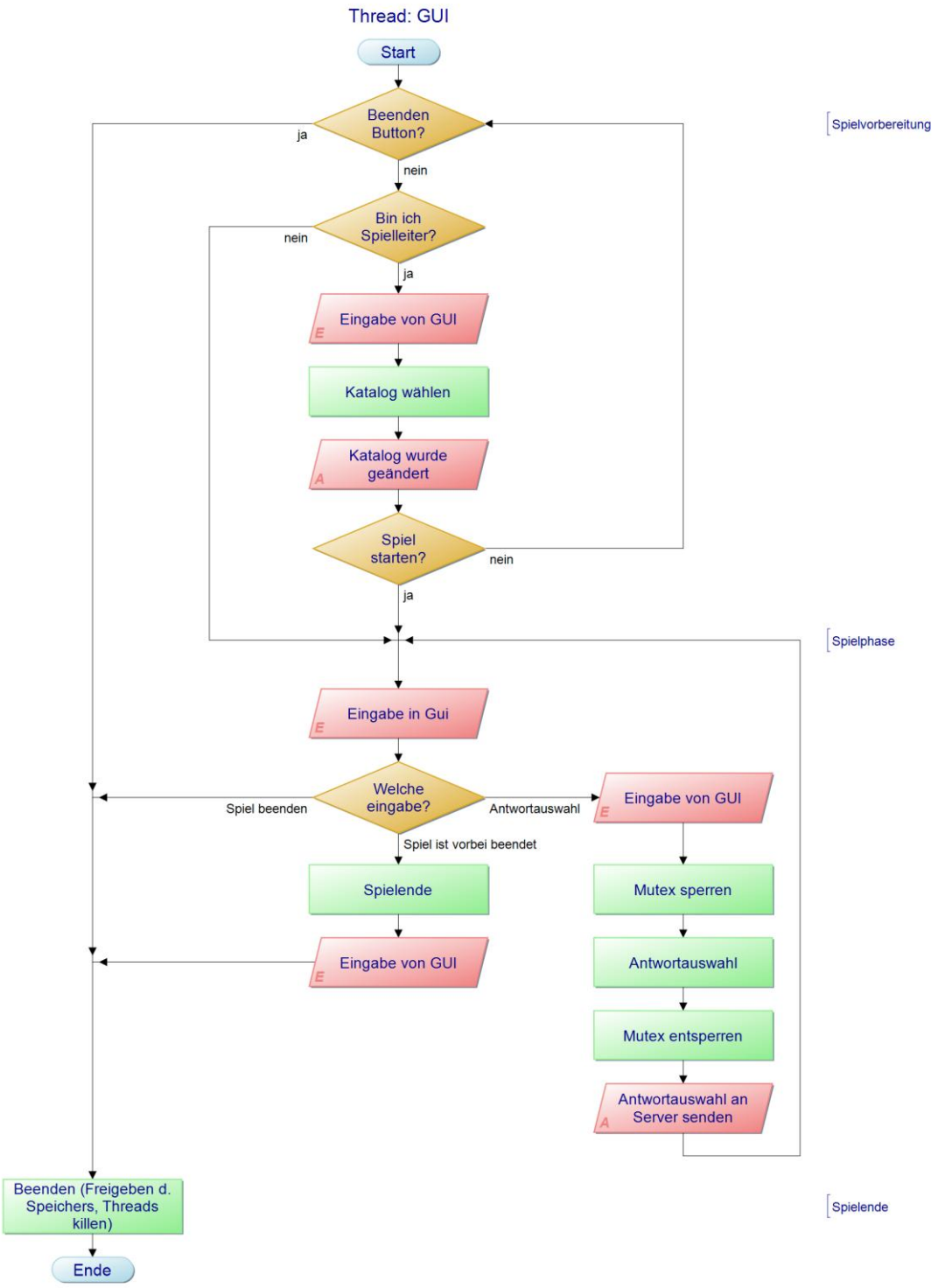
3. Abläufe in den Prozessen und Threads

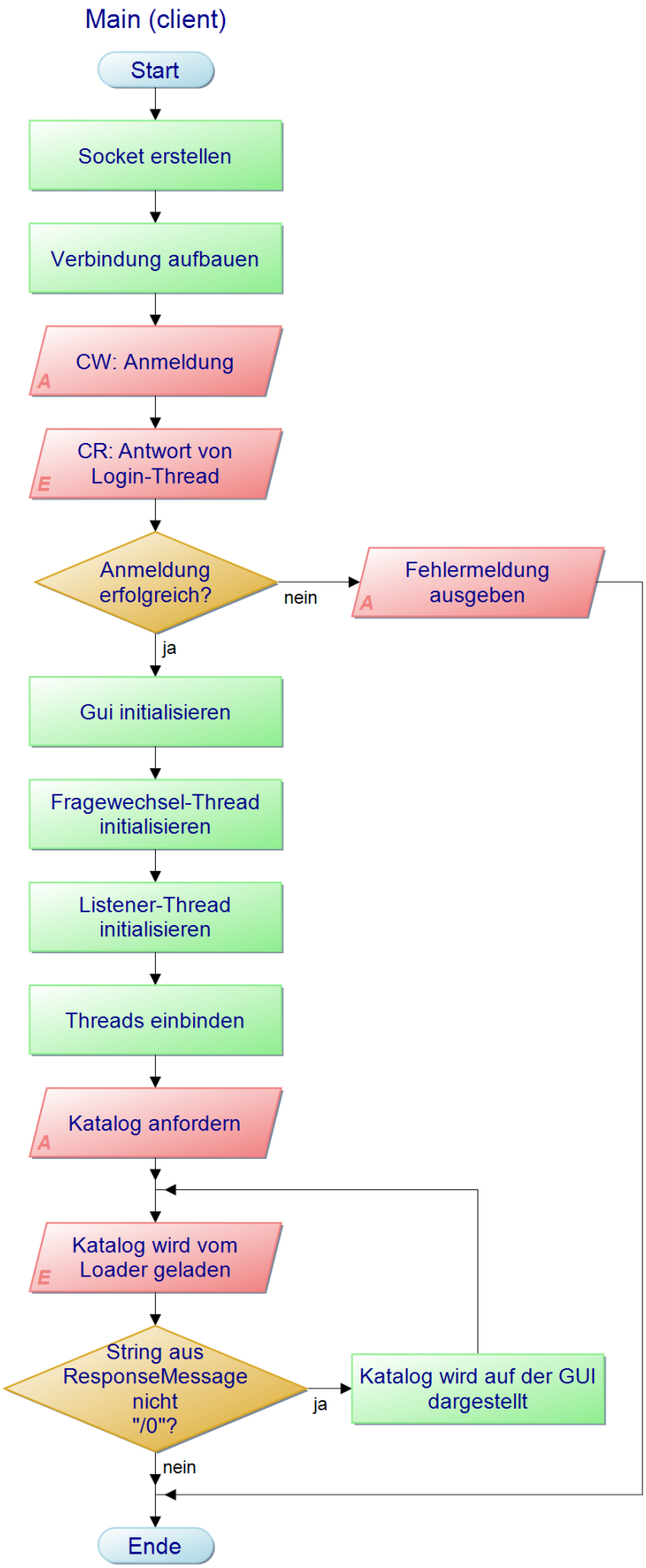
3.1. Ablaufpläne zu den Threads / Prozessen?

3.1.1. Client

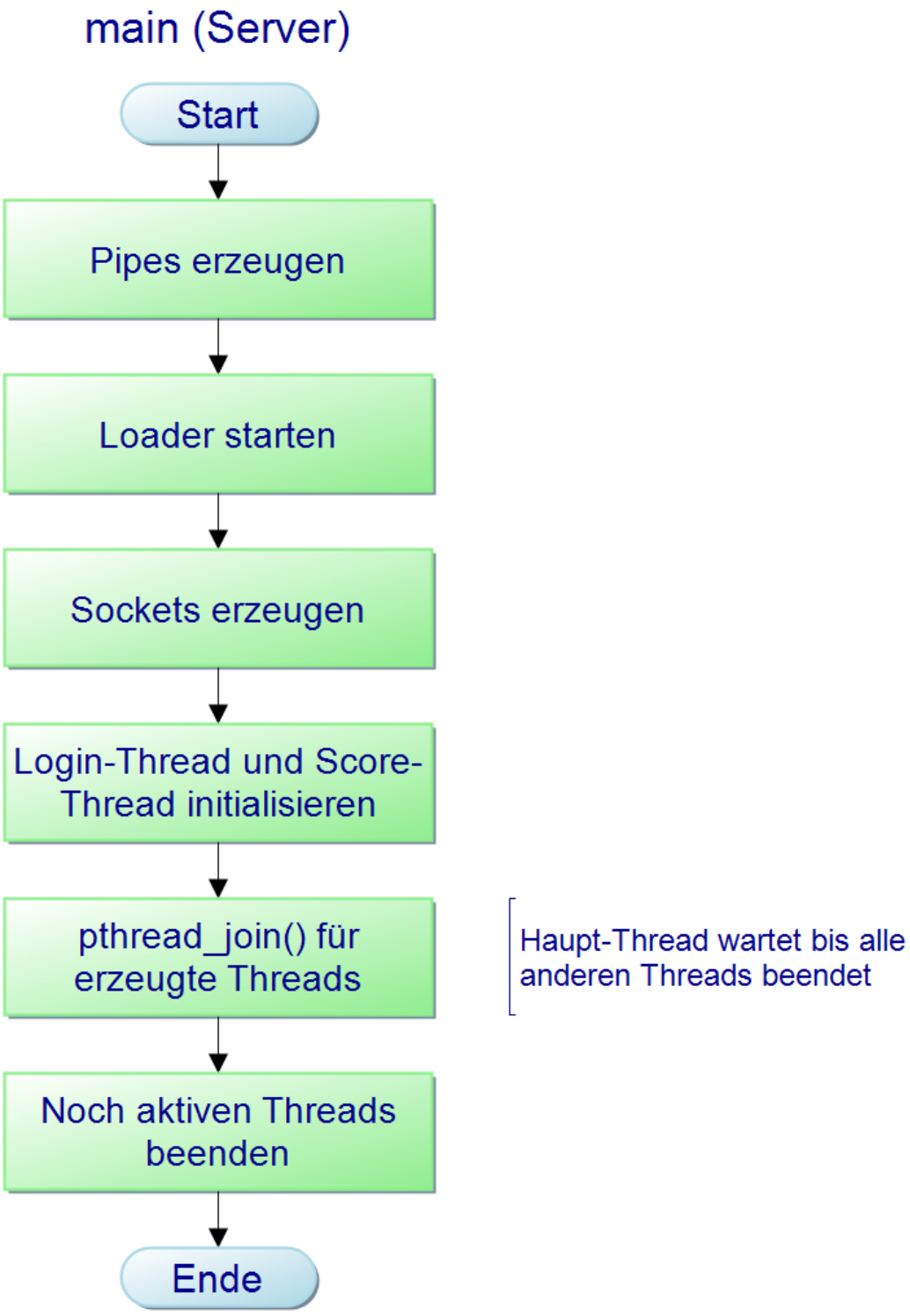


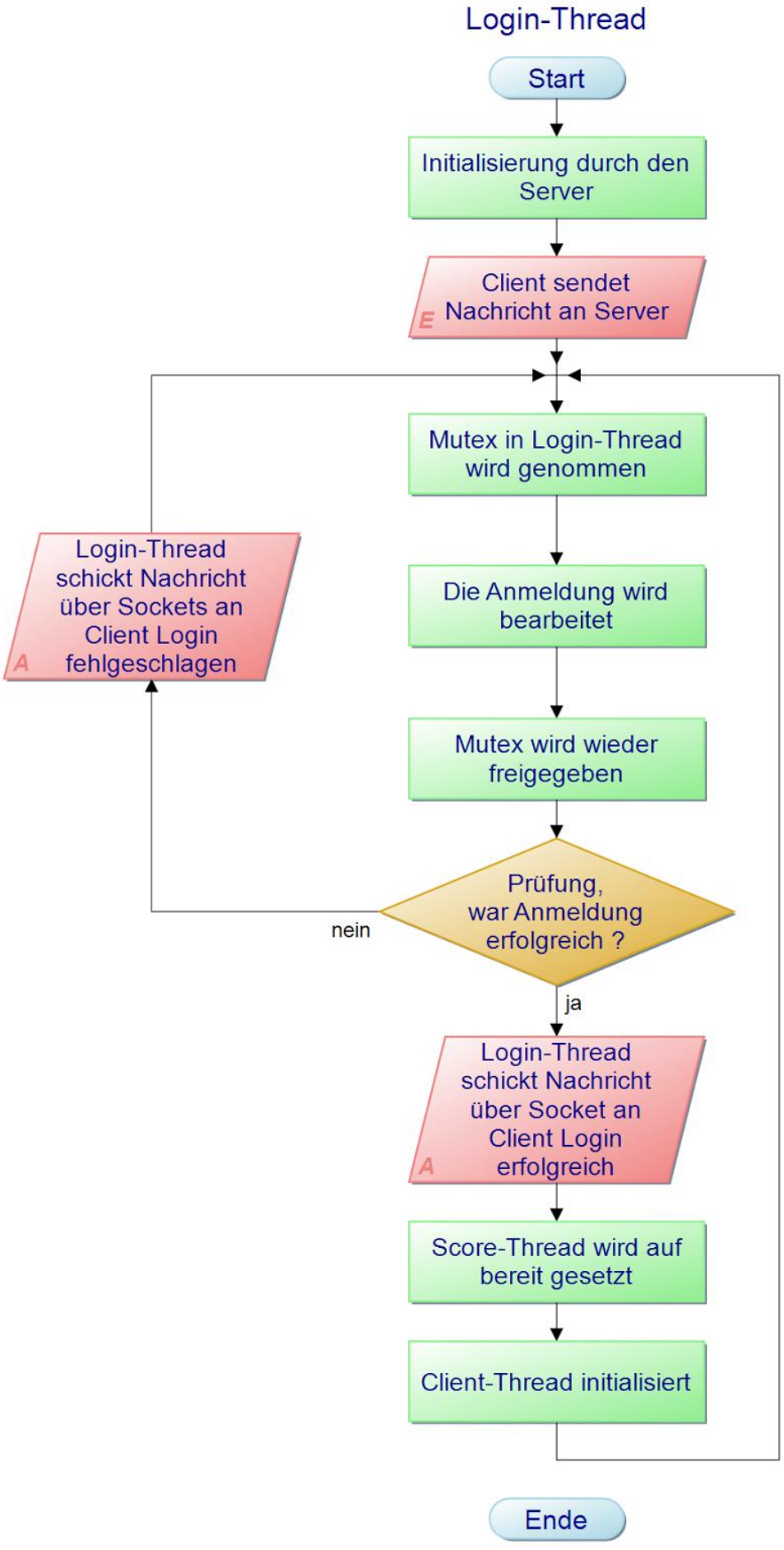

Thread: Listener
Start
Nachricht von Client
E
Welche eingabe wurde gemacht
Spieler auflisten
Spielerliste in GUI anzeigen
Katalog ändern
Katalog in GUI anzeigen
Spiel starten
Spielerdaten aktualisieren
Spielvorbereitung
Startgame von Server
E
Frage von Server anfordern
A
Frage anfordern
Spielbeginn
Spielphase
CR
E
ist Frage verfügbar?
Frage verfügbar
Auswertung der Frage anzeigen
Leere Frage kommt zurück
QUE/LST vom Server
Spieler entfernen/Fragen anzeigen/Punkestand aktualisieren
Thread: GUI benachrichtigen
QRE vom Server
E
Semaphor ->Thread: Fragewechsel
A
Mutex entsperren
Antwort auswerten und in GUI anzeigen
Mutex sperren
Spielende
Punktestand aktualisieren
Gameover nachricht vom Server
E
Playerlsite vom Server
E
ist Spiel beendet?
nein
ja
Ergebnis anzeigen
Ergebnis Menü öffnen
Ende

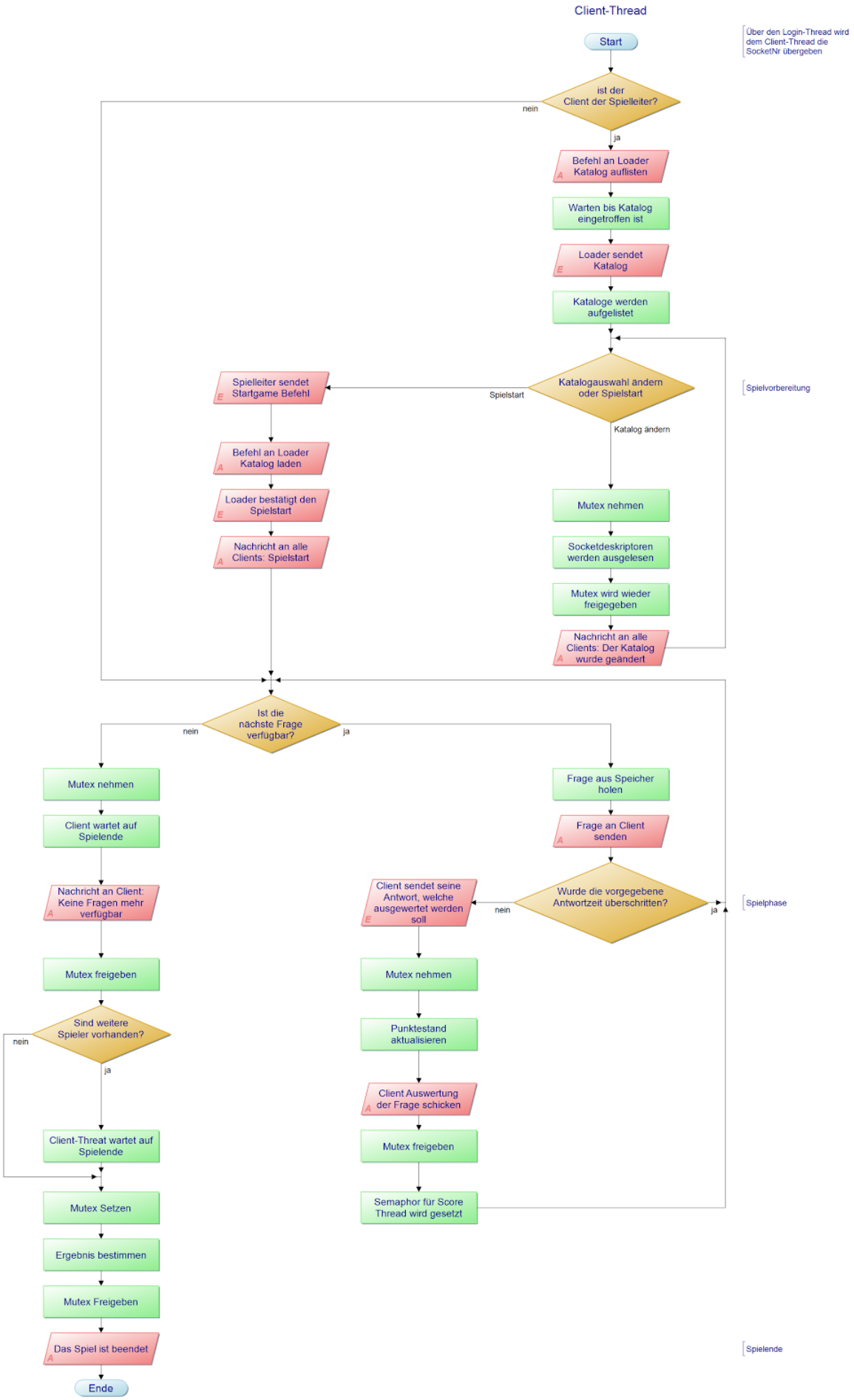

Thread: GUI
Start
Beenden Button?
ja
nein
Bin ich Spielleiter?
nein
ja
Eingabe von GUI
E
Katalog wählen
Katalog wurde geändert
A
Spiel starten?
nein
ja
Spielvorbereitung
Spielphase
Eingabe in Gui
E
Welche eingabe?
Spiel beenden
Antwortauswahl
Eingabe von GUI
E
Spiel ist vorbei beendet
Spielende
Mutex sperren
Eingabe von GUI
E
Antwortauswahl
Mutex entsperren
Antwortauswahl an Server senden
A
Beenden (Freigeben d. Speichers, Threads killen)
Ende
Spielende


Main (client)
Start
Socket erstellen
Verbindung aufbauen
CW: Anmeldung
A
CR: Antwort von Login-Thread
E
Anmeldung erfolgreich?
nein
Fehlermeldung ausgeben
A
ja
Gui initialisieren
Fragewechsel-Thread initialisieren
Listener-Thread initialisieren
Threads einbinden
Katalog anfordern
A
Katalog wird vom Loader geladen
E
String aus ResponseMessage nicht "/0"?
ja
Katalog wird auf der GUI dargestellt
nein
Ende

3.1.2. Server


main (Server)
Start
Pipes erzeugen
Loader starten
Sockets erzeugen
Login-Thread und Score-Thread initialisieren
pthread_join() für erzeugte Threads
Haupt-Thread wartet bis alle anderen Threads beendet
Noch aktiven Threads beenden
Ende


Login-Thread
Start
Initialisierung durch den Server
E
Client sendet Nachricht an Server
Mutex in Login-Thread wird genommen
Die Anmeldung wird bearbeitet
Mutex wird wieder freigegeben
Prüfung, war Anmeldung erfolgreich ?
nein
A
Login-Thread schickt Nachricht über Sockets an Client Login fehlgeschlagen
ja
A
Login-Thread schickt Nachricht über Socket an Client Login erfolgreich
Score-Thread wird auf bereit gesetzt
Client-Thread initialisiert
Ende


Client-Thread
Start
Über den Login-Thread wird dem Client-Thread die SocketNr übergeben
ist der Client der Spielleiter?
nein
ja
Befehl an Loader Katalog auflisten
Warten bis Katalog eingetroffen ist
Loader sendet Katalog
Kataloge werden aufgelistet
Katalogauswahl ändern oder Spielstart
Spielstart
Katalog ändern
Spielvorbereitung
Spielleiter sendet Startgame Befehl
Befehl an Loader Katalog laden
Loader bestätigt den Spielstart
Nachricht an alle Clients: Spielstart
Mutex nehmen
Socketdeskriptoren werden ausgelesen
Mutex wird wieder freigegeben
Nachricht an alle Clients: Der Katalog wurde geändert
Ist die nächste Frage verfügbar?
nein
ja
Mutex nehmen
Client wartet auf Spielende
Nachricht an Client: Keine Fragen mehr verfügbar
Mutex freigeben
Sind weitere Spieler vorhanden?
nein
ja
Client-Threat wartet auf Spielende
Mutex Setzen
Ergebnis bestimmen
Mutex Freigeben
Das Spiel ist beendet
Ende
Frage aus Speicher holen
Frage an Client senden
Wurde die vorgegebene Antwortzeit überschritten?
nein
ja
Client sendet seine Antwort, welche ausgewertet werden soll
Mutex nehmen
Punktestand aktualisieren
Client Auswertung der Frage schicken
Mutex freigeben
Semaphor für Score Thread wird gesetzt
Spielphase
Spielende

Score-Thread

Wird von Client und Login-Thread angestoßen

Player mit der höchsten Punktzahl wird zuerst gesendet

Wird nach Spielende von der main beendet

3.2. Wann müssen sich die Prozesse/Threads synchronisieren bzw. Kommunizieren?

- Der **Listener-Thread** muss mit dem Client-Thread kommunizieren, wenn Fragen bzw. ein Fragenkatalog geändert wurden/wurde.
- Der **Fragewechsel-Thread** kommuniziert mit dem Server (Client-Thread), wenn eine Frage beantwortet wurde bzw. wenn eine neue Frage an den Listener-Thread übermittelt werden soll..
- Der **Client-Thread** kommuniziert mit dem Spielstand-Thread, um den Punktestand zu aktualisieren. Dieser wird synchronisiert, wenn eine neue Anmeldung erfolgt.
- Der **Server** kommuniziert mit dem Loader über die Pipes, wenn der Spielleiter einen neuen Fragekatalog auswählt oder beim Spielstart ein neuer Katalog geladen wird.
- Der **Login-Thread** kommuniziert mit dem Client-Thread und dem Client, wenn der Login durchgeführt wird.
- Der **Client** und der **Server** kommunizieren über Sockets, wenn der Client Anfragen an den Server hat oder der Server an den Client Daten übermitteln möchte.

3.3. Welche Synchronisations- /Kommunikationstechniken werden wo verwendet?

- Der Client kommuniziert mit dem Server über Sockets.
- Der Server kommuniziert über Pipes/Shared-Memory mit dem Loader.
- Zwischen den einzelnen Threads findet die Kommunikation durch Mutexte/Semaphore statt.

4. Beschreibung komplexer Datenstrukturen

4.1. Datenpakete zwischen Client und Server (ist im RFC vorgegeben)

```
0             7             15             23
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Type        |  Length                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

Type:          uint8_t, gibt die Art der Nachricht an
Length:        uint16_t, Länge der nachfolgenden Zusatzdaten in Bytes
```

Das vorgegebene Protokoll setzt direkt auf das TCP auf.
Der Client kommuniziert mit dem Server über das TCP-Protokoll.
Es gibt folgende Datenpakete:

LoginRequest (LRQ)
Der LoginRequest ist für die Anmeldung am Client zuständig und wird anschließend an den Server gesendet.

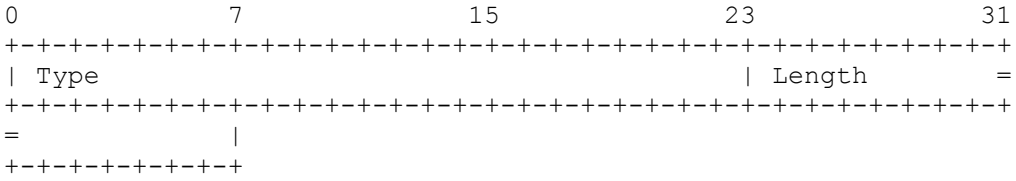
```
0             7             15             23              31
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Type                                        | Length        =
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
=             | RFCVersion    | Name ....................... |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

Type:          LRQ
Length:        Länge des Namens + 1 (Length <= 32)
RFCVersion:    uint8_t, dieses Semester = 6
Name:          Login-Name, UTF-8, nicht nullterminiert, maximal
               31 Bytes
```

LoginResponseOK (LOK)
Der Client sendet eine Login-Anfrage. Diese wird dann im Login-Thread des Servers entweder als „erfolgreich“ oder als „Anmeldung nicht möglich“ verbucht.

```
0             7             15             23              31
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Type                                        | Length        =
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
=             | RFCVersion    | ClientID      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

Type:          LOK
Length:        2
RFCVersion:    uint8_t, dieses Semester = 6
ClientID:      uint8_t, ID des Clients (0 := Spielleiter)
```

CatalogRequest (CRQ)

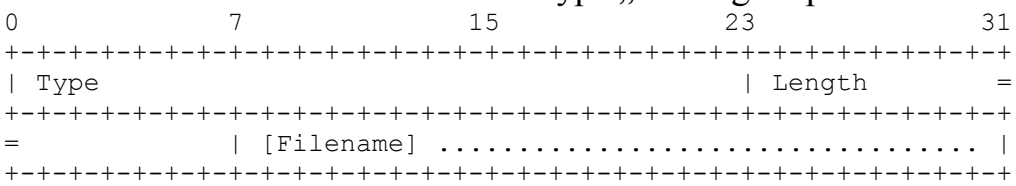
Der Client fordert die Liste der Fragenkataloge vom Server an. Der Loader greift auf den Speicher zu und gibt die Fragen an den Client weiter.




```
Type:           CRQ
Length:         0
```

CatalogResponse (QRE)

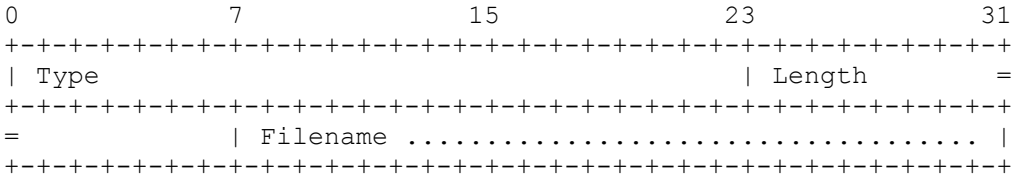
Der Server gibt dem Client eine Antwort. Die komplett-vollständige Liste ergibt sich aus mehreren Nachrichten des Typs „CatalogResponse“.




```
Type:           CRE
Length:         Länge des Dateinamens, oder 0 für Endemarkierung
Filename:       Dateiname eines Fragekataloges (UTF-8, nicht null-
                terminiert), oder leer als Kennzeichnung für Ende
                der Auflistung
```

CatalogChange (CCH)

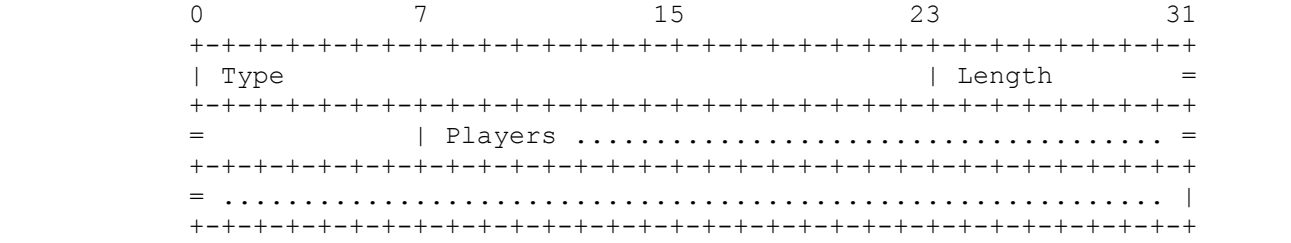
Der Spielleiter sendet diese Nachricht an den Server, wenn ein Fragekatalog in der Spielephase angeklickt wird. Die geänderte Nachricht wird anschließend vom Server an alle Clients (außer Spielleiter) gesendet.




```
Type:           CCH
Length:         Länge des Dateinamens
Filename:       Dateiname des gewählten Fragekataloges (UTF-8, nicht
                nullterminiert)
```

PlayerList (LST)

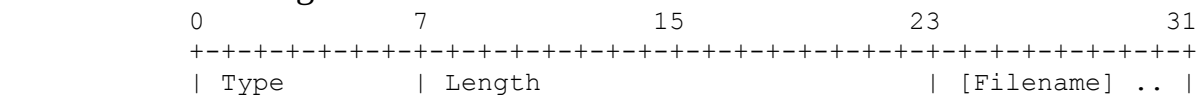
Der Server sendet die Liste der User an die Clients bei der An-oder Abmeldung. Der Spielstart und Änderung des Punktestands werden hier angegeben. Während der Spielphase muss diese Liste vom Server absteigend nach den Punktesständen der Spieler sortiert werden.




```
Type:           LST
Length:         37*Spieleranzahl (maximal 4*37 = 148)
Players:        Liste aller derzeit angemeldeten Benutzer, siehe unten

Aufbau der Spielerliste "Players":

        32 Bytes        Spielername 1 (UTF-8, nullterminiert)
         4 Bytes        Punktestand Spieler 1, vorzeichenlos
         1 Byte         ID Spieler 1

        32 Bytes        Spielername 2 (UTF-8, nullterminiert)
         4 Bytes        Punktestand Spieler 2, vorzeichenlos
         1 Byte         ID Spieler 2
                  .
                  .
                  .
```

StartGame (STG)

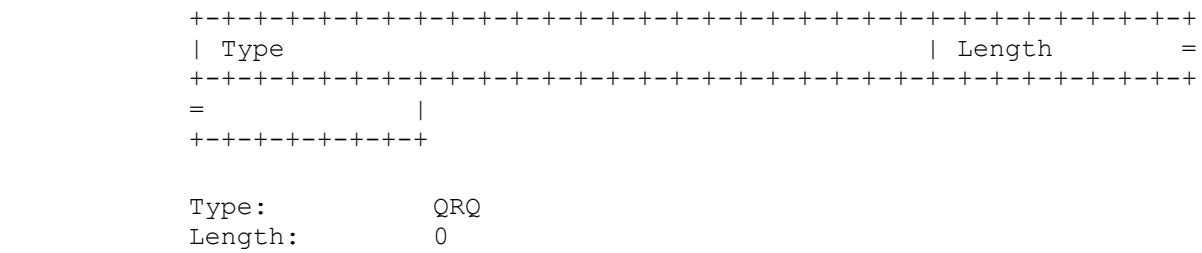
Das Spiel beginnt, sobald der Spielleiter das Spiel startet. Anfrage wird an den Server gesendet. Dieser leitet es an die Clients weiter.




```
Type:           STG
Length:         Länge des Dateinamens
Filename:       Dateiname des zu spielenden Fragekataloges (UTF-8, nicht
                nullterminiert), kann beim Versand Server ==> Client
                (nicht Client ==> Server!) auch leer gelassen werden
```

QuestionRequest (QRQ)

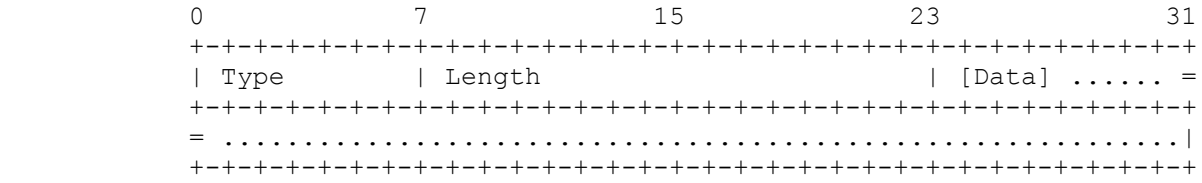
Der Client erfragt eine Quiz-Frage von dem Server an.



```
Type:           QRQ
Length:         0
```

Question (QUE)

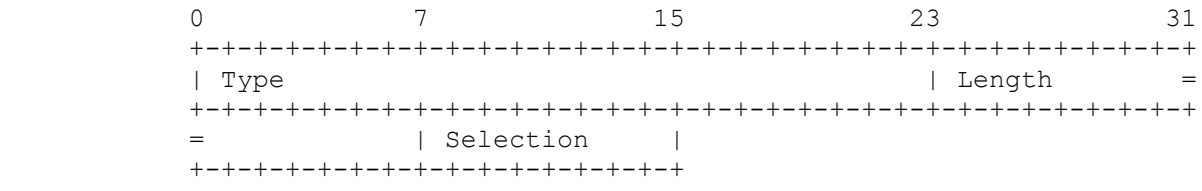
Der Server transportiert eine Quiz-Frage über den Server an die Clients mit dieser Nachricht.



```
Type:           QUE
Length:         769 oder 0 (falls keine Fragen mehr)
Data:           Eine Struktur, die wie unten angegeben aufgebaut ist,
                oder leer falls Ende des aktuellen Kataloges erreicht

Aufbau des Data-Felds:
        256 Bytes       Text der Fragestellung (UTF-8, nullterminiert)
        128 Bytes       Antworttext 1 (UTF-8, nullterminiert)
        128 Bytes       Antworttext 2 (UTF-8, nullterminiert)
        128 Bytes       Antworttext 3 (UTF-8, nullterminiert)
        128 Bytes       Antworttext 4 (UTF-8, nullterminiert)
          1 Byte        Zeitbegrenzung in Sekunden
```

QuestionAnswered (QAN)

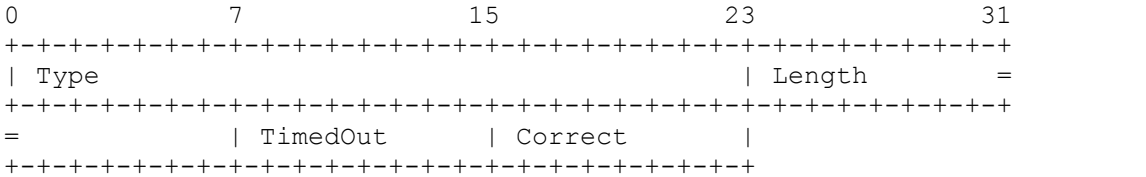
Der Client gibt eine Meldung an den Server, wenn eine Frage beantwortet wurde.



```
Type:           QAN
Length:         1
Selection:      uint8_t, Bitmaske der vom Spieler ausgewählten
                Antwortmöglichkeiten. Das niederwertigste Bit
                ist gesetzt, falls die erste Antwort ausgewählt
                wurde, das nächsthöhere Bit entspricht Antwort 2 usw.
                Die 4 höchstwertigen Bits werden nicht verwendet und
                dürfen nicht gesetzt sein.
```

QuestionResult (QRE)

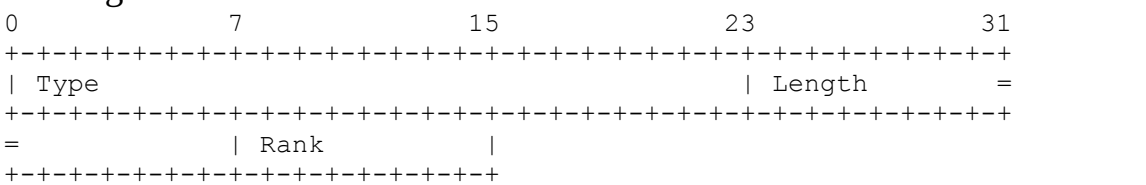
Die Antwort auf die Fragen wird vom Server an die Clients weitergesendet. In der Nachricht steht die Information, ob die Frage richtig oder falsch beantwortet wurde.




```
Type:           QRE
Length:         2
Selection:      uint8_t, wenn Timeout für Frage erreicht wurde
                ungleich 0, sonst 0
Correct:        uint8_t, Index der richtigen Antwort (0 <= Correct <= 3)
```

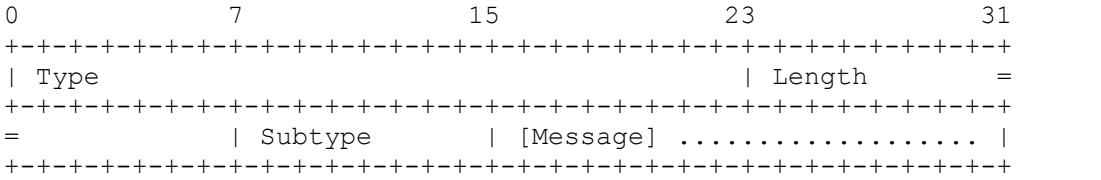
GameOver (GOV)

Alle Clients sind fertig mit dem Beantworten der Fragen. Die Rückmeldung und die Platzierung der Spieler erfolgt vom Server und die Nachricht wird an den Client gesendet.




```
Type:           GOV
Length:         1
Rank:           uint8_t, Endposition des Benutzers in der Rangliste
                (1 <= Rank <= 4)
```

ErrorWarning (ERR)
Falls ein Fehler auftritt, sendet der Server eine Fehlermeldung an die Clients (z.B. der Katalog kann nicht geladen werden, der Spielleiter verlässt den Server, Abbruch wegen weniger als 2 Spieler). Tritt z.B. bei zu langem Namen, zu wenig Teilnehmer, Verlassen des Spielleiters, Server voll... auf.




```
Type:           ERR
Length:         1 + Länge(Message)
Subtype:        uint8_t
                0 -> Warnung
                1 -> fataler Fehler, Client muss sich beenden
Message:        Beschreibung des Fehlers im Textformat (UTF-8), nicht
                nullterminiert

Eine ErrorWarning wird in folgenden Fällen gesendet:

Warnung:
* Katalog kann nicht geladen werden
* Spiel kann nicht gestartet werden, weil noch zu wenig Teilnehmer (nur
  während Vorbereitung, sonst fatal)

fatal:
* Login nicht möglich (z.B. Server voll, Spiel läuft schon oder Name
  bereits vergeben)
* Spielleiter verlässt den Server
* Spielabbruch wegen weniger als 2 Teilnehmern in der Spielphase
```

4.2. Shared Memory (ist vorgegeben)

Der Server selbst greift nicht auf alle Daten zu (sonst uneffizient), in dem Projekt ist der Loader dafür zuständig. Der Server ist dafür zuständig die Daten, die für das Spiel notwendig sind (z.B. Der Spielekatalog) an den Client zu geben. Wenn eine Anfrage von dem Client an den Server kommt, z.B. wenn eine neue Frage generiert werden soll, wird diese Frage aus dem Fragenkatalog des Shared-Memory entnommen. Der Loader greift auf den Speicher zu, lädt den Fragekatalog ins Shared-Memory, sodass auch der Server Zugriff darauf hat. Der Server liest die Daten aus dem Shared-Memory aus, und gibt sie dann an den Client weiter.

4.3. Userdaten+

4.4. eventuelle weitere Daten im Client oder im Server

Zum Laden eines Katalogs erzeugt der Loader ein Shared Memory damit der Server auf die Daten, die der Loader von der Festplatte lädt, zugreifen kann. Diese werden dann an den Client weitergesendet. Von dem geladenen Katalog kann der Server dann einzelne Fragen entnehmen, um diese an die Clients zu schicken. Die Daten landen dann beim Listener-Thread, welcher diese auf die Oberfläche ausgibt. Die Eingaben die der Spieler macht, werden im GUI-Thread erfasst und wieder an den Server weitergeleitet (falls der "Beenden"-Befehl nicht getätigt wurde). Dieser analysiert und wertet die Daten dann aus, aktualisiert den Punktestand (über den Spielstand-Thread) und schickt die Auflösung zurück an den Listener-Thread. Die Antwort wird von ihm ausgegeben und ein Fragewechsel kommt zustande. Dieser wird vom Fragewechsel-Thread übernommen. Damit die Auflösung noch ausgegeben werden kann, wartet der Fragewechsel-Thread eine kurze Zeit und sendet dann an den Server eine neue Anfrage. Falls der Server keine Fragen mehr zur Verfügung hat, sendet dieser eine Nachricht an den Client, der Listener-Thread reagiert und leitet die Beendigung ein.

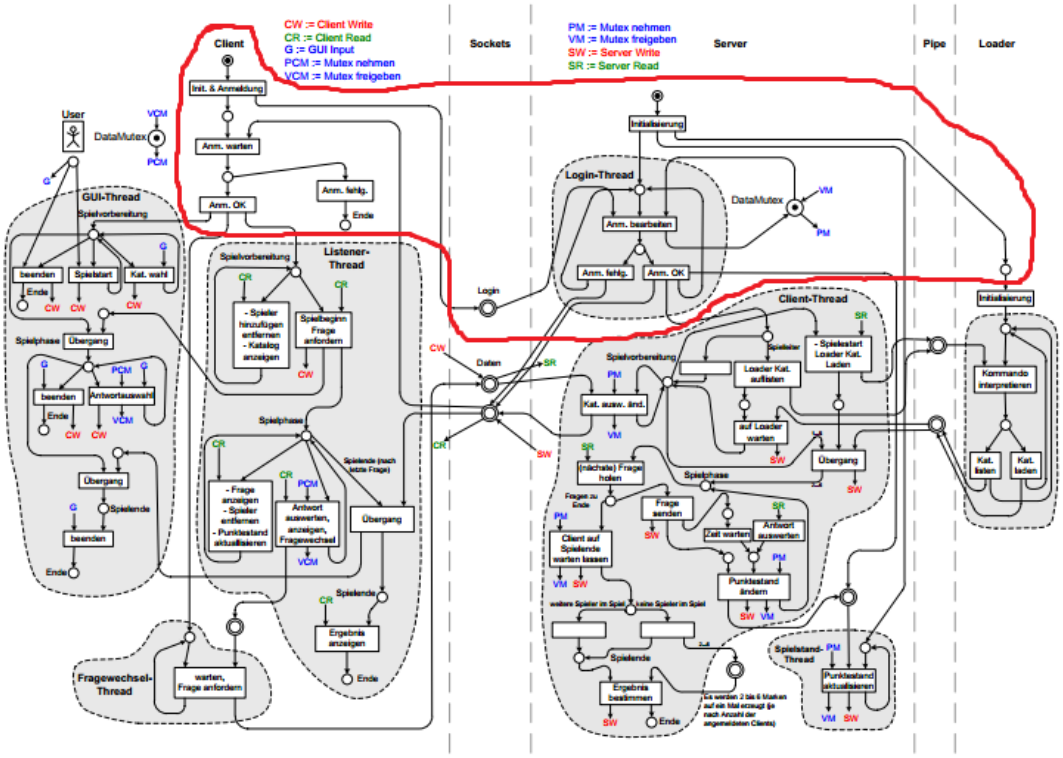
Userdaten sind Socketnummern, Namen der Spieler, Punktestand der Spieler, die in den Nachrichten zwischen den Threads und Prozessen ausgetauscht werden.

5. Programmentwurf

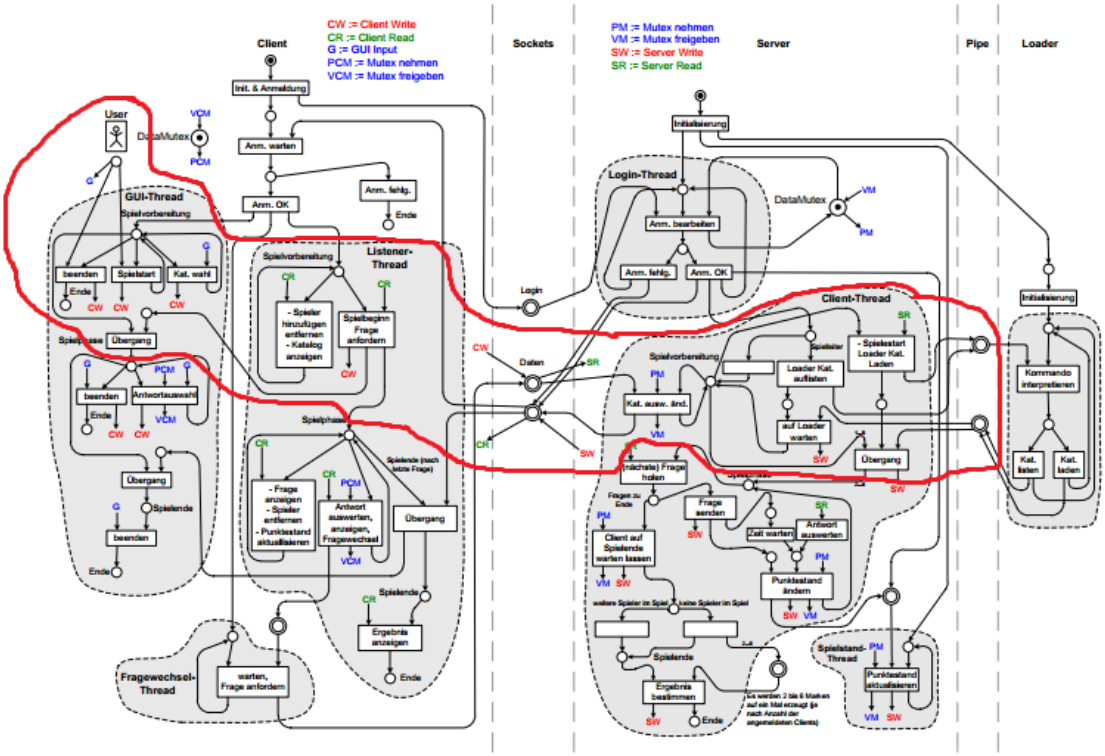
5.1. Welche Programmmodule sind zu realisieren?

Das Petrinetz und damit auch die Threads werden in einzelne Spielphasen eingeteilt. Diese wären Login-Phase, Spielvorbereitungs-Phase, Spielphase-Phase und Spielende-Phase.
In den jeweiligen Phasen werden entsprechende Funktionen verwendet (ungefähre Vorlage siehe 5.2), die vor allem für die Aufgabeneinteilung sinnvoll sind. Wie in der Aufgabeneinteilung (siehe 1.) beschrieben wird das Projekt in **Server**, **Client** und **Sockets** aufgeteilt.
Die Sockets werden vom Server und Client verwendet. Daher muss das RFC mit seinen Nachrichten für die Socket-Kommunikation verwendet werden.
Die einzelnen Phasen sind in unserem Projekt besitzen dementsprechend die **Module: Client, Server, Sockets (RFC)**.

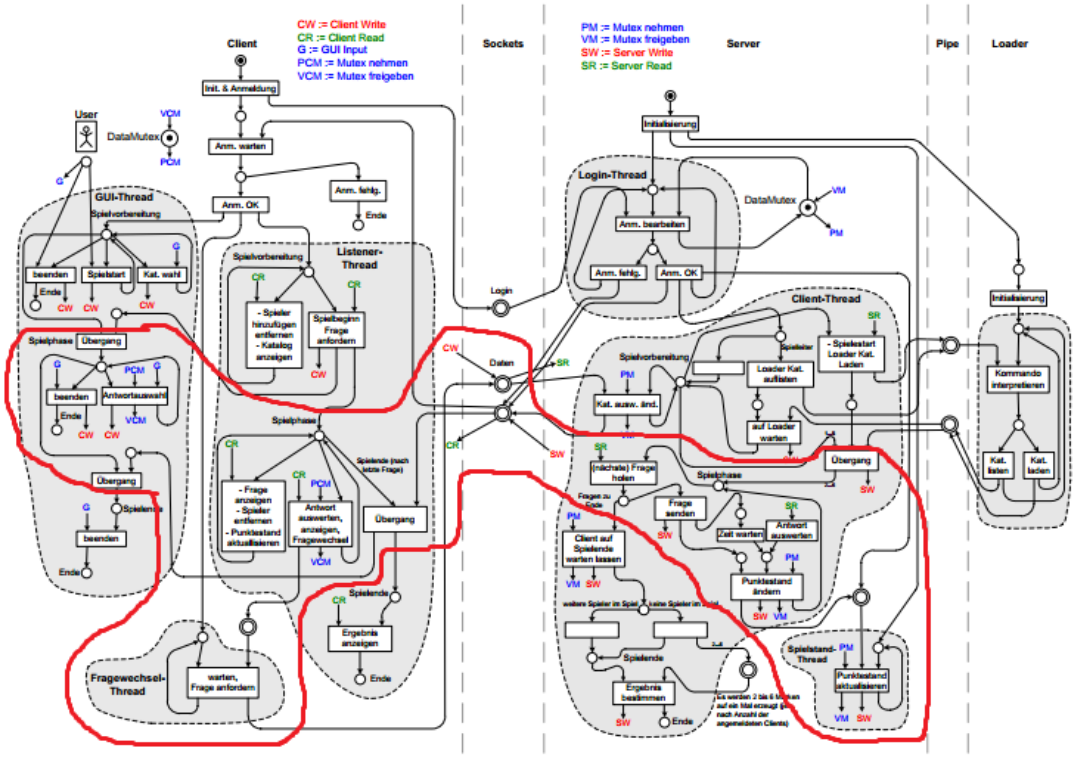
Login


CW := Client Write
CR := Client Read
G := GUI Input
PCM := Mutex nehmen
VCM := Mutex freigeben
PM := Mutex nehmen
VM := Mutex freigeben
SW := Server Write
SR := Server Read
Client
Sockets
Server
Pipe
Loader
User
DataMutex
Init. & Anmeldung
Anm. warten
Anm. OK
Anm. fehlg.
Ende
GUI-Thread
Spielvorbereitung
beenden
Spielstart
Kat. wahl
Spielphase
Übergang
Antwortauswahl
Spielende
Listener-Thread
- Spieler hinzufügen entfernen - Katalog anzeigen
Spielbeginn Frage anfordern
- Frage anzeigen - Spieler entfernen - Punktestand aktualisieren
Antwort auswerten, anzeigen, Fragewechsel
Ergebnis anzeigen
Fragewechsel-Thread
warten, Frage anfordern
Login
Daten
Initialisierung
Login-Thread
Anm. bearbeiten
Client-Thread
- Spielstart Loader Kat. Laden
Loader Kat. auflisten
auf Loader warten
Kat. ausw. änd.
[Initiale] Frage holen
Frage senden
Zeit warten
Antwort auswerten
Client auf Spielende warten lassen
Punktestand ändern
Ergebnis bestimmen
Spielstand-Thread
Punktestand aktualisieren
Kommando interpretieren
Kat. laden

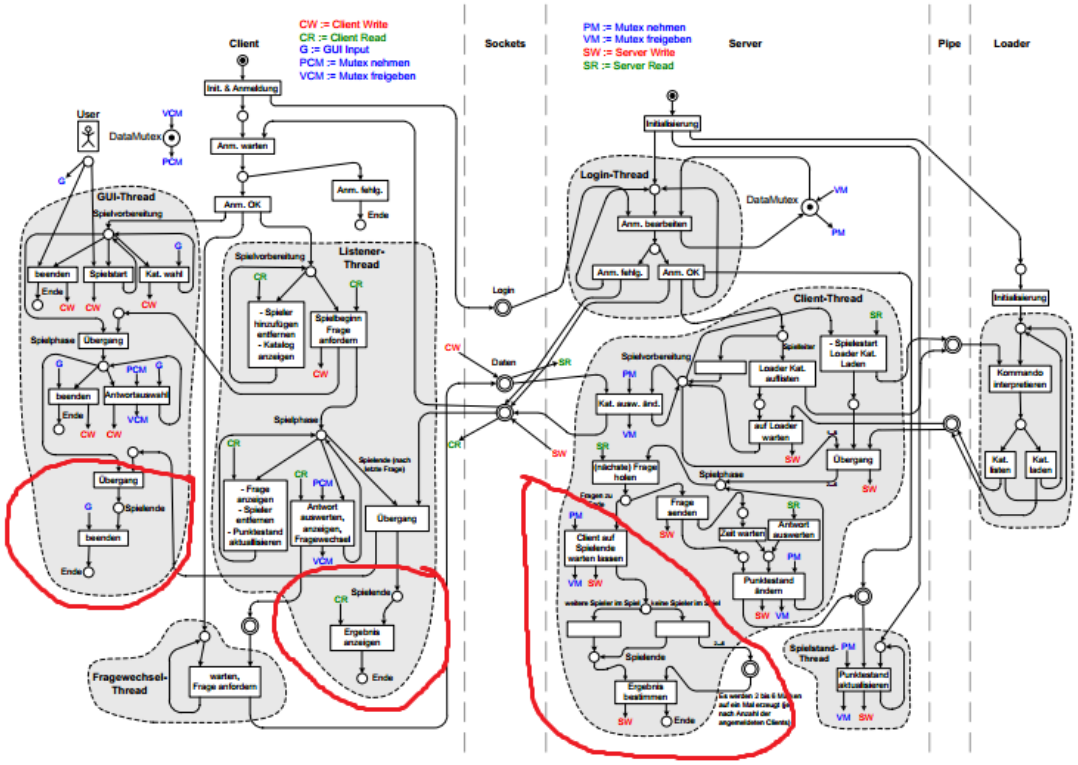
Spielvorbereitungsphase



Spielphase



Spielend-Phase


CW := Client Write
CR := Client Read
G := GUI Input
PCM := Mutex nehmen
VCM := Mutex freigeben
PM := Mutex nehmen
VM := Mutex freigeben
SW := Server Write
SR := Server Read
Client
Sockets
Server
Pipe
Loader
Init. & Anmeldung
User
DataMutex
Anm. warten
Anm. fehlg.
Ende
Anm. OK
GUI-Thread
Spielvorbereitung
beenden
Spielstart
Kat. wahl
Spielphase
Übergang
Antwortauswahl
Spielende
Listener-Thread
- Spieler hinzufügen entfernen - Katalog anzeigen
Spielbeginn Frage anfordern
- Frage anzeigen - Spieler entfernen - Punktestand aktualisieren
Antwort auswerten, anzeigen, Fragewechsel
Ergebnis anzeigen
Fragewechsel-Thread
warten, Frage anfordern
Login
Daten
Initialisierung
Login-Thread
Anm. bearbeiten
Client-Thread
- Spielestart Loader Kat. Laden
Loader Kat. auflisten
auf Loader warten
Kat. ausw. änd.
[nächste] Frage holen
Frage senden
Zeit warten
Antwort auswerten
Client auf Spielende warten lassen
Punktestand ändern
Ergebnis bestimmen
Spielestand-Thread
Punktestand aktualisieren
Kommando interpretieren
Kat. laden

5.2. Welche Funktionen enthalten diese Module?

Modul	Client-Funktionen	Server-Funktionen
Login	login();	login(); client_init();

listener	showCat();
	startGame();
	getQuestion();
	showQuestion();
	resultAnswer();
	showResult();
	guiUebergang();	

gui_thread
(gui_interface)
	startGame();
	choseCat();
	setAnswer();
	endGame();	

fragewechsel	sendQuestionRequest();	

main	gui_init();
	sock_init();
	list_init();
	questChang_init();
		login_init();
		//pipe_init();
		score_init();
		loader_init();

sync	hat Angaben zu Semaphoren und Mutexen	

user	set_user();
	get_user();
	add_user();
	set_userList();
	get_userList();
	akt_userList();
	sort_user();
	rank_player();	

rfc (socket)	send_msg_login();
	send_msg_prep();
	send_msg_game();
	send_msg_end();
	
	recv_msg_login();	

catalog
 	
score
 	
client_thread
 	
_____
server_loader_protocol
 	
util
_____
question
 	
_____
server_loader_interact();
	recv_msg_prep();
	recv_msg_game();
	recv_msg_end();
	
	list_cat();
	get_cat();
	send_cat();
	change_cat(); ->get + send
	
	result_score();
	set_score();
	get_score();
	change_score();
	
	start_game();
	send_waitForMe();
	send_changeScore();
	result_answer();
	logout_client();
	
	_____
	_____
	_____
	_____
	_____
	get_quest();
	send_quest();
	show_quest();
	_____
	_____
	init_pipe();	

5.3. Wer im Team ist zuständig für die Implementierung des Moduls?

- Implementierung Patrick Fetscher