PPB - Protokoll - Thomas Pleiner

Das Programm besteht aus Server-Klasse (mit Message-Handler und RequestHandler), Datenbank-Klasse, Game-Klasse und User-Klasse.
Der Zeitaufwand für das Projekt ist in einem eigenen PDF aufgeschlüsselt.

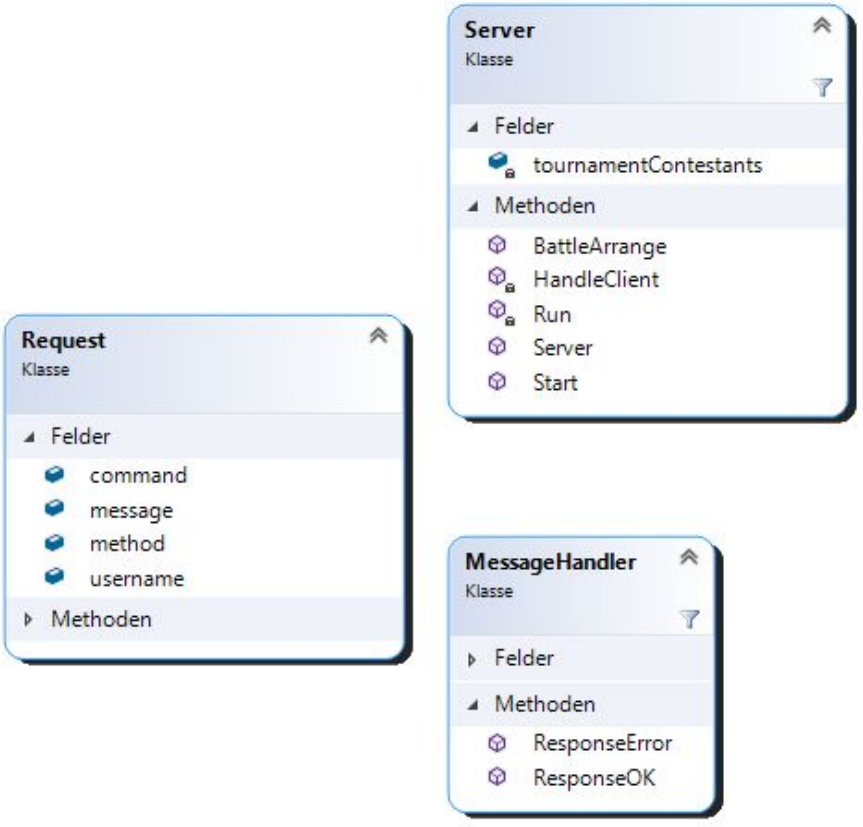
Datenbank Klasse

Diese Klasse speichert die notwendigen Querys sowie Verbindungsdaten zur Interaktion mit der Datenbank und führt diese bei Bedarf aus.

Server-Klasse und Threading

Aufbau

Der Server startet und zweigt dann für verschiedene User in jeweilige Threads ab. Hier musste seit dem letzten Stand, wie auch zuvor beim MTCG, wenig geändert werden. Requests an den Server werden in der Request-Klasse geparst und an den Message-Handler weitergegeben, welcher den nötigen Code ausführt und eine Antwort mittels der Methoden ResponeOK() und ResponseError() an den Client sendet.



Challenges & Solutions

Der Grundaufbau ist simpel, die Interaktion bei Anfragen von mehreren Threads eher weniger. Obwohl die Komplikationen bei mir meistens konzeptionelle Verständnis-Hürden waren, ließen sich die meisten Dinge in der Praxis relativ simpel mit Mutexes lösen.
Das allergrößte Problem war demnach, dass von verschiedenen Threads aus auf die gleichen Ressourcen zugegriffen werden muss.

Im Falle der globalen Playlist löse ich dieses Problem, indem ich gleich alles in der Datenbank speichere. Somit können verschiedene Threads auf die Playlist zugreifen.

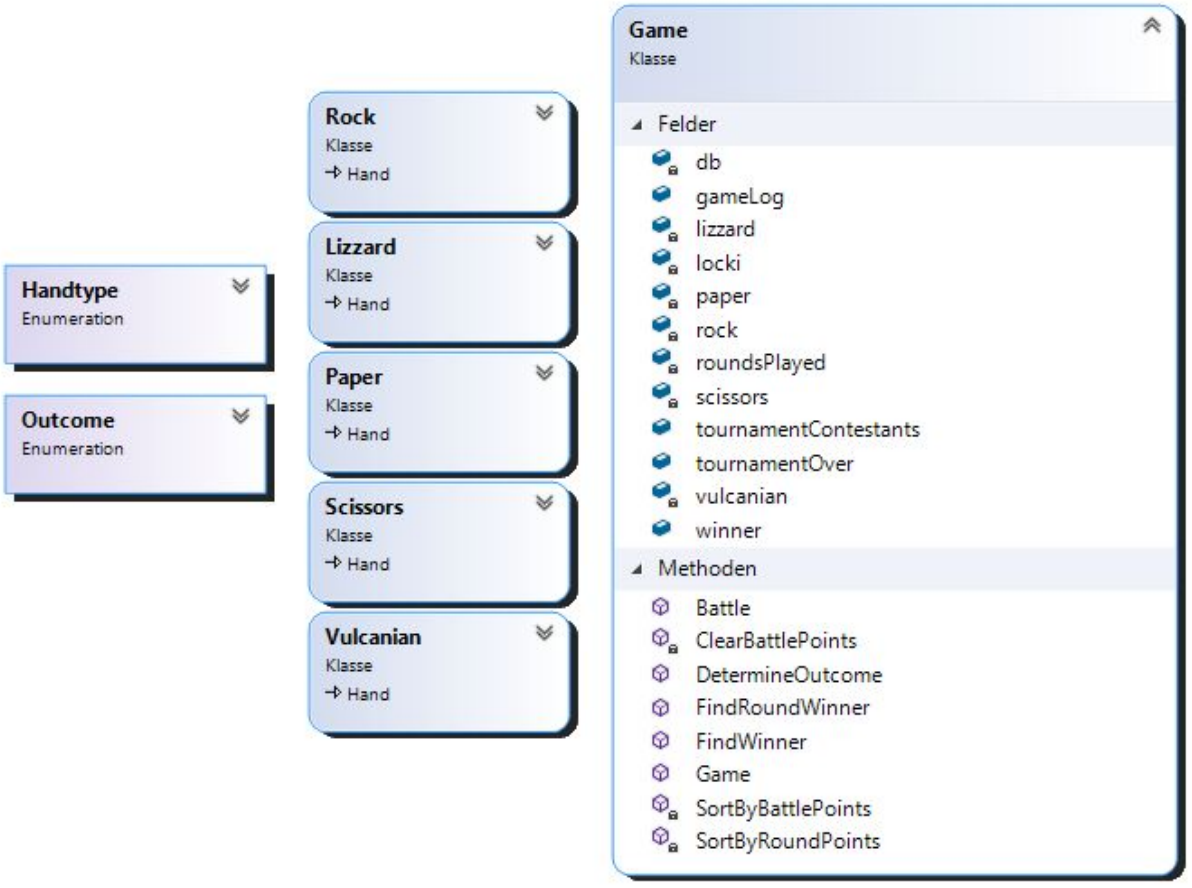
Im Falle der Game Klasse erfolgt die Zuordnung von Spielern zu Battles außerhalb der Threads in einer Methode namens BattleArrange(). Damit nur ein Game aufgerufen wird, existiert eine if else Condition, die diesen Fall abfängt. Mutexes verhindern schlussendlich, dass bei gleichzeitigem Aufruf von BattleArrange() durch mehrere Threads keine zwei Games entstehen.

Game-Klasse und User

Aufbau

Die Game Klasse wird aufgerufen, um für User mit einer gegebenen Menge an Handtypes, also einer Auswahl aus Schere, Stein, Papier, Echse und Spock, den passenden Sieger herauszufinden. Für diesen Zweck kommen Enumerations zum Einsatz. Die erste ist Outcome, welche ich nutze um Win, Lose und Draw schön gekapselt dem Programm zu übergeben, da diese Ausgänge besonders oft verwendet werden. Die zweite Enumeration kapselt die jeweiligen Handtypes.

In der Game Klasse wird für jeden Handtype eine Klasse initialisiert, welche speichert gegen welche Handtypes dieser Typ verliert. Alle verschiedenen Handtypes erben von einer abstrakten Klasse namens Hand.
Außerdem enthält jede Game Klasse eine Liste an Usern, welche verwendet werden um verschiedene Interaktionen mit der Datenbank und das Speichern der gewonnen Battle- und Rundenpunkte zu verwalten.



Challenges & Solutions

Eine erste Herausforderung in der Konzeption war das Regeln der Interaktion der verschiedenen Handtypes untereinander. Das ließ sich mit den verschiedenen Klassen ausreichend geordnet lösen. Ich überlegte zunächst die Interaktion in eine fixe 5x5-Matrix zu speichern, habe mich aber dann für Klassen entschieden, da diese leichter zu warten sind und den objektorientierter Ansatz anwendet.

Das eigentliche Programmieren an sich warf nur mehr wenige essentielle Fragen auf. Die größten Problem waren beim Bestimmen eines Gewinners im jeweiligen Game. In meinem Programm wird die Liste der Teilnehmer zuerst nach gewonnen Punkten sortiert, dann wird von oben nach unten kontrolliert, ob es Draws gibt. Der oberste Teilnehmer in der Liste, der kein Draw hat, gewinnt. Auf diese Weise wird sowohl der Runden-Gewinner als auch der Spiel-Gewinner bestimmt.