VSY Übung
Netzwerkfähiges Bullshit Bingo

April 8, 2011

Tobias Hegemann - 929668

Tobias Keinath - 929650

Axel Wilberz - 927147

Christoph Seip - 825141

Tommy Vinh Lam - 926419

Contents

Abstract

Beginnend mit der Aufgabenstellung und der Beantwortung der Frage "was ist Bullshit Bingo eigentlich" führt sie dieses Dokument durch die eingesetzten Mittel. Beiläufig werden kleinere Probleme die beim Arbeiten mit diesen Mitteln auftraten erörtert, schwerwiegendere Probleme allerdings haben eine eigene Rubrik. Nachdem noch kurz auf die Zusammenarbeit im Team eingegangen wurde folgt ein kleines Fazit zum gesamten Projekt.

Fangen wir hinten an und stellen zuerst die Frage was am Ende des Projektes vorhanden sein soll. Wir erörtern kurz welche Vorgaben realistisch sind und welche Ideen dahinter stecken.

1 Aufgabenstellung

Aufgabe ist is ein Buzzword Bingo als verteilte, grafische Anwendung zu entwickeln. Die Wahl der Plattform sowie der Art des verteilten Systems (Zentraler Server oder Ad-hoc Netzwerk) ist freigestellt.

2 Buzzword Bingo (Bullshit Bingo)

Anstelle von Zahlen sind hier Schlagwörter die in Meetings und ähnlichen mit Frasen und leeren Inhalten aufgeblähten Lesungen auftreten werden auf den ausgegebenen Zetteln abgedruckt. Sobald ein Mitspieler 5 solcher Schlagwörter in einer Reihe abgehakt hat steht dieser auf und hat dann "bullshit" zu sagen. Welches keinem wirklichen Gewinn entspricht.

3 Eingesetzte Mittel

Nachfolgend widmen wir uns den einzelnen Mitteln im Detail. Zunächst jedoch eine kurze Erläuterung, welche Kriterien der Entscheidung zu Grunde lagen

Als wichtigste Neuerung zu bisherigen Projekten die als Aufgaben an der HS Mannheim ausgegeben wurden steht die bis dato recht schwach beläuchtete Programmiersprache C#. Die Idee hinter C# war seine relativ ausführliche Netzwerkbibliothek sowie die zahlreichen vorhandenen Beispiele, anhand derer die Arbeit vermeindlich erleichtert werden sollte.

Nach kurzem zögern ob nicht vielleicht eine PHP Lösung als Ideal angesehen werden sollte, da die Entwicklung des Servers somit keine Rolle gespielt hätte, fiel die Entscheidung zugunsten einer Server- Client Lösung.

Die große Frage die hinter alle dem stand war: entweder es werden die vorhandenen Ressourcen in Form von bereits erstellter, verlgeichbarer Software genutzt

oder aber es werden völlig neue noch unbekannte Programme eingesetzt und somit ein möglichst großes Lernpotential erzielt. Zweiteres erhielt den Zuschlag.

3.1 Die Programmiersprache C#

C# greift Konzepte der Programmiersprachen JAVA, C++, Haskell, C sowie Delphi auf. C# zählt zu den objektorientierten Programmiersprachen und unterstützt sowohl die Entwicklung von sprachunabhängigen .NET- Komponenten als auch COM-Komponenten für den Gebrauch mit Win32-Applikationen.

Quelle: `de.wikipedia.org/wiki/C-Sharp`

C# erleichtert einem durch viele Erweiterungen und Bibliotheken die Arbeit. So wird in der hier besprochenen Softwarelösung vorrangig die .NET sowie die .NET.SOCKETS Bibliothek verwendet. Durch die es relativ leicht war Sockets zu erstellen und zu verwalten.

Oberflächen wurden mit dem in Visual Studio (auf das wir später genauer eingehen) bereits vorinstallierten Editor erstellt und durch die relativ einfache Verwaltung von Pannels organisiert.

3.2 Das Netzwerkkonzept

Wir haben uns für die vielfältigen Netzwerkbibliotheken von C# entschieden. Genauer gesagt nutzen wir die Klassen TcpListener und TcoClient, welche auf dem sehr verbreiteten Socket von C aufsetzen.

Während der Vorbereitung und Implementierung fielen zwei Anforderungen an Netzwerkklassen besonders auf: Die Non-Blocking Ausführung der Netzwerkaufgaben sowie die Möglichkeit der Kapselung der Anwendungslogik in Threads um ein asynchrone Netzwerkkommunikation zu schafen.

Während der Vorbereitung und Implementierung fielen zwei Anforderungen an Netzwerkklassen besonders auf: Die Non-Blocking Ausführung der Netzwerkaufgaben sowie die Möglichkeit der Kapselung der Anwendungslogik in Threads um ein asynchrone Netzwerkkommunikation zu schafen.

3.3 Visual Studio

Als Entwicklungsumgebung kam Visual Studio sehr gelegen. Trotz der oft verschrieenen Handhabung waren die msdn Hilfe sowie die Autovervollständigung sehr gut eingebunden.

4 Abstrakter Lösungsansatz

Die Software wird als Client- Server Anwendung realisiert. Als Model hierzu dient die Entwicklung eines Dedicated Server, der wahlweise eigenständig agiert und zum Beispiel parallel auf einem Client mit laufen kann oder separat auf einem externen Server läuft um den Client PC zu entlasten. Der Server hat lediglich eine Konsolenoberfläche und wird anhand von Befehlen gesteuert. Der Client jedoch besitzt eine vollständige Windows Forms Oberfläche. Anhand derer er sich in eine Art Lobby einklinkt und sobald mehr als ein Spieler darin enthalten sind das Spiel starten kann. Es sollte möglich sein sich in ein bereits laufendes Spiel einzuklinken.

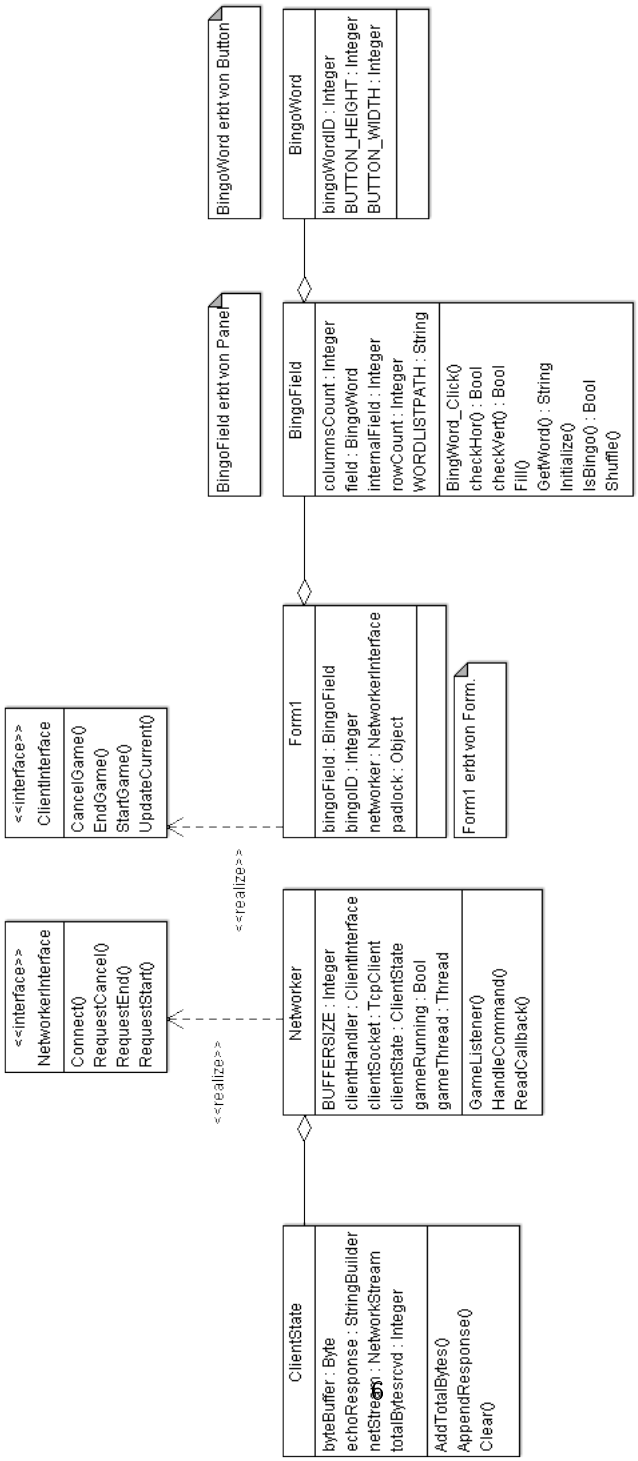
Die gesamte Spiellogik wird auf dem Client abgearbeitet. Hier wird überprüft ob der Spieler das richtige Wort angeklickt beziehungsweise bereits gewonnen hat. Ist dies der Fall wird eine Nachricht an den Server geschickt, der das Spiel dann beendet und jedem Teilnehmer eine Nachricht mit dem Namen des Gewinners weiterleitet.

Der Server verteilt Zufallszahlen. Die Clients weisen den Zahlen danach die richtigen Wörter aus einer Wortliste zu und zeigen diese am oberen Spielfeldrand an. Außerdem verarbeitet der Server die Namen aller Spieler in einer ArrayList und weist den Gewinner des Spiel aus.

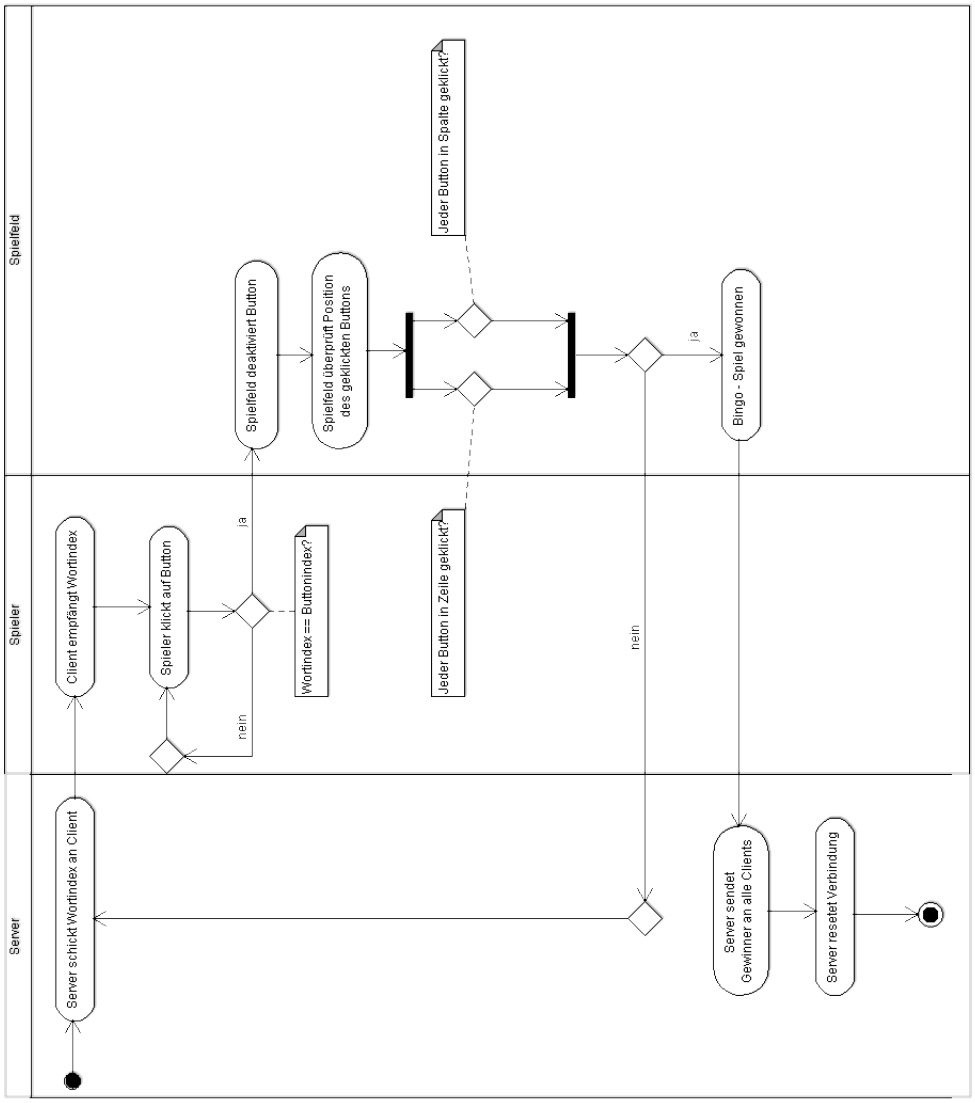
5 Detaillierte Vorgehensweise

Nachfolgend werden die einzelnen Schritte sowie die verwendeten Klassen und deren Methoden aufgezeigt. Die Wichtigsten davon werden etwas genauer beläuchtet und beschrieben.

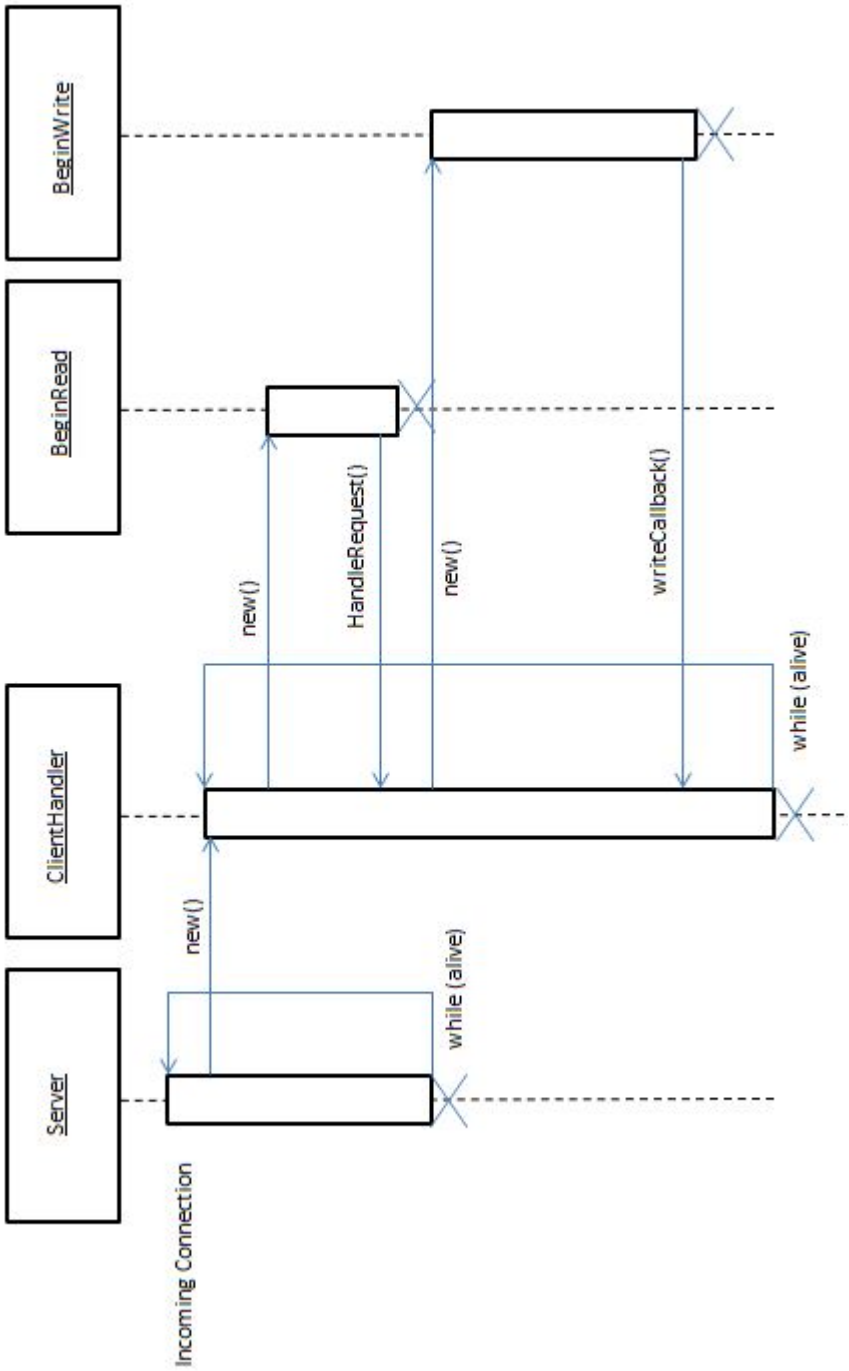
5.1 Klassendiagramm



5.2 Aktivitätsdiagramm



5.3 Sequenzdiagramm



5.4 Server-Klassen

Unser Server besteht aus einem Thread, der bei jeder neuen Verbindungsanforderung ein neues Client Handle Objekt erstellt und dieses in einem genau dieser Verbindung zugeordnetem Thread ausführt, bis die Verbindung abgebrochen oder serverseitig beendet wird.

`Server`: Diese Klasse repräsentiert den eigentlichen Server. Sie läuft in einem Thread und erstellt neue Clients falls dies erforderlich ist. Er beinhaltet die Methoden `.Start()`, `.Stop()` und `.Run()` und bekommt als Instanzparameter den Port übergeben.

`Game`: Wird ein neues Spiel erstellt wird zudem ein spezielles Game-Objekt als Hintergrund-Threat ausgeführt, welches in einem definierten Zeitintervall Zufallszahlen erzeugt und diese allen Clients zusendet. Hier wurde mit Locks gearbeitet um die Asynchronität zu wahren. Ihre Methoden sind `.Start()`, `.Stop()`, `.Run()` und `.WriteCallback(IAsyncResult asyncResult)`.

`ClientState`: In dieser Klasse wird der Status des Clients festgehalten. Jedesmal wenn der Client eine Änderung erfährt, wie zum Beispiel durch einen Disconnect wird dies durch die ClientState Klasse repräsentiert und an den Server weitergeleitet. Die Methoden sind im Einzelnen: `.AppendResponse(String response)`, `.AddTotalBytes(int count)`, `.Clear()`.

`ClientHandler`: Das ClientHandle-Objekt durchläuft während seiner Laufzeit eine Schleife, in welcher Signale empfangen, ausgewertet, bearbeitet und beantwortet werden. Die einzelnen Befehle sind: `START`, `CANCEL` und `END`. Die Methoden dieser Klasse sind wie folgt: `.Start()`, `.Stop()`, `.Run()`, `.HandleRequest()`.

5.5 Client-Klassen

`ClientInterface`: Dieses Interface dient der Klasse Form1 dazu die Methoden `.StartGame()`, `.UpdateCurrent(int currentID)`, `.EndGame(String winnerName)`, sowie `.CancelGame()` zu implementieren.

5.6 Netzwerk-Klassen

`NetworkerInterface`: Das NetworkerInterface dient der Networker Klasse. Es beinhaltet die abstrakten Methoden `.Connect(String ipAddress, int port, String name)`, `.RequestStart()`, `.RequestCancel()` und `.RequestEnd()`.

`Networker`: Die Klasse erbt von NetworkerInterface. Zu dieser Klasse gehören Objekte vom Typ `TcpClient`, `ClientState` und `ClientInterface`. Die Klasse Networker beinhaltet alle Methoden, die beim Aufbau eines Spiels benötigt werden. Unter anderem gibt die Methode `.Connect(String ipAddress, int port, String playername)` den PlayerNamen aus bzw. gibt eine Fehlermeldung zurück falls nicht verbunden werden konnte und sie erstellt einen Socket für den neuen Client. Außerdem beinhaltet die Klasse Networker noch folgende Methoden: `.RequestStart()`, `.RequestCancel()`, `.RequestEnd()`, `.ReadCallback(IAsyncResult asyncResult)`, `.HandleCommand(String command)`, `.GameListener()`.

5.7 Oberfläche

Die Verwendung von asynchronen Netzwerkmethoden bietet uns die Vorteile einer leichten und vorallem nicht blockierenden GUI-Interaktion, wodurch die Darstellung flüssig und problemloser erfolgt.

Form1: Die Oberfläche wurde mit einer einzigen Form erstellt, welche in Panels unterteilt wurde durch welche dann die einzelnen Fenster dargestellt sind. Die Panels sind im einzelnen:

- pnlConnection - Hier kann sich der Client durch Angabe von Port, IP-Adresse und Spielernamen mit dem Server verbinden.
- pnlStartMenue - Das Hauptmenü in dem der Spieler Navigiert.
- pnlInstruction - Hier findet man die Anleitung zum Bingo Spiel.
- pnlBingoWord - Zeigt das nächste Bingo Wort an.
- pnlLobby - Eine Auflistung aller zur Zeit bereitstehenden Spieler.

6 Aufgetretene Probleme und ihre Lösung

Im Laufe der Wochen mussten wir uns mit größeren und kleineren Problemen auseinandersetzen. Nachfolgend eine Auflistung der interessantesten Probleme aus unserer Sicht.

Sockets: Nachdem wir die Aufgabenstellung erhalten hatten wurde munter drauf losgezimmert und nach relativ kurzer Zeit war klar, ... so nicht. Um ein Verteiltes System zu erstellen waren eine Menge Grundkenntnise nachzuholen und nachzulesen. So musste zu aller erst erläutert werden, wie die Arbeit mit Sockets von statten geht und welche Probleme dabei auftreten können. Der beiderseitige Austausch von Daten war weit komplexer zu realisieren als zuerst angenommen. So war es uns möglich Daten vom Server aus zu senden und beim Client zu empfangen und umgekehrt, damit dies allerdings nicht in heilloses Durcheinander endete waren Klassen notwendig um diesen Ablauf zu synchronisieren. Aus 3 Basismethoden mit denen der Server schnell generiert war wurde schnell ein mehrere Klassen schweres Paket das sich um die einzelnen Anfragen kümmern und diese verarbeiten musste.

Zeit: Zeitmanagement wurde von uns als weniger wichtig empfunden. Und Zeitempfinden ist sehr relativ. So kamen uns 3 Wochen zur Fertigstellung lange vor... bis die ersten Probleme, sowie weitere prallele Projekte uns ins straucheln brachten. Am Ende hieß es Beine in die Hand um jede freie Minute zu nutzen. Die Lösung dieses Problems liegt auf der Hand. Meilensteine und eine sehr gute Zusammenarbeit im Team werden das nächste Projekt mit Sicherheit dominieren.

7 Kurze Erläuterung zur Teamarbeit

Die Teamarbeit lief relativ unproblematisch von statten. Die Ressourcen im Team selbst wurden ausreichend genutzt und die Aufgaben den Fähigkeiten entsprechend verteilt. Es gab außerdem niemals die Gier nach ausgleichender Gerechtigkeit. Zunächst wurde ermittelt, wer sich in welchem Bereich am besten auskennt um anschließend Aufgaben zuweisen zu können. So wurde das 5 Köpfige Team in 2 Netzwerkentwickler, 2 Qualitätsmanager und einen Oberflächenguru unterteilt. Diese Rollen jedoch wechselten gelegentlich und wurden auch ohne Probleme angenommen. Das Teammitglied mit der größten Programmiererfahrung wurde einstimmig und ohne öffentliche Abstimmung als Leiter des Ganzen anerkannt.

8 Fazit

Durch die größe des Projektes und die geringe uns zur Verfügung stehende Zeit, war die gut organisierte Arbeit im Team unerlässlich. Arbeits - und Aufgabenteilung waren Grundvoraussetzung für das Gelingen der Aufgabe. Es waren bei Fehlplanungen manche Teammitglieder dazu gezwungen Sonntage und sehr viel Freizeit zu opfern. Aus heutiger Sicht hätte vieles einfach besser organisiert und geplant werden müssen um ein noch besseres Ergebnis zu erzielen. Alles in Allem war das Projekt eine sehr gute Einstimmung auf das vor uns liegende MEP.