関西大学『社会学部紀要』第45巻第1号，2013，pp.93-127　　ISSN 0287-6817

資料

Visual Basicによるネットワークプログラミングと社会心理学への応用

社会的ジレンマを理解するためのネットワーク・ゲームの作成

久　本　博　行

Network programming with Visual Basic and its application to social psychology

Creating of network games to understand social dilemmas

Hiroyuki HISAMOTO

Abstract

It was explained that the basic method of network programming by programming the chat system created with Visual Basic. In addition, it was created, and introduced that the network game to facilitate an understanding of the structure of students' social dilemmas as an application to the social psychology experiment.

Key words: network programming, social dilemmas, network games, Visual Basic

抄　録

Visual Basicで作成したチャットのプログラムを使ってネットワークプログラミングの基本的な方法を解説した。さらにその社会心理学実験への応用として，学生に社会的ジレンマの構造について理解を促すためのネットワーク・ゲームを作成し，紹介した。

キーワード：ネットワークプログラミング，社会的ジレンマ，ネットワーク・ゲーム，Visual Basic

１．はじめに

社会心理学では，実験の中で参加者間のコミュニケーションが必要なものやコミュニケーションそのものを研究対象としたものなど，研究要素にコミュニケーションが関っているものが多くあります。

Bavelas, A.（1950）やLeavit, HJ.（1951）は５人グループの幾何学的な図形のコミュニケーション・パターンを４つ設定し（図1.1），コミュニケーション・パターンとグループのパフォーマンスの関係について，実験を行っています。

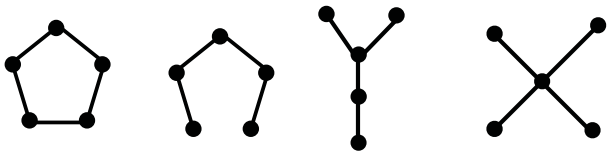

図 1.1　コミュニケーション・パターン（Leavit, H. J.（1951）より）

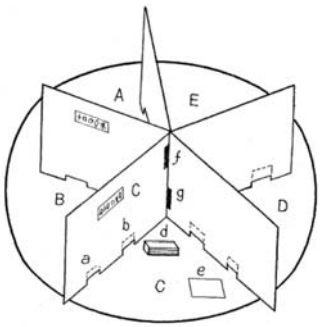

図 1.2　実験装置（廣田（1953）より）

この実験については，廣田君美（1953）も同様の実験を行っています。左図は廣田の実験装置です。大きな円形テーブルの上に，左図のように５人の実験参加者を隔てる５枚の衝立を用意し，紙を使ってメッセージの交換を行うようになっています。メッセージを交換する口を開いたり，閉じたりすることによって４つのコミュニケーション・パターンが作れるように工夫されています。

現在では，このような実験にコンピュータ・ネットワークを利用して，より容易に実験することができます。実際にコンピュータ・ネットワークを利用して実験しているものも現れてきています。

Latane, B. & L'Herrou, T.（1996）は，ダイナミック社会的インパクト理論の検証に電子メールをコミュニケーション手段として使っています。森尾（2007）も同じくダイナミック社会的インパクト理論の検証にパーソナル・コンピュータ（以下PC）を利用したシステムで実験を行っています。

柿本・細野（2010）は，広瀬（1997）の仮想世界ゲームの電子試作版を作成し，従来型

のものと比較を行っています。

コンピュータ・ネットワークを簡単に利用できるようになってきたとはいえ，まだ十分に活用できていないことも事実です。その理由としては，柿本らも指摘しているように，コンピュータ・ネットワークを利用したコミュニケーションでは，現実感が乏しくなりがちであることが一つ挙げられます。また，コンピュータ・ネットワークを利用した実験プログラムを作成することが，技術的に難しい部分があるということも挙げられます。

しかし，コンピュータ・ネットワークを利用する長所もあります。

(1) Bavelas, A.のようなコミュニケーション・パターンを作成することが必要な場合，コミュニケーションの経路を比較的容易にアレンジできる。

(2) コンピュータ上で実験を行うので，データがそのままコンピュータに残り，データを活用しやすい。

また，例えばAxelrod, R.（1980）が行った対戦型の囚人のジレンマゲームでは，プログラムはFortranで作成し，インターフェースが統一されていましたが，コンピュータ・ネットワークを利用すると，プログラム言語を統一することやOSを統一することは不要であり，通信プロトコルさえ合わせれば，遠隔地にいても対戦が可能になります。

コンピュータ・ネットワークの利用はこのような長所を持っていますが，プログラムを作成することが難しいという問題があります。そこで実験プログラムを容易に作成できるようにここでは，コンピュータ・ネットワークの基礎知識をまず解説し，Visual Basicで作成したチャットのプログラムでネットワークプログラミングの基本的な方法を説明します。さらにその社会心理学実験への応用として，Visual Basicで作成した学生に社会的ジレンマの構造について理解を促すためのネットワーク・ゲームを紹介します。

このゲームは，関西大学社会学部の学生が林直保子教授の授業の中でグループワークによって原案を考案し，林教授が若干の修正を加えられたもので，最初はコンピュータを使わずに行われていたものを，コンピュータ・ネットワークを使って行えるようにしたものです。

2．ネットワークプログラミングの基礎

ネットワークプログラミングをする上で必要な事柄について，簡単にまとめておきます。

2.1　プロトコル

コンピュータで通信を行う場合，通信の手順をあらかじめ決めておく必要があります。

通信の手順とは，あるコンピュータが特定の情報を相手に送った場合，その相手はどのような情報を送り返すか，といった情報のやり取りの順番です。私たちが電話をかける場合，初めに「もしもし」と言って相手に聞こえているかどうか確認し，「はい」といった返答があれば，話し始めるということをします。人間が電話で会話を行う場合，こうした手順を明確に決めておく必要はなく，「もしもし」と言わずにいきなり話し始めてもあまり問題は起こりません。しかし，コンピュータ同士の通信では最初に接続に行ったコンピュータが「モシモシ」というメッセージを送り，次にその応答として，もう一方のコンピュータは「ハイ」というメッセージを返す，というように通信手順を明確にしておかなければうまく通信できません。その通信手順をプロトコルといいます。

プロトコルには色々なものがありますが，現在もっともよく使われているプロトコルはTCP/IP（Transmission Control Protocol/Internet Protocol）というものです。これはインターネットやLAN（Local Area Network）の標準的なプロトコルとなっています。ここでもTCP/IPを使って通信を行うプログラムを作成します。

2.2 IP（Internet Protocol）アドレス

電話の場合，電話番号が各電話機に割り振られています。また各戸には住所があり，郵便が届くようになっています。コンピュータ間で通信を行う場合も，電話番号や住所に相当するものが必要になります。TCP/IPでこの役割を果たすものがIPアドレスです。現在IPにはVersion4（IPv4）とVersion6（IPv6）があります。IPv4とv6の大きな違いはIPアドレスの長さでv4は32bitに対し，v6は128bitあります。v4のアドレスは2^{32}（約42億）個に対し，v6は2^{128}（約3.4×10^{38}）個あります。v4のアドレスは，もうすぐ枯渇するのでv6への移行がいろいろなところで試行されています。しかし，まだほとんどはv4のアドレスを使用しているので，ここでもv4のアドレスを使ったプログラミングを行います。

2.3 ポート番号

IPアドレスでどのコンピュータと通信を行うかを特定できますが，ネットワークを利用するプログラムはWebブラウザー，メールの送受信を行うメーラーなど複数あります。そのため受信したデータをどのプログラムが処理をするのかを区別する必要があります。ポート番号はこのプログラムを識別するためのものです。

ポート番号は，well-knownポート番号，registeredポート番号，dynamic/privateポート番号の3種類に大別されています。このうちwell-knownポート番号は0〜1023でIANA

（Internet Assigned Numbers Authority）によって管理されており，勝手にこれを使うと混乱が起こる恐れがあるので，使わない方が安全です。表2.3は well-known ポート番号でよく使われるものを示しています。

registered ポート番号は1024～49151でこれも IANA に登録されているものです。dynamic/private ポート番号は49152～65535で自由に利用できるものです。

表 2.3　well-known ポート番号の一例

サービス名	ポート番号
ftp	21
telnet	23
smtp	25
http	80
pop3	110

2.4　ネットワーク・システムの構成

ネットワーク・システムの構成には，大きく分けてクライアント・サーバーシステムとピアツーピア（peer to peer）システムがあります。クライアント・サーバーシステムでは，ユーザーが使用するコンピュータをクライアントと呼び，クライアントからの処理要求に対して種々の処理を行い，応答を返すコンピュータをサーバーと呼びます。LAN（Local Area Network）やインターネット上で利用されているネットワークシステムのほとんど（Web，メールなど）がこの構造のものです（図2.4）。

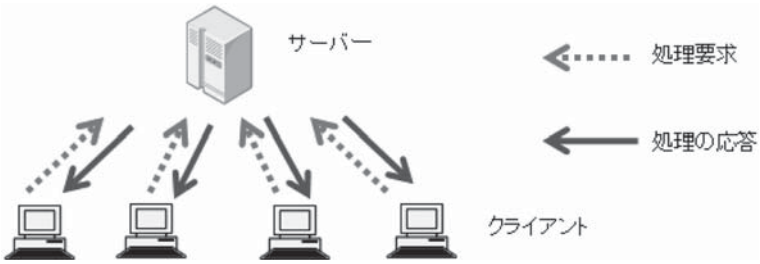

図 2.4.1　クライアント・サーバーシステム

ピアツーピアシステムは，ネットワーク上のコンピュータが対等な立場で，情報の交換を行うシステムです。

ここでは実験者がサーバーを操作し，実験参加者がクライアントを操作するクライアント・サーバーシステムを作成します。

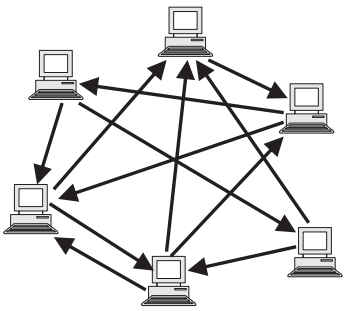

図 2.4.2　ピアツーピアシステム

2.5　マルチスレッド（multithread）

スレッド（thread）は「筋道」を意味する言葉ですが，プログラミングでは処理の流れを指します。マルチスレッドは１つのプログラムの中で，複数の処理の流れを並列に実行することを言います。

ネットワークプログラミングでは，通常ユーザーインターフェースの処理を行うスレッドと通信処理を行うスレッドに分けてマルチスレッドで処理を行います。それは，ユーザーインターフェースの処理と通信処理を並列に実行できるようにしておかないと，プログラムがデータの受信待ちになった場合，データを受信できるまでユーザーインターフェースの操作を全く行えなくなるからです。

マルチスレッドで注意しなければいけないのは，共有している情報へのアクセスに対して排他制御することです。

例えばＡ，Ｂ２つのスレッドが共有の DataList という配列変数にデータを入れていくことを考えます。その時に現在どこまでデータが入っているかを知らないとデータを入れる場所が分からないので，それを示す共有の変数 Counter を用意します。それぞれのスレッドはデータを入れる際には，Counter に１を足してそれをデータの挿入位置とし，DataList にデータを入れます。

今Ａ，Ｂ２つのスレッドが同時に Counter に１を足しデータを挿入した場合，同じ位置にデータが入れられてしまい，どちらかのデータが欠落してしまいます。そこで，こうした共有変数にアクセスする場合，アクセスしようとするスレッドが共有変数にロックをかけ，他のスレッドがアクセスしようとすると待たされるようにします。そして，共有変数の更新が終わった後，ロックを解除するようにします。このような処理を排他制御といいます。

排他制御をしない場合の処理の順序

スレッドＡの処理	スレッドＢの処理
① Counter ＋ 1	
	② Counter ＋ 1
③ DataList（ Counter ）＝ data1	
	④ DataList（ Counter ）＝ data2

排他制御をしない場合では，上記のように①～④の順にプログラムが実行されることが起こりえます。この場合スレッドＡでカウンター変数に１を足し，次に配列 DataList の

Counter の位置にデータを入れるのですが，その間にスレッドＢがカウンター変数に１を足した場合には，スレッドＡ，Ｂともに同じ位置にデータを入れてしまい，先に入れた方のデータが欠落するという現象を起こします。

このような現象を防ぐためにカウンター変数に１を足し，配列変数にデータを入れるという部分全体にロックをかけます。SyncLock から End SyncLock の部分がロックをかけている部分です。ロックをかけておくと，ロックが終了するまで他のスレッドは待たされますので，データの整合性を保つことができます。

排他制御を行う場合の処理の順序

スレッドＡの処理	スレッドＢの処理
① SyncLock　Obj	
②　　Counter ＋ 1	
③　　DataList（ Counter ）＝ data1	
④ End SyncLock	
	⑤ SyncLock　Obj
	⑥　　Counter ＋ 1
	⑦　　DataList（ Counter ）＝ data2
	⑧ End SyncLock

上記のように排他制御を行った場合，①～⑧の順で実行され，処理の途中で他のスレッドの処理が割り込むことはありません。

３．簡単な通信プログラム

初めに簡単なチャットのシステムを作成してみます。

3.1　プログラムの構成

図3.1.1はクライアントのウィンドウです。クライアントの操作は，サーバーの IP アドレスを入力した後「接続」ボタンを押します。サーバーに接続できれば，メッセージを入力し「送信」ボタンを押すとメッセージが送信されます。

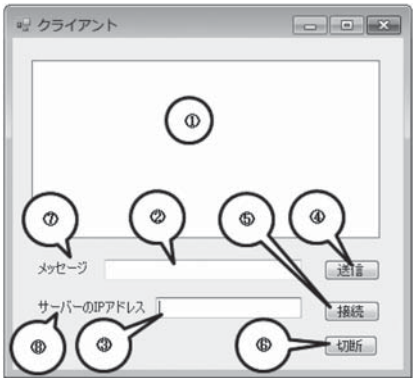

図 3.1.1　クライアント

接続を切る時は，「切断」ボタンを押します。

表 3.1　クライアントウィンドウのコントロール

番号	オブジェクト	機能
①	ListBox1	受信メッセージの表示
②	TextBox1	送信メッセージの入力
③	TextBox2	サーバーの IP アドレスの入力
④	Button1	メッセージを送信する
⑤	Button2	サーバーと接続する
⑥	Button3	サーバーとの接続を断つ
⑦	Label1	Text: メッセージ
⑧	Label2	Text: サーバーの IP アドレス

図3.1.2はサーバーのウィンドウです。サーバーのウィンドウでは，受信したメッセージを表示するリストボックス（ListBox1）があります。サーバーには特別な操作はなく，メッセージを表示するだけです。

図 3.1.2　サーバー

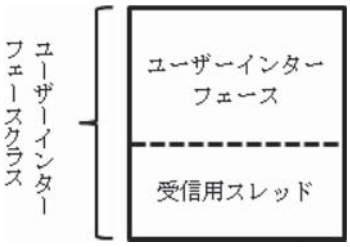

図 3.1.3　クライアント

クライアント・プログラムの大まかな構造は図3.1.3のような形になります。全体は１つのクラスで成り立っており，ユーザーインターフェースの部分がメインスレッド，通信のうち受信機能を別のスレッドとしています。受信部分を別スレッドとして動かしておかないと，受信待ちになった場合ユーザーインターフェースが全く使えなくなるからです。

別スレッドとして動かすと受信待ちになっても，受信スレッドはデータを受信するまで動きませんが，その他のスレッドは並行して処理することができます。

サーバー・プログラムの構造は図3.1.4のような形になります。マルチクライアントに対応したサーバーでクライアントより複雑になります。全体はユーザーインターフェースを行うクラスとクライアントと通信を行うクラスの２つのクラスから成り立っています。ユーザーインターフェースを行うクラスのインスタンスは１つですが，通信を行うクラスのインスタンスはクライアントの数だけ生成し，クライアント毎に割り当てるようにします。

ユーザーインターフェースクラスでは，ユーザーインターフェースの処理を行うメインスレッドとクライアントからの接続処理を行う待機スレッドに分かれています。一つのスレッドで処理を行うと，クライアントからの接続を待っている間ユーザーインターフェースの処理ができなくなるためです。

クライアントとの接続ができると，そのクライアント専用の通信用クラスのインスタンスを作成し，そのクライアントとの通信はすべて通信用クラスに任せます。通信用クラスもメインスレッドと受信用スレッドに分かれて並列で処理を行うようにします。

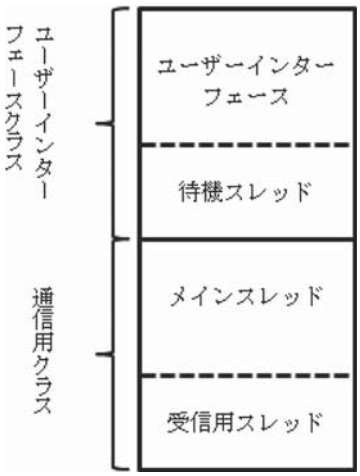

図 3.1.4　サーバー

3.2　通信手順

クライアントとサーバーの通信手順は，以下のようになります。

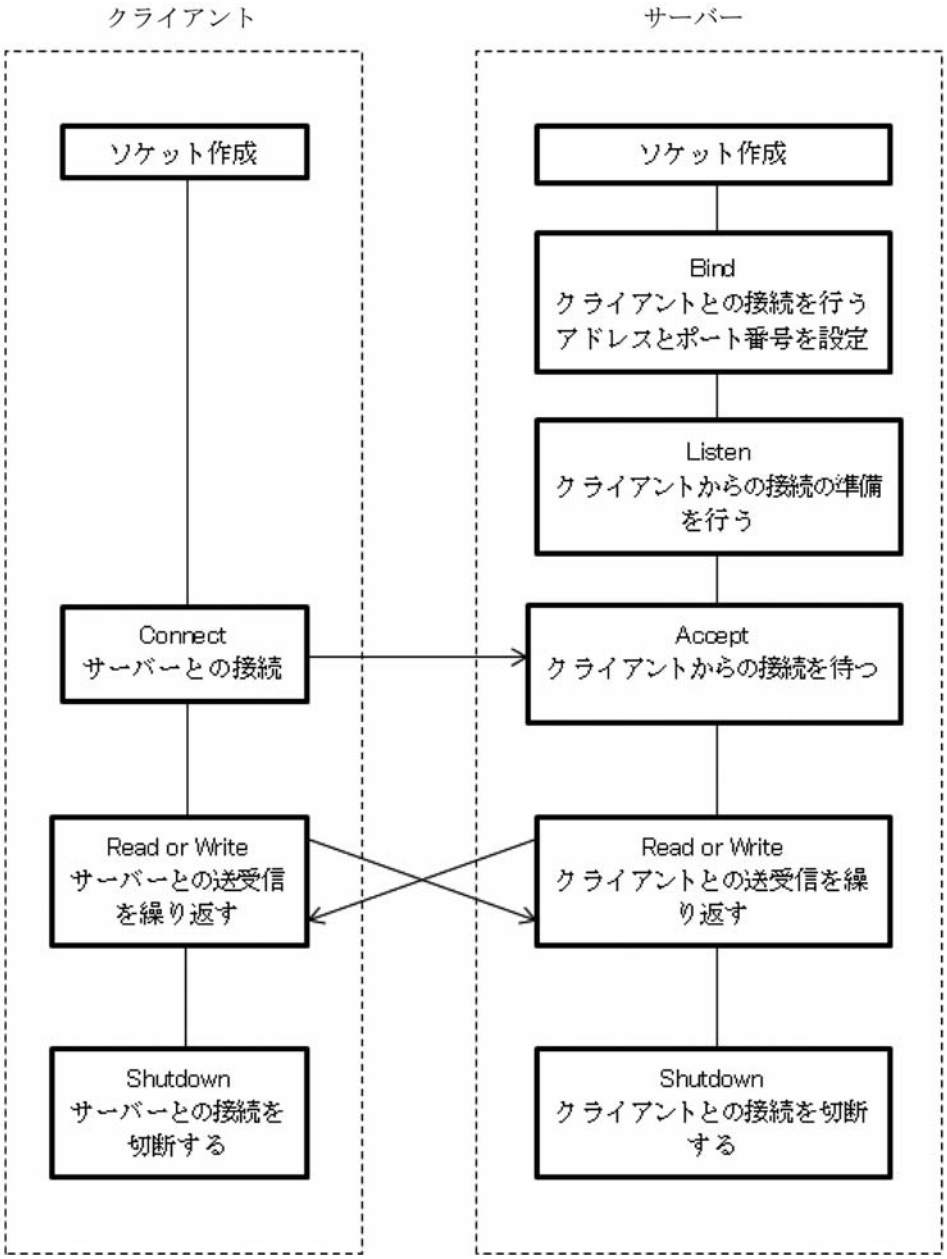

図 3.2.1　クライアントとサーバーの通信手順

3.3　クライアント・プログラムの解説

クライアントのコードについて解説していきます。

```
1:Imports System.IO
2:Imports System.Net
3:Imports System.Net.Sockets
4:Imports System.Text
5:Imports System.Threading
6:
7:Public Class Form1
8:
9:    Dim CIntSocket As Socket              '通信用ソケット
10:   Dim CIntStream As NetworkStream       'ネットワークストリーム
11:   Dim CIntReader As StreamReader
12:   Dim CIntWriter As StreamWriter
13:   Dim CommThread As Thread              '通信用スレッド
14:
15:   Delegate Sub DisplayDlgt(ByVal s As String)
16:   Delegate Function MsgDlgt(ByVal owner As IWin32Window, _
                                ByVal s As String, ByVal cptn As String, _
                                ByVal btn As MessageBoxButtons, _
                                ByVal icn As MessageBoxIcon)
```

1行目～5行目はクラスの名前空間の参照を容易にするための設定です。それぞれの名前空間の内容は，下表のとおりです。

表 3.3.1　Imports する名前空間の内容

名前空間	内容
System.IO	ファイルやストリームの入出力
System.Net	ネットワークの種々のプロトコルのインターフェース
System.Net.Sockets	Windows ソケットのインターフェース
System.Text	文字コードのエンコーディング
System.Threading	マルチスレッド

9行目～13行目は，クラス内で共通に使う変数を宣言しています。

15，16行目は，Delegate の宣言です。

このプログラムでは，受信用スレッドを使って交信処理を行っていますが，受信用スレッドで受信したメッセージを Form1の ListBox1に直接書き込むことができません。これは Windows フォーム上のコントロールに，複数のスレッドからのアクセスがある場合，アクセスが競合し正しい結果が反映されない可能性があるからです。

このような場合，Delegateを通してコントロールにアクセスするプログラムを呼び出します。そうするとコントロールのあるFormと同じスレッドでそのプログラムを実行させることができるのです。

15行目は，受信用スレッドからListBox1に書き込むプログラムのDelegateで，16行目はMessageBoxを表示するためのDelegateの宣言です。

次に接続ボタンをクリックしたときの処理について解説します。

```
17:   ' 接続ボタンの処理
18:   Private Sub Button2_Click(sender As System.Object, e As System.EventArgs) _
                                                          Handles Button2.Click
19:       Dim SvrHostEntry As IPHostEntry
20:       Dim SvrAddress As IPAddress = Nothing
21:       Dim SvrPort As Integer = 50000
22:       Dim SvrEndPoint As EndPoint
23:       Dim i As Integer
24:
25:       Try
26:           ClntSocket = New Socket(AddressFamily.InterNetwork, _
                                    SocketType.Stream, ProtocolType.Tcp)
27:           SvrHostEntry = Dns.GetHostEntry(TextBox2.Text)
28:           ' IPV4 対応の IP アドレスをアドレスリストから検索
29:           For i = 0 To SvrHostEntry.AddressList.Length - 1
30:               If SvrHostEntry.AddressList(i).AddressFamily = AddressFamily.InterNetwork Then
31:                   SvrAddress = SvrHostEntry.AddressList(i)
32:               End If
33:           Next
34:           SvrEndPoint = New IPEndPoint(SvrAddress, SvrPort)
35:           ClntSocket.Connect(SvrEndPoint)
36:           ClntStream = New NetworkStream(ClntSocket)
37:           ClntReader = New StreamReader(ClntStream, Encoding.UTF8)
38:           ClntWriter = New StreamWriter(ClntStream, Encoding.UTF8)
39:           ClntWriter.NewLine = vbNewLine
40:           ClntWriter.AutoFlush = True
41:           ' 通信用スレッドの起動
42:           CommThread = New Thread(AddressOf ClntCommThread)
43:           CommThread.IsBackground = True
44:           CommThread.Start()
45:       Catch ex As Exception
46:           MessageBox.Show(Me, ex.Message, "接続エラー", _
                                MessageBoxButtons.OK, MessageBoxIcon.Error)
47:       End Try
48:   End Sub
```

プログラム19行目～23行目は，ローカル変数の宣言です。

19行目のSvrHostEntryはサーバーのIPアドレスとホスト名のリストを保存しておく変数

です。

20行目の SvrAddress はサーバーの IP アドレスを，次の SvrPort はポート番号を保存しておく変数です。ポート番号は dynamic/private ポート番号の49152～65535であればどれでもよいのですが，ここでは50000に設定しておきます。

22行目の SvrEndPoint はサーバーの EndPoint を保存しておく変数で，EndPoint は IP アドレスとポート番号を統合したものです。

26行目は通信のためのソケットを作成しています。ここでは，アドレスファミリーは IPv4 のアドレスを，ソケットタイプはストリーム型を，プロトコルタイプは TCP を指定して作成しています。

表 3. 3. 2　主な AddressFamily 列挙型

メンバー名	内容
InterNetwork	Internet Protocol version4のアドレス
InterNetworkV6	Internet Protocol version6のアドレス
AppleTalk	Apple Talk のアドレス
Ipx	IPX/SPX のアドレス

表 3. 3. 3　主な SockeType 列挙型

メンバー名	内容
Stream	ストリーム型
Dgram	データグラム型

表 3. 3. 4　主な ProtocolType 列挙型

メンバー名	内容
Tcp	トランスミッション・コントロール・プロトコル
Udp	ユーザー・データグラム・プロトコル

27行目は，TextBox から入力された IP アドレスまたはホスト名を DNS サーバーに問い合わせ，IPHostEntry を取得しています。

29～33行目は，IPHostEntry のアドレスリストから IPv4 のアドレスを取り出し，それをサーバーの IP アドレスとして保存しています。

34行目は，サーバーの IP アドレスとポート番号から IP エンドポイントを作成しています。

35行目は，サーバーへの接続をしています。サーバーとの接続ができると，次の36行目に移りますが，エラーが発生すると45行目の例外処理に移ります。

36行目は，接続できたソケットを使ってネットワークストリームオブジェクトを作成して

います。

37～38行目は受信用のStreamReaderオブジェクトと，送信用のStreamWriterオブジェクトを作成しています。

39行目は，データを送信する際にWriteLineメソッドを使いますが，WriteLineメソッドを呼び出したときに書き込む行の終端を示す文字列を設定しています。ここでは改行文字（vbNewLine）の定数を設定しています。

40行目は，データを送信した場合，即座にバッファーからデータを送信するように設定しています。データを送信するためにWriteLineメソッドを呼び出すと，システムはバッファーにデータをためてバッファリングを行い，即座にデータが送信されません。そのために，このようにAutoFlushプロパティをTrueに設定することによって，強制的にバッファーからデータを送信するようにしています。

42～44行目は，通信用のスレッドを起動しています。42行目はClntCommThreadというメソッドをスレッドとして作成しています。43行目は作成されたスレッドをバックグラウンドスレッドに設定しています。スレッドにはバックグラウンドスレッドとフォアグラウンドスレッドがありますが，バックグラウンドスレッドはフォアグラウンドスレッドが終了すれば，処理が終了していなくとも同時に終了します。フォアグラウンドスレッドには，そのようなことはありません。ここでは，フォアグラウンドスレッドが終了した場合は，通信用スレッドのみが処理を続けても意味がないので同時に終了するように，バックグラウンドスレッドに設定します。

44行目はスレッドを起動する操作です。

45～47行目はサーバーとの接続時にエラーが発生した場合の処理を行っています。

次は，送信ボタンの処理です。

```
50:   '送信ボタンの処理
51:   Private Sub Button1_Click(sender As System.Object, e As System.EventArgs) _
                                                          Handles Button1.Click
52:       Try
53:           ClntWriter.WriteLine(TextBox1.Text)
54:       Catch ex As Exception
55:           MessageBox.Show(Me, ex.Message, "送信エラー", _
                          MessageBoxButtons.OK, MessageBoxIcon.Error)
56:       End Try
57:   End Sub
```

ここでは，テキストボックスに入力されたメッセージをサーバーに送信しています。

53行目がデータの送信をしているところです。エラーが発生した場合のエラーメッセージの表示が55行目で行われています。

```
60:   'リストボックスにデータを表示
61:   Private Sub Display(ByVal s As String)
62:       ListBox1.Items.Add(s)
63:   End Sub
```

61行目～63行目は受信データをリストボックスに表示するためのプログラムです。このプログラムは，通信用スレッドから直接は呼び出されず，Delegate を通して呼び出されます。

次は通信用スレッドです。

```
65:   '通信用スレッド
66:   Private Sub ClntCommThread()
67:       Dim Buffer As String
68:       Try
69:           Do
70:               Buffer = ClntReader.ReadLine()
71:               If IsNothing(Buffer) Then
72:                   Exit Do
73:               End If
74:               Me.Invoke(New DisplayDlgt(AddressOf Display), Buffer)
75:           Loop
76:           '通信の終了処理を行う
77:           ClntSocket.Shutdown(SocketShutdown.Both)
78:           CloseSocket()
79:       Catch ex As Exception
80:           Me.Invoke(New MsgDlgt(AddressOf MessageBox.Show), _
                      New Object() {Me, ex.Message, "通信スレッドエラー", _
                                    MessageBoxButtons.OK, MessageBoxIcon.Error})
81:       End Try
82:   End Sub
```

このスレッドは「接続ボタン」が押され，サーバーとの接続が完了したときに起動されます（プログラム42～44行目参照）。

70行目でデータの受信を行います。このプログラムでは通常ここで受信待ちになっており，データを受信した時に動き始めます。

71行目は接続が切れていないかどうかをチェックしています。空のデータを受信した場合，接続が切れたことを意味していますので，その時には Do ループから抜け77行目に跳びソ

ケットをシャットダウンし，クローズ処理を行います。

77行目のソケットのShutdownメソッドは，このソケットからの送信を止め，サーバーはNothingを受信することでこのソケットが切断されたことを知ります。78行目のCloseSocketは通信の終了処理を行うメソッドで，93～102行目にあります。

ShutdownメソッドのパラメータのSocketShutdown列挙値は以下の通りです。

表3.3.5　Shutdown列挙型

メンバー名	内容
Send	この後の送信を禁止します。
Receive	この後の受信を禁止します。
Both	この後の送受信の両方を禁止します。

74行目では受信したデータをListBoxに表示するため，Invokeメソッドを使って書き込みを行っています。

80行目は受信で生じたエラーを表示しています。ここでもInvokeメソッドを使ってDelegateを通してエラーを表示しています。

次は切断ボタンをクリックしたときの処理です。

```
83:   ' 切断処理
84:   Private Sub Button3_Click(sender As System.Object, e As System.EventArgs) _
                                                                    Handles Button3.Click
85:       ' 送信を無効にする
86:       ClntSocket.Shutdown(SocketShutdown.Send)
87        ' スレッドの終了待機
88:       If Not CommThread.Join(2000) Then
89:           CloseSocket()
90:       End If
91:   End Sub
92:   ' 通信の終了処理
93:   Private Sub CloseSocket()
94:       ' 送受信を終了する
95:       '  ネットワークストリームとソケットを閉じる
96:       If Not IsNothing(ClntStream) Then
97:           ClntStream.Close()
98:       End If
99:       If Not IsNothing(ClntSocket) Then
100:          ClntSocket.Close()
101:      End If
102:  End Sub
```

86行目ではソケットを Shutdown しています。
88～90行目は，受信スレッドが終了するのを待っています。クライアントが Shutdown したので，それを受け取ったサーバーも Shutdown し，受信スレッドはそれを受信した後，終了するはずです。スレッドが終了するまで２秒間待機し，それでも終了しなかった場合は，ストリームとソケットを強制的にクローズします。
93～102行目が CloseSocket メソッドで，ストリームとソケットのオブジェクトをチェックし，まだクローズされていなければクローズします。

3.4　サーバー・プログラムの解説

```
 1:Imports System.IO
 2:Imports System.Net
 3:Imports System.Net.Sockets
 4:Imports System.Text
 5:Imports System.Threading
 6:Module Module1
 7:    Public ClientList As ArrayList = New ArrayList()    ' 接続クライアントのクラスリスト
 8:    Delegate Sub DisplayDlgt(ByVal s As String)    'リストボックスへの表示用デリゲート
 9:    Delegate Function MsgDlgt(ByVal owner As IWin32Window, ByVal s As String, ByVal cptn As String, _
                                 ByVal btn As MessageBoxButtons, ByVal icn As MessageBoxIcon)
10:End Module
```

１行目～５行目はクライアント・プログラムと同様クラスの名前空間の参照設定を行っています。
６行目～10行目はクラス間で共有する変数を Module で宣言しています。
７行目は接続してきたクライアントに対応した通信用クラスを格納しておく ArrayList 型の変数です。
８行目は ListBox へ書き込むための Delegate で，９行目は MssageBox を表示するための Delegate です。

```
12:    Private Sub Form1_Load(sender As Object, e As System.EventArgs) Handles MeLoad
13:        Dim LThread As Thread
14:        'クライアントとの接続を行うスレッドを起動
15:        LThread = New Thread(AddressOf ListenThread)
16:        LThread.IsBackground = True
17:        LThread.Start()
18:    End Sub
```

12行目から18行目は Form1の Load のイベントプロシージャです。Form1は起動時にクラ

イアントからの接続を待つスレッドをバックグラウンドで起動しています。15行目でスレッドを作り，16行目でバックグラウンドの設定を行い，17行目で起動しています。

次はクライアントとの接続処理を行うスレッドです。

```
20:    ' クライアントとの接続を行うスレッド
21:    Private Sub ListenThread()
22:        Dim ListenSocket As Socket  ' 待機ソケット
23:        Dim ComSocket As Socket     ' 通信ソケット
24:        Dim SvAddress As IPAddress  ' IPアドレス
25:        Dim SvPort As Integer = 50000       ' ポート番号
26:        Dim SvEndPoint As EndPoint
27:        Dim SvComm As ServerCommClass       ' サーバーの送受信を行うクラスでクライアント
毎に作成
28:
29:        Try
30:            ' 待機ソケット作成
31:            ListenSocket = New Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp)
32:            SvAddress = IPAddress.Any       '全てのネットワークインターフェースで待機
33:            SvEndPoint = New IPEndPoint(SvAddress, SvPort)  ' IPアドレスとポート番号から
EndPointを作成
34:            ListenSocket.Bind(SvEndPoint)       ' ソケットとEndPointを関連付ける
35:            ListenSocket.Listen(10)         '待機の準備 接続待ちは最大10
36:            Do
37:                ' クライアントからの接続要求を受付け，新しいソケットを作成する
38:                ComSocket = ListenSocket.Accept
39:                ' 通信用のクラスを作成
40:                SvComm = New ServerCommClass(ComSocket, Me)
41:            Loop
42:        Catch ex As Exception
43:            Me.Invoke(New MsgDlgt(AddressOf MessageBox.Show), New Object() {Me, ex.Message, "接
続スレッドエラー", MessageBoxButtons.OK, MessageBoxIcon.Error})
44:        End Try
45    End Sub
```

20行目～45行目まではクライアントからの接続を待ち，接続できた時にはそのクライアントと通信を行うソケットとクラスを作成します。

31行目でクライアントからの接続を待つ待機ソケットを作成しています。

32行目のIPAddress.Anyはサーバーが全てのネットワークインターフェースでクライアントからの接続を待機することを示すアドレスです。クライアントと異なりサーバーとなるコンピュータは複数のネットワークインターフェースを持っていることが多く，そのためこのような指定をします。

33行目はIPアドレスとポート番号からIPEndPointを作成しています。

34行目の Bind メソッドはソケットにローカルの IP アドレスとポート番号（IPEndPoint）を割り当てています。相手からの接続を待つ場合にはローカルアドレスを設定する必要があり，自分から接続する場合にはローカルアドレスを設定する必要はありません。クライアントは自分から接続するので，Bind をする必要がないのです。クライアントのプログラムでは，通信を行う際にシステムによってアドレスが自動的に割り当てられます。
35行目の Listen メソッドでクライアントからの接続を待つ準備をします。引数の10は受信待ちの数です。つまり，接続処理中に他のクライアントから接続要求があった場合，処理待ちにできるクライアントの数です。Windows では MaxConnections で最大数が設定されており，既定値は10となっています。
38行目の Accept メソッドは受信待ちのキューにある接続要求を取り出し，新しいソケットを作成しています。このメソッドは接続要求がなければ次のステップに進まないので，ここで待機することになります。
40行目では，前の行で接続したクライアント専用の通信クラスを作成しています。
36行目から41行目までのループでは，１つのクライアントからの接続要求を受け付け，そのクライアントとの通信を行う専用のクラスを作成する，ということを繰り返して行います。
43行目は通信時のエラー処理を行っています。

```
46:    Public Sub Display(ByVal s As String)
47:        ListBox1.Items.Add(s)
48:    End Sub
```

46行目～48行目は ListBox にクライアントから送られてきたデータを表示するメソッドです。

50行目からは，サーバーの通信処理を行うクラスです。

```
49: ' Serverの通信処理を行うクラス
50:Public Class ServerCommClass
51:     Dim ClsSocket As Socket
52:     Dim MainForm As Form1
53:     Dim ClsStream As NetworkStream
54:     Dim ClsReader As StreamReader
55:     Dim ClsWriter As StreamWriter
56:     Dim ClientName As String
57:
58:     Public Sub New(ByVal ComSocket As Socket, ByVal MForm As Form1)
59:         Dim Receive As Thread
60:         ClsSocket = ComSocket    ' 通信用ソケットをクラス変数に保存
61:         MainForm = MForm         ' Form1をクラス変数に保存
62:         ' 送受信のための準備
63:         ClsStream = New NetworkStream(ClsSocket)
64:         ClsReader = New StreamReader(ClsStream, Encoding.UTF8)
65:         ClsWriter = New StreamWriter(ClsStream, Encoding.UTF8)
66:         ClsWriter.AutoFlush = True
67:         ClsWriter.NewLine = vbNewLine
68:         ' 自分自身をクライアントリストに追加
69:         SyncLock ClientList.SyncRoot
70:             ClientList.Add(Me)
71:         End SyncLock
72:         ' クライアントのハンドル名をIPアドレスとクライアントの番号から作成
73:         ClientName = String.Format("{0}#{1}", CType(ClsSocket.RemoteEndPoint, IPEndPoint).Address,
ClientList.Count)
74:         ' 受信処理用スレッドを起動
75:         Receive = New Thread(AddressOf ReceiveThread)
76:         Receive.IsBackground = True
77:         Receive.Start()
78:     End Sub
```

58～78行目はコンストラクタ・メソッドで初期化を行っています。

60，61行目はパラメータで渡された通信用のソケットとForm1をクラス変数に保存しています。

63～67行目は送受信のためのStreamReaderとStreamWriterのオブジェクトを作っています。

69～71行目はクライアントのリストに自分自身を登録しています。その際，変数ClientListは複数のスレッドからアクセスされる可能性がありますので，SyncLockを使ってロックをかけてから登録しています。

73行目はクライアントのIPアドレスとクライアントの番号からクライアントのハンドル名

を作成しています。ソケットの中の IP アドレスは，ソケットの中の RemoteEndPoint プロパティを IPEndPoint に変換し，IPEndPoint の Address プロパティから取り出さなければなりません。そのため，CType 関数でソケットの RemoteEndPoint を IPEndPoint に変換しています。

75行目から77行目は受信処理用スレッドを作成し，それをバックグラウンドで起動しています。

次は受信用のスレッドです。

```
80:   ' 受信スレッド
81:   Private Sub ReceiveThread()
82:       Dim ReadData, WriteData As String
83:       Try
84:           Do
85:               ReadData = ClsReader.ReadLine    'データを受信
86:               If IsNothing(ReadData) Then      'データがなければ，クライアントからの切断
87:                   Exit Do
88:               End If
89:               WriteData = ClientName & ">>" & ReadData
90:               SyncLock ClientList.SyncRoot     ' 全クライアントへメッセージを送信
91:                   For Each tmp As ServerCommClass In ClientList
92:                       tmp.Send(WriteData)
93:                   Next
94:               End SyncLock
95:               MainForm.Invoke(New DisplayDlgt(AddressOf MainForm.Display), WriteData)
96:           Loop
97:           ' クライアントの切断処理
98:           CloseClient()
99:       Catch ex As Exception
100:          MainForm.Invoke(New MsgDlgt(AddressOf MessageBox.Show), New Object() {MainForm,
ex.Message, "受信スレッドエラー", MessageBoxButtons.OK, MessageBoxIcon.Error})
101:      End Try
102:  End Sub
```

受信スレッドでは，各クライアントからのメッセージを受信し，サーバーのフォームに表示することと，接続している全クライアントに受信メッセージを送信します。

85行目がデータを受信しているところです。

86～88行目はクライアントがソケットを切断したかどうかを判断しています。ソケットを切断した場合，Null データが送られてきますので，Null データであればクライアントがソケットを切断したと判断し，受信処理を終了します。

89行目はクライアントのハンドル名と受信データから，全クライアントへ送信するデータ

を作成しています。

90～94行目は全クライアントへメッセージを送信しています。90行目はクライアントリストにロックをかけ，データを送信中にクライアントリストが書き換えられないようにしています。

92行目のSendメソッドは，103～106行目にあるデータを送信するだけのメソッドです。

95行目は受信データをサーバーのListBoxに，Invokeメソッドを使って書き込んでいます。

98行目はソケットの切断処理をするためにCloseClientメソッドを呼び出しています。

100行目は通信エラーが発生した場合の処理です。MessageBoxを使ってエラーメッセージを表示します。

次はデータの送信とソケットのクローズ処理です。

```
103:    'データ送信
104:    Public Sub Send(ByVal s As String)
105:        clsWriter.WriteLine(s)
106:    End Sub
107:    Private Sub CloseClient()
108:        'クライアントリストから削除
109         SyncLock ClientList.SyncRoot
110:            ClientList.Remove(Me)
111:        End SyncLock
112:        'ソケットのシャットダウン
113         clsSocket.Shutdown(SocketShutdown.Both)
114         clsSocket.Close()
115:    End Sub
116End Class
```

104～106行目はデータ送信用のメソッドです。引数で渡された文字列を送信するだけのプログラムです。

107～115行目はクローズ処理を行うプログラムです。

109～111行目はクライアントリストからクローズするクライアントを削除しています。モジュール変数で共有されているので，削除する際にロックをかけてから削除しています。

113，114行目は，それぞれソケットのシャットダウンとクローズを行っています。

4．社会的ジレンマゲームの作成

4.1　ゲームの概要

このゲームは学生に社会的ジレンマの構造について理解を促すためのネットワーク・ゲームです。ゲームは５つのグループで行われるもので，各グループは１～５までのステップを入力することができ，入力したステップ数進むことができます。ただし，５を入力することができるのは１グループのみで，同時に２グループ以上が５を入力した場合は，５を入力したグループ全てが０になります。同様に４を入力できるグループは２グループまでで，その他のステップも以下のように入力できるグループ数が制限されており，10回の試行が行われ最終的にどのグループが最も進んだかを競うゲームです。（表4.1，4.1.1グループの構成，4.1.2ゲームのルールは林直保子教授の資料をもとに作成したものです。）

表 4.1　入力ステップ数と進めるグループ数

ステップ	進めるグループ数
5	1
4	2
3	3
2	4
1	5

実験者は図4.1.1のサーバーのプログラムを使い，実験参加者は図4.1.2のクライアントのプログラムを使ってゲームを行います。

4.1.1　グループの構成

４人前後で１グループとし，グループ数は５の倍数とします。５つのグループで１つのクラスを構成し，最大５クラスが同時に実験できるシステムとします。

4.1.2　ゲームのルール

① 各グループは１～５までのステップを入力して進むことができ，それを10回繰り返す。

② 表4.1に示すようにステップによって進めるグループ数が決められている。

③ 41以上進んだチームには豪華賞品が与えられる。

④ ９以下のチームには罰ゲームがある。

⑤ 全てのチームが30を超えるとクリア

⑥ ３回目と６回目の後，各チームの順位が分かり，かつチャットによる話し合いが行われる。

⑦ 毎回自分のチームの進度は確認することができる。

図 4.1.1　サーバーの画面

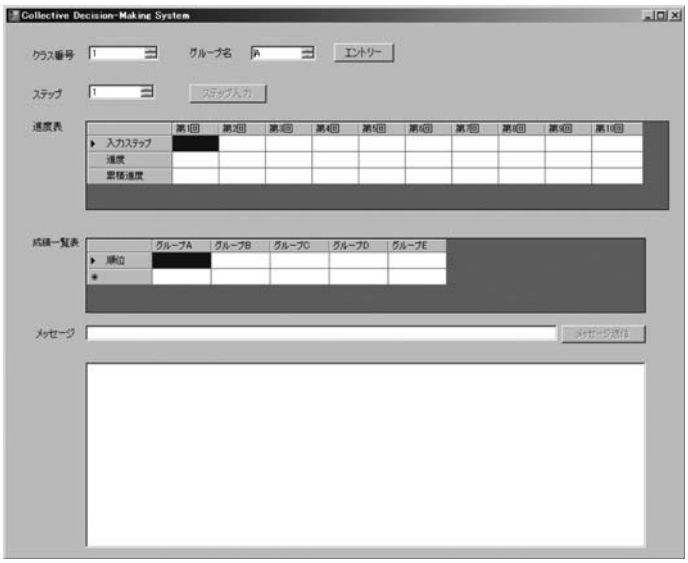

図 4.1.2　クライアントの画面

4.2　システムの使用法

このシステムは，実験者が操作するサーバーと実験参加者が操作するクライアントの２つのプログラムからなっており，サーバーとクライアントはネットワーク接続されている必要があります。実験者が操作するコンピュータにはサーバーのプログラムを，実験参加者が操作するコンピュータにはクライアントのプログラムと一緒に IPPort.txt というファイルを配布します。このファイルにはサーバーの IP アドレスとポート番号が入っています。

実験に入る前に，サーバーの IP アドレスを調べ，このファイルの IP アドレス欄を必要に応じて書き換えてから，実験参加者に配布してください。

4.2.1　IP アドレスの調べ方

サーバーのコンピュータの IP アドレスは，以下のような手順で調べることができます。「スタート」ボタンをクリックし，「すべてのプログラム」をクリックします。そして，「アクセサリ」をクリックし，「コマンドプロンプト」をクリックします。「ipconfig /all」とコマンドを入力し，Enter キーを押してください。そうすると IP の構成について様々な情報が出力されます。その中で，IPv4アドレスと書かれたものが IP アドレスです。図4.2.1の例では192.168.0.2となっています。

図 4.2.1

図 4.2.2

サーバーの IP アドレスが分かれば，それを IPPort.txt に入力します。これは，メモ帳などのテキストエディタを使えば簡単に修正できます。メモ帳でこのファイルを開くと左図のようになっています。1 行目の IPAddress=localhost とありますが，localhost というのはそのコンピュータ自身をさす言葉です。これはサーバーとクライアントを同じコンピュータで走らせてシステムが正しく動作しているかどうか，確認する場合などに使います。通常はこの localhost の代わりに IP アドレスを入力します。図4.2.1の例では192.168.0.2と入力します。IP アドレスの下の PortNo は2.3で説明したポート番号です。ここでは50000を使いますので，これを変更する場合はサーバーのプログラム中のポート番号も変更する必要があります。

4.2.2 実験の手順

実験の大まかな流れは以下のようになります。

① サーバーを起動します。

② クライアントを起動します。サーバーより先にクライアントを起動することはできません。

③ 実験参加者がクラス番号，グループ名を選択し，エントリーボタンをクリックし，エントリーを行います。クライアントからのエントリーを受信すると，サーバーの進捗モニターのクラス別タブの各グループの列の Status 欄に Entry が表示されます。

図 4.2.3 サーバーの進捗モニターの Status 欄

④ ５つのグループがエントリーを完了したクラスがあれば，サーバーは実験者に対し，下図のような通知を表示し実験者に実験開始を促します。実験者は「実験開始」のボタンを押し，実験開始の指示を出します。全クラス一斉に実験を開始しようとする場合は，全クラスのエントリーが終了するまで待ち，全クラスのエントリーが確認できた後，「全クラス実験開始」ボタンを押すと全クラスの実験が開始されます。

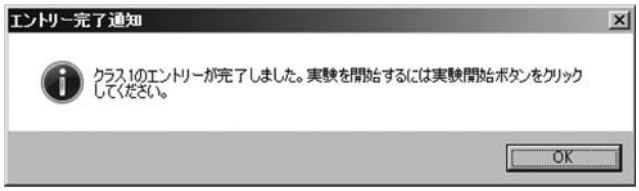

図 4.2.4　サーバーのエントリー完了通知

⑤ 実験開始の指示を受信したクライアントは，実験参加者に下図のようなメッセージを表示し，ステップ数の入力を促します。

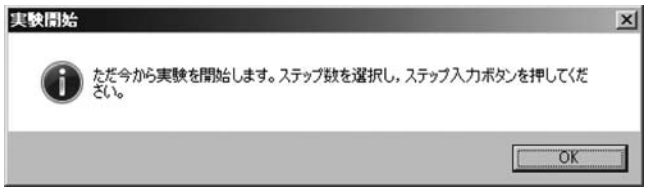

図 4.2.5　クライアントの実験参加者への実験開始通知

⑥ 実験参加者はステップを入力し，ステップ入力ボタンを押すとそのグループの入力値がサーバーに送られます。

⑦ ５つのグループがステップの入力を完了したクラスには，サーバーが各グループの進度を計算し，それをクライアントに伝えます。クライアントはサーバーからの進度を受信すると，実験参加者にその進度を通知します（図4.2.6)。実験者にもステップの入力が完了した通知を行い（図4.2.7)，実験者の進捗モニターに結果が表示されます。クライアントは進度受信確認メッセージの OK ボタンを押さないと，次のステップ入力ができません。詳細なデータはクライアントの進度表（図4.2.8)，サーバーの進捗モニターにそれぞれ表示されます（図4.2.9)。

図 4.2.6　実験参加者への進度通知

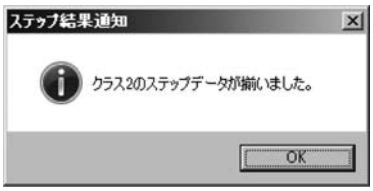

図 4.2.7　サーバーの実験者への通知

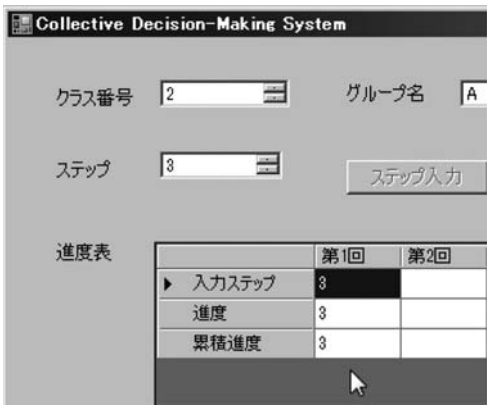

図 4.2.8　クライアントの進度表

図 4.2.9　サーバーの進捗モニター

⑧ ⑥～⑦の処理を繰り返し，第3試行あるいは第6試行を終了した時点でサーバーはそれまでの順位と累積進度をクライアントに通知し，チャットを開始できる旨を実験者に通知します（図4.2.11）。順位，累積進度の詳細は，クライアントは成績一覧表に表示されます（図4.2.12）。

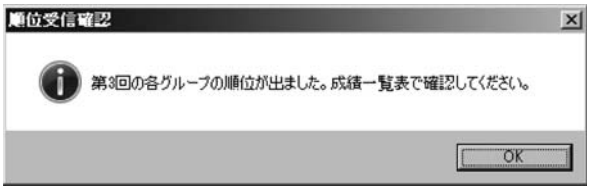

図 4. 2. 10　クライアントから実験参加者への順位の受信メッセージ

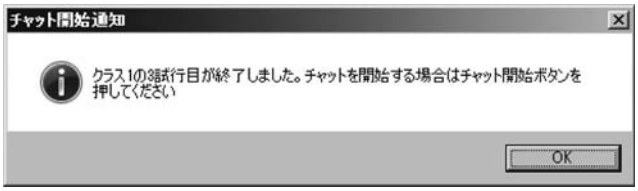

図 4. 2. 11　実験者へのチャットが可能になった通知

図 4. 2. 12　クライアントに表示される各グループの成績一覧表

⑨ 実験者が「チャット開始」のボタンを押し，チャット開始の指示を出すと，そのクラスの全グループにそのことが通知され（図4.2.13），話し合いが始まります。

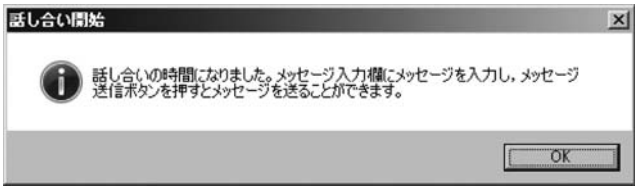

図 4. 2. 13　実験参加者へのチャット開始の通知

⑩ 実験参加者は，メッセージ欄に他グループへのメッセージを入力し，送信ボタン

を押すと，そのメッセージがクラス内の全グループに配信されます。チャットの詳細は，クライアントはメッセージ欄の下のテキストボックスに表示され，サーバーではチャットモニターに表示されます。

⑩ 実験者は話し合いを終了させるために，「チャット終了」のボタンを押し，チャット終了の指示を出すとチャット終了の通知が全クライアントに送信され，クライアントはチャットができなくなりチャットが終了します（図4.2.14）。

図 4.2.14　実験参加者へのチャットの終了通知

⑪ 話し合い終了メッセージのOKボタンを押すと，再びステップ入力ボタンを押すことができるようになります。また，⑥の操作に戻り，第10試行まで行います。

⑫ 実験者が「実験終了」ボタンをクリックすると，そのメッセージがクライアントに通知され，クライアントは自動的に終了します。

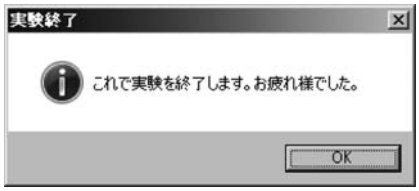

図 4.2.15　クライアントで表示される終了メッセージ

下図は，クライアントがシャットダウンしたときのサーバーのStatus欄に表示される，Downのメッセージです。

図 4. 2. 16　サーバーの Status 欄に表示されたクライアントの終了状態

実験者が終了ボタンを押すとサーバーが終了し，結果ファイルがサーバーと同じフォルダに作成されます。結果ファイルには，進捗モニターに表示されたデータを csv ファイルにしたものと，チャットデータを csv ファイルにしたチャットログファイルがあります。

進捗モニターのデータファイルは，下記のような名前になっています。

Sc-yyyymoddhhmiss.csv

S：進捗モニターのデータファイルを示す文字です。

c：クラス番号　　yyyy：年　　mo：月　　dd：日　hh：時　mi：分　ss：秒

.csv：カンマ区切りのファイルであることを示す拡張子です。

チャットのログファイルは，上記と同様で 1 文字目が C になっています。

4. 3　システムの作成

「3．簡単な通信プログラム」で述べたチャットのプログラムより複雑になりますが，通信の流れに関しては全く同じと考えることができます。ですから，「図3.2.1　クライアントとサーバーの通信手順」をそのまま使うことができます。

4. 3. 1　メッセージ

複雑になるのは，クライアントとサーバー間で交換するメッセージの種類です。チャットのプログラムでは，メッセージが 1 種類しかありませんでしたので，ひじょうに簡単に済んだのですが，この実験のプログラムではクライアントとサーバーで交換するメッセージの種類が多くなります。

このような場合，まずどのようなメッセージの流れがあるか，そのメッセージにはどの

ようなデータが必要かを検討し，メッセージを設計します（表4.3.1，図4.3.1）。

メッセージの一般的な形式を下記のように定めます。

コマンド［Tab］パラメータ1［Tab］パラメータ2［Tab］・・・パラメータn

［Tab］はコマンドとパラメータ，パラメータ同士の区切り文字として使います。区切り文字は，他の用途に使われる文字でなければどれでもよいのですが，チャットのメッセージの中に区切り文字と同じものが使われていると不具合が生じますので，今回の場合可読文字は避けTabにしています。

表4.3.1　メッセージ一覧

コマンド	内容	パラメータ	C-S
ENTRY	エントリー	クラス番号，グループ名	→
EERR	エントリーエラー	クラス番号，グループ名	←
EXSTART	実験開始メッセージ	なし	←
EXEND	実験終了メッセージ	なし	←
STEP	ステップデータ	クラス番号，グループ名，ステップ数	→
PRGRSS	進度	進度	←
ALLDATA	クラス内の順位と進度	A順位，A進度，B順位，B進度，…	←
CHTSTART	チャット開始メッセージ	なし	←
CHTEND	チャット終了メッセージ	なし	←
MSG	チャットメッセージ	n，クラス番号，グループ名，メッセージ	⟷

表の「C-S」はクライアントとサーバー間の送受信の方向を示しています。

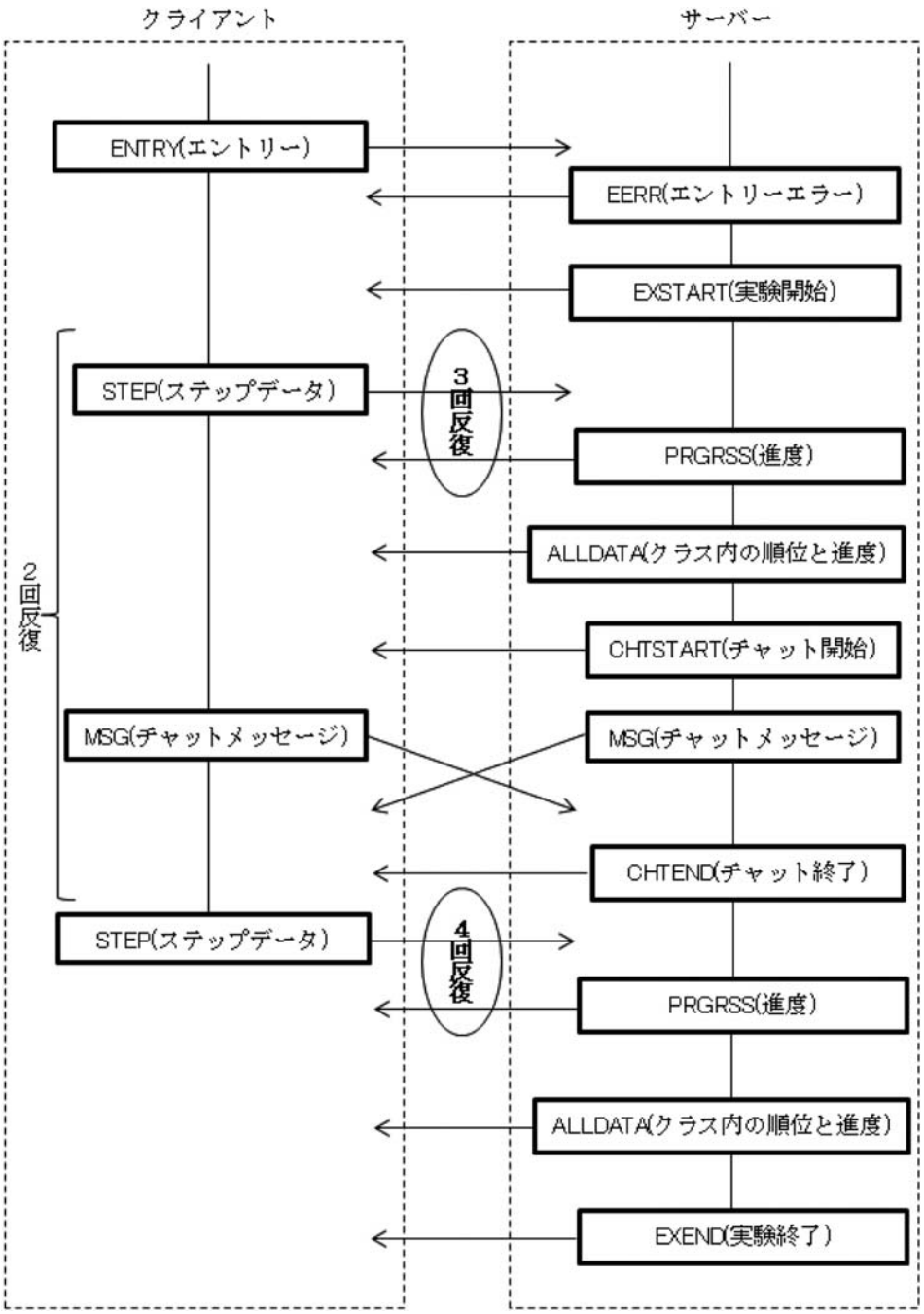

図 4.3.1　メッセージの流れ

4.3.2　メッセージの分解

メッセージを受信すると，まず何のメッセージであるか判断する必要があります。そのためには，区切り文字ごとにこのメッセージのような，何らかの区切り文字で区切られている文字列はSplitメソッドによって1語ずつに分解することができます。

Splitメソッドの構文は，下記のようになっています。

String.Split（Char（））

Stringは分解する文字列であり，引数のChar（）は区切り文字の配列です。結果はString型の配列に返されます。

例　下の例では，変数Separatorに区切り文字としてTabを設定し，変数msgに表4.3.1のENTRYメッセージでクラス1グループAというメッセージ設定し，分解しています。結果は，配列cmdに表4.3.2のように代入されます。

```
Dim Separator As Char = vbTab
Dim msg As String = "ENTRY" & vbTab & "1" & vbTab & "A"
Dim cmd() As String
cmd = msg.Split(Separator)
```

表4.3.2　分解結果

配列の要素	内容
cmd(0)	ENTRY
cmd(1)	1
cmd(2)	A

4.4　ソースプログラムについて

ソースプログラムは長いものでそれを全て掲載して解説することは難しいため，ソースプログラムを載せていません。先に解説したチャットプログラムとネットワーク・ゲームは下記のURLからダウンロードできます。ソースプログラムは，Visual Basic 2008 Express Editionで書かれておりますが，Visual Basic 2010や2012でも動作します。上位のバージョンで動かす場合，変換が必要になる場合がありますが，指示に従って変換すれば利用できます。Visual BasicはVisual Basic 2010 Expressあるいは，Visual Studio Express 2012 for Windows Desktopを Microsoft社のサイトから無料でダウンロードできます。

「IPPort.txt」ファイルは，クライアントのフォルダ「bin」の下の「Debug」と「Release」の両方にあります。利用する前にIPアドレスを修正してから利用してください。ネットワ

ーク・ゲームを利用するだけの場合は，Server と Client それぞれのフォルダ「bin」の下の「Debug」あるいは「Release」にある「Client.exe」と「Server.exe」をコピーして使用してください。特に実験参加者のコンピュータには，あらかじめ「Client.exe」と「IPPort.txt」の２つを配布しておくと容易に実験を始められます。

なお，これらのソフトは通信を行いますので，Windows ファイアウォールでブロックされることがあります。その場合には，ネットワーク管理者と相談して利用してください。

下記 URL の心理学実験のページからダウンロードできます。

URL：http://www2.ipcku.kansai-u.ac.jp/~hisamoto/

［参考文献］

Axelrod, R.（1980）Effective Choice in the Prisoner's Dilemma. *Journal of Conflict Resolution* 24, 3-25

Bavelas, A.（1950）Communication Patterns in Task-Oriented Groups. *The Journal of the Acoustical Society of America 22*, 6, 725-730

廣田君美（1953）集團の課題解決と Communication　*心理学研究　24*，105-113

広瀬幸雄（1997）*シミュレーション世界の社会心理学　ゲームで解く葛藤と共存*　京都　ナカニシヤ出版

柿本・細野（2010）状況の現実感尺度の再検討：２つの仮想世界ゲーム実験から　*実験社会心理学研究　49*, 2, 149-159

Latane, B. & L'Herrou, T.（1996）Spatial Clustering in the Conformity Game: Dynamic Social Impact in Electric Groups. *Journal of Personality & Social Psychology 70*, 6, 1218-1230

Leavit, HJ.（1951）Some effects of certain communication patterns on group performance. *Journal of abnormal and social psychology 46*, 1, 38-50

森尾博昭（2007）ダイナミック社会的インパクト理論の予測する態度の自己組織化の実証：コンピュータ・コミュニケーションを用いた実験的研究　*実験社会心理学研究　47*, 1, 1-12

ソケたん製作委員会（2005）*Visual Basic ではじめるネットワークプログラミング超入門*　東京　毎日コミュニケーションズ

―2013.4.26受稿―