Network Programming Primer

Quiz Chapter 16

Distributed Computing

Use for example in MMOs

Distributed System

"a collection of autonomous hosts that are connected through a computer network. Each host executes components and operates a distribution middleware, which enables the components to coordinate their activities in such a way that users perceive the system as a single, integrated computing facility."

– Emmerich

Requirements of Distributed Systems

- **Transparency**
  - Location: *users don't need to know where resources are*
  - Migration: *processes/recourses may move without users' knowledge*
  - Replication: *users don't need to know how many copies of resources are distributed around the system*
  - Concurrency: *users don't need to know or be affected by the load*
  - Failure: *system hides failure points*

Requirements of Distributed Systems

- **Heterogeneity**
- **Resource Sharing:** *remote objects*
- **Scalability:** *use of load balancing, and leases*
- **Openness:** *easily extended and modified*

Process Interaction: Client/ Server

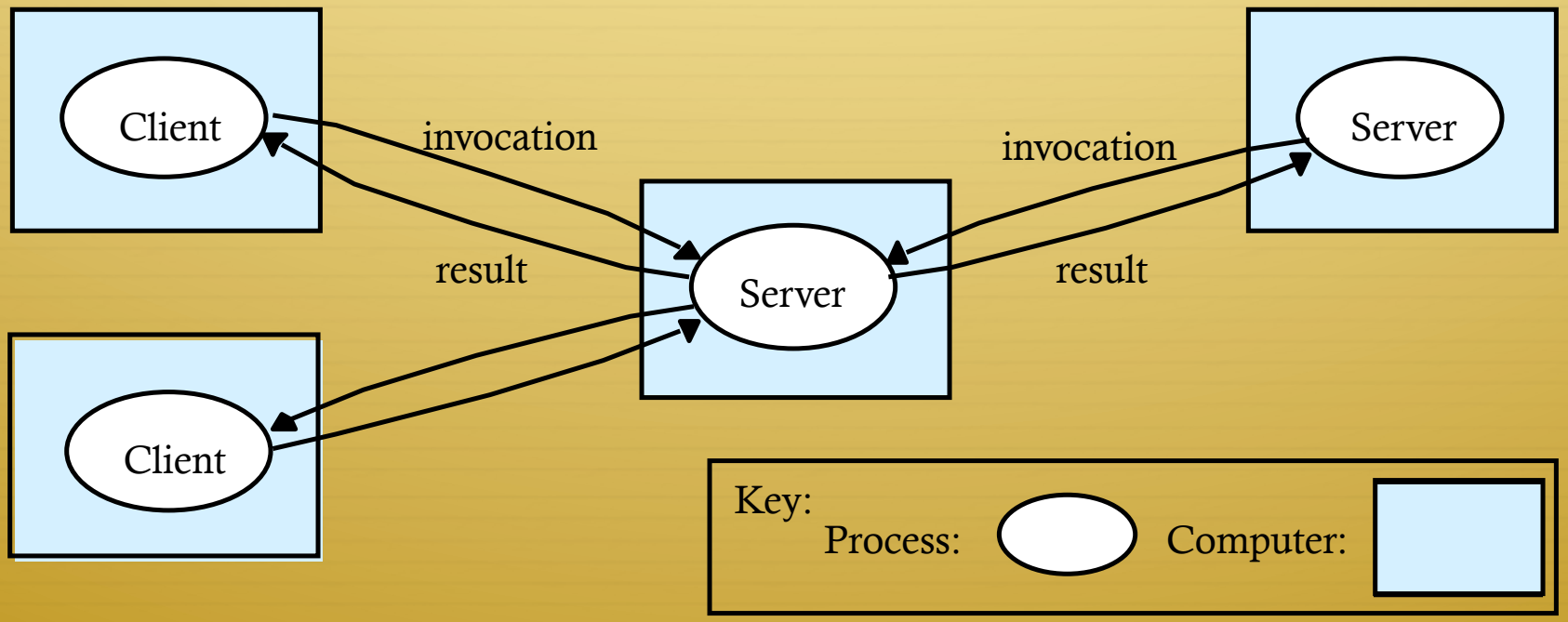
Server: A subsystem that provides a particular type of service to *a priori* unknown clients.

- Control functionally distributed among the various servers in the system
- Control of *individual* resources is centralized in a server
- Problems:

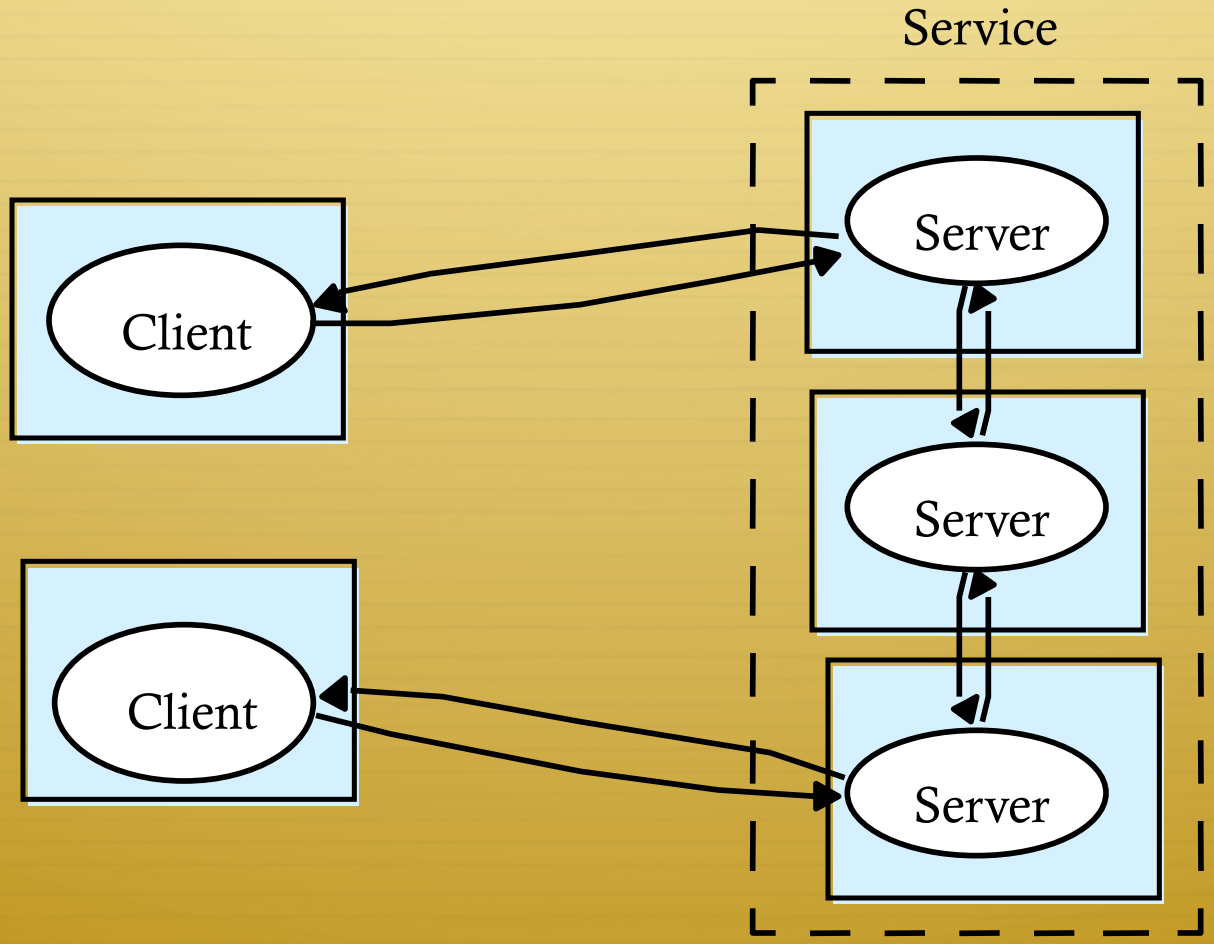
  Reliability/Availability, Scalability

  How do you solve these problems?

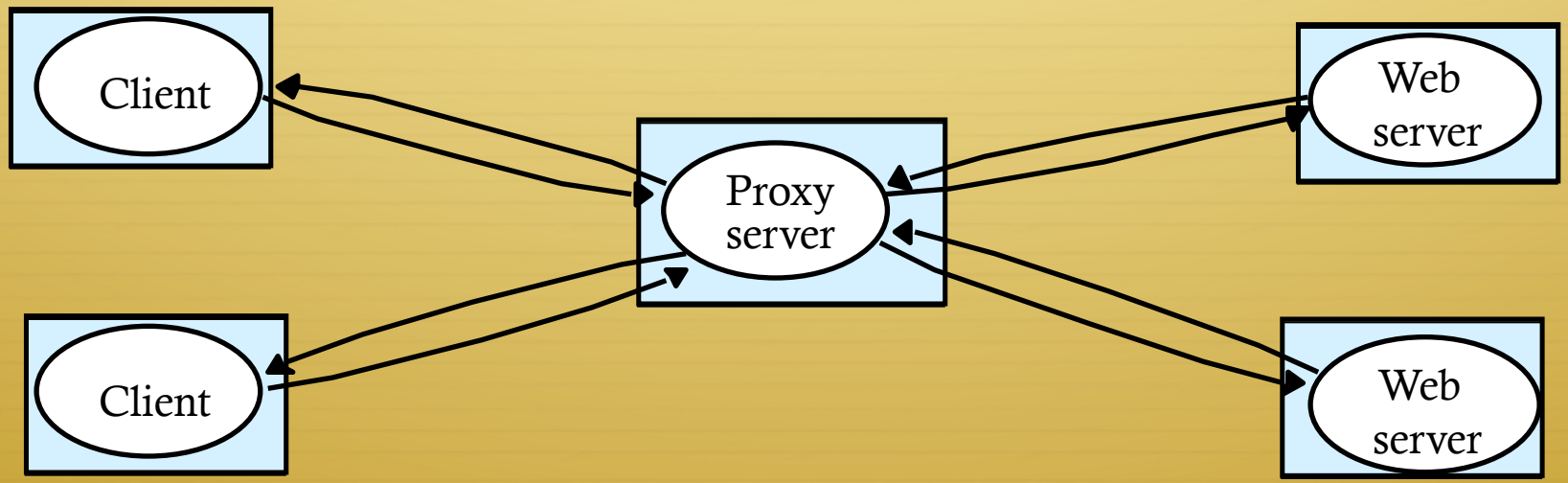
Clients invoke individual servers



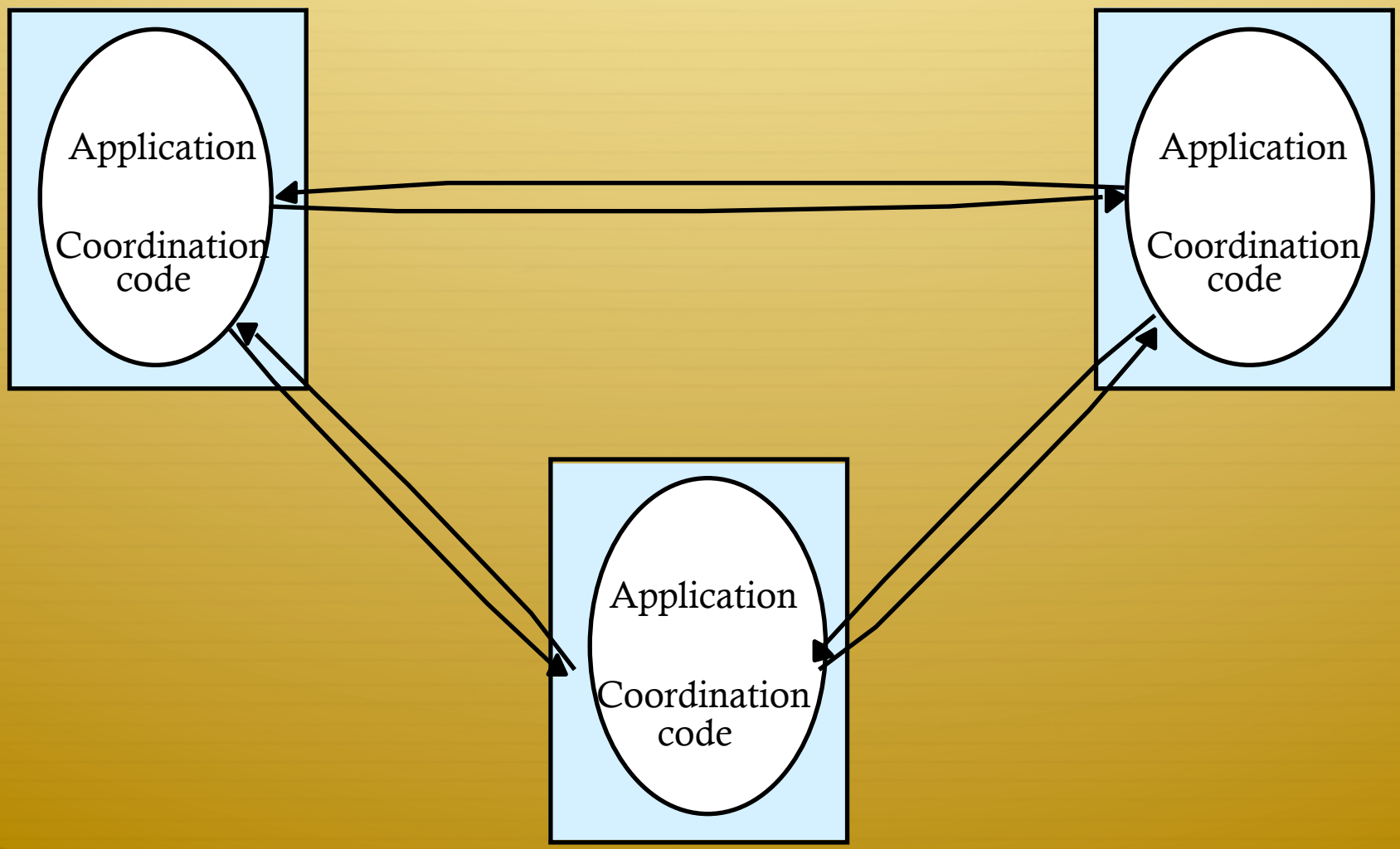
Multiple servers



Proxy server



Peer-to-Peer Communication

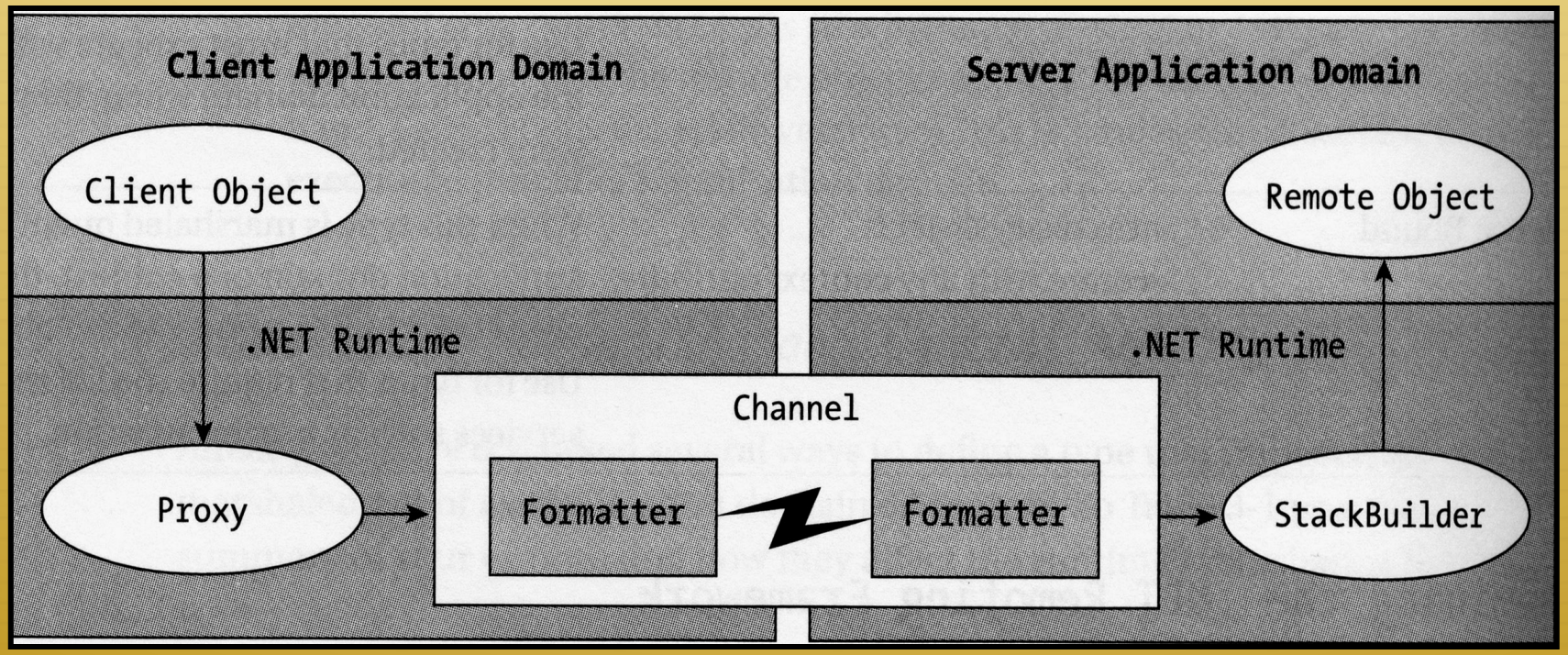


Remote Objects

- What is a remote object:

an object that is not in not local to the current process, i.e. does not exist in current process memory space

.NET Remoting



Remote Objects

Design Requirement:

- Access to methods in remote objects should be ***transparent***

What you really want is:

Call a remote object method:

CalculateScore (playerObject);

as if it is in the local space

All other details should be handled by the middle layer

Object Activation & Lifetime

- **Marshal by Reference**
  - CLR creates a proxy for the object
- **Marshal by Value**
  - A copy of the object is sent to client
  - Client controls the lifetime

Creating an Object

- Library .dll
- Example:

```
namespace DiceGameLibrary
{
    public class DiceGameLib : MarshalByRefObject
    {
    }
}
```

Object Activation & Lifetime

- **Well-Known: activated at the server**
  - Singleton (can hold state)
    - one object for all clients
    - Lifetime is leased based on usage (can be modified)
    - Uses threads, so user needs to worry about synchronization issues
  - SingleCall (better for replication and load balancing applications)
    - One object for each call
- **Client-activated**
  - Object is activated for each client
  - Can hold state for each client

An Example : Server

```
TcpChannel channel = new TcpChannel(13101);
ChannelServices.RegisterChannel (channel);

RemotingConfiguration.RegisterWellKnownServiceType (
                    typeof(DiceGameLibrary.DiceGameLib ),
                    "MyURI.rem",
                    WellKnownObjectMode.Singleton );

Console.WriteLine ("Server Started. Press Enter to end ");
Console.ReadLine ();
```

An Example : Client

```
TcpChannel channel = new TcpChannel();

ChannelServices.RegisterChannel (channel);
```

```
Object remoteObj = Activator.GetObject (
                        typeof (DiceGameLibrary.DiceGameLib ),

                        "tcp://localhost:13101/MyURI.rem");
```

```
myGameStateObject = (DiceGameLib) remoteObj;
```

Channels

- HTTP Channel
  - By default uses SOAP formatter

```
Using System.Runtime.Remoting.Channels;

HttpChannel channel = new HttpChannel (portNumber);

ChannelServices.RegisterChannel (channel);
```

- TCP Channel
  - By default uses binary formatter

```
Using System.Runtime.Remoting.Channels;

TcpChannel channel = new TcpChannel (portNumber);

ChannelServices.RegisterChannel (channel);
```

Remote Objects Example

Server

```
using System;
using System.Runtime .Remoting ;
using System.Runtime .Remoting .Channels.Tcp ;
using System.Runtime .Remoting .Channels;
using DiceGameLibrary;

namespace SimpleGameServer
{
    /// <summary>
    /// Server for a simple Dice Game
    /// </summary>
    class GameServer
    {
        [STAThread]
        static void Main(string[] args)
        {
            TcpChannel channel = new TcpChannel(13101);

            ChannelServices.RegisterChannel (channel);

            RemotingConfiguration.RegisterWellKnownServiceType (
                typeof(DiceGameLibrary.DiceGameLib ),
                "MyURI.rem",
                WellKnownObjectMode.Singleton );

            Console.WriteLine ("Server Started. Press Enter to end ");
            Console.ReadLine ();
        }
    }
}
```

Client

```
public delegate void AddRolledNumber(string number);
public delegate void EnableButton();
public delegate void DisableButton();

public Form1()
{
    //
    // Required for Windows Form Designer support
    //
    InitializeComponent();
    // Here put the channel and object initialization stuff
    TcpChannel channel = new TcpChannel();
    ChannelServices.RegisterChannel (channel);
    Object remoteObj = Activator.GetObject (
                        typeof (DiceGameLibrary.DiceGameLib ),
                        "tcp://localhost:13101/MyURI.rem");
    myGameStateObject = (DiceGameLib) remoteObj;
    MyNumber = myGameStateObject.getPlayerNumber ();
    //Console.WriteLine ("My Number is"+MyNumber);
    button1.Enabled = false;
    Thread TurnThread = new Thread(new ThreadStart(CheckTurn));
    TurnThread.Start();
}

private void PutText(string text)
{
    Rolled.AppendText(text);
}
```

```
private void button1_Click(object sender, System.EventArgs e)
{
    // call Roll
    int Number = myGameStateObject.RollDice(MyNumber);
    if (this.Rolled.InvokeRequired)
    {
        AddRolledNumber d = new AddRolledNumber(PutText);
        this.Invoke(d, new object[] { Number.ToString() });
    }
    else
        Rolled.Text = Number.ToString();
    if (myGameStateObject.returnScore(MyNumber) == 10)
    { //won
        MessageBox.Show("   YOU WIN    ");
    }
    if (this.button1.InvokeRequired)
    {
        EnableButton b = new EnableButton(disablebuttonmethod);
        this.Invoke(b);
    }
    else
        button1.Enabled = false;
}
```

Library

```
using System;
using System.Runtime .Remoting ;

namespace DiceGameLibrary
{
    /// <summary>
    /// Summary description for Class1.
    /// </summary>
    public class DiceGameLib : MarshalByRefObject
    {
        int [] Score;
        int Roll;
        int PlayerCount = 0;
        int Turn = 0;
        public DiceGameLib()
        {
            Score = new int[4];
        }

        public int getPlayerNumber ()
        {
            PlayerCount++;
            Console.WriteLine ("Player Number "+ PlayerCount +" Just joined");
            return PlayerCount-1;
        }
```

```
        public int RollDice(int playerNumber)
        {
            Random R =new Random();
            int MaxLimit = 6;
            int MinLimit = 1;
            Roll = (R.Next(MinLimit, MaxLimit)+R.Next(MinLimit, MaxLimit));
            Console.WriteLine ("Rolled "+ Roll);
            if (Roll == 7)
            {
                Score [playerNumber] = 10;
                for (int i=0; i< 4; i++)
                    if (i != playerNumber)
                        Score [i] = 0;
            }
            Console.WriteLine ("player Number is "+ playerNumber);
            Turn = ((playerNumber+1)%PlayerCount);
            Console.WriteLine ("Turn is "+ Turn);
            return Roll ;
        }

        public int returnScore (int index)
        {
                return Score[index];
        }

        public int returnTurn()
        {
            return Turn;
        }
    }
}
```

Assignment 5

Use your Tetris example and add a multi-player component. The design of this is up to you. You can use a co-operative or competitive design to your multi-player game.

The basic requirements are:

- Implement asynchronous multi-player component (using remote objects or sockets) (4 points)
- Document describing the design of the system and justification for the chosen methods, e.g. remote objects or serialization (2 points)
- Architectural design and style (exception handling) (2 points)
- Document with the new architecture showing the class hierarchy, class interaction, and threading in UML (2 points)