```
//-------------------------------------------------------------------------------
--------------
//File:   BServerCom.cs
//Desc:   This program defines a class BServerCom which handles communication be
tween the game
//          the player client.
//-------------------------------------------------------------------------------
---------------


using Battleship;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Threading.Tasks;

namespace BattleShip_Server
{
    class BServerCom : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;

        private string LOCK = "Lock";

        public const int Port = 6500;
        public TcpListener BServer;
        bool connected = false;

        private string gameNames;                                       //Contain
s the string representation of the game names
        public string StrNames
        {
            get
            {
                return gameNames;
            }
            set
            {
                gameNames += value;
                SetProperty("StrNames");
            }
        }
        Dictionary<string, Game> games = new Dictionary<string, Game>(); //Dicti
onay to hold the game instances
        public Dictionary<string, Game> Games
        {
            get
            {
                return games;
            }
            set
            {
                games = value;
                SetProperty("Games");
            }
        }
        public Dictionary<string, Game>.KeyCollection GameNames;            //ho
lds a collection of the names of the games that have been created.

        private StringBuilder messages = new StringBuilder();               //st
ringbuilder used by the Log Messages
        public string Messages
        {
            get
```



```
            {
                return messages.ToString();
            }
            set
            {
                messages.AppendLine(value);
                SetProperty("Messages");
            }
        }

        //Constructor for the BServer class, initializing the GameNames Property
        public BServerCom()
        {
            BServer = new TcpListener(IPAddress.Any, Port);
            BServer.Start();

            GameNames = new Dictionary<string, Game>.KeyCollection(games);
        }

        //Handles communication between client in its param and the listener.
        public void Commun(TcpClient player)
        {
            string address = player.Client.RemoteEndPoint.ToString();
            LogMsg("Connection request from: " + address);

            connected = true;
            try
            {
                using (NetworkStream conctn = player.GetStream())
                {
                    Game game;

                    StreamReader reader = new StreamReader(conctn);
                    StreamWriter writer = new StreamWriter(conctn);

                    writer.WriteLine("Welcome to Battleship. What game do you wish to join? ");
                    writer.Flush();

                    do
                    {
                        if (player.Available > 0)
                        {
                            string gameName = reader.ReadLine();

                            lock (LOCK)
                            {
                                if (!GameNames.Contains(gameName))
                                {
                                    game = new Game(10);

                                    Games.Add(gameName, game);
                                    game.Players.Add(address);
                                    StrNames = gameName + "\n";
                                }
                                else
                                {
                                    game = Games[gameName];
                                    game.Players.Add(address);
                                }
                            }

                            writer.WriteLine(GameState(game));
                            writer.Flush();

                            while (connected)
                            {
                                string request = reader.ReadLine();
                                string response = null;

                                while (request != null)
```

```
                                {
                                    LogMsg(gameName + " Request: " + request);
                                    try
                                    {
                                        lock (LOCK)
                                        {
                                            var requestQ = Request.Deserialize(r
equest);
                                            var responseQ = requestQ.Execute(gam
e);
                                            response = responseQ.Serialize();
                                        }
                                    }
                                    catch
                                    {
                                        response = null;
                                    }

                                    writer.WriteLine(response);
                                    writer.Flush();
                                    LogMsg("Response:\n" + response);

                                    request = reader.ReadLine();
                                }
                            }
                        }
                    } while(player != null);

                    LogMsg("Player disconnected.");
                };
            }
            catch (Exception ex)
            {
                Debug.WriteLine("Check: " + ex.Message);
            }
        }

        //Retries the state of the game in its params and returns a string repre
sentation of the boards
        public string GameState(Game g)
        {
            string status = "active";
            if (g.IsGameOver == true)
            {
                status = "ended " + g.Winner;
            }
            string states = LogMsg("GameStateResponse " + status + "\n" + g.UpdateSt
ate(g.Human) + "---\n" + g.UpdateState(g.AI) + "\n");
            return states;
        }

        //Loads the message in its params to the Messages property and returns t
he message in its params
        string LogMsg(string msg)
        {
            Messages = msg + "\n";
            return msg;
        }

        protected void SetProperty(string source)
        {
            PropertyChangedEventHandler handle = PropertyChanged;
            if (handle != null)
            {
                PropertyChanged(this, new PropertyChangedEventArgs(source));
            }
        }
    }

    class Player : IDisposable
```



```
    {
        public StreamReader Reader { get; }

        public StreamWriter Writer { get; }

        public TcpClient PlayerClient { get; set; }

        private NetworkStream stream;

        public Player(TcpListener listener)
        {
            PlayerClient = listener.AcceptTcpClient();
            stream = PlayerClient.GetStream();
            Reader = new StreamReader(stream);
            Writer = new StreamWriter(stream);

        }

        public void Dispose()
        {
            PlayerClient.Close();
        }
    }
}
```

```
//-------------------------------------------------------------------------------
--------------
//File:   Game.cs
//Desc:   This program defines a class Game which contains the data for the game
 BattleShip.
//-------------------------------------------------------------------------------
---------------


using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Diagnostics;
using System.ComponentModel;
using System.Media;

namespace Battleship
{
    public class Game : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;

        public List<string> Players { get; set; }

        public OceanGrid Human;
        public OceanGrid AI;
        public int timeLimit;
        private int hShips;
        private int aiShips;

        SoundPlayer soundPlayer;
        static Random limit = new Random();

        public List<Array> Attacked { get; set; }
//List containing the coordinates the AI has attacked
        public bool HAttacked { get; set; }
        public int TimeLim
        {
            get
            {
                return timeLimit;
            }
            set
            {
                timeLimit = value;
                SetProperty("TimeLim");
            }
        }
        public int Size { get; set; }
        public int NumShips { get; set; }
        public int HShips
        {
            get
            {
                return hShips;
            }
            set
            {
                hShips = value;
                SetProperty("HShips");
            }
        }
        public int AIShips
        {
            get
            {
                return aiShips;
            }
```



```
            set
            {
                aiShips = value;
                SetProperty("AIShips");
            }
        }
        public bool IsGameOver { get; set; }
        public string Winner { get; set; }                          //Declared winne
r of the game
        public string Message { get; set; }                         //Message to dis
play when game ends;
        public string EndSound { get; set; }                        //Sound to play
when game ends;

        //Constructor for Game class
        public Game(int size)
        {
            Size = size;
            NumShips = 5;
            Human = new OceanGrid(size);
            AI = new OceanGrid(size);


            Place();
            Attacked = new List<Array>();
            timeLimit = 5;
            HShips = Human.Ships.Count;
            AIShips = AI.Ships.Count;

            Players = new List<string>();
        }

        //Checks in the grid in the params for a ship object with the coordinate
s x and y, and returns true if not found, false if found
        public bool ValidatePosition(OceanGrid grid, int x, int y)
        {
            return !grid.GetShipCoord(x, y);
        }

        //verifies that the numbers picked are valid positions and places ships
on the grids
        public void Place()
        {
            List<OceanGrid> grids = new List<OceanGrid> { Human, AI };

            Random ornt = new Random(1);
            Random len = new Random();

            foreach (OceanGrid grid in grids)
            {
                List<int> lengths = new List<int>();

                int two = 0;

                while (grid.Ships.Count < NumShips)
                {
                    int x = limit.Next(Size);
                    int y = limit.Next(Size);
                    int length = len.Next(5);
                    int orient;


                    bool validPos = false;

                    while (validPos == false)
                    {
                        if (length > 1)
                        {
                            orient = ornt.Next(1, 3);
                            validPos = grid.TestLoc(x, y, length, orient);
```

```
                        if (length == 2 && two != 2)
                        {
                            two++;
                            lengths.Add(length);
                        }
                        else if (!lengths.Contains(length))
                        {
                            lengths.Add(length);
                        }
                        else
                        {
                            validPos = false;
                        }

                        if (validPos == true)
                        {
                            grid.AddShip(x, y, length, orient);
                        }
                    }
                    else if (ValidatePosition(grid, x, y) == true)
                    {
                        grid.AddShip(x, y);
                        validPos = true;
                    }

                    x = limit.Next(Size);
                    y = limit.Next(Size);
                    length = len.Next(5);
                }

                UpdateState(grid);
            }
        }

    }

    //Calls the Attack method (see definition in OceanGrid.cs) in the either
of the grids, and returns a array containing the results of the both attacks
    // and the coordinates the AI attacked.
    public Array[] Attack(int x, int y)
    {
        Array[] results = new Array[3];

        bool[] hCheck = AI.Attack(x, y);

        if (hCheck[0] == true)
        {
            Play("Hit.wav");
        }
        else
        {
            Play("Miss.wav");
        }

        EndGame();

        HAttacked = true;
        AIShips = AI.Ships.Count;

        Array[] AIResults = AIAttack();
        HShips = Human.Ships.Count;

        int[] aiCoords = (int[])AIResults[1];
        bool[] aiCheck = (bool[])AIResults[0];

        results[0] = hCheck;
        results[1] = aiCheck;
        results[2] = aiCoords;
```



```
        EndGame();

        return results;
    }

    //Calls AIAttack when the TimeLimit for the Human to move has ended, ret
urning the results of the attack and the coordinated that were attacked in an ar
ray of arrays
    public Array[] TimedAttack()
    {
        if (TimeLim == 0)
        {
            Array[] AIResults = AIAttack();

            EndGame();
            if (IsGameOver == true)
            {
                return null;
            }

            TimeLim = 5;
            return AIResults;
        }
        else
        {
            TimeLim--;
        }
        return null;
    }

    //Computes a location for the AI to attack and returns the results of th
e attack and the coordinates that were attacked in an array of arrays
    public Array[] AIAttack()
    {
        Array[] AIResults = new Array[2];

        Random aiAttack = new Random();

        int[] aiCoords = new int[2];
        bool[] aiCheck = new bool[2];

        while (aiCheck[1] == false)
        {
            int aiX = aiAttack.Next(Size);
            int aiY = aiAttack.Next(Size);
            aiCoords[0] = aiX;
            aiCoords[1] = aiY;

            aiCheck = Human.Attack(aiX, aiY);
        }

        HShips = Human.Ships.Count;

        AIResults[0] = aiCheck;
        AIResults[1] = aiCoords;

        if (aiCheck[0] == false)
        {
            Attacked.Add(aiCoords);
            Play("Miss.wav");
        }
        else if (aiCheck[0] == true)
        {
            Play("Hit.wav");
        }

        return AIResults;

    }
```

```
        //Plays the sound from the source passed in its parameters.
        public void Play(string sound)
        {
            soundPlayer = new SoundPlayer(sound);
            soundPlayer.Play();
        }

        //Verifies that Grids still contain ships, setting IsGameOver to true if
 one of them doesn't and supplying the appropriate Message and Winner of the gam
e.
        public void EndGame()
        {
            if (Human.Ships.Count == 0)
            {
                EndSound = "Lost.wav";
                IsGameOver = true;
                Message = "TOO BAD. I WON!!!";
                Winner = "Computer";
            }

            if (AI.Ships.Count == 0)
            {
                EndSound = "Win.wav";
                IsGameOver = true;
                Message = "NICE. YOU WON!!!";
                Winner = "Human";
            }
        }

        //Shows the state of the grid in passed into its parameter
        public string UpdateState(OceanGrid ocean)
        {
            Debug.WriteLine("Board:");

            StringBuilder fnlState = new StringBuilder();

            for (int x = 0; x < Size; x++)
            {
                StringBuilder builder = new StringBuilder();
                builder.Append("\r");
                for (int y = 0; y < Size; y++)
                {
                    switch (ocean.BoardLoc[x, y])
                    {
                        case OceanGrid.States.Ship:
                            builder.Append('X');
                            break;
                        case OceanGrid.States.Hit:
                            builder.Append('*');
                            break;
                        case OceanGrid.States.Missed:
                            builder.Append('O');
                            break;
                        default:
                            builder.Append('~');
                            break;
                    }
                    //if (ocean.BoardLoc[x, y] == OceanGrid.States.Ship)
                    //{
                    //    builder.Append('X');
                    //}
                    //else
                    //{
                    //    builder.Append('~');
                    //}
                }

                string state = builder.ToString();
                fnlState.AppendFormat("{0}\r\n", state);
                Debug.WriteLine(state);
```



```
            }
            Debug.WriteLine("\n");

            return fnlState.ToString();

        }

        //Event handler for the ProperyChanged event Notifying the object bound
of the change in the source in its parameters.
        protected void SetProperty(string source)
        {
            PropertyChangedEventHandler handle = PropertyChanged;
            if (handle != null)
            {
                PropertyChanged(this, new PropertyChangedEventArgs(source));
            }
        }
    }
}
```

```
//------------------------------------------------------------------------------
--------------
//File:   MainWindown.cs
//Desc:   This program defines a class MainWindow which contains startup logic f
or the server,
//          while establishing connection between the GUI and its model.
//------------------------------------------------------------------------------
--------------


using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.Windows.Threading;

namespace BattleShip_Server
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        BServerCom server;
        DispatcherTimer Pending;
        public MainWindow()
        {
            InitializeComponent();

            server = new BServerCom();
            DataContext = server;

            Pending = new DispatcherTimer()
            {
                Interval = new TimeSpan(0,0, 0, 0, 200)
            };
        }

        void LogMessages()
        {
            txtRR.Text = server.Messages;
            txtRR.ScrollToEnd();
        }

        //Establishes a connection to the server when a client attempts connecti
on
        void EstConnection()
        {
            try
            {
                if (server.BServer.Pending())
                {
                    var player = new Player(server.BServer);
                    server.Commun(player.PlayerClient);
                }
            }
            catch (Exception)
            {
                throw;
            }
```



```
        }

        //
        private void Window_Loaded(object sender, RoutedEventArgs e)
        {

            txtGame.SetBinding(TextBox.TextProperty, "StrNames");
            txtRR.SetBinding(TextBox.TextProperty, "Messages");
            Task.Run(() =>
            {
                do
                {
                    Task.Run(() => EstConnection());
                } while (true);
            });
        }
    }
}
```

```
//-------------------------------------------------------------------------------
--------------
//File:   Request.cs
//Desc:   This program defines a base class Request and the subclasses used to p
roccess the
//          requests sent by player client.
//-------------------------------------------------------------------------------
---------------

using Battleship;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BattleShip_Server
{
    //Base class for handling all requests after the initial handshake
    public abstract class Request
    {
        public abstract Response Execute(Game g);

        //Contains logic to deserialize the string request, returning a Request
object base on the rqst TYPE;
        public static Request Deserialize(string rqst)
        {
            string[] request = rqst.Split(' ');
            string com = request[0];
            switch(com)
            {
                case DemoRequest.TYPE:
                    return new DemoRequest();
                case AttackRequest.TYPE:
                    AttackRequest attack = new AttackRequest(null, null);
                    return attack.Deserialize(request);
                case GameStateRequest.TYPE:
                    return new GameStateRequest();
                default:
                    throw new Exception("Not a command.");
            }
        }
    }

    //Contains code for handling the demonstration of the server functionality
    public class DemoRequest: Request
    {
        public const string TYPE = "demo";

        //Implement the Execute method, but does not use the game in its params
        public override Response Execute(Game g)
        {
            return new Demo();
        }
    }

    //Contains logic for handling the Attack Requests
    public class AttackRequest: Request
    {
        public const string TYPE = "Attack";

        public string Row { get; set; }
        public string Col { get; set; }
        public string[] Request { get; set; }

        //constructor for the AttackRequest class
        public AttackRequest(string x, string y)
        {
            Row = x;
            Col = y;
```



```
        }

        //initializes an instance of the AttackRequest with the array in its par
ams
        public Request Deserialize(string[] rqst)
        {
            Request = rqst;

            if(Request.Length == 3)
            {
                return new AttackRequest(Request[1], Request[2]);
            }
            return new AttackRequest(null, null);
        }

        //Executes the Attack command on the Game in its params, returning a new
 AttackResponse object
        public override Response Execute(Game g)
        {
            string result = "invalid";
            string vldPlace = "0 1 2 3 4 5 6 7 8 9";

            string[] reqs = new string[] { Row, Col };


            if (vldPlace.Contains(reqs[0]) && vldPlace.Contains(reqs[1]))
            {
                int[] HCoords = new int[2] { Convert.ToInt32(reqs[0]), Convert.T
oInt32(reqs[1]) };
                Array[] atResults = g.Attack(HCoords[0], HCoords[1]);

                if (!g.IsGameOver)
                {
                    bool[] hResults = (bool[])atResults[0];
                    int[] aiCoords = (int[])atResults[2];

                    string hResult = ProccessResults(hResults);

                    return new AttackResponse(hResult, aiCoords);

                }
            }
            return new AttackResponse(result, null);
        }

        //Proccesses the results of the human attack, then returns the result
        string ProccessResults(bool[] hResults)
        {
            string hResult = "invalid";

            if (hResults[0] == true)
            {
                hResult = "hit";
            }
            else if (hResults[0] == false)
            {
                hResult = "missed";
            }
            if (hResults[1] == false)
            {
                hResult = "dup";
            }

            return hResult;
        }

    }

    //Contains logic for handling the GameState requests
    public class GameStateRequest: Request
```

```
    {
        public const string TYPE = "GameState";

        //Executes the GameState command on the Game in its params, then returns
 an instance of the GameStateResponse class
        public override Response Execute(Game g)
        {
            string status = "active";
            if (g.IsGameOver == true)
            {
                status = "ended " + g.Winner;
            }

            return new GameStateResponse(status, g);
        }
    }

}
```



```
//-------------------------------------------------------------------------------
--------------
//File:   Response.cs
//Desc:   This program defines a base class Response and the subclasses used to
formulate a
//          response to send to the player client.
//-------------------------------------------------------------------------------
---------------

using Battleship;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BattleShip_Server
{
    //base class for Response
    public abstract class Response
    {
        public abstract string Serialize();
    }

    //contains the response for the demo of the server
    class Demo: Response
    {
        //Returns a demo of the format of the GameState messages
        public override string Serialize()
        {
            return "GameStateResponse " + "active" + @"
~~~~~X~~~~
~~X~~*~~~~
~O~~~~~~~~
~~~~~~~~~~
~~~~*~~~~~
~X~~~~~~~~
~~~~~~~~~~
~~~~~~~~~~
~~~~~~~~~~
~~~~~~~~~~
---
~~~~~~~~~~
~~~~~~~~~~
~~X~~*~~~~
~O~~~~~~~~
~~~~~X~~~~
~~~~~~~~~~
~~~~*~~~~~
~X~~~~~~~~
~~~~~~~~~~
~~~~~~~~~~
";
        }

    }

    //Contains logic for the Attack Response
    class AttackResponse: Response
    {
        public string HResult { get; set; }
        public int[] AICoords { get; set; }
        public string result;

        //Constructor for the AttackResponse class that recieves the human attac
k results and the coords of the AI
        public AttackResponse(string hResult, int[] coords)
        {
            HResult = hResult;
            AICoords = coords;
```

```
            result = "invalid";
        }

        //Serializes the results of the both the human and AI attacks
        public override string Serialize()
        {
            if (HResult == "invalid" || HResult == "dup")
            {
                result = HResult;
            }
            else
            {
                result = HResult + " " + AICoords[0].ToString() + " " + AICoords[
1].ToString();
            }

            return "AttackResponse " + result;
        }

    }

    //Contains logic for the GameState Response
    class GameStateResponse: Response
    {
        public string Status { get; set; }
        Game Game { get; set; }

        //Contructor for the GameStateResponse class that receives the status(ac
tive, ended) of the game and the game itself in the params
        public GameStateResponse(string status, Game g)
        {
            Game = g;
            Status = status;
        }

        //Serializes the state of the Board returning a string
        public override string Serialize()
        {
            string state = "GameStateResponse " + Status + "\n" + Game.UpdateState(G
ame.Human) + "---\n" + Game.UpdateState(Game.AI) + "\n";
            return state;
        }
    }

}
```