Text-Based RPG Game: TCP Socket Client-Server Implementation

Objective
This project implements a simple multiplayer text-based RPG game using C# and TCP sockets. It allows players to engage in combat, chat, and experience RPG elements such as leveling up and item acquisition. Below are the solutions for each task.

Task 1: Multiple Attack Methods in `/combat` Command

Solution:
To allow more than one attack method, additional cases were added in the `ProcessCombat` function on the server-side code. Each case applies a different attack method with varying damage.

Code:

```
private void ProcessCombat(TcpClient client, string message) {
    Player player = _players[client];
    string[] parts = message.Split(' ');

    if (parts.Length < 3) {
        SendMessage(client, "Usage: /combat [player] [attack_type] to attack. Available attack types: punch, kick, slash, shoot");
        return;
    }

    string targetName = parts[1];
    string attackType = parts[2];
    Player? target = FindPlayerByName(targetName);

    if (target != null)
        StartBattle(player, target, attackType);
    else
        SendMessage(client, $"{targetName}: Absence of keyboard warrior!");
}
```

Task 2: Attacks Visible Only to Involved Players

Solution:
To make attacks visible only to the two players involved, `StartBattle` was updated to send messages only to the `attacker` and `defender`, removing broadcast messaging.

Code:

```
private void StartBattle(Player attacker, Player defender, string attackType) {
    // Calculate damage based on attackType
    int damage = DetermineDamage(attackType);
    defender.Hp -= damage;

    SendMessage(attacker.Client, $"You dealt {damage} damage to {defender.Name}. Their HP is now {defender.Hp}");
    SendMessage(defender.Client, $"You received {damage} damage from {attacker.Name}. Your HP is now {defender.Hp}");
}
```

Task 3: Handling `/exit` Command on Server

Solution:
The server now listens for an `/exit` command. When received, the server removes the client from the list and broadcasts an exit message to remaining players.

Code:

```
case "/exit":
    player.SaveState(); // Save player state
    RemovePlayerFromServer(client, player);
    return;
```

Task 4: `/list` Command to Show All Players

Solution:
The `/list` command was implemented to display connected players and their HP. It's triggered by the client and returns a list of player names with their current HP.

Code:

```
case "/list":
    string list = _players.Aggregate("Player List: ", (current, entry) => current + $"{entry.Value.Name} ({entry.Value.Hp} HP), ");
    SendMessage(client, list);
    return;
```

Task 5: Player Level Up

Solution:
Players gain experience with each battle victory. After reaching 1000 XP, the player levels up, increasing their attack power.

Code:

```
if (defender.Hp <= 0) {
    attacker.Experience += 100;
    if (attacker.Experience >= 1000) {
        attacker.Level++;
        attacker.Experience = 0;
        SendMessage(attacker.Client, $"{attacker.Name} leveled up! New level: {attacker.Level}. Attack power increased.");
        attacker.AttackPower += 5;
    }
}
```

(Optional) Task 6: Item System

Solution:
Implemented an item spawn mechanism, broadcasting item appearance every 30 seconds. Players can pick up items using `/get`.

Code for Item Spawn:

```
private void SpawnItem(object state) {
    string itemName = _itemsList[_random.Next(_itemsList.Count)];
    BroadcastMessage($"A {itemName} has appeared! Type /get to pick it up.");
}
```

(Optional) Task 7: Player State Saving

Solution:
Added a JSON-based save/load feature to persist player states between sessions.

Code:

```
public void SaveState() {
    string json = JsonSerializer.Serialize(this);
    File.WriteAllText($"{Name}.json", json);
}

public static Player? LoadState(string name) {
    if (!File.Exists($"{name}.json")) return null;
    string json = File.ReadAllText($"{name}.json");
    return JsonSerializer.Deserialize<Player>(json);
}
```

Screenshots

Screenshot 1: https://ibb.co/vx7DGXZ

Screenshot 2: https://ibb.co/Ht4j9JN

Code

Client :

```
 using System.Text;
using System.Net.Sockets;

//Connects to a server, exchanges messages between
//   the user and the server, and manages the state of the client.
class RPGClient {
    TcpClient client;
    NetworkStream stream;

    public void Connect(string host, int port)
    //Initiates a connection to the server,
    //   takes input from the user, and sends a message.
    {
        client = new TcpClient(host, port);
        stream = client.GetStream();

        Console.WriteLine("Connected to Battle Net...");
        Console.Write("Enter the name you want to use: ");
        string playerName = Console.ReadLine();

        // Send player name to server
        SendMessage(playerName);

        // Start a thread to receive messages from the server.
        Thread receiveThread = new Thread(ReceiveMessages);
        receiveThread.Start();

        while (true) {
            string message = Console.ReadLine();
            if (message.ToLower() == "/exit") {
                Console.WriteLine("Exit the game!");
                break;
            }
            SendMessage(message);
        }

        Disconnect();
    }

    //Continuously receives messages from the server
    //   and outputs them to the client console.
    private void ReceiveMessages()
    {
        try {
            while (true) {
                byte[] buffer = new byte[256];
                int bytesRead = stream.Read(buffer, 0, buffer.Length);
                // If the server closes the connection
                if (bytesRead == 0) {
                    Console.WriteLine("The connection to the server was lost. You have been removed.");
                    break;
                }
                string message = Encoding.UTF8.GetString(buffer, 0, bytesRead);
                Console.WriteLine(message);
            }
        } catch {
            Console.WriteLine("The connection to the server was lost. You have been removed.");
```

```
        } finally {
            Disconnect();
        }
    }

    //Send a string message to the server
    private void SendMessage(string message)
    {
        byte[] data = Encoding.UTF8.GetBytes(message);
        stream.Write(data, 0, data.Length);
    }

    //Closes the connection to the server and
    //   clears the client's connection status.
    private void Disconnect()
    {
        if (client == null) return;
        client.Close();
        client = null;
        Console.WriteLine("The connection has been terminated.");
    }
}

class Program {
    private static void Main()
    {
        RPGClient client = new RPGClient();
        client.Connect("127.0.0.1", 9000);
    }
}
```

Server :

```
 using System.Net;
using System.Text;
using System.Text.Json;
using System.Net.Sockets;

// Define RPG server class with item spawning and player state-saving functionality
class RPGServer {
    private TcpListener _server = null!;
    private readonly Random _random = new Random();
    private readonly List<TcpClient> _clients = [];
    private readonly Dictionary<TcpClient, Player> _players = new Dictionary<TcpClient, Player>();
    private readonly List<string> _itemsList = ["potion", "sword", "shield"];

    public RPGServer() {
        // Start item spawn timer
        Timer itemSpawnTimer = new Timer(SpawnItem!, null, 0, 30000); // Spawn item every 30 seconds
    }

    //Start server and connect clients
    public void Start() {
        _server = new TcpListener(IPAddress.Any, 9000);
        _server.Start();
        Console.WriteLine("========================");
        Console.WriteLine("=  B a t t l e  N e t  =");
        Console.WriteLine("=  Server Started ...  =");
        Console.WriteLine("========================");

        while (true) {
            TcpClient client = _server.AcceptTcpClient();
            _clients.Add(client);

            // Start a thread to handle new client connections
```

```
            Thread clientThread = new Thread(HandleClient!);
            clientThread.Start(client);
        }
    }

    private void HandleClient(object clientObj) {
        TcpClient client = (TcpClient)clientObj;
        NetworkStream stream = client.GetStream();

        // Receive player name from client
        byte[] buffer = new byte[256];
        int bytesRead = stream.Read(buffer, 0, buffer.Length);
        string playerName = Encoding.UTF8.GetString(buffer, 0, bytesRead).Trim();

        // Load player state if exists, otherwise create a new player
        Player player = Player.LoadState(playerName) ?? new Player(playerName, 100, 10, client);
        _players[client] = player;

        Console.WriteLine($"New Keyboard Warrior Access: {player.Name}");
        BroadcastMessage($"{player.Name} Keyboard Warrior Access!");

        while (true) {
            try {
                bytesRead = stream.Read(buffer, 0, buffer.Length);
                if (bytesRead == 0) break;
                string message = Encoding.UTF8.GetString(buffer, 0, bytesRead);
                Console.WriteLine($"{player.Name}: {message}");
                ProcessMessage(client, message);
            } catch {
                Console.WriteLine($"{player.Name} Warrior Connection Terminate");
                player.SaveState(); // Save player state on disconnect
                _clients.Remove(client);
                _players.Remove(client);
                BroadcastMessage($"{player.Name} Warrior Exit");
                break;
            }
        }
    }

    private void ProcessCombat(TcpClient client, string message) {
        Player player = _players[client];
        string[] parts = message.Split(' ');

        if (parts.Length < 3) {
            SendMessage(client, "Usage: /combat [player] [attack_type] to attack. Available attack types: punch, kick, slash, shoot"
            return;
        }

        string targetName = parts[1];
        string attackType = parts[2];
        Player? target = FindPlayerByName(targetName);

        if (target != null)
            StartBattle(player, target, attackType);
        else
            SendMessage(client, $"{targetName}: Absence of keyboard warrior!");
    }

    private void ProcessMessage(TcpClient client, string message) {
        Player player = _players[client];
        string[] parts = message.Split(' ');

        switch (parts[0]) {
            case "/help":
                SendMessage(client, "Available commands: /help, /combat, /heal, /list, /get, /exit");
                return;
```

```
        case "/combat":
            ProcessCombat(client, message);
            return;
        case "/heal":
            if (player.PickedItem.Type == ItemType.Consumable) {
                player.Hp += player.PickedItem.Value;
                SendMessage(client, $"You used {player.PickedItem.Name}. HP: {player.Hp}");
                DropItem(player);
            } else {
                SendMessage(client, "You don't have a potion.");
            }
            return;
        case "/list":
            string list = _players.Aggregate("Keyboard Warrior List: ", (current, entry) => current + $"{entry.Value.Name} ({ent
            SendMessage(client, list);
            return;
        case "/get":
            player.PickedItem = AttributeItem(player);
            return;
        case "/exit":
            player.SaveState(); // Save player state
            RemovePlayerFromServer(client, player);
            return;
        default:
            SendMessage(client, "Invalid command. Type /help for available commands.");
            return;
    }
}

private void SpawnItem(object state) {
    string itemName = _itemsList[_random.Next(_itemsList.Count)];
    BroadcastMessage($"A {itemName} has appeared! Type /get to pick it up.");
}

private Item AttributeItem(Player player) {
    string itemType = _itemsList[_random.Next(_itemsList.Count)];
    if (player.PickedItem.Type == ItemType.NoItem) {
        SendMessage(player.Client, $"You got {itemType}");
        return itemType switch {
            "potion" => new Item(itemType, ItemType.Consumable, 20),
            "sword" => new Item(itemType, ItemType.Weapon, 5),
            "shield" => new Item(itemType, ItemType.Armor, 10),
            _ => new Item("unknown", ItemType.NoItem, 0)
        };
    }
    SendMessage(player.Client, "You already have an item. Use it first.");
    return player.PickedItem;
}

private void DropItem(Player player) {
    if (player.PickedItem.Type != ItemType.NoItem) {
        SendMessage(player.Client, $"You dropped {player.PickedItem.Name}");
        player.PickedItem = new Item("unknown", ItemType.NoItem, 0);
    } else {
        SendMessage(player.Client, "You don't have an item to drop.");
    }
}

private void StartBattle(Player attacker, Player defender, string attackType)
{
    int damage;
    switch (attackType) {
        case "punch":
            damage = attacker.AttackPower;
            break;
```

```
            case "kick":
                damage = attacker.AttackPower + 5;
                break;
            case "slash":
                damage = attacker.AttackPower + 10;
                break;
            case "shoot":
                damage = attacker.AttackPower + 15;
                break;
            default:
                SendMessage(attacker.Client, "Invalid attack type. Use /combat [player] [attack_type] to attack. Available attack ty
                return;
        }

        if (attacker.PickedItem.Type == ItemType.Weapon) {
            damage += attacker.PickedItem.Value;
            SendMessage(attacker.Client, $"{attacker.Name} uses {attacker.PickedItem.Name}. Damage increased by {attacker.PickedItem
            SendMessage(defender.Client, $"{attacker.Name} uses {attacker.PickedItem.Name}. Damage increased by {attacker.PickedItem
            DropItem(attacker);
        }

        if (defender.PickedItem.Type == ItemType.Armor) {
            damage -= defender.PickedItem.Value;
            if (damage < 0) damage = 0;
            SendMessage(attacker.Client, $"{defender.Name} has a shield. Damage reduced by {defender.PickedItem.Value}");
            SendMessage(defender.Client, $"{defender.Name} has a shield. Damage reduced by {defender.PickedItem.Value}");
            DropItem(defender);
        }

        defender.Hp -= damage;
        SendMessage(attacker.Client, $"{attacker.Name} damages {defender.Name} by {damage}. {defender.Name}'s HP: {defender.Hp}");
        SendMessage(defender.Client, $"{attacker.Name} damages {defender.Name} by {damage}. {defender.Name}'s HP: {defender.Hp}");

        if (defender.Hp > 0) return;
        SendMessage(attacker.Client, $"{defender.Name} Keyboard Warrior Death!");
        SendMessage(defender.Client, $"{defender.Name} Keyboard Warrior Death!");
        attacker.Experience += 100;
        if (attacker.Experience >= 1000) {
            attacker.Level++;
            attacker.Experience = 0;
            SendMessage(attacker.Client, $"{attacker.Name} Level Up! Level: {attacker.Level}");
            attacker.AttackPower += 5;
        }
        TcpClient? defenderClient = GetClientByPlayer(defender);
        if (defenderClient != null) {
            RemovePlayerFromServer(defenderClient, defender);
        }
    }

    // Returns a client (TcpClient) associated with a specific Player object.
     private TcpClient? GetClientByPlayer(Player player)
     {
         return (from entry in _players where entry.Value == player select entry.Key).FirstOrDefault();
     }

    // Find Player objects based on name
     private Player? FindPlayerByName(string name)
     {
         return _players.Values.FirstOrDefault(player => player.Name == name);
     }



    private void RemovePlayerFromServer(TcpClient client, Player player) {
        if (!_clients.Contains(client)) return;
         clients.Remove(client);
```

```
        _clients.Remove(client);
        _players.Remove(client);
        client.Close();
        BroadcastMessage($"{player.Name} Warrior Exit");
    }

    private void BroadcastMessage(string message) {
        byte[] data = Encoding.UTF8.GetBytes(message);
        foreach (TcpClient client in _clients) {
            NetworkStream stream = client.GetStream();
            stream.Write(data, 0, data.Length);
        }
    }

    private static void SendMessage(TcpClient client, string message) {
        byte[] data = Encoding.UTF8.GetBytes(message);
        NetworkStream stream = client.GetStream();
        stream.Write(data, 0, data.Length);
    }
}

[Serializable]
internal class Player(string name, int hp, int attackPower, TcpClient client)
{
    public TcpClient Client { get; set; } = client;
    public string Name { get; set; } = name;
    public int Hp { get; set; } = hp;
    public int AttackPower { get; set; } = attackPower;
    public int Level { get; set; } = 1;
    public int Experience { get; set; } = 0;
    public Item PickedItem { get; set; } = new Item("unknown", ItemType.NoItem, 0);

    // Save player state to JSON
    public void SaveState() {
        string json = JsonSerializer.Serialize(this);
        File.WriteAllText($"{Name}.json", json);
    }

    // Load player state from JSON
    public static Player? LoadState(string name) {
        if (!File.Exists($"{name}.json")) return null;
        string json = File.ReadAllText($"{name}.json");
        return JsonSerializer.Deserialize<Player>(json);
    }
}

internal class Item(string name, ItemType type, int value)
{
    public string Name { get; set; } = name;
    public ItemType Type { get; set; } = type;
    public int Value { get; set; } = value;
}

internal enum ItemType {
    Weapon,
    Armor,
    Consumable,
    NoItem
}

internal static class Program {
    private static void Main() {
        RPGServer server = new RPGServer();
        server.Start();
    }
}
```