DyeWars Implementation Guide

A step-by-step guide for adding new features to the multiplayer game.

Table of Contents

---

1. Adding a New Packet Type

Step 1.1: Choose an Opcode

Check existing opcodes and pick an unused one:

Range	Purpose
`0x01-0x0F`	Client → Server (player actions)
`0x10-0x1F`	Server → Client (single player updates)
`0x20-0x2F`	Server → Client (batch/multi-player updates)
`0x30-0x3F`	Server → Client (effects, events)

Step 1.2: Define the Packet Structure

Document your packet format:

```
Opcode: 0xXX
Name: Your Packet Name
Direction: C → S  OR  S → C
Payload: [field1:bytes][field2:bytes]...

Example:
Opcode: 0x30
Name: Play Effect
Direction: S → C
Payload: [effect_id:1][x:2][y:2]
```

Step 1.3: Server - Send the Packet

In `GameSession.cpp` (or appropriate file):

```cpp
void GameSession::SendYourPacket(/* params */) {
    Packet pkt;
    PacketWriter::WriteU8(pkt.payload, 0xXX);  // Your opcode
    PacketWriter::WriteU16(pkt.payload, someValue);
    // ... add more fields
    pkt.size = pkt.payload.size();
    SendPacket(pkt);
}
```

Step 1.4: Server - Receive the Packet (if C → S)

In `GameSession::HandlePacket()`:

```cpp
case 0xXX:
    if (data.size() >= EXPECTED_SIZE) {
        size_t offset = 1;  // Skip opcode
        uint16_t value = PacketReader::ReadU16(data, offset);
        // ... read more fields

        // Handle the packet
        DoSomething(value);
    }
    break;
```

Step 1.5: Client - Send the Packet (if C → S)

In `NetworkManager.cs`:

```csharp
public void SendYourCommand(/* params */) {
    byte[] message = new byte[] { 0xXX, (byte)param1, (byte)param2 };
    SendMessage(message);

    // Client-side prediction (optional)
    // Update local state immediately
}
```

Step 1.6: Client - Receive the Packet (if S → C)

In `NetworkManager.ProcessPacket()`:

```csharp
case 0xXX:
    if (payload.Length >= EXPECTED_SIZE) {
        int offset = 1;  // Skip opcode
        int value = ReadU16(payload, ref offset);
        // ... read more fields

        lock (queueLock) {
            mainThreadActions.Enqueue(() => {
                // Update state and fire events on main thread
                YourEvent?.Invoke(value);
            });
        }
    }
    break;
```

---

2. Adding Server State

Step 2.1: Add to Player Class (if per-player)

Player.h:

```cpp
```

```
class Player {
public:
    // Getter
    int GetHealth() const { return health_; }

    // Setter or action
    void TakeDamage(int amount);

private:
    int health_ = 100;
};
```

Player.cpp:

```
cpp

void Player::TakeDamage(int amount) {
    health_ = std::max(0, health_ - amount);
}
```

Step 2.2: Update PlayerData Struct (if broadcasted)

Common.h:

```
cpp

struct PlayerData {
    uint32_t player_id;
    int x;
    int y;
    uint8_t facing;
    int health;  // NEW
};
```

Step 2.3: Update GetPlayerData (in GameSession)

```
cpp
```

```
PlayerData GetPlayerData() const {
    return {
        player_->GetID(),
        player_->GetX(),
        player_->GetY(),
        player_->GetFacing(),
        player_->GetHealth()  // NEW
    };
}
```

---

3. Adding Client State

Step 3.1: Add Properties to NetworkManager

```
csharp

// For local player
public int MyHealth { get; private set; } = 100;

// For other players
public Dictionary<uint, int> OtherPlayerHealth { get; private set; } = new();
```

Step 3.2: Add Events

```
csharp

public System.Action<int> OnMyHealthUpdated;
public System.Action<uint, int> OnOtherPlayerHealthUpdated;
```

Step 3.3: Update in Packet Handler

```
csharp

lock (queueLock) {
    mainThreadActions.Enqueue(() => {
        MyHealth = newHealth;
        OnMyHealthUpdated?.Invoke(MyHealth);
    });
}
```

---

4. Adding Unity Events

Step 4.1: Subscribe in Start()

In your renderer or UI script:

```csharp
void Start() {
    networkManager = FindFirstObjectByType<NetworkManager>();
    if (networkManager != null) {
        networkManager.OnMyHealthUpdated += OnMyHealthUpdated;
        // ... other subscriptions
    }
}
```

Step 4.2: Implement the Handler

```csharp
private void OnMyHealthUpdated(int health) {
    // Update UI, sprites, etc.
    healthBar.fillAmount = health / 100f;
}
```

Step 4.3: Unsubscribe in OnDestroy()

```csharp
void OnDestroy() {
    if (networkManager != null) {
        networkManager.OnMyHealthUpdated -= OnMyHealthUpdated;
        // ... other unsubscriptions
    }
}
```

IMPORTANT: Always unsubscribe! Forgetting this causes memory leaks and errors.

---

5. Adding Visual Rendering

Step 5.1: Add Serialized Fields

```csharp
[SerializeField] private Sprite[] healthSprites;  // Or UI elements
[SerializeField] private GameObject healthBarPrefab;
```

Step 5.2: Create Visuals for Other Players

```csharp
private Dictionary<uint, GameObject> healthBars = new();

private void OnOtherPlayerHealthUpdated(uint playerId, int health) {
    if (!healthBars.ContainsKey(playerId)) {
        // Create health bar for this player
        healthBars[playerId] = Instantiate(healthBarPrefab);
    }

    // Update health bar
    healthBars[playerId].GetComponent<HealthBar>().SetHealth(health);
}
```

Step 5.3: Clean Up When Player Leaves

```csharp
private void OnPlayerLeft(uint playerId) {
    if (healthBars.ContainsKey(playerId)) {
        Destroy(healthBars[playerId]);
        healthBars.Remove(playerId);
    }
}
```

---

6. Complete Example: Adding Health System

Here's a full walkthrough of adding a health system.

6.1: Define Packets

```
Opcode: 0x16
Name: Health Update (own player)
Direction: S → C
Payload: [health:2]

Opcode: 0x17
Name: Player Health Update (other players)
Direction: S → C
Payload: [player_id:4][health:2]

Opcode: 0x05
```

Name: Attack
Direction: C → S
Payload: [target_id:4]

6.2: Server - Player.h

cpp

```cpp
#pragma once
#include <cstdint>
#include "GameMap.h"

class Player {
public:
    Player(uint32_t id, int start_x, int start_y);

    bool AttemptMove(uint8_t direction, const GameMap& map);
    void SetFacing(uint8_t direction);
    void TakeDamage(int amount);
    void Heal(int amount);

    uint32_t GetID() const { return id_; }
    int GetX() const { return x_; }
    int GetY() const { return y_; }
    uint8_t GetFacing() const { return facing_; }
    int GetHealth() const { return health_; }
    bool IsAlive() const { return health_ > 0; }

private:
    uint32_t id_;
    int x_;
    int y_;
    uint8_t facing_ = 2;
    int health_ = 100;
    static const int MAX_HEALTH = 100;
};
```

6.3: Server - Player.cpp

cpp

```
void Player::TakeDamage(int amount) {
    health_ = std::max(0, health_ - amount);
}

void Player::Heal(int amount) {
    health_ = std::min(MAX_HEALTH, health_ + amount);
}
```

6.4: Server - GameSession.cpp (Send Functions)

cpp

```
void GameSession::SendHealthUpdate() {
    Packet pkt;
    PacketWriter::WriteU8(pkt.payload, 0x16);
    PacketWriter::WriteU16(pkt.payload, player_->GetHealth());
    pkt.size = pkt.payload.size();
    SendPacket(pkt);
}

void GameSession::SendPlayerHealthUpdate(uint32_t id, int health) {
    Packet pkt;
    PacketWriter::WriteU8(pkt.payload, 0x17);
    PacketWriter::WriteU32(pkt.payload, id);
    PacketWriter::WriteU16(pkt.payload, health);
    pkt.size = pkt.payload.size();
    SendPacket(pkt);
}
```

6.5: Server - GameSession.cpp (Handle Attack)

cpp

```
case 0x05: // Attack
    if (data.size() >= 5) {
        size_t offset = 1;
        uint32_t targetId = PacketReader::ReadU32(data, offset);

        // Tell GameServer to handle the attack
        server_->HandleAttack(player_->GetID(), targetId);
    }
    break;
```

6.6: Server - GameServer.cpp

cpp

```cpp
void GameServer::HandleAttack(uint32_t attackerId, uint32_t targetId) {
    std::lock_guard<std::mutex> lock(sessions_mutex_);

    auto attackerIt = sessions_.find(attackerId);
    auto targetIt = sessions_.find(targetId);

    if (attackerIt == sessions_.end() || targetIt == sessions_.end()) return;

    auto& attacker = attackerIt->second;
    auto& target = targetIt->second;

    // Check if in range (adjacent tiles)
    PlayerData attackerData = attacker->GetPlayerData();
    PlayerData targetData = target->GetPlayerData();

    int dx = std::abs(attackerData.x - targetData.x);
    int dy = std::abs(attackerData.y - targetData.y);

    if (dx + dy <= 1) {  // Adjacent
        target->GetPlayer()->TakeDamage(10);

        // Send health update to target
        target->SendHealthUpdate();

        // Broadcast to all players
        BroadcastToAll([targetId, targetData](auto session) {
            session->SendPlayerHealthUpdate(targetId, targetData.health);
        });
    }
}
```

6.7: Client - NetworkManager.cs (State & Events)

csharp

```
// Properties
public int MyHealth { get; private set; } = 100;
public Dictionary<uint, int> OtherPlayerHealth { get; private set; } = new();

// Events
public System.Action<int> OnMyHealthUpdated;
public System.Action<uint, int> OnOtherPlayerHealthUpdated;
```

6.8: Client - NetworkManager.cs (Send Attack)

csharp

```
public void SendAttack(uint targetId) {
    byte[] message = new byte[5];
    message[0] = 0x05;
    message[1] = (byte)(targetId >> 24);
    message[2] = (byte)(targetId >> 16);
    message[3] = (byte)(targetId >> 8);
    message[4] = (byte)targetId;
    SendMessage(message);
}
```

6.9: Client - NetworkManager.cs (ProcessPacket)

csharp

```
case 0x16: // My health update
    if (payload.Length >= 3) {
        int offset = 1;
        int health = ReadU16(payload, ref offset);

        lock (queueLock) {
            mainThreadActions.Enqueue(() => {
                MyHealth = health;
                OnMyHealthUpdated?.Invoke(MyHealth);
            });
        }
    }
    break;

case 0x17: // Other player health update
    if (payload.Length >= 7) {
        int offset = 1;
        uint playerId = ReadU32(payload, ref offset);
        int health = ReadU16(payload, ref offset);

        lock (queueLock) {
            mainThreadActions.Enqueue(() => {
                OtherPlayerHealth[playerId] = health;
                OnOtherPlayerHealthUpdated?.Invoke(playerId, health);
            });
        }
    }
    break;
```

6.10: Client - HealthUI.cs (New File)

csharp

```
using UnityEngine;
using UnityEngine.UI;

public class HealthUI : MonoBehaviour
{
    [SerializeField] private Image healthBar;
    private NetworkManager networkManager;

    void Start() {
        networkManager = FindFirstObjectByType<NetworkManager>();
        if (networkManager != null) {
            networkManager.OnMyHealthUpdated += OnMyHealthUpdated;
        }

        // Set initial health
        OnMyHealthUpdated(networkManager?.MyHealth ?? 100);
    }

    private void OnMyHealthUpdated(int health) {
        healthBar.fillAmount = health / 100f;
    }

    void OnDestroy() {
        if (networkManager != null) {
            networkManager.OnMyHealthUpdated -= OnMyHealthUpdated;
        }
    }
}
```

---

7. Debugging Checklist

When something doesn't work, check in this order:

Server Side

- [ ] Is the opcode correct and unique?
- [ ] Is `HandlePacket` case added for incoming packets?
- [ ] Is the send function being called?
- [ ] Are packet helpers reading/writing correct byte sizes?
- [ ] Is the payload size check correct? (count all bytes)

Client Side

- [ ] Is the opcode case added in `ProcessPacket`?

- [ ] Is the payload length check correct?
- [ ] Is the offset being used correctly with `ref`?
- [ ] Is the action being queued to main thread?
- [ ] Is the event being invoked?

Unity Events

- [ ] Is the event declared in NetworkManager?
- [ ] Is there a subscription in `Start()`?
- [ ] Is the subscription to the correct event? (check for typos/duplicates)
- [ ] Is there an unsubscription in `OnDestroy()`?
- [ ] Is the handler method signature correct?

Quick Debug Pattern

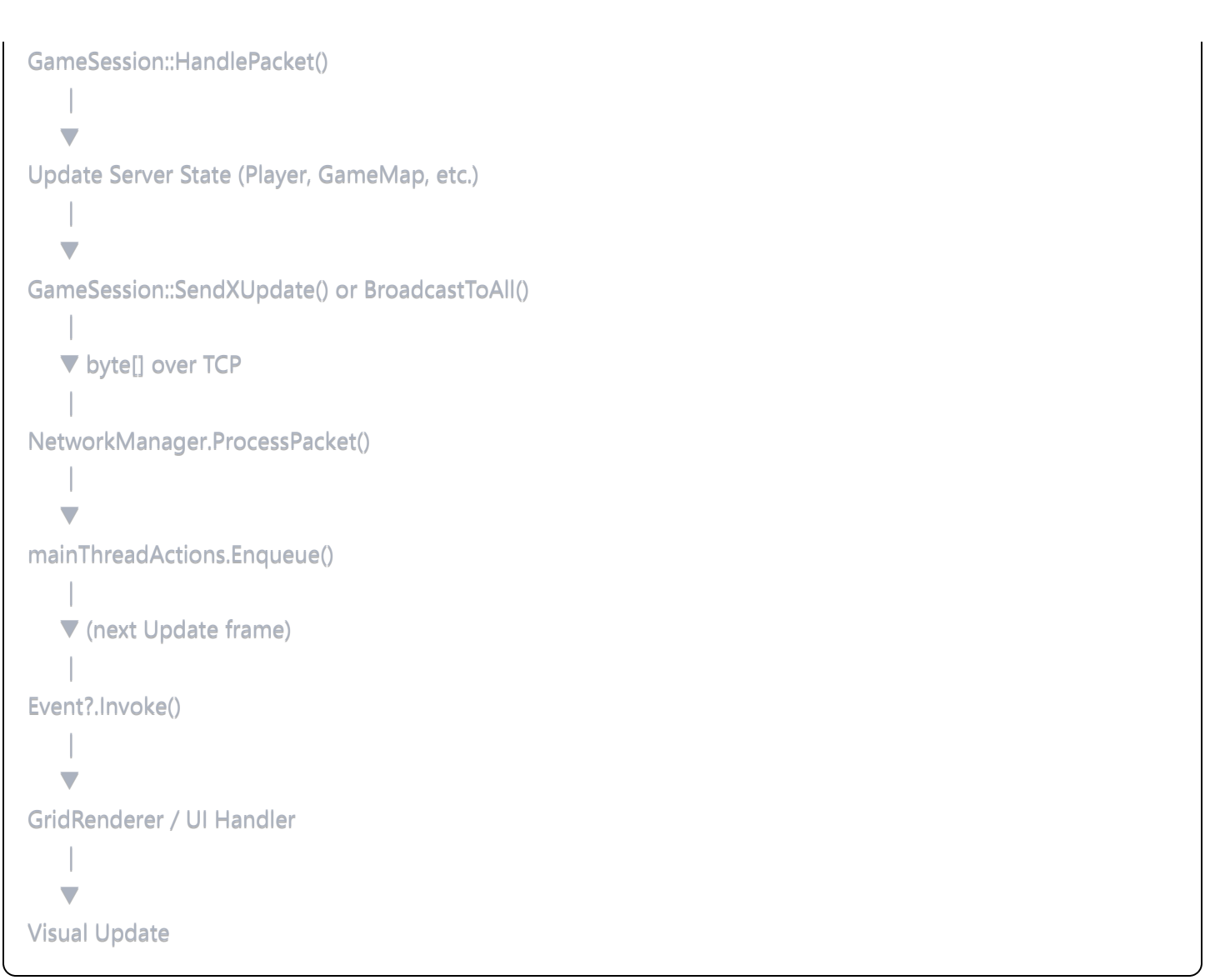
```csharp
// In NetworkManager - check if event has subscribers
Debug.Log($"Subscribers: {YourEvent?.GetInvocationList()?.Length ?? 0}");

// In handler - check if it's being called
private void OnYourEvent(int value) {
    Debug.Log($"OnYourEvent called with: {value}");
}
```

Quick Reference: Byte Sizes

Type	Bytes	C++ Write	C++ Read	C# Read
`uint8_t` / `byte`	1	`WriteU8`	`ReadU8`	`ReadU8`
`uint16_t` / `ushort`	2	`WriteU16`	`ReadU16`	`ReadU16`
`uint32_t` / `uint`	4	`WriteU32`	`ReadU32`	`ReadU32`

Quick Reference: Packet Flow


GameSession::HandlePacket()
Update Server State (Player, GameMap, etc.)
GameSession::SendXUpdate() or BroadcastToAll()
byte[] over TCP
NetworkManager.ProcessPacket()
mainThreadActions.Enqueue()
(next Update frame)
Event?.Invoke()
GridRenderer / UI Handler
Visual Update