# SYSTEM - Project Design Document & Reference

## Table of Contents



## Project Overview

SYSTEM is a Unity-based multiplayer game that uses SpacetimeDB as its backend. Players mine "wave packets" (formerly "quanta") from orbs using crystals that resonate with specific frequencies.

### Core Concepts

- **Wave Packets**: Energy units with specific frequency signatures
- **Crystals**: Mining tools that extract wave packets matching their frequency
- **Orbs**: Containers of wave packets that spawn in the world
- **Frequency System**: 6-color system based on radians (R, RG, G, GB, B, BR)

## Key Architecture Components

## Server-Side (Rust/SpacetimeDB)

```
SYSTEM-server/src/lib.rs
├── Tables
│   ├── Player
│   ├── WavePacketOrb
│   ├── WavePacketStorage
│   ├── PlayerCrystal
│   ├── World
│   └── WorldCircuit
├── Reducers
│   ├── Mining Operations (start_mining, stop_mining, capture_wave_packet)
│   ├── Player Management (create_player, update_position)
│   └── World Operations (tick, emit_wave_packet_orb)
└── Types
    ├── WavePacketSignature
    ├── CrystalType
    └── MiningSession
```

## Client-Side (Unity/C#)

```
SYSTEM-client-3d/Assets/Scripts/
├── Core Management
│   ├── GameManager.cs (connection, scene management)
│   ├── GameData.cs (singleton, persistent data)
│   └── WorldManager.cs (world loading, player spawning)
├── Mining System
│   ├── WavePacketMiningSystem.cs (main mining logic)
│   ├── MiningController.cs (component-based mining)
│   └── WavePacketTransport.cs (visual effects)
├── Player Systems
│   ├── PlayerController.cs (movement, input)
│   └── PlayerSubscriptionController.cs (player tracking)
└── Auto-Generated
    └── SpacetimeDBClient.g.cs (tables, reducers, types)
```

## SpacetimeDB Patterns & Lessons Learned

### 1. Event Handler Pattern

**CORRECT Pattern:**

csharp

```csharp
// Event handlers receive ReducerEventContext + direct arguments
private void HandleStartMining(ReducerEventContext ctx, ulong orbId)
{
    // No .Status or .Message properties
    // Check database state for validation
}

// Table event handlers receive EventContext
private void OnOrbInsert(EventContext ctx, WavePacketOrb orb)
{
    // Handle new orb
}
```

**INCORRECT Pattern (doesn't exist):**

csharp

```csharp
// This pattern does NOT exist in SpacetimeDB
private void HandleStartMining(ReducerEvent<Reducer.StartMining> evt)
{
    if (evt.Status == ReducerStatus.Success) // ❌ No Status property
}
```

## 2. Table Iteration Pattern

**CORRECT:**

csharp

```csharp
// Tables don't support LINQ directly
foreach (var player in conn.Db.Player.Iter())
{
    if (player.Identity == conn.Identity)
        return player;
}
```

**INCORRECT:**

csharp

```csharp
// Tables don't have Where() method
var player = conn.Db.Player.Where(p => p.Identity == identity); // ❌
```

## 3. Subscription Pattern

csharp

```csharp
conn.SubscriptionBuilder()
    .OnApplied(() => HandleSubscriptionApplied())
    .OnError((ctx, error) => HandleError(ctx, error))
    .Subscribe("SELECT * FROM *");
```

## 4. Singleton Pattern

- GameData uses Unity singleton pattern with DontDestroyOnLoad
- GameManager also persists across scenes
- Always check Instance != null before use

### 5. Type Conversions

- Server uses `ulong` for IDs, client sometimes needs `uint`
- Cast appropriately: `uint id = (uint)player.PlayerId;`

## Design Decisions

### 1. Wave Packet Terminology

- Renamed from "quanta" to "wave packets" for better thematic consistency
- Maintains individual packet tracking with unique IDs
- Server-authoritative packet lifecycle

### 2. Color/Frequency System

```
Red (R):     0 radians      = 0.0 normalized
Yellow (RG): π/3 radians    = 1/6 normalized
Green (G):   2π/3 radians   = 1/3 normalized
Cyan (GB):   π radians      = 1/2 normalized
Blue (B):    4π/3 radians   = 2/3 normalized
Magenta (BR): 5π/3 radians  = 5/6 normalized
```

### 3. Mining Mechanics

- Toggle-based mining (not hold)
- 30 unit maximum range
- 2 seconds per packet extraction
- 5 units/second packet travel speed
- Server validates all captures

### 4. Component Architecture

- Modular design with separate components for inventory, targeting, transport
- Event-driven communication between systems
- Visual effects separated from logic

## API Reference

### Key GameManager Methods

csharp

```
// Connection - STATIC PROPERTIES
GameManager.IsConnected() // Check connection status
GameManager.Conn // Static property - Access DbConnection
GameManager.LocalIdentity // Static property - Get local player identity

// Events
GameManager.OnConnected // Subscribe to connection events
GameManager.OnDisconnected // Subscribe to disconnection
```

### GameData Access

csharp

```
GameData.Instance.Username // Get username
GameData.Instance.IsLoggedIn // Check login status
GameData.Instance.GetCurrentWorldCoords() // Get world position
GameData.Instance.SetCurrentWorldCoords(coords) // Update world
```

### Mining System

```csharp
// Start/stop mining
miningController.ToggleMining()

// Check if can mine orb
miningController.CanMineOrb(orb)

// Events
OnMiningStateChanged // Mining started/stopped
OnWavePacketCaptured // Packet successfully captured
```

## Common Pitfalls & Solutions

### 1. Creating Duplicate Classes

**Problem:** Creating new class definitions without checking existing code **Solution:** Always search project knowledge before creating new classes

### 2. Using Wrong Event Pattern

**Problem:** Trying to use ReducerEvent<> wrapper that doesn't exist **Solution:** Use ReducerEventContext with direct arguments

### 3. Forgetting Singleton Pattern

**Problem:** Creating new instances instead of using Instance **Solution:** Always use GameData.Instance, GameManager.Instance

### 4. LINQ on SpacetimeDB Tables

**Problem:** Trying to use Where(), Select() on tables **Solution:** Use Iter() and manual iteration

### 5. Property Access Issues

**Problem:** Direct property access when only getters exist **Solution:** Add appropriate setter methods or use existing ones

## Preventing Compile Errors

### Critical Rules to Follow

1. **ALWAYS Check Method Signatures Before Calling**
   - Search for the method in the codebase before writing code that calls it
   - Check SpacetimeDBClient.g.cs for auto-generated reducer signatures
   - Never assume parameter counts or types
2. **Verify SpacetimeDB Types**
   - SpacetimeDB may use custom types (Vector3f, Position, etc.)
   - Check the Types namespace in auto-generated code
   - Don't assume Unity types are used directly
3. **Check Static vs Instance Members**
   - GameManager.Conn is a STATIC property, not instance
   - Always verify if accessing via instance or class name
4. **Look for Existing Implementations**
   - Before implementing a method, search if it already exists
   - Check for alternative method names (UpdateData vs UpdateFromNetwork)
   - Review how other code calls the same methods

5. **Read Compile Errors Carefully**
    - "No overload takes X arguments" = check actual parameter count
    - "Cannot access with instance reference" = use static access
    - "Does not contain definition" = method doesn't exist or wrong name

### Pre-Code Checklist

Before writing any code that interacts with existing systems:

- [ ] Search for the method/property definition
- [ ] Verify parameter types and count
- [ ] Check if static or instance member
- [ ] Look for usage examples in codebase
- [ ] Confirm types match (especially for SpacetimeDB)

## Future Considerations

### Wave Physics Integration

- Phase coherence affecting extraction rates
- Interference patterns from multiple miners
- Standing wave visualization
- Resonance quality mechanics

### Storage System Enhancements

- Frequency band organization
- Crafting system (2:1 combinations)
- Visual spectrum analyzer

### Crystal Progression

- Multiple crystal slots for paid players
- Crystal upgrades/enhancements
- Special crystals for Shell 2+ colors

### Performance Optimizations

- Object pooling for wave packets (already implemented)
- LOD system for distant orbs
- Batch processing for multiple captures

### Network Optimizations

- Predictive packet movement
- Graceful disconnection handling
- State reconciliation

## Questions for Future Development

1. **Shell 2+ Colors**: How will combinations beyond the base 6 work?
2. **Crafting Mechanics**: What device/interface for combining packets?
3. **World Navigation**: How do players move between worlds?
4. **Competitive Elements**: PvP mechanics for mining competition?
5. **Progression System**: How do players advance beyond crystals?
6. **Economic System**: What drives packet value/trading?

## Code Quality Guidelines

1. **Always Check Existing Code**: Search project before creating new components
2. **Follow Established Patterns**: Use the patterns documented here
3. **Null Checks**: Always verify objects exist before use
4. **Event Cleanup**: Unsubscribe in OnDisable/OnDestroy
5. **Consistent Naming**: Follow existing conventions (Handle*, On*, etc.)
6. **Component Independence**: Keep systems modular and event-driven
7. **Verify Before Implementation**: Check method signatures, types, and patterns

---

**Last Updated**: Current session **Game Name**: SYSTEM **Primary Systems**: Wave Packet Mining, Crystal Selection, World Management **Backend**: SpacetimeDB with Rust reducers **Frontend**: Unity 2022.3 with C# client