Quanta Processing System - Code Cleanup & Implementation Plan

1. Naming Convention Updates

Server-Side Renaming

Replace all instances of "simple_energy" with "quanta":

```rust
// File rename: simple_energy.rs → quanta_system.rs

// Table renames:
simple_energy_signature → quanta_signature
simple_energy_orb → quanta_orb
simple_energy_storage → quanta_storage

// Struct renames:
SimpleEnergySignature → QuantaSignature
SimpleEnergyOrb → QuantaOrb
SimpleEnergyStorage → QuantaStorage

// Reducer renames:
emit_simple_energy_orb → emit_quanta_orb
collect_simple_energy_orb → collect_quanta_orb
transfer_simple_energy → transfer_quanta
debug_simple_energy_status → debug_quanta_status
```

Unity-Side Renaming

csharp

```
// Folder: Assets/Scripts/SimpleEnergy/ → Assets/Scripts/QuantaSystem/

// Script renames:
SimpleEnergyTypes.cs → QuantaTypes.cs
SimpleEnergyVisualizer.cs → QuantaVisualizer.cs
SimpleEnergyOrbController.cs → QuantaOrbController.cs
SimpleEnergyInventoryUI.cs → QuantaInventoryUI.cs
SimpleEnergyManager.cs → QuantaManager.cs
```

2. Core System Architecture

Quanta Signature Structure

rust

```rust
#[derive(SpacetimeType, Debug, Clone, Copy, PartialEq)]
pub struct QuantaSignature {
    pub frequency: f32,     // 0.0-1.0 (maps to color spectrum)
    pub resonance: f32,     // 0.0-1.0 (stability/purity)
    pub flux_pattern: u16,  // Bit pattern for unique variations
}

impl QuantaSignature {
    pub fn calculate_hash(&self) -> u32 {
        let freq_bits = (self.frequency * 1000.0) as u32;
        let res_bits = (self.resonance * 100.0) as u32;
        (freq_bits << 16) | (res_bits << 8) | (self.flux_pattern as u32 & 0xFF)
    }

    pub fn get_frequency_band(&self) -> FrequencyBand {
        match self.frequency {
            f if f < 0.15 => FrequencyBand::Infrared,
            f if f < 0.3 => FrequencyBand::Red,
            f if f < 0.4 => FrequencyBand::Orange,
            f if f < 0.5 => FrequencyBand::Yellow,
            f if f < 0.65 => FrequencyBand::Green,
            f if f < 0.8 => FrequencyBand::Blue,
            f if f < 0.95 => FrequencyBand::Violet,
            _ => FrequencyBand::Ultraviolet,
        }
    }
}
```

Frequency Band System

rust

```
#[derive(SpacetimeType, Debug, Clone, Copy, PartialEq, Eq, Hash)]
pub enum FrequencyBand {
    Infrared,    // Deep red
    Red,         // Red spectrum
    Orange,      // Orange spectrum
    Yellow,      // Yellow spectrum
    Green,       // Green spectrum
    Blue,        // Blue spectrum
    Violet,      // Violet spectrum
    Ultraviolet, // Beyond violet
}
```

3. Database Schema Updates

Quanta Tables

rust

```
#[spacetimedb::table(name = quanta_orb, public)]
pub struct QuantaOrb {
    #[primary_key]
    #[auto_inc]
    pub orb_id: u64,
    pub world_coords: WorldCoords,
    pub position: DbVector3,
    pub velocity: DbVector3,
    pub signature: QuantaSignature,
    pub quanta_amount: u32,     // Amount of quanta in this orb
    pub creation_time: u64,
    pub lifetime_ms: u32,       // How long before despawn (default 30000)
}

#[spacetimedb::table(name = quanta_storage, public)]
pub struct QuantaStorage {
    #[primary_key]
    #[auto_inc]
    pub storage_id: u64,
    pub owner_type: String,     // "player", "device", "world_circuit"
    pub owner_id: u64,
    pub frequency_band: FrequencyBand,
    pub total_quanta: u32,      // Total amount stored
    pub signature_samples: Vec<QuantaSample>, // Detailed breakdown
    pub last_update: u64,
}

#[derive(SpacetimeType, Debug, Clone)]
pub struct QuantaSample {
    pub signature: QuantaSignature,
    pub amount: u32,
    pub source_shell: u8,       // Which shell it came from
}
```

4. Core Reducers

Emission System

rust

```rust
#[spacetimedb::reducer]
pub fn emit_quanta_orb(
    ctx: &ReducerContext,
    world_coords: WorldCoords,
    circuit_position: DbVector3,
) -> Result<(), String> {
    let world = ctx.db.world()
        .world_coords()
        .find(&world_coords)
        .ok_or("World not found")?;

    let circuit = ctx.db.world_circuit()
        .world_coords()
        .find(&world_coords)
        .ok_or("Circuit not found")?;

    // Generate signature based on shell level and circuit
    let seed = ctx.timestamp.as_millis() as u64 ^ circuit.circuit_id;
    let mut rng = StdRng::seed_from_u64(seed);

    let signature = QuantaSignature {
        frequency: generate_frequency_for_shell(world.shell_level, &mut rng),
        resonance: 0.5 + (rng.gen::<f32>() * 0.5), // 0.5-1.0 range
        flux_pattern: rng.gen::<u16>(),
    };

    // Volcano-style emission
    let angle = rng.gen::<f32>() * 2.0 * PI;
    let h_speed = 15.0 + rng.gen::<f32>() * 10.0;
    let v_speed = 20.0 + rng.gen::<f32>() * 15.0;
    let orb = QuantaOrb {
        orb_id: 0,
        world_coords,
        position: circuit_position,
        velocity: DbVector3::new(
            angle.cos() * h_speed,
            v_speed,
            angle.sin() * h_speed,
        ),
        signature,
        quanta_amount: 10 + (circuit.qubit_count as u32 * 5), // More qubits = more quanta
        creation_time: ctx.timestamp.as_millis() as u64,
        lifetime_ms: 30000,
    };

    ctx.db.quanta_orb().insert(orb);
    Ok(())
}
```

Collection System

rust

```
#[spacetimedb::reducer]
pub fn collect_quanta_orb(
    ctx: &ReducerContext,
    orb_id: u64,
    player_id: u64,
) -> Result<(), String> {
    let orb = ctx.db.quanta_orb()
        .orb_id()
        .find(&orb_id)
        .ok_or("Orb not found")?;

    let player = ctx.db.player()
        .player_id()
        .find(&player_id)
        .ok_or("Player not found")?;

    // Verify player identity
    if player.identity != ctx.sender {
        return Err("Not your player".to_string());
    }

    // Add to player's storage
    add_quanta_to_storage(
        ctx,
        "player".to_string(),
        player_id,
        orb.signature,
        orb.quanta_amount,
    )?;

    // Remove orb
```

```
        ctx.db.quanta_orb().delete(orb);

        log::info!(
            "Player {} collected {} quanta of frequency {}",
            player.username,
            orb.quanta_amount,
            orb.signature.frequency
        );

        Ok(())
    }
```

5. Unity Integration Points

QuantaManager.cs

csharp

```
public class QuantaManager : MonoBehaviour
{
    public static QuantaManager Instance { get; private set; }

    [Header("Prefabs")]
    public GameObject quantaOrbPrefab;

    [Header("World Settings")]
    public Transform worldCenter;
    public float worldRadius = 300f;

    private Dictionary<ulong, GameObject> activeOrbs = new Dictionary<ulong, GameObject>();
    private Dictionary<FrequencyBand, QuantaPool> quantaPools = new Dictionary<FrequencyBand, QuantaPool

    void Awake()
    {
        if (Instance == null)
        {
            Instance = this;
            InitializeFrequencyPools();
        }
        else
        {
            Destroy(gameObject);
        }
    }

    void Start()
    {
        // Subscribe to quanta events
        GameManager.Instance.Conn.Db.QuantaOrb.OnInsert += OnQuantaOrbSpawned;
```

```
            GameManager.Instance.Conn.Db.QuantaOrb.OnDelete += OnQuantaOrbCollected;
            GameManager.Instance.Conn.Db.QuantaStorage.OnUpdate += OnStorageUpdated;
        }

        private void InitializeFrequencyPools()
        {
            foreach (FrequencyBand band in Enum.GetValues(typeof(FrequencyBand)))
            {
                quantaPools[band] = new QuantaPool(band);
            }
        }
    }
```

QuantaVisualizer.cs

csharp

```
public class QuantaVisualizer : MonoBehaviour
{
    [Header("Visual Settings")]
    public Gradient frequencyGradient;
    public AnimationCurve resonancePulse;
    public ParticleSystem coreParticles;
    public Light quantaLight;

    private QuantaSignature signature;
    private Material orbMaterial;
    private float pulseTime;

    public void SetSignature(QuantaSignature sig)
    {
        signature = sig;
        UpdateVisuals();
    }

    void UpdateVisuals()
    {
        // Map frequency to color
        Color baseColor = frequencyGradient.Evaluate(signature.frequency);

        // Apply resonance as intensity
        float intensity = 1f + (signature.resonance * 2f);
        Color emissiveColor = baseColor * intensity;

        // Update materials
        orbMaterial.SetColor("_BaseColor", baseColor);
        orbMaterial.SetColor("_EmissionColor", emissiveColor);
        // Update light
        quantaLight.color = baseColor;
        quantaLight.intensity = intensity;

        // Configure particles based on flux pattern
        var main = coreParticles.main;
        main.startColor = baseColor;
        main.startSpeed = 2f + (signature.resonance * 3f);
    }
}
```

6. Implementation Steps

Phase 1: Server Cleanup (Day 1)

1. Rename all files and update imports
2. Update table names and structures
3. Test compilation and basic functionality
4. Run migration to preserve existing data

Phase 2: Unity Cleanup (Day 1-2)

1. Rename folders and scripts
2. Update all references in prefabs
3. Regenerate SpacetimeDB bindings
4. Test scene loading and basic connectivity

Phase 3: Core Functionality (Day 2-3)

1. Implement frequency-based pooling system

2. Create quanta visualization system
3. Test orb spawning and collection
4. Verify storage aggregation works

Phase 4: UI Implementation (Day 3-4)

1. Create frequency band inventory UI
2. Implement quanta transfer interface
3. Add debug visualization tools
4. Polish visual feedback

Phase 5: Testing & Polish (Day 4-5)

1. Full gameplay loop testing
2. Performance optimization
3. Visual effect tuning
4. Bug fixes and edge cases

7. Testing Checklist

- □ Server compiles with renamed modules
- □ Unity receives quanta orb events
- □ Orbs spawn with correct visuals
- □ Collection updates storage properly
- □ Frequency bands aggregate correctly
- □ UI displays quanta by frequency
- □ Transfer between players works
- □ Performance is acceptable with 100+ orbs

- [ ] Visual effects match frequency/resonance
- [ ] No memory leaks or orphaned objects

8. Future Considerations

Once this baseline is working, we can add:

- Quantum mixing mechanics
- Resonance tuning puzzles
- Frequency-based crafting
- Quanta decay over time
- Shell-specific frequency ranges
- Advanced visualization shaders