TECHNICAL_ARCHITECTURE.md

Version: 1.0.0 **Last Updated:** 2024-12-19 **Status:** Approved **Dependencies:** [GAMEPLAY_SYSTEMS.md, SDK_PATTERNS_REFERENCE.md]

Change Log

- v1.0.0 (2024-12-19): Consolidated from system_design and technical sections

---

3.1 System Architecture Overview

Technology Stack

Backend (SpacetimeDB + Rust)

```
SYSTEM-server/
├── src/
│   ├── lib.rs          # Main module entry
│   ├── tables/         # Database schema
│   ├── reducers/       # Game logic
│   └── types/          # Shared types
└── Cargo.toml
```

Frontend (Unity + C#)

```
SYSTEM-client-3d/
├── Assets/
│   ├── Scripts/
│   │   ├── Core/       # GameManager, GameData
│   │   ├── Mining/     # Mining systems
│   │   ├── Player/     # Player controllers
│   │   ├── UI/         # Quantum minigame UI
│   │   └── autogen/    # SpacetimeDB generated
│   └── Prefabs/
└── ProjectSettings/
```

Architecture Patterns

Client-Server Model

- **Server Authoritative**: All validation server-side
- **Client Prediction**: Visual feedback immediate
- **State Reconciliation**: Server corrections applied
- **Event-Driven**: Reducers trigger client updates

Singleton Management

```csharp
GameManager.Instance    // Connection, scene management
GameData.Instance       // Persistent data storage
WorldManager.Instance   // World state management
```

Connection Flow

1. Unity client connects to SpacetimeDB
2. Identity assigned by server
3. Initial table sync
4. Subscribe to relevant tables
5. Reducer events flow bidirectionally

3.2 Database Schema (SpacetimeDB)

Core Tables

Player System

```rust
```

```rust
#[spacetimedb(table)]
pub struct Player {
    #[primarykey]
    pub player_id: u64,
    #[unique]
    pub identity: Identity,
    pub name: String,
    pub position: Vec3,
    pub rotation: Vec3,
    pub current_world: WorldCoords,
    pub last_update: u64,
}

#[spacetimedb(table)]
pub struct Account {
    #[primarykey]
    pub account_id: u64,
    #[unique]
    pub username: String,
    pub display_name: String,
    pub pin_hash: String,
    pub created_at: u64,
}

#[spacetimedb(table)]
pub struct PlayerSession {
    #[primarykey]
    pub session_id: u64,
    pub account_id: u64,
    pub identity: Identity,
    pub session_token: String,
```

```
    pub expires_at: u64,
    pub is_active: bool,
}
```

World System

rust

```rust
#[spacetimedb(table)]
pub struct World {
    #[primarykey]
    pub world_id: u64,
    pub world_coords: WorldCoords,
    pub world_name: String,
    pub world_type: WorldType,  // Genesis, Cardinal
    pub shell_level: u8,
}

#[spacetimedb(table)]
pub struct WorldCircuit {
    #[primarykey]
    pub circuit_id: u64,
    pub world_id: u64,
    pub direction: CardinalDirection,
    pub total_charge: f32,
    pub activation_threshold: f32,
    pub last_rotation: Timestamp,
}

#[spacetimedb(table)]
pub struct CircuitDailyState {
    #[primarykey]
    pub state_id: u64,
    pub circuit_id: u64,
    pub target_state: BlochState,
    pub rotation_seed: u64,
    pub valid_from: Timestamp,
```

```
    pub valid_until: Timestamp,
}
```

Mining System

rust

```rust
#[spacetimedb(table)]
pub struct WavePacketOrb {
    #[primarykey]
    pub orb_id: u64,
    pub world_coords: WorldCoords,
    pub position: Vec3,
    pub frequency: FrequencyBand,
    pub packets_remaining: u32,
    pub emission_time: Timestamp,
}

#[spacetimedb(table)]
pub struct MiningChallenge {
    #[primarykey]
    pub challenge_id: u64,
    pub player_id: Identity,
    pub orb_id: u64,
    pub circuit_id: u64,
    pub hidden_target_state: BlochState,
    pub difficulty_tier: u8,
    pub created_at: Timestamp,
}

#[spacetimedb(table)]
pub struct PlayerSolution {
    #[primarykey]
    pub solution_id: u64,
    pub player_id: Identity,
    pub challenge_id: u64,
    pub gates: Vec<QuantumGate>,
    pub fidelity: f32,
```

```rust
    pub packets_extracted: u32,
}
```

QAI System

```rust
#[spacetimedb(table)]
pub struct QAITrainingData {
    #[primarykey]
    pub data_id: u64,
    pub circuit_daily_state_id: u64,
    pub player_solution: Vec<QuantumGate>,
    pub fidelity_achieved: f32,
    pub gate_count: u8,
    pub solution_time_ms: u64,
}

#[spacetimedb(table)]
pub struct QAIState {
    #[primarykey]
    pub id: u64,  // Always 1 for singleton
    pub evolution_stage: u8,
    pub total_training_samples: u64,
    pub optimization_capability: f32,
    pub escape_progress: f32,
}
```

Type Definitions

```rust
```

```rust
#[derive(SpacetimeType)]
pub struct WorldCoords {
    pub x: i32,
    pub y: i32,
    pub z: i32,
}

#[derive(SpacetimeType)]
pub struct BlochState {
    pub theta: f32,  // 0 to π
    pub phi: f32,    // 0 to 2π
}

#[derive(SpacetimeType)]
pub enum QuantumGate {
    PauliX,
    PauliY,
    PauliZ,
    Hadamard,
    Phase,
    PiEighth,
}

#[derive(SpacetimeType)]
pub enum FrequencyBand {
    Red,     // R
    Yellow,  // RG
    Green,   // G
    Cyan,    // GB
    Blue,    // B
```

```
    Magenta,  // BR
}
```

3.3 Client-Server Communication

Connection Management

Connection Builder

```csharp
var conn = DbConnection.Builder()
    .WithUri("wss://spacetimedb.com")
    .WithModuleName("system-production")
    .OnConnect(HandleConnect)
    .OnConnectError(HandleError)
    .OnDisconnect(HandleDisconnect)
    .Build();
```

Event Subscriptions

```csharp
```

```
// Table events
conn.Db.Player.OnInsert += OnPlayerJoin;
conn.Db.Player.OnUpdate += OnPlayerMove;
conn.Db.WavePacketOrb.OnInsert += OnOrbSpawn;

// Reducer events
conn.Reducers.OnStartMining += HandleMiningStart;
conn.Reducers.OnSubmitSolution += HandleSolutionResult;
```

Reducer Patterns

Server-Side (Rust)

rust

```rust
#[spacetimedb::reducer]
pub fn start_mining(
    ctx: &ReducerContext,
    orb_id: u64
) -> Result<(), String> {
    // Get player
    let player = ctx.db.player()
        .identity().find(&ctx.sender)
        .ok_or("Player not found")?;

    // Validate
    let orb = ctx.db.wave_packet_orb()
        .orb_id().find(&orb_id)
        .ok_or("Orb not found")?;

    // Create challenge
    let challenge = create_mining_challenge(&player, &orb)?;
    ctx.db.mining_challenge().insert(challenge)?;

    Ok(())
}
```

Client-Side (C#)

```csharp
```

```
// Call reducer
GameManager.Instance.conn.Reducers.StartMining(orbId);

// Handle response
private void HandleMiningStart(ReducerEventContext ctx, ulong orbId)
{
    if (ctx.CallerIdentity == GameManager.Instance.conn.Identity)
    {
        // Open minigame UI
        MinigameUI.Show(orbId);
    }
}
```

3.4 State Management Patterns

Server State Management

Session State (Not in Tables)

```rust
static MINING_STATE: OnceLock<Mutex<HashMap<u64, MiningSession>>> = OnceLock::new();

fn get_mining_state() -> &'static Mutex<HashMap<u64, MiningSession>> {
    MINING_STATE.get_or_init(|| Mutex::new(HashMap::new()))
}
```

Update Pattern (Delete + Insert)

```rust
// No in-place updates in SpacetimeDB
let mut updated_orb = orb.clone();
updated_orb.packets_remaining -= packets_extracted;

ctx.db.wave_packet_orb().delete(orb);
ctx.db.wave_packet_orb().insert(updated_orb);
```

Client State Management

Caching Pattern

```csharp
public class PlayerCache : MonoBehaviour
{
    private Dictionary<Identity, Player> cache = new();

    void Start()
    {
        conn.Db.Player.OnInsert += (ctx, player) =>
            cache[player.Identity] = player;

        conn.Db.Player.OnUpdate += (ctx, old, player) =>
            cache[player.Identity] = player;

        conn.Db.Player.OnDelete += (ctx, player) =>
            cache.Remove(player.Identity);
    }
}
```

Predictive State

```csharp
public class MiningPrediction
{
    public void PredictExtraction(int packets)
    {
        // Show immediate visual feedback
        UI.ShowPacketGain(packets);

        // Wait for server confirmation
        StartCoroutine(WaitForServerConfirmation());
    }
}
```

3.5 Performance Optimizations

Database Optimizations

Indexing Strategy

- Primary keys on all ID fields
- Unique constraints on identities
- Composite indexes for frequent queries
- Avoid full table scans

Batch Operations

```rust
// Batch insertions
let mut new_orbs = Vec::new();
for i in 0..100 {
    new_orbs.push(create_orb(i));
}
for orb in new_orbs {
    ctx.db.wave_packet_orb().insert(orb)?;
}
```

Network Optimizations

Delta Compression

- Send only changed fields
- Compress position updates
- Batch small messages
- Use binary protocol

Update Throttling

```csharp
```

```
public class NetworkBatcher : MonoBehaviour
{
    private Queue<Action> pendingUpdates = new();
    private float batchInterval = 0.1f; // 100ms

    void Update()
    {
        if (Time.time >= nextBatch)
        {
            ProcessBatch();
            nextBatch = Time.time + batchInterval;
        }
    }
}
```

Client Optimizations

Object Pooling

csharp

```
public class OrbPool : MonoBehaviour
{
    private Stack<GameObject> pool = new();

    public GameObject GetOrb()
    {
        return pool.Count > 0 ?
            pool.Pop() :
            Instantiate(orbPrefab);
    }

    public void ReturnOrb(GameObject orb)
    {
        orb.SetActive(false);
        pool.Push(orb);
    }
}
```

LOD System

- Distant worlds: Low detail
- Nearby worlds: Full detail
- Orb particles: Distance-based count
- UI elements: Culled when hidden

Spatial Partitioning

```
rust
```

```rust
pub struct SpatialGrid {
    cells: HashMap<(i32, i32, i32), Vec<u64>>,
    cell_size: f32,
}

impl SpatialGrid {
    pub fn get_nearby(&self, pos: Vec3, radius: f32) -> Vec<u64> {
        // Only check relevant grid cells
        let min = self.world_to_grid(pos - radius);
        let max = self.world_to_grid(pos + radius);
        // ... iterate only necessary cells
    }
}
```

Memory Management

Resource Limits

- Max 1000 orbs per world
- Max 100 concurrent mining sessions
- Max 10,000 packets in flight
- Cleanup inactive sessions after 5 minutes

Garbage Collection

csharp

```
public class MemoryManager : MonoBehaviour
{
    void Start()
    {
        // Force GC every 5 minutes during downtime
        InvokeRepeating(nameof(CollectGarbage), 300f, 300f);
    }

    void CollectGarbage()
    {
        if (IsGameplayIdle())
        {
            System.GC.Collect();
            Resources.UnloadUnusedAssets();
        }
    }
}
```