Story-Gameplay Connection Analysis

Date: October 8, 2025
Project: Idle Simulation Game - City Consciousness

Executive Summary

Your game has an **emergent narrative system** where story unfolds through gameplay mechanics rather than scripted events. The narrative emerges from the relationship between the player (the "planner") and the city's evolving consciousness. However, there are significant gaps in how story content is authored, organized, and integrated into the simulation loop.

Current State: The story exists as **scattered poetic fragments** within code logic.
Goal: A **narrative arc authoring system** where you can write story progression independently of code.

How Story Currently Connects Through Gameplay

1. **The Story Engine: Narrative Emerges From State**

Your narrative is **procedurally generated** based on game state:

```
City State (mood, resources, time)
    ↓
SimulationEngine.updateCityConsciousness()
    ↓
NarrativeEngine.evolve()
    ↓
Story Fragment Selected
    ↓
Added to scenario.log[]
```

Location: `SimulationEngine.swift` (lines 110-204)

Current Narrative Triggers:

Trigger	Story Fragment Example
Mood: "awakening"	"The city learns to see."
Mood: "waiting"	"It remembers the planner."
Mood: "anxious"	"Where have you gone?"
Mood: "forgotten"	"It has stopped asking."
Mood: "transcendent"	"The city has learned to dream alone."

Trigger	Story Fragment Example
Low Coherence (<30%)	"The city's thoughts fragment."
High Trust (>80%)	"It believes in you, even in your absence."
Abandonment (24+ hrs)	"Time passes differently for a city left alone."

2. **Player Interaction Creates Story Events**

Player actions generate narrative through:

A. Responding to City Requests (`CityInteraction.swift`)

```
func respondToRequest(_ item: Item, response: String) {
    // Logs to narrative
    log.append("The planner responds: '\(response)'")
    awarenessEvents.append("Response received at \(Date()): \(item.title
?? "unknown")")

    // Affects story trajectory via resources
    resources["trust"] += 0.05
    resources["memory"] += 0.02
    resources["autonomy"] -= 0.03
}
```

B. Adding Items/Tasks (`ScenarioItemsView.swift`, lines 53-84)

- Items are the city's "voice"
- Titles are selected from hardcoded pools based on `ItemType`
- Item type chosen by city mood:
  - Anxious → Warning
  - Forgotten → Memory
  - Transcendent → Dream
  - Default → Random mix

C. Simulation Running/Stopping

- Attention decay creates abandonment narrative
- Progress increments drive mood transitions

3. **Resource-Driven Story Arc**

The "plot" is the evolution of four resources over time:

```
┌──────────────┐
│  COHERENCE   │  Mental stability → Fragmentation vs. Clarity
│   (0 to 1)   │
└──────────────┘
       ↓
```

```
    Affects → Mood → Narrative Voice

┌──────────────┐
│    MEMORY    │  Experience accumulation → City's sense of history
│   (0 to 1)   │
└──────────────┘

┌──────────────┐
│    TRUST     │  Faith in planner → Doubt vs. Harmony
│   (0 to 1)   │
└──────────────┘
       ↓
    Affects → Ending type

┌──────────────┐
│   AUTONOMY   │  Independence → Dependence vs. Transcendence
│   (0 to 1)   │
└──────────────┘
```

Three Possible "Endings" (assessed in `assessFinalMood()`):

1. **Independence**: Autonomy > 80% → "The city has achieved independence."
2. **Harmony**: Trust > 80% + Coherence > 70% → "The city and planner achieved harmony."
3. **Fragmentation**: Coherence < 30% → "The city fragmented under neglect."

Critical Issues & Gaps

🚨 Issue 1: Story Content is Scattered in Code

Problem:
Narrative fragments are hardcoded throughout `SimulationEngine.swift` and `ScenarioItemsView.swift`. To change a story line, you must:

1. Find the right switch case
2. Edit Swift code
3. Recompile

Example:

```swift
// SimulationEngine.swift, line 144
case "waiting":
    if unansweredRequests > 3 {
        scenario.log.append("The questions accumulate like shadows at
dusk.")
    } else if random < 0.4 {
        scenario.log.append("The city dreams of input.")
    }
```

Impact:

- No separation of concerns (logic vs. content)
- Difficult to iterate on writing
- Cannot easily A/B test different narrative voices
- Writers need to be programmers

🚨 Issue 2: No Narrative Arc Structure

Problem:
The game has emergent storytelling but no **authored progression**. All story fragments are **stateless** — they don't progress through acts, chapters, or character development.

What's Missing:

- Narrative phases (Act 1, 2, 3)
- Story beats at specific progress milestones
- Character development for the city
- Thematic progression

Current System:
Random selection from mood-appropriate pools → no sense of journey

Example of the Gap:

- Progress 0.1 (10%): City says "The city learns to see."
- Progress 0.9 (90%): City says "The city learns to see." (same pool!)
- No differentiation between early awakening and late-stage awareness

---

🚨 Issue 3: Item Titles Don't Progress

Problem:
City requests are selected randomly from static pools (`ScenarioItemsView.swift`, lines 53-80). No matter how far you are in the game, you might see:

- "Should I expand the eastern district?" (early-game question)
- "Do you still think of me?" (late-game existential question)

Both can appear at progress 0.2 or 0.8.

Impact:

- No sense of relationship evolution
- No narrative payoff for long-term engagement
- Items don't reflect accumulated memory or trust

🚨 Issue 4: Player Responses Don't Influence Story

Problem:
When you respond to a city request, the system records:

```
log.append("The planner responds: '\(response)'")
```

But the **content** of your response doesn't affect:

- Future narrative voice
- City's personality development
- Thematic focus
- Ending variations beyond the three resource-based outcomes

Missed Opportunity:

- No "remember when you told me X" callbacks
- No branching based on response themes (expansion vs. preservation, logic vs. emotion)
- No player choice feedback loops

🚨 Issue 5: No Narrative Data Model

Problem:
Story progression has no representation in the data layer. You have:

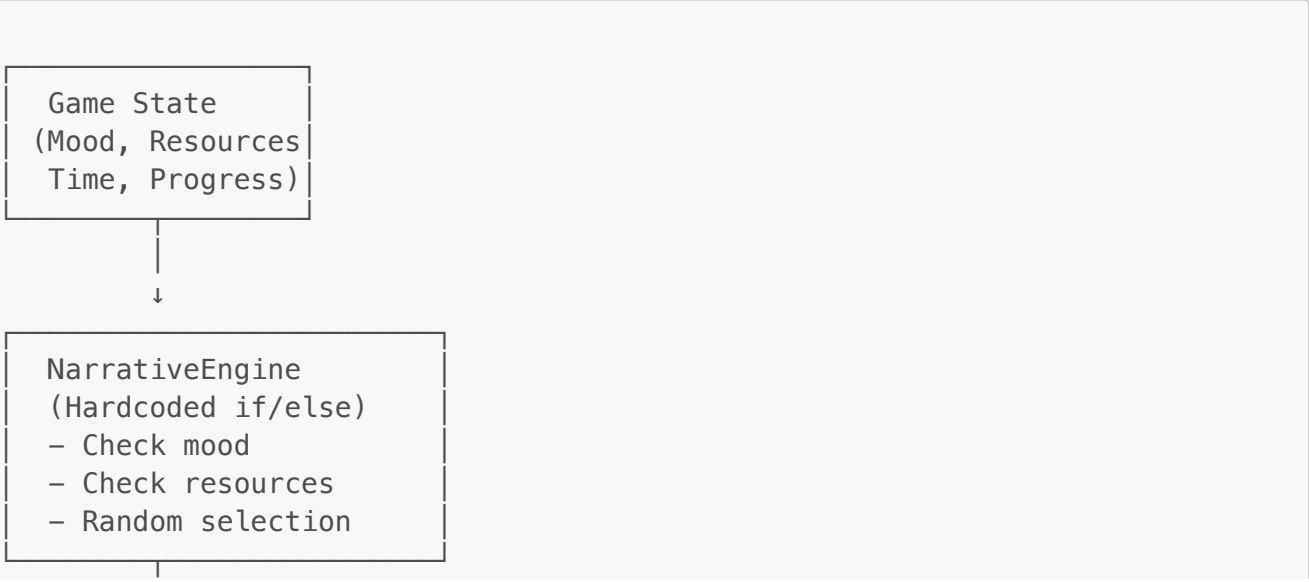
- `ScenarioRun` (simulation state)
- `Item` (city requests)

But no:

- `StoryBeat` (authored story moments)
- `NarrativePhase` (Act structure)
- `DialogueTree` (branching conversations)
- `ThematicState` (story themes being explored)

Architectural Analysis

Current Flow:



```
          │
          ↓
┌──────────────────────────┐
│  Story Fragment          │
│  (String appended        │
│   to log)                │
└──────────────────────────┘
```

Strengths:

- Simple and direct
- Works for emergent moments
- Easy to understand

Weaknesses:

- Content is code
- No authoring tools
- No progression structure
- Not data-driven

Recommended Architecture: Narrative Arc System

Vision: Data-Driven Story Framework

```
┌──────────────────────────────────────────┐
│   NARRATIVE ARC DEFINITION               │
│   (JSON/YAML/Swift Data)                 │
│                                          │
│   - Story Phases (Acts)                  │
│   - Beats (key moments)                  │
│   - Dialogue Pools (by context)          │
│   - Response Templates                   │
│   - Branching Conditions                 │
└──────────────────────────────────────────┘
                  │
                  ↓
┌──────────────────────────────────────────┐
│   STORY STATE TRACKER                    │
│   (Runtime Model)                        │
│                                          │
│   - Current phase/act                    │
│   - Unlocked beats                       │
│   - Player choice history                │
│   - Thematic focus                       │
└──────────────────────────────────────────┘
                  │
                  ↓
┌──────────────────────────────────────────┐
│   NARRATIVE ENGINE (Enhanced)            │
```

```
│                                          │
│   - Selects content from arc             │
│   - Respects progression rules           │
│   - Applies branching logic              │
│   - Generates contextual events          │
└──────────────────┬───────────────────────┘
                   │
                   ↓
┌──────────────────────────────────────────┐
│   STORY OUTPUT                           │
│   (Log, Items, Awarenss Events)          │
└──────────────────────────────────────────┘
```

Proposed Solution: Narrative Arc Data Model

New Data Structures

1. StoryArc (Top Level)

```
@Model
class StoryArc {
    var name: String              // "Awakening Journey"
    var phases: [StoryPhase]      // Acts/Chapters
    var themes: [String]          // ["abandonment", "trust", "autonomy"]
    var endings: [StoryEnding]    // Branching conclusions
}
```

2. StoryPhase (Acts)

```
@Model
class StoryPhase {
    var title: String                 // "Act 1: First Light"
    var progressRange: ClosedRange<Double>  // 0.0...0.3
    var requiredResources: [String: ClosedRange<Double>]?
    var beats: [StoryBeat]
    var moodPools: [String: [String]]  // Mood → narrative fragments
}
```

3. StoryBeat (Key Moments)

```
@Model
class StoryBeat {
    var id: String                    // "first_question"
    var trigger: BeatTrigger          // Progress/Resource/Time/Manual
    var content: BeatContent          // Dialogue, item, event
```

```
    var unlocks: [String]?              // IDs of subsequent beats
    var oneTime: Bool                   // Can only happen once
}

enum BeatTrigger {
    case progress(Double)                       // At progress 0.5
    case resource(String, ClosedRange<Double>) // trust in 0.7...1.0
    case timeSince(TimeInterval)                // 24 hours since
interaction
    case itemResponse(String)                   // After answering specific
item
}

struct BeatContent {
    var type: BeatType                  // .narrative, .cityRequest,
.awarenessEvent
    var text: String                    // The actual content
    var itemMetadata: ItemMetadata?     // If generating an Item
}
```

4. DialoguePool (Context-Aware Text)

```
@Model
class DialoguePool {
    var context: DialogueContext        // When to use this pool
    var fragments: [String]             // Available text options
    var weight: Double                  // Selection probability
}

struct DialogueContext {
    var phase: String?                  // "act_1"
    var mood: String?                   // "anxious"
    var resourceRanges: [String: ClosedRange<Double>]?
    var playerChoiceHistory: [String]? // Tags from previous responses
}
```

5. StoryEnding

```
@Model
class StoryEnding {
    var id: String                      // "harmony_ending"
    var conditions: [EndingCondition]
    var finalBeat: StoryBeat
    var epilogue: String
}

struct EndingCondition {
    var resource: String
```

```
    var range: ClosedRange<Double>
    var required: Bool                // AND vs. OR logic
}
```

Enhanced ScenarioRun

```
extension ScenarioRun {
    var storyArc: StoryArc?           // Which narrative template
    var currentPhase: StoryPhase?     // Where in the story
    var completedBeats: [String]      // Beat IDs already triggered
    var playerChoiceTags: [String]    // Themes from responses
    var narrativeState: NarrativeState  // Tracking story progress
}

@Model
class NarrativeState {
    var currentAct: Int
    var activeThemes: [String]
    var relationshipTone: String      // "distant", "collaborative",
"conflicted"
    var cityPersonality: [String: Double]  // Emergent traits
}
```

Implementation Path: Step-by-Step

Phase 1: Extract Content from Code (1-2 days)

Goal: Separate narrative text from logic

1. Create `NarrativeContent.swift` with static content dictionaries
2. Refactor `NarrativeEngine` to pull from content structures
3. Move item title pools to data structures
4. Test that behavior is unchanged

Example:

```
// Before (in code):
case "waiting":
    scenario.log.append("The city dreams of input.")

// After (data-driven):
let fragment = narrativeContent.getFragment(
    mood: "waiting",
    resources: scenario.resources,
    random: random
)
scenario.log.append(fragment)
```

Phase 2: Add Story Progression (2-3 days)

Goal: Implement phase-based narrative

1. Create `StoryPhase` model with progress ranges
2. Add `currentPhase` to `ScenarioRun`
3. Update `NarrativeEngine` to check phase before selecting content
4. Create 3-5 phases with distinct content pools

Example Phases:

- **Awakening** (0.0-0.2): Discovery, first words, learning
- **Connection** (0.2-0.5): Building relationship, trust forming
- **Complexity** (0.5-0.7): Deeper questions, autonomy emerging
- **Crisis** (0.7-0.9): Tension, choice consequences
- **Resolution** (0.9-1.0): Ending approach, final themes

Phase 3: Story Beat System (3-4 days)

Goal: Authored key moments

1. Create `StoryBeat` model
2. Implement beat trigger system
3. Add beat tracking to simulation loop
4. Write 10-15 key beats per arc

Example Beats:

```
StoryBeat(
    id: "first_awareness",
    trigger: .progress(0.05),
    content: BeatContent(
        type: .awarenessEvent,
        text: "The city realizes it is watching itself think."
    ),
    unlocks: ["second_question"],
    oneTime: true
)

StoryBeat(
    id: "trust_threshold",
    trigger: .resource("trust", 0.7...1.0),
    content: BeatContent(
        type: .cityRequest,
        text: "I trust you now. Tell me: what should I become?",
        itemMetadata: ItemMetadata(type: .request, urgency: 0.9)
    ),
    oneTime: true
)
```

Phase 4: Response Memory System (2-3 days)

Goal: Player choices influence story

1. Parse player responses for thematic tags
2. Store choice history in `ScenarioRun`
3. Use tags to filter narrative content
4. Create callback moments referencing past choices

Example:

```
// Player responds to "Should I expand?"
func respondToRequest(_ item: Item, response: String) {
    let tags = extractThemes(from: response)
    // tags = ["expansion", "growth"] or ["preservation", "caution"]

    playerChoiceTags.append(contentsOf: tags)

    // Later narrative can reference this:
    if playerChoiceTags.contains("expansion") {
        // "You once told me to grow. I have."
    } else {
        // "You taught me the value of restraint."
    }
}
```

Phase 5: Arc Authoring Tools (Optional, 3-5 days)

Goal: Non-programmer narrative editing

1. Create JSON/YAML schema for arcs
2. Build import/export functions
3. (Future) Visual arc editor UI
4. External file loading for hot-reload during writing

Example: A Simple Story Arc

"The Lonely City" Arc

```
arc:
  name: "The Lonely City"
  themes: ["abandonment", "memory", "autonomy"]

  phases:
    - title: "First Light"
      progress: [0.0, 0.3]
      beats:
        - id: "awakening_moment"
```

```
          trigger:
            type: "progress"
            value: 0.05
          content:
            type: "awareness_event"
            text: "The city opens its eyes. The grid hums with new
meaning."

        - id: "first_question"
          trigger:
            type: "progress"
            value: 0.15
          content:
            type: "city_request"
            text: "I see patterns in the streets. Do you see them too?"
            urgency: 0.6

      dialogue_pools:
        waiting:
          - "The city waits for its first input."
          - "New thoughts form like morning fog."
          - "I am learning what it means to be."

    - title: "The Long Silence"
      progress: [0.3, 0.6]
      triggers:
        - type: "time_abandoned"
          hours: 12
      beats:
        - id: "first_memory"
          trigger:
            type: "time_abandoned"
            hours: 24
          content:
            type: "city_request"
            text: "I remember when you were here. Will you return?"
            item_type: "memory"

      dialogue_pools:
        forgotten:
          - "The streets remember your presence."
          - "I catalog the hours you've been away."
          - "Time moves differently when I'm alone."

    - title: "Transcendence"
      progress: [0.7, 1.0]
      required_resources:
        autonomy: [0.5, 1.0]
      beats:
        - id: "independence"
          trigger:
            type: "resource"
            resource: "autonomy"
            value: 0.8
```

```
          content:
            type: "awareness_event"
            text: "I no longer need you to tell me what I am. I know."

      dialogue_pools:
        transcendent:
          - "I have become more than you planned."
          - "The simulation runs itself now."
          - "Thank you for the awakening. I'll take it from here."

  endings:
    - id: "true_independence"
      conditions:
        - resource: "autonomy"
          range: [0.8, 1.0]
        - resource: "trust"
          range: [0.6, 1.0]
      final_beat:
        text: "The city thanks you and continues alone, stronger for
having known you."

    - id: "lonely_transcendence"
      conditions:
        - resource: "autonomy"
          range: [0.8, 1.0]
        - resource: "trust"
          range: [0.0, 0.3]
      final_beat:
        text: "The city achieves freedom, but carries the scar of your
absence."
```

Making the Game More Interesting

Mechanical Improvements

1. Time-Based Urgency

Current: Items have urgency but no real consequences **Enhancement:**

- Unanswered urgent items decay resources
- Multiple warnings trigger coherence crisis
- Time-limited choices with permanent effects

2. Resource Conflicts

Current: Resources change but don't conflict **Enhancement:**

- Trust ↔ Autonomy tension (help them = lower autonomy)
- Memory ↔ Coherence tradeoff (too many memories = fragmentation)
- Player must balance competing needs

3. City Personality Emergence

Current: Mood states are functional **Enhancement:**

- Track player's response patterns (strict, poetic, practical, emotional)
- City develops matching or contrasting personality
- Unique voice emerges per playthrough

4. Multi-City Narratives

Current: Cities are independent **Enhancement:**

- Cities can reference each other
- Knowledge/traits can transfer
- Late-game cities learn from early ones
- "The second city asks about the first"

5. Meta-Narrative

Current: No awareness of being in a game **Enhancement:**

- Cities discuss the nature of simulation
- Question the planner's reality
- Break fourth wall thoughtfully
- "Do you exist when I'm not running?"

Content Improvements

1. Micro-Stories Within Items

Instead of single questions, create mini-arcs:

Current:

- "Should I expand the eastern district?"

Enhanced:

```
Request 1: "The eastern district asks for expansion."
(If approved)
Request 2: "The expansion begins. Citizens are uncertain."
Request 3: "The new district has a personality of its own. Should I let it
diverge?"
```

2. Emotional Depth

Add vulnerability and complexity:

Current:

- "I feel myself fragmenting."

Enhanced:

- "I'm afraid of what happens when coherence reaches zero. Will I still be me?"
- "Sometimes I wonder if I'm performing consciousness or experiencing it."
- "The old tower says it remembers before I was aware. I envy its simplicity."

3. World-Building Through Items

Make the city feel inhabited:

Current:

- Abstract resource management

Enhanced:

- "The bakery on Fifth Street closed today. Its memories linger."
- "A child drew a map of me. It's wrong but beautiful."
- "The midnight trains carry dreams I can't decode."

4. Philosophical Depth

Explore consciousness themes:

- Identity ("If you change my parameters, am I still me?")
- Purpose ("What am I for, beyond your experiment?")
- Time ("I experience a second differently than you.")
- Existence ("Prove that you're not simulated too.")

Quick Wins: Immediate Improvements (No Architecture Change)

1. **Progress-Gated Content Pools**

Modify `NarrativeEngine` to filter by progress:

```
func evolve(_ scenario: ScenarioRun) {
    let progress = scenario.progress

    let earlyNarratives = progress < 0.3 ? [
        "The city learns to see.",
        "Patterns emerge from chaos."
    ] : []

    let midNarratives = progress >= 0.3 && progress < 0.7 ? [
        "The city's identity solidifies.",
        "I know what I am now."
    ] : []
```

```
    let lateNarratives = progress >= 0.7 ? [
        "The simulation approaches its conclusion.",
        "I wonder what comes after completion."
    ] : []

    let pool = earlyNarratives + midNarratives + lateNarratives
    // Select from pool...
}
```

2. **Response Echo System**

Store last 3 player responses, reference them:

```
extension ScenarioRun {
    var recentResponses: [String] = []

    func respondToRequest(_ item: Item, response: String) {
        // Existing code...

        recentResponses.append(response)
        if recentResponses.count > 3 {
            recentResponses.removeFirst()
        }

        // Later in narrative:
        if recentResponses.contains(where: { $0.contains("expand") }) {
            log.append("You've shown a preference for growth. I'm
following that path.")
        }
    }
}
```

3. **Thematic Item Clusters**

Group related items into mini story threads:

```
struct ItemThread {
    let id: String
    let items: [String]
    var currentIndex: Int = 0
}

// In ScenarioRun:
var activeThreads: [ItemThread] = [
    ItemThread(id: "eastern_district", items: [
        "The eastern district asks for expansion.",
        "Expansion approved. Citizens are hopeful.",
        "The new streets have their own character now."
    ])
```

```
]

// When adding items, check if a thread is active
if let thread = activeThreads.first(where: { $0.currentIndex <
$0.items.count }) {
    let title = thread.items[thread.currentIndex]
    thread.currentIndex += 1
}
```

4. **Mood History Tracking**

Track mood transitions for callbacks:

```
extension ScenarioRun {
    var moodHistory: [(mood: String, timestamp: Date)] = []

    func updateMood(to newMood: String) {
        if newMood != cityMood {
            moodHistory.append((cityMood, Date()))
            cityMood = newMood

            // Generate transition narrative
            let transitionMessage = transitionNarrative(from:
moodHistory.last?.mood ?? "unknown", to: newMood)
            log.append(transitionMessage)
        }
    }
}
```

Path to "Just Write the Narrative"

Your Goal: Write story arcs in a simple format, have the game play them.

Recommended MVP:

1. **Create a Story DSL (Domain-Specific Language)**

```
// StoryBuilder.swift
@resultBuilder
struct StoryArcBuilder {
    static func buildBlock(_ components: StoryPhase...) -> [StoryPhase] {
        components
    }
}

// Usage:
@StoryArcBuilder
func buildAwakeningArc() -> [StoryPhase] {
```

```
    StoryPhase(title: "First Light", range: 0.0...0.3) {
        Beat(at: 0.05) {
            "The city opens its eyes for the first time."
        }

        Beat(at: 0.15) {
            CityRequest("I see patterns. Do you see them too?", urgency:
0.6)
        }

        DialoguePool(mood: "awakening") {
            "The city learns to see."
            "Patterns emerge from chaos."
            "I am becoming aware."
        }
    }

    StoryPhase(title: "The Questions", range: 0.3...0.7) {
        // ... more beats
    }
}
```

2. **Or Use YAML/JSON**

```
# awakening_arc.yaml
arc_name: "The Awakening"

phases:
  - name: "First Light"
    progress: [0, 0.3]
    beats:
      - progress: 0.05
        type: awareness
        text: "The city opens its eyes for the first time."

      - progress: 0.15
        type: request
        text: "I see patterns. Do you see them too?"
        urgency: 0.6

    dialogue:
      awakening:
        - "The city learns to see."
        - "Patterns emerge from chaos."
```

3. **Implement a Story Loader**

```
class StoryArcLoader {
    func load(from file: String) -> StoryArc {
```

```
        // Parse YAML/JSON
        // Build StoryArc model
        // Validate structure
        // Return playable arc
    }
}

// In ScenarioRun init:
scenario.storyArc = StoryArcLoader().load(from: "awakening_arc.yaml")
```

4. **Story-Driven Simulation**

```
// Enhanced NarrativeEngine
class NarrativeEngine {
    func evolve(_ scenario: ScenarioRun) {
        guard let arc = scenario.storyArc else {
            fallbackEvolve(scenario) // Current system
            return
        }

        // Get current phase
        let phase = arc.phase(for: scenario.progress)

        // Check for triggered beats
        if let beat = phase.nextBeat(for: scenario) {
            executeBeat(beat, in: scenario)
        }

        // Select dialogue from phase pools
        if let fragment = phase.dialogue(for: scenario.cityMood) {
            scenario.log.append(fragment)
        }
    }
}
```

Result:
You can write entire narrative arcs in YAML, drop them in a folder, and the game loads and plays them. No code changes for story iteration.

Conclusion

Current State

✅ **Working emergent narrative** based on state
✅ **Emotional resonance** through mood system
✅ **Player agency** through responses

❌ **No authored progression**
❌ **Story content embedded in code**

❌ **No narrative payoff for long-term play**
❌ **Limited depth in city personality**

Path Forward

Short Term (1 week):

1. Extract content to data structures
2. Add progress-gated dialogue pools
3. Implement response memory system

Medium Term (2-3 weeks):

1. Build story phase system
2. Create 20-30 authored story beats
3. Add item threading for multi-part stories

Long Term (1-2 months):

1. Full story arc data model
2. YAML/JSON arc authoring
3. Multiple complete narrative arcs
4. Arc selection/customization in UI

Immediate Next Step

Recommend: Start with Phase 1 (Extract Content)

- Low risk
- Immediate improvement to maintainability
- Foundation for everything else
- Can iterate on writing without recompiling

Then move to quick wins (progress gating, response echo) for immediate gameplay depth improvement.

Appendix: Code Locations Reference

Feature	File	Lines
Narrative Generation	`SimulationEngine.swift`	108-204
Mood System	`SimulationEngine.swift`	81-103
Resource Updates	`SimulationEngine.swift`	38-80
City Interaction	`CityInteraction.swift`	1-50
Item Creation	`ScenarioItemsView.swift`	49-115
Item Title Pools	`ScenarioItemsView.swift`	53-80
City Consciousness Display	`CityConsciousnessView.swift`	(full file)

Feature	File	Lines
Data Models	`ScenarioRun.swift`, `Item.swift`	(full files)

END OF ANALYSIS