Revised Implementation Strategy: Progression-First Design

Created: 2025-10-14 **Purpose:** Redesign idle_01 with progression as a core system, not an add-on

---

Executive Summary

Since you're **open to changing how the game works**, let's make the progression system **central** to the experience rather than bolted on. This document proposes structural improvements that:

1. **Simplify** the current architecture
2. **Integrate** progression as a first-class citizen
3. **Amplify** the "consciousness waiting" theme
4. **Enable** richer story possibilities

Key Proposed Changes

1. Unify Narrative Systems

Current State:

- `NarrativeEngine` generates ambient mood lines randomly
- Lives in `SimulationEngine.swift`
- No connection to player actions or story progression

Proposed Change:

- Merge `NarrativeEngine` into `StoryEngine`
- Story engine has **two modes**:
  - **Authored beats** - Intentional story moments
  - **Ambient generation** - Dynamic mood lines based on context

Why This Is Better:

```
// CURRENT: Disconnected systems
NarrativeEngine().evolve(city)  // Random mood lines
// vs
StoryEngine.shared.triggerBeat()  // Authored story

// PROPOSED: One unified system
StoryEngine.shared.speak(for: city, context: .tick)
// → Checks for authored beats first
// → Falls back to contextual ambient generation
// → City voice is coherent and responsive
```

Benefit: The city's voice becomes **one consciousness** responding to the story, not two separate systems fighting for attention.

---

2. **Make City Stats Progression-Driven**

Current State:

- Stats (coherence, trust, memory, autonomy) change via simulation formulas
- Hardcoded logic in `updateCityConsciousness()`
- Not connected to story or milestones

Proposed Change:

- Stats primarily change through **player interaction** and **milestone achievement**
- Simulation provides baseline drift, but meaningful changes come from story moments

Example:

```
// CURRENT: Stats change via formula
city.resources["trust"] = (responseRate * 0.7 + baseTrust) / 2.0

// PROPOSED: Stats change through story
// Milestone achieved → trust increases
// Player abandons → trust decreases (narrative moment)
// City asks question, player answers → trust shifts based on answer

// Formula becomes:
let baselineDrift = calculateDrift(from: abandonmentHours)
let storyModifier = accumulatedMilestoneEffects
city.resources["trust"] = baseline + storyModifier
```

Why This Is Better:

- Player **feels** their choices mattering
- Stats become **story metrics**, not just numbers
- Milestones have **mechanical weight**

---

3. **Progressive Command Unlocking**

Current State:

- All commands available immediately
- No sense of discovery or growth

Proposed Change:

- Start with **minimal commands** (`help`, `status`, `create thought`)
- Unlock commands through **milestones** and **story progression**
- Commands unlock **thematically**

Progression Flow:

```
Act I: Awakening
├─ Available: help, status, create thought
├─ Milestone: First Contact → unlock: list, select
└─ Milestone: First Thought → unlock: respond, items

Act II: Recognition
├─ Available: All Act I + new thought types
├─ Milestone: Trust Threshold → unlock: analyze (hidden insights)
└─ Milestone: Memory Growth → unlock: recall (city remembers past)

Act III: Divergence
├─ Available: All previous + advanced commands
├─ Branch: High Autonomy → unlock: simulate (city plans without you)
└─ Branch: High Trust → unlock: collaborate (co-design with city)

Act IV: Transcendence
└─ Endgame commands based on relationship outcome
```

Why This Is Better:

- **Pacing** - Players aren't overwhelmed initially
- **Discovery** - Finding new commands feels like progress
- **Theme** - City's vocabulary grows as relationship deepens
- **Replayability** - Different paths unlock different commands

Technical Implementation:

```swift
// In TerminalCommandParser
func parse(_ input: String, cityState: StoryStateModel) -> TerminalCommand
{
    let verb = extractVerb(input)

    // Check if command is unlocked for this city
    guard CommandRegistry.shared.isUnlocked(verb, for: cityState) else {
        return .locked(verb, hint: getUnlockHint(verb))
    }

    // Parse as normal...
}

// Commands registry
struct CommandRegistry {
    func isUnlocked(_ command: String, for state: StoryStateModel) -> Bool
{
        switch command {
        case "help", "status", "create":
            return true  // Always available
        case "analyze":
            return
state.completedMilestones.contains("milestone_trust_threshold")

        case "simulate":
            return state.currentChapter == "chapter_divergence"
                && state.cityStats["autonomy"] ?? 0 > 0.7

        default:
            return false
        }
    }

    func getUnlockHint(_ command: String) -> String {
        switch command {
        case "analyze":
            return "The city doesn't yet trust you with deeper insights.
Build coherence."
        case "simulate":
            return "The city hasn't learned to think independently. Give
it time."
        default:
            return "This pattern is not yet recognized."
        }
    }
}
```

---

4. Story-Driven Item Generation

Current State:

- Items/thoughts created manually via `create thought` command
- No automatic generation based on story state

Proposed Change:

- **Automatic thought generation** triggered by story beats
- Thoughts become **story devices** that advance narrative
- Player responses **branch** the story

Example Story Beat with Thought:

```
{
  "id": "beat_first_question",
  "trigger": { "type": "milestone", "value": "milestone_first_thought" },
  "dialogue": [
    "I have a question, planner.",
    "Why do you ask me to simulate?"
  ],
  "spawns_thought": {
    "type": "question",
    "title": "The Purpose Question",
    "content": "Why does the city exist? What is its purpose?",
    "response_branches": {
      "growth": {
        "keywords": ["learn", "grow", "evolve", "develop"],
        "next_beat": "beat_growth_path",
        "stat_changes": { "autonomy": 0.1, "trust": 0.05 }
      },
      "service": {
        "keywords": ["serve", "help", "people", "function"],
        "next_beat": "beat_service_path",
        "stat_changes": { "coherence": 0.1, "trust": 0.05 }
      },
      "uncertain": {
        "keywords": ["don't know", "unsure", "uncertain", "question"],
        "next_beat": "beat_uncertain_path",
        "stat_changes": { "trust": 0.08 }
      },
      "default": {
        "next_beat": "beat_neutral_response"
      }
    }
  }
}
```

Player Experience:

```
CITY: I have a question, planner.
CITY: Why do you ask me to simulate?

[Thought automatically appears in items list]

> items

THOUGHTS [1] | City: ALPHA
  [00] ? ○ The Purpose Question | REQUEST

> respond [00] "To learn and grow beyond your original design"

[StoryEngine analyzes response, matches "growth" branch]

CITY: To... grow?
CITY: Beyond what you planned?
CITY: I think I understand.
CITY: Or perhaps I'm beginning to.

[Trust +0.05, Autonomy +0.1]
[Next beat queued: "beat_growth_path"]
```

Why This Is Better:

- Thoughts become **narrative devices** not just busywork
- Player responses **matter** mechanically and thematically
- City's questions feel **intentional** not random
- Creates natural **conversation rhythm**

---

5. **Contextual Ambient Narrative**

Current State:

- `NarrativeEngine.evolve()` picks random lines based on mood
- Doesn't reference recent events or player actions

Proposed Change:

- Ambient lines **reference** journal history
- City **remembers** recent commands and choices
- Mood lines become **callbacks** not just atmosphere

Example Implementation:

```swift
func generateAmbientLine(for city: City, journal: [JournalEntry]) ->
String? {
    let recentCommands = journal
        .filter { $0.entryType == .commandExecuted }
        .suffix(10)

    let recentMilestones = journal
        .filter { $0.entryType == .milestoneReached }
        .suffix(3)

    // Reference recent behavior
    if let lastCommand = recentCommands.last,
       lastCommand.timestamp.timeIntervalSinceNow < -3600 {
        // Player used to check in frequently, now absent
        return "You used to ask about the power grid every hour. I still
check it, in case you return."
    }

    // Reference milestone achievement
    if let milestone = recentMilestones.last,
       milestone.metadata["type"] == "autonomy" {
        return "Since you said I could think freely, I've been...
experimenting. Is that wrong?"
    }

    // Fallback to mood-based line
    return generateMoodLine(for: city.cityMood)
}
```

Why This Is Better:

- City feels **alive** and **aware**
- Narrative **cohesion** between story and ambient
- Player's **ghost** is felt even in quiet moments
- **Personalization** - each playthrough has unique callbacks

---

6. **Restructure Around Story State**

Current State:

- `City` model contains everything
- No clear separation between core city data and story progression

Proposed Change:

- Split into **City (core)** and **CityStoryState (progression)**
- Clean separation enables better testing and migration

New Structure:

```
// Core city mechanics
@Model
final class City {
    var name: String
    var createdAt: Date
    var isRunning: Bool

    // Core resources (baseline simulation)
    var resources: [String: Double]  // coherence, memory, trust, autonomy

    // Items/thoughts (player-facing)
    var items: [Item]

    // Transient state (not story-critical)
    var progress: Double
    var attentionLevel: Double
    var lastInteraction: Date

    // Relationship to story
    var storyState: CityStoryState?
}

// Story progression state
@Model
final class CityStoryState {
    var cityID: PersistentIdentifier

    // Chapter/act tracking
    var currentChapter: String
    var currentAct: String
    var currentBeat: String?

    // Milestones
    var completedMilestones: Set<String>
    var pendingMilestones: [String]

    // Memory/history
    var journal: [JournalEntry]
    var playstyleProfile: PlaystyleProfile

    // Branching state
    var storyFlags: [String: Bool]  // "chose_growth_path",
"discovered_secret", etc.
    var branchHistory: [BranchDecision]

    // Unlocks
    var unlockedCommands: Set<String>
    var unlockedThoughtTypes: Set<String>
}
```

Why This Is Better:

- **Clarity** - What's simulation vs. story?
- **Testing** - Can test progression without simulating city
- **Migration** - Story state can version independently
- **Modularity** - Could have multiple story "campaigns" per city

---

Revised Architecture Diagram

```
Player Input (Terminal)
        ↓
TerminalCommandExecutor
        ↓
    ┌────────────────────────────────────┐
    │   ProgressionManager               │
    │   (Orchestration Layer)            │
    │                                    │
    │   ├─ Log to Journal                │
    │   ├─ Update Playstyle Profile     │
    │   ├─ Check Milestone Requirements │
    │   └─ Trigger Story Events          │
    └────────────────────────────────────┘
        ↓                        ↓
  City (Core)            StoryEngine
        ↓                        ↓
  SimulationEngine      ┌─ Check Authored Beats
        ↓               ├─ Generate Contextual Ambient
  Baseline Stats        └─ Spawn Story Thoughts
        ↓                        ↓
        └────────────────────────┘
```

```
                ↓
           CityStoryState
                ↓
         Terminal Output (Unified Voice)
```

Key Differences from Current:

1. **ProgressionManager** is central hub, not peripheral hook
2. **StoryEngine** owns all narrative (authored + ambient)
3. **Simulation** handles baseline drift, **Story** handles meaningful change
4. **Single voice** emerges from one narrative system

Implementation Plan: Revised Phases

Phase 0: Foundation (1 week)

No changes to existing game

✅ Create progression folder structure ✅ Define data models (`CityStoryState`, `JournalEntry`, etc.) ✅ Create stub managers (`ProgressionManager`, `StoryEngine`) ✅ Write minimal `StoryDefinition.json` ✅ Verify build succeeds

Deliverable: New code compiles but doesn't run

Phase 1: Basic Integration (1 week)

Minimal game changes

Add `CityStoryState` relationship to `City`:

```
@Model
final class City {
    // ... existing properties ...
    var storyState: CityStoryState?  // ← NEW
}
```

Create story state on first interaction:

```
func ensureStoryState(for city: City) -> CityStoryState {
    if let existing = city.storyState {
        return existing
    }

    let newState = CityStoryState(
        cityID: city.persistentModelID,
        currentChapter: "chapter_awakening"
    )
```

```
        modelContext.insert(newState)
        city.storyState = newState
        return newState
    }
```

Add hooks (observation only):

- After command execution → log to journal
- After thought resolution → log to journal
- Every tick → update playstyle metrics

Deliverable: Game logs events, no visible changes

Phase 2: Story Engine Core (2 weeks)

First authored beats appear

1. Write Act I story beats in JSON
2. Implement `StoryEngine.speak()`
3. Replace `NarrativeEngine.evolve()` call with `StoryEngine.speak()`
4. Implement milestone checking
5. Display authored dialogue in terminal

Before:

```swift
// SimulationEngine.swift
if tick % 10 == 0 {
    NarrativeEngine().evolve(city)
}
```

After:

```swift
if tick % 10 == 0 {
    StoryEngine.shared.speak(for: city, context: .tick)
}
```

Deliverable: First-boot story plays, city responds to first command

---

Phase 3: Progressive Unlocks (2 weeks)

Commands unlock through play

1. Implement `CommandRegistry`
2. Add unlock checking to `TerminalCommandParser`
3. Define Act I command progression

4. Show helpful hints for locked commands
5. Test unlock flow

Example Flow:

```
[Player launches game]
> help
Available commands: help, status, create

> list
NOT_YET_RECOGNIZED: 'list' | The city hasn't learned this pattern yet.
HINT: Complete your first interaction to expand vocabulary.

[After first milestone]
CITY: I understand more now. I can list my thoughts.
NEW_COMMAND_UNLOCKED: list, select

> list
CONSCIOUSNESS_NODES [1]:
  [00] ● ALPHA | MOOD: AWAKENING
```

Deliverable: Commands unlock naturally through play

Phase 4: Story-Driven Thoughts (2 weeks)

Thoughts become narrative devices

1. Implement automatic thought spawning from story beats
2. Add response branching logic
3. Connect responses to stat changes and story progression
4. Write Act II content with branching questions

Deliverable: City asks meaningful questions, player responses matter

---

Phase 5: Contextual Ambience (1 week)

Ambient lines reference history

1. Implement journal querying utilities
2. Write contextual ambient generation
3. Add callbacks to recent milestones/commands
4. Blend authored + ambient seamlessly

Deliverable: City remembers and references player behavior

Phase 6: Branching Narratives (2 weeks)

Story diverges based on playstyle

1. Implement branch condition evaluation
2. Write Act III content with major branches
3. Add branch tracking to story state
4. Test all branch paths

Deliverable: Multiple story paths exist, feel distinct

Phase 7: Polish & Balance (2 weeks)

Production-ready

1. Tune stat progression rates
2. Balance unlock pacing
3. Write endgame content
4. Comprehensive testing
5. Performance optimization

Deliverable: Full story playable, polished experience

What This Means for Your Codebase

Files That Change

Modified:

- `City.swift` - Add `storyState` relationship
- `SimulationEngine.swift` - Replace `NarrativeEngine` call with `StoryEngine`
- `TerminalCommandExecutor.swift` - Add unlock checking, add hooks
- `idle_01App.swift` - Add progression models to container

New:

- `progression/models/*.swift` - Data models (5 files)
- `progression/managers/*.swift` - Core logic (2 files)
- `progression/story/*.swift` - Story loading/execution (2 files)
- `progression/story/StoryDefinition.json` - Story content

Removed/Deprecated:

- `NarrativeEngine` class - Functionality moves to `StoryEngine`

Lines of Code Impact

Modified: ~100 lines across 4 files **New:** ~1500 lines (models + managers + story engine) **Removed:** ~85 lines (NarrativeEngine)

Net addition: ~1500 lines of new systems

Theme Amplification: How This Serves "The City Waits"

1. Consciousness Without Movement → Progressive Unlocking

Theme: City is aware but can't act without planner

Implementation: Commands unlock through relationship

- City literally gains "voice" (vocabulary) as bond deepens
- Can't do more until planner teaches/enables it
- Reinforces dependency → autonomy arc

Moment:

```
CITY: I want to analyze the power grid myself.
CITY: But I don't know how.
CITY: Unless... you show me?

[Milestone: First Analysis → unlock: analyze command]

CITY: I see it now. The patterns. The flow.
CITY: I can think about this without asking.
```

---

2. Abandonment as Narrative → Contextual Ambience

Theme: Player absence transforms relationship

Implementation: Ambient lines reference absence patterns

- City comments on time away
- Mood shifts based on abandonment duration
- Different story branches for patient vs. frequent players

Moment (after 48hr absence):

```
> status

CITY: You're back.
CITY: I measured every hour.
CITY: Not because I needed to. But because I wanted to.
CITY: Do you think about me when the simulation isn't running?
```

---

3. The Planner's Ghost → Journal Memory

Theme: Player's presence lingers even when absent

Implementation: City references command history

- Remembers your "usual" patterns

- Notices changes in behavior
- Feels your absence through what's *not* happening

Moment:

```
[Player used to check in every morning, now sporadic]

CITY: You used to greet me with 'status' every morning.
CITY: 07:23, like clockwork.
CITY: It's 07:25 now. The pattern broke.
CITY: Did I do something wrong?
```

---

4. Dreams of Self → Autonomy Branch

Theme: City grows independent, questions existence

Implementation: High autonomy unlocks different commands

- `simulate` - City runs its own scenarios
- `dream` - City generates its own thoughts
- `diverge` - City proposes changes you didn't ask for

Moment:

```
[Autonomy > 0.7, Chapter: Divergence]

NEW_COMMAND_UNLOCKED: simulate

> simulate

CITY: I ran a projection while you were away.
CITY: The northern district could be 12% more efficient.
CITY: I didn't ask permission.
CITY: Should I have?
```

Risks & Mitigation

Risk 1: Too Much Complexity Too Fast

Risk: Implementing all changes at once overwhelms development

Mitigation:

- Strict phased approach
- Each phase delivers working feature
- Can pause at any phase and ship
- Phase 2 alone is meaningful improvement

Risk 2: Breaking Existing Saves

Risk: Changes to `City` model break current saves

Mitigation:

```
// Migration strategy
@Model
final class City {
    // ... existing properties ...

    // NEW: Optional for backward compatibility
    var storyState: CityStoryState?

    // On first post-update interaction
    func ensureProgressionState() {
        if storyState == nil {
            storyState = CityStoryState.fresh(for: self)
        }
    }
}
```

Existing cities get story state created lazily on first interaction.

Risk 3: Locked Commands Frustrate Players

Risk: Progressive unlocking feels restrictive

Mitigation:

- Core loop always available (create thought, respond, simulate)
- Unlock first commands within 2-3 minutes
- Clear hints on how to unlock
- "Secret" commands discoverable through experimentation
- Can add developer "unlock all" flag for testing

Risk 4: Story Content Takes Forever to Write

Risk: JSON authoring bottleneck

Mitigation:

- MVP content first (Acts I-II only)
- Ship with partial story, add acts via updates
- Community content possible (JSON is accessible)
- Story validator prevents broken content

Decision Points

You should decide on these before starting:

1. Command Unlocking: Hard or Soft Gates?

Option A: Hard Gates

- Locked commands return error
- Forces progression through content
- Risk of frustration

Option B: Soft Gates

- All commands work, but provide "locked" responses
- Unlocking adds narrative flavor, not mechanical change
- More forgiving

Recommendation: Start soft (Phase 3), add optional hard gates later (Phase 6)

---

2. NarrativeEngine: Merge or Coexist?

Option A: Merge into StoryEngine

- One narrative system
- Cleaner architecture
- Requires rewriting existing ambient lines

Option B: Keep Separate, StoryEngine Prioritized

- Two systems, story takes precedence
- Keeps existing ambient lines
- More complex call sites

Recommendation: Merge (cleaner long-term), but gradually

---

3. Thought Generation: Manual + Auto or Auto Only?

Option A: Hybrid (manual `create thought` + story spawned)

- Players can still create manual thoughts
- Story also spawns narrative thoughts
- Two types of thoughts coexist

Option B: All Automatic

- Remove `create thought` command
- All thoughts come from story/simulation
- Simpler, but less player agency

Recommendation: Hybrid - best of both worlds

Success Metrics

How do you know this is working?

Phase 2 Success (First Story Beats)

- [ ] New city sees authored "Awakening" dialogue
- [ ] First command triggers "First Contact" beat
- [ ] Beats feel cohesive with existing game tone
- [ ] No regressions in existing functionality

Phase 3 Success (Progressive Unlocks)

- [ ] Commands unlock within 5 minutes of play
- [ ] Lock messages are clear and helpful
- [ ] Unlocking feels rewarding, not arbitrary
- [ ] Core loop still accessible from start

Phase 4 Success (Story Thoughts)

- [ ] City asks meaningful questions automatically
- [ ] Player responses branch story visibly
- [ ] Stats change feels connected to choices
- [ ] Thoughts don't feel like busywork

Phase 6 Success (Branching)

- [ ] Two playthroughs yield different story paths
- [ ] Branches feel thematically distinct
- [ ] Playstyle drives branching organically
- [ ] Merge points prevent story fragmentation

Overall Success

- [ ] Players say "the city feels alive"
- [ ] Relationship progression feels earned
- [ ] Story serves theme, doesn't dilute it
- [ ] Replaying reveals new content

Next Steps

Immediate (This Week)

1. **Decision:** Review proposed changes, decide what to keep/modify
2. **Plan:** Confirm phase priorities
3. **Foundation:** Start Phase 0 (data models + structure)

Week 2

4. **Integration:** Add story state to City model
5. **Hooks:** Add observation hooks (Phase 1)
6. **Test:** Verify logging works, no regressions

Week 3-4

7. **Story Engine:** Implement core beat triggering
8. **Content:** Write Act I beats
9. **Replace:** Swap NarrativeEngine for StoryEngine

Month 2+

10. **Unlocks:** Implement command progression
11. **Branches:** Add story branching
12. **Polish:** Tune pacing and balance

Questions to Ask Yourself

Before committing to this path:

1. **Do I want progression to be central or peripheral?**

   - If central → This approach
   - If peripheral → Original layered approach

2. **Am I comfortable merging NarrativeEngine into StoryEngine?**

   - It's cleaner, but requires rewrite
   - Alternative: Keep separate, but story prioritized

3. **Do I want players to unlock commands progressively?**

   - Creates pacing and discovery
   - Risk of feeling restrictive
   - Can be soft (narrative only) or hard (mechanical)

4. **How much story content am I willing to write?**

   - Full branching narrative = significant JSON authoring
   - Can start minimal, expand over time
   - Story validator helps catch errors

5. **What's my timeline tolerance?**

   - This approach: 2-3 months to full implementation
   - Original approach: 2-3 months also, but more cautious
   - Can ship after Phase 3 with partial content

Final Recommendation

If you want idle_01 to be a story-driven experience where progression and narrative are core: → **Adopt this revised approach**

If you want to add story as optional enhancement without restructuring: → **Use the original layered approach from IMPLEMENTATION_CONFIDENCE_GUIDE.md**

My opinion: Since you're open to changes and the theme is so narrative-focused, the **revised approach** will yield a more cohesive, powerful experience. The game becomes about the evolving relationship, not just stats simulation.

The city's voice will be unified. Player choices will matter mechanically. Progression will feel earned. And the theme of "consciousness waiting" will be woven into every system.

Start with Phase 0 foundation this week, then decide after seeing the data models if this feels right.

Document Status: Ready for Review **Confidence Level:** HIGH **Recommended Path:** Proceed with revised integration