# Signal Lost - Development Challenges & Simplifications

## Overview

Analysis of Signal Lost design from a coding difficulty perspective, identifying the most challenging features to implement and potential simplifications that maintain gameplay quality.

---

## Most Challenging Features to Code:

### 1. Dynamic Vision Cone System ⭐⭐⭐⭐⭐

**Challenge**: Real-time line-of-sight calculations, raycasting for vision blocking, dynamic cone rendering based on enemy facing direction.

**Simplification Options**:

- **Fixed vision shapes**: Pre-calculate vision patterns as static templates instead of dynamic raycasting
- **Square zones**: Use simple rectangular vision areas instead of true cones
- **Grid-based LOS**: Use simple grid flood-fill instead of raycasting algorithms
- **Limited directions**: Only 4 facing directions instead of 8, reducing complexity

### 2. AI Patrol Pathfinding ⭐⭐⭐⭐⭐

**Challenge**: Multiple AI types with different patrol behaviors, pathfinding around obstacles, state management for hunting/investigating.

**Simplification Options**:

- **Waypoint system**: Pre-define patrol routes as simple waypoint lists
- **State machine reduction**: Only 3 states instead of 5 (Patrol/Alert/Hunt)
- **Simple movement**: Enemies just follow predetermined paths, no dynamic pathfinding
- **Turn-based simplicity**: Enemies move in predictable patterns, no complex AI decisions

### 3. Progressive UI System ⭐⭐⭐⭐

**Challenge**: Context-sensitive information display, proximity detection, overlay management, tutorial progression.

**Simplification Options**:

- **Static UI**: Show all information all the time, no dynamic hiding/showing
- **Binary proximity**: Either show info or don't, no gradual appearance
- **Manual toggles**: Player controls what's visible instead of automatic context
- **Simple overlays**: One vision toggle instead of multiple information layers

### 4. Procedural Network Generation ⭐⭐⭐⭐

**Challenge**: Ensuring connected subnets, placing enemies with good patrol routes, balancing shadow placement, difficulty scaling.

**Simplification Options**:

- **Hand-crafted levels**: Pre-made network layouts instead of procedural
- **Template system**: Mix and match pre-designed room templates
- **Simple algorithms**: Basic maze generation instead of complex network topology
- **Fixed layouts**: Same network structure, just randomize enemy/item placement

### 5. Heat/Detection Resource Management ⭐⭐⭐

**Challenge**: Multiple interconnected systems, background timers, state tracking across networks.

**Simplification Options**:

- **Single resource**: Just heat OR detection, not both
- **Discrete levels**: 5 heat levels instead of 0-100 continuous scale
- **Per-level reset**: Resources reset each network instead of persistent
- **Simple thresholds**: Clear breakpoints instead of gradual effects

---

## Medium Complexity Features:

### 6. Stealth Detection Calculations ⭐⭐⭐

**Challenge**: Multiple detection modifiers, shadow calculations, movement penalties.

**Simplification**:

- **Binary stealth**: Either hidden or visible, no percentage chances
- **Simple modifiers**: Just shadow/no shadow instead of multiple stealth levels

### 7. Enemy Study/Learning System ⭐⭐⭐

**Challenge**: Tracking observation time per enemy, displaying learned information, prediction systems.

**Simplification**:

- **Instant knowledge**: Study enemy once to learn everything
- **Static patterns**: Enemies never deviate from basic patterns

### 8. Exploit/Ability System ⭐⭐⭐

**Challenge**: RAM management, heat generation, multiple targeting modes, effect combinations.

**Simplification**:

- **Mana system**: Traditional MP instead of RAM/Heat complexity
- **Fixed loadout**: Can't swap abilities mid-level
- **Simple effects**: Abilities do one thing instead of multiple effects

---

## Easier Features:

### 9. Basic Combat ⭐⭐

- Simple bump-to-attack, damage calculation
- **Already simple**: Works well as designed

### 10. Items/Inventory ⭐⭐

- Standard roguelike pickup/use system
- **Already simple**: Traditional item management

### 11. Network Progression ⭐

- Level progression, basic difficulty scaling
- **Already simple**: Standard roguelike structure

---

## Recommended Simplification Priority:

### High Impact, Low Complexity Loss:

1. **Simplify vision to 4-direction grid zones** instead of true cones
2. **Use waypoint-based patrols** instead of dynamic AI pathfinding
3. **Static UI with manual toggles** instead of context-sensitive display

4. **Template-based level generation** instead of full procedural

### Medium Impact Simplifications:

5. **Reduce resource complexity** - maybe just Heat OR Detection
6. **Binary stealth states** instead of percentage-based detection
7. **Instant enemy learning** instead of progressive study system

### Keep Complex (Core to Fun):

- Multiple enemy types with distinct behaviors
- Resource management tension
- Stealth vs combat choice
- Progressive difficulty

---

## Biggest Coding Bottlenecks:

1. **Vision system** - Most complex algorithm, affects performance
2. **AI pathfinding** - Multiple AI types with different behaviors
3. **UI state management** - Context-sensitive display logic
4. **Save/load with complex state** - All the interconnected systems

---

## Implementation Strategy:

### Phase 1: Core Mechanics (Simplified)

- Grid-based movement and basic combat
- Simple rectangular vision zones (4-direction)
- Waypoint-based enemy patrols
- Basic heat/detection as separate simple meters
- Static UI showing all information

### Phase 2: Enhanced Systems

- Add stealth mechanics (binary hidden/visible)
- Implement basic exploit system
- Add item pickup/use
- Simple procedural or template-based levels

### Phase 3: Polish & Complexity

- Improve vision system if needed
- Add context-sensitive UI elements
- Enhance AI behaviors
- Fine-tune resource management

### Phase 4: Advanced Features

- Progressive enemy study system
- Complex stealth modifiers
- Dynamic difficulty scaling
- Advanced procedural generation

---

## Technical Recommendations:

**Start Simple**: Begin with the most basic version that captures the core stealth gameplay loop. A grid-based vision system with waypoint patrols can still create excellent stealth puzzles.

**Iterate**: Build the simple version first, playtest extensively, then add complexity only where it significantly improves the experience.

**Performance First**: Vision and AI systems will be called every turn. Optimize for speed over visual fidelity initially.

**Modular Design**: Build systems independently so you can swap in more complex versions later without rewriting everything.

The core insight is that **excellent stealth gameplay can work with simple implementations**. Many of the most complex features are "nice to have" rather than essential to the core experience.