Logging & Configuration Reference v2.0 - Snake Evolution

Logging Patterns, Configuration Management & Operational Procedures

Redatto da: Senior Software Architect
Data: Novembre 2025
Versione: 2.0 (Adattato da Tic-Tac-Toe)
Status: Production Ready

Executive Summary

Questo documento fornisce **reference completo** per logging patterns, configuration options, monitoring, debugging, e operational procedures di Snake Evolution v2.0. È il manuale operativo per amministratori e SRE teams.

1. Logging Architecture

1.1 Logging Pipeline

```
Application Events
  ↓
JavaScript Logger Module
  ├→ Console Handler (human-readable)
  │   └→ Colorized output (DEBUG-INFO-WARNING-ERROR-CRITICAL)
  │       └→ Browser console + DevTools
  │
  ├→ File Handler (browser localStorage)
  │   ├→ LocalStorage key: "snakeEvolution_logs"
  │   ├→ Rotating: max 100 entries
  │   └→ JSON format
  │
  ├→ Remote Handler (optional, v2)
  │   ├→ Sentry.io integration
  │   ├→ Error tracking
  │   └→ Performance monitoring
  │
  └→ Memory Handler
      └→ In-memory buffer (10,000 entries max)
          └→ Accessible via console for debugging
```

1.2 Logging Levels

Level	Priority	Usage	Example
DEBUG	0	Detailed debugging info	"Player moved to (5,10)", "Board state: ..."
INFO	1	General informational	"Game started", "Evolution triggered", "High score saved"
WARN	2	Warning conditions	"Input rejected (rate limit)", "Storage quota low"
ERROR	3	Error conditions	"Collision detection failed", "Save failed"
CRITICAL	4	Critical failures	"Game engine crashed", "Data corruption detected"

2. Logger Implementation

2.1 Logger Module

File: `src/utils/Logger.js`

```
class Logger {
  static levels = {
    DEBUG: 0,
    INFO: 1,
    WARN: 2,
    ERROR: 3,
    CRITICAL: 4
  };

  static currentLevel = this.levels[process.env.LOG_LEVEL || 'INFO'];
  static logs = [];
  static maxLogs = 10000;

  static log(level, message, data = {}) {
    const logEntry = {
      timestamp: new Date().toISOString(),
      level,
      message,
      data,
      module: new Error().stack.split('\n')[2].trim(),
      sessionId: this.getSessionId()
    };

    // Console output
    this.outputToConsole(logEntry);

    // Store in memory
    if (this.logs.length &gt;= this.maxLogs) {
      this.logs.shift();
    }
    this.logs.push(logEntry);

    // Store in localStorage
    this.storeInLocalStorage(logEntry);
    // Remote logging (optional)
    if (level === 'ERROR' || level === 'CRITICAL') {
      this.reportToRemote(logEntry);
    }
  }

  static debug(message, data) { this.log('DEBUG', message, data); }
  static info(message, data) { this.log('INFO', message, data); }
  static warn(message, data) { this.log('WARN', message, data); }
  static error(message, data) { this.log('ERROR', message, data); }
  static critical(message, data) { this.log('CRITICAL', message, data); }

  static outputToConsole(entry) {
    const colors = {
      DEBUG: 'color: #888',
      INFO: 'color: #00AA00',
      WARN: 'color: #FFAA00',
      ERROR: 'color: #FF0000',
      CRITICAL: 'color: #FF00FF'
    };

    const style = colors[entry.level];
    console.log(
      `%c[${entry.level}] ${entry.message}`,
      style,
      entry.data
    );
  }

  static storeInLocalStorage(entry) {
    try {
      const key = 'snakeEvolution_logs';
      let logs = [];

      const stored = localStorage.getItem(key);
      if (stored) {
        logs = JSON.parse(stored);
      }

      logs.push(entry);
      if (logs.length &gt; 100) {
        logs.shift();
      }

      localStorage.setItem(key, JSON.stringify(logs));
    } catch (e) {
      // Silent fail, don't log logging errors
    }
  }

  static reportToRemote(entry) {
    // Optional: Send to Sentry.io or similar
    // if (window.Sentry) {
    //   Sentry.captureException(entry);
    // }
  }

  static getSessionId() {
    if (!window.__sessionId) {
      window.__sessionId = 'session_' + Date.now() + '_' + Math.random().toString(36).sub
    }
    return window.__sessionId;
  }

  static getLogs() {
    return [...this.logs];
  }

  static exportLogs() {
    return JSON.stringify(this.logs, null, 2);
  }

  static clearLogs() {
    this.logs = [];
    localStorage.removeItem('snakeEvolution_logs');
  }
}

export default Logger;
```

2.2 Usage Examples

```
import Logger from './utils/Logger.js';

// Debug
Logger.debug('Game state updated', { snake: { length: 25 }, score: 250 });

// Info
Logger.info('Game started', { playerName: 'John', difficulty: 'medium' });

// Warning
Logger.warn('Input rejected', { reason: 'rate_limit', timeSinceLastInput: 25 });

// Error
Logger.error('Save failed', { error: 'quota_exceeded', bytesNeeded: 5000 });

// Critical
Logger.critical('Game engine crashed', { error: 'null_pointer', stack: '...' });

// Export logs for debugging
const logs = Logger.exportLogs();
console.log(logs);
```

3. Configuration Management

3.1 Configuration Hierarchy

```
Default Config (in code)
  ↓
Environment Config (based on NODE_ENV)
  ├─ development.yaml
  ├─ test.yaml
  └─ production.yaml
  ↓
Environment Variables (.env files)
  ├─ .env.local
  ├─ .env.production
  └─ Override values
  ↓
Final Config
```

3.2 Configuration Files

File: `config/development.yaml`

```
app:
  name: "Snake Evolution"
  version: "2.0.0"
  environment: "development"

game:
  gridSize: 20
  cellSize: 25
  targetFPS: 60
  moveTickMs: 100

evolution:
  stages: 5
  speedIncrement: 0.05

audio:
  enabled: true
  volume: 0.8
  bgmLoop: true

logging:
  level: DEBUG
  console: true
  storage: true
  maxLogs: 10000

performance:
  enableProfiler: true
  frameTimeThreshold: 16.67

storage:
  enableChecksum: true
  backupEnabled: true
  maxEntries: 10
```

File: `config/production.yaml`

```
app:
  name: "Snake Evolution"
  version: "2.0.0"
  environment: "production"

game:
  gridSize: 20
  cellSize: 25
  targetFPS: 60
  moveTickMs: 100

evolution:
  stages: 5
  speedIncrement: 0.05

audio:
  enabled: true
  volume: 0.8
  bgmLoop: true

logging:
  level: INFO
  console: false
  storage: true
  maxLogs: 5000

performance:
  enableProfiler: false
  frameTimeThreshold: 16.67

storage:
  enableChecksum: true
  backupEnabled: true
  maxEntries: 10
```

File: `.env.local`

```
APP_ENV=development
LOG_LEVEL=DEBUG
AUDIO_VOLUME=0.8
```

File: `.env.production`

```
APP_ENV=production
LOG_LEVEL=INFO
```

```
AUDIO_VOLUME=0.8
SENTRY_DSN=https://examplePublicKey@o0.ingest.sentry.io/0
```

3.3 Configuration Loader

File: `src/config/ConfigManager.js`

```
class ConfigManager {
  static config = {};

  static async load() {
    // Load default config
    const defaults = {
      app: {
        name: 'Snake Evolution',
        version: '2.0.0',
        environment: process.env.APP_ENV || 'development'
      },
      game: {
        gridSize: 20,
        cellSize: 25,
        targetFPS: 60
      },
      logging: {
        level: process.env.LOG_LEVEL || 'INFO'
      }
    };

    // Load environment-specific config
    let envConfig = {};
    try {
      const env = process.env.NODE_ENV || 'development';
      envConfig = await import(`./config/${env}.yaml`);
    } catch (e) {
      console.warn('Environment config not found, using defaults');
    }

    // Merge
    this.config = this.deepMerge(defaults, envConfig);

    return this.config;
  }

  static get(key, defaultValue = null) {
    const keys = key.split('.');
    let value = this.config;

    for (const k of keys) {
      value = value?.[k];
      if (value === undefined) {
        return defaultValue;
      }
    }

    return value;
  }

  static set(key, value) {
    const keys = key.split('.');
    let obj = this.config;

    for (let i = 0; i &lt; keys.length - 1; i++) {
      if (!obj[keys[i]]) {
        obj[keys[i]] = {};
      }
      obj = obj[keys[i]];
    }

    obj[keys[keys.length - 1]] = value;
  }

  static deepMerge(target, source) {
    for (const key in source) {
      if (source[key] instanceof Object &amp;&amp; key in target) {
        target[key] = this.deepMerge(target[key], source[key]);
      } else {
        target[key] = source[key];
      }
    }
    return target;
  }
}

export default ConfigManager;
```

4. Performance Monitoring

4.1 Performance Metrics

```
// src/utils/PerformanceMonitor.js

class PerformanceMonitor {
  static metrics = {
    fps: [],
    frameTime: [],
    inputLatency: [],
    renderTime: []
  };

  static recordFrameTime(duration) {
    this.metrics.frameTime.push(duration);
    const fps = 1000 / duration;
    this.metrics.fps.push(fps);

    if (this.metrics.frameTime.length &gt; 300) {
      this.metrics.frameTime.shift();
      this.metrics.fps.shift();
    }
  }

  static getReport() {
    return {
      averageFPS: this.calculateAverage(this.metrics.fps),
      minFPS: Math.min(...this.metrics.fps),
      maxFPS: Math.max(...this.metrics.fps),
      frameTimeP95: this.calculatePercentile(this.metrics.frameTime, 95),
      frameTimeP99: this.calculatePercentile(this.metrics.frameTime, 99)
    };
  }

  static calculateAverage(arr) {
    return arr.reduce((a, b) =&gt; a + b, 0) / arr.length;
  }

  static calculatePercentile(arr, percentile) {
    const sorted = [...arr].sort((a, b) =&gt; a - b);
    const index = Math.ceil((percentile / 100) * sorted.length) - 1;
    return sorted[index];
  }
}

export default PerformanceMonitor;
```

5. Debugging Procedures

5.1 Enable Debug Mode

```
// In browser console
localStorage.setItem('LOG_LEVEL', 'DEBUG');
location.reload();

// Now all DEBUG logs appear in console
```

5.2 Export Logs for Analysis

```
// In browser console
const logs = window.Logger.getLogs();
const json = JSON.stringify(logs, null, 2);
console.save(json, 'snake-evolution-logs.json');
```

5.3 Check Performance

```
// In browser console
const report = window.performanceMonitor.getReport();
console.table(report);

// Output:
```

```
// averageFPS: 58.5
// minFPS: 30.2
// maxFPS: 60.0
// frameTimeP95: 17.2ms
// frameTimeP99: 18.5ms
```

6. Monitoring Checklist

6.1 Production Monitoring

- [ ] FPS maintained ≥ 55 (95% of time)
- [ ] Input latency < 50ms
- [ ] Crash rate < 0.5%
- [ ] Error logs reviewed daily
- [ ] Performance metrics tracked
- [ ] Storage quota monitored

6.2 Error Alerts

```
if (report.averageFPS &lt; 50) {
  Logger.warn('Performance degradation detected', { fps: report.averageFPS });
}

if (error.type === 'collision_detection_failure') {
  Logger.error('Critical collision detection failed', error);
}
```

7. Operational Procedures

7.1 Daily Checks

```
# Check recent error logs
# In browser console:
window.Logger.getLogs()
  .filter(l =&gt; l.level === 'ERROR')
  .slice(-10)

# Check performance metrics
window.performanceMonitor.getReport()

# Check storage usage
console.log(new Blob([JSON.stringify(localStorage)]).size / 1024 + 'KB')
```

7.2 Incident Response

Incident	Check	Action
High crash rate	Error logs	Review stack traces, rollback if needed
Low FPS	Performance metrics	Profile, identify bottleneck
Storage errors	Log level ERROR	Increase quota or implement cleanup
Input latency	Input handler logs	Check debounce settings

8. Appendices

8.1 Log Levels Reference

```
DEBUG    - Development only, verbose info
INFO     - General flow, important events
WARN     - Unusual conditions, potential issues
ERROR    - Failures requiring attention
CRITICAL - System failures, immediate action needed
```

8.2 Common Log Patterns

```
// Game lifecycle
Logger.info('Game started', { playerName, difficulty });
Logger.info('Game over', { finalScore, stage });

// Gameplay events
Logger.debug('Snake moved', { position, direction });
Logger.info('Food eaten', { score, snakeLength });
Logger.info('Evolution triggered', { stage, speedMult });

// Errors
Logger.error('Collision detection failed', { error });
Logger.error('Save failed', { reason });
Logger.error('Audio context suspended', { recovery: 'pending' });

// Performance
Logger.warn('Frame time exceeded', { frameTime, threshold });
Logger.warn('Storage quota low', { used, available });
```

Logging & Configuration Reference v2.0 - Production Ready
Data: Novembre 2025