Novellium

by James Hill

A comprehensive, browser-based visual novel engine with an integrated game builder. Create, play, and share interactive narrative experiences entirely in your web browser.

Table of Contents

Product Objective

Novellium aims to democratize interactive storytelling by providing:

1. **Accessibility**: Zero-installation browser-based platform requiring no specialized software
2. **Dual Interface**: Professional JSON editing for developers, intuitive GUI forms for creators
3. **Complete Workflow**: Integrated tools for creation, testing, asset management, and deployment
4. **Flexibility**: Support for branching narratives, character dialogue, choice-driven gameplay
5. **Portability**: Games stored as JSON with browser-based save/load functionality

Target Audience:

- Writers creating interactive fiction
- Game developers prototyping narrative systems
- Educators teaching storytelling and game design
- Hobbyists exploring visual novel creation

Technology Stack

Core Technologies

- **Frontend Framework**: Vanilla JavaScript (ES6 Modules)
- **Rendering**: HTML5 Canvas + CSS3 Animations
- **Data Storage**:
  - LocalStorage (save games, settings)
  - IndexedDB (asset management, large files)

- **File Handling**: JSZip (package export/import)
- **Build Tool**: None (pure browser execution)

Development Environment

- **Server**: http-server (Node.js) for development
- **Browser Requirements**: Modern browsers with ES6+ support
- **No Dependencies**: Zero npm packages for runtime

File Formats

- **Configuration**: JSON
- **Assets**:
  - Images: PNG, JPG, WebP
  - Audio: MP3, OGG
- **Exports**: ZIP archives, JSON bundles

Architecture Overview

System Architecture

```
┌──────────────────────────────────────────────────────────────┐
│                          Novellium                           │
├──────────────────────────────────────────────────────────────┤
│                                                              │
│  ┌──────────────┐                ┌──────────────┐           │
│  │   Builder    │◄──────────────►│   Library    │           │
│  │  (Editor)    │    Deploy      │  (Runtime)   │           │
│  └──────────────┘                └──────────────┘           │
│         │                              │                     │
│         │                              │                     │
│    ┌────▼────┐                    ┌────▼────┐               │
│    │  Form   │                    │ Asset   │               │
│    │ Editor  │                    │ Loader  │               │
│    └────────┘                    └────────┘               │
│         │                              │                     │
│    ┌────▼────┐                    ┌────▼────┐               │
│    │  JSON   │                    │ Save    │               │
│    │ Editor  │                    │Manager  │               │
│    └────────┘                    └────────┘               │
│         │                              │                     │
│    ┌────▼──────────────────────────────▼────┐               │
│    │          Storage Layer                 │               │
│    │  ┌──────────────┐   ┌──────────────┐   │               │
│    │  │LocalStorage│   │  IndexedDB   │   │               │
│    │  └──────────────┘   └──────────────┘   │               │
│    └────────────────────────────────────────┘               │
└──────────────────────────────────────────────────────────────┘
```

Component Architecture

```
src/
├── engine.js              [Core Library Controller]
│   └── Orchestrates: Renderer, SaveManager, AssetLoader
│
├── managers/
│   ├── AssetLoader.js     [Resource Management]
│   │   └── Handles: Loading images, audio, JSON files
│   ├── SaveManager.js     [Persistence Layer]
│   │   └── Handles: Save/Load games, Export/Import
│   └── ConditionEvaluator.js [Logic Engine]
│       └── Handles: Variable evaluation, conditionals
│
├── models/
│   ├── Character.js       [Character Data Model]
│   ├── Event.js          [Story Event Model]
│   └── GameState.js      [Runtime State]
│
└── ui/
    └── Renderer.js       [Display Layer]
        └── Handles: Canvas rendering, UI updates
```

Data Flow

```
User Action → Library → State Update → Renderer → Display
     ↓                    ↓
Save Manager      Asset Loader
     ↓                    ↓
LocalStorage         Cache/Fetch
```

Design Patterns

1. **Module Pattern** (ES6 Modules)

```
// Encapsulation of functionality
export class NovelliumEngine {
  // Private state
  #characters = new Map();
  #events = new Map();
}
```

Purpose: Namespace isolation, dependency management

2. **Singleton Pattern**

```
// SaveManager - single instance per application
class SaveManager {
  static instance;
  static getInstance() { ... }
}
```

Purpose: Centralized save state management

3. **Observer Pattern**

```
// Event-driven UI updates
renderer.on('choiceSelected', (choice) => {
  engine.processChoice(choice);
});
```

Purpose: Decoupled communication between components

4. **Factory Pattern**

```
// Event creation from JSON
class Event {
  static fromJSON(data) {
    return new Event(data.id, data);
  }
}
```

Purpose: Object creation abstraction

5. **Strategy Pattern**

```
// Different event types with unified interface
class DialogueEvent { render() {...} }
class ChoiceEvent { render() {...} }
class SceneEvent { render() {...} }
```

Purpose: Polymorphic event handling

6. **Facade Pattern**

```
// AssetLoader simplifies complex loading logic
assetLoader.loadGame(path) {
  // Internally: loadJSON, loadImages, preload
}
```

Purpose: Simplified interface to complex subsystems

7. **State Pattern**

```
// GameState manages current game context
class GameState {
  currentEvent: string;
  variables: Map;
  history: Array;
}
```

Purpose: Encapsulated state transitions

Directory Structure

```
novellium/
│
├── index.html                      # Main library runtime interface
│   ├── Game selection UI
│   ├── Save/Load management
│   ├── Settings panel with CSS customization
│   └── Game rendering container
│
├── build.html                      # Interactive game builder
│   ├── Form-based editor
│   ├── JSON editor mode
│   ├── Asset uploader
│   └── Deployment tools
│
├── styles.css                      # Global styling
│   ├── Game UI components
│   ├── Builder interface
│   ├── Responsive layouts
│   └── Animation definitions
│
├── src/                           # Source code (ES6 modules)
│   │
│   ├── engine.js                  # Core engine controller
│   │   ├── Game initialization
│   │   ├── Event processing loop
│   │   ├── State management
```

```
│   │   └── Component orchestration
│   │
│   ├── managers/                  # Business logic layer
│   │   ├── AssetLoader.js         # Resource loading system
│   │   │   ├── JSON file parsing
│   │   │   ├── Image preloading
│   │   │   ├── Audio management
│   │   │   └── Deployed game support
│   │   │
│   │   ├── SaveManager.js         # Persistence management
│   │   │   ├── LocalStorage CRUD
│   │   │   ├── Export to .vnsave files
│   │   │   ├── Import from backups
│   │   │   ├── Auto-backup system
│   │   │   └── Slot management
│   │   │
│   │   └── ConditionEvaluator.js  # Logic evaluation
│   │       ├── Variable parsing
│   │       ├── Conditional checks
│   │       └── Expression evaluation
│   │
│   ├── models/                    # Data models
│   │   ├── Character.js           # Character definition
│   │   │   ├── Name, color, sprites
│   │   │   └── Dialogue attribution
│   │   │
│   │   ├── Event.js              # Story event model
│   │   │   ├── Event types: dialogue, choice, scene, narration
│   │   │   ├── Background/sprite references
│   │   │   ├── Text content
│   │   │   └── Navigation (next event, options)
│   │   │
│   │   └── GameState.js          # Runtime state container
│   │       ├── Current event tracking
│   │       ├── Variable storage
│   │       ├── History stack
│   │       └── Game metadata
│   │
│   └── ui/                        # Presentation layer
│       └── Renderer.js           # Display management
│           ├── Background rendering
│           ├── Character sprite display
│           ├── Dialogue box rendering
│           ├── Typewriter effect
│           ├── Choice button generation
│           └── Error/feedback display
│
├── gamefolder/                    # Game content directory
│   └── [game-id]/                # Individual game folder
│       ├── config.json           # Game configuration
│       │   ├── title: Game name
│       │   ├── author: Creator name
│       │   ├── characters: Path to characters file
```

```
│       │   ├── story: Path to events file
│       │   └── startEvent: Initial event ID
│       │
│       ├── characters.json       # Character definitions
│       │   └── { "char-id": { name, color, sprites[] } }
│       │
│       ├── events.json          # Story events
│       │   └── { "event-id": { type, text, next, options, ... } }
│       │
│       ├── backgrounds/          # Background images
│       │   └── *.jpg, *.png
│       │
│       ├── characters/           # Character sprites
│       │   └── *.png (transparent)
│       │
│       ├── music/               # Background music
│       │   └── *.mp3, *.ogg
│       │
│       └── sounds/              # Sound effects
│           └── *.mp3, *.ogg
│
├── games-list.json              # Game registry
│   └── { games: [{ id, title, description, thumbnail, folder }] }
│
└── check-game-data.js           # Validation utility (Node.js)
    ├── JSON schema validation
    ├── File existence checks
    └── Event flow verification
```

Key File Descriptions

index.html - Library Runtime

- Entry point for playing visual novels
- Manages game library display (vertical sidebar)
- Integrates save system with export/import
- Dynamic CSS theming system
- Deployed game loader (LocalStorage integration)

build.html - Game Builder

- Dual-mode editor (Form GUI / JSON)
- Project Info: Game metadata
- Characters: Character database with color/sprite management
- Events: Story builder with event type templates
- Assets: Upload system using IndexedDB
- Import & Deploy: ZIP upload with validation

src/engine.js - Library Core

- Loads game from folder or LocalStorage
- Processes events sequentially
- Manages game state and history
- Coordinates all subsystems

src/managers/SaveManager.js

- Prefix-based save storage (`vn_save_*`)
- Auto-backup to downloadable files
- Full backup/restore system
- Slot enumeration and management

src/managers/AssetLoader.js

- Folder-based asset loading
- Deployed game support (base64 assets)
- Image preloading with promises
- Path resolution

src/ui/Renderer.js

- Canvas-based background rendering
- HTML overlay for dialogue/UI
- Typewriter text effect
- Animated transitions

Features

Library Features

Game Playback

- **Event System**
  - Dialogue events with character attribution
  - Choice events with branching paths
  - Scene transitions with background changes
  - Narration events (no character)
  - Automatic progression and manual advancing
- **Visual Presentation**
  - Full-screen background images
  - Character sprites with positioning
  - Dialogue box with typewriter effect

    - Character name/color display
    - Smooth transitions and fades

- **Audio System** (Framework Ready)
    - Background music loop support
    - Sound effect triggers
    - Volume controls

Save System

- **Slot-Based Saves**
    - Multiple save slots per game
    - Timestamp tracking
    - Quick save/load functionality
    - Save preview with metadata
- **Backup System**
    - Export individual saves (`.vnsave` format)
    - Export all saves (`.vnbackup` bundle)
    - Import from backup files
    - Auto-backup toggle (downloads on save)
- **Data Persistence**
    - LocalStorage for save data
    - Survives page refreshes
    - Game-specific save isolation
    - Clear warning about cache clearing

Settings & Customization

- **CSS Theme Editor**
    - Primary color (gold default)
    - Secondary color (red)
    - Background colors
    - Text colors
    - Dialogue box (color + opacity)
    - Button colors and hover states
    - Live preview with `!important` overrides
- **Runtime Controls**
    - Text speed (typewriter)
    - Auto-advance timing
    - Skip read text
    - History review

Game Library

- **Vertical Sidebar Layout**
  - Game card thumbnails
  - Title and description
  - Quick launch
  - Visual selection highlight
- **Deployed Games**
  - LocalStorage-based games
  - Merged with folder-based games
  - Instant loading (no HTTP requests)

🔧 Builder Features

Form Editor Mode

- **Project Info Tab**
  - Game title (auto-generates ID)
  - Author name
  - Description
  - Initial event configuration
- **Characters Tab**
  - Add/Edit/Delete characters
  - Character ID and display name
  - Color picker for dialogue styling
  - Sprite list (comma-separated paths)
  - Visual character list with edit buttons
- **Events Tab**
  - Event type selector: Dialogue, Choice, Scene, Narration
  - Dynamic form fields per type
  - Background image assignment
  - Character/sprite selection (from characters)
  - Next event linking
  - Choice options with branching
  - Visual event list with preview
- **Assets Tab**
  - File upload (multiple files)
  - Asset type categorization (backgrounds, characters, music, sounds)
  - IndexedDB storage (large capacity)
  - Image preview modal

    - Individual download
    - Download all assets
    - File size and upload date display
    - Delete functionality

- **Export/Build Tab**
    - Export `config.json`
    - Export `characters.json`
    - Export `events.json`
    - Export complete ZIP package (with assets)
    - Manual installation instructions

- **Import & Deploy Tab**
    - Upload ZIP package
    - Automatic extraction and parsing
    - Comprehensive validation:
        - Required files check
        - JSON syntax validation
        - Event linkage verification
        - Initial event existence
        - Character references
    - Detailed error reporting
    - Deploy current project (from forms)
    - Instant integration to engine
    - "Go to Engine and Play" button

JSON Editor Mode

- **Toggle Switch**
    - Visible only on: Project Info, Characters, Events
    - Hidden on: Assets, Export, Import
    - Smooth transition animation

- **Section-Synced Editing**
    - Auto-loads current section's JSON
    - Project → config.json structure
    - Characters → characters.json
    - Events → events.json
    - Dynamic title/description updates

- **Editor Tools**
    - Syntax-highlighted textarea
    - Monospace font for readability
    - Format JSON button (auto-indent)

    - Validate JSON button (syntax check)
    - Apply Changes (parse and save)
    - Error feedback with line numbers

- **Workflow Integration**
    - Changes reflect in form mode
    - Form refreshes after apply
    - Preserves project state
    - Auto-saves to LocalStorage

Auto-Save System

- Changes saved to browser storage
- Project restoration on page load
- Field-level auto-save triggers
- Manual save button

Asset Management

- IndexedDB for large files
- Base64 encoding for portability
- Organized by type folders
- Game-specific asset isolation
- Preview before download

Deployment Features

Validation System

- **Pre-Deploy Checks**
    - Game ID present
    - Title non-empty
    - Initial event defined
    - At least one event exists
    - Initial event found in event list
    - All event links valid (no dead references)
    - All choice options link to valid events

- **Error Reporting**
    - Color-coded status messages:
        - 🟠 Orange: Processing/Validating
        - 🔴 Red: Errors with detailed list
        - 🟢 Green: Success with statistics
    - Specific error descriptions
    - Actionable fix suggestions

Deployment Pipeline

1. Validate game data structure
2. Convert to engine-compatible format

- Store in LocalStorage with prefix

4. Store assets as base64 bundle
5. Register in games list
6. Display success with metrics

Library Integration

- Games appear immediately in library
- No server restart needed
- Instant loading (cached data)
- Seamless with folder-based games
- Thumbnail from first background

Getting Started

Prerequisites

- Modern web browser (Chrome, Firefox, Edge, Safari)
- Node.js (for development server only)

Installation

1. **Clone or Download Repository**

```
git clone https://github.com/SpeedyDuck790/Novelluim.git
cd Novelluim/visual-novel-engine
```

2. **Install Development Server**

```
npm install -g http-server
```

3. **Start Server**

```
http-server -p 8081 --cors -c-1
```

   - Port: 8081
   - CORS: Enabled (for asset loading)

- Cache: Disabled (-c-1 for development)

4. **Open in Browser**

```
http://localhost:8081/index.html    # Library
http://localhost:8081/build.html    # Builder
```

Quick Start - Creating Your First Game

1. **Open Builder**: Navigate to `build.html`
2. **Project Info**:
   - Title: "My First Story"
   - Author: Your name
   - Initial Event: "start"
3. **Add Character**:
   - ID: `hero`
   - Name: "Hero"
   - Color: Choose any
4. **Create Start Event**:
   - Type: Dialogue
   - ID: `start`
   - Character: hero
   - Text: "Welcome to my story!"
   - Next: `end`
5. **Create End Event**:
   - Type: Narration
   - ID: `end`
   - Text: "The end."
6. **Deploy**:
   - Go to "Import & Deploy" tab
   - Click "Deploy Current Project"
   - Click "Go to Library and Play"

📖 Usage Guide

For Creators

Using Form Mode

1. Fill out Project Info first (generates game ID)
2. Add all characters before creating events
3. Use Characters dropdown in event forms
4. Link events with "next" field or choice options
5. Save frequently (auto-saves on field change)
6. Upload assets before referencing in events
7. Export complete package for sharing

Using JSON Mode

1. Switch to JSON Editor (toggle in top bar)
2. Edit structure directly
3. Use Format JSON to auto-indent
4. Validate before applying
5. Click Apply Changes to save
6. Switch back to Form to verify

Asset Management

- Recommended sizes:
  - Backgrounds: 1920x1080px (16:9)
  - Character Sprites: 512x1024px (transparent PNG)
  - Thumbnails: 400x225px
- Organize uploads by type
- Preview before finalizing
- Download all for deployment

Testing Your Game

1. Deploy from Builder
2. Switch to Library (index.html)
3. Select your game from sidebar
4. Test all branches and choices
5. Return to Builder for edits
6. Re-deploy and test again

For Players

Playing Games

1. Select game from left sidebar
2. Click to advance dialogue
3. Click choice buttons to branch
4. Use Save button (top-right) anytime
5. Load previous saves from Settings panel
6. Export saves for backup

Customizing Appearance

1. Click Settings (⚙) button
2. Choose colors with pickers
3. Click Apply Colors
4. Changes apply immediately
5. Settings persist across sessions

Contact

- **Repository**: GitHub - Novellium
- **Issues**: Use GitHub Issues for bug reports
- **Discussions**: GitHub Discussions for questions

Version: 1.0.0
Last Updated: October 2025
Status: Active Development

Known Limitations

- Browser storage limits (~10MB LocalStorage, unlimited IndexedDB)
- No server-side processing
- Assets must be base64 for deployed games
- Limited to browser API capabilities
