Novellium

A simple browser-based visual novel engine where you can create and play interactive stories.

What is this?

Novellium lets you:

- **Create** visual novels using a simple web interface
- **Play** visual novels in your browser
- **Share** your stories as files

No downloads, no complex setup - just open it in your browser and start creating.

Quick Start

1. **Start the server**:

   ```
   # Install http-server if you don't have it
   npm install -g http-server

   # Run the server
   http-server -p 8000 -c-1
   ```

2. **Open in browser**:
   - Go to `http://localhost:8000`
   - Click "Builder" to create stories
   - Click "Library" to play stories

How to Create a Story

1. **Open the Builder** (`build.html`)
2. **Add your game info** - title, author, description
3. **Create characters** - give them names and colors
4. **Write events** - dialogue, choices, narration
5. **Deploy** - click "Deploy" to make it playable
6. **Test** - go back to the library and play your story

How Stories Work

Stories are made of **events** that link together:

- **Dialogue**: Characters talking
- **Narration**: Story text without a character
- **Choices**: Let players make decisions

- **Scene**: Change backgrounds or music

Each event can link to the next one, creating your story flow.

File Structure

```
Novellium/
├── index.html                          # Main library page (play games)
├── build.html                          # Game builder interface
├── styles.css                          # Global styles and themes
├── package.json                        # Project configuration
├── README.md                           # This documentation
│
├── src/                                # Core engine code
│   ├── engine.js                       # Main game engine
│   ├── models/                         # Data models
│   │   ├── Character.js                # Character class definition
│   │   ├── Event.js                    # Story event class
│   │   └── GameState.js                # Game state management
│   ├── managers/                       # System managers
│   │   ├── AssetLoader.js              # Load images/audio assets
│   │   ├── ConditionEvaluator.js       # Handle conditional logic
│   │   └── SaveManager.js              # Save/load game progress
│   └── ui/                             # User interface
│       └── Renderer.js                 # Display engine for scenes
│
├── config/                             # Configuration files
│   └── games-list.json                 # Registry of available games
│
├── scripts/                            # Utility scripts
│   ├── check-game-data.js              # Validate game data
│   └── navbar.html                     # Shared navigation component
│
├── gamefolder/                         # Game storage directory
│   ├── adventure-game/                 # Example adventure game
│   │   ├── config.json                 # Game metadata
│   │   ├── characters.json             # Character definitions
│   │   ├── story.json                  # Story events and flow
│   │   ├── backgrounds/                # Background images
│   │   └── sprites/                    # Character sprites
│   └── dating-game/                    # Example dating sim
│       ├── config.json                 # Game metadata
│       ├── characters.json             # Character definitions
│       ├── story.json                  # Story events and flow
│       ├── backgrounds/                # Background images
│       └── sprites/                    # Character sprites
│
├── NovelliumLogo/                      # Brand assets and icons
│   ├── logo.png                        # Main logo
│   ├── favicon.ico                     # Browser icon
│   ├── favicon.svg                     # Vector browser icon
```

```
│   ├── apple-touch-icon.png            # iOS home screen icon
│   ├── favicon-96x96.png               # High-res favicon
│   ├── web-app-manifest-192x192.png    # PWA icon (192x192)
│   ├── web-app-manifest-512x512.png    # PWA icon (512x512)
│   └── site.webmanifest                # PWA manifest
│
├── docs/                               # Documentation
│   └── README.pdf                      # PDF version of docs
│
└── .git/                               # Git repository data
```

Features

For Creators:

- Visual editor with forms
- JSON editor for advanced users
- Asset upload (images, music)
- Export/import game files
- Live preview

For Players:

- Save/load games
- Customizable themes
- Typewriter text effects
- Choice-driven stories

Tech Stuff

- **No dependencies** - pure HTML/CSS/JavaScript
- **Browser storage** - saves in localStorage
- **ES6 modules** - modern JavaScript
- **Canvas rendering** - for backgrounds
- **File exports** - share as ZIP files

Need Help?

- Check the builder's help sections
- Look at example games in `gamefolder/`
- File issues on GitHub if something breaks

License

Created by James Hill. Use it however you want.

Simple. Clean. It just works. 🎮

Solutions for Persistent Game Imports on Vercel

🟢 **Simple Solutions** (Easy Implementation)

1. GitHub Integration

- Use GitHub API to commit imported games directly to repository
- Requires GitHub token and automatic commits
- Games become part of the repo and persist for all users
- ☑ No backend needed, uses GitHub as storage
- ❌ Requires authentication, public commits

2. Vercel KV Storage

- Use Vercel's built-in Redis-like key-value storage
- Simple API calls to store/retrieve game data
- Fast access, built into Vercel platform
- ☑ Easy setup, integrated with Vercel
- ❌ Paid feature, data limits

3. Browser IndexedDB Enhancement

- Upgrade from localStorage to IndexedDB for larger storage
- Add import/export features for sharing
- Better performance for large games
- ☑ Still client-side, no backend needed
- ❌ Still per-user, not globally shared

🟡 **Medium Solutions** (Moderate Setup)

4. Vercel Serverless Functions + Database

- Add API endpoints via Vercel Functions
- Connect to external database (MongoDB, PostgreSQL)
- Full CRUD operations for games
- ☑ Scalable, proper backend architecture
- ❌ Requires database setup and management

5. Firebase Integration

- Use Firebase Firestore for game storage
- Real-time sync across users
- Built-in authentication
- ☑ Google-managed, real-time features
- ❌ Google dependency, learning curve

6. Supabase Backend

- PostgreSQL database with REST API
- Built-in auth and file storage

- Open-source alternative to Firebase
- ☑ Full-featured, good free tier
- ❌ Another service to manage

🟢 **Advanced Solutions** (Complex Implementation)

7. Headless CMS Integration

- Use Strapi, Sanity, or Contentful
- Treat games as content entries
- Admin interface for game management
- ☑ Professional content management
- ❌ Overkill for simple games, costly

8. Blockchain/IPFS Storage

- Store games on decentralized storage
- Immutable, censorship-resistant
- Unique game NFTs or tokens
- ☑ Decentralized, future-proof
- ❌ Complex, slow, expensive

9. Custom Backend Service

- Separate Node.js/Python backend
- Deploy on Railway, Render, or DigitalOcean
- Full control over architecture
- ☑ Complete flexibility
- ❌ Most complex, separate hosting costs

🔧 **Hybrid Solutions** (Best of Both Worlds)

10. Static + Dynamic Hybrid

- Keep static games in repository
- Add optional cloud sync for user imports
- Graceful degradation when offline
- ☑ Works everywhere, enhanced when connected
- ❌ More complex state management

11. Pull Request Automation

- Users submit games via automated PRs
- GitHub Actions validate and merge
- Community moderation workflow
- ☑ Transparent, version controlled
- ❌ Requires approval workflow

12. CDN + Edge Functions

- Store games on CDN (Cloudflare R2, AWS S3)
- Use edge functions for fast access
- Global distribution
- ☑ Fast worldwide, scalable
- ❌ Multiple services to configure

📊 Recommended Implementation Order

Phase 1: Quick Win

```
1. GitHub API integration for direct commits
2. Enhanced export/import with better UX
```

Phase 2: Proper Backend

```
3. Vercel Functions + Vercel KV
4. User authentication (GitHub OAuth)
```

Phase 3: Scale & Polish

```
5. Migration to full database if needed
6. Advanced features (ratings, search, etc.)
```

💡 Code Examples Available

Each solution above can be implemented with specific code examples:

- API endpoint structures
- Database schemas
- Authentication flows
- Import/export mechanisms

Choose based on your priorities: **simplicity**, **cost**, **features**, or **scalability**.